An Ontology-Driven Framework for Data Transformation in Scientific Workflows*

Shawn Bowers and Bertram Ludäscher

San Diego Supercomputer Center
University of California, San Diego
La Jolla, CA, 92093-0505, USA,
{bowers, ludaesch}@sdsc.edu

Abstract. Ecologists spend considerable effort integrating heterogeneous data for statistical analyses and simulations, for example, to run and test predictive models. Our research is focused on reducing this effort by providing data integration and transformation tools, allowing researchers to focus on "real science," that is, discovering new knowledge through analysis and modeling. This paper defines a generic framework for transforming heterogeneous data within scientific workflows. Our approach relies on a formalized ontology, which serves as a simple, unstructured global schema. In the framework, inputs and outputs of services within scientific workflows can have structural types and separate semantic types (expressions of the target ontology). In addition, a *registration mapping* can be defined to relate input and output structural types to their corresponding semantic types. Using registration mappings, appropriate data transformations can then be generated for each desired service composition. Here, we describe our proposed framework and an initial implementation for services that consume and produce XML data.

1 Introduction

For most ecological and biodiversity forecasting, scientists repeatedly perform the same process: They select existing datasets relevant to their current study, and then use these datasets as input to a series of analytical steps (that is, a *scientific workflow*). However, ecological and biodiversity data is typically heterogeneous. Researchers must spend considerable effort integrating and synthesizing data so that it can be used in a scientific workflow. Reusing analytical steps within workflows involves a similar integration effort. Each analytic step (or *service* since they can be implemented as web services [CCMW01]) in a workflow consumes and produces data with a particular structural representation, much like a dataset. To compose existing services, the structural and semantic differences between the services must be resolved, and this resolution is typically performed by the scientist either manually or by writing a special-purpose program or script.

* This work supported in part by the National Science Foundation (NSF) grants ITR 0225676 (SEEK) and ITR 0225673 (GEON), and by DOE grant DE-FC02-01ER25486 (SciDAC-SDM).

The Science Environment for Ecological Knowledge (SEEK)[1] [Mic03] is a multidisciplinary effort aimed at helping scientists discover, access, integrate, and analyze distributed ecological information. We envision the use of web-enabled repositories to store and access datasets, including raw data and derived results, software components, and scientific workflows. Additionally, we wish to exploit formalized, ecological ontologies to help scientists discover and integrate datasets and services, as workflows are designed and executed.

This paper proposes a framework that exploits ontological information to support structural data transformation for scientific workflow composition. We believe data transformation is an integral part of *semantic mediation*, which aims at providing automated integration services within SEEK. The framework is designed to allow researchers to easily construct scientific workflows from existing services, without having to focus on detailed, structural differences. In particular, when a service is stored in SEEK, each input and output is annotated with a structural and (optionally) a semantic type. In our framework, a structural type—similar to a conventional programming-language data type—defines the allowable data values for an input or output, whereas a semantic type describes the high-level, conceptual information of an input or output, and is expressed in terms of the concepts and properties of an ontology. Thus, although structurally different, two services may still be semantically compatible based on their semantic types.

The goal of the framework is to exploit semantic types to (semi-) automatically generate mappings between services with heterogeneous structural types. By defining input and output *registration mappings*, which link a structural type to a corresponding semantic type, ontological information is used to inform data transformation. When a scientist wishes to compose two services, the input and output registration mappings are combined to create a correspondence between the two structural types. The correspondence is then used, when possible, to generate the desired data transformation, thus making the services structurally compatible.

The rest of this paper is organized as follows. Section 2 briefly describes scientific workflows and introduces an example workflow used throughout the paper. Section 3 defines our proposed framework for data transformation. Section 4 describes an initial implementation of the framework for services that exchange XML data. In particular, we define a language for specifying registration mappings, based on structural types expressed using XML Schema. Section 5 discusses related work. Section 6 concludes with a discussion of future work.

2 Scientific Workflows

SEEK extends the Ptolemy system [BCD$^+$02] to support scientific workflows. Ptolemy is an open-source, Java-based application that provides interfaces for designing and executing data-flow process networks [LP95]. In particular, we

[1] See http://seek.ecoinformatics.org/

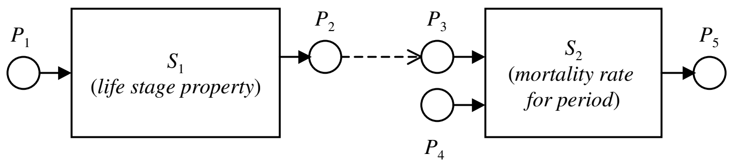


Fig. 1. Two connected services to calculate the k-value [BHT96] during a particular life-stage period.

are extending Ptolemy to support web-services, web-enabled repositories and workflow execution, scientific datasets, and semantic-type annotations through ontologies. This section briefly describes scientific workflows and introduces our running example. We note that our definition of a scientific workflow is inspired by, and compatible with, the dataflow models of Ptolemy.

We define a *scientific workflow* as a set of connected services. By *service*, we mean any software component that takes input and produces output, including but not limited to web services. A service has zero or more uniquely named *ports*. A port serves as either an input or an output to a service. Services exchange information using *connections*, which link an output port to one ore more input ports. When a pipeline is executed, data is transferred via connections from output to input ports according to an execution model. An execution model is an algorithm that determines how services should be scheduled and how data should flow through services.

Figure 1 depicts a simple workflow that contains two services S_1 and S_2, where the output port P_2 of S_1 is connected to the input port P_3 of S_2. The purpose of this simple workflow is to compute mortality rates for periods within the lifecycle of an organism [BHT96]. For example, consider the two datasets shown in Figure 2. The table on the left, Figure 2(a), gives population samples for specific development phases of the common field grasshopper (taken from Begon, *et al* [BHT96]). The dataset on the right, Figure 2(b), gives periods of development defined in terms of phases. Note that only one such period is given. Here, S_2 applies the "killing power," or k-value statistic to determine the rate of mortality for a set of observations (given in P_3) and a set of phases (given in P_4). For the datasets in Figure 2, S_2 would output a single pair (Nymphyl, 0.44) on port P_5. We note that, in SEEK, S_1 and S_2 represent stand-alone services that would be selected by an ecologist from a repository and connected, possibly as part of a larger workflow.

We require each port to have a *structural type*, which is similar to a conventional data type found in programming languages. In general, a structural type is expressed according to a *type system*, which consists of a set of base types, a description of the allowable types (of the type system), and rules defining when one type is a subtype of another. A type definition is a restriction on the structure of values being produced or consumed by a port. Thus, any value

(a)

Phase	*Observed*
Eggs	44,000
Instar I	3,513
Instar II	2,529
Instar III	1,922
Instar IV	1,461
Adults	1,300

(b)

Period	*Phases*
Nymphal	{Instar I, Instar II, Instar III, Instar IV}

Fig. 2. Example datasets for computing k-values during lifecycle periods.

that conforms to the structural type (or a subtype of the structural type) is an allowed value for the port.

We assume the function $\mathit{structType}(P)$ is well defined for each port P and returns the structural type of P. Given two ports P_a and P_b, we write $P_a \preceq P_b$ to denote that the structural type of P_a is a subtype of the structural type of P_b. If $P_a \preceq P_b$, we say P_a is *structurally compatible* with P_b.

One of the goals of SEEK is to allow scientists to reuse existing services when building new ecological models. In general, we assume services are not "designed to fit." That is, two distinct services may produce and consume compatible information, but have incompatible structural types.

3 Ontologies for Data Transformation

In this section, we define our proposed framework for ontology-driven data transformation. We first describe the use of semantic-type annotations in SEEK for enriching workflow services. We then describe how semantic types are exploited in our framework via registration mappings, which are used to generate data transformations between structurally incompatible services.

3.1 Ontologies and Services

As part of SEEK, we are developing technology that lets ecologists define, manage, and exploit ontologies. In general, we permit scientists to construct domain or project-specific ontologies and define mappings between them, when appropriate. In addition, we are developing, with the help of ecologists, an upper-level ecological ontology to serve as a framework for incorporating the various domain-specific ontologies. The upper-level ontology includes concepts and relationships for ecological properties, methods, measurements, and taxonomies. In the rest of this paper, we assume a single global ontology, however, in our envisioned environment there will more likely be many ontologies that when combined, for example, through mappings, form a single, global ontology.

A *semantic type* is defined using the concepts and properties of the ontology. In SEEK, we use semantic types to annotate services and datasets, where a semantic-type annotation defines the conceptual information represented by the

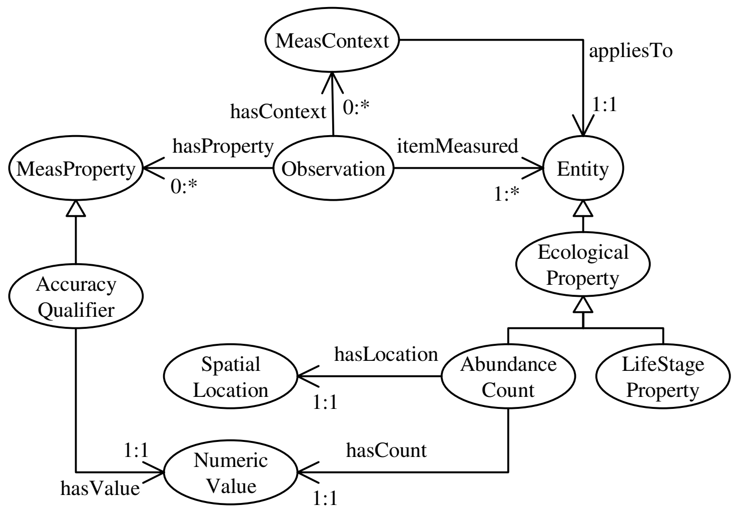


Fig. 3. Portion of a SEEK ontology for ecological measurements.

item. A semantic type defines the conceptual information that is either consumed or produced by a port. Thus, semantic types for ports are similar to semantic pre- and post-conditions in DAML-S [ABH$^+$02]. We note that in SEEK, however, semantic types are independent of the structural details of a port. We believe decoupling structural and semantic types has the following advantages. First, although we require structural types, semantic types are optional. Thus, a service can still be used even if it is not fully specified. Second, a port's semantic type can be defined or updated after the service is deployed, without requiring changes to the structural type. Finally, we believe semantic types are easier to specify if they do not mix structural information. For example, a port's semantic type may be as simple as a single concept label, even though it has a complex structural type.

We have adopted the Web Ontology Language (OWL) [MvH03] to express ontologies in SEEK. Figure 3 shows a fragment of the SEEK measurement ontology. The ontology is represented graphically, using RDF-Schema conventions [BG03]. Only a subset of the OWL constructs are used in Figure 3. In particular, we only consider class and property definitions, subclass relationships, and cardinality constraints on properties. (The notation 0:* and 1:* represent value and existential restrictions, respectively, and 1:1 represents an exactly-one number restriction [BN03].) Similarly, Figure 4 gives the semantic types for ports P_2 and P_3 of Figure 1. As shown, the semantic type of port P_3 accepts observations, each of which measures an abundance count within the context of a life-stage property. The semantic type of P_2 outputs similar observations, but with accuracy qualifiers. In general, the semantic type of a port is defined as an OWL concept.

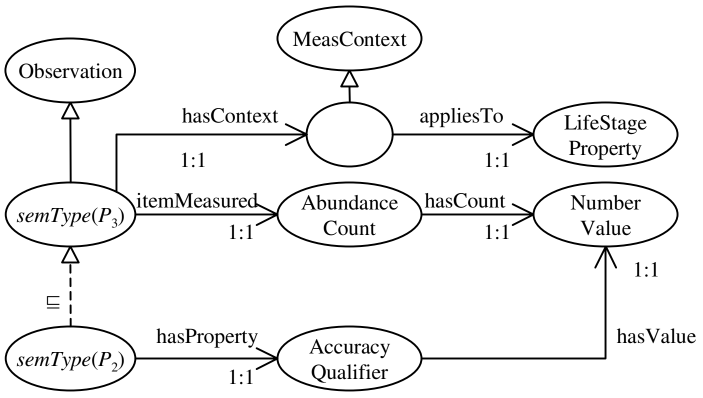


Fig. 4. Example semantic types for output and input ports P_2 and P_3.

We assume the function $\mathit{semType}(P)$ returns the semantic type of a port P. As stated above, a semantic type denotes a concept, which either exists as, or is a restricted subclass of, a concept within an ontology. As shown in Figure 4, $\mathit{semType}(P_2)$ is a subtype of $\mathit{semType}(P_3)$, which we denote using the standard $\sqsubseteq$ relation used in description logics. Intuitively, $P_2 \sqsubseteq P_3$ holds if every instance of the concept $\mathit{semType}(P_2)$ is also an instance of the concept $\mathit{semType}(P_3)$.

3.2 Connecting Semantically Compatible Services

To correctly compose two services, a valid connection must be defined between an output port of the source service and an input port of the target service. Intuitively, a desired connection (see Figure 5) is valid if it is both semantically and structurally valid. A connection from port P_s to port P_t is *semantically valid* if $P_s \sqsubseteq P_t$ and *structurally valid* if $P_s \preceq P_t$. The connection is *structurally feasible* if there exists a structural transformation δ such that $\delta(P_s) \preceq P_t$.

Our focus here is on the situation shown in Figure 5. Namely, that we have semantically valid connections that are not structurally valid, but feasible. Thus, the goal is to find a δ that implements the desired structural transformation.

3.3 Structural Transformation Using Registration Mappings

Figure 6 shows our proposed framework for data transformation in scientific workflows. The framework uses *registration mappings*, which consists of a set of rules that define associations between a port's structural and semantic types. In this paper, we use registration mappings to derive data transformations. A rule $q \leftrightarrow p$ in a registration mapping associates data objects identified with a query q to concepts identified with a concept expression p. For example, q may expressed as an XQuery [BCF+03] for XML sources or as an SQL query for relational sources, selecting a set of objects belonging to the same concept

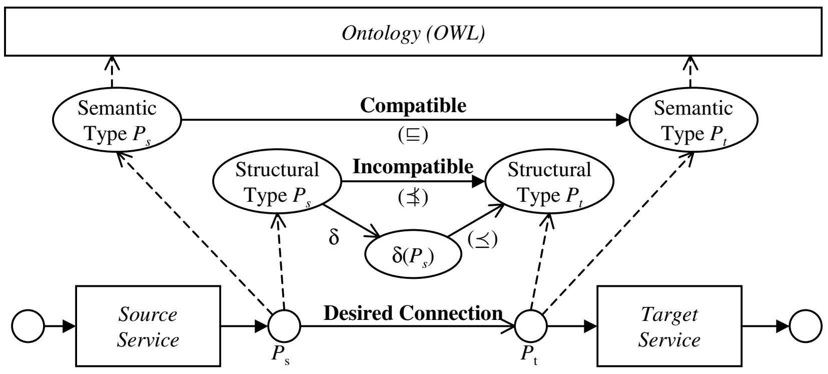


Fig. 5. A semantically valid, but structurally invalid connection.

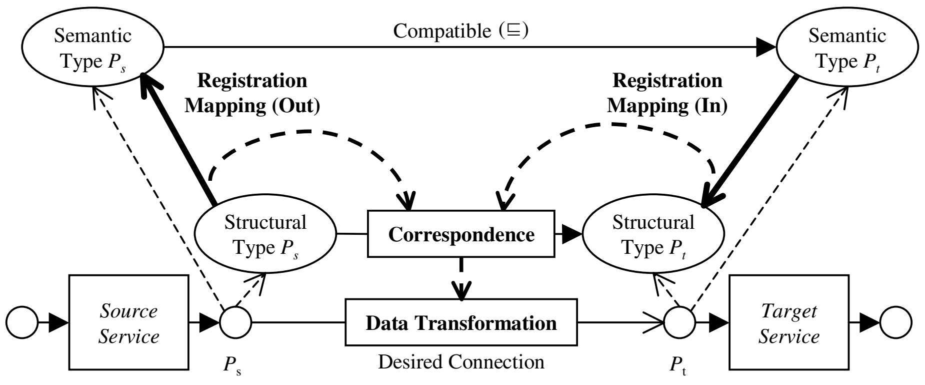


Fig. 6. The proposed transformation framework.

expression p. Thus, $q \leftrightarrow p$ is the part of a registration mapping that registers the q-selected objects with concepts denoted by p. We note that in general, p can represent a single concept or possibly a complex, description logic expression. In this section, we describe the general approach shown in Figure 6, and in the next section we define a specific implementation of the framework for XML sources, which includes a specific language for expressing registration mappings.

A registration mapping consists of one or more rules of the form $q \leftrightarrow p$. The expression q is a query that selects instances of the structural type to register to a concept denoted by p from the semantic type. We call the query q a *sub-structure* selection. In this paper, we only consider concept expressions p that denote *contextual paths* as opposed to arbitrary description logic expressions. A context path denotes a concept, possibly within the context of other concepts. For example, using the ontology fragment given in Figure 3, *Observation.itemMeasured.hasLocation* is a contextual path, where the concept selected by the path is *SpatialLocation* within the context of an observation measurement. Given a registration mapping rule $q \leftrightarrow \mathit{Observation.itemMeasured.hasLocation}$, we say that each data object selected by q is a *SpatialLocation* object for an $\mathit{AbundanceCount}$, which is the item being measured for some *Observation* object. It is important to note that we do not explicitly provide unique identifiers for data objects within a mapping rule. Instead, we use the contextual path simply as an annotation mechanism. Thus, data objects are identified locally to a structural-type instance, and these local identifiers are annotated with the appropriate semantic types. When a contextual-path expression is defined, it can be checked for validity (that is, that it makes sense) with respect to the semantic type, for example, with the help of a description-logic reasoner..

We distinguish between *output* and *input* registration mappings, where an output mapping is a registration mapping defined for an output port, and an input mapping is a registration mapping defined for an input port. Output and input registration mapping rules are composed to construct a *correspondence mapping* between the source and target structural types (see figure below). A correspondence mapping consists of rules of the form $q_s \mapsto q_t$, where q_s and q_t are queries against instances of output (source) and input (target) structural types, respectively. A rule in a correspondence mapping states an association between the two sub-structures defined by q_s and q_t. We construct individual correspondence mapping rules by composing registration mapping rules as shown in the figure below. Note that from a correspondence mapping, we wish to construct the appropriate transformation function δ.

$$\begin{tikzcd}
p_s \arrow[r, "\sqsubseteq_{path}"] & p_t \arrow[d] \\
q_s \arrow[u] \arrow[r, dashed, "\delta"'] & q_t
\end{tikzcd}$$

Given a set of output registration-mapping rules R_s and a set of input registration-mapping rules R_t, the correspondence mapping M between R_s and R_t is exactly the result of the *semantic join* of R_s with R_t (written $R_s \bowtie_{sem} R_t$), which we defined as follows. If $q_s \leftrightarrow p_1 \in R_s$ and $q_t \leftrightarrow p_2 \in R_t$, then $q_s \mapsto q_t \in R_s \bowtie_{sem} R_t$ if and only if $p_1 \sqsubseteq_{path} p_2$. The expression $p_1 \sqsubseteq_{path} p_2$ holds if p_1 is a *contextual subpath* of p_2, that is, if the concept denoted by p_1 is a subconcept of the concept denoted by p_2, and the context of p_1 is a subcontext of the context of p_2. We formally define the $\sqsubseteq_{path}$ relation for a simple semantic-path language in the next section.

As shown in Figure 6, we use a set of correspondence mappings M to generate a data transformation that can convert valid source data to valid target data. We note that correspondence mappings may be *underspecified*, that is, there may be more than one possible data transformation based on the correspondence mapping.

In general, we require registration mappings to be well-formed, that is, each query and contextual path is well-formed. We also make the following assumptions concerning registration mappings. We note that the complexity of enforcing the following properties depends on the languages used to express structural types, structural queries, semantic types, and contextual paths.

- **Consistent with respect to cardinality constraints**. A registration mapping is consistent with respect to cardinality constraints if, based on the rules of the mapping, the structural-type cardinality constraints imply the semantic-type cardinality (i.e., value restriction) constraints. Note that if the cardinality constraints between the structural and semantic types are inconsistent, the generated data transformation program may not be correct.
- **Partially complete**. A registration mapping is considered *structurally* complete if every data item required by a structural type is registered with some contextual path in the semantic type. Similarly, a registration mapping is *semantically* complete if every required concept in a semantic type (according to cardinality constraints) is registered by some data item in an instance of the structural type. We require all input registration mappings to be structurally complete. We also require all output registration mappings to be semantically complete with respect to the input registration mapping. That is, the output registration mapping should map data items to the needed concepts of the input registration mapping.

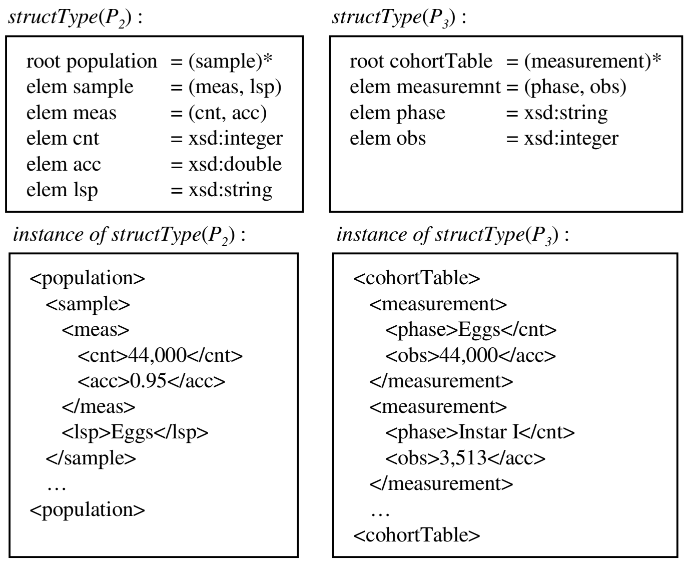
4 A Framework Implementation for XML-based Services

In this section, we describe an instantiation of the framework defined in Section 3 in which XML is used to interchange data between services. We define a subset of XML Schema [BM01, TBMM01] for the structural-type system, a simple XPath-based query language, a contextual-path language, and the contextual subpath relationship for contextual paths.

4.1 XML-based Structural Types

We consider a subset of XML Schema for expressing structural types, which is very close to Document Type Definitions (DTDs), but with XML datatypes [BM01]. In particular, a structural-type consists of one or more element definitions, one of which is a distinguished root element. For simplicity, we do not consider attributes. Instead, we use the common assumption that attributes are modeled as simple elements containing text values (where attribute names are often prefixed with the '@' symbol).

A structural type takes the form `root` e=cm or `elem` e=cm, where e is an element tag name and cm is the content model for e. Every structural type has exactly one root-element definition (which is distinguished by the `root` qualifier). A structural type may also have any number of additional element definitions (which are distinguished by the `elem` qualifier).



structType(P_2) :

```
root population  = (sample)*
elem sample      = (meas, lsp)
elem meas        = (cnt, acc)
elem cnt         = xsd:integer
elem acc         = xsd:double
elem lsp         = xsd:string
```

structType(P_3) :

```
root cohortTable = (measurement)*
elem measuremnt = (phase, obs)
elem phase       = xsd:string
elem obs         = xsd:integer
```

instance of structType(P_2) :

```
<population>
  <sample>
    <meas>
      <cnt>44,000</cnt>
      <acc>0.95</acc>
    </meas>
    <lsp>Eggs</lsp>
  </sample>
  …
<population>
```

instance of structType(P_3) :

```
<cohortTable>
  <measurement>
    <phase>Eggs</cnt>
    <obs>44,000</acc>
  </measurement>
  <measurement>
    <phase>Instar I</cnt>
    <obs>3,513</acc>
  </measurement>
  …
<cohortTable>
```

Fig. 7. Example structural types and instances for port P_2 (left) and port P_3 (right).

We permit datatype and regular-expression content models. For example, the expression `elem cnt = xsd:integer` restricts `cnt` elements to contain integer values (where `xsd:integer` is an XML-Schema datatype). A content model is defined using standard DTD regular-expression syntax. For example, the element definition `elem sample = (meas,lsp)` states that a `sample` element consists of a `meas` element followed by a `lsp` element. We require each element definition to be nested under the root element either directly, as a subelement, or indirectly, through any number of intermediate subelements.

To illustrate, we assume that ports P_2 and P_3 in Figure 1 have the structural-type definitions given in the top of Figure 7. The structural type of P_2 is shown on the top-left and the structural type of P_3 is shown on the top-right. Sample instances (XML documents) are given below each corresponding structural type. As shown, the structural types are not subtypes, that is, $P_2 \not\preceq P_3$. In general, subtyping in XML is based on datatype compatibility (for example, `xsd:negativeInteger` is a subtype of `xsd:integer`) and various rules concerning content models (for example, the content model `(a,b,c)` is a subtype of the content model `(a|b|c)*` for element definitions `a`, `b`, and `c`).

4.2 Structural-Type Queries and Contextual Paths

We define a structural-type query q_{xp} for XML using a subset of XPath [CD99]. A query q_{xp} is expressed using the following syntax. We note that the fragment of XPath we consider is similar to *tree patterns* [MS02].

$q_{xp} = \texttt{/}p$
$p \quad = n \mid p\texttt{/}p \mid p\texttt{/text()} \mid p\texttt{[}q\texttt{]}$
$q \quad = p \mid p\texttt{=}v$

As shown, n represents an element tag name and v represents a text value (that is, PCDATA). The expression `/text()` selects the PCDATA content of an element.

A query is expressed as a simple path that always starts from the root element type defined in the XML structural type. A query returns either fragments (that is, sub-trees) of a document, or using the `text()` operator, a set of data values. In addition, we permit simple qualifiers q, which select sub-trees that contain a particular set of subelements, possibly with a specific data value.

The language we consider for expressing contextual paths is defined as follows. A contextual path defined over port P takes the form $\mathit{semType}(P).r_1.r_2...r_n$ for $n \geq 0$, where r_1 to r_n are valid properties defined for the semantic type of P. A contextual path always begins from the semantic type of a port. Additionally, a contextual path always denotes a single concept, which is either the semantic type of the port or a concept related to the semantic type through properties r_1 to r_n. Given a contextual path p, the function $\mathit{leadsTo}(p)$ returns the concept denoted by the contextual path p.

4.3 Registration and Correspondence Mappings

Figure 8 gives an example input and output registration mapping for ports P_2 and P_3 of Figure 6 (using the structural types for P_2 and P_3 defined in Figure 7). Assuming we wish to connect port P_2 to port P_3, we first use the input registration mapping R_s and the output registration mapping R_t of Figure 8 to generate a correspondence mapping. Recall that a correspondence mapping M for R_s and R_t is equivalent to $R_s \bowtie_{sem} R_t$, which is computed using the semantic subpath relation $\sqsubseteq_{path}$ (see Section 3.3).

We define the semantic subpath relation for our contextual path language as follows. Let p_1 and p_2 be the following contextual paths expressed over ports P_a and P_b, respectively.

$p_1 = \mathit{semType}(P_a).r_{11}.r_{12}. \ ... \ .r_{1n}$
$p_2 = \mathit{semType}(P_b).r_{21}.r_{22}. \ ... \ .r_{2m}.$

The relation $p_1 \sqsubseteq_{path} p_2$ is true if and only if $m = n$, $P_a \sqsubseteq P_b$, and for $1 \leq i \leq n$, $r_{1i} = r_{2i}$ and $\mathit{leadsTo}(\mathit{semType}(P_a).r_{11}. \ ... \ .r_{1i}) \sqsubseteq \mathit{leadsTo}(\mathit{semType}(P_b).r_{21}. \ ... \ .r_{2i})$. To illustrate, the correspondence mapping M that results from applying the semantic join of R_s and R_t (from Figure 8) is shown in Figure 9.

4.4 Generating Data-Transformation Programs

In general, we want to use correspondence mappings to construct valid target data instances (that is, XML documents that conform to the target structural

P_2 **output registration-mapping rules** ($q_s \leftrightarrow p$)**:**

`/population/sample`	$\leftrightarrow$	$semType(P_2)$
`/population/sample/meas/cnt`	$\leftrightarrow$	$semType(P_2)$.itemMeasured
`/population/sample/meas/cnt/text()`	$\leftrightarrow$	$semType(P_2)$.itemMeasured.hasCount
`/population/sample/meas/acc`	$\leftrightarrow$	$semType(P2)$.hasProperty
`/population/sample/meas/acc/text()`	$\leftrightarrow$	$semType(P2)$.hasProperty.hasValue
`/population/sample/lsp/text()`	$\leftrightarrow$	$semType(P2)$.hasContext.appliesTo

P_3 **input registration-mapping rules** ($p \leftrightarrow q_t$)**:**

`/cohortTable/measurement`	$\leftrightarrow$	$semType(P3)$
`/cohortTable/measurement/obs`	$\leftrightarrow$	$semType(P3)$.itemMeasured
`/cohortTable/measurement/obs/text()`	$\leftrightarrow$	$semType(P3)$.itemMeasured.hasCount
`/cohortTable/measurement/phase/text()`	$\leftrightarrow$	$semType(P3)$.hasContext.appliesTo

Fig. 8. Example registration mappings for ports P_2 and P_3.

`/population/sample`	$\mapsto$	`/cohortTable/measurement`
`/population/sample/meas/cnt`	$\mapsto$	`/cohortTable/measurement/obs`
`/population/sample/meas/cnt/text()`	$\mapsto$	`/cohortTable/measurement/obs/text()`
`/population/sample/lsp/text()`	$\mapsto$	`/cohortTable/measurement/phase/text()`

Fig. 9. Correspondence mapping that results from connecting ports P_2 and P_3.

type) from valid source data instances. The correspondence mapping from Figure 9 provides the following guidelines for developing such a mapping. Each population sample (`/population/sample`) should be used to generate a cohort table measurement (`/cohortTable/measurement`), where each sample's measurement count (`meas/cnt`) corresponds to an observation in the measurement element of the cohort table (`obs`), and each sample's life stage property (`lsp/text()`) corresponds to a phase in the measurement element of the cohort table (`phase`). Based on the correspondence mapping, the following data transformation (expressed using XQuery [BCF$^+$03]) should be generated.

```
<cohortTable>
   { for $s in /population/sample return
         <measurement>
            { for $c in $s/meas/cnt return <obs>{$c/text()}</obs> }
            { for $l in $s/lsp return <phase>{$l/text()}</phase> }
         </measurement> }
</cohortTable>
```

To generate the above data transformation, we must make the following assumptions.

1. **Common prefixes refer to the same element**. We assume correspondence mapping rules that share a common prefix (in the source or target

side of the rule) refer to the same element (in the source or target side, respectively). Note that all correspondence rules, by definition, share at least the root element. For example, the first two correspondence rules in Figure 9 share the common source prefix `/population/sample` and target prefix `/cohortTable/measurement`. Thus, for a `/population/sample` element *s* mapped to a `/cohortTable/measurement` element *m*, we assume that each `cnt` element *c* under *s* is mapped to an `obs` element *o* under *m*.

2. **Correspondence rules have compatible cardinality restrictions**. By cardinality restrictions for XML, we mean whether a subelement can occur zero or more, one or more, zero or one, or exactly once under an element. For example, the XML fragments selected by the source and target side of the first correspondence rule in Figure 9 have identical cardinality restrictions. Namely, a root `population` element can contain zero or more `sample` elements and similarly, a root `cohortTable` element can contain zero or more `measurement` elements. In general, the source restrictions should be equivalent or stricter than the target restrictions. Thus, we do not want to map multiple source elements to a single target element.
3. **Datatypes are compatible**. We assume that each correspondence mapping rule between XML text values is valid. In particular, we assume that the datatype of the source is a subtype of the target. For example, if `cnt` was defined as a double type instead of an integer type, the third correspondence mapping rule would violate our subtype assumption, because a double is not a subtype of an integer (where `obs` requires integer content).

For registration mappings that satisfy the above assumptions and are known to be partially complete (see Section 3.3), it is straightforward to generate the appropriate data-transformation program. However, for mappings that are not partially complete (as given in Section 3.3) or do not have compatible cardinality restrictions, the resulting correspondence mapping is *underspecified*. That is, there may be many potential data transformations that can be generated from the correspondences. As future work, we want to further explore approaches for generating data-transformation programs from correspondence mappings, when the above assumptions do not hold.

5 Related Work

In this paper, we have presented a generic framework for enabling service composition in scientific workflows. The framework exploits three distinct specifications for services: structural types, semantic types, and registration mappings (which link structural and semantic types). We believe that separating these three specifications is a practical solution for enabling the reuse of legacy services. Also, under certain assumptions, registration mappings can be used to automatically generate desired structural transformations, allowing scientists to easily connect heterogeneous, but semantically similar services to form new workflows.

Although our goal is to integrate workflow services, our framework attempts to solve a different problem than typical data-integration systems

[Ull97, PB03, LGM01, PS98, PAGM96]. In particular, we want to construct data transformations that can take valid data under one schema, and convert it to valid data under another schema. In data integration, the focus is on combining source data expressed in heterogenous local schemas into an integrated view under a global schema. Note that some work has explored the use of ontologies as global schemas for data integration [SSR94, LVL03].

A number of languages have been defined for explicitly expressing data transformations between heterogeneous schemas [KLK91, CDSS98, DK97]. However, in scientific workflows, we believe it is important to compute data transformations whenever possible, as opposed to requiring scientists to express transformations manually, each time a connection is required. We note that a particular service may be used in a number of workflows, which would require a distinct data transformation mapping for each desired connection.

Instead, we use registration mappings from structural to semantic types (where a semantic type is similar to a global schema) to generate correspondence mappings. A number of recent approaches have explored the use of correspondences in data integration and transformation [PS98, PB03, PVM+02]. In Clio [PVM+02], a user specifies a set of correspondences between items in two structurally heterogeneous schemas, and using the constraints of the schemas, the system attempts to fill in, or generate, a complete mapping. Similarly, Pottinger and Bernstein [PB03] use simple correspondences to merge structurally heterogeneous schemas for data integration.

6 Future Work

We plan to further develop and extend the ontology-driven framework described here within the context of SEEK. Our next step is to extend Ptolemy with the XML-based framework described in Section 4. As part of this work, we also want to explore the relationships between correspondence mappings and the generation of data-transformation programs. In particular, we are interested in developing techniques to help generate data transformations for underspecified correspondence mappings and to extend the approach to support additional semantic conversions, for example, to automatically perform unit conversion. We also believe a more complex structural-type query language may be required for complex structural conversions, and intend to extend the simple XPath-based approach presented here as needed. Finally, we plan to investigate how the use of different computation models and workflow scheduling algorithms can influence, and help generate, data transformations.

References

[ABH+02] Anupriya Ankolenkar, Mark Burstein, Jerry R. Hobbs, Ora Lassila, David L. Martin, Drew McDermott, Sheila A. McIlraith, Srini Narayanan, Massimo Paolucci, Terry R. Payne, and Katia Sycara. DAML-S: Web service description for the semantic web. In *The First International Semantic Web Conference (ISWC)*, June 2002.

[BCD$^+$02] Shuvra S. Bhattacharyya, Elaine Cheong, John Davis II, Mudit Goel, Christopher Hylands, Bart Kienhuis, Edward A. Lee, Jie Liu, Xiaojun Liu, Lukito Muliadi, Steve Neuendorffer, John Reekie, Neil Smyth, Jeff Tsay, Brian Vogel, Winthrop Williams, Yuhong Xiong, and Haiyang Zheng. Heterogeneous concurrent modeling and design in java. Technical Report Memorandum UCB/ERL M02/23, EECS, University of California, Berkeley, August 2002.

[BCF$^+$03] Scott Boag, Don Chamberlin, Mary F. Fernández, Daniel Florescu, Jonathon Robie, and Jérôme Siméon, editors. *XQuery 1.0: An XML Query Language*. W3C Working Draft. World Wide Web Consortium (W3C), November 2003. http://www.w3.org/TR/2003/WD-xquery-20031112/.

[BG03] Dan Brickley and R.V. Guha, editors. *RDF Vocabulary Description Language 1.0: RDF Schema*. W3C Working Draft. World Wide Web Consortium (W3C), February 2003. http://www.w3.org/TR/2003/WD-rdf-schema-20030123/.

[BHT96] Michael Begon, John L. Harper, and Colin R. Townsend. *Ecology: Individuals, Populations, and Communities*. Blackwell Science, 1996.

[BM01] Paul V. Biron and Ashok Malhotra, editors. *XML Schema Part 2: Datatypes*. W3C Recommendation. World Wide Web Consortium (W3C), May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[BN03] Franz Baader and Werner Nutt. Basic description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P.F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, 2003.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana, editors. *Web Services Description Language (WSDL) 1.1*. W3C Note. World Wide Web Consortium (W3C), March 2001. http://www.w3.org/TR/2001/NOTE-wsdl-20010315.

[CD99] James Clark and Steve DeRose, editors. *XML Path Language Version 1.0*. W3C Recommendation. World Wide Web Consortium (W3C), November 1999. http://www.w3.org/TR/1999/REC-xpath-19991116.

[CDSS98] Sophie Cluet, Claude Delobel, Jérôme Siméon, and Katarzyna Smaga. Your mediators need data conversion! In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 177–188. ACM Press, 1998.

[DK97] Susan B. Davidson and Anthony Kosky. WOL: A language for database transformations and constraints. In *Proceedings of the 13th International Conference on Data Engineering (ICDE)*, pages 55–65. IEEE Computer Society, 1997.

[KLK91] Ravi Krishnamurthy, Witold Litwin, and William Kent. Language features for interoperability of databases with schematic discrepancies. In *Proceedings of the SIGMOD International Conference on Management of Data*, pages 40–49. ACM Press, 1991.

[LGM01] Bertram Ludäscher, Armanath Gupta, and Maryann E. Martone. Model-based mediation with domain maps. In *Proceedings of the 17th International Conference on Data Engineering (ICDE)*, pages 81–90. IEEE Computer Society, April 2001.

[LP95] Edward A. Lee and Thomas M. Parks. Dataflow process networks. *Proceedings of the IEEE*, 83(5):773–801, 1995.

[LVL03] Fereidoon Sadri Laks V.S. Lakshmanan. Interoperability on XML data. In *Proceedings of the International Semantic Web Conference (ISWC)*, volume 2870, pages 146–163. Lecture Notes in Computer Science, 2003.

[Mic03] William K. Michener. Building SEEK: the science environment for ecological knowledge. In *DataBits: An electronic newsletter for Information Managers*, 2003. Spring Issue.

[MS02] Gerome Miklau and Dan Suciu. Containment and equivalence for an XPath fragment. In *Proceedings of the 21st Symposium on Principles of Database Systems (PODS)*, pages 65–76. ACM Press, June 2002.

[MvH03] Deborah L. McGuinness and Frank van Harmelen, editors. *OWL Web Ontology Language Overview.* W3C Candidate Recommendation. World Wide Web Consortium (W3C), August 2003. http://www.w3.org/TR/2003/CR-owl-features-20030818/.

[PAGM96] Yannis Papakonstantinou, Serge Abiteboul, and Hector Garcia-Molina. Object fusion in mediator systems. In *Proceedings of 22nd International Conference on Very Large Data Bases (VLDB)*, pages 413–424. Morgan Kaufmann, September 1996.

[PB03] Rachel Pottinger and Philip A. Bernstein. Merging models based on given correspondences. In *Proceedings of the 29th International Conference on Very Large Data Bases (VLDB)*, pages 826–837. Morgan Kaufmann, September 2003.

[PS98] Christine Parent and Stefano Spaccapietra. Issues and approaches of database integration. *Communications of the ACM*, 41(5):166–178, May 1998.

[PVM$^+$02] Lucian Popa, Yannis Velegrakis, Renée J. Miller, Maricio Hernández, and Ronald Fagin. Translating Web data. In *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB)*, 2002.

[SSR94] Edward Sciore, Michael Siegel, and Arnon Rosenthal. Using semantic values to falilitate interoperability among heterogeneous information systems. *ACM Transactions on Database Systems*, 19(2):254–290, 1994.

[TBMM01] Henry S. Thompson, David Beech, Murray Maloney, and Noah Mendelsohn, editors. *XML Schema Part 1: Structures.* W3C Recommendation. World Wide Web Consortium (W3C), May 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

[Ull97] Jeffrey D. Ullman. Information integration using logical views. In *Proceedings of the International Conference on Database Theory (ICDT)*, volume 1186, pages 19–40. Lecture Notes in Computer Science, 1997.