On Characterising and Identifying Mismatches in Scientific Workflows

Khalid Belhajjame, Suzanne M. Embury, and Norman W. Paton

School of Computer Science
University of Manchester
Oxford Road, Manchester, UK
{khalidb, embury, paton}@cs.man.ac.uk

Abstract. Workflows are gaining importance as a means for modelling and enacting *in silico* scientific experiments. A major issue which arises when aggregating a collection of analysis operations within a workflow is the compatibility of their inputs and outputs: the analysis operations are supplied by independently developed web services which are likely to have incompatible inputs and outputs. We use the term mismatch to refer to such incompatibility. This paper characterises the mismatches a scientific workflow may suffer from and specifies mappings for their resolution.

1 Introduction

Scientific workflows are gaining considerable momentum as a mechanism for specifying and automating the execution of scientific experiments [1,10]. During the design of a scientific workflow, the designer's focus is on selecting and composing the analysis operations that will carry out the work of the experiment. Analysis operations are supplied by third parties and as such it is often the case that their inputs and outputs are incompatible with those of the other operations to which they must be connected. We use the term mismatch to refer to such incompatibility. In order to resolve a mismatch, the designer must expend some effort in discovering or implementing special operations that can be plugged into the workflow at the point of incompatibility, and can transform the data sets as necessary to resolve it.

Manual detection and correction of such mismatches is time-consuming and unreliable, and thus reduces the claimed benefits of scientific workflows in facilitating the rapid specification of experiments. In this paper, we propose a classification of the kinds of mismatches that can occur in data-driven workflows and derive the additional information that must be captured about workflow operations if potential mismatches are to be identified automatically. This additional information takes the form of annotations on web service inputs and outputs, based on three separate ontologies.

The remainder of the paper is organised as follows. First, in Section 2, we formally define scientific workflows. In Section 3, we describe the three additional ontologies used for annotating operation inputs and outputs, and use them (in Section 4) to present the mismatch classification and (in Section 5) to specify further annotations for transformation functions that characterise the kinds of mismatches they can address. Finally we close the paper by comparing our work against existing works, and drawing conclusions in Section 6.

U. Leser, F. Naumann, and B. Eckman (Eds.): DILS 2006, LNBI 4075, pp. 240–247, 2006.

2 Scientific Workflows

A scientific workflow is a set of operations connected together using data links. For the purposes of this paper, we define a scientific workflow *SWf* as *SWf* = ⟨*nameWf, OP, DL*⟩, where *nameWf* is a unique identifier for the workflow, *OP* is the set of operations from which the workflow is composed, and *DL* is the set of data links connecting the operations in *OP*.

Operation. An operation $op \in OP$ is defined as $op = \langle nameOp, loc, in, out \rangle$, where *nameOP* is the unique identifier for the operation, *loc* is the URL of the web service that implements the operation, and *in* and *out* are two sets representing the input and output parameters of the operation, respectively.

Parameter. A parameter provides information on the data type of a given operation input/output. It is defined by the pair ⟨*nameP, type*⟩, where *nameP* is the parameter's identifier (unique within the operation) and *type* is the parameter's data type. In our work, we assume an XML type system, so that parameter data types may be either simple types, such as *xs:string* and *xs:int*, or complex types, built from simple ones.

Data Links. Let $IN = \cup_{(op \in OP)} op.in$ be the set of inputs of all the operations comprising a scientific workflow, and $OUT = \cup_{(op \in OP)} op.out$ be the set of outputs of all its operations. The set of data links connecting the workflow operations must then satisfy the following: $DL \subseteq (OP \times OUT) \times (OP \times IN)$. A data link relating the output o of the operation *op1* to the input i of the operation *op2* is therefore denoted by the quadruple *(op1,o,op2,i)*.

3 Ontologies for Characterising Mismatches

Information on the types of operation parameters is usually easily available to scientific workflow systems. For example, where operations are actually web services, the data types can be extracted from the WSDL specification of the service. However, as we have seen, not all mismatches are visible in the types of the connected parameters. In order to automatically detect mismatches, the implicit information about the form and role of operation parameters must be made explicit, just as the information about the type of the parameter currently is. This additional information concerns the semantics of the parameter (i.e. the real world entity to which the parameter corresponds), the representation format used for the parameter over and above any data typing given to it and the extent of the parameter (i.e. the set of possible values which the parameter may take). For each of these, we must create an *ontology* of terms that can be used to annotate services with the information required to detect mismatches.

An ontology is commonly defined as an explicit specification of a conceptualisation [4]. Formally, an ontology θ can be defined as a set of concepts, $\theta = \{c1, \ldots, cn\}$. We use the following ontologies to annotate service parameters for mismatch detection.

Domain Ontology, θ_{domain}. This ontology captures information about the application domains covered by the operations, and enables us to describe the real world concepts to which each parameter corresponds. An example of such an ontology is that developed

by the myGrid project describing the domain of molecular biology [9]. Typical concepts from this ontology are *ProteinSequence* and *MolecularWeight*.

For the identification of mismatches, we assume the existence of a function *domain()*, the signature of which is presented below. Given an operation and an input/output parameter, the function *domain()* returns the corresponding concept from the domain ontology.

$$domain: OP \times (IN \cup OUT) \rightarrow \theta_{domain}$$

Representation Ontology, $\theta_{represent}$. As in many other application areas, a variety of different formats have been defined for representing the same kind of biological data (i.e. data corresponding to the same domain concept). For example, a protein sequence can be represented using *Fasta* format or *Uniprot* format or any of several other similar formats. These formats can be represented using complex data types but at present it is much more common for workflow operations to treat them as simple string objects and to ignore their internal structuring. This is partly due to legacy design (since many of the more popular biological web services were implemented before the development of XML and its associated programming tools) and partly because current workflow systems do not always have very rich type systems.

In order to detect mismatches in representation format as well as data type mismatches, it is necessary for services to be annotated according to the formats expected and produced by their inputs and outputs. We therefore require an ontology of terms for describing data formats. Such an ontology has already been designed by the myGrid project [9].

We therefore assume the existence of a function *represent()*, with the signature presented below. Given an operation and an input or output parameter, the function *represent()* returns the corresponding concept from the representation ontology.

$$represent: OP \times (IN \cup OUT) \rightarrow \theta_{represent}$$

In order to compare two representation concepts for mismatch identification, we need an additional binary comparison operator for concepts in $\theta_{represent}$. We use the *contains()* function, the signature of which is presented below, to describe the relationship between these formats. Given two formats *x* and *y*, the function *contains(x,y)* is true if *x* contains all the data content needed for creating an instance of *y*, and false otherwise.

$$contains: (\theta_{represent} \times \theta_{represent}) \rightarrow Boolean$$

Extent Ontology, θ_{extent}. The concepts of this ontology define the scope of possible values of a given operation parameter. Although in general it is not possible to accurately describe the extent of a parameter, it is the case that the relationships between the extents of some web services is known in advance and can be used in detecting mismatches. For example, the TrEMBL database[1] is known to be a superset of SwissProt, whereas the various species specific gene databases are known not to overlap.

No ontologies currently exist for describing the extent of biological data sets, and we have therefore constructed one ourselves. We assume the existence of a function *extent()* for retrieving the extent of input/output parameters, with the signature:

[1] http://www.ebi.ac.uk/trembl/

$$extent: OP \times (IN \cup OUT) \rightarrow \theta_{extent}$$

In order to be able to compare extents, we use the function *coveredBy()*, the signature of which is presented below. Given two concepts from the extent ontology, *e1* and *e2*, *coveredBy(e1,e2)* is true if the space of values designated by *e1* is a subset of the space of values designated by *e2* and false otherwise.

$$coveredBy: (\theta_{extent} \times \theta_{extent}) \rightarrow Boolean$$

4 Characterising Mismatches

Using annotations of the form described in the previous section, we can automatically detect a variety of forms of mismatch that go beyond simple data type mismatches. We now present a classification of these mismatch types and define the criteria for identifying each one.

Type Mismatch. refers to incompatibility in terms of data type between connected parameters. In order to be compatible, the data type of the output parameter must be the same as or a subtype of the data type required by the input parameter. Formally, a data link $(op1,o,op2,i) \in DL$ suffers from a type mismatch iff[2]:

$$o.type \not\preceq i.type$$

Cardinality Mismatch. is a particular kind of type mismatch. For example, assuming a type system in which the only means of forming collection types is an array constructor, we can say that a data link $(op1,o,op2,i) \in DL$ suffers from a cardinality mismatch iff:

$$(o.type = ArrayOf\,(i.type)) \text{ or}$$
$$(i.type = ArrayOf\,(o.type))$$

ArrayOf(t) is a type. An instance of *ArrayOf(t)* is an array whose elements are *t* instances.

Domain Mismatch. refers to incompatibility in terms of semantic domain between connected output and input parameters. In order to be compatible, the domain of the output must be the same as or a sub-concept of the domain of the subsequent input. Specifically, a data link $(op1,o,op2,i) \in DL$ suffers from a domain mismatch iff[3]:

$$domain(op1,o) \not\sqsubseteq domain(op2,i)$$

For example, consider a data link *(op1,o,op2,i)* such that *domain(op1,o)* = *DNA_sequence* and *domain(op2,i)* = *Protein_sequence*. According to the molecular biology ontology mentioned earlier [9], *DNA_sequence* $\not\sqsubseteq$ *Protein_sequence*, therefore, *(op1,o,op2,i)* suffers from a domain mismatch.

[2] The symbol $\not\preceq$ stands for not a sub-type of.
[3] The symbol $\not\sqsubseteq$ stands for not a sub-concept of.

Representation Mismatch. Two operation parameters, which belong to compatible semantic domains, can be represented using different data formats. Representation mismatch refers to the difference in terms of format between connected input and output parameters, which are domain compatible. Specifically, a data link $(op1,o,op2,i) \in DL$ suffers from a representation mismatch iff:

$$(domain(op1,o) \subseteq domain(op2,i)) \text{ and } (represent(op1,o) \neq represent(op2,i))$$

For example, suppose that $domain(op1,o) = domain(op2,i) = Protein\_record$, $represent(op1,o) = Uniprot\_record$, and $represent(op2,i) = Fasta\_record$. The output and the input parameters have the same semantic domain. However, they adopt different representations. We conclude that the data link suffers from a representation mismatch.

Content Mismatch. is a particular kind of representation mismatch, in which the formats conflict in terms of data scope as well as in terms of pure representation—that is, the format of the output carries less data content than is required by the format of the subsequent input. Formally, a data link $(op1,o,op2,i) \in DL$ suffers from a content mismatch iff:

$$(domain(op1,o) \subseteq domain(op2,i)) \text{ and } (represent(op1,o) \neq represent(op2,i)) \text{ and } (contains(represent(op1,o),represent(op2,i)) = false)$$

This situation is distinguished because it represents a particularly serious form of representation mismatch. Even if we can find a web service that can perform the transformation between the two mismatched data formats, there may still be a problem with the workflow, since the transformed output may not contain all the information expected by succeeding operations. Note that we say there "may" be a problem with the workflow. It is possible that the succeeding operation will only access those parts of the transformed data structure that are contained within the initial data format, in which case there will be no problem.

As an example of this, consider the data link $(GetFasta,o,GetSequence,i)$. Here, $represent(GetFasta,o) = Fasta\_record$, and $represent(GetSequence,i) = Uniprot\_record$. This data link suffers from a content mismatch, since a *Fasta_record* does not contains all the elements required for creating a *Uniprot_record* instance: $contains(Fasta\_record, Uniprot\_record) = false$. However, in reality, the *GetSequence* operation will only read the protein sequence parts of the *Uniprot* record supplied as its input, and therefore a simple transformation between formats is sufficient to resolve the mismatch.

Extent Mismatch. refers to incompatibility in terms of the space of possible values between two connected output and input parameters. Specifically, a data link $(op1,o,op2,i)$ suffers from an extent mismatch if it does not suffer from a type mismatch, a domain mismatch or a representation mismatch, but the extent of the input i does not cover the extent of the output o. Formally, a data link $(op1,o,op2,i) \in DL$ suffers from an extent mismatch iff:

$$(o.type \preceq i.type) \text{ and } (represent(op1,o) = represent(op2,i)) \text{ and } (domain(op1,o) \subseteq domain(op2,i)) \text{ and } (coveredBy(extent(op1,o),extent(op2,i)) = false)$$

For example, consider a data link *(op1,o,op2,i)* such that *domain(op1,o)* = *domain(op2,i)* = *ORF*. *ORF* stands for open reading frame. Suppose now that *extent(op1,o)* = *FlyBase* and *extent(op2,i)* = *SGD*. *FlyBase* is a database that stores information on the genetics and molecular biology of Drosophila. *SGD* is a scientific database of the molecular biology and genetics of the yeast *Saccharomyces cerevisiae*. The two databases are non-overlapping: none of the *ORFs* found in *FlyBase* are present in *SGD* (i.e. *coveredBy(FlyBase,SGD)* = *false*). Therefore, *(op1,o,op2,i)* suffers from an extent mismatch. Even though the parameters appear to match exactly in terms of domain, data type and representation format, the workflow will still not be able to produce a result.

5 Annotation of Parameter Mapping Operations

The same ontologies that allow us to create annotations for identifying mismatches can also support the annotation of transformation functions that can resolve them. In our context, we refer to such functions as *mappings*, since they map from one parameter type to another. In order to compare the available mappings with the identified mismatches, we annotate the mappings. Given a data link *(op1,o,op2,i)*, which suffers from a mismatch, a mapping is used for transforming the data produced by o to meet the requirements of the input i. Formally, a mapping is defined as follows:

$$\langle T1,T2,C1_{represent},C2_{represent},C1_{domain},C2_{domain},f_{map}\rangle$$

where *T1* and *T2* are data types, $C1_{represent}$ and $C2_{represent}$ are formats from the representation ontology, and $C1_{domain}$ and $C2_{domain}$ are concepts from the domain ontology. f_{map}*: T1* $\rightarrow$ *T2* is a function. Given an instance *t1* of *T1* that follows the format $C1_{represent}$ and belongs to the domain $C1_{domain}$, f_{map} *(t1)* returns an instance *t2* of *T2*, which follows the format $C2_{represent}$ and belongs to the domain $C2_{domain}$. The extent of the mapping function f_{map} *()* is specified by a pair (e_1,e_2), where $e_1, e_2 \in \theta_{extent}$; e_1 designates the extent of the domain of f_{map}, and e_2 designates the extent of its range.

The above annotation system can be used for locating the appropriate mappings for correcting the identified mismatches. It is possible that none of the existing mappings can be used for correcting a given mismatch. Instead of building a new mapping, there are cases in which the desired mapping can be obtained by composing in sequence two or more existing mappings.

6 Related Work and Concluding Remarks

Several problem solving environments (PSEs) have been proposed to support the design and enactment of scientific workflows [10]. Generally, they do not provide means for identifying and correcting mismatches. Taverna, for example, allows the designer to connect any two operations regardless of whether the connected outputs and inputs are compatible. Some PSEs are able to identify type mismatches. In Triana [8], data links are checked at design time and a warning message is displayed whenever two connected parameters have incompatible data types. In terms of parameter mapping, Kepler [3] is, to our knowledge, the only system which supports the mapping of operation parameters that have type mismatches. Note, however, that parameter mapping

is a work in progress that is not supported by the current distribution of Kepler. The semantic domain of the operation inputs and outputs are described using an ontology. Whenever two connected parameters belong to compatible domains but have incompatible data types, a mapping is generated to transform the output parameter structure into the structure of the succeeding input parameter [2]. Incompatibilities due to differences in representation format, content and extent are not handled by this proposal.

In this paper, we have characterised a range of mismatches that can occur in scientific workflows. Our categorisation goes beyond existing work in this area by identifying the need for additional parameter annotations describing representation formats and extents, and in showing how they interact with the more familiar notions of domain and data type annotations. The categorisation can be used to implement mismatch detection and resolution services that allow workflow designers to concentrate their attention first on the core aspects of workflow semantics, and to consider the necessary data transformations afterwards. For complex workflows, this opens up the possibility for domain experts to rapidly specify abstract workflows, which they then pass to staff with technical expertise in managing data mapping and transformation to resolve any gaps in the workflow.

We have developed a prototype that implements the proposed framework as an extension of the Taverna workbench [7]. Using the prototype, we have conducted a preliminary evaluation of our mismatch categorisation. Real scale trials are not yet a practical possibility, due to the lack of rich service and annotation mappings. The Feta registry we are currently using, for example, contains the descriptions of around 30 service operations, though more extensive annotations are planned. However, we wished to gain some insight into the degree to which the mismatches we have identified occur in practice. To this end, we collected together a sample set of 14 bioinformatics workflows, which were designed in the context of e-science projects such as ISPIDER[4], myGrid[5] and Pegasys[6]. We then examined the operations contained within the workflows and made a judgement as to whether the operation was part of the core semantics of the workflow or whether its role in the workflow was to resolve incompatibilities between core operation parameters. We also attempted to classify the mismatches we found based on the categorisation presented in this paper.

The results of this small study showed that the most commonly occurring types of mismatches are the representation and domain mismatches. The majority of the workflows that we analysed suffered from these kind of incompatibilities, with the next most common kind being cardinality and extent mismatches. In fact, most of the type mismatches we identified were actually cardinality mismatches; non-cardinality-based type mismatches appear to be rare, if our small sample set is a reliable guide. This can be explained by the following two observations. First, the inputs and outputs of most bioinformatics analysis operations are weakly typed [5]. In most cases, parameters are either defined as strings or arrays of strings, regardless of the complexity of the data values actually being communicated. The second reason for the comparative rarity of non-cardinality-based type mismatch is that, for the time being, most of the

[4] http://www.ispider.man.ac.uk
[5] http://www.mygrid.org.uk
[6] http://bioinformatics.ubc.ca/pegasys/

available scientific workflow systems are not able to process complex types. Thus the work of parsing and constructing such data values is pushed down into the operations themselves.

Clearly, open questions remain regarding the best approaches to identify candidate mappings for identified mismatches. We are currently investigating the possibility of using mapping quality as a search criterion during mismatch resolution. This refers to the non-functional properties of mappings that may help the designer to select the best mapping for a given context [6]. Examples of such properties include the information and computational resources used for performing the mapping.

Acknowledgements

The work presented in this paper was funded by a grant from the BBSRC. We are also grateful to Duncan Hull and Robert Stevens, and our colleagues in the ISPIDER project, for useful discussions on mismatches in scientific workflows.

References

1. K. Belhajjame, S. M. Embury, H. Fan, C. A. Goble, H. Hermjakob, S. J. Hubbard, D. Jones, P. Jones, N. Martin, S. Oliver, C. Orengo, N. W. Paton, A. Poulovassilis, J. Siepen, R. Stevens, C. Taylor, N. Vinod, L. Zamboulis, and W. Zhu. Proteome data integration: Characteristics and challenges. In *UK All Hands Meeting*, 2005.
2. Sh. Bowers and B. Ludäscher. An ontology-driven framework for data transformation in scientific workflows. In *DILS*, pages 1–16, 2004.
3. Sh. Bowers and B. Ludäscher. Actor-oriented design of scientific workflows. In *ER*, pages 369–384, 2005.
4. T. Gruber. A translation approach to portable ontology specifications. *Knowledge Acquisition*, 5(2):199–220, 1993.
5. D. Hull, R. Stevens, P. Lord, C. Wroe, and C. Goble. Treating shimantic web syndrome with ontologies. In *First Advanced Knowledge Technologies workshop on Semantic Web Services (AKT-SWS04)*, 2004.
6. E. M. Maximilien and M. P. Singh. A framework and ontology for dynamic web services selection. *IEEE Internet Computing*, 8(5):84–93, 2004.
7. Th. M. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, R. M. Greenwood, T. Carver, K. Glover, M. R. Pocock, A. Wipat, and P. Li. Taverna: a tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20(17):3045–3054, 2004.
8. I. J. Taylor, M. S. Shields, I. Wang, and O. F. Rana. Triana applications within grid computing and peer to peer environments. *J. Grid Comput.*, 1(2):199–217, 2003.
9. Ch. Wroe, R. Stevens, C. A. Goble, A. Roberts, and R. M. Greenwood. A suite of daml+oil ontologies to describe bioinformatics web services and data. *Int. J. Cooperative Inf. Syst.*, 12(2):197–224, 2003.
10. J. Yu and R. Buyya. A taxonomy of scientific workflow systems for grid computing. *SIGMOD Record*, 34(3):44–49, 2005.