Automatic Composition of Aggregation Workflows for Transportation Modeling

José Luis Ambite
Information Sciences Institute
University of Southern California
4676 Admiralty Avenue
Marina del Rey, CA 90254
ambite@isi.edu

Matthew Weathers
Information Sciences Institute
University of Southern California
4676 Admiralty Avenue
Marina del Rey, CA 90254
matthew.weathers@usc.edu

ABSTRACT

Many scientific problems can be modeled as computational workflows that integrate data from heterogeneous sources and process such data to derive new results. These data analysis problems are pervasive in the physical and social sciences, as well as in government practice. In this paper, we present an approach to automatically create computational workflows in response to user data requests. We represent both data access and data processing operations uniformly as web services. We describe the inputs and outputs of the services according to an ontology of the application domain expressed in RDF/RDFS. Our system uses the Triple logic engine to formally represent the ontology and the services, and to automatically generate the workflows.

This work is part of the Argos project that is developing a flexible data query and analysis system based on the web services paradigm. Our application domain is goods movement analysis and its effects on spatial urban structure. Since our ontology represents data items as multi-dimensional objects, with hierarchical values for each dimension, in this paper we focus on automatically generating workflows that include aggregation operations.

Categories and Subject Descriptors

H.2.5 [**Information Systems**]: Database Management—*Heterogeneous Databases*; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*Web-based services*; D.1.6 [**Software**]: Programming Techniques—*Logic Programming*

General Terms

Web Service Composition, Workflow, Information Integration, Triple Logic, RDF

1. INTRODUCTION

Decision makers at all levels of government are awash with information. However, there is a critical lack of tools to locate, access and, most importantly, analyze such information efficiently. The need is particularly acute in economic modeling and in planning agencies. In the Argos project we are developing a flexible data analysis system based on the web services paradigm to address this need.

Recent advances in computer science research provide tools that greatly reduce the cost of accessing and processing information. There has been significant progress in data integration, the problem of accessing and querying data integrated from distributed and heterogeneous sources (see [16, 11] for surveys). In parallel, proposals for web services standards, such as WSDL [6], BPEL4WS [3] define XML-based protocols for distributed computational services. However, there are limitations to the current state of the art both in data integration and in web services. First, query evaluation plans in data integration systems are composed of classical relational algebra operations, but do not include arbitrary computations as required in scientific workflows. Second, current web services tools expect a programmer to compose web services manually, either by programming in a computer language such as Java, or by specifying a workflow more declaratively as in BPEL4WS. This misses much of the potential for web services, which are computational modules with declarative input/output descriptions. To address these limitations and combine the benefits of the data integration and web services approaches, we are developing Argos, a novel architecture for web service composition based on expressive web service descriptions that enables services compositions to be automatically derived similarly to the way query plans are generated in data integration systems.

As an application domain, we examine several goods movement planning problems and their effects on spatial urban structure. This domain is an excellent candidate to ground our architecture. First, commodity flows carry a significant economic impact. Second, the domain encompasses all levels of government: federal, state, and local. Third, there is a lack of tools to help practitioners to track, analyze, and monitor these flows, both at government agencies as well as academic analysis. Finally, it is a technically challenging domain from the computer science perspective. The data and

dg.o2005 Atlanta, GA, USA

processing operations are complex, distributed, and heterogeneous.

In this paper, we describe our modeling approach and we focus on automatic workflow composition in the presence of aggregation operations. The rest of the paper is organized as follows. First, we describe the structure of our ontology. Second, we show how we model data sources and operations. Third, we show our encoding of the web service composition problem as a Triple [26] logic program, focusing on aggregation operations. Fourth, we present some scalability experiments. Finally, we discuss related work, discuss our contributions and our plans for future work.

2. MODELING THE DOMAIN

In order to integrate data from multiple sources and to analyze it automatically, we need to assign formal semantics to the data. Therefore, we have defined an ontology for our application domain in consultation with our domain experts.

The ontology describes the data items in the different sources and the data produced by the operations uniformly as multidimensional objects. That is, each object is defined as having values along a set of attributes or dimensions. These values are themselves objects that are organized in hierarchies, most of which have part-of semantics.

Our ontology is organized around the concept of measurement, since most of the data we process can be characterized as measuring some economic quantity. Figure 1 shows the basic structure of the measurement concept. A measurement has the following core dimensions:

- Geo: This is the geographical entity to which a measurement applies. For example, we may be measuring employment data in the Los Angeles Consolidated Metropolitan Statistical Area (CMSA). We defined an ontology of geospatial regions relevant to our domain. This includes obvious choices like countries, states, counties, and cities, and the technical areas used in describing the data in our domain, such as CSMAs, Traffic Analysis Zones (TAZs), Census Tracts, Highways, Ports, etc. Whenever useful we also identified the relationships between the different areas. In particular, we recorded spatial containment. For example, we record that Los Angeles County is part of California, which in turn is part of the United States; that the LA CSMA is comprised by Los Angeles, Orange, Riverside, San Bernardino, and Ventura counties; the TAZ that corresponds to the Long Beach port; and so on.

- Time Interval: This is the temporal extent covered by a measurement. For example, we may have the exports of cars from LA in year 1997, in the first quarter of 1998, in January 2000; or have hourly truck counts at a given point in a highway. To precisely compare time intervals, we define them as a pair or time points, with date and time attributes. That is, the interval January2000 would have a starting time point of 2000-1-1T00:00:00 and an ending time point of 2000-1-31T23:59:59 (using ISO 8601 notation). This representation induces a part-of hierarchy at different levels of granularity, similarly to our spatial dimension.

- Product: Much of the data in our domain refers to economic parameters of different industries, products, and commodities. This economic data is reported in a variety of classifications, including the Standard Classification of Transported Goods (SCTG), the Standard Industrial Classification (SIC), the North American Industry Classification System (NAICS), among many others. Unfortunately, there are no shortage of "standard" classifications. Thus, we must carefully record the product classification system used for each measurement and provide translations services among the different classifications if we want to reconcile data from different sources. Figure 2(a) shows a fragment of the SCTG product classification used in the Commodity Flow Survey of the US Economic Census. Figure 2(b) shows the corresponding fragment of the ontology displayed in the Protege ontology editor [23]. The aggregation hierarchy of products is specified by the `productContains` and `productPartOf` slots. In the example, the `SCTG01_05` product category is composed of `SCTG01`, `SCTG02`, `SCTG03`, `SCTG04`, and `SCTG05`. In turn `SCTG01_05` is a part of `SCTGAllCommodities`.

- Unit: This is the unit in which the measurement is expressed. Our ontology includes physical units, such as area, volume, and weight; monetary value expressed in different currencies; ratios, and counts. For example, cargo is often measured in metric tons (1 metric ton = 1000 Kg) or in short tons (1 short ton = 2000 pounds = 907.18474 Kg).

- Flow: A central concept in our transportation domain is the flow of commodities. We represent a flow with its origin and destination, which are geospatial entities (`geo`s), its mode of transportation (such as air, truck, rail, ship, or pipeline), and its conduit, which is a distinguished `geo` through which it passes. For example, some data from the Waterborne Commerce of the United States can be precisely characterized as a flow with origin the Los Angeles CMSA and destination the rest of the World except Canada (since Canada is treated separately) leaving by ship from the port of Long Beach.

According to this multidimensional representation, we can consider all the measurement data distributed across the sources or computed by operations as a virtual data cube. Each cell of the cube contains the corresponding value as described (indexed) by the dimensions. Of course, only a small subspace of the cube is actually available in the sources. The main task of our data analysis system is to fill the remaining values in this virtual datacube as requested by the user.

We have taken a minimalistic approach to ontology development. We restrict our ontology to concepts that are present in actual sources and/or are data in expected user requests. We refrain from creating much of an upper ontology, other than the minimum necessary to organize the entities coherently for our application domain. We do not attempt to represent the data with all the detail that would be needed

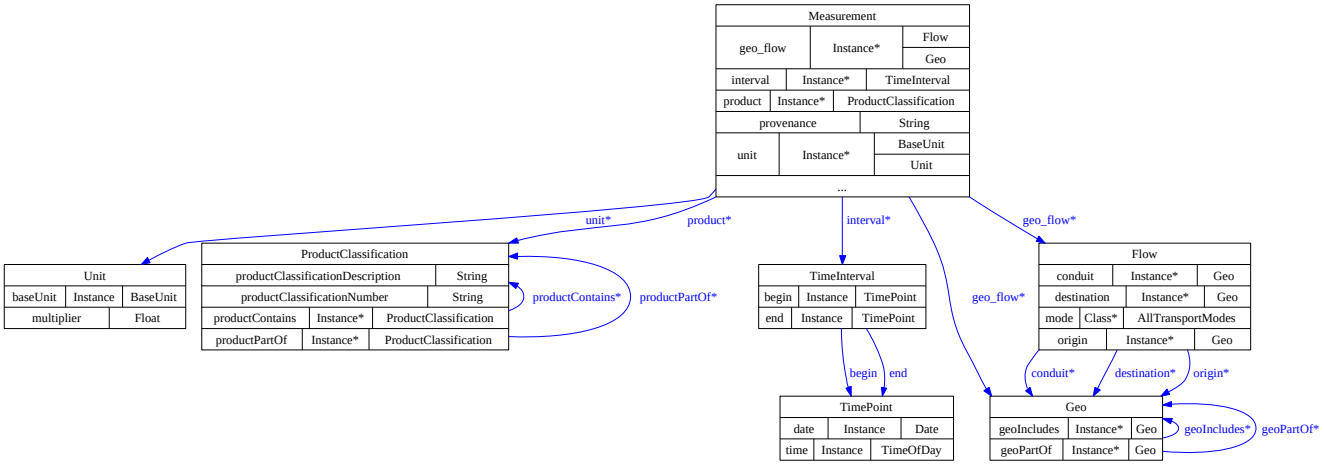
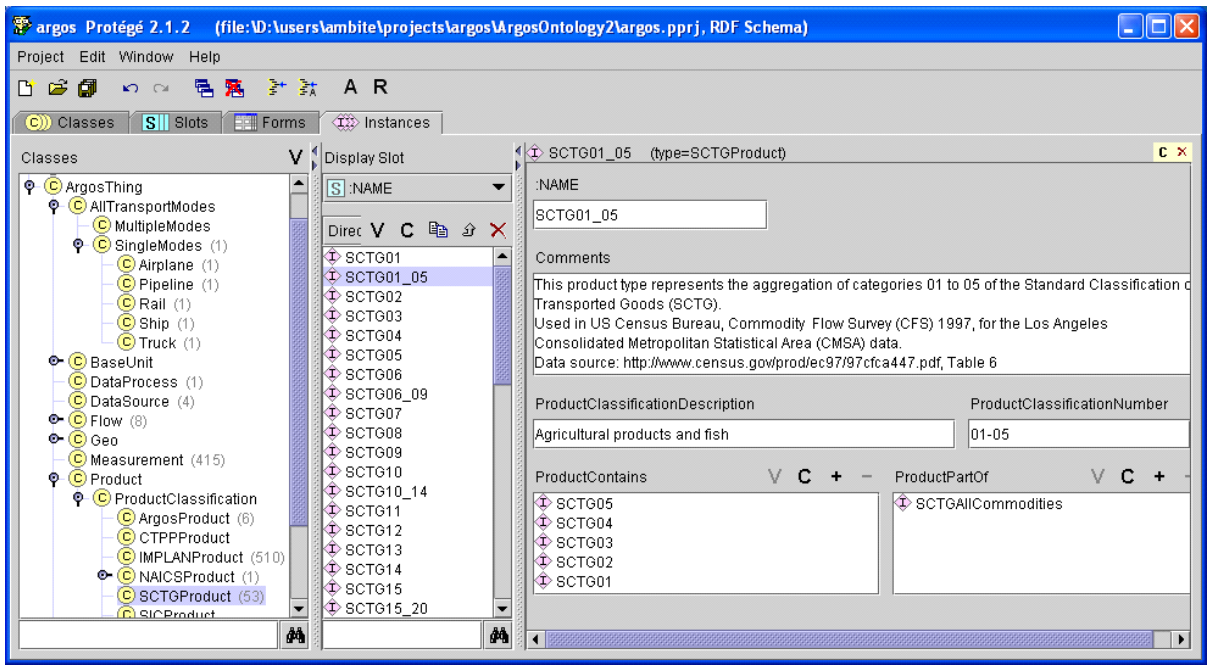


Figure 1: Ontology Fragment: Measurement

```
01-05 Agricultural products and fish
   01 Live animals and live fish
   02 Cereal grains
   03 Agricultural products, except live animals, cereal grains and forage products
   04 Animal feed and feed ingredients, cereal, straw, and eggs and other products of animal origin, n.e.c.
   05 Meat, fish, seafood, and preparations
06-09 Grains, alcohol, and tobacco products ...
```

(a) Fragment of the SCTG product classification

.



(b) Formalization (shown in the Protege Ontology Editor)

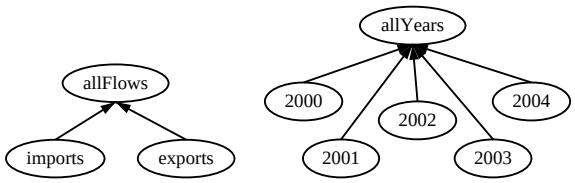
Figure 2: Ontology Fragment: SCTG product classification

for common-sense reasoning (cf. [15]), just enough to answer data requests. We have found this approach useful in this and previous projects [1]. So, we can see our ontology as structured controlled vocabulary with some critical relationships between concepts added.

We are also minimalistic in terms of the representation language for the ontology. We use the Resource Description Framework (RDF) [19] and RDF Schema [5], as opposed to more complex description languages, like the Web Ontology Language (OWL) [20]. So far the combination of RDF and the logic rule language Triple [26] has been sufficient for our representational needs. Triple is a variant of F-logic [12] specifically designed to manipulate RDF and well-suited for semantic web applications.

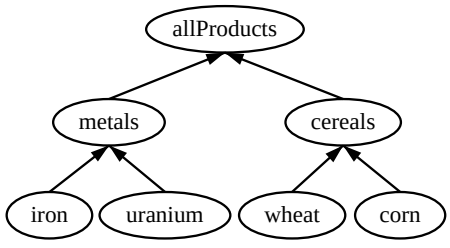
3. MODELING THE SOURCES

We formally describe the contents of each data source by defining the objects they provide according to our ontology. To illustrate our approach for modeling sources and services, and later our automatic composition techniques, we will use a simplified transportation ontology. In this simplified domain, we will use only 3 dimensions, flow, product, and time, to describe the total value of flows of different products in different years. These dimensions are also simplified. Figure 3 shows the hierarchies for the three dimensions. The flow hierarchy contains 3 nodes: `imports`, `exports`, and `allFlows` (which represents the total trade of a region and is the sum of imports and exports). The product hierarchy consists of seven nodes: `allProducts`, representing the total value over all kinds of products, that is decomposed into `metals` and `cereals`; in turn `metals` is composed of `iron` and `uranium`, and `cereals` of `wheat` and `corn`. Finally, the time hierarchy just considers a range of years of interest, 2000 to 2004, plus `allYears`, the total for the period.



(a) Flow dimension

(b) Time dimension



(c) Product dimension

Figure 3: Sample dimension hierarchies

As an example of modeling source contents, consider the six sources of Fig 4(a). We explicitly model each type of object that each source provides according to our virtual datacube. Fig 4(b) shows some of source descriptors expressed as Triple statements. For example, source s3 has two descriptors, one for exports and one for imports (of iron in 2000).

The data model of Triple includes the data model of RDF (plus additional classical logic programming constructs like functional terms and n-ary predicates.) The basic statement in Triple is a triple `subject[predicate->object]`. A full description of the syntax and semantics of Triple is beyond the scope of this paper, however we will describe them by example. For example, the third statement of Fig 4(b) declares an object named `source(s3,1)` that is related to the objects `imports`, `iron`, and `2000` through relationships `flow`, `product`, and `time`, respectively. This is a compound statement equivalent to the conjunction of the triples `source(s3,1)[flow->imports]`, `source(s3,1)[product->iron]`, and `source(s3,1)[time->2000]`.

Each source is described at the appropriate level of aggregation in the dimension hierarchies. For example, `s5` has as `product` the aggregated value for `cereals`, but does not provide values for `wheat` or `corn`, and for the total period, `allYears`, not for individual years. If the user requested the imports of wheat in 2001, the system would not be able to answer such query given these sources.[1]

Source	Flow	Product	Time
s1	imports	iron	2002-2004
s2	exports	iron	2002-2004
s3	imports, exports	iron	2000
s4	imports, exports	iron, uranium	2000, 2001
s5	imports, exports	cereals	allYears
s6	imports, exports	iron, uranium, metals	allYears

(a) Sources

```
source(s1,1)[flow->imports,product->iron,
             time->2000].  ...
source(s2,1)[flow->exports,product->iron,
             time->2000].  ...
source(s3,1)[flow->imports,product->iron,
             time->2000].
source(s3,2)[flow->exports,product->iron,
             time->2000].
source(s4,1)[flow->imports,product->uranium,
             time->allYears].  ...
source(s5,1)[flow->imports,product->cereals,
             time->allYears].
source(s5,2)[flow->exports,product->cereals,
             time->allYears].
source(s6,1)[flow->imports,product->uranium,
             time->allYears].
source(s6,2)[flow->exports,product->uranium,
             time->allYears]. ...
```

(b) Source descriptions in Triple

Figure 4: Sample sources and descriptions

[1]Unless we included a disaggregation operation that assumed, for example, a given proportion of the cereals and the yearly production. If we assumed a uniform distribution, then the imports of wheat in 2002 would be 1/10 of the import value provided by s5 (5 years and 2 kinds of cereal).

4. MODELING THE OPERATIONS

We treat the data transformation operations as black boxes defined only by their input/output behavior. This is consistent with our use of web services as implementation devices, since web services are also described by their input/output signature. In particular, we describe the inputs and outputs of a service as objects from our ontology.

Sometimes, we can directly specify the inputs and outputs, as we did for sources (which can be seen as services with no inputs). However, in more complex cases, we use Triple programs to compute the required inputs and outputs dynamically. In particular, for aggregation operations along the dimension hierarchies, our system consults the ontology and generates the required inputs to compute a given data item dynamically for each given user request. For example, assume the user requests the total value for `allProducts`. Given the sources of Figure 4, this request cannot be answered directly. However, the system can aggregate the values for `cereals`, `iron` and `uranium` from the different sources, since these values provide an exhaustive partition of the required `allProducts`.

We focus on modeling aggregation operations in hierarchies, since these are very common in our application domain. To model these operations, we use Triple logic rules that express that when the inputs to an aggregation operation are all available, then the output of the operation will also be available. The dimension hierarchy describes which dimension values can be computed as aggregations of other values.

Figure 5 shows some of the Triple rules that our system uses to reason about the required inputs to evaluate an aggregation operation. The first rule just collects the objects that can be aggregated to compute another object. This information is readily available in the ontology. In the example, the relationship `parent` represents all those relationships with part-of semantics, like geographical containment, or the `productPartOf` of Figure 2.

The second rule states that in order to obtain an data item aggregated along one of its dimensions, the `flow` dimension in this example, we can use the `opSumFlow` operation on the children of the desired aggregated value in the dimension hierarchy (`SUM_FLOW` in this case). The rule also enforces that the system has obtained the *complete* set of inputs (all the *part* data objects) before it can produce the output of the operation. In particular, in the rule antecedent we use the logical equivalence $a \rightarrow b \equiv \neg a \vee b$ to ensure that for all elements `X` in a dimension hierarchy, if `X` is a children of the desired aggregated value `SUM_FLOW`, then there must exist a data item `DATA2` that provides such element. In addition, such data item (`DATA2`) must be compatible in the rest of the dimensions (`prod` and `time` in this case). Note that the non-aggregated dimensions variables, `PROD` and `YEAR`, are not universally quantified in the body of the rule, thus they are the same for all children data objects. If the system did not enforce completeness, the aggregation operations would be incorrect. For example, to obtain the `exports` of `metals` in `2002`, we need sum *both* the `exports` of `iron` and `uranium` in `2002`.[2]

[2]We are assuming in this that the children of an element in a hierarchy form a partition.

```
// Collect children of nodes in the
// dimension hierarchies
FORALL PARENT, CHILD
 f(set,PARENT)[element->CHILD] <-
    CHILD[argos:parent->PARENT].

// An operation can compute a data item if
// it is complete, i.e., it has all the children
// in the aggregation .
FORALL DATA, SUM_FLOW, PROD, YEAR
 DATA[flow->SUM_FLOW, prod->PROD, time->YEAR] <-
    operation(opSumFlow,SUM_FLOW,PROD,YEAR) AND
    (FORALL DATA2, X
      ((NOT (f(set,SUM_FLOW)[element->X])) OR
       (DATA2[flow->X,prod->PROD,time->YEAR]))).
```

Figure 5: Triple logic program for aggregation operations (simplified)

5. AUTOMATICALLY COMPOSING WORKFLOWS

Once we have described the application domain, the sources, and the operations, the system is ready to automatically generate computational workflows that answer specific user request by combining the available sources and operations. For our running example, we load into the Triple logic engine, the domain ontology (Figures. 1 and 2), the source descriptions (Figure. 4), and the operation programs (Figure. 5). The resulting program computes all possible workflows that answer a given user request.

Figure 6 shows a graph that encodes all the workflows that compute the total exports for all products during the period of interest (2000-2004). In the graph, operations and source accesses are shown in rectangles, and data items are shown in ovals. This graph encodes 4 alternative workflows. The first workflow is composed of three services: the sources `s5` and `s6`, and the operation `opSumProd`. It obtains `exports` of `cereals` for `allYears` from `s5` and `exports` of `metals` for `allYears` from `s6`. The second workflow is composed of 4 services: the sources `s5` and `s6`, and the two `opSumProd` operations. It obtains `exports` of `iron` and `uranium` for `allYears` from `source6`, and aggregates them using the first `opSumProd`. Then, it uses another `opSumProd` to aggregate with the `cereals` data from source `s5`, and compute the requested data. The third workflow obtains `exports` of `iron` for `allYears` by aggregating data from sources `s2` and `s4`, using the `opSumYear` operation (instead of getting such data directly from `s6`). Finally, the last workflow may choose to use `s3` for `exports` of `iron` in `2000`, and `s4` for `exports` of `iron` in `2001` (instead of getting both data items from `s4`, as in the previous workflow). Of course, each of these workflows may have different properties, including execution costs (time and monetary), reliability, etc. In this paper, we focus on modeling and generating the space of valid workflows. In our immediate future work, we plan to include a cost optimizer to choose among the alternative workflows.

Although we have focused on aggregation operations, these are representative of the kind of reasoning that the system needs to perform to identify the inputs required to produce

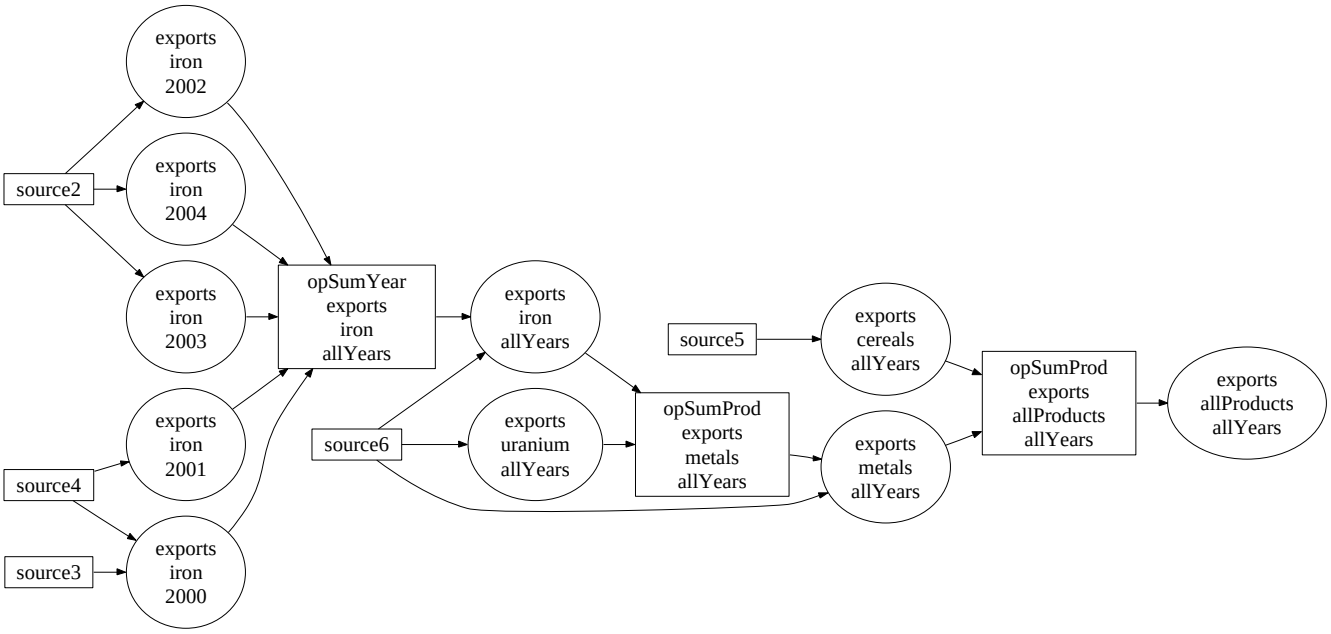


Figure 6: Sample automatically composed workflow

a desired output. Since our system has the reasoning power of the Triple logic, we are confident that we can model the operations that are likely to appear in our application domain.

6. EXECUTION ARCHITECTURE

Once a workflow has been produced by the Triple logic program, our system translates the resulting operation graph to the XML-based workflow language BPEL4WS, which can then be executed. The resulting workflow can also be made a new stand-alone web service, so that it can be deployed and reused. To do so, the system also generates the corresponding WSDL [6] specification. In our current implementation, we use IBM's BPWS4J [7] java runtime to execute BPEL4WS workflows, the Apache Tomcat java server [10], and the Axis framework [29] to deploy web services.

The following BPEL4WS specification shows the smallest workflow among those encoded in the graph of Figure 6, namely, the workflow that obtain exports of `cereals` from `source5` and exports of `metals` directly from `source6` (instead of invoking additional aggregation operations). This workflow composes three services: `source5`, `source6`, and the operation `opSumProd` (referred as `osp1` below).

```
<process name="argos"
    targetNamespace="urn:argos:process:osp1"
    xmlns:tns="urn:argos:process:osp1"
    xmlns:s5="http://localhost:8080/axis/services/S5"
    xmlns:s6="http://localhost:8080/axis/services/S6"
    xmlns:osp1="http://localhost:8080/axis/services/OSP1"
    xmlns="http://schemas.xmlsoap.org/
           ws/2003/03/business-process/">
 <variables>
  <variable name="request" messageType="tns:request"/>
  <variable name="response" messageType="tns:response"/>
  <variable name="s5in" messageType="s5:s5Request"/>
  <variable name="s5out" messageType="s5:s5Response"/>
  <variable name="s6in" messageType="s6:s6Request"/>
  <variable name="s6out" messageType="s6:s6Response"/>
  <variable name="osp1in" messageType="osp1:osp1Request"/>
  <variable name="osp1out" messageType="osp1:osp1Response"/>
 </variables>
 <partnerLinks>
  <partnerLink name="caller" partnerLinkType="tns:OSP1_PLT"/>
  <partnerLink name="source5" partnerLinkType="s5:S5"/>
  <partnerLink name="source6" partnerLinkType="s6:S6"/>
  <partnerLink name="osp1" partnerLinkType="osp1:OSP1"/>
 </partnerLinks>
 <sequence>
  <receive name="receive" partnerLink="caller"
           operation="osp1" variable="request"
           portType="tns:OSP1_PT" createInstance="yes"/>
  <flow>
   <sequence>
    <assign>
     <copy>
      <from variable="request" part="rdf_data_in"/>
      <to variable="s5in" part="rdf_data_s5_in"/>
     </copy>
    </assign>
    <invoke name="invoke" partnerLink="source5"
            operation="S5" portType="s5:S5"
            inputVariable="s5in" outputVariable="s5out"/>
   </sequence>
   <sequence>
    <assign>
     <copy>
      <from variable="request" part="rdf_data_in"/>
      <to variable="s6in" part="rdf_data_s6_in"/>
     </copy>
    </assign>
    <invoke name="invoke" partnerLink="source6"
            operation="S6" portType="s6:S6"
            inputVariable="s6in" outputVariable="s6out"/>
   </sequence>
  </flow>
  <assign>
   <copy>
    <from variable="s5out" part="S5Return"/>
    <to variable="osp1in" part="rdf_data_s5"/>
```

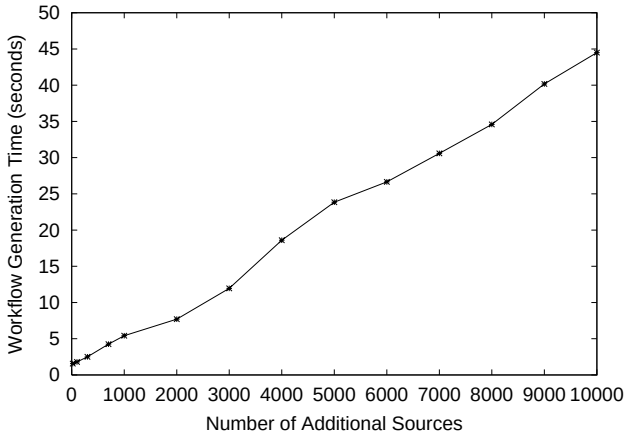
```
   </copy>
   <copy>
    <from variable="s6out" part="S6Return"/>
    <to variable="osp1in" part="rdf_data_s6"/>
   </copy>
  </assign>
  <invoke name="invoke" partnerLink="osp1"
          operation="OSP1" portType="osp1:OSP1"
          inputVariable="osp1in" outputVariable="osp1out"/>
  <assign>
   <copy>
    <from variable="osp1out" part="rdf_data_out_osp1"/>
    <to variable="response" part="rdf_data_out"/>
   </copy>
  </assign>
  <reply name="reply" partnerLink="caller"
         operation="osp1" portType="tns:OSP1_PT"
         variable="response"/>
 </sequence>
</process>
```

The following WSDL specification corresponds to input/-output behavior of the service composed according to the BPEL4WS specification above. This WSDL description is used to invoke the workflow as a stand-alone service. In both the services invoked within the BPEL4WS workflow as well as in the composed service, the data exchanged among the different services is represented in RDF-XML.[3]

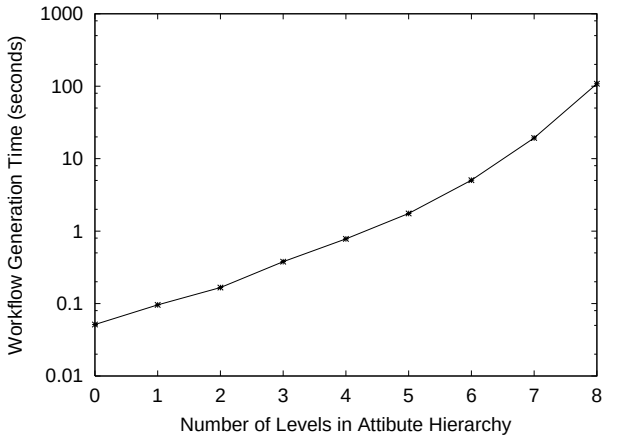
```
<definitions targetNamespace="urn:argos:process:osp1"
    xmlns:tns="urn:argos:process:osp1"
    xmlns:plnk="http://schemas.xmlsoap.org/
                ws/2003/03/partner-link/"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="request">
    <part name="rdf_data_in" type="xsd:string"/>
  </message>
  <message name="response">
    <part name="rdf_data_out" type="xsd:string"/>
  </message>
  <portType name="OSP1_PT">
    <operation name="osp1">
      <input message="tns:request"/>
      <output message="tns:response"/>
    </operation>
  </portType>
  <plnk:partnerLinkType name="OSP1_PLT">
    <plnk:role name="caller">
      <plnk:portType name="tns:OSP1_PT"/>
     </plnk:role>
  </plnk:partnerLinkType>
  <service name="OSP1">
  </service>
</definitions>
```

In [2] we demonstrated our BPEL4WS-based execution architecture. As part of the architecture, we created services that wrap instances of Triple engines. Thus, in Argos we can use Triple not only to automatically generate workflows, but also to implement RDF processing services, which are driven by declarative Triple logic programs.

[3]For flexibility, in the example WSDL, we have not strongly typed the messages. The RDF data is defined simply as strings. However, the correct types are being enforced by the Triple descriptions of the services.



(a) Scaling the number of sources



(b) Scaling the depth of the hierarchy

Figure 7: Scalability Experiments

7. EXPERIMENTAL EVALUATION

To test the scalability of the workflow generation program, we designed two experiments with synthetic domains. We run our experiments with data described in three dimensions, like the examples in Figures 3, 4, 5, and 6. Each data point is the average of five runs. We assume that the ontology, and the source and operation descriptions are already loaded into Triple, so we report the time needed to generate the workflow graph.

The first experiment tests the ability of the program to find a solution workflow in the presence of a large number of sources, out of which only a small fraction provides answers to the user request. In particular, the solution workflow contains 6 sources (with 18 data descriptors relevant to the workflow) and 6 operations, for a total of 12 services. We scale the number of additional sources up to 10000 (with 3 descriptors each, for a total of 30000 descriptors). Figure 7(a) shows that our current system is able to generate workflows with up to 10000 additional sources (30000 descriptors) in less than 45 seconds. We are encouraged by this result that our approach can scale to model a large number of sources and services.

The second experiment tests how the growth of the depth of the hierarchy affects the workflow generation. In this experiment, we ask for the top level object of a given dimension, but only the leaves of the hierarchy are available at the sources. We test on complete binary trees. So for a tree of depth n, there are 2^n leaves and a total of $2^{n+1} - 1$ nodes. Our current modeling of the aggregation operations will compute all intermediate data items along the hierarchy, so for a depth n the workflow will include $2^{n+1} - 1$ operations. Figure 7(b) shows that the system takes up to 108 seconds to compute a workflow with 256 sources and 511 operations. By design, this experiment computes exponentially larger workflows, so the shape of Figure 7(b) is not a surprise. Nevertheless, we are currently investigating ways to improve the workflow generation time, including using a different logic specification of the problem, and optimizing Triple or XSB [25], the logic engine on top of which Triple is implemented. One option is to directly aggregate the data at the *leaves* of a workflow graph, instead of computing all the intermediate results. This saves an exponential amount of work in the worst case.

8. RELATED WORK

The present work is related to research on mediator systems, such as the Information Manifold [17], InfoMaster [8], Ariadne [14], and our previous work on the Energy Data Collection project [1]. We differ in that our schema (ontology) focuses on partonomies and on more complex operations, such as aggregation.

There has been a significant interest in composing web service workflows in the scientific community, in particular for computationally-intensive applications physics (e.g. [30, 13]) and bioinformatics (e.g. [18]) running on the grid [9]. Much of this work focuses on the performance of the computational problems and on mixed-initiative composition approaches. The myGrid project [28] is an ambitious data grid services projects in bioinformatics. The semantic descriptions of web services in myGrid are used for discovery purposes, but the composition of services is performed manually with the help of user interfaces, such as the Taverna workbench [24]. Similarly, Kim et al. [13] present a mixed initiative approach for composing web services in the domain of earthquake science. In contrast, we focus on automatic composition.

There has been research on automatic web service composition within the AI planning community. Blythe et al. [4] developed a planner to compose executable grid workflows, which has been applied to physics problems. Sirin et al. [27] describe a system to automatically compose web services based on the hierarchical task network planner SHOP2 [22]. In addition to the input and output constraints, these systems can also handle web services with preconditions and effects. McIlraith and Son [21] present an approach to composition based on the logical action language Golog. They rely on Golog templates to compose the different web services. While this representation is powerful and can handle web services with preconditions and effects, their system requires a human to write different plan templates before the system can answer user queries. These works do not consider aggregation operations.

9. DISCUSSION

We have described work towards an approach to automatically generate computational workflows for transportation modeling problems. We rely on an ontology of the application domain to provide formal semantics to the sources and operations available. We describe the ontology, the sources, and operations as a Triple logic program. This program generates workflows to answer user requests. We have focused on workflows with aggregation operations. We represent all data sources and operations as web services, and our workflows are compositions of web services. We use the BPEL4WS standard and IBM's BPWS4J engine to describe and execute, respectively, the web services compositions.

Our immediate future work is to model the different types of operations that we encounter in our application domain to ensure that we can automatically compose workflows that answer typical data requests. We also want to include a cost-optimizer for workflow execution. As we noted earlier, our current logic program generates all possible workflows. The system should select the most cost-effective workflow for execution. Finally, as a result of this work we expect to provide a tool that transportation practitioners can use to gather and analyze data more efficiently.

10. ACKNOWLEDGMENTS

We would like to thank the current and former members of the Argos group for their comments and insights, including our social scientists: Genevieve Giuliano, Peter Gordon, Qisheng Pan, LanLan Wang, and Paul Pender; computer scientists: Stefan Decker, Andreas Harth, Naqeeb Abbasi, and Karanbir Jassar; as well as our government partner agencies.

This material is based upon work supported by the National Science Foundation under Award No. EIA-0306905. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation.

11. REFERENCES

[1] J. L. Ambite, Y. Arens, E. Hovy, A. Philpot, L. Gravano, V. Hatzivassiloglou, and J. Klavans. Simplifying data access: The energy data collection (EDC) project. *IEEE Computer, Special Issue on Digital Government*, 34(2), February 2001.

[2] J. L. Ambite, G. Giuliano, P. Gordon, A. Harth, K. Jassar, Q. Pan, and L. Wang. Argos: An ontology and web service composition infrastructure for goods movement analysis. In *Proceedings of the 5th Annual National Conference on Digital Government Research (dg.o2004)*, Seattle, Washington, 2004. System Demonstration.

[3] T. Andrews, F. Curbera, H. Dholakia, Y. Goland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services, version 1.1. http://www-106.ibm.com/developerworks-/library/ws-bpel, 2003.

[4] J. Blythe, E. Deelman, and Y. Gil. Automatically composed workflows for grid environments. *IEEE Intelligent Systems*, pages 16–23, July/August 2004.

[5] D. Brickley and R. Guha. RDF vocabulary description language 1.0: RDF schema W3C recommendation 10 february 2004. http://www.w3.org/TR/rdf-schema/, 2004.

[6] R. Chinnici, M. Gudgin, J.-J. Moreau, J. Schlimmer, and S. Weerawarana. Web services description language WSDL, version 2.0, part 1: Core language, W3C Working Draft, 3 august 2004. http://www.w3.org/TR/wsdl20, 2004.

[7] F. Curbera, M. J. Duftler, R. Khalaf, N. Mukhi, W. A. Nagy, and S. Weerawarana. IBM business process execution language for web services java run time, version 2.1. http://www.alphaworks.ibm.com/tech/bpws4j, 2004.

[8] O. M. Duschka and M. R. Genesereth. Answering recursive queries using views. In *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Tucson, Arizona, May 1997.

[9] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, 2002.

[10] T. A. Foundation. Jakarta Tomcat. http://jakarta.apache.org/tomcat/, 2004.

[11] A. Y. Halevy. Answering queries using views: A survey. *The VLDB Journal*, 10(4):270–294, 2001.

[12] M. Kifer, G. Lausen, and J. Wu. Logical foundations of object-oriented and frame-based languages. *Journal of ACM*, 42:741–843, July 1995.

[13] J. Kim, M. Spraragen, and Y. Gil. An intelligent assistant for interactive workflow composition. In *International Conference on Intelligent User Interfaces*, Madeira, Portugal, 2004.

[14] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. G. Philpot, and S. Tejada. The Ariadne approach to web-based information integration. *International Journal of Cooperative Information Systems (IJCIS) Special Issue on Intelligent Information Agents: Theory and Applications*, 10(1-2):145–169, 2001.

[15] D. Lenat and R. Guha. *Building Large Knowledge-Based Systems: Representation and Inference in the Cyc Project*. Addison-Wesley, Reading, Massachusetts, 1990.

[16] A. Y. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Logic-Based Artificial Intelligence*. Kluwer Publishers, November 2000.

[17] A. Y. Levy, D. Srivastava, and T. Kirk. Data model and query evaluation in global information systems. *Journal of Intelligent Information Systems, Special Issue on Networked Information Discovery and Retrieval*, 5(2):121–143, 1995.

[18] P. Lord, S. Bechhofer, M. D. Wilkinson, G. Schiltz, D. Gessler, D. Hull, C. Goble, and L. Stein. Applying semantic web services to bioinformatics: Experiences gained, lessons learnt. In *International Semantic Web Conference (ISWC)*, Hiroshima, Japan, 2004.

[19] F. Manola, E. Miller, and B. McBride. RDF primer, W3C recommendation, 10 february 2004. http://www.w3.org/TR/rdf-primer/, 2004.

[20] D. L. McGuinness and F. van Harmelen. OWL web ontology language, W3C recommendation, 10 february 2004. http://www.w3.org/TR/owl-features/, 2004.

[21] S. McIlraith and T. C. Son. Adapting Golog for composition of semantic web services. In *Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002)*, pages 482–493, 2002.

[22] D. S. Nau, H. Muñoz-Avila, Y. Cao, A. Lotem, and S. Mitchell. Total-order planning with partially ordered subtasks. In *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence*, pages 425–430, 2001.

[23] N. F. Noy, M. Sintek, S. Decker, M. Crubzy, R. W. Fergerson, and M. A. Musen. Creating semantic web contents with protege-2000. *IEEE Intelligent Systems*, 16(2):60–71, 2001.

[24] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, 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.

[25] K. Sagonas, T. Swift, and D. S. Warren. XSB as an efficient deductive database engine. pages 442–453, 1994.

[26] M. Sintek and S. Decker. TRIPLE: A query, inference, and transformation language for the semantic web. In *International Semantic Web Conference (ISWC)*, Sardinia, June 2002.

[27] E. Sirin, B. Parsia, D. Wu, J. Hendler, and D. Nau. HTN planning for web service composition using SHOP2. *Web Semantics*, 1(4):377–396, 2004.

[28] R. D. Stevens, A. J. Robinson, and C. A. Goble. myGrid: personalised bioinformatics on the information grid. *Bioinformatics*, 19(1):302–304, 2003.

[29] The Apache Foundation. Axis. http://ws.apache.org/axis/, 2004.

[30] Y. Zhao, M. Wilde, I. Foster, J. Voeckler, T. Jordan, E. Quigg, and J. Dobson. Grid middleware services for virtual data discovery. *submitted to Concurrency and Computation: Practice and Experience*.