Semi-automatic Composition of Ontologies for ASKALON Grid Workflows*

Muhammad Junaid Malik, Thomas Fahringer, and Radu Prodan

Institute of Computer Science, University of Innsbruck,
Technikerstraße 21a, A-6020 Innsbruck, Austria
{malik,tf,radu}@dps.uibk.ac.at

Abstract. Automatic workflow composition with the help of ontologies has been addressed by numerous researchers in the past. While ontologies are very useful for automatic and semiautomatic workflow composition, ontology creation itself remains a very important and complex task.

In this paper we present a novel tool to synthesize ontologies for the Abstract Grid Workflow Language (AGWL) which has been used for years to successfully create Grid workflow applications at a high level of abstraction. In order to semi-automatically generate ontologies we use an AGWL Ontology (AGWO - an ontological description of the AGWL language), structural information of one or several input workflows of a given application domain, and semantic enrichment of the structural information with the help of the user. Experiments based on two separate application domains (movie rendering and meteorology) will be shown that demonstrate the effectiveness of our approach by semi-automatically generating ontologies which are then used to automatically create workflow applications.

1 Introduction

The Grid [7] has eliminated the need for dedicated computational facilities and enables the scientists to leverage the computational power of resources located at geographically remote distances.

The Grid workflow programming model [19] has been highly successful in the last decade in facilitating the composition of medium to large-scale applications from existing basic activities defining a sequence of tasks to manage computational science processes. A Grid workflow application represents a collection of computational tasks (activities) interconnected in a directed graph through control and data flow dependencies that are suitable for execution on the Grid.

Domain scientists create Grid workflows with the help of Grid workflow composition tools and execute them with runtime environments [6], [5], [20], and [3]. These tools have greatly simplified the effort of a workflow designer through features such as auto-completion and design suggestions [11], [3] for workflows.

* This research is partially funded by the "Tiroler Zukunftsstiftung", Project name: "Parallel Computing with Java for Manycore Computers" and by the European Union under grant agreement number 261585/SHIWA project.

M. Alexander et al. (Eds.): Euro-Par 2011 Workshops, Part I, LNCS 7155, pp. 169–180, 2012.

Nevertheless manual workflow creation is still a very demanding task. It not only requires comprehensive application domain knowledge but also sufficient skills of the incorporated workflow composition tool.

Numerous research projects tried to automatically generate workflows [6], [3], and [17]. Most of these approaches are based on the analysis of the input data and functional description of the workflow activities with the help of domain ontologies [16]. Successful approaches rely on the completeness of domain ontologies (a formal representation of knowledge as a set of classes within a given application domain) which commonly must be manually created.

A major problem faced by scientists when they try to manually create domain ontologies is that manual ontology creation not only requires extensive domain knowledge and excellent ontology development skills but is also very time consuming.

Semi-automated ontology synthesis expedites the process of ontology creation and automatic updates of ontologies. To address the above mentioned problem, this paper proposes a new method for semi-automatic ontology synthesis for Grid workflows specified with AGWL.

Our approach for ontology creation is performed in three phases. In the first phase one or several AGWL activities or workflows of the same application domain are parsed and analyzed for the information available in the input AGWL data. We extract information about activities and their input and output ports which we refer to as classes. The next phase associates these classes (for instance, all data ports are associated with an AGWL *Data* class) with the AGWL ontology, which results in an intermediate ontology. Thereafter, in the final phase the user has to provide semantic information for these classes which is used to derive the final ontology associated with the domain of the input AGWL data. The final ontology can then be used to automatically create workflows of the given domain.

The paper is organized as follows; Section 2 provides the prerequisites and terminology for the proposed method. In Section 3 we introduce our novel method for semi-automatically creating ontologies. Section 4 presents experimental results, followed by Section 5 that discusses related work. Finally, Section 6 provides the concluding remarks and potential future research.

2 Prerequisite

An account of the terminology used in this paper is presented in this section. A real-world movie rendering algorithm workflow (Fig.1) known as *POVray* [8] is used as an example workflow. The *POVray* workflow is composed of three independent activities.

The first activity takes scene data as input and initializes the process by creating multiple *initialization* files for parallel execution of the Render activity. The Render activity takes this input from the *initialize* activity and generates images for individual frames of the movie. It then packs the frames into a *tarball*, which is consumed by the third activity *Convert* that combines these individual

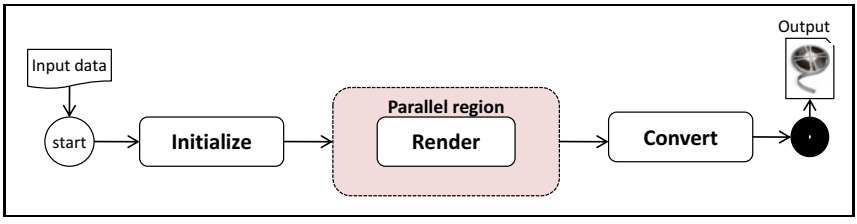

Fig. 1. *POVray* movie rendering workflow

frames and creates a movie. A Grid workflow is a collection of computational tasks (activities) orchestrated in a predefined order to achieve the required execution goals on the Grid. Each activity is described using a number of attributes such *name*, *type*, and *function*.

We use the Askalon Grid application development environment [6] for creating and executing Grid workflows. Askalon simplifies the development and execution of Grid applications, in particular workflows. It is based on a set of high level services which realize the automatic enactment and execution of Grid workflows in a transparent and efficient manner. With the help of its high level middleware services (scheduling, resource management, performance prediction, and enactment) Askalon hides the underlying details of the Grid from the user.

Askalon uses AGWL to describe Grid workflows. AGWL is an XML-based language designed in such a way that the user can concentrate on creating scientific applications without dealing with either the complexity of the Grid or any specific implementation technology such as Web/Grid services. it provides a rich set of constructs to simplify the specification of Grid workflow applications. AGWL has been used for creating workflows from diverse scientific domains, like material sciences, meteorology, astrology, mathematics, and computer graphics. The activity representation in AGWL is called ActivityType *(AT)*. An *AT* is comprised of function description, data *inPorts* and data *outPorts*.

In order to specify ontologies for the semantic description of workflows OWL [9] is used. AGWL ontology (AGWO) is the ontological representation of the AGWL constructs in the form of an OWL document. These constructs encoded as OWL classes are inherited from the *OWL Thing* class (the base class for all OWL ontologies). These classes are used as parent class for the domain specific classes when creating a new ontology. AGWO contains two base classes *Function* and *Data*; that provide foundations for the extended domain classes. Each of these classes may have one or more associated properties, or subclasses. These properties are used to specify additional features or attributes of these classes. An AGWL activity comprises a description of the computational task and its input/output, these components of the activity are represented using the base classes *Function* and *Data* respectively in the AGWO. Further semantic information for these classes is added by specifying *rdfs:onProperty*, and by declaring them as subclasses of existing classes in the ontology using *rdfs:subClassOf* construct. The object-Properties defined for the *Function* class include *hasInput* and *hasOutput*. Whereas the class *Data* has object-Properties *hasRepresentation*, *isInputTo*, *isOutputOf*, *isPartOf*,

and *hasPart* etc. The two base classes *Function* and *Data* of AGWO, are extended for each new domain ontology by creating subclasses *[Domain]Function* and *[Domain]Data*. For instance, activities from the *POVray* domain will be inherited from the class *POVrayFunction*, which is in turn inherited from *Function*. The *Function* and *Data* base classes are also related to each other using *hasInput* and *hasOuput* object-Properties. For example, a *POVray* movie rendering function is a *POVrayFunction* that takes as input some *POVrayData* and will produce some *POVrayData* as output. Fig.2 shows the definition of these ontological classes for AGWO.

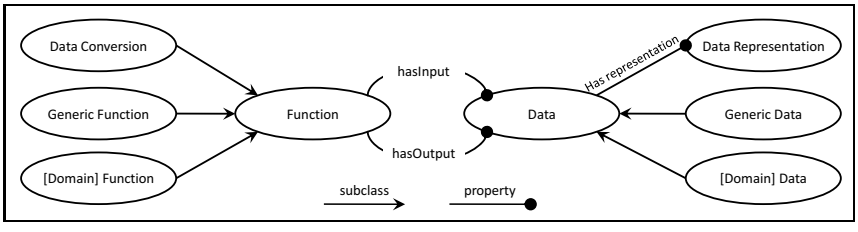

Fig. 2. AGWL ontology (AGWO) *Function* and *Data* classes

To synthesize ontologies, the proposed system extracts and processes extensive information which is stored in the Lexicon database *LDB* and in the Ontology database *ODB*. The *LDB* contains the lexicon of classes with semantic information (e.g. an image in a specific format is provided through an *inPort*), whereas the *ODB* contains the AGWL data and intermediate ontology structure etc., created during the ontology synthesis process.

3 Methodology

The proposed ontology synthesis system is built as part of the ASKALON [6] grid workflow development environment. The three phases of the ontology synthesis process are explained in the following subsections. A block-diagram of the overall system is shown in Fig.3.

3.1 Parsing and Canonicalization

The first component of the system the AGWL parser takes one or a set of AGWL activity constructs belonging to a certain scientific domain as user input. Fig.4 shows an input AGWL activity representation.

The parsing process extracts datasets (e.g. activity datasets such as {*activity, name, type, function*} and data *inPort* datasets such as {*data port, name, type, category, source*}) from the input AGWL data. As all the AGWL activity elements are not structurally identical, and also the attributes of similar workflow elements may vary in number, these datasets need to be normalized before they

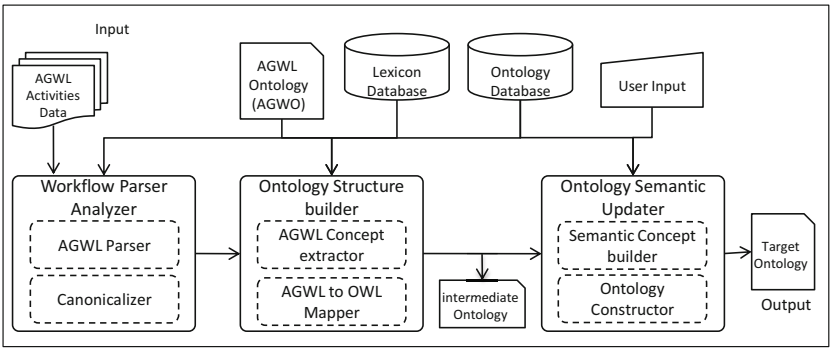

Fig. 3. Block diagram of the ontology synthesis system

can be canonicalized and stored in the database. For instance, some activities may not have a function attribute because they were not meant to be used for semantic analysis. Normalization is the process of adding missing attributes to these AGWL activities thus all the elements have the same set of attributes. After normalization, these datasets are transformed into the canonicalized form, which can be analyzed and processed using standard SQL commands.

```
<activity function="" name="Convert" type="Povray:Convert">
 <dataIns>
  <dataIn category="" name="movieyuv" source="gsiftp://..." type="agwl:file">
   <dataRepresentation>
    <storageType>FileSystem</storageType>
    <contentType>File</contentType>
    <archiveType>none</archiveType>
    <cardinality>multiple</cardinality>
   </dataRepresentation>
  </dataIn>
 </dataIns>
 <dataOuts>
  <dataOut category="" name="movie.avi" saveto="gsiftp://..." type="agwl:file">
   <dataRepresentation>
    <storageType>FileSystem</storageType>
    <contentType>File</contentType>
    <archiveType>none</archiveType>
    <cardinality>single</cardinality>
   </dataRepresentation>
  </dataOut>
 </dataOuts>
</activity>
```

Fig. 4. AGWL representation of the activity *Convert* from *POVray* workflow

In the canonicalization process, all the attributes of the extracted AGWL structures are analyzed. When an attribute without a value is found, the missing information is added to it, this information is collected from existing data contained in the *LDB* or otherwise through user input. The canonicalized form (database relations) of a sample AGWL activity appearing in Fig.4 is shown in Fig.5, which illustrates the three database relations, *activity*, *activity_port* and *data_representation*. The canonicalization process filled the missing attribute values for *function* in *activity*, and *category* in *activity_port* relation.

ACTIVITY				
id	**wf-id**	**name**	**type**	**function**
1	1	Convert	Povray:Convert	MovieConv

ACTIVITY_PORT							
id	**parent**	**element**	**name**	**category**	**type**	**source**	**saveto**
1	1	DataIn	movieyuv	Movie	agwl:file	gsiftp://	NULL
2	1	DataOut	movieavi	Movie	agwl:file	NULL	gsiftp://

DATA_REPRESENTATION					
Id	**parent**	**storage**	**content**	**archive**	**cardinality**
1	1	FileSystem	File	None	multiple
2	2	FileSystem	File	tgz	single

Fig. 5. Canonical form (in relational format) of the activity shown in Fig.4

The process is repeated for all the datasets. and the canonicalized information is stored in the ontology database *ODB*. The *ODB* has five relations. The workflow and activity related data is stored in the relations *workflow* and the *activity* respectively. The remaining three relations *act_port* and *wf_port* and *data_representation* are used to store information about activity dataports, workflow dataports, and their corresponding data representations respectively.

3.2 Creation of an Intermediate Ontology

In this phase the intermediate structure of the ontology is created. This is done by retrieving the canonical information for each AGWL element from the Ontology database *ODB* and transforming this information into a unique OWL class representation. Each newly created class is inherited from a *Data* or *Function* class (input/output port or activity) in the AGWO. This process results in an intermediate ontology. Continuing our *POVray* example, the structure of the ontology created for the *convert* activity (Fig.4) is shown in Fig.6. Only one *POVray* activity *Convert* is being used to create the intermediate ontology. The declaration of class *#Convert* is show in Fig.6 which is a placeholder for the *Function* class named *Convert* taken from the AGWL data of the activity.

```
1  <?xml version="1.0"?>
2  <!--ENTITY DECLARATIONS -->
3  <!--NAMESPACE DECLARATIONS -->
4   <owl:Ontology rdf:about=""> <owl:imports rdf:resource="http://www.askalon.org/ontologies/agwl"/></owl:Ontology>
5   <owl:Class rdf:about="#POVRAYFunction"><rdfs:subClassOf rdf:resource="&agwl;Function"/> </owl:Class>
6   <owl:Class rdf:about="#POVRAYData"><rdfs:subClassOf rdf:resource="&agwl;Data"/>  </owl:Class>
7   <owl:Class rdf:about="#Movieyuv"/> <rdfs:subClassOf rdf:resource="#POVRAYData"/>  </owl:Class>
8   <owl:Class rdf:about="#Movieavi"/>  <rdfs:subClassOf rdf:resource="#POVRAYData"/>  </owl:Class>
9   <owl:Class rdf:about="#Convert">
10  <rdfs:subClassOf rdf:resource="#POVRAYFunction"/>
11   <rdfs:onProperty rdf:resource="&AGWL;hasInput"/>
12     <rdf:Description rdf:about="#Movieyuv"/>
13   <rdfs:onProperty rdf:resource="&AGWL;hasOutput"/>
14     <rdf:Description rdf:about="#Movieavi"/>
15  </owl:Class>
16 </rdf:RDF>
```

Fig. 6. Structure of intermediate ontology before semantic enrichment

3.3 Semantic Association and Meaning Specification

The last phase deals with the semantic enrichment of the intermediate ontology. As discussed earlier there are two base classes, i.e. *Data* and *Functions*. Similarly an AGWL element can be either AGWL data (input/output port) or an AGWL activity. Every class from the intermediate ontology representing AGWL data or AGWL activity is presented to the user who then has to provide the semantic information for each of these classes. The user can also choose the semantic information for these classes from the *LDB*. Which means specifying a meaningful name for the class and its association with the appropriate parent class in the AGWO.

In case of an input/output port, a name is given to the port which specifies the contents of the port (e.g. an image in a specific data format). The port is declared as a new class inherited from the *[domain]Data* class, which, in turn, is a subclass of the *Data* AGWO class.

The user can also create a new class to specify a certain type of a *Data* class. For example, if the data port represents a *png* or *jpeg* image format, the user might create a class *image* as a subclass of *[Domain]Data* and then declare this data port as a subclass of the *image* class. Similarly in case of activities, the meaningful name provided by the user describes the function performed by each activity, and is declared as a subclass of *[domain]Function* class which, in turn, is a subclass of *Function* AGWO class. Again the user can create new class definitions to categorize the activities by function they perform as done for the data ports. During this interactive process all the information given by the user is stored in the *LDB*. The final ontology created for the activity *Convert* from the *POVray* workflow is illustrated in Fig.7. For the sake of simplicity, the definitions for the OWL entities and XML namespaces have been omitted. In the intermediate ontology the classes *Movieyuv* and *Movieavi* (line 7,8 Fig.6) are extended from the *POVrayData* class. The function *Convert* is by default declared as the subclass of *POVrayFunction* class. In the final ontology (Fig.7) which is enriched with the semantic information there are more class definitions.

On (line 7,8) the definition of *MovieData* and *MovieConv* appears as sub-classes of *POVRAYData* and *POVRAYFunction* respectively. The classes *Movieyuv* and *Movieavi* are now extended from the *MovieData* class (line 9,10), and the *Convert* class is extended from *MovieConv* class. Also as a result of the semantic enrichment the function definition (lines 13-34) of the *Convert* class has complete description of the input port of the function. This final ontology is used by Askalon automatic workflow composition tools to generate Grid workflows.

4 Experiments

Two types of experiments are performed; first to evaluate (validity checking) the ontologies and secondly to measure the performance of our ontology synthesis approach.

```
<?xml version="1.0"?>
<!--ENTITY DECLARATIONS -->
<!--NAMESPACE DECLARATIONS -->
 <owl:Ontology rdf:about=""><owl:imports rdf:resource="http://www.askalon.org/ontologies/agwl"/></owl:Ontology>
 <owl:Class rdf:about="#POVRAYFunction"><rdfs:subClassOf rdf:resource="&agwl;Function"/></owl:Class>
 <owl:Class rdf:about="#POVRAYData"><rdfs:subClassOf rdf:resource="&agwl;Data"/></owl:Class>
 <owl:Class rdf:about="#MovieData"> <rdfs:subClassOf rdf:resource="#POVRAYData"/></owl:Class>
 <owl:Class rdf:about="#MovieConv"><rdfs:subClassOf rdf:resource="#POVRAYFunction"/></owl:Class>
 <owl:Class rdf:about="#Movieyuv"/>  <rdfs:subClassOf rdf:resource="#MovieData"/></owl:Class>
 <owl:Class rdf:about="#Movieavi"/>   <rdfs:subClassOf rdf:resource="#MovieData"/></owl:Class>
 <owl:Class rdf:about="#Convert">
 <rdfs:subClassOf rdf:resource="#MovieConv"/>
  <rdfs:onProperty rdf:resource="&AGWL;hasInput"/>
   <owl11:onClass>
        <owl:Class>
                <owl:intersectionOf rdf:parseType="Collection">
                      <rdf:Description rdf:about="#Movieyuv"/>
                      <owl:Restriction>
                      <owl:onProperty rdf:resource="http://www.askalon.org/ontologies/agwl#hasRepresentation"/>
                        <owl:intersectionOf rdf:parseType="Collection">
                          <rdf:Description rdf:about="http://www.askalon.org/ontologies/agwl#DataRepresentation"/>
                          <owl:Restriction>
                           <owl:onProperty rdf:resource="http://www.askalon.org/ontologies/agwl#hasStorageType"/>
                              <owl:hasValue rdf:resource="http://www.askalon.org/ontologies/agwl#FileSystem"/>
                              <owl:onProperty rdf:resource="http://www.askalon.org/ontologies/agwl#hasContentType"/>
                              <owl:someValuesFrom rdf:resource="#File"/>
                              <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#nonNegativeInteger">
                              1
                              </owl:cardinality>
                      </owl:Restriction>
                      </owl:Restriction>
                </owl:intersectionOf>
        </owl:Class>
    </owl11:onClass>
   <rdfs:onProperty rdf:resource="&AGWL;hasOutput"/>
   <!--DETAILS OMITTED -->
  </owl:Class>
</rdf:RDF>
```

Fig. 7. Structure of the final ontology

4.1 Ontology Evaluation

For the evaluation of domain ontologies the gold standard approach [13] is most widely used. This approach is based on comparison of the newly synthesized ontologies to existing ontologies in the same domain. These reference ontologies in the domain which are known to be valid and are known as "gold standard". Other approaches include real life testing of the ontologies [15], verification of ontologies manually; involving human effort [12], and by comparing the ontology to the source data [2] that is used to generate the ontology. For our experiment, we combine the "gold standard" approach with a real life application of ontologies to evaluate the domain ontologies synthesized by using the described method of this paper.

For the gold standard evaluation an existing manually created ontology from the meteorology domain is used as the "gold standard" ontology [16]. We used existing AGWL activities of the meteorology domain to semi-automatically synthesize a meteorology ontology based on the described approach of this paper. The resulting ontology has then been used to automatically generate a meteorology workflow with the Askalon workflow synthesis tool [16]. The so generated workflow represents the same functionality as given by the workflow created by hand or when using the gold standard meteorology ontology with the main difference that our approach takes much less time (see Table.1). Apart from the

major objective of synthesizing valid ontologies, the secondary goal of the ontology synthesis approach was to improve user experience and reduce the effort required to synthesize ontologies. Table.1 shows the time needed to generate the workflow manually versus semi-automatically based on our approach. It took almost two weeks to manually create the gold standard meteorology ontology (used as reference ontology in our experiments) after after acquiring complete domain knowledge. Whereas our approach required substantially less time which was in the order of several hours.

4.2 Performance Measurements for Ontology Synthesis Process

To evaluate the performance behavior of the ontology synthesis tool, We used the AGWL data from the movie rendering workflow *POVray* [8]. The movie rendering application used for the experiments takes four parameters: a set of input frames, initialization file, configuration file, and number of frames to be processed per activity. The original *POVray* workflow contains only three activities as shown in Fig.1. We introduce intermediate artificial activities in the workflow to evaluate the performance behavior of the tool with higher number of activities. These additional activities are just data conversion activities and do not change the actual functionality of the workflow.

Table.1 tabulates the time required to synthesize the ontologies for the above mentioned variations of the *POVray* workflow. The manual input is taking most of the time in the synthesis process. Yet it is clear from the time measurements that by using our ontology synthesis approach, the time required to create ontologies for AGWL data is substantially shorter than a pure manual approach.

Table 1. The Ontology synthesis system performance measurements

Method	Activities	I/O ports	Database seek	Processing	Manual input	Total
			Time in seconds.			
Movie Rendering Domain (Workflow:: *POVray*)						
Semi-Automatic	3	12	0.104	1.56	620	621.664
Semi-Automatic	6	18	0.379	8.338	1200	1208.717
Semi-Automatic	8	24	0.711	17.775	1800	1818.486
Semi-Automatic	10	30	1.273	36.917	2800	2838.19
Meteorology Domain (Workflow:: *MeteoAG*)						
Semi-Automatic	15	79	3.53	11.47	17567	17582
Manual	15	71	$\approxeq$ 120 work hours			

5 Related Work

The use of ontologies in the context of automatic workflow composition has driven the scientists towards the development of automatic ontology synthesis techniques.

Two approaches are particularly similar to our method, which use three phases for ontology creation. The OBO-Edit tool [22] involves the extraction of terms or keywords for ontologies from the source text. This leads to definitions of the terms (classes) by using online information. Relationships between the terms are explored which lead to ontology classes. The OBO-framework loads classes from existing ontologies. Only these classes can act as potential parents for the newly discovered classes. In our approach, besides the use of a lexicon database for searching existing classes, we extend this functionality by providing the user an opportunity to create new classes on-the-fly, if no suitable parent class is found automatically. Moreover, our system automatically establishes relations based on the occurrences of such classes in AGWO. This reduces the chance of having incorrect relations between two classes because AGWO is proven to be correctly defined for AGWL [16].

Another attempt on automatic ontology synthesis [14], synthesizes ontologies based on existing ontologies for similar input information. This approach introduces two additional phases, namely validation and evolution. It tries to match and align the newly found classes with existing classes in the reference ontologies. This method very much depends on the availability and richness of existing ontologies, whereas in our approach ontologies can be generated from scratch without any need of existing domain ontologies. Numerous other attempts explored ontology synthesis [18], [4], [21] and [10], all of them largely focused on very specific aspects of a full ontology synthesis scenario. For example, a comprehensive study of the classification of classes and matching and alignment techniques is presented in [4]. In [10] a detailed analysis of ontology alignment techniques is presented.

In [1] the idea of mapping XSD file entities to OWL classes is described. A limitation of their approach is the absence of relations between different types of classes.

6 Conclusion

A crucial aspect towards automatic generation of Grid workflow applications is the availability of effective ontologies. This paper introduces a novel tool for semi-automatic synthesis of domain ontologies as a basis for the automatic generation of Grid workflows (written in the language AGWL). For this purpose we use an AGWL Ontology, structural information as AGWL data of a given domain, and semantic enrichment of the structural information with the help of the user. Experiments have been shown based on two separate application domains (movie rendering and meteorology) that demonstrate the quality of our approach. Semi-automatically generated ontologies are used by ASKALON to automatically generate workflows. Furthermore, performance experiments shown in the Table 1 reflect that the time it takes to synthesize ontologies even for large number of input activities is very reasonable.

References

1. Bohring, H., Auer, S.: Mapping xml to owl ontologies. In: Jantke, K.P., Fhnrich, K.-P., Wittig, W.S. (eds.) Leipziger Informatik-Tage. LNI, vol. 72, pp. 147–156. GI (2005)
2. Brewster, C., Alani, H., Dasmahapatra, S., Wilks, Y.: Data driven ontology evaluation. In: Proceedings of the International Conference on Language Resources and Evaluation (LREC), Lisbon, Portugal (2004)
3. Callahan, S.P., Freire, J., Santos, E., Scheidegger, C.E., Silva, C.T., Vo, H.T.: Vistrails: visualization meets data management. In: Proceedings of the 2006 ACM SIGMOD International Conference on Management of Data, SIGMOD 2006, pp. 745–747. ACM, New York (2006)
4. Castano, S., Ferrar, A., Montanelli, S., Hess, G.N., Bruno, S.: State of the art on ontology coorination and matching. deliverable 4.4 version 1.0 final (2007)
5. Deelman, E., Singh, G., Su, M.H., Blythe, J., Gil, A., Kesselman, C., Mehta, G., Vahi, K., Berriman, G.B., Good, J., Laity, A., Jacob, J.C., Katz, D.S.: Pegasus: a framework for mapping complex scientific workflows onto distributed systems. Scientific Programming Journal 13, 219–237 (2005)
6. Fahringer, T., Jugravu, A., Pllana, S., Prodan, R., Seragiotto, C., Truong, H.L.: ASKALON: a tool set for cluster and Grid computing. Concurrency - Practice and Experience 17(2-4), 143–169 (2005)
7. Foster, I., Kesselman, C. (eds.): The grid: blueprint for a new computing infrastructure. Morgan Kaufmann Publishers Inc., San Francisco (1999)
8. Povray, `http://www.povray.org/`
9. Web ontology language, `http://www.w3.org/2004/OWL/`
10. Euzenat, J., Le Bach, T., Barrasa, J., Bouquet, P., De Bo, J., Dieng, R., Ehrig, M., Hauswirth, M., Jarrar, M., Lara, R., Maynard, D., Napoli, A., Stamou, G., Stuckenschmidt, H., Shvaiko, P., Tessaris, S., Van Acker, S., Zaihrayeu, I.: State of the art on ontology alignment. knowledge web deliverable d2.2.3 (2004)
11. Junaid, M., Berger, M., Vitvar, T., Plankensteiner, K., Fahringer, T.: Workflow composition through design suggestions using design-time provenance information. In: 5th IEEE International Conference on E-Science Workshops, pp. 110–117 (December 2009)
12. Lozano-Tello, A., Gómez-Pérez, A.: Ontometric: A method to choose the appropriate ontology. Journal of Database Management 15(2) (April-June 2004)
13. Maedche, A., Staab, S.: Measuring Similarity between Ontologies. In: Gómez-Pérez, A., Benjamins, V.R. (eds.) EKAW 2002. LNCS (LNAI), vol. 2473, pp. 251–263. Springer, Heidelberg (2002)
14. Automatic Ontology Generation: State of the Art. University of Versailles Technical report. Ivan bedini and benjamin nguyen (2007)
15. Porzel, R., Malaka, R.: A Task-based Approach for ontology Evaluation. In: Workshop on Ontology Learning and Population, ECAI 2004 (2004)
16. Qin, J., Fahringer, T.: A novel domain oriented approach for scientific grid workflow composition. In: Proceedings of the 2008 ACM/IEEE Conference on Supercomputing, SC 2008, pp. 21:1–21:12. IEEE Press, Piscataway (2008)
17. Slota, R., Zieba, J., Kryza, B., Kitowski, J.: Knowledge evolution supporting automatic workflow composition. In: e-Science, p. 37. IEEE Computer Society (2006)
18. Sure, Y., Studer, R., Fensel, C.D., Lebensversicherungsund, S., Reimer, C.U.: On-to-knowledge methodology - final version (2002)

19. Taylor, I., Deelman, E., Gannon, D., Shields, M.: Workflows for e-science. XXII, 530 p. 181 illus., Hardcover (2007)
20. Taylor, I., Wang, I., Shields, M., Majithia, S.: Distributed computing with triana on the grid: Research articles. Concurr. Comput.: Pract. Exper. 17, 1197–1214 (2005)
21. Vrandecic, D., Pinto, H.S., Sure, Y., Tempich, C.: The diligent knowledge processes. Journal of Knowledge Management 9(5), 85–96 (2005)
22. Wächter, T., Schroeder, M.: Semi-automated ontology generation within obo-edit. Bioinformatics 26, i88–i96 (2010)