Dynamic Discovery of Composable Type Adapters for Practical Web Services Workflow

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

As the Web Services and Grid community adopt Semantic Web technology, we observe a shift towards higher-level workflow composition and service discovery practices. While this provides excellent functionality to non-expert users, more sophisticated middleware is required to hide the details of service integration. By investigating a common Bioinformatics use case, we observe the need for Type Adaptor components to be inserted into Workflows to harmonise syntactically incompatible interfaces. In this paper, we propose a generic Type Adaptor description approach that can be used in conjunction with existing service registries to facilitate automatic syntactic mediation. We demonstrate our implementation before evaluating both the translation approach we employ, and the relative cost of using a registry for Type Adaptor discovery.

1. Introduction

Workflow technology has been adopted by e-Science Grid applications to encode scientific processes, allowing users to perform *in silico* science [7]. The MYGRID (www.mygrid.org.uk) project supply an example of such a system, supporting Bioinformaticians in the construction, execution and sharing of Workflows through the Taverna (http://tavera.sf.net) graphical workbench. Recent advances in the MYGRID have focused on supporting users in the discovery and composition of services by using rich service annotations. FETA [9] has incorporated Semantic Web [3] technology into service descriptions using ontologies to capture the semantics of Web Services - essentially supplying users with conceptual definitions of what the service does using domain specific terminology. This has proven to be a valuable commodity in a system potentially making use of thousands of services where searching over service descriptions alone is a cumbersome and tedious task.

With the introduction of semantically annotated Web Services, Workflow composition in MYGRID has shifted to a higher-level design process: bioinformaticians can choose to include services in a Workflow to achieve particular goals based on the conceptual service definitions. While this makes Workflow design more accessible to untrained users, it does lead to more complex architectural requirements. Indeed, the situation often arises where users wish to connect two services together that are *conceptually compatible* but have different *syntactic interfaces*. To harmonise any data incompatibilities in a Workflow, additional processing is required, often taking the form of translation script, bespoke application, or Web Service. Within MYGRID, these *Type Adaptor* components must be discovered manually and inserted into the Workflow by hand, enforcing additional effort on the Bioinformatician. Consequently, the Bioinformatician is distracted from the scientific process at hand, spending additional time understanding why an incompatibility has been encountered and how it can be harmonised.

In this paper, we present a generic approach for describing Type Adaptors, which separates abstract functionality (i.e. the data types converted) from implementation (translation script, executable code, Web Services, etc). By using such a Type Adaptor description with a service registry, we show that it is possible to advertise Type Adaptors and discover them at run-time, aiding the user by automatically including the necessary conversion components. By combining the implementation of a composable translation language with the Grimoires service registry (www.grimoires.org), our proposed architecture is able to automate the discovery and execution of Type Adaptors in Web Service Workflows. We evaluate our implementation to show that the translation approach scales well, offers composability with little cost, and requires relatively low overhead for the use of Grimoires. Our contributions include:

1. An approach for Type Adaptor interface description in WSDL using Grimoires for advertising and discovery;
2. A language to describe Type Adaptors and an engine to process them and perform type conversion;
3. A complete system implementing the approach with empirical evaluation through benchmarking.

This paper is organised as follows: Section 2 introduces the need for Type Adaptors and presents a motivating example within a Bioinformatics scenario. In Section 3, the use of WSDL for Type Adaptor description and discovery is shown, before we present our implementation in Section 4. Evaluation of our implementation is given in Section 5, followed by the examination of related work in Section 6. Finally, we conclude and present further work in Section 7.

2. Motivation and Use Case

A typical task within Bioinformatics involves retrieving sequence data from a database and passing it to an alignment tool to check for similarities with other known sequences. Within MYGRID, this interaction is modelled as a simple Workflow, with each stage in the task being fulfilled by a Web Service, illustrated in Figure 1. Many Web Services are available for retrieving

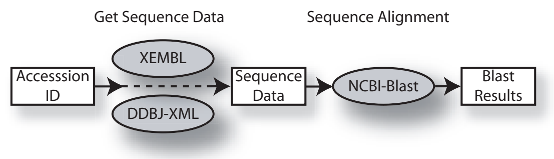
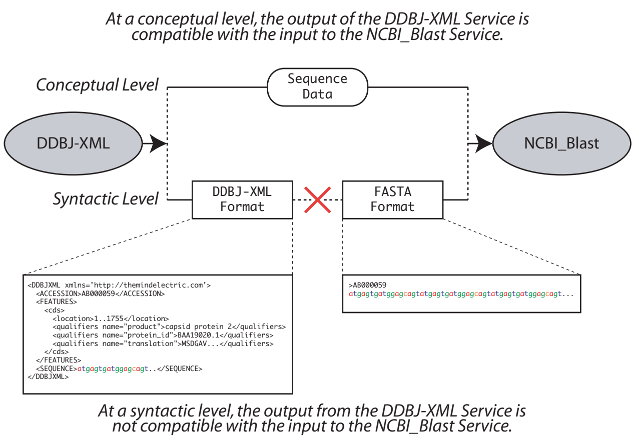


Figure 1. A simple bioinformatics task: get sequence data from a database and perform a sequence alignment on it.

sequence data; e.g. the ones used here are DDBJ-XML (`http://xml.ddbj.nig.ac.jp/`) and XEMBL (`http://www.ebi.ac.uk/xembl/`). To obtain a sequence data record, an accession number is passed as input to the service, which returns an XML document. This document, returned from either service, essentially contains the same information, 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 way this information is represented differs - XEMBL returns an INSD[1] formatted record whereas

[1]http://www.ebi.ac.uk/embl/Documentation/DTD/INSDSeq_v1.3.dtd.txt


At a conceptual level, the output of the DDBJ-XML Service is compatible with the input to the NCBI_Blast Service.

```
<DDBJXML xmlns='http://themindelectric.com'>
  <ACCESSION>AB000059</ACCESSION>
  <FEATURES>
    <cds>
      <location>1..1755</location>
      <qualifiers name="product">capsid protein 2</qualifiers>
      <qualifiers name="protein_id">BAA19020.1</qualifiers>
      <qualifiers name="translation">MSDGAV...</qualifiers>
    </cds>
  </FEATURES>
  <SEQUENCE>atgagtgatggagcagt..</SEQUENCE>
</DDBJXML>
```

```
>AB000059
atgagtgatggagcagtatgagtgatggagcagtatgagtgatggagcagt...
```

At a syntactic level, the output from the DDBJ-XML Service is not compatible with the input to the NCBI_Blast Service.

Figure 2. The output from the DDBJ-XML Service is not compatible for input to the NCBI-Blast Serivce.

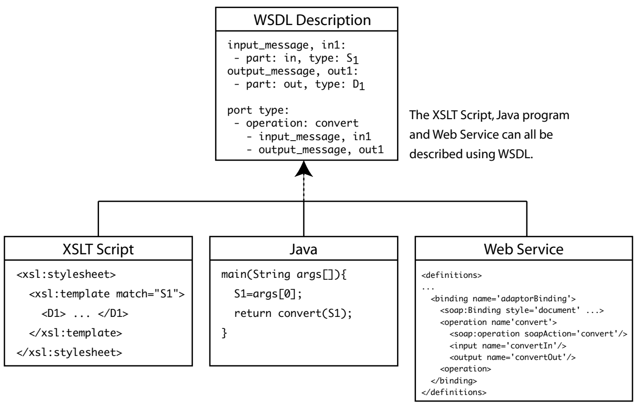
DDBJ-XML returns a document using their own custom format. The next stage in the Workflow is to pass the sequence data to an alignment service, such as the BLAST service at NCBI[2]. This service can consume a string of FASTA[3] formatted sequence data.

Intuitively, a Bioinformatician will view the two sequence retrieval tasks as the same type of operation, expecting both to be compatible with the NCBI-Blast service. The semantic annotations attached through FETA affirm this as the output types are assigned the same conceptual type, namely a Sequence Data Record concept. However, when plugging the two services together, we see that the output from either sequence data retrieval service is not directly compatible for input to the NCBI Blast service (Figure 2). To harmonise the Workflow, some intermediate processing is required on the data outputted from the first service to make it compatible for input to the second service. We define this translation step as *syntactic mediation*, an additional Workflow stage that is carried out by a particular class of programs we define as *Type Adaptors*.

3. Generic Type Adaptor Description

To augment the manual selection of programs or services to harmonise data incompatibles in Web Service Workflows, we propose a solution that utilises a registry of Type Adaptors, each of them described by WSDL, to support the automated discovery of harmonisation components. In this Section, we characterise a generic approach for the description, sharing and discovery of Type Adaptors and how they can be used to perform syntactic mediation in

[2]http://www.ncbi.nlm.nih.gov/BLAST/
[3]http://www.ebi.ac.uk/help/formats_frame.html


WSDL Description
The XSLT Script, Java program and Web Service can all be described using WSDL.
XSLT Script
Java
Web Service

```
input_message, in1:
 - part: in, type: S1
output_message, out1:
 - part: out, type: D1

port type:
 - operation: convert
   - input_message, in1
   - output_message, out1
```

```
<xsl:stylesheet>
  <xsl:template match="S1">
    <D1> ... </D1>
  </xsl:template>
</xsl:stylesheet>
```

```
main(String args[]){
  S1=args[0];
  return convert(S1);
}
```

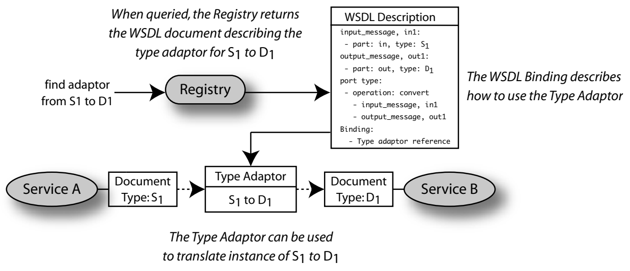
```
<definitions>
...
  <binding name='adaptorBinding'>
    <soap:Binding style='document' ...>
    <operation name'convert'>
      <soap:operation soapAction='convert'/>
      <input name='convertIn'/>
      <output name='convertOut'/>
    <operation>
  </binding>
</definitions>
```

Figure 3. Using WSDL to describe different Type Adaptors

workflows.

There are many applications and tools that support the translation of data between different formats. XSLT [6] enables the specification of data translation in a script format using pattern matching and template statements. Such a script can be consumed by an XSLT engine to drive the translation of data to a different representation. Other forms of Type Adaptors are not so transparent; a black box approach is used frequently in MYGRID where users create custom translation programs using languages such as JAVA and Perl. In other cases, a Type Adaptor may take the form of a distinct mediator Web Service, described by WSDL and executed using SOAP over HTTP. Any of these Type Adaptors can be viewed as a component that converts data from a source type to a destination type. To describe the capabilities of all Type Adapters, irrespective of implementation, we separate concrete implementation details from the abstract definition. Under this assumption, all Type Adaptors can be described using WSDL [5].

WSDL is a declarative language used to specify service capabilities and how to access them through the definition of service end-points. The operations implemented by the service are defined in terms of the messages consumed and produced, the structure of which is specified by XML Schema. The service, operations and messages are described at an abstract level and bound to a concrete execution model via the service binding. The service binding describes the type of protocol used to invoke the service and the requested datatype encoding. Because of this two-tier model, many different Web Service implementations may be viewed through a common interface. By applying the same principle to data harmonisation components, we can use WSDL to describe the capabilites of any Type Adaptor. Using this approach allows different implementations of the same Type Adaptor


When queried, the Registry returns the WSDL document describing the type adaptor for S1 to D1
find adaptor from S1 to D1
Registry
WSDL Description
The WSDL Binding describes how to use the Type Adaptor
Service A
Document Type: S1
Type Adaptor
S1 to D1
Document Type: D1
Service B
The Type Adaptor can be used to translate instance of S1 to D1

```
input_message, in1:
 - part: in, type: S1
output_message, out1:
 - part: out, type: D1
port type:
 - operation: convert
   - input_message, in1
   - output_message, out1
Binding:
  - Type adaptor reference
```

Figure 4. The use of a registry to discover Type Adaptors

to be described with the same abstract definition (i.e. in terms of the input and output XML schema types) and different bindings. This is illustrated in Figure 3, where three Type Adaptors are shown: an XSLT script, a JAVA program and a SOAP Web Service, all providing the same functionatlity - to convert data of type S_1 to D_1.

With a uniform method for the description of Type Adaptors in the form of WSDL, we can utilise existing registry technologies to support sharing and discovery - this feature is described in more detail in Section 4 where we present our implementation. Figure 4 shows a high level view of how a registry containing WSDL definitions of Type Adaptors can be used in a Web Service workflow to perform syntactic mediation. The output from Service A, of type S_1 is used as input to Service B, which consumes type D_1. The binding section of the WSDL definition describes how to execute the translator, for example, by providing the location of an XSLT script or the JAVA method details.

4. Architecture

In this Section, we describe an architecture that utilises the Type Adaptor description outlined above. After presenting the motivation for an intermediary representation, we briefly describe our mapping language, FXML-M, that is used to specify Type Adaptor behaviour, present our Type Adaptor registration and discovery implementation, and show it working against our use case scenario.

4.1. Scalability and Reuse

As the amount of Web Services is increasing, currently over 1000 in MYGRID [9, 8], scalability and reuse is an important issue. In the simplest case, we assume a Type Adaptor exists to convert data directly between every compatible representation. For n compatible data formats, $O(n^2)$ Type Adaptors are required to achieve maximum interoperability. Therefore, as more Web Services are introduced, the number of compatible interfaces increases and a quadratic expansion for the number of Type Adaptors will occur. Also, when introducing a new representation for information that is already present in other formats, Type Adaptors must be created to transform the new representation to all other formats.

By introducing an intermediate representation, to which all data formats are converted, the problem of scalability can be diminished. For n compatible interfaces, $O(n)$ Type Adaptors are required, resulting in a linear complexity as more services are added. When introducing new formats, only one Type Adaptor is required to convert the new data format to and from the intermediate representation. Therefore, our Architecture is based around the use of an intermediary representation. When considering the mechanisms necessary to support users in the use and agreement of a common representation, we see existing work already tackles a similar problem within MYGRID. FETA uses descriptions that supply users with conceptual definitions of what the service does using domain specific terminology. Part of this solution involves the construction of a large ontology for the bioinformatics domain covering the types of data shown in our use case. Therefore, we extend these existing OWL ontologies to capture the semantics and structure of the data representation.

In terms of sharing and reuse, it is common for Web Services to supply many operations that operate over the same, or subsets of the same data. Therefore, the transformation for a given source type may come in the form of a Type Adaptor designed to cater for multiple types. Hence, the generation of a WSDL description for a given Type Adaptor should capture *all* the transformation capabilities of the adaptor. This can be achieved by placing multiple operation definitions in the WSDL, one for each of the possible Type conversions.

4.2. Bindings

To describe the relationship between XML schema components and OWL concepts / properties, we devised a mapping language, FXML-M (Formalised XML mapping), described in previous work [15]. FXML-M is a composable mapping language in which mapping statements translate XML schema components from a source schema to a destination schema. FXML-T (Formalised XML Translator) is an interpreter for FXML-M which consumes an *M-Binding* (collection of mappings) and the source XML document and produce a destination document. Broadly, an M-Binding B contains a sequence of mappings $m_1, m_2, \ldots, m_n$. In the style of XML schema, M-Bindings may also import other M-Bindings (e.g. $B_1 = \{m_1, m_2\} \cup B_2$) to support composition - an important feature when services offer operations over multiple XML schemas. M-Bindings themselves are XML documents which be viewed as a specialised mini Workflow for type conversion.

4.3. Binding Publication and Discovery

Since our M-Binding language is used to drive XML to XML document conversion, and our intermediary language is an OWL ontology, conversion between XML instances and OWL concept instances is specified in terms of a canonical XML representation for OWL individuals. While the use of XML to represent OWL concepts is common, XML schemas to describe these instances are not available. Hence, we generate XML schemas (OWL instance schema) to describe valid concept instances for a given ontology.

With an intermediary schema in place (in the form of an OWL instance schema), the appropriate M-Bindings to translate data to and from XML format, and FXML-T (the FXML-M translator), Web Service harmonisation is supported. To fully automate the process, i.e. discover the appropriate Bindings (or Type Adaptors) based on translation requirements, we require a registry to advertise M-Bindings. Since Type Adaptors can be described using WSDL, we use the Grimoires service registry (www.grimoires.org) to record, advertise and supply adaptors. Grimoires is an extended UDDI [1] registry that enables the publishing and querying of Service interfaces while also supporting semantic annotations of service descriptions [11]. Figure 5 illustrates how Bindings are registered with Grimoires using the *Binding Publisher Service*.

The *Binding Publisher Service* consumes an M-Binding, along with the source and destination XML schemas, and produces a WSDL document that describes the translation capability of the M-Binding. This generated WSDL in then publicly hosted and registered with Grimoires using the `save_service` operation. Afterwards, an M-Binding WSDL may be discovered using the standard Grimoires

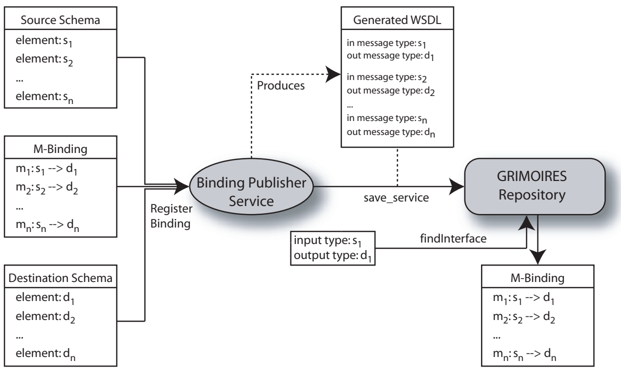


Figure 5. Registration and discovery of Binding using the Grimoires repository

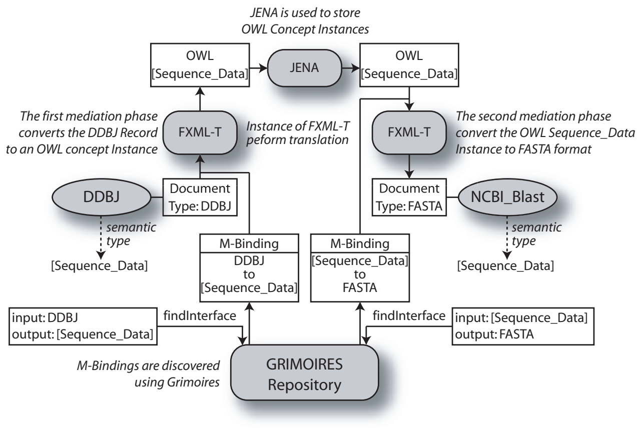

JENA is used to store OWL Concept Instances
OWL [Sequence_Data]
JENA
OWL [Sequence_Data]
The first mediation phase converts the DDBJ Record to an OWL concept Instance
FXML-T
Instance of FXML-T peform translation
FXML-T
The second mediation phase convert the OWL Sequence_Data Instance to FASTA format
DDBJ
Document Type: DDBJ
Document Type: FASTA
NCBI_Blast
semantic type
semantic type
[Sequence_Data]
M-Binding DDBJ to [Sequence_Data]
M-Binding [Sequence_Data] to FASTA
[Sequence_Data]
input: DDBJ output: [Sequence_Data]
findInterface
findInterface
input: [Sequence_Data] output: FASTA
M-Bindings are discovered using Grimoires
GRIMOIRES Repository

Figure 6. Automated syntatic mediation for our use case using Grimoires to store Binding descriptions and FXML-T for document translation

query interface (the `findService` operation) based on the input and output schema types desired.

4.4. Automated Mediation

To achieve syntactic mediation, two M-Bindings are necessary: a *realisation M-Binding* (to convert XML to the intermediate OWL representation), and a *serialisation M-Binding* (to convert OWL to XML). Figure 6 is an expansion of Figure 4 showing the use of Grimoires and FXML-T to automate syntactic mediation for our use case. When registering the DDBJ and NCBI-Blast services with FETA, semantic annotation are used to specify the input and output *semantic types* using a reference to a concept in the Bioinformatics ontology - in this case the `Sequence_Data` concept. At workflow composition time, the Bioinformatician may wish to feed the output from the DDBJ service into the NCBI-Blast service because they are deemed semantically compatible (i.e. they share the same semantic type). While this workflow can be specified, it is not executable because of the difference in data formats assumed by each service provider - a stage of syntactic mediation is required. Figure 6 shows the mediation phase, using Grimoires to discover M-Bindings, FXML-T to perform the translation from DDBJ format to FASTA format with JENA used to hold the intermediate representation (in the form of an OWL concept instance).

5. Evaluation

To evaluate this approach, we have examined the performance of our Translation Engine (FXML-T) and the use of Grimoires as a repository for advertising M-Binding documents. The aims are threefold: (i) to test the scalability of our mapping language approach; (ii) to establish FXML-T as scalable translation engine by examining the performance costs against increasing document sizes, increasing schema sizes, and increasingly complex M-Binding composition; (iii) to confirm the use of Grimoires for M-Binding discovery is not a significant overhead in the context of Workflow execution. The following sub-Sections describe each of the tests with hypothesis given in *italics*. All tests were carried out using a 2.6 Ghz Pentium4 PC with 1GB RAM running Linux (kernel 2.6.15-20-386) using unix `time` to record program execution times. FXML-T is implemented in SCHEME and run using the Guile Scheme Interpreter (v1.6). Tests in Section 5.1 are run 10 times with a mean value plotted.

5.1. Translation Engine Scalability

Expanding document and schema size will increase the translation cost linearly or better.
We test the scalability of FXML-T in two ways: by increasing input document size (while maintaining uniform input XML schema size), and by increasing both input schema size and input document size. We test FXML-T against the following XML translation tools:

- XSLT: Using Perl and the XML::XSLT module[4].
- XSLT: Using JAVA (1.5.0) and Xalan[5](v2.7.0).
- XSLT: Using Python (v2.4) and the 4Suite Module[6](v0.4).
- SXML: A SCHEME implemention for XML parsing and conversion (v3.0).

Since FXML-T is implemented using an interpreted language, and Perl is also interpreted, we would expect them to peform slowly in comparison to JAVA and Python XSLT which are compiled[7].

Figure 7 shows the time taken to transform a source document to a structurally identical destination document for increasing document sizes. The maximum document size tested is 1.2 MB, twice that of the Blast results obtained in our use-case. From Figure 7 we see that FXML-T has a linear expansion in transformation time against increasing document size. Both Python and JAVA implementations also scale linearly with better peformance than FXML-T due to JAVA and Python using compiled code. Perl exhibits the worst performance in terms of time taken, but a linear expansion is still observed.

Our second performance test examines the translation cost with respect to increasing XML schema size. To perform this test, we generate structurally equivalent source and destination XML schemas and input XML documents

[4]http://xmlxslt.sourceforge.net/
[5]http://xml.apache.org/xalan-j/
[6]http://4suite.org/
[7]although Python is interpreted, the 4Suite library is statically linked to natively compiled code

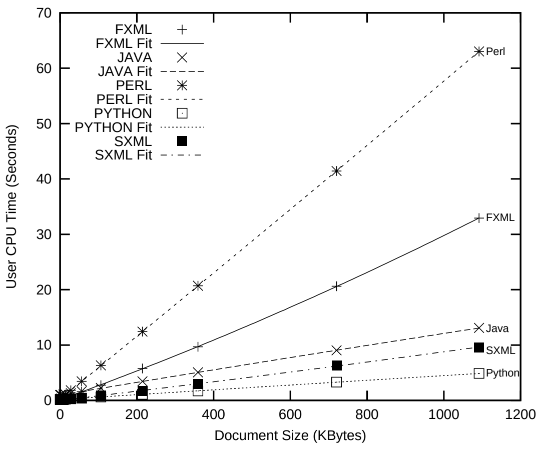


Figure 7. Transformation Performance against increasing XML Document Size

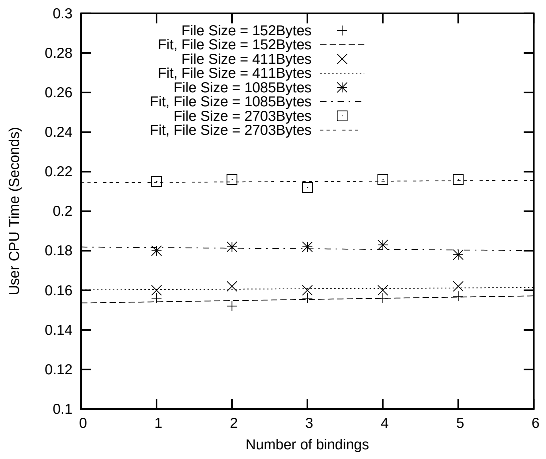


Figure 9. Transformation Performance against number of bindings

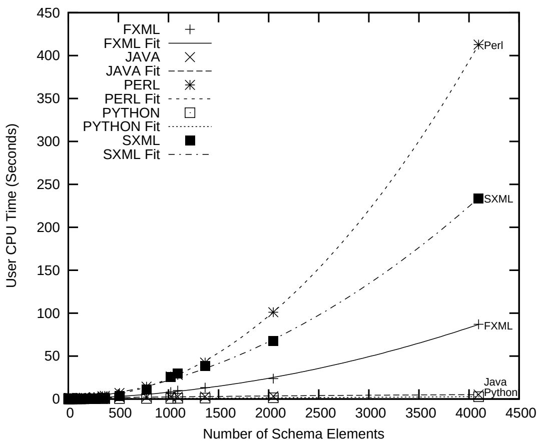


Figure 8. Transformation Performance against increasing XML Schema Size

which satisfy them. XML input document size is directly proportional to schema size; with 2047 schema elements, the input document is 176KBytes, while using 4095 elements a source document is 378KBytes. Figure 8 shows translation time against the number of schema elements used. Python and JAVA perform the best - a linear expansion with respect to schema size that remains very low in comparison to FXML-T and Perl. FXML-T itself has a quadratic expansion; however, upon further examination, we find the quadratic expansion emanates from the XML parsing sub-routines used to read schemas and M-Bindings, whereas the translation itself has a cost linear to the size of its input. The SCHEME XML library used for XML parsing is common to FXML-T and SXML, hence the quadratic expansion for SXML. Therefore, our translation approach is linear and can be implemented with a suitable XML parser.

5.2. Composition Cost

Binding composition comes with virtually no performance cost.
One important feature of our translation language (FXML) is the ability to compose M-Bindings at runtime. This can achieved by creating an M-Binding that includes individual mappings from an external M-Binding, or imports all mappings from an external M-binding. For Service interfaces operating over multiple schemas, M-Bindings can be composed easily from existing Type adaptors. Ideally, this composability should come with minimal cost. To examine M-Binding cost, we increased the number of M-Bindings and observed the time required to transform the document. To perform the translation, 10 mappings are required $m_1, m_2, \ldots, m_{10}$. M-Binding 1 contains all the required mapping statements: $B_1 = \{m_1, m_2, \ldots, m_{10}\}$. M-Binding 2 is a composition of two M-Bindings where $B_2 = \{m_1, \ldots, m_5\} \cup B_{2a}$ and $B_{2a} = \{m_6, \ldots, m_{10}\}$. To fully test the cost of composition, we increased the number of M-Bindings used and run each test using 4 source documents with sizes 152Bytes, 411Bytes, 1085Bytes, and 2703Bytes. While we aim for zero composability cost, we would expect a small increase in translation time as more M-Bindings are included. By increasing source document size, a larger proportion of the translation time will be spent on reading in the document and translating it. Consequently, the relative cost of composing M-Bindings will be greater for smaller documents and therefore the increase in cost should be greater. Figure 9 shows the time taken to transform the same four source documents against the same mappings distributed across an increasing number of M-Bindings. On the whole, a very subtle increase in performance cost is seen, with the exception of the file size

1085Bytes. We attribute this anomaly to the rate of error in time recording which is only accurate to ± 10 milliseconds.

5.3. Registry Cost

The cost of M-Binding discovery using Grimoires is not significant when compared to cost of executing the target service.
Our final test examines the cost of using Grimoires to discover the required M-Bindings. While this feature facilitates automation, it will require additional web service invocations in a workflow.

Activity	*Average*
DDBJ Execution	2.50
Realisation Discovery	0.22
Realisation Transformation	0.47
Jena Mediation	0.62
Serialisation Discovery	0.23
Serialisation Translation	0.27
Total Mediation	1.81

The Table above shows the average time taken (from 5 runs) for a type translation using OWL as an intermediarry representation. These types of translation consist of 5 steps: (i) discover realisation M-Binding, (ii) transformation of result to OWL instance, (iii) import OWL instance to JENA KB, (iv) discovery of serialisation M-Binding, (v) trasformation of OWL instance to XML. Results show that the total mediation time is roughly 2 seconds, with the largest portion of the time taken importing the OWL instance into JENA. The discovery overhead (realisation and serialisation M-Bindings) is acceptable in comparison to service execution and translation times.

6. Related Work

The Interoperability and Reusability of Internet Services (IRIS) project have also recognised the need to assist Bioinformaticians in the design and execution of Workflows. Radetzki *et al* [13] describe Adaptor services using WSDL with additional profiles in the Mediator Profile Language (MPL)[8]. Adaptor services are collated in a *Mediator Pool* and queried using a derivation from the input and output descriptions of services connected by a dataflow. The query is a combination of syntactic information and semantic concepts from a domain ontology. A Matchmaking algorithm presents a ranked list of Adaptors, some of which may be composed from multiple Adaptor instances. To aid the Adaptor description stage, WSDL descriptions of services and their documentation are linguistically analised to establish the sort of service. For example, the 'getNucSeq' method is decomposed into the terms 'get', 'nuc', and 'seq'. To this end, the matching algorithm is relaxed and not intended to be used automatically; instead, users are aided during workflow composition.

[8]http://www.cs.uni-bonn.de/III/bio/iris/MediatorProfile.owl

Within the SEEK framework [4], each service has a number of ports which expose a given functionality. Each port advertises a *structural type* which defines the input and output data format by a reference to an XML schema type. 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 equal. Each service port can also be allocated a *semantic type* which is specified by a reference to a concept within an 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*. The SEEK system provides data integration between different logical organisations of data using a common conceptual representation, the same technique that we adopt. However, their work is only applicable to services within the bespoke SEEK framework. The architecture we present is designed to work with arbitrary WSDL Web Services annotated using conventional semantic Web Service techniques.

Hull *et al* [8] dictate that conversion services, or *shims*, can be placed in between service whenever some type of translation is required - exactly as the current MYGRID solution. They explicitly specify that a shim service is *experimentally neutral* in the sense that it has no side-effect on the result of the experiment. By enumerating the types of shims required in bioinformatics Grids and classifying all instances of shim services, it is hoped that the necessary translation components could be automatically inserted into a workflow. However, their focus is not on the translation between different data representation, rather the need to manipulate data sets; extracting information from records, finding alternative sources for data, and modifying workflow designs to cope with iterations over data sets.

Moreau *et al* [12], have investigated the same problem within the Grid Physics Network, GriPhyn[9]. To provide a homogeneous access model to varying data sources, Moreau *et al* propose a separation between logical and physical file structures. This allows access to data sources to be expressed in terms of the logical structure of the information, rather than the way it is physically represented. The XML Data Type and Mapping for Specifying Datasets

[9]http://griphyn.org/

(XDTM) prototype provides an implementation which allows data source to be navigated using XPATH. While this approach is useful when amalgamating data from different physical representations (i.e. XML files, binary files and directory structures), it does not address the problem of data represented using different logical representations (i.e. different schemas with the same physical representation). Our service integration problem arises from the fact that different service providers use different logical representations of conceptually equivalent information.

7. Conclusions and Future Work

In this paper, we have described the motivation for a generic Type Adaptor description policy to support automated Workflow harmonisation when syntactic incompatibilities are encountered. By using WSDL to describe Type Adaptor capabilities, and the Grimoires registry to advertise them, translation components can be discovered at run-time and placed into the running workflow. Using FXML-M as a Type Adaptor language and OWL ontologies as an intermediate representation, we show an implementation that supports a common Bioinformatics use case. Evaluation of our Binding language approach, its implementation, and the use of Grimoires as a registry shows our architecture is scalable, supports Type Adaptor composablity with virtually no cost, and has a relatively low overhead for Binding discovery.

While our Architecture has been based on Type Adaptors taking the form of FXML-M Binding documents, the approach will work with any Type Adaptor language, such as XSLT and JAVA, providing the appropriate code is put in place to automatically generate WSDL descriptions of their capabilities and the Workflow engine is able to interpret their WSDL Binding (i.e. execute an XSLT script, run a JAVA program, invoke a particular SOAP service). Even though our Architecture has been designed to fit within the existing MYGRID application, this approach will apply to any Grid or Web Services architecture. When incorporating the use of Semantic Web technology, namely the association of WSDL message parts with concepts from an ontology, we have followed existing practices such as those used by the FETA system, OWL-S [10], WSMO [14], and WSDL-S [2].

In future work, we aim to improve the FXML-T implementation in the area of XML parsing. This will make FXML-T translation cost scale at an acceptable rate when increasing both XML document size and XML schema size.

8. Acknowledgment

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

References

[1] UDDI technical white paper, September 2000.

[2] R. Akkiraju, J. Farrell, J.Miller, M. Nagarajan, M. Schmidt, and A. S. K. Verma. Web service semantics - WSDL-S. Technical report, UGA-IBM, 2005.

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

[4] 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.

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

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

[7] C. Goble, S. Pettifer, R. Stevens, and C. Greenhalgh. Knowledge Integration: In silico Experiments in Bioinformatics. In I. Foster and C. Kesselman, editors, *The Grid: Blueprint for a New Computing Infrastructure Second Edition*. Morgan Kaufmann, November 2003.

[8] D. Hull, R. Stevens, and P. Lord. Describing web services for user-oriented retrieval. 2005.

[9] P. Lord, P. Alper, C. Wroe, and C. Goble. Feta: A light-weight architecture for user oriented semantic service discovery. In *The Semantic Web: Research and Applications: Second European Semantic Web Conference, ESWC 2005, Heraklion, Crete, Greece*, pages 17 – 31, Jan. 2005.

[10] D. Martin, M. Burstein, G. Denker, J. Hobbs, L. Kagal, O. Lassila, D. McDermott, S. McIlraith, M. Paolucci, B. Parsia, T. Payne, M. Sabou, E. Sirin, M. Solanki, N. Srinivasan, and K. Sycara. OWL-S: Semantic markup for web service. Technical report, The OWL Services Coalition, 2003.

[11] S. Miles, J. Papay, T. Payne, M. Luck, and L. Moreau. Towards a protocol for the attachment of metadata to service descriptions and its use in semantic discovery. *Scientific Programming*, pages 201–211, 2005.

[12] L. Moreau, Y. Zhao, I. Foster, J. Voeckler, and M. Wilde. XDTM: the XML Dataset Typing and Mapping for Specifying Datasets. In *Proceedings of the 2005 European Grid Conference (EGC'05)*, Amsterdam, Nederlands, Feb. 2005.

[13] U. Radetzki, U. Leser, S. Schulze-Rauschenbach, J. Zimmermann, J. Lussem, T. Bode, and A. Cremers. Adapters, shims, and glue - service interoperability for in silico experiments. *Bioinformatics*, 22(9):1137–1143, 2006.

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

[15] M. Szomszor, T. R. Payne, and L. Moreau. Automatic syntactic mediation for web service integration. In *Submitted to International Conference on Web Services 2006*.