A Novel Domain Oriented Approach for Scientific Grid Workflow Composition

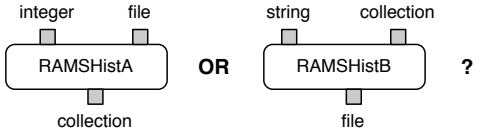
Jun Qin and Thomas Fahringer
Institute of Computer Science, University of Innsbruck
Technikerstr. 21a, A-6020 Innsbruck, Austria
Email: {jerry, tf}@dps.uibk.ac.at

***Abstract*—Existing knowledge based Grid workflow languages and composition tools require sophisticated expertise of domain scientists in order to automate the process of managing workflows and its components (activities). So far semantic workflow specification and management has not been addressed from a general and integrated perspective. This paper presents a novel domain oriented approach which features separations of concerns between data meaning and data representation and between activity function (semantic description of workflow activities) and activity type (syntactic description of workflow activities). These separations are implemented as part of Abstract Grid Workflow Language (AGWL) which supports the development of Grid workflows at a high level (semantic) of abstraction. The corresponding workflow composition tool simplifies Grid workflow composition by (i) enabling users to compose Grid workflows at the level of data meaning and activity function that shields the complexity of the Grid, any specific implementation technology (e.g. Web or Grid service) and any specific data representation, (ii) semi-automatic data flow composition, and (iii) automatic data conversions. We have implemented our approach as part of the ASKALON Grid application development and computing environment. We demonstrate the effectiveness of our approach by applying it to a real world meteorology workflow application and report some preliminary results. Our approach can also be adapted to other scientific domains by developing the corresponding ontologies for those domains.**

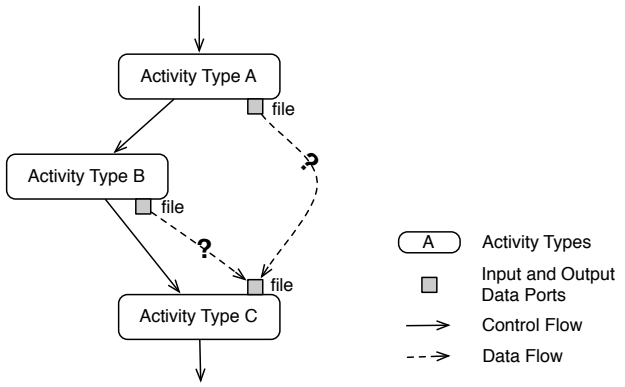
I. Introduction

With the development of Grid technologies, scientists and engineers are building more and more complex applications to execute scientific experiments on distributed Grid resources. Grid workflows, as a main programming paradigm for addressing large scale e-science problems, play a paramount important role in this process.

A Grid workflow application can be seen as a collection of computational tasks (activities) that are processed in a well-defined order to accomplish a specific goal. Numerous Grid workflow languages and tools [1], [2], [3], [4], [5], [6] are developed to facilitate Grid workflow composition and execution. However, Grid workflow composition is still a challenging task for domain scientists. To simplify Grid workflow composition, some Grid workflow development environments [1], [5], [7], [6], [8] enable users to compose Grid workflow applications at a high level of abstraction through using activity types (an abstract description of a group of concrete implementations of computational entities deployed in the Grid, which share the same functionality and the same input and output data structure). Some Grid workflow development environments [2], [9], [5], [8], [10] adopted a knowledge based process to help users build Grid workflows. However, there are still two problems with existing approaches: (i) users still have to understand the data type systems used in Grid workflow runtime environments to select correct activity types for their workflows, and (ii) how to specify data flow among workflow activities, especially in a workflow where many data ports have the same data type (e.g. `string`, `file`).



(a) two activity types having the same functionality but different input and output data structures



(b) data flow composition among three data ports which have the same data type `file`

Fig. 1. Two Grid Workflow Composition Problems

As an illustrating example, let us consider a real world meteorology Grid workflow, where users are offered two activity types: *RAMSHistA* and *RAMSHistB* (Fig. 1(a)). Both activity types represent the same functionality, that is, to evolve the physical representation of the atmosphere from an earlier RAMS model to the specified final simulation time. The activity type *RAMSHistA* requires an input integer in the

SC2008 November 2008, Austin, Texas, USA 978-1-4244-2835-9/08 $25.00 ©2008 IEEE

range [1–12] indicating *Month* and a zipped input file indicating *SeaSurface*, and produces *RAMSModeledAtmosphere* as a collection of files. The activity type *RAMSHistB* requires and produces similar kind of data, except that the input data *Month* must be a string with one value from the set {Jan, Feb, ..., Dec}, the input data *SeaSurface* must be a collection of files, and the output data *RAMSModeledAtmosphere* is a *.tar.gz* file. All the other data ports of both activity types are omitted for the reason of simplicity. In this case, users have to choose between *RAMSHistA* and *RAMSHistB* for their workflow based on the data types of the data produced by predecessor activity types in the workflow. Once one of the two activity types is selected, users also have to make decision on the successor activity types in the workflow so that the successor activity types can correctly process the data produced by the previously selected activity type.

When all activity types are selected and connected with control flow, data flow composition is another complex tasks for domain users, as illustrated in Fig. 1(b) which shows part of a workflow. In this workflow, both activity type *A* and activity type *B* produce a file (indicated by the data type `file`) and activity type *C* requires a file as input. The problem here is where the input file of activity type *C* should come from, from the output of activity type *A* or from the output of activity type *B* (as illustrated by two data flow connections with question marks in Fig. 1(b)). The data flow composition could be even worse in a real world Grid workflow application because (i) there may be ten to hundred data ports having the same data type and are available for a successor activity type, and (ii) the closer the successor activity type is positioned to the end of the workflow, the more data ports are available. The process to determine which data ports have to be connected by data flow is error prone and requires solid expertise of domain users.

In this paper, we present a novel domain oriented approach to solve the two aforementioned problems as part of Abstract Grid Workflow Language (AGWL) [11] which is a widely known language for describing Grid workflow at a high level of abstraction. We developed a generic AGWL ontology which mainly consists of three top level concepts: *Function*, *Data* and *DataRepresentation*. The concept *Function* and *Data* can be extended by domain specific ontologies such as the meteorology ontologies illustrated in this paper. With the help of these ontologies, we separate between data meaning and data representation and between activity function and activity type in AGWL. These separations allow users to compose workflows at the level of data meaning and activity function and leave the task of dealing with data representations and activity types to the Grid workflow composition tool (to solve the first problem). These separations also enable semi-automatic data flow composition based on the semantics of the data in the workflow (to solve the second problem). We have implemented the prototype of this approach in the ASKALON Grid environment [6]. Experiments demonstrate that the overhead of our approach is insignificant and about 90% of the data flows for reasonable complex Grid workflow applications can be established automatically.

The contributions of this paper are as follows:

- We introduce the concept of data representation in Grid workflows, and some comprehensive properties associated with the concept.
- We separate concerns between data meaning and data representation and between activity function and activity type, which makes AGWL more abstract and makes the Grid workflow composition tool easier to use for domain scientists.
- With the help of the generic AGWL ontology and the extended domain ontologies, Grid workflows in three different levels (semantic, syntactic and concrete) of abstraction can be described with one single Grid workflow language.
- We applied our approach to a real world meteorology Grid workflow application.

The remainder of this paper is organized as follows. A brief overview of AGWL is provided in Section II. Section III introduces the structure of the generic AGWL ontology. Section IV presents how Grid workflow activities are described based on the ontology introduced in the previous section and how an activity function based semantic Grid workflow representation can be mapped to the corresponding syntactic representation. We discuss the prototype implementation of our approach in Section V and report some preliminary results in Section VI. Section VII compares important related work against our approach. The paper ends with a short conclusion and an outline of the future work.

II. Overview of AGWL

AGWL [11] is an XML-based language for describing Grid workflow applications at a high level of abstraction. It is the main interface to the ASKALON [6] Grid application development and computing environment and has been applied to numerous real world Grid workflow applications. AGWL has been designed such that users can concentrate on specifying Grid workflow applications without dealing with either the complexity of the Grid or any specific implementation technology (e.g. Web or Grid Service, software components or Java classes, etc.). AGWL allows a programmer to define a graph of activities that refer to computational tasks or user interactions. AGWL workflows consist of activities, control flow constructs, data flow links and properties and constraints.

An AGWL activity can be an *atomic activity* or a *compound activity*. An atomic AGWL activity is represented by an *Activity Type (AT)* with input and output data ports. An AT is an abstract description of a group of *Activity Deployments (ADs)* (concrete implementations of computational entities deployed in the Grid) which have the same functionality and the same input and output data structure, but probably different performance behaviors, QoS characteristics and costs. Each input and output data port of an AT is described by a data port name and a data type such as `xsd:string` or `agwl:file`. The number and the types of the input and output data ports of an atomic activity are determined by the associated AT. ATs

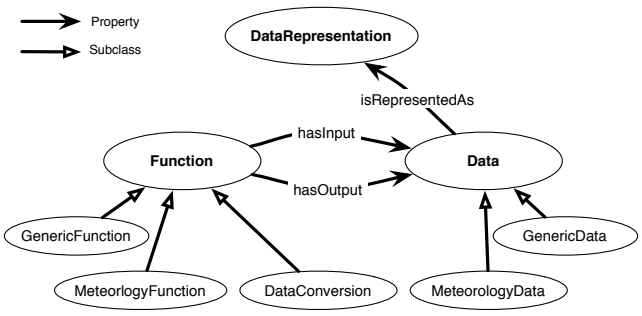


Fig. 2. The Upper Ontology

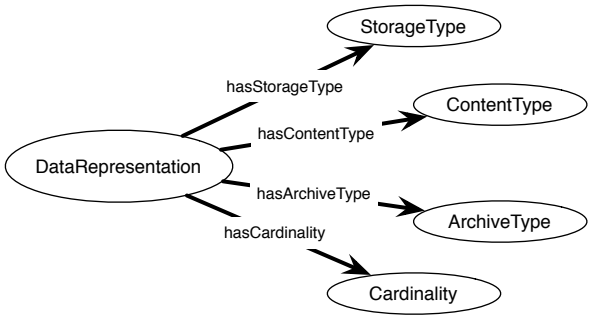


Fig. 3. The Ontology of DataRepresentation

shield the implementation details of ADs from the AGWL programmer.

A rich set of control flow constructs (compound activities) has been provided in AGWL to simplify the specification of Grid workflow applications such as `sequence`, `parallel`, `if`, `switch`, `while`, `doWhile`, `for`, `forEach`, `parallelFor` and `parallelForEach` with semantics similar to comparable constructs in high-level programming languages. The `dag` construct is also provided for describing a directed acyclic graph of activities. AGWL also supports sub-workflows, which are similar to procedures in high-level programming languages. Sub-workflows are used to modularize, encapsulate and reuse code regions.

The data flow in AGWL is expressed by data flow links from source data ports to sink data ports. This is done by set the values of the property `source` of sink data ports to the corresponding source data ports. For sink data ports, source data ports are called *data sources*.

Properties and constraints can be defined in AGWL to provide additional information for workflow runtime systems to optimize and steer the executions of workflow applications. Properties provide hints about the behavior of activities and constraints should be complied with by the underlying workflow runtime system. The user can specify properties and constraints for both activities and data ports.

III. Ontologies

Based on our experiences in collaborating with scientists from multiple domains, such as meteorology, material science and astrophysics [12], we have identified some questions raised by users when they compose Grid workflows. Among others, the following questions are frequently asked:

- Which activity provides a solution for a requested task, e.g. to evolve the physical representation of the atmosphere?
- Which activity produces or consumes given data, e.g. SeaSurface, LandCover?

To answer these questions, we have developed an upper ontology which is shown in Fig. 2. It is motivated by the need to ease Grid workflow composition. There are three main concepts in the upper ontology: *Function*, *Data* and *DataRepresentation* which are described as follows.

Data is a superclass of any kind of workflow data. It provides a high level description of workflow data, that is, the meaning of workflow data. *Data* can be extended by a *GenericData* (e.g. Month). *Data* can also be extended for domain specific data. For example, *MeteorologyData* can be defined as a subclass of *Data* and a superclass of all kinds of meteorological data. *Data* is the input to or the output of *Function*. Users can search *Functions* based on input and output *Data*.

Function is a superclass of both generic and domain specific functions which are implemented by Web or Grid services, executables, Java classes, etc. *Function* has properties *hasInput* and *hasOutput* which indicate input and output data of *Function*. The values of the properties *hasInput* and *hasOutput* are *Data*. Beside these two properties, *Function* also has a *hasDescription* property, whose value is a textual description on what exact functionality it represents. This enables domain users to search *Functions* based on free text. For example, a user may input "atmosphere evolution" to find *Functions* which can simulate the evolution of the atmosphere. We adopted the *string-matching* algorithm presented by Cardoso et al. in [13] to match free text against the descriptions of *Functions* in our approach. To follow the IOPE (Input, Output, Precondition and Effect, i.e., four properties to semantically describe capabilities of Web Services) [14] pattern of the modern semantic technologies, we also associated the properties *hasPrecondition* and *hasEffect* with *Function*. However, we defer the *Precondition* and *Effect* related research to future work since we do not observe much requirement on this from the scientific domains we are involved. *Function* can also be extended by domain specific function concepts such as *MeteorologyFunction*. Specific meteorological functions can then be defined by subclassing *MeteorologyFunction*. The subclass *DataConversion* of the class *Function* in Fig. 2 will be explained in Section IV-B.

DataRepresentation describes storage related information of *Data*, that is, how data is stored in computer systems. While *Data* specifies the meaning (semantic information) of workflow data that is easy to understand by human beings, *DataRepresentation* indicates additional storage related information (syntactic information) of workflow data that is required by services or executables to correctly process the

TABLE I
THE PROPERTIES OF DATAREPRESENTATION

Property	Description	Possible Value
hasStorageType	Where the *Data* is stored	Memory, FileSystem, Database
hasContentType	Which format the *Data* is stored in	XML Schema Datatypes, File Types
hasArchiveType	Whether and how the *Data* is archived	zip, gzip, bz2, tar, rar, none, etc.
hasCardinality	What is the cardinality of the *Data*	single, multiple

data. We define some comprehensive properties for the concept *DataRepresentation* which are shown in Fig. 3. Table I shows the descriptions and the possible values of these properties. While the property *hasStorageType* is easy to understand, the property *hasContentType* indicates that the format in which *Data* is stored. Its value indicates data types, including simple or complex data types supported by the XML Schema Datatype specification, and file types such as BinaryFile, TextFile, CSVFile (a subclass of TextFile), XMLFile (a subclass of TextFile), CDFFile (a scientific data format that can include scalar, vector, and multi-dimensional data arrays, a subclass of BinaryFile), etc. The value *none* of the property *hasArchiveType* indicates that *Data* is not archived. This could be the case of unarchived files, or data stored in memory like strings or integers. The property *hasCardinality* describes whether *Data* is a single data element or a collection of data elements. Obviously, *DataRepresentation* is a broader concept than data types and provides richer information required by scientific applications than the latter. The same exact data can have alternative *DataRepresentations*, such as a string stored in memory, a record in a database table, or a set of file each structured either as a table, an XML structure, or a labeled list (Gil [15] has recently proposed similar ideas). Note that users do not have to specify the data representations of workflow data while composing a Grid workflow. The *DataRepresentation* information comes from the deployed ATs. The data coversions between different data representations of *Data* are done automatically by the Grid workflow composition tool when a semantic Grid workflow representation is mapped to the corresponding syntactic representation. This is explained in Section IV.

The main difference between our approach and existing semantic Grid approaches is that we do not associate data types (e.g. string, integer, etc.) directly with *Data* itself. Instead, we introduce the idea that *Data* is represented as *DataRepresentations*, which include the corresponding data type information, as well as some other storage related information. The separation of concerns between data meaning (*Data*) and data representation (*DataRepresentation*) enables domain scientists to semantically compose Grid workflow applications without dealing with specific data representations including data types. By associating data meaning with data ports, semi-automatic data flow composition is also supported.

```
1  <activity function="RAMSHist">
2    <dataIns>
3      <dataIn name="m" data="Month"/>
4      <dataIn name="in" data="SeaSurface"/>
5      <!-- other input data -->
6    </dataIns>
7    <dataOuts>
8      <dataOut name="out" data="RAMSModeledAtmosphere"/>
9      <!-- other output data -->
10   </dataOuts>
11 </activity>
```

Fig. 4. The Activity Function *RAMSHist*

We will refer to the generic AGWL ontology and the extended domain specific ontologies as the AGWO ontology in the remainder of this paper.

IV. ONTOLOGY BASED GRID WORKFLOW REPRESENTATION

In this section, we explain how the AGWO ontology is used to describe Grid workflow activities and how a semantic Grid workflow representation can be mapped to the corresponding syntactic representation.

A. Activity Function

Activity Function (AF) is an abstract concept to describe workflow activities at the semantics level. It describes input and output data with data meaning. An AF is formally defined as follows:

$$AF = \{F, I, O\}$$

where F indicates the functionality, I is an ordered set of input data meanings, and O is an ordered set of output data meanings. The functionality is a subclass of *Function* defined in the AGWO ontology. The input and output data meanings are subclasses of *Data* in the AGWO ontology. No data representation is defined in an AF. For example, in the meteorology domain, an AF *RAMSHist*, to evolve the physical representation of the atmosphere from an earlier RAMS model to the specified final simulation time, is shown in Fig. 4. The functionality is specified through the attribute *function* at line 1. The data meaning of each data port is specified through the attribute *data* at line 3, line 4, and line 8, where *SeaSurface* indicates water body properties, e.g. temperature, ice cover, etc. and *RAMSModeledAtmosphere* is the forecasted atmosphere based on an RAMS model. AFs can be extracted from the AGWO ontology because the ontology specifies knowledge about which *Function hasInput* or *hasOutput* which *Data*.

B. Activity Type

We also extend the concept of AT in AGWL. An AT describes a workflow activity at the syntax level. An AT is specified as:

$$AT = \{F, T, I, R_I, O, R_O\}$$

where F, I, and O are same as they are defined in an AF, T indicates the name of the specific activity type, R_I is an

ordered set of data representations of input data I, and R_O is an ordered set of data representations of output data O. An AT is associated with an AF through the functionality of the AT, i.e. $AT.F$. If $AT.F = AF.F$, then $AT.I \cong AF.I \wedge AT.O \cong AF.O$, where the relation $\cong$ indicates *semantically compatible*. Given two ordered set S and T each having n elements, $S \cong T$ is defined as follows:

$$S \cong T = \begin{cases} true & i \in [1,n] : S_i = T_i \vee S_i \text{ is a subclass} \\ & \text{of } T_i \text{ based on the AGWO ontology} \\ false & \text{otherwise} \end{cases}$$

Multiple ATs with semantically compatible input and output data but different data representations can be associated with one AF. If one wants to define an AT to be associated with an AF but with non-semantically compatible input and output data, then a subclass of the corresponding functionality should be defined in the AGWO ontology and the AT should be associated with the AF extracted from the ontology based on the subclass. Fig. 5 illustrates an AT which is associated with the AF illustrated in Fig. 4. Besides containing the same information as the AF, the AT also specifies the name of the specific activity type (line 1) and the data representation for each data port (lines 4 – 9, lines 12 – 17 and lines 23 – 28).

ATs are deployed in an AT repository which is a registry service in the Grid workflow runtime environment. A special kind of ATs called *data conversion ATs* are also deployed in the AT repository. For *data conversion ATs*, $AT.F = DataConversion$ (see Fig. 2). A *data conversion AT* can convert *Data* from one representation to another. For example, there can be a *data conversion AT* which converts *Month* from a string in the set {Jan, Feb, ..., Dec} to an integer in the range [1–12].

Instead of using ATs, users compose Grid workflows using AFs with their domain knowledge only. The Grid workflow composition tool can then help establish data flow among AFs semi-automatically based on the semantics of data ports, and convert between different data representations of workflow data automatically if necessary. Let us refer to AF based Grid workflow representations as semantic Grid workflow representations and to AT based Grid workflow representations as syntactic Grid workflow representations. Semantic Grid workflow representations are not executable and have to be mapped to the corresponding syntactic representation before they are submitted for scheduling and execution. This is explained in the following section.

C. From Semantics to Syntax

To map a semantic Grid workflow representation to the corresponding syntactic representation, the Grid workflow composition tool starts from the initial node of the graphical representation (e.g. the InitialNode of the UML 2 Activity Diagram [16], as used in ASKALON [6]) of the Grid workflow and maps all AFs to ATs while following control flow, until all AFs are mapped to ATs. The reason that the mapping process starts from the initial node is that only when the predecessor AFs are mapped to ATs, are the data representations of *data sources* of input data ports of successor AFs determined. Those data representations can then be used to map AF to ATs. Let us refer to the data representations of *data sources* of the input data ports of an AF as the *input source data representations* of the AF. When mapping AFs to ATs, only the *input source data representations* of AFs is necessary to be compared with the input data representations of ATs. No output data representations of ATs need to be considered.

```
1  <activity function="RAMSHist" type="rams_hist_c">
2    <dataIns>
3      <dataIn name="m" data="Month">
4        <dataRepresentation>
5          <storageType>Memory</storageType>
6          <contentType>xsd:string</contentType>
7          <archiveType>none</archiveType>
8          <cardinality>one</cardinality>
9        </dataRepresentation>
10     </dataIn>
11     <dataIn name="in" data="SeaSurface">
12       <dataRepresentation>
13         <storageType>FileSystem</storageType>
14         <contentType>agwl:file</contentType>
15         <archiveType>none</archiveType>
16         <cardinality>one</cardinality>
17       </dataRepresentation>
18     </dataIn>
19     <!-- other input data -->
20   </dataIns>
21   <dataOuts>
22     <dataOut name="out" data="RAMSModeledAtmosphere">
23       <dataRepresentation>
24         <storageType>FileSystem</storageType>
25         <contentType>agwl:file</contentType>
26         <archiveType>zip</archiveType>
27         <cardinality>multiple</cardinality>
28       </dataRepresentation>
29     </dataOut>
30     <!-- other output data -->
31   </dataOuts>
32 </activity>
```

Fig. 5. The Activity Type *rams_hist_c*

The algorithm of mapping an AF against ATs in the AT repository is illustrated in Fig. 6. First, a set of ATs (Ω_{sub}) associated with the AF are retrieved from the AT repository (line 3). Then for each AT in the set, the set of input data representations of the AT ($AT.R_I$) are compared with the set of *input source data representations* (R_{src}) of the AF (line 4-9). If $AT.R_I$ is semantically compatible with R_{src}, then the AF is replaced with the AT directly. Let us denote this kind of match by *direct match*. If multiple such ATs exist (in case that duplication is allowed in the AT repository, or the AT repository is distributed with cache enabled), then any one of these ATs (the first found AT is returned in the algorithm) can be selected because they are considered to be the same one. Note that ATs are abstract descriptions of workflow activities and no performance or cost criteria can be applied here on the selection of ATs. If no direct match found, then for each AT in the set, all *data conversion ATs* are checked to find a set of *data conversion ATs* which can fulfill the data conversion requirements from R_{src} to $AT.R_I$ (line 10-15). If such an AT and the corresponding *data conversion ATs* exist, then the AF

1: **Input**:
   AF: the AF to be mapped to ATs;
   R_{src}: the ordered set of the *input source data representations* of AF;
   Ω: set of ATs in the AT repository
2: **Output**: None $\{AF$ *is replaced with the found ATs*$\}$
3: $\Omega_{sub} := \{AT | AT \in \Omega \wedge AT.F = AF.F\}$ $\{$*a set of ATs associated with the* $AF\}$
   $\{$*find direct match*$\}$
4: **for all** $AT \in \Omega_{sub}$ **do**
5: **if** $(AT.R_I \cong R_{src})$ **then** $\{$*see Section IV-B*$\}$
6: replace AF with AT
7: **return**
8: **end if**
9: **end for**
   $\{$*no direct match found, find indirect match now*$\}$
10: **for all** $AT \in \Omega_{sub}$ **do**
11: **if** $(\exists C \mid canConvert(C, AF.I, R_{src}, AT.R_I)\ )$ **then**
12: replace AF with the *data conversion ATs* $\in C$ followed by the AT.
13: **return**
14: **end if**
15: **end for**
16: report an error to users
17: **return**

Fig. 6. Mapping an AF to ATs

is replaced by the corresponding *data conversion ATs* followed by the AT. Let us denote this kind of match by *indirect match*. Fig. 7 illustrates two scenarios of indirect match: (i) mapping from (a) to (b) indicates the case where only one pair of data representations are not semantically compatible and need to be converted; and (ii) mapping from (a) to (c) indicates the case where more than one pair of data representations need to be converted. *DC1*, *DC2*, *DC3*, and *DC4* in the figure are *data conversion ATs*. A `parallel` compound activity (see Section II) is used in Fig. 7(c) to enable parallel execution of data conversions. In case that an AF can not be mapped due to unavailable ATs, an error is reported to users (line 16). Then users can either choose another AF to use in the workflow or ask for deploying more ATs. The function *canConvert* used in the algorithm is defined as follows, where S, T, C and D are four ordered sets each having n elements.

$$canConvert(C, D, S, T) = \begin{cases} true & i \in [1, n] : S_i = T_i \\ & \vee C_i \text{ can convert } D_i \\ & \text{from } S_i \text{ to } T_i \\ false & \text{otherwise} \end{cases}$$

Once the semantic Grid workflow representation is converted to the corresponding syntactic representation, the workflow can be scheduled and executed. During this process, based on the scheduling algorithm, the Grid site, the specific service or executable on that Grid site and some other concrete information (e.g., working directory, executable usage, input

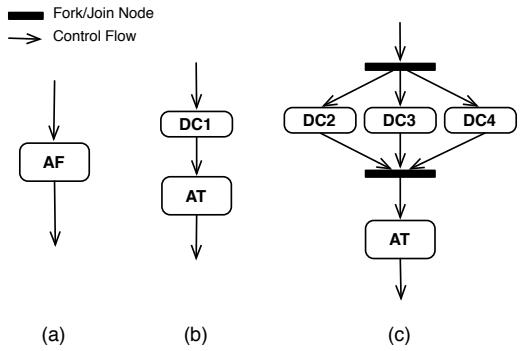


Fig. 7. Mapping from an AF to ATs: indirect match

file name patterns) can be determined for each AT. These concrete information are saved into the `constraints` of activities (see Section II). At this point, an AT becomes an *Activity Deployment (AD)*. From this point of view, we can say that we describe Grid workflows at three different levels (semantic, syntactic, and concrete) of abstraction with a single Grid workflow language, AGWL.

V. Implementation

We have extended the generic AGWL ontology with the ontologies for the meteorology simulation domain. The aim is to ease composition of workflows in the numerical model based meteorology simulation domain. We designed the ontology using the Protege-OWL editor [17] and implement a prototype of our approach using the Jena APIs [18]. Fig. 8 shows the main *Data* classes of the meteorology ontologies, where all meteorological data classes are defined as subclasses of the class *MeteorologyData*. Fig. 9 shows the main *Function* classes, where all meteorological functionalities are defined as subclasses of the class *MeteorologyFunction*. Some classes are hidden in the two figures (indicated by small triangles) for the reason of simplicity. The hierarchy of these classes is visualized by the OWLViz plugin for Protege.

These defined *Data* and *Function* classes are used to guide the composition of the Grid workflow MeteoAG [19]. MeteoAG is a Grid workflow application for meteorology simulations based on the numerical model RAMS [20]. The simulations produce precipitation fields of heavy precipitation cases over the western part of Austria, at a spatial and temporal grid, in order to resolve most alpine watersheds and thunderstorms. Fig. 10 illustrates the UML 2 Activity Diagram representation of the workflow which consists of 8 AFs connected with several control flow constructs, e.g. `parallelForEach`, `parallelFor` and `If`.

The prototype of our approach is implemented under the ASKALON Grid environment. The architecture of the prototype is shown in Fig. 11. When the ASKALON Grid workflow composition GUI starts, the ontologies are loaded from a shared URL and the AFs are extracted and saved in the memory of the GUI. Then users can search AFs based on free text or input and output data and construct a Grid workflow.

Fig. 8. The Main Data Classes of the Meteorology Ontologies

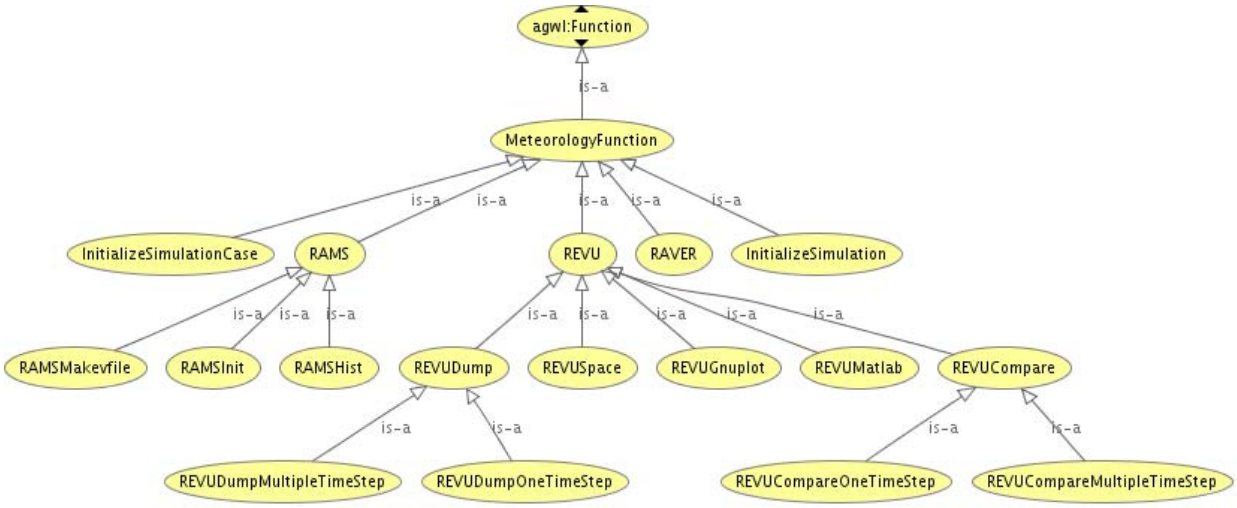


Fig. 9. The Main Function Classes of the Meteorology Ontologies

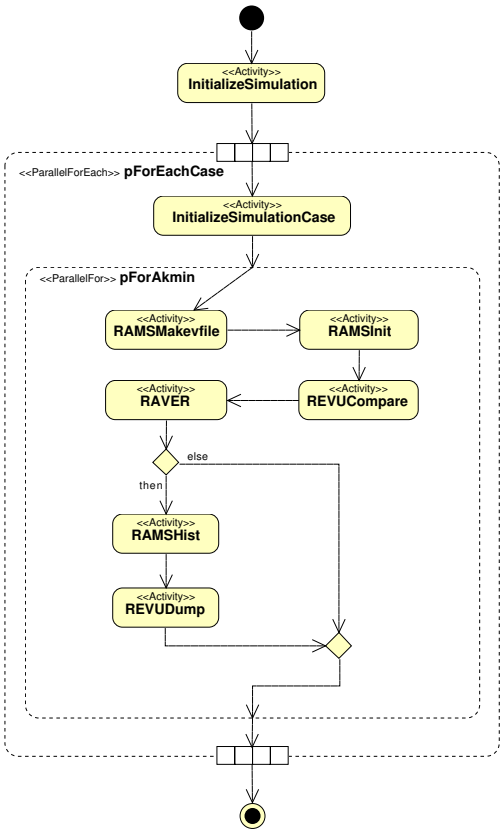


Fig. 10. The MeteoAG Workflow

For each AF put in the workflow, the *Workflow Analyzer* analyzes the control flow dependencies and provides a list of available *data sources* for this AF. Based on the semantics of input and output data of AFs, the data flows are established semi-automatically. For the data ports which have more than one semantically matching *data sources*, user interaction is required. Once the Grid workflow is constructed, the *Workflow Mapper* maps the semantic Grid workflow representation to the corresponding syntactic representation by consulting the AT repository. When the mapping is done, the syntactic Grid workflow representation is ready to be submitted to the ASKALON runtime system for scheduling and execution.

In the prototype, the AT repository runs locally in the same JVM as the ASKALON Grid workflow composition tool. The AT repository loads ATs from the local file system and stores ATs in a hash map by using the functions of ATs as the keys, in order to quickly find a set of ATs for a given *Function*. To share the AT repository among multiple users, we are currently integrating the AT repository into the ASKALON resource management service.

Note that the meteorology ontologies represent the knowledge in the meteorology simulation domain, it can also be used to compose other workflows in the domain. We believe that our approach can be adapted for other scientific domains by developing the corresponding ontologies for those domains, as we demonstrated in the previous sections for the meteorology simulation domain.

VI. Experimental Results

We conducted three experiments to evaluate our Grid workflow composition approach presented in this paper: (i) how much time it takes to find a requested AF, (ii) how many data flows in a Grid workflow can be established automatically, and (iii) how fast a semantic Grid workflow representation can be mapped to the corresponding syntactic Grid workflow representation.

The computer used to run the experiments is a normal desktop computer with 2 GB memory and one 2.4 GHz Intel Core 2 Duo CPU. The Java runtime environment used is JRE 1.5.0_13. The Grid workflow application used in the experiments is MeteoAG (see Section V).

The number of the subclasses of the class *Meteorology-Function* defined in the ontologies is 17, which means 17 AFs can be extracted and used for the composition of the Grid workflow MeteoAG. To evaluate the performance of searching AFs in the case where there are hundreds AFs available, we manually duplicate the defined AFs to 200 AFs and check the performance behavior while the number of AFs increases. Two curves in Fig. 12 illustrate the execution time of searching for an AF based on free text and based on input and output data, respectively. The horizontal axis indicates the number of available AFs. The vertical axis (in logarithmic scale) shows the execution time. In case of 200 AFs, searching an AF based on free text costs a few milliseconds, and searching AFs based on input or output data costs lightly more than half a second (673.05 ms). The execution time of searching AFs based on input and output data is longer because all given input and output data used for searching have to be compared with all defined input and output data of AFs. Note that when the number of the available AFs is 17, the execution time of searching AFs based on free text and based on input and output data are 3.32 milliseconds and 45.03 milliseconds, respectively. We can conclude that the overhead is insignificant and the workflow composition tool can respond fast enough to search requests of users.

When AFs are put in a workflow and connected with control flow, data flow among the AFs can be established based on the semantics of data ports. Fig. 13 compares the number of the available *data sources* for each data port in the workflow MeteoAG in three cases: the total number of *data sources*, denoted by *No heuristics*, the number of *data sources* filtered based on data types of data ports, denoted by *Syntax heuristics*, and the number of *data sources* filtered based on semantics of data ports, denoted by *Semantics heuristics*. The horizontal axis indicates the data ports specified in the workflow, in the order in which they are available during the workflow execution. We marked the name of each AF above the plot point of the first data port of this AF in the figure. Obviously, if the number of available *data sources* for a data port is 1, the data flow of this data port can be established automatically

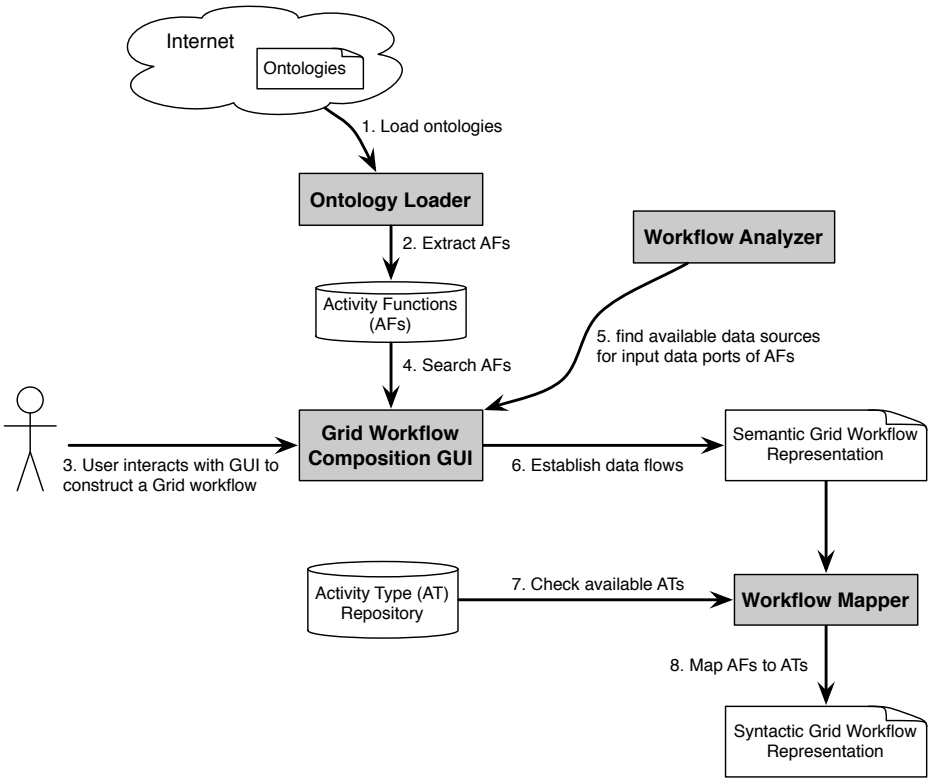


Fig. 11. The Architecture of the Prototype

because there is only one option. In case of *No heuristics*, all plot points form a specific shape is because the numbers of *data sources* for all input data ports of an AF in this case are always the same since no syntactic or semantic information of data ports are considered. In case of *Syntax heuristics*, most numbers of data sources are larger because many data ports in the workflow have type `agwl:file`. In contrast, the case *Semantics heuristics* shows that the numbers of *data sources* of 40 out of 45 (89%) data ports are 1, i.e. the corresponding data flow can be established automatically. The reason that some numbers in case of *Semantics heuristics* are still greater than 1 (which are 2 or 3) is as follows. Both AF *InitializeSimulationCase* and AF *RAMSMakevfile* produce the same kind of output data *Soil*, thus two *data sources* are available for the data ports with data meaning *Soil* of the successor AFs (e.g., *RAMSInit*); AF *RAMSInit* produces two *RAMSModeledAtmosphere* and AF *RAMSHist* produces one *RAMSModeledAtmosphere*, which means three *data sources* are available for the data port with data meaning *RAMSModeledAtmosphere* of AF *REVUDump*. See Fig. 10 for these AFs in the MeteoAG workflow. For the data ports with more than one *data sources* available, user interaction is required during the mapping process. This is also the case where the automatic Grid workflow composition approach does not work because of the requirement of user interaction.

To map an AF in the workflow to ATs, we compare the *input source data representations* of the AF with the input data

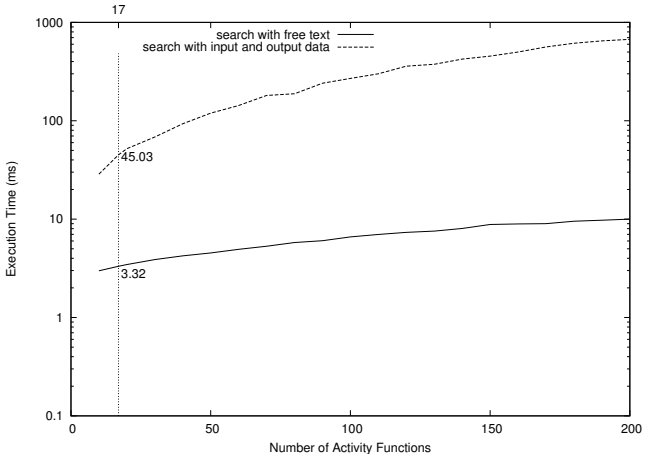


Fig. 12. Execution Time of Searching for AFs Base on Free Text and Base on Input and Output Data

representations of those ATs in the AT repository associated with the AF. Fig. 14 illustrates the execution time of mapping AF *RAMSMakevfile* (direct match), mapping AF *RAMSHist* (indirect match), and mapping the entire workflow MeteoAG. The vertical axis is shown in logarithmic scale. In case of direct match, the maximum average execution time is less than 3 milliseconds. In case of indirect match, the execution time is 274.50 milliseconds when 200 ATs are associated with

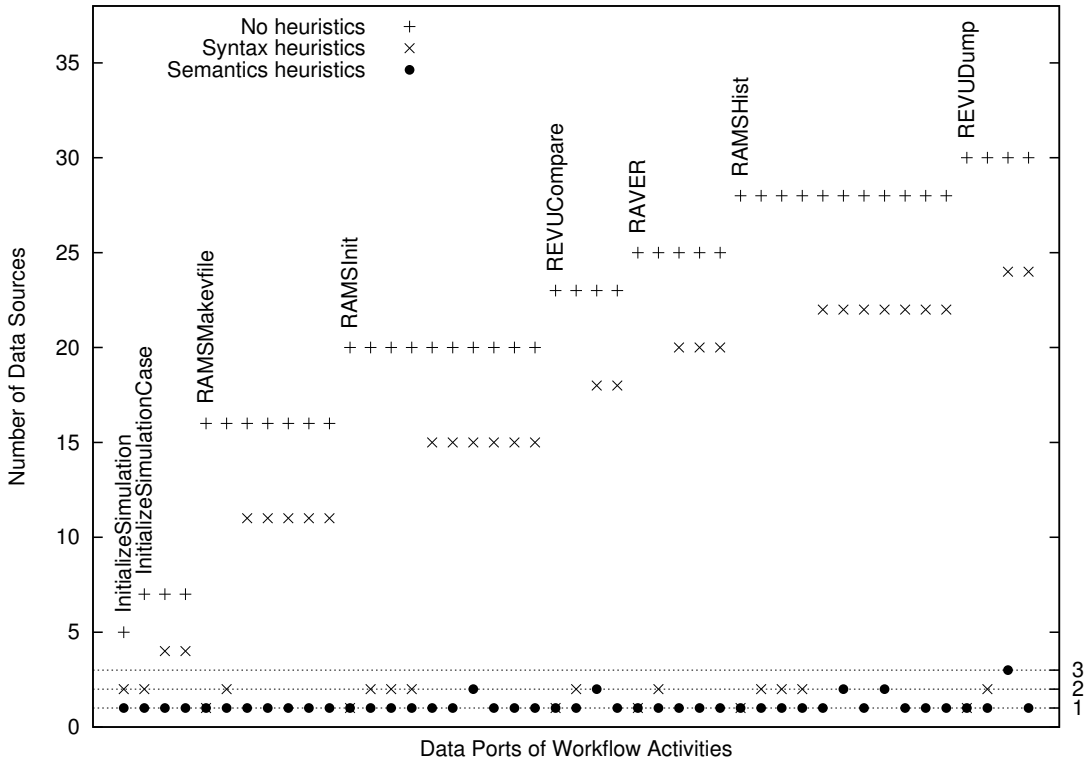


Fig. 13. Comparison of the Numbers of Available Data Sources for Data Ports in the Grid Workflow MeteoAG

the same AF. The execution time of indirect match is longer because only when all ATs are compared and no direct match found, do we try to find *data conversion ATs*. In the case where 200 ATs are available for each given AF, the execution time of mapping the entire Grid workflow MeteoAG is 449.70 milliseconds which consist of the execution time of 2 indirect matches and 6 direct matches. Since MeteoAG consists 8 AFs and two nested level of parallel loops, we can draw the conclusion that a few seconds are enough to map a moderate sized Grid workflow application to the corresponding syntactic representation. Note that the number of AFs specified in the Grid workflow is not the number of activities to be actually executed on the Grid which is usually much larger because a workflow may have parallel loops.

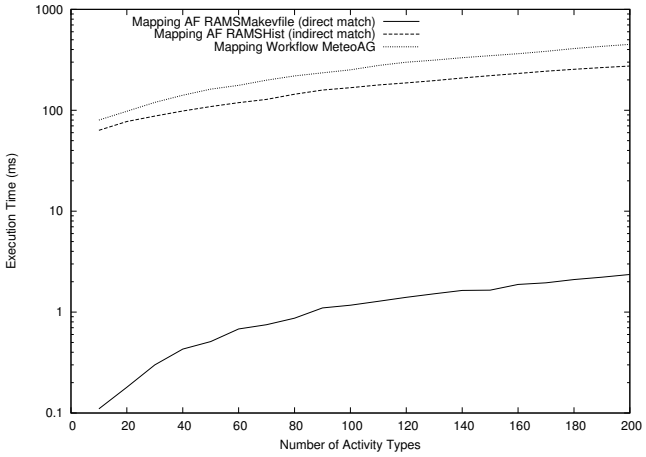


Fig. 14. Execution Time of Mapping AF *RAMSMakevfile* (direct match), Mapping AF *RAMSHist* (indirect match), and Mapping the Entire Grid Workflow MeteoAG

VII. Related Work

Many scientific Grid workflow systems have been developed in different projects, e.g., Pegasus [1], Taverna [2], Kepler [3], Triana [4], ICENI [5], etc. However, most of them have no or very limited domain knowledge support to facilitate Grid workflow composition. Domain knowledge support is considered to be an essential part for workflow composition tools to be easy-to-use for domain users. We limit this section to selected work with domain knowledge support and compare them against our approach.

Taverna [2] is a data centric workflow development environment in the ^{my}Grid project specially developed for the bioinformatics domain. It supports semantic service discovery through the Feta plugin [21]. Feta models the input and output data of service operations with `parameter` entities which is described by the properties *semantic type*, *format*, *collectionType*, *collectionFormat*, etc. Although these properties are similar as the properties of *DataRepresentation* in our approach (our approach has two additional properties: *hasStorageType* and *hasArchiveType*), we separate data *semantic type* (or data meaning) from the rest (which we call *DataRepresentation*) so that domain users can compose Grid workflows by only

focusing on *semantic types* (or data meaning) without dealing with the specific data representations. Furthermore, the semantic information in our approach is used not only for AF discovery but also for semi-automatic data flow composition.

Berkley et al. [9] adopted semantic annotations to datasets and services in Kepler [3] to facilitate workflow authoring. Compared with our approach, they still present syntax information (e.g. data types) to users when composing workflows, and the semantic annotations are used to verify whether two connected services (or data sources) are "semantically compatible" instead of establishing the data flow automatically as we do. Zhang [22] described an ontology-driven scientific Grid workflow composition approach in Kepler for hyperspectral image processing applications. He developed a workflow component ontology and a data type ontology and plugged them into Kepler. But the mapping from semantic data types to structural data types [22] is considered inefficient and unnecessary in his work.

In ICENI [5], Mayer et al. [23] presented a component description language which features a separation of concerns between component meaning, behavior and implementation. While the idea sounds similar to ours, there are big differences between their approach and our approach. In their approach, the meaning level, i.e., the highest level of abstraction, concerns data flow, including data types. The behavior level concerns control flow. No semantic descriptions of data are used in their approach. In contrast, the three levels in our approach are semantic, syntactic and concrete levels. And ontologies are introduced in our approach to enable semantic Grid workflow composition.

The Workflow Composition Tool (WCT) [8] in the K-Wf Grid project adopted an automatic workflow composition based on semantic descriptions of services' operations. The algorithm of Input-to-Output comparison is based an concept *Data Template* which describes *content*, *format* and *storage* constraints of data. While data representation in our approach is inspired by the concept *Data Template*, unlike WCT, our approach separates data representation from data meaning in the workflow language and users focus on data meaning to compose Grid workflows.

The Model-Based Workflow (MBW) approach [10] generates abstract workflow representations based on the Workflow-Driven Ontologies (WDO) [24], which are developed by domain scientists and contain knowledge about *method* consumes *data* and produces *data* and *product*. While the generated abstract workflow representations in MBW are same as the semantic Grid workflow representation presented in this paper, there is no data representation concept in MBW and it did not present an approach to map an abstract workflow representation to the corresponding executable one.

As a part of METEOR-S [25], Cardoso and Sheth [13] presented a formal description of Web Service Template (ST) and Web Service Object (SO), and a detailed algorithm to compute the degree of similarity of ST and SO taking into account semantic, syntactic and operational information. Their work is complementary to our approach and their algorithm can be used to find AFs based on free text or based on input and output data, and to map AFs to ATs.

The Ontology-driven workflow management system [26] introduces ontologies into biosequence processing system to assist workflow composition and optimization. Chen et al. [27], [28] presented a knowledge-based framework for semantic service composition. Nadarajan et al. [29] proposed a semantic-based hybrid workflow composition method within a three-layered framework. Majithia et al. [30] presented a framework of automated composition of semantic Grid services with a dynamic and adaptive mechanism for service discovery and composition. However, none of them support separation of data meaning and data representation as we presented in this paper.

There is also some other work done by the Semantic Web Service community to semantically describe services and compose services as workflows. OWL-S [14] is a set of ontology definitions designed to capture the behavior of services. The service presents the service profile, a description of what the service does. The service is described by the service model, which tell how the service works. Finally, the service supports the service grounding which specifies the invocation method for the service. The functionality provided by the service is specified through *hasInput*, *hasOutput*, *hasPrecondition* and *hasResult* of the service profile. OWL-WS [31] extends OWL-S to support workflow descriptions. WSDL-S [32] associates semantic annotations with elements of WSDL documents. However, none of them have the data representation concept and separate concerns between data meaning and data representation as we do.

VIII. Conclusions and Future Work

Grid workflow composition is still a complex task for domain scientists due to a lack of enough domain knowledge support in existing tools and the diverse structures of scientific data. This paper presents a novel domain oriented approach to alleviate this problem by separations of concerns between data meaning and data representation on the one hand and between activity function and activity type on the other hand. With this approach, the effort to manage and compose Grid workflows is significantly reduced because: (i) users compose Grid workflows with AFs, which requires domain knowledge only; (ii) data flows can be established semi-automatically based on data semantics of data ports; and (iii) data conversions between different data representations of workflow data are done automatically. We have implemented our approach as part of the ASKALON Grid application development and computing environment. Experiments with a real world meteorology workflow demonstrate that the overhead of semantic Grid workflow composition is negligible and about 90% of the data flows can be established automatically for this application.

For future work, we plan to apply our approach to Grid workflow applications in other scientific domains to further improve our approach for Grid workflow composition. We also plan to enrich our technology with machine learning techniques to optimize data flow composition of Grid workflows.

In addition to data flow composition, automatic control flow composition of Grid workflows based on semantic descriptions of activities is also an interesting direction which we are investigating.

ACKNOWLEDGMENT

We would like to thank our colleagues especially Dr. Maximilian Berger and Dr. Stefan Podlipnig for their constructive discussions and proof readings.

This work is partially funded by the European Union through the IST-034601 edutain@grid and INFSO-RI-222667 EGEE-III projects.

REFERENCES

[1] E. Deelman, G. Singh, M.-H. Su, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, G. B. Berriman, J. Good, A. Laity, J. C. Jacob, and D. Katz, "Pegasus: a Framework for Mapping Complex Scientific Workflows onto Distributed Systems," *Scientific Programming Journal*, vol. 13, no. 2, November 2005.

[2] T. Oinn, M. Addis, J. Ferris, D. Marvin, M. Senger, M. Greenwood, T. Carver, K. G. andMatthew R. Pocock, A. Wipat, and P. Li, "Taverna: A tool for the composition and enactment of bioinformatics workflows," *Bioinformatics Journal*, vol. 20, no. 17, pp. 3045–3054, June 2004.

[3] I. Altintas, C. Berkley, E. Jaeger, M. Jones, B. Ludäscher, and S. Mock, "Kepler: An Extensible System for Design and Execution of Scientific Workflows," in *16th Intl. Conf. on Scientific and Statistical Database Management (SSDBM'04)*. Santorini Island, Greece: IEEE Computer Society Press, June 21-23, 2004.

[4] I. Taylor, I. Wang, M. Shields, and S. Majithia, "Distributed computing with Triana on the Grid," *Concurrency and Computation: Practice and Experience*, 2005.

[5] A. Mayer, S. McGough, N. Furmento, J. Cohen, M. Gulamali, L. Young, A. Afzal, S. Newhouse, J. D. V. Getov, and T. Kielmann, *Component Models and Systems for Grid Applications*, ser. CoreGRID series. Springer, June 2004, vol. 1, ch. ICENI: An Integrated Grid Middleware to Support e-Science, pp. 109–124.

[6] T. Fahringer, R. Prodan, R. Duan, J. Hofer, F. Nadeem, F. Nerieri, S. Podlipnig, J. Qin, M. Siddiqui, H.-L. Truong, A. Villazon, and M. Wieczorek, *Workflows for eScience, Scientific Workflows for Grids*. Springer Verlag, 2007, ch. ASKALON: A Development and Grid Computing Environment for Scientific Workflows.

[7] I. Foster, J. Voeckler, M. Wilde, and Y. Zhao, "Chimera: A Virtual Data System for Representing, Querying, and Automating Data Derivation," in *14th International Conference on Scientific and Statistical Database Management (SSDBM'02)*, Edinburgh, Scotland, July 2002.

[8] T. Gubała, D. Harezlak, M. Bubak, and M. Malawski, "Constructing Abstract Workflows of Applications with Workflow Composition Tool," in *Proceedings of Cracow Grid Workshop (CGW'06), K-WfGrid - The Knowledge-based Workflow System for Grid Applications*, 2006.

[9] C. Berkley, S. Bowers, M. Jones, B. Ludäscher, M. Schildhauer, and J. Tao, "Incorporating Semantics in Scientific Workflow Authoring," in *SSDBM'2005: Proceedings of the 17th international conference on Scientific and statistical database management*. Berkeley, CA, US: Lawrence Berkeley Laboratory, 2005, pp. 75–78.

[10] L. Salayandia, P. P. da Silva, A. Q. Gates, and A. Rebellon, "A Model-Based Workflow Approach for Scientific Applications," in *Proceedings of the 6th OOPSLA Workshop on Domain-Specific Modeling*, 2006.

[11] T. Fahringer, J. Qin, and S. Hainzer, "Specification of Grid Workflow Applications with AGWL: An Abstract Grid Workflow Language," in *Proceedings of IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005)*. Cardiff, UK: IEEE Computer Society Press, May 9-12, 2005.

[12] J. Qin and T. Fahringer, "Advanced Data Flow Support for Scientific Grid Workflow Applications," in *Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis (SC07)*. Reno, NV, USA: IEEE Computer Society Press, November 10-16 2007.

[13] J. Cardoso and A. Sheth, "Semantic E-Workflow Composition," in *Journal Of Intelligent Information Systems*, vol. 21. Hingham, MA, USA: Kluwer Academic Publishers, 2003, pp. 191–225.

[14] The World Wide Web Consortium (W3C), "OWL-S: Semantic Markup for Web Services." [Online]. Available: http://www.w3.org/Submission/OWL-S/

[15] Y. Gil, *Workflows for e-Science – Scientific Workflows for Grids*. Springer Verlag, 2007, ch. Workflow Composition: Semantic Representations for Flexible Automation.

[16] The Object Management Group (OMG), "UML 2 Activity Diagram." [Online]. Available: http://www.omg.org/spec/UML/2.1.2/Superstructure/PDF/

[17] Protege Team, "Protege-OWL Editor." [Online]. Available: http://protege.stanford.edu/overview/protege-owl.html

[18] Jena Team, "Jena Semantic Web Framework API." [Online]. Available: http://jena.sourceforge.net/

[19] F. Schüller, J. Qin, F. Nadeem, R. Prodan, T. Fahringer, and G. Mayr, "Performance, Scalability and Quality of the Meteorological Grid Workflow MeteoAG," in *Proceedings of 2nd Austrian Grid Symposium*. Innsbruck, Austria: OCG Verlag, September 21-23, 2006.

[20] W. R. Cotton, R. A. Pielke, R. L. Walko, G. E. Liston, C. J. Tremback, H. Jiang, R. L. McAnelly, J. Y. Harrington, M. E. Nicholls, G. G. Carrio, and J. P. McFadden, "RAMS 2001: Current status and future directions," *Meteorology and Atmospheric Physics*, vol. 82, pp. 5–29, 2003.

[21] P. Lord, P. Alper, C. Wroe, and C. Goble, *The Semantic Web: Research and Applications*. Springer, 2005, ch. Feta: A Light-Weight Architecture for User Oriented Semantic Service Discovery, pp. 17–31.

[22] J. Zhang, "Ontology-Driven Composition and Validation of Scientific Grid Workflows in Kepler: a Case Study of Hyperspectral Image Processing," in *Proceedings of 5th International Conference on Grid and Cooperative Computing Workshops*, 2006.

[23] A. Mayer, S. McGough, M. Gulamali, L. Young, J. Stanton, S. Newhouse, and J. Darlington, "Meaning and Behaviour in Grid Oriented Components," in *GRID '02: Proceedings of the Third International Workshop on Grid Computing*. London, UK: Springer-Verlag, 2002, pp. 100–111.

[24] L. Salayandia, P. P. da Silva, A. Q. Gates, and F. Salcedo, "Workflow-Driven Ontologies: An Earth Sciences Case Study," in *Proceedings of Second IEEE International Conference on e-Science and Grid Computing (e-Science'06)*, vol. 0. Los Alamitos, CA, USA: IEEE Computer Society, 2006, p. 17.

[25] METEOR-S Team, "METEOR-S: Semantic Web Services and Processes." [Online]. Available: http://lsdis.cs.uga.edu/projects/meteor-s/

[26] M. Lemos, M. A. Casanova, L. F. B. Seibel, J. A. F. de Macedo, and A. B. de Miranda, "Ontology-Driven Workflow Management for Biosequence Processing Systems," in *Proceedings of 15th International Conference Database and Expert Systems Applications (DEXA 2004)*, vol. 3180/2004. Zaragoza, Spain: Springer, August 30-September 3 2004, pp. 781–790.

[27] L. Chen, N. Shadbolt, C. Goble, F. Tao, S. Cox, C. Puleston, and P. Smart, "Towards a Knowledge-based Approach to Semantic Service Composition," in *Proc. of the 2nd International Semantic Web Conference (ISWC2003)*, Florida, USA, 2003, pp. 319–334.

[28] L. Chen, N. R. Shadbolt, F. Tao, C. Goble, C. Puleston, and S. J. Cox, "Semantics-Assisted Problem Solving on the Semantic Grid," *Computational Intelligence*, vol. 21, pp. 157–176, 2005.

[29] G. Nadarajan, Y.-H. Chen-Burger, and J. Malone, "Semantic-Based Workflow Composition for Video Processing in the Grid," in *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, Hong Kong, China, December 12-18 2006.

[30] S. Majithia, D. W.Walker, and W.A.Gray, "Automated Composition of Semantic Grid Services," in *Proceedings of the UK e-Science All Hands Meeting 2004*, S.J.Cox, Ed., Nottingham, UK, August 31st - September 3rd 2004.

[31] S. Beco, B. Cantalupo, L. Giammarino, N. Matskanis, and M. Surridge, "OWL-WS: A Workflow Ontology for Dynamic Grid Service Composition," in *1st IEEE International Conference on e-Science and Grid Computing*. IEEE Computer Society, December 5–8 2005, pp. 148–155. [Online]. Available: http://eprints.ecs.soton.ac.uk/12773/

[32] The World Wide Web Consortium (W3C), "Web Service Semantics - WSDL-S." [Online]. Available: http://www.w3.org/Submission/WSDL-S/