Using Semantic Web Technology to Automate Data Integration in Grid and Web Service Architectures

Martin Szomszor, Terry R. Payne and Luc Moreau
School of Electronics and Computer Science
University of Southampton
Southampton, SO17 1BJ, UK
{mns03r, trp, L.Moreau}@ecs.soton.ac.uk

Abstract

While the Grid and Web Services have helped us support heterogeneous resource access through the use of service oriented architectures, they have not addressed the issue of heterogeneous data representation. Since service providers often describe their service interfaces using different data models than those assumed by the client, it is common for additional processing to be required to compensate for the mismatch in data formats. By utilising technology from the Semantic Web, we are able to augment existing Web Service systems with middleware to automatically perform data harmonisation when a syntactic mismatch occurs. To achieve this, we have developed a mapping language which can be used to annotate XML *data structures with* OWL *concepts and properties, a Mapping Language Engine to implement this language, and a Dynamic Web Service Invocation component to execute Web Services.*

1 Introduction

Web Services are software components designed to support interoperable machine to machine interaction over a network. By defining standard languages to present software interfaces, such as WSDL [6], and protocols that describe interaction mechanisms, it is possible for computers to communicate across organisational boundaries from a range of heterogeneous platforms. This benefit has been noted by both the Grid computing and eBusiness communities who have adopted Web Services as a fundamental building block for the development of large scale service-oriented architectures [8]. In these systems, it is often desirable to integrate disparate resources, for example, through the creation of a Virtual Organisation on a Grid, or through Enterprise Application Integration in eBusiness. During such a collaboration of resources, it is necessary for participants to exchange information in a format that is mutually intelligible. Given the wide range of heterogeneous data models used by service providers and service consumers, it cannot be assumed that data formats are compatible. Therefore, additional processing is required to integrate components using different syntactic structures, a term we refer as *syntactic mediation*. While this process can be specified manually, either through the definition of data transformations or the creation of bespoke mediator components, it is desirable to automate it because it will save users effort and allow them to compose services without concern for data incompatibilities. To achieve this, we propose to utilise Semantic Web technology.

The *Semantic Web* [3] is an extension of the existing Web that aims to support the description of Web resources in formats that are machine understandable. On the Semantic Web, resources are given well defined meaning by annotating them with concepts and terminology that correlate with those used by humans. This can be achieved through the use of ontologies [9], providing a conceptual model that is common to all but independent of concrete representation. Therefore, to provide a framework that supports the automated mediation of syntactic structures, ontologies can be created that describe information models at a conceptual level, and used as a common vocabulary of terms for the exchange of data.

To focus our work, we examine a common service interaction from a bioinformatics Grid application. We identify where syntactic incompatibility occurs and why automated mediation is desirable. We then show the benefits of using ontologies to describe XML data structures and how this link can be specified using a mapping language. There follows a description of our mapping language and examples of how it can be used to annotate XML data structures. We then present our Mapping Language Engine which implements the mapping language and performs translations between XML data and OWL concepts. Finally, we show how our Mapping Language Engine can be incorporated with our Dynamic Web Service Invocation component to create a system that performs syntactic mediation between Web Services using different data representations.

This paper is organised as follows: Section 2 introduces our bioinformatics use case, Section 3 presents the theory behind using semantic annotations, and Section 4 describes our mapping language, the Mapping Language Engine and how it is combined with the Dynamic Web Service Invoker. Section 5 reviews related work before we conclude and show further work in Section 6.

2 Motivation - Bioinformatics Use Case

Bioinformatics is the application of computational techniques to the management and analysis of biological information. With the collection and storage of large quantities of genomic and proteomic data, coupled with advanced computational analysis tools, a bioinformatician is able to perform experiments and test hypothesis without using conventional 'wet bench' equipment - a technique commonly referred to as *in silico* experimentation. To support this kind of science, a large collection of databases and tools has been developed to provide bioinformaticians with access to massive amounts of biological data and powerful computational software.

The MYGRID [1] project provides an open-source Grid middleware that supports *in silico* biology. Using a service-oriented architecture based on Web Service standards such as WSDL and UDDI [1], a complex infrastructure has been created to provide bioinformaticians with a virtual workbench with which they can perform biological experiments. Access to data and computational resources is provided through Web Services which can be composed using the workflow language XSCUFL [2] and executed using the FREEFLUO [3] enactment engine. The biologist is provided with a user interface (Taverna[4]) which presents the services available, enables them to create and view workflows graphically, execute them, and view the results.

For our use case, we examine a common bioinformatics task: retrieve sequence data from a database and pass it to an alignment tool to check for similarities with other known sequences. According to the service-oriented view of resource access adhered to by MYGRID, this interaction can be modelled as a simple workflow with each stage in the task being fulfilled by a Web Service.

Many Web Services are available to retrieve sequence data. For our example, we use one available at XEMBL `http://www.ebi.ac.uk/xembl/` and another at DDBJ-XML `http://xml.ddbj.nig.ac.jp/index.html`. To obtain a record, an accession number is passed as input and an XML document is returned. These documents essentially contain the same data, namely the sequence data as a string (e.g. `atgagtga...`), references to publications, and features of the sequence (such as the protein translation). However, the format returned by each provider is different - XEMBL returns a BSML formatted document[5] and DDBJ returns a document using to their own custom format[6].

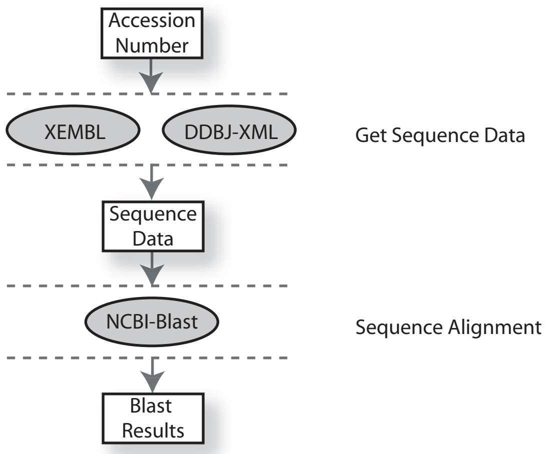

Figure 1. A simple bioinformatics task: get sequence data from a database and perform a sequence alignment on it.

The next stage in the workflow is to pass the sequence data to an alignment service such as the BLAST service at NCBI[7]. This service can take a string of sequence data as input and return the result set in XML. We show this simple workflow in Figure 1.

Intuitively, a bioinformatician will view the two sequence retrieval tasks as the same type of operation, expecting both of them to be compatible with the NCBI Blast service. However, when plugging the two components together, additional information must be provided to specify how data is extracted from one data structure and passed into the next. This could be achieved using a data transformation language such as XSLT [7] or XQUERY [4], but it would require the manual specification of all possible transformations. For n compatible data formats, $(n-1)n$ transformations are required for maximum interoperability. Also, when a new data type is introduced, mappings to and

[1]http://www.mygrid.org.uk
[2]http://taverna.sourceforge.net/docs/xscuflspecification.html
[3]http://sourceforge.net/projects/freefluo/
[4]http://taverna.sourceforge.net/

[5]http://www.ebi.ac.uk/xembl/dtd/BSML2_2.DTD
[6]http://getentry.ddbj.nig.ac.jp/xml/DDBJXML.dtd
[7]http://www.ncbi.nlm.nih.gov/BLAST/

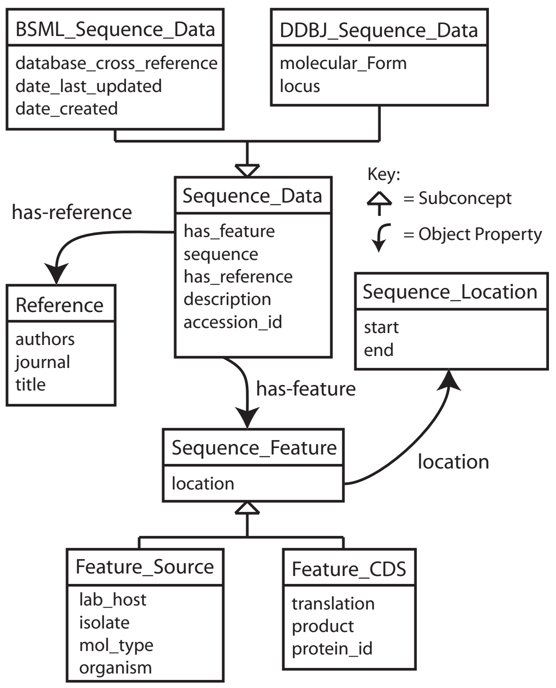

Figure 2. An ontology to describe sequence data. See http://www.ecs.soton.ac.uk/˜mns03r/ont/Sequence for full OWL description.

from all other compatible types would have to be specified. Finally, users are not interested in the details of the service interaction; they prefer them to be hidden so they can focus on the scientific problem.

We propose an architecture in Section 4 that utilises Semantic Web technology to enable the automated mediation of syntactic structure between Web Services. By annotating XML structures with ontology concepts and properties, described in Section 3, we are able to automatically integrate syntactically incompatible services.

3 Semantic Annotations

In this section we show how an ontological definition of a data format can be used to integrate data structures passed between Web Services. We continue using the bioinformatics services presented in Section 2. This example is centred around the concept of some 'sequence data'. We have devised a simple ontology to express this information, which is shown in Figure 2. The main concept, `Sequence_Data` , has the datatype property `sequence` (denoting the string of sequence data), `description` (a text annotation) and `accession_id` (unique id). Each sequence has a number of references which is represented by the `has-reference` object property type and a number of features, represented by the `has-feature` object property. There are a number of sequence features, we show two common ones in this example; `feature_source` (where and how the sequence was gathered), and `feature_CDS` (which shows the protein sequence translation and id). Since BSML format and DDBJ format also contain additional information on the sequence, we introduce subconcepts called `BSML_Sequence_Data` and `DDBJ_Sequence_Data`.

When examining the two services presented by XEMBL and DDBJ, we can consider their input and output to be similar; each take a sequence data accession id as input and both return some sequence data. To be more specific, the XEMBL service returns the concept `BSML_Sequence_Data` and the DDBJ service returns the concept `DDBJ_Sequence_Data` . The next service in the workflow, NCBI Blast, takes some sequence data as input, namely an individual of type `Sequence_Data` with the `sequence` property type specified. Given that the `BSML_Sequence_Data` and `DDBJ_Sequence_Data` concepts are both subsumed by the `Sequence_Data` concept, i.e. the `Sequence_Data` concept is considered more general, we say that the output from both of the sequence data retrieval services is *semantically compatible* with the input to the BLAST service. However, the services are not *syntactically compatible* since the output dataset cannot be passed directly as input to the BLAST service. Therefore, a stage of *syntactic mediation* is required to extract data from one dataset and transform it to create a new dataset.

To automate the process of syntactic mediation, we require mappings from concrete XML structures to conceptual ontology structures. To enable this specification, we have developed a mapping language, presented in Section 4.1, which can be used to specify mappings between XML and OWL [12]. Partial mappings for the two sequence retrieval services is shown in Figure 3. These statements show how the sequence data and accession id can be retrieved from the XML data structure and used to create new OWL concepts. A full mapping for each can be found online[8]. Due to their complexity, they cannot be listed in full within this paper.

When using OWL concepts and properties to annotate an XML data structures, we do not require mappings between all compatible formats. Instead, each data format requires only one mapping to the ontological specification. With this approach, the number of mappings required for each compatible data format has a complexity of $O(n)$ instead of the quadratic complexity discussed in Section 2. It is also more convenient when adding new formats to an existing system since only one mapping is required to achieve maximum interoperability.

[8]http://www.ecs.soton.ac.uk/˜mns03r/mapping/bsml_mapping.mp and http://www.ecs.soton.ac.uk/˜mns03r/mapping/ddbj_mapping.mp

```
{xml}
bsml:Bsml(
 bsml:Definitions(
  bsml:Sequences(
   bsml:Sequence[ic-acckey = $accession](
   bsml:Seq-data($sequence)
))))
<->
{owl}
ont:BSML_Sequence_Data(
 ont:accession_id($accession),
 ont:sequence($sequence),
)
USING
ont for <http://www.ecs.soton.ac.uk/~mns03r/ont/Sequence#>,
bsml for <http://www.ecs.soton.ac.uk/~mns03r/schema/BSML>
```

(a) BSML to Sequence Data mapping

```
{xml}
ddbj:ddbjxml(
 ddbj:accession($accession),
 ddbj:sequence($sequence)
)
<->
{owl}
seq:DDBJ_Sequence_Data(
 seq:accession_id($accession),
 seq:sequence($sequence),
)
USING
seq for <http://www.ecs.soton.ac.uk/~mns03r/ont/Sequence#>,
ddbj for <http://www.ecs.soton.ac.uk/~mns03r/schema/DDBJ#>
```

(b) DDBJ to Sequence Data mapping

Figure 3. Partial mappings from XML to OWL for Sequence Data.

```
⟨mapping⟩  ::=  {⟨type⟩} ⟨exp⟩ ⟨mapsym⟩ {⟨type⟩} ⟨exp⟩ ⟨using⟩ |
                {⟨type⟩} ⟨exp⟩ ⟨mapsym⟩ {⟨type⟩} ⟨exp⟩ |

⟨type⟩ ::= xml | owl                                  ⟨atom⟩ ::= ⟨constant⟩ |
⟨exp⟩ ::= ⟨elem⟩ [ ⟨attr*⟩ ] ( ⟨exp*⟩ ) |                        ⟨variable⟩
          ⟨elem⟩ ( ⟨exp*⟩ ) |                          ⟨atom*⟩ ::= ⟨atom⟩ |
          ⟨elem⟩ [ ⟨attr*⟩ ] ( ⟨exp*⟩ ) ⟨elli⟩ |                  ⟨atom*⟩ , ⟨atom⟩
          ⟨elem⟩ ( ⟨exp*⟩ ) ⟨elli⟩ |                    ⟨elem⟩ ::= ⟨qname⟩
          ⟨concat⟩ ( ⟨atom*⟩ ) |                        ⟨qname⟩ ::= ⟨chars⟩ : ⟨chars⟩ |
          ⟨concat⟩ ( ⟨atom*⟩ ) ⟨elli⟩ |                            ⟨chars⟩
          ⟨split⟩ ( ⟨atom*⟩ ) |                     ⟨constant⟩ ::= "⟨chars⟩"
          ⟨split⟩ ( ⟨atom*⟩ ) ⟨elli⟩ |                   ⟨var⟩ ::= $⟨chars⟩
          ⟨constant⟩ |                                ⟨mapsym⟩ ::= <->
          ⟨var⟩                                         ⟨elli⟩ ::= ...
⟨exp*⟩ ::= ⟨exp⟩ |                                    ⟨concat⟩ ::= concat
          ⟨exp*⟩ , ⟨exp⟩                               ⟨split⟩ ::= split
⟨attr⟩ ::= ⟨qname⟩ = ⟨var⟩ |                           ⟨using⟩ ::= USING ⟨binding*⟩
          ⟨qname⟩ = "⟨constant⟩"                     ⟨binding⟩ ::= ⟨prefix⟩ for < ⟨url⟩ >
⟨attr*⟩ ::= ⟨attr⟩ |                                ⟨binding*⟩ ::= ⟨binding⟩ |
          ⟨attr*⟩ , ⟨attr⟩                                         ⟨binding*⟩ , ⟨binding⟩
                                                      ⟨prefix⟩ ::= ⟨chars⟩
```

Figure 4. The mapping language grammar in BNF notation.

4 Architecture

In this section we present the grammar and semantics of our mapping language before showing the design of our Mapping Language Engine and its integration with out Dynamic Web Service Invocation component.

4.1 Mapping Language

Our mapping language can be used to specify two types of mapping: ontology concept instances to XML and XML to ontology concept instances. The grammar for the language is given in Figure 4 using standard BNF notation. A mapping is composed of a source type ($\{type\}$), source expression, a mapping symbol (`<->`), a destination type, a destination expression and set of using statements that map URLs to prefixes. An expression can be one of five kinds: $\langle elem\rangle$, $\langle constant\rangle$, $\langle var\rangle$, $\langle split\rangle$ or $\langle concat\rangle$. An element expression corresponds to a concept or property type name for an ontology concept instance or the element name within XML document. The contents of an element, contained within parenthesis, is a sequence of further expressions delimited by a comma. These sub-expressions correspond to child nodes of the parent element (for XML) or property types of the parent concept (for OWL). If the sub-expression is a variable, this denotes that the text child of the parent is bound to the the variable (prefixed by a `$` symbol). Hence, the value of a variable within the source expression is mapped to the corresponding variable value in the destination expression. Constants may be specified for elements in the destination expression to define element or concept constructions that are created independently of the inputs. A list of attributes may also be specified for an XML mapping by enclosing them within square brackets after the element name. An attribute expression may either assign a variable to an attribute value or specify the condition that an element must have an attribute with a specific value to be valid. An example of the variable assignment attribute construct can be found in Figure 3(a) where the accession id is extracted from an attribute named `ic-acckey` in the `<bsml:Sequence>` element.

The $\langle split\rangle$ and $\langle concat\rangle$ expressions may be used in source and destination expressions respectively. The $\langle split\rangle$ function takes three arguments; a variable, a constant and another variable. When applied to the text of an XML element or the value of an OWL datatype property, the string is split into two according to the delimiter specified in the second argument and assigned to the two variables specified. If more than one match is found, the string is broken at the first instance of the delimiter. The $\langle concat\rangle$ expression can be used in destination expressions to indicate the concatenation of constants or variables.

We include the $\langle elli\rangle$ (*ellipsis*) construct to enable the processing of lists. This can be utilised when many instances of the same element within XML are to be mapped to many concept relations in the ontology (or vice versa). It is also possible to use the $\langle elli\rangle$ suffix with the $\langle concat\rangle$ oper-

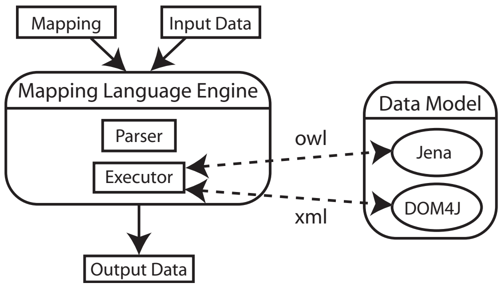

Figure 5. The mapping language engine design.

ator to indicate multiple element values that map to a single element. In both cases, the ellipsis construct preserves the order of list elements. The inspiration for the ellipsis construct came from the Scheme [10] macro language where it is used for list processing in a similar way.

4.2 Mapping Language Engine

Our Mapping Language Engine (MLE), pictured in Figure 5, is a JAVA component built on the Jena Framework[9] and Dom4J[10]. To carry out a transformation, the MLE can be passed a mapping statement and a source data structure. A source data structure may be an XML document (using OWL serialisations for ontology concepts) or a reference to an individual within a Jena Ontology Model. The MLE parses the mapping expression and builds a list of variable bindings from the source expression. The destination expression is then evaluated and a new data structure is created. The result can be returned in either XML (again using OWL syntax for the serialisation of ontology concepts) or as a reference to a newly created individual within the Jena Ontology Model.

Because we use Jena to store our OWL models, we can take advantage of the in-built reasoning it provides, the most useful of which is subsumption. Subsumption, usually denoted as $C \sqsubseteq D$, is the reasoning process through which the concept denoted by D (the *subsumer*) is checked to see if it is more general that the concept denoted by C (the *subsumee*). With Jena, this task is performed automatically when new concepts and individuals are introduced into the current Jena model. In our example, we see that the results from the first stage of the workflow (i.e. the sequence retreival services) can be `BSML_Sequence_Data` or `DDBJ_Sequence_Data` concepts. When the MLE creates a new individual to represent the service output from either of these services, it uses the approriate concept (ie. BSML or DDBJ sequence data). When inserted into the current model, Jena will automatically classify an individual of either concept as `Sequence_Data` too since it subsumes both concepts. Therefore, when creating the XML data set for input into the NCBI_Blast service, either concept type is valid and the sequence data can be extracted. With this approach, users have the freedom to extend existing ontology definitions with their own more specific concepts without breaking compatibility with other more general data models.

[9]http://jena.sourceforge.net/
[10]http://www.dom4j.org/

4.3 Service Invocation

To enable the execution of Web Services, we have created a Dynamic Web Service Invoker (DWSI). The DWSI takes an XML representation of the WSDL input and the service endpoint and invokes the service. The results of the service are returned in XML. In Figure 6, we show how the DWSI can be combined with the MLE to create a system that automatically mediates between different representations of the same data. This diagram shows one possible execution of our bioinformatics use case. In this instance, the XEMBL service is used to retrieve the sequence data after which it is passed to the NCBI_ Blast service for analysis. The first step is shown in the bottom left of the diagram where the accession id is passed to the XEMBL service. The result is a BSML formatted representation of the sequence data. This is then passed to the MLE, along with Mapping 1, where it is translated into a `BSML_Sequence_Data` concept and inserted into the Jena model. The uppermost box in Figure 6 shows a snapshot of the Jena model with the datatype properties holding example data. To enable the invocation of the NCBI_Blast service, the MLE takes Mapping 2 and creates an XML representation of the sequence data that is compatible with the blast service. This XML is then passed to a new instance of the DWSI which invokes the service. Finally, the results of the blast service are returned by the DWSI.

We have tested the performance cost of our preliminary prototype against hard coded XSLT transformations. On average, an XSLT transformation takes $30ms$ where our MLE takes approximately $190ms$ - six time more processing time to perform the same translation. We consider this an acceptable cost considering the high level of interoperability our system supports. This cost is also a small fraction of the network time required in a Web Service invocation which is usually around $5000ms$ or more.

5 Related Work

OWL-S [2] is a set of ontology definitions designed to capture the behaviour of services. The top level *service* ontology presents the service *profile*, a description of what the

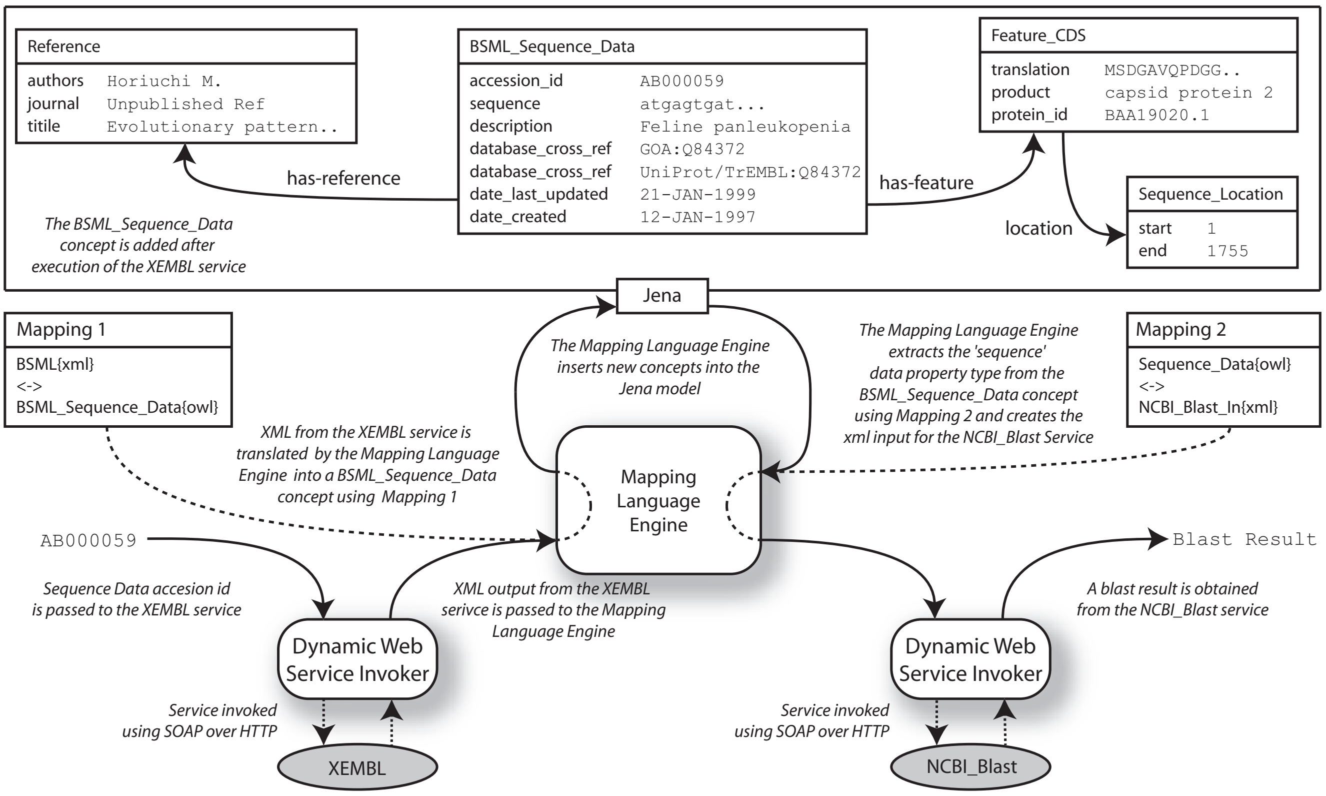
Reference
authors Horiuchi M.
journal Unpublished Ref
titile Evolutionary pattern..
BSML_Sequence_Data
accession_id AB000059
sequence atgagtgat...
description Feline panleukopenia
database_cross_ref GOA:Q84372
database_cross_ref UniProt/TrEMBL:Q84372
date_last_updated 21-JAN-1999
date_created 12-JAN-1997
Feature_CDS
translation MSDGAVQPDGG..
product capsid protein 2
protein_id BAA19020.1
Sequence_Location
start 1
end 1755
has-reference
has-feature
location
The BSML_Sequence_Data concept is added after execution of the XEMBL service
Jena
Mapping 1
BSML{xml} <-> BSML_Sequence_Data{owl}
Mapping 2
Sequence_Data{owl} <-> NCBI_Blast_In{xml}
The Mapping Language Engine inserts new concepts into the Jena model
The Mapping Language Engine extracts the 'sequence' data property type from the BSML_Sequence_Data concept using Mapping 2 and creates the xml input for the NCBI_Blast Service
XML from the XEMBL service is translated by the Mapping Language Engine into a BSML_Sequence_Data concept using Mapping 1
Mapping Language Engine
AB000059
Blast Result
Sequence Data accesion id is passed to the XEMBL service
XML output from the XEMBL serivce is passed to the Mapping Language Engine
A blast result is obtained from the NCBI_Blast service
Dynamic Web Service Invoker
Dynamic Web Service Invoker
Service invoked using SOAP over HTTP
Service invoked using SOAP over HTTP
XEMBL
NCBI_Blast

Figure 6. An example of how our Mapping Language Engine and Dynamic Web Service Invoker can be combined to automatically perform syntactic mediation.

service does (e.g. that a service is used to buy a book). The service is described by the service *model*, which tell us how the service works (e.g. a book buying service requires the customer to select the book, provide credit card details and shipping information and produces a transaction receipt). Finally, the service supports the service *grounding* which specifies the invocation method for the service. In the service grounding, XSLT is used to describe how OWL structures are converted to XML SOAP messages. This essentially performs the same task as our mapping language, but since it is based on transforming the XML serialisation of the OWL concepts, it is unable to utilise any reasoning techniques. For example, if we expressed the mapping from an instance of the `Sequence_Data` concept to the BLAST service input using XSLT, it would not be able to transform an instance of the `BSML_Sequence_Data` concept because the tag names used in its XML serialisation would be different.

The Web Services Modelling Ontology (WSMO) [13], adopts a different approach to OWL-S. They also intend to provide a framework to support automated discovery, composition, and execution of Web Services based on logical inference mechanisms, but with a specific focus on Enterprise Application Integration. Conceptually, WSMO is based on an event driven architecture so services do not directly invoke each other, instead goals are created by clients and submitted to the WSMO infrastructure which automatically manages the discovery and execution of services. Like OWL-S, WSMO uses ontologies to define formal models of information that have explicit semantics. However, the WSMO framework imposes a standardised message format (WSML) which WSMO participants use to communicate with each other. Message adapters can then be placed infront of existing components (such as WSDL Web Services and databases) to deal with the translations to and from traditional syntactic data structures. An example of such an adapter can be found in Section 5.3 of [11] which performs translations between WSML and Universal Business Language (UBL). With this approach the syntactic interface to a business service is hidden because its interface is exposed only through the WSMO framework. As such, explicit mappings from conceptual models to syntactic structures are not required.

The SEEK project [5] also address the problem of heterogeneous data representation in service oriented architectures. Within their framework, each service has a number of ports which expose a given functionality. Each port advertises a *structural type* which represents the format of the data the service is capable of processing. If the output of one service port is used as input to another service port, it is defined as *structurally valid* when the two types are the same. Each service port can also be allocated a *semantic type* which is defined by a reference to a concept within an OWL ontology. If two service ports are plugged together, they are *semantically valid* if the output from the first port is subsumed by the input to the second port. Structural types are linked to semantic types by a registration mapping using a custom mapping language based on XPATH. If the concatenation of two ports is semantically valid, but not structurally valid, an XQUERY transformation can be generated to integrate the two ports, making the link *structurally feasible*.

6 Conclusions and Further Work

In this paper, we have used a bioinformatics Grid application to show the problem of data integration in open, service oriented architectures. We have identified a typical scenario where different syntactic structures are used by service providers, and how this effects the workflow process. After presenting the motivation behind a framework to support the automated mediation of syntactic structures, we describe our solution, which is based on the use of Semantic Web technology. By mapping XML data structures to OWL concepts and properties, we can describe service inputs and outputs according to their conceptual types. When services are then plugged together, as in our use case where sequence data is retrieved from a database and passed to an alignment service, we can automatically transform data structures between different formats.

In terms of our mapping language, it would be useful to incorporate regular expression support for string matching. Our current language only provides a simple `split` operator that can be used to break down atomic string values into separate components. With regular expression support, we could allow users to specify more complex string manipulation functions.

Our current architecture assumes that mappings are known, therefore, it would be beneficial to create a mapping repository which exposes a query interface allowing users to register new mappings, discovery new mappings and identify the semantic type of a given XML fragment. Once such a registry has been implemented, we can integrate it with out MLE and DWSI so the appropriate mappings are retrieved automatically.

Finally, our last task is to formalise the link between syntactic type systems and the description logic models that underpin the OWL reasoning methods. We believe that a sound understanding of the problem will enable us to support a generic solution that is expressive enough to cope with a wide range of complex data structures.

7 Acknowledgment

This research is funded in part by EPSRC myGrid project (reference GR/R67743/01).

References

[1] UDDI technical white paper, September 2000.

[2] OWL-S: Semantic markup for web service. Technical report, The OWL Services Coalition, 2003.

[3] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, pages 34 – 43, 2001.

[4] S. Boag, D. Chamberlin, M. F. Fernandez, D. Florescu, J. Robie, and J. Simeon. Xquery 1.0: An XML query langauge. Technical report, W3C, 2003.

[5] S. Bowers and B. Ludascher. An ontology-driven framework for data transformation in scientific workflows. In *Intl. Workshop on Data Integration in the Life Sciences (DILS'04)*, 2004.

[6] E. Christensen, F. Curbera, G. Meredith, and S. Weerawarana. Web services description language (WSDL) 1.1, March 2001. W3C.

[7] J. Clark. XSL transformations (XSLT) version 1.0. Technical report, W3C, 1999.

[8] I. Foster, C. Kesslemann, J. M. Nick, and S. Tuecke. The physiology of the grid, an open grid services architecture for distributed systems integration, June 2002.

[9] T. R. Gruber. A translation approach to portable ontology specification. *Knowledge Acquisition*, (5):199–220, 1993.

[10] R. Kesley, W. Clinger, and J. Rees. Revised (5) report on the alogrithmic language scheme. *Higher-Order and Symbolic Computation*, pages 7 – 105, 1998.

[11] M. Moran. D13.5v0.1 WSMX implementation, July 2004. WSMO Working Draft.

[12] P. F. Patel-Schneider, P. Hayes, and I. Horrocks. OWL web ontology language semantics and abstract syntax. Technical report, W3C, 2004.

[13] D. Roman, H. Lausen, and U. Keller. D2v1.0. web service modeling ontology (WSMO), September 2004. WSMO Working Draft.