Bioinformatics Service Reconciliation By Heterogeneous Schema Transformation

Lucas Zamboulis[1,2], Nigel Martin[1], and Alexandra Poulovassilis[1]

[1] School of Computer Science and Information Systems, Birkbeck, Univ. of London
[2] Department of Biochemistry and Molecular Biology, University College London

Abstract. This paper focuses on the problem of bioinformatics service reconciliation in a generic and scalable manner so as to enhance interoperability in a highly evolving field. Using XML as a common representation format, but also supporting existing flat-file representation formats, we propose an approach for the scalable semi-automatic reconciliation of services, possibly invoked from within a scientific workflows tool. Service reconciliation may use the AutoMed heterogeneous data integration system as an intermediary service, or may use AutoMed to produce services that mediate between services. We discuss the application of our approach for the reconciliation of services in an example bioinformatics workflow. The main contribution of this research is an architecture for the scalable reconciliation of bioinformatics services.

1 Introduction

In recent years, the bioinformatics field has seen an explosion in the number of services offered to the community. These platform-independent software components have consequently been used for the development of complex tasks through service composition within workflows, thereby promoting reusability of services. However, the large number of services available impedes service composition and so developing techniques for semantic service discovery that would significantly reduce the search space is of great importance [12].

After discovering services that are relevant to one's interests, the next step is to identify whether these services are functionally compatible. Bioinformatics services are being independently created by many parties worldwide, using different technologies and data types, hindering integration and reusability [21]. In particular, after discovering two such services, the researcher needs to first identify whether the output of the first is compatible with the input of the second based on a number of factors, such as the technology employed by each service, the representation format and the data type used.

In practice, compatible services are rare. Within Taverna (see `http://taverna.sourceforge.net`), service technology reconciliation is addressed by using Freefluo [19], an extensible workflow enactment environment that bridges the gap between web services and other service types, such as web-based REST services (stateless services that support caching). However, the researcher still needs to reconcile the outputs and inputs of services in terms of content, data type and representation format, spending time and effort in developing functionality that, even though essential for the services to interoperate, is irrelevant to the experiment.

The primary cause of this problem is the existence of multiple different data types and representation formats used even for basic concepts, such as DNA sequences. These data types and representation formats, used for the same or overlapping concepts, have been developed over the years by collaborative work between researchers and/or industry and so even though standardisation efforts are important and encouraged by the community, non-standardised efforts are likely to persist and new ones are bound to appear in this constantly evolving field. For this reason, service composition solutions that take into consideration this factor are essential. Unfortunately, most current tools concentrate on a specific data type and representation format (or combinations of pairs of types and formats, when translation is needed) to accomplish a highly specific task, rather than being generic [13]. As a result, reusability of existing tools is low.

Another common practice in bioinformatics is the use of flat-file representation formats for the overwhelming majority of data types, while the adoption rate of XML is low. This practice does not allow the application of Semantic Web technologies and solutions to their full extent, such as semantically annotating fields within a bioinformatics data type. For example, even though it is possible to annotate a service as having FASTA output, it is not possible to annotate the different fields within the non-tagged FASTA data type. But, even if a data type *is* tagged, e.g. UniProt, annotation cannot be performed in a generic way, as it would require data type-specific annotation tools.

We also observe that, even though the use of semantic annotations is key to service discovery and composition, service providers are disinclined to supply comprehensive annotations for their services. Relying on a centralised approach for such a task is clearly not scalable, and so any proposed solution for the reconciliation of bioinformatics services must ensure that the amount of required annotations is kept to a minimum and that it is reused as much as possible.

We argue that (a) the use of XML and (b) allowing the annotation and manipulation of service inputs and outputs at a fine-grained level, can boost service interoperability in a scalable manner. We therefore propose and exemplify an architecture for the reconciliation of services by exploiting the (manual) semantic annotation of service inputs and outputs using one or more interconnected ontologies, and the subsequent automatic restructuring of the XML output of one service to the required XML input of another. Although our approach uses XML as the common representation format, non-XML services are also supported by the use of converters to and from XML. Our schema and data transformation approach is supported by the AutoMed heterogeneous data integration system (see `http://www.doc.ic.ac.uk/automed`) and can accommodate two types of service reconciliation: either using AutoMed as a service itself, e.g. from within a workflow tool, or using AutoMed to generate mediating services.

In the remainder of this paper, Section 2 first reviews current approaches related to service interoperability. Section 3 then provides an overview of the

AutoMed system, to the level of detail necessary for this paper. Section 4 introduces our proposed approach for a scalable solution to the problem of bioinformatics service reconciliation. Section 5 presents our ongoing work in applying our approach to the reconciliation of bioinformatics services. Section 6 provides an overall discussion of our approach and gives our plans for future work.

2 Related Work

In the context of service composition, research such as [20, 18, 2] has mainly focused on service technology reconciliation, matchmaking and routing, assuming that service inputs and outputs are a priori compatible. This assumption is restrictive, as it is often the case that two services are semantically compatible, but cannot interoperate due to data type and/or representation format mismatches.

This problem has forced service consumers to handle such mismatches with custom code from within the calling services. In an effort to minimise this issue and promote service reusability, ^{my}Grid (see `http://www.mygrid.org.uk`) has fostered the notion of *shims* [7], i.e. services that act as intermediaries between services and reconcile their inputs and outputs. However, a new shim needs to be manually created for each pair of services that need to interoperate. [8] states that, even though in theory the number of shims that ^{my}Grid needs to provide is quadratic in the number of services it contains, the actual number of shims should be much smaller. However, this manual approach is not scalable, as in 2005 ^{my}Grid gave access to 1,000 services [12] and this number is now over 3,000.

[3] describes a scalable framework that uses mappings to one or more ontologies, possibly containing subtyping information, for reconciling the output of a service with the input of another. The sample implementation of this framework is able to use mappings to a single ontology in order to generate an XQuery query as the transformation program.

We observe that [3] only provides for shim generation, whereas our approach, by using the AutoMed data integration system, provides a uniform approach to workflow and data integration, both of which are key aspects of in silico biological experiments. Furthermore, the work presented here differs from [3] in a number of aspects and provides a more generic solution to the problem of bioinformatics service reconciliation. First, we also consider services that produce or consume non-XML data and also allow primitive data type reconciliation, whereas [3] does not. Moreover, we allow 1-n GLAV correspondences, compared to the 1-1 LAV correspondences of [3] and we also define a methodology for reconciling services that correspond to more than one ontology. We also note that our XML restructuring algorithm is able to avoid loss of information during data transformation, by analysing the hierarchical nature of the source and target schemas and by using subtype information provided by the ontologies.

[22] also uses a mediator system for service composition. However, the focus is either to provide a service over the global schema of the mediator whose data sources are services, or to generate a new service that acts as an interface over other services. In contrast, we use the AutoMed toolkit to reconcile a sequence

of semantically compatible services that need to form a pipeline: there is no need for a single 'global schema' or a single new service to be created.

Concerning the use of ontologies for data integration, a number of approaches have been proposed. For example, [1] uses an ontology as a virtual global schema for heterogeneous XML data sources using LAV mapping rules, while [4] undertakes data integration using mappings between XML data sources and ontologies, transforming the source data into a common RDF format. In contrast, we use XML as the common representation format and focus on restructuring the source data into a target XML format, rather than on integration.

3 Overview of AutoMed

AutoMed is a heterogeneous data transformation and integration system which offers the capability to handle virtual, materialised and hybrid data transformation/integration across multiple data models. It supports a low-level **hypergraph-based data model (HDM)** and provides facilities for specifying higher-level modelling languages in terms of this HDM. An HDM schema consists of a set of nodes, edges and constraints, and each modelling construct of a higher-level modelling language is specified as some combination of HDM nodes, edges and constraints (the constraints are expressed in the IQL query language — see below).

For any modelling language $\mathcal{M}$ specified in this way (via the API of AutoMed's Model Definitions Repository) AutoMed provides a set of primitive schema transformations that can be applied to schema constructs expressed in $\mathcal{M}$. In particular, for every construct of $\mathcal{M}$ there is an add and a delete primitive transformation which add to/delete from a schema an instance of that construct. For those constructs of $\mathcal{M}$ which have textual names, there is also a rename primitive transformation.

Instances of modelling constructs within a particular schema are identified by means of their *scheme* enclosed within double chevrons $\langle\langle \ldots \rangle\rangle$. AutoMed schemas can be incrementally transformed by applying to them a sequence of primitive transformations, each adding, deleting or renaming just one schema construct (thus, in general, AutoMed schemas may contain constructs of more than one modelling language). A sequence of primitive transformations from one schema X_1 to another schema X_2 is termed a *pathway* from X_1 to X_2 and denoted by $X_1 \rightarrow X_2$. All source, intermediate, and integrated schemas, and the pathways between them, are stored in AutoMed's Schemas & Transformations Repository.

Each add and delete transformation is accompanied by a query specifying the extent of the added or deleted construct in terms of the rest of the constructs in the schema. This query is expressed in a functional query language, IQL [9]. Also available are extend and contract primitive transformations which behave in the same way as add and delete except that they state that the extent of the new/removed construct cannot be precisely derived from the rest of the constructs. Each extend and contract transformation takes a pair of queries that specify a lower and an upper bound on the extent of the construct. These bounds may be Void or Any, which respectively indicate no known information about the lower or upper bound of the extent of the new construct.

The queries supplied with primitive transformations can be used to translate queries or data along a transformation pathway $X_1 \rightarrow X_2$ (see [15, 16] for details). For translating data from X_1 to data on X_2 the add, extend and rename steps are used. The queries supplied with primitive transformations also provide the necessary information for these transformations to be automatically *reversible*, in that each add/extend transformation is reversed by a delete/contract transformation with the same arguments (including the same query arguments), while each rename is reversed by a rename with the two arguments swapped. As discussed in [15], this means that AutoMed is a **both-as-view (BAV)** data integration system: the add/extend steps in a transformation pathway correspond to Global-As-View (GAV) rules while the delete and contract steps correspond to Local-As-View (LAV) rules. If a GAV view is derived from solely add steps it will be *exact* in the terminology of [11]. If, in addition, it is derived from one or more extend steps using their lower-bound (upper-bound) queries, then the GAV view will be *sound* (*complete*) in the terminology of [11]. Similarly for LAV views. An in-depth comparison of BAV with the GAV and LAV approaches to data integration can be found in [15], while [16, 17] discusses the use of BAV in a peer-to-peer data integration setting. [10] discusses how Global-Local-As-View (GLAV) rules [5, 14] can also be derived from BAV pathways. We note that AutoMed and BAV transform both schema and data together, and thus do not suffer from any data/schema divide.

4 Bioinformatics Service Reconciliation

In this section, we present the problems encountered during service reconciliation and describe our proposed approach for overcoming them, including a brief discussion of how our approach could be incorporated within a workflow tool. We then provide details of XML DataSource Schema (XMLDSS), the XML schema type used in our approach, and of our own earlier work on schema transformation using ontologies that has been extended to enable service reconciliation.

4.1 Proposed Approach

Consider a service S_1 that produces data that need to be consumed by another service S_2. In general, the following issues need to be resolved when trying to handle data exchange between S_1 and S_2:

1. **Data model heterogeneity:** different data models (e.g. legacy flat files and XML) or different schema types (e.g. DTD and XML Schema) may be used. It may also be the case that a service producing or consuming XML data does not have an accompanying XML schema.
2. **Semantic heterogeneity:** schematic differences caused by the use of different terminology, or describing the same information at different levels of granularity.

3. **Schematic heterogeneity:** schematic differences caused by modelling the same information in different ways. This heterogeneity is common to all data modelling languages, but is amplified in XML due to its hierarchical nature, as well as the possibility of using elements with a single text node and attributes interchangeably.
4. **Primitive data type heterogeneity:** differences caused by the use of different primitive data types, e.g. `int` and `varchar`, for the same concept.

To resolve these issues, we propose the following 4-step approach, illustrated in Figure 1:

Step 1: XML as the common representation format. We handle differences in the representation format by using XML as the common representation format. If the output/input of a service is not in XML, then a format converter is needed to convert to/from XML.
Step 2: XMLDSS as the schema type. We use our own XMLDSS schema type for the XML documents input to and output by services. An XMLDSS schema can be automatically extracted from an XML document or automatically derived from an accompanying DTD/XML Schema, if one is available.
Step 3: Correspondences to typed ontologies. We use one or more ontologies as a 'semantic bridge' between services. Providers or users of services semantically annotate the inputs and outputs of services by defining correspondences between an XMLDSS schema and an ontology. Ontologies in our approach are typed, i.e. each concept is associated with a data type, and so defining correspondences resolves issues 2 and 4 discussed above.
Step 4: Schema and data transformation. We use the AutoMed toolkit to automatically transform the XMLDSS schema of the output of service S_1 to the XMLDSS schema of the input of service S_2. This is achieved using the two automatic algorithms discussed in Section 4.4.

If service S_1 does not have an accompanying DTD or XML Schema for its output, sample XML output documents for S_1 must be provided, and these must represent all valid formats that S_1 is able to produce, so as to create an XMLDSS schema that represents all possible instances of the output of S_1. If this is not possible, then an XMLDSS can be extracted at run-time for every new instance XML document output by S_1. The same applies for the input of S_2.

4.2 Integration of Approach With Workflow Tools

Our architecture for service reconciliation supports two different approaches identified below, depending on the preferred form of interoperability between AutoMed and the workflow tool.

Mediation service. With this approach, the workflow tool invokes service S_1, receives its output, and submits this output and a handle on service S_2 to a service provided by the AutoMed system. This uses our approach to transform the output of S_1 to a suitable input for consumption by S_2.

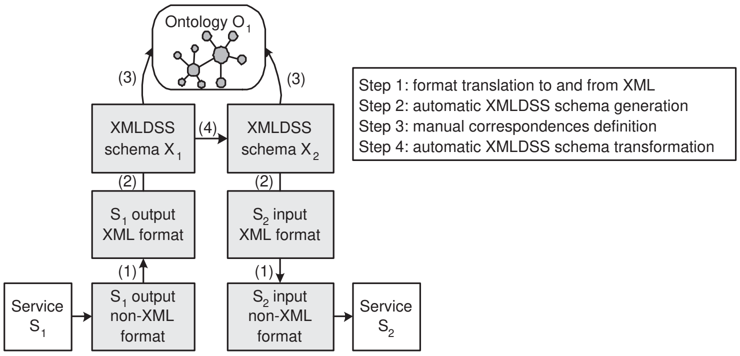

Fig. 1. Reconciliation of services S_1 and S_2 using ontology O_1.

Shim generation. With this approach, the AutoMed system is used to generate shims, i.e. tools or services for the reconciliation of services, by generating transformation scripts which are then incorporated within the workflow tool.

In the following, we provide an overview of the shim generation architecture. The mediation service architecture is described in more detail in Section 5.

With the shim generation approach, AutoMed is not part of the architecture, and so it is necessary to export AutoMed's mediation functionality described and exemplified in Section 5. This functionality consists of the format converters, the algorithms for generating an XMLDSS schema from an XML document, DTD or XML Schema, and the XMLDSS schema transformation algorithms.

Format converters are not a part of the AutoMed toolkit and so can be used from within a workflow tool, without exporting any AutoMed functionality. The converters can be either incorporated within the workflow tool, or their functionality can be imported using services. As an example, a number of shims in ^{my}Grid are format converters.

The XMLDSS schema type is currently used only within the AutoMed system, but it does not require AutoMed functionality. As a result, the XMLDSS schema generation algorithms can be used from within a workflow tool in the same way as format converters.

The two XMLDSS schema transformation algorithms described in Section 4.4 are currently tightly coupled with the AutoMed system, since they use the BAV approach, which is currently supported only by AutoMed. To use our approach without dynamically integrating AutoMed with a workflow tool, we need to export the functionality of the schema transformation algorithms, in order for this AutoMed-dependent functionality to be used statically by a workflow tool. To this effect, we have designed an XQuery query generation algorithm, as detailed in [25], that derives a single XQuery query Q, able to materialise an XMLDSS schema X_2 using data from the data source of an XMLDSS schema X_1, and a transformation pathway $X_1 \rightarrow X_2$. In summary, to derive query Q, the algorithm first uses AutoMed's Query Processor to create the IQL view definition V of each construct c of X_2 in terms of constructs of X_1, and to translate each V into an equivalent XQuery query, V_{XQuery}. The algorithm then creates a single XQuery query Q, for materialising X_2 by following a bottom-up approach as follows. The algorithm first creates the XQuery queries for materialising the leaf elements of X_2, together with their attributes and child text nodes. These queries are then used to create the queries that materialise the parent elements of the leaf elements, together with their attributes and text nodes. This process is repeated until the root of X_2 is reached and the overall query Q is formulated.

4.3 XML DataSource Schema (XMLDSS)

The standard schema definition languages for XML are DTD and XML Schema. However, both of these provide grammars to which conforming documents adhere, and they do not explicitly summarise the tree structure of the data sources. In our schema transformation setting, tree-structured schemas are preferable as they facilitate schema traversal, structural comparison between a source and a target schema, and restructuring of the source schema. Moreover, such a schema type means that the queries supplied with AutoMed primitive transformations are essentially path queries, which are easily generated.

The AutoMed toolkit therefore supports a modelling language called *XML DataSource Schema* (XMLDSS), which summarises the tree structure of XML documents, much like DataGuides [6]. XMLDSS schemas consist of four kinds of constructs: Element, Attribute, Text and NestList (see [23] for details of their specification in terms of the HDM). The last of these defines parent-child relationships either between two elements e_p and e_c or between an element e_p and the Text node. These are respectively identified by schemes of the form $\langle\langle \mathsf{i}, \mathsf{e_p}, \mathsf{e_c} \rangle\rangle$ and $\langle\langle \mathsf{i}, \mathsf{e_p}, \mathsf{Text} \rangle\rangle$, where i is the position of e_c or Text within the list of children of e_p in the XMLDSS schema.

In an XMLDSS schema there may be elements with the same name occurring at different positions in the tree. To avoid ambiguity, the identifier elementName$count is used for each element, where count is incremented every time the same elementName is encountered in a depth-first traversal of the schema.

4.4 XML Schema and Data Transformation using Ontologies

We now describe the two algorithms, the schema conformance algorithm (SCA) and the schema restructuring algorithm (SRA), used in our approach to transform a source XMLDSS schema X_1 and its data to the structure of a target XMLDSS schema X_2. In this setting, these are the XMLDSS schemas of the outputs and inputs of services. Our own previous work in [23, 26, 27] addressed the issue of XML schema and data transformation. This section describes an extended version of the approach of [27], in that the expressiveness of the correspondences used in our approach has been enriched, and the SCA algorithm has been extended to support this.

The SCA uses manually defined correspondences between XMLDSS schemas X_1 and X_2 and an ontology O, in order to automatically transform X_1 and X_2 into equivalent schemas X'_1 and X'_2 that use the same terms as O. As a result, transformation pathways $X_1 \to X'_1$ and $X_2 \to X'_2$ are created. By the bidirectionality of BAV, a pathway $X'_2 \to X_2$ can be automatically derived from the pathway $X_2 \to X'_2$.

In [27], a *correspondence* defines an Element, Attribute or NestList of an XMLDSS schema by means of an IQL query over a typed ontology.[3] In particular, an Element e may map either to a Class c; or to a path ending with a class-valued property of the form $\langle\langle \mathsf{p}, \mathsf{c1}, \mathsf{c2} \rangle\rangle$, where p is the property name and c_1 and c_2 are source and target classes; or to a path ending with a literal-valued property $\langle\langle \mathsf{p}, \mathsf{c}, \mathsf{Literal} \rangle\rangle$, where p is the property name and c the source class; additionally, the correspondence may state that the instances of a class are constrained by membership in some subclass. An Attribute may map either to a literal-valued property or to a path ending with a literal-valued property.

We now extend the correspondences of [27] as follows. An XMLDSS scheme of the form $\langle\langle \mathsf{i}, \mathsf{e}, \mathsf{Text} \rangle\rangle$ (where i denotes the order of $\langle\langle \mathsf{Text} \rangle\rangle$ in the list of children of Element $\langle\langle \mathsf{e} \rangle\rangle$) may map to a literal-valued property of the form $\langle\langle \mathsf{p}, \mathsf{c}, \mathsf{Literal} \rangle\rangle$. In addition to 1-1 correspondences, we now also allow 1-n correspondences as follows. An Element/Attribute may map to more than one path over the ontology. In this case, n correspondences are required, each associating the same XMLDSS Element/Attribute to a different path over the ontology, and specifying an expression that determines the part of the extent of the Element/Attribute to which the correspondence applies (an example of this is given in Section 5). This expression is in general a select-project IQL query. We note that these extended correspondences are GLAV, in contrast with the LAV correspondences defined in our own earlier work [27], as an expression over an XMLDSS construct (rather than just an XMLDSS construct) maps to a path in the ontology.[4]

The SCA uses correspondences from an Element or Attribute to a single path over the ontology to rename that construct, ensuring consistency with the terminology of the ontology. In the case of a 1-n correspondence relating to an Element e with parent p, the algorithm first retrieves all relevant correspondences, then inserts n Elements under p (in the position previously held by e), named after the paths specified by the correspondences, and finally deletes e and its underlying structure. When inserting the n Elements under p, the algorithm also replicates the underlying structure of the old Element e under each one of the newly inserted Elements. A 1-n correspondence relating to an Attribute is handled similarly: the owner Element is replaced by n Elements with the same name, each containing a different Attribute named after the paths specified by the correspondences. A

[3] In principle, it would be possible to use more high-level query languages such as XQuery to specify correspondences in our setting. Currently, AutoMed provides an XQuery-to-IQL translator component, capable of translating (possibly nested) FLWR XQuery queries to (possibly nested) select-project-join IQL queries.

[4] Even though BAV pathways could have been used to express these GLAV mappings, we specify the mappings directly as GLAV rules for compactness.

correspondence mapping an Attribute or a scheme of the form $\langle\langle \mathsf{i}, \mathsf{e}, \mathsf{Text} \rangle\rangle$ in the XMLDSS to a literal-valued property in the ontology is used to perform primitive data type reconciliation: if the data type of the Attribute or scheme in the XMLDSS schema is not the same as in the ontology, the algorithm replaces the Attribute or scheme by performing a type-casting operation.

After the transformation of schemas X_1 and X_2 into schemas X_1' and X_2' that use the same terms as O, our second algorithm, the SRA presented in [27], automatically transforms X_1' to the structure of X_2', producing a transformation pathway $X_1' \rightarrow X_2'$. To do so, the SRA first inserts into X_1' those constructs present in X_2' but not in X_1'. After this *growing phase*, a *shrinking phase* follows, in which the SRA removes from X_1' those constructs present in X_1' but not in X_2'. The SRA is able to generate synthetic structure to avoid loss of data caused by structural incompatibilities between X_1' and X_2'. The SRA is also able to use information that identifies an element/attribute in X_1' to be either equivalent to, or a superclass of, or a subclass of an element/attribute in X_2'. This information may be produced by, e.g. a schema matching tool or, in our context here, via correspondences to an ontology.

Consequently, an overall transformation pathway from X_1 to X_2 can now be obtained by composing the pathways $X_1 \rightarrow X_1'$, $X_1' \rightarrow X_2'$ and $X_2' \rightarrow X_2$. This pathway can be used to automatically transform data that is structured according to X_1 to be structured according to X_2, and an XML document structured according to X_2 can finally be materialised (the pathway $X_1 \rightarrow X_2$ could also be used to translate queries expressed on X_2 to operate on X_1).

Note that we do not assume the existence of a single ontology. As discussed in [27], it is possible for XMLDSS schema X_1 to have a set of correspondences C_1 to an ontology O_1, and for XMLDSS schema X_2 to have a set of correspondences C_2 to another ontology O_2. Provided there is an AutoMed transformation pathway between O_1 and O_2, either directly or through one or more intermediate ontologies, we can use C_1 and the transformation pathway between O_1 and O_2 to automatically produce a new set of correspondences C_1' between X_1 and O_2. As a result, this setting is now identical to a setting with a single ontology. There is a proviso here that the new set of correspondences C_1' must conform syntactically to the correspondences accepted as input by the schema conformance process. Determining necessary conditions for this to hold is an area of future work.

5 Case Study

We now describe our approach in more detail and demonstrate the use of AutoMed as a mediation service by specifying a sample bioinformatics workflow. Note that listings of all service inputs, outputs and XMLDSS, XML Schema and DTD schemas discussed in this section are given in [25].

Figure 2 illustrates a sample workflow with three services that will be used to demonstrate our approach. The first service takes as input an IPI (`http://www.ebi.ac.uk/IPI`) accession number, e.g. `IPI00015171`, and outputs the corresponding IPI entry as a flat file using the UniProt (`http://www.ebi.uniprot.`

Fig. 2. Sample Workflow.

`org`) format. The second service receives an InterPro (`http://www.ebi.ac.uk/interpro`) accession number and returns the corresponding InterPro entry. The third service receives a Pfam (`http://www.sanger.ac.uk/Software/Pfam`) accession number and returns the corresponding Pfam entry. In this workflow, two transformations are needed: T_1 extracts the InterPro accession number from an IPI entry using the UniProt format, while T_2 extracts the Pfam accession number from an InterPro entry.

We now apply the mediation service approach described in Section 4.2, for the reconciliation of the services of the workflow of Figure 2.

Step 1: XML as a common representation format. Service *getIPIEntry* outputs a flat file that follows the UniProt representation format and contains a single entry consisting of multiple lines. Each line consists of two parts, the first being a two-character line code, indicating the type of data contained in the line, while the second contains the actual data, consisting of multiple fields.

Since UniProt also has an XML representation format specified by an XML Schema, we created a format converter that, given an IPI flat file f that follows the UniProt format, converts f to an XML file conforming to that XML Schema.

Service *getInterProEntry* outputs an XML file and so there is no need for a format converter. Concerning the input of the second and the third service, they each take as input a single string, representing an InterPro/Pfam accession number, respectively. The input XML documents for these contain a single XML element, `ip_acc` and `pf_acc`, respectively, with a PCData node as a single child, as shown below. For these, the format converters implement the functionality of the XPath expressions `/ip_acc/text()` and `/pf_acc/text()`, respectively.

```
<ip_acc>InterPro_accession_string</ip_acc>
<pf_acc>Pfam_accession_string</pf_acc>
```

Step 2: XMLDSS schema generation. After resolving representation format issues, we now give details on the generation of XMLDSS schemas for our setting. As discussed above, service *getIPIEntry* outputs a flat file which is converted to an XML file that conforms to the UniProt XML Schema. An XMLDSS schema for the output of this service is automatically derived from that XML Schema. Similarly, an XMLDSS schema for the output of service *getInterProEntry* is automatically derived using the InterPro DTD schema.

Concerning the input of the second and the third service, the corresponding XMLDSS schemas are automatically extracted by using a single sample XML document for each, such as the ones given earlier.

Step 3: Correspondences. After generating the XMLDSS schemas for our workflow, we need to specify the correspondences between these schemas and an ontology. In this case, we have used the typed ^{my}Grid OWL domain ontology.

In general, all XMLDSS elements and attributes should be mapped to the ontology. However, if an element or attribute cannot be mapped to the ontology, this construct is not affected by our SCA and SRA algorithms that use the correspondences to transform X_1 to the structure of X_2. An advantage of this is that data transformation is still possible with only a partial set of correspondences from an XMLDSS schema to the ontology. This property is particularly significant in terms of the applicability and scalability of our approach, as it allows for incrementally defining the full set of correspondences between an XMLDSS schema and an ontology: one can define only those correspondences relevant to the specific problem at hand, instead of the full set of correspondences.

In our example, this means that we only need to specify correspondences for those constructs of the XMLDSS schema of the output of $getIPIEntry$ that contribute to the input of service *getInterProEntry*. Consequently, we need to specify correspondences for only two constructs, $\langle\langle\mathsf{dbReference\$9}\rangle\rangle$ and $\langle\langle\mathsf{dbReference\$9, id}\rangle\rangle$ (see Table 1). The first models an entry in a bioinformatics data resource, whose type is specified by $\langle\langle\mathsf{dbReference\$9, type}\rangle\rangle$. The type of a resource is modelled in IPI using data values, whereas in the ontology it is modelled as classes, and so n correspondences are required for this construct, where n is the number of types of resources that IPI supports and that also exist in the ontology. Each of these correspondences maps $\langle\langle\mathsf{dbReference\$9}\rangle\rangle$ to a class in the ontology representing a bioinformatics data resource record and specifies the part of the extent of $\langle\langle\mathsf{dbReference\$9}\rangle\rangle$ to which the correspondence applies. For example, the second correspondence states that those instances of $\langle\langle\mathsf{dbReference\$9}\rangle\rangle$ whose $\langle\langle\mathsf{dbReference\$9, type}\rangle\rangle$ Attribute has a data value of `'Pfam'`, map to the $\langle\langle\mathsf{Pfam\_record}\rangle\rangle$ ontology class. Due to space limitations, but without loss of generality, we only provide the two correspondences related to InterPro and Pfam.

The XMLDSS schema of the input of service *getInterProEntry* consists of a single Element construct, $\langle\langle\mathsf{ip\_acc}\rangle\rangle$, which corresponds to class $\langle\langle\mathsf{InterPro\_accession}\rangle\rangle$ in the ontology, and of a NestList construct, $\langle\langle\mathsf{1, ip\_acc, Text}\rangle\rangle$. The correspondences are given in Table 2. The correspondences for the XMLDSS schema of the input of the third service, $getPfamEntry$, are not listed as they are similar.

Step 4: Schema transformation. After manually specifying correspondences, the SCA and SRA algorithms can automatically transform the outputs of services $getIPIEntry$ and *getInterProEntry* to the required inputs for services *getInterProEntry* and $getPfamEntry$ respectively.

Concerning the output of service $getIPIEntry$, the schema conformance algorithm (SCA) first retrieves all correspondences related to $\langle\langle\mathsf{dbReference\$9}\rangle\rangle$ (in this case 2 correspondences) and inserts $\langle\langle\mathsf{InterPro\_record\$1}\rangle\rangle$ and $\langle\langle\mathsf{Pfam\_record\$1}\rangle\rangle$, using the correspondences' expressions to select the appropriate $\langle\langle\mathsf{dbReference\$9}\rangle\rangle$ instances, i.e. those that have a type Attribute with value `'InterPro'` and `'Pfam'` respectively. As discussed in Section 4.4, the SCA then replicates under the newly

Table 1. Correspondences between the XMLDSS schema of the output of *getIPIEntry* and the ^{my}Grid ontology.

Construct:	$\langle\langle$dbReference\$9$\rangle\rangle$
Extent:	$[\{d\}\vert\{d,t\} \leftarrow \langle\langle \mathsf{dbReference\$9, type}\rangle\rangle; t =' InterPro']$
Path:	$\langle\langle$InterPro_record$\rangle\rangle$
Construct:	$\langle\langle$dbReference\$9$\rangle\rangle$
Extent:	$[\{d\}\vert\{d,t\} \leftarrow \langle\langle \mathsf{dbReference\$9, type}\rangle\rangle; t =' Pfam']$
Path:	$\langle\langle$Pfam_record$\rangle\rangle$
Construct:	$\langle\langle$dbReference\$9, id$\rangle\rangle$
Extent:	$[\{d,i\}\vert\{d,i\} \leftarrow \langle\langle \mathsf{dbReference\$9, id}\rangle\rangle;$ $\{d,t\} \leftarrow \langle\langle \mathsf{dbReference\$9, type}\rangle\rangle; t =' InterPro']$
Path:	$[\{ir,l\}\vert\{ia,ir\} \leftarrow \langle\langle \mathsf{part\_of, InterPro\_accession, InterPro\_record}\rangle\rangle;$ $\{ia,l\} \leftarrow \langle\langle \mathsf{datatype, InterPro\_accession, Literal}\rangle\rangle]$
Construct:	$\langle\langle$dbReference\$9, id$\rangle\rangle$
Extent:	$[\{d,i\}\vert\{d,i\} \leftarrow \langle\langle \mathsf{dbReference\$9, id}\rangle\rangle;$ $\{d,t\} \leftarrow \langle\langle \mathsf{dbReference\$9, type}\rangle\rangle; t =' Pfam']$
Path:	$[\{pr,l\}\vert\{pa,pr\} \leftarrow \langle\langle \mathsf{part\_of, Pfam\_accession, Pfam\_record}\rangle\rangle;$ $\{pa,l\} \leftarrow \langle\langle \mathsf{datatype, Pfam\_accession, Literal}\rangle\rangle]$

Table 2. Correspondences between the XMLDSS schema of the input of *getInterPro* and the ^{my}Grid ontology.

Construct:	$\langle\langle$ip_acc\$1$\rangle\rangle$
Extent:	$\langle\langle$ip_acc\$1$\rangle\rangle$
Path:	$[\{ia\}\vert\{ia,ir\} \leftarrow \langle\langle \mathsf{part\_of, InterPro\_accession, InterPro\_record}\rangle\rangle]$
Construct:	$\langle\langle$1, ip_acc\$1, Text$\rangle\rangle$
Extent:	$\langle\langle$1, ip_acc\$1, Text$\rangle\rangle$
Path:	$[\{ia,l\}\vert\{ia,ir\} \leftarrow \langle\langle \mathsf{part\_of, InterPro\_accession, InterPro\_record}\rangle\rangle;$ $\{ia,l\} \leftarrow \langle\langle \mathsf{datatype, InterPro\_accession, Literal}\rangle\rangle]$

inserted Elements the structure located under ⟨⟨dbReference$9⟩⟩ (again using the correspondences' expressions to select the appropriate structure), and then removes ⟨⟨dbReference$9⟩⟩. Note that this removal is postponed until after any other insertions are performed, as other insertions may need to use the extent of ⟨⟨dbReference$9⟩⟩ in the queries supplied with the AutoMed transformations.

The SCA then retrieves all correspondences related to ⟨⟨dbReference$9, id⟩⟩ (in this case 2 correspondences) and inserts Attributes ⟨⟨InterPro_record$1,InterPro_record.part_of.InterPro_accession⟩⟩ and ⟨⟨Pfam_record$1,InterPro_record.part_of.Pfam_accession⟩⟩, using the correspondences' expressions to select the appropriate ⟨⟨dbReference$9, id⟩⟩ instances (as discussed earlier, ⟨⟨dbReference$9⟩⟩ has not yet been removed). Concerning primitive data types, ⟨⟨dbReference$9, id⟩⟩ is of type `string`, and the same applies for all accession numbers in the ^{my}Grid domain ontology, so there is no need for any type-casting operations.

Concerning the input of *getInterProEntry*, the SCA uses the first correspondence to rename ⟨⟨ip_acc$1⟩⟩ to ⟨⟨InterPro_record.part_of.InterPro_accession$1⟩⟩, while the second correspondence, which is a primitive data type reconciliation correspondence, is of no consequence as both the input of the service and the ontology model InterPro accession numbers using the `string` data type.

After the application of the SCA, the XMLDSS schema X_2 of the input of service *getInterProEntry* contains three constructs, ⟪InterPro_record.part_of.InterPro_accession$1⟫, ⟪Text⟫ and a NestList linking these two constructs. The XMLDSS schema of the output of service *getIPIEntry*, X_1, contains a number of constructs, but the only ones relevant to those of X_2 are ⟪InterPro_record$1⟫ and ⟪InterPro_record$1,InterPro_record.part_of. InterPro_accession⟫. The schema restructuring algorithm (SRA) therefore applies a number of contract transformations supplied with the queries Void and Any, so as to remove non-relevant constructs. The only non-trivial transformation is the attribute-to-element transformation: first Element ⟪InterPro_record.part_of.InterPro_accession$1⟫ is added to X_1 using the extent of Attribute ⟪InterPro_record$1,InterPro_record.part_of.InterPro_accession⟫, then NestList ⟪InterPro_record.part_of.InterPro_accession$1, Text⟫ is added, again using the Attribute extent, and finally the Attribute is deleted.

After applying the SRA, we finally employ the XMLDSS schema materialisation algorithm defined in [26] to materialise X_2, i.e. the input of service *getInterProEntry*, using data from the data source of X_1, i.e. the output of service *getIPIEntry*, using the transformation pathway $X_1 \rightarrow X_1' \rightarrow X_2' \rightarrow X_2$.

The application of Step 4 for the second part of our workflow is similar.

6 Conclusions and Future Work

In this paper we have presented a generic and scalable architecture for bioinformatics service reconciliation within a wider data transformation framework. Our approach makes no assumptions about representation format, primitive data type usage or the number of ontologies used. Moreover, this approach can be used either dynamically or statically from within a workflow tool.

The architecture exploits format converters to establish a common XML format for all service inputs and outputs, thus reducing the overall complexity of service reconciliation by establishing a common representation format. Service inputs and outputs are then abstracted using the XMLDSS schema type which can be automatically generated either from XML documents, or from accompanying DTD or XML Schema specifications using our algorithms.

Our approach is able to use correspondences to multiple ontologies for defining the semantics of services. This 'semantic bridge' is utilised by two automatic algorithms that use the correspondences to allow data transformation between services. The schema conformance algorithm is able to use 1-1 and 1-n GLAV correspondences to ontologies, in order to produce schemas with no semantic heterogeneity. The schema restructuring algorithm then restructures the source schema to the target schema. This algorithm is able to avoid loss of information that may be caused due to structural incompatibilities of the data sources.

While the correspondences to ontologies must be produced manually or semi-automatically, an advantage of our approach is that correspondence reusability is promoted by allowing the use of multiple ontologies. Moreover, our approach does not require a full set of correspondences to be defined, but instead allows the definition of only those correspondences between the XMLDSS schema and the ontology that are relevant to the problem at hand - we therefore allow an incremental approach for the definition of correspondences.

The architecture has been illustrated with a bioinformatics workflow characteristic of those currently available with string-based inputs. Future workflows with more complex inputs are to be expected, which our architecture will also readily support.

Concerning the integration of our approach with workflow tools, we defined two possible architectures. The first, using AutoMed as a mediation service, can be used from within a workflow tool by invoking AutoMed as a service and does not require XMLDSS or XQuery support. On the other hand, in the shim generation architecture AutoMed is used to statically generate shims, which can then be incorporated into any workflow tool that supports XQuery.

Our current implementation has supported testing of the transformation pathways underpinning the service reconciliation examples presented within the AutoMed toolkit. Ongoing work is aimed at integrating our approach with the Taverna workflow tool. The resulting implementation will be evaluated within the proteomics grid infrastructure being developed in the ISPIDER project [24].

In future work, we will investigate the necessary conditions under which a set of correspondences, transformed by a 'semantic bridge' defined between multiple ontologies, adheres to the required format of our schema conformance algorithm. Other extensions to our work include investigating the effect on our approach of constraints on XMLDSS schemas and/or the ontologies, and also considering the effect of the evolution of the inputs and outputs of services.

Acknowledgements. The work presented in this paper is part of the BBSRC-funded ISPIDER project. The authors would also like to thank the ISPIDER members and especially Khalid Belhajjame, Suzanne Embury and Norman Paton for the fruitful discussions that helped shape the work presented in this paper.

References

1. B. Amann, C. Beeri, I. Fundulaki, et al. Ontology-based integration of XML web resources. In *Proc. of Int. Semantic Web Conference*, pages 117–131, 2002.
2. B. Benatallah et al. Declarative composition and peer-to-peer provisioning of dynamic web services. In *Proc. of ICDE'02*, pages 297–308, 2002.
3. S. Bowers and B. Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *Proc. of Data Integration in the Life Sciences (DILS'04)*, pages 1–16, 2004.
4. I. F. Cruz, H. Xiao, and F. Hsu. An ontology-based framework for XML semantic integration. In *Proc. IDEAS'04*, pages 217–226, 2004.
5. M. Friedman, A. Levy, and T. Millstein. Navigational plans for data integration. In *National Conference on Artificial Intelligence*, pages 67–73. AAAI Press, 1999.
6. R. Goldman and J. Widom. DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases. In *Proc. VLDB'97*, pages 436–445, 1997.

7. D. Hull et al. Treating shimantic web syndrome with ontologies. In *Proc. of Advanced Knowledge Technologies workshop on Semantic Web Services*, 2004.
8. D. Hull, R. Stevens, and P. Lord. Describing web services for user-oriented retrieval. In *Proc. of W3C Workshop on Frameworks for Semantics in Web Services*, 2005.
9. E. Jasper, A. Poulovassilis, and L. Zamboulis. Processing IQL queries and migrating data in the AutoMed toolkit. AutoMed Technical Report 20, July 2003.
10. E. Jasper, N. Tong, P.J. McBrien, and A. Poulovassilis. View generation and optimisation in the AutoMed data integration framework. In *Proc. of 6th Baltic Conference on Databases and Information Systems*, 2004.
11. M. Lenzerini. Data integration: A theoretical perspective. In *Proc. PODS'02*, pages 233–246, 2002.
12. P. Lord, P. Alper, C. Wroe, and C. Goble. Feta: A light-weight architecture for user oriented semantic service discovery. In *Proc. of European Semantic Web Conference (ESWC'05)*, pages 17–31, 2005.
13. P. Lord, S. Bechhofer, M. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying Semantic Web services to bioinformatics: experiences gained, lessons learnt. In *Proc. of Int. Semantic Web Conference*, pages 350–364, 2004.
14. J. Madhavan and A.Y. Halevy. Composing mappings among data sources. In *Proc. of VLDB'03*, pages 572–583, 2003.
15. P. McBrien and A. Poulovassilis. Data integration by bi-directional schema transformation rules. In *Proc. ICDE'03*, pages 227–238, March 2003.
16. P. McBrien and A. Poulovassilis. Defining peer-to-peer data integration using both as view rules. In *Proc. Workshop on Databases, Information Systems and Peer-to-Peer Computing (at VLDB'03), Berlin*, 2003.
17. P.J. McBrien and A. Poulovassilis. P2P query reformulation over Both-as-View data transformation rules. In *Proc. of Databases, Information Systems and Peer-to-Peer Computing (at VLDB'06)*, page TBC. Springer, 2006.
18. B. Medjahed, A. Bouguettaya, and A.K. Elmagarmid. Composing web services on the Semantic Web. *VLDB Journal*, 12(4):333–351, 2003.
19. T. Oinn, M. Addis, J. Ferris, et al. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
20. B. Srivastava and J. Koehler. Web Service composition - current solutions and open problems. In *Proc. of Workshop on Planning for Web Services (ICAPS'03)*, pages 28–35, 2003.
21. L. Stein. Creating a bioinformatics nation. *Nature*, 417:119–120, May 2002.
22. S. Thakkar, J.L. Ambite, and C. A. Knoblock. Composing, optimizing, and executing plans for bioinformatics web services. *VLDB Journal*, 14(3):330–353, 2005.
23. L. Zamboulis. XML data integration by graph restructuring. In *Proc. British National Conference on Databases (BNCOD'04)*, pages 57–71, 2004.
24. L. Zamboulis, H. Fan, K. Belhajjame, J. Siepen, A. Jones, N.J. Martin, A. Poulovassilis, S.J. Hubbard, S. Embury, and N.W. Paton. Data access and integration in the ISPIDER proteomics grid. In *Proc. Data Integration for the Life Sciences (DILS'06), LNCS 4075*, pages 3–18, 2006.
25. L. Zamboulis, N. Martin, and A. Poulovassilis. Bioinformatics service reconciliation by heterogeneous schema transformation. Birkbeck TR BBKCS-07-03, March 2007.
26. L. Zamboulis and A. Poulovassilis. Using AutoMed for XML Data Transformation and Integration. In *Proc. International Workshop on Data Integration over the Web (at CAiSE'04)*, pages 58–69, 2004.
27. L. Zamboulis and A. Poulovassilis. Information sharing for the Semantic Web - a schema transformation approach. In *Proc. International Workshop Data Integration and the Semantic Web (at CAiSE'06)*, pages 275–289, 2006.