

Workflow Composer and Service Registry for Grid Applications

Marian Bubak^{a,b} Tomasz Gubała^c Michał Kapałka^a Maciej Malawski^a
Katarzyna Rycerz^a

^a*Institute of Computer Science AGH, al. Mickiewicza 30, 30-059 Kraków, Poland*

^b*Academic Computer Centre CYFRONET AGH, Nawojki 11, 30-950 Kraków, Poland*

^c*Section Computational Science, University of Amsterdam, Kruislaan 403, 1098 SJ Amsterdam, The Netherlands*

Abstract

Automatic composition of workflows from Web and Grid services is an important challenge in today's distributed applications. The system presented in this paper supports the user in composing an application workflow from existing Grid services. The flow composition system builds workflows on an abstract level with semantic and syntactic descriptions of services available on the Grid. Two main modules of the system are the flow composer and the distributed Grid service registry. We present motivation, the concept of the overall system architecture and the results of a feasibility study.

Key words: Grid workflow, workflow composition, distributed registry, ontologies, Grid programming

1. Introduction

Scientific and commercial applications often require large-scale computational and data storage resources which are globally distributed. The idea of resource sharing among diverse organizations has led to the development of Grid technologies which are applied mostly to scientific computations and data sharing. Meanwhile, in the industry, the rising need for standard mechanisms of B2B integration of distributed software systems have resulted in development of the Web services technology. It was achieved through exploiting mechanisms derived from the World Wide Web [1]. At the same time, developments in artificial intelligence and knowledge representation have provided a background for the

Email address: malawski@agh.edu.pl (Maciej Malawski).

Semantic Web [3] which, in turn, extends the Web with semantic descriptions of its contents and makes it a subject of search and reasoning by software agents. At present, all the aforementioned technologies are merging, leading to such concepts as Grid services, Semantic Grid [2][5] and semantic Web services.

The growing society of active Grid users provides an increasing number of available resources. One of the most promising approaches to building applications is the workflow model, which is addressed by many research projects [16]. However, the problem of building an application requires finding and orchestrating appropriate services which is frequently a nontrivial task for a human. In this paper we describe a framework for automatic composition of workflows of Grid services. The rationale is the following:

- The number of services that can perform a certain action may be very large. Instead of manually browsing their descriptions, the programmer should be assisted by automated tools.
- There are many possibilities for constructing a workflow graph from matching services. It would be helpful to have tools that can advise in selecting the optimal services.
- The Grid is a very dynamic environment. When new services appear and others become unavailable, there emerges a need to update existing workflows using new services.
- In scientific computing and also in rapidly-evolving business applications it is often necessary to provide rapid prototypes of novel applications.

The problem of workflow composition for distributed applications was first approached from the Web services matchmaking and discovery perspective. An example is the DAML-S/UDDI Matchmaker extending UDDI with semantic capabilities [7]. That technique supported by semantic Web service descriptions [10] is used in various approaches to complex Web service composition [11]. Nevertheless, these solutions do not try to solve the generic workflow composition problem and are not suitable for long-lasting scientific computations. CAT [17] is an approach to interactive composition of Web services. It uses semantic descriptions of services to assist the user in constructing computational pathways. The results are presented in the form of corrections and suggestions for interactive user input. Other automatic solutions to the composition problem exploit the techniques of regression planning [12], PPDL [13] and Hierarchical Task Network-based planning [18].

In order to check whether it is possible to compose workflows in an automatic way, we have developed the Application Flow Composer [8]. It is based on the Common Component Architecture (CCA) and it creates workflows from CCA components. The user prepares a so-called Initial Workflow Description with the first and the last component specified, and possibly several intermediate ones. The AFC reads the document and tries to find the missing components using a component registry as a source of information. It produces a Final Workflow Description - a document which is already complete and can be used as input for an engine that will set up and run the workflow on the Grid. The first prototype has shown that automatic composition of workflows is possible and that an architecture consisting of separate composer and registry modules is appropriate. However, the need to enrich the service description with semantics has also been demonstrated. Moreover, distribution of the entire system appears necessary to provide better suitability for the Grid environment, which is distributed by nature.

The experiences from the first prototype and the studies of related work lead us to the following conclusions, driving the development of our workflow composition system:

- The system should exploit semantics of Grid services for a more accurate matchmaking

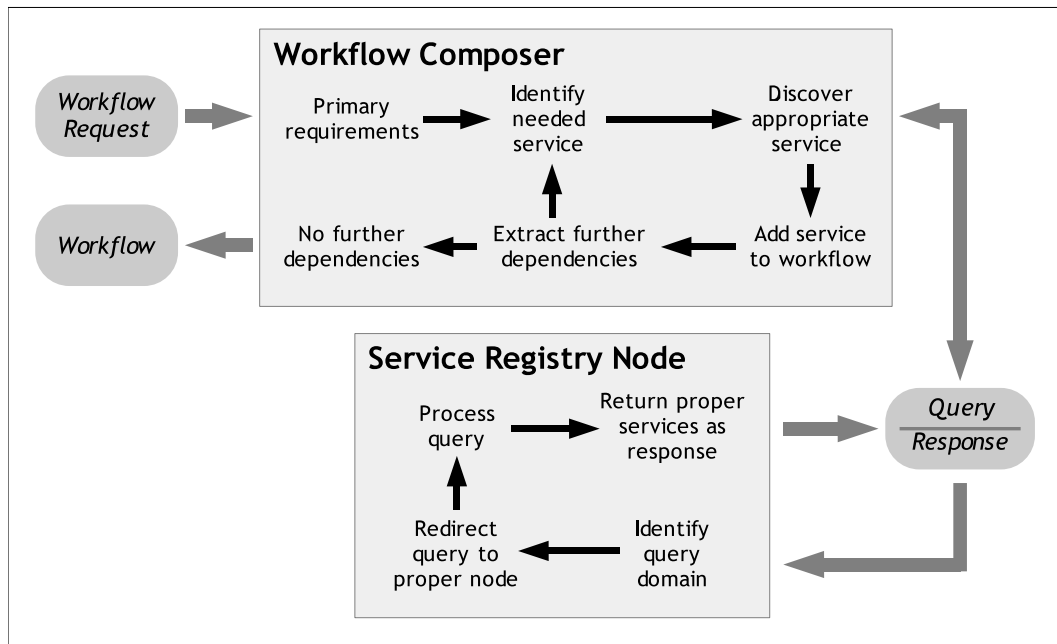


Fig. 1. System architecture

process.

- The system requires a fault-tolerant, effective, distributed service registry serving as a knowledge base.
- Addressing the latest standardization efforts in the Grid technologies [15] provides a wider user base.
- Complex scientific applications demand more elaborated workflows to be composed (e.g. with loops or conditional branches).
- The registry lookup mechanism ought to be fast yet still general, supporting widely-known standards (such as the XPath language).

2. Workflow Composition System

The workflow composition system consists of two modules: the Grid Registry for storing service descriptions and the Flow Composer, using them for workflow composition (see Fig. 1). In order to separate these two functionalities the registry is implemented as a completely independent system which stores and publishes documents describing Grid services. To suit the system better, the registry query resolving capability is optimized according to the needs of the composer module. The registry still may be used by any external client through its WSDL interface and is a standalone part of the workflow composition system.

The flow composer requires access to the service description documents store and it uses specific queries to retrieve information from that store. Therefore the composer is constantly aware of the location of the Grid registry and may open a connection any time. The interface between these two parts is based on Web service protocols (WSDL,

SOAP). The flow composer works as a client for the Grid service exposed by the registry. The communication is non-blocking, based on a pull-type notification model.

2.1. Workflow Composition

The objective of the flow composer part of the system is to construct a service-based workflow according to user demands. A user may query for a provider of particular data. Let us assume that this data has to be computed by several Grid services cooperating together. The composer has to find this set of services and propose a sensible way of connecting them in one workflow in order to achieve the desired results. Currently, the system uses SCUFL workflow notation developed by the *myGrid* project [4] as the output workflow format.

When users want to obtain data on a certain structure and meaning, they start the workflow composition. Next, the flow composer agent opens a connection to a Grid registry node and starts the workflow building process (see Fig. 1). The agent tries to find a service which provides the requested data as output; if this service needs any kind of input, the composer seeks a provider of that particular message. The process may finish either with success or with failure. The latter case occurs if there are no providers for some kind of input data. Success means that either there are no further dependencies of the services in the workflow or all of them are matched with data explicitly provided by the user. Even when the system is unable to find every needed data provider, it tries to return as complete a workflow description as possible, leaving undiscovered data sources as leaves of the workflow graph.

In order to choose the proper provider the composer employs a specific matching algorithm, based on semantic comparison: the composer looks for a service providing a data piece of a certain structure and meaning. This meaning is included inside the data (or service) description by means of ontologies. If a service which exactly fits the needs cannot be found, the composer extracts a list of similar meanings (in the form of ontology classes) and then tries to find a provider for these similar solutions. We use the OWL Web ontology notation [14] to indicate the placement of data in the data meaning space of a certain application domain. The ontology vocabulary describing the domain is set by the user who usually knows the scientific area the computation belongs to. With this input, the composition agent employs an ontology inference engine to compare the meaning of two (or more) entities of data. This approach enables the agent to find not only exact matches but also these solutions which seem similar enough to be of interest for the composition process. It uses such notions as class equivalence, inheritance or inclusion of classes to achieve results. Every matching service is then classified with a level of similarity between the requested meaning and the located one. Depending on this classification, the flow composer chooses the best services to include them into the produced workflows.

It is obvious that the workflow graph building process includes construction of many independent branches, sometimes connected together only in one node of one service. Most of these branches may be constructed by distinct entities, communicating together either in a master-worker paradigm or in a peer-to-peer fashion. Therefore, we have parallelized the presented composition algorithm to make it more efficient. The results of performance tests may be found in Sect. 3.

2.2. *The Grid Registry*

The Grid Registry is responsible for storing and managing documents which contain descriptions of services. It allows for typical database operations, such as adding new data, updating existing documents and searching for services that meet specific requirements. The last operation is supposed to be the most frequent one and thus it is highly optimized. The efficiency of other operations – those that change data in the registry – will not influence its overall performance significantly, so they can be more expensive.

All the documents stored in the registry share a common structure to support effective information lookup. The flow composer part needs both syntactic and semantic descriptions of services which have to be put in one document format. Although there are many standards for describing syntax and semantics of services, we have developed our own document format that has two advantages: it is simple, from both the conceptual and technical points of view, and it allows for effective searching for semantic information with use of XPath queries. It is an extension to the WSDL standard with the most important WSDL elements having a “meaning” attribute added. Its value is the name of the ontology class describing the meaning of the corresponding element. In order to introduce hierarchy into ontology trees we use a certain structure of domains. The decision to exploit the WSDL standard is convergent with emerging WS-RF Grid computation proposals [15].

Technically, the registry is a set of independent parts, called registry nodes, forming a P2P network. Each registry node is responsible for a specified domain and each one performs a specific task, such as storing documents, providing indexes or resolving general queries (i.e. these that have to be answered by more than one registry node). Therefore, the internal registry structure is heterogeneous, making the proposed architecture flexible and scalable. In order to make that complex structure easy to use its internals are hidden behind a common, simple interface. From the user’s point of view all registry nodes are equivalent. Generally, the registry is used as a black box; however it may provide additional information, such as the origin of each response. This information may be used by the querying system to optimize subsequent queries. To support that functionality, we have developed an algorithm for query routing. Each registry node, given a query together with its domain, is able to send it to the appropriate node of the registry. To provide an efficient and scalable solution we have decided to use local, hierarchical routing based on routing tables – a concept extensively used on the Internet and in local networks. We have conducted several tests to show that the routing algorithm behaves well and is both effective and scalable.

3. Feasibility Study

In order to prove our solution for the workflow composition problem in the Grid environment we have conducted a set of tests with the prototype implementation. The application used for testing was a sample simulation (see Fig. 2). It consisted of up to eight services connected together to form a processing flow. Every service had two different implementations with slightly different external interfaces and semantics. It was up to decision mechanism of the flow composer which service instance to choose in a given workflow node. The complexity of the composition algorithm is dependent on the number

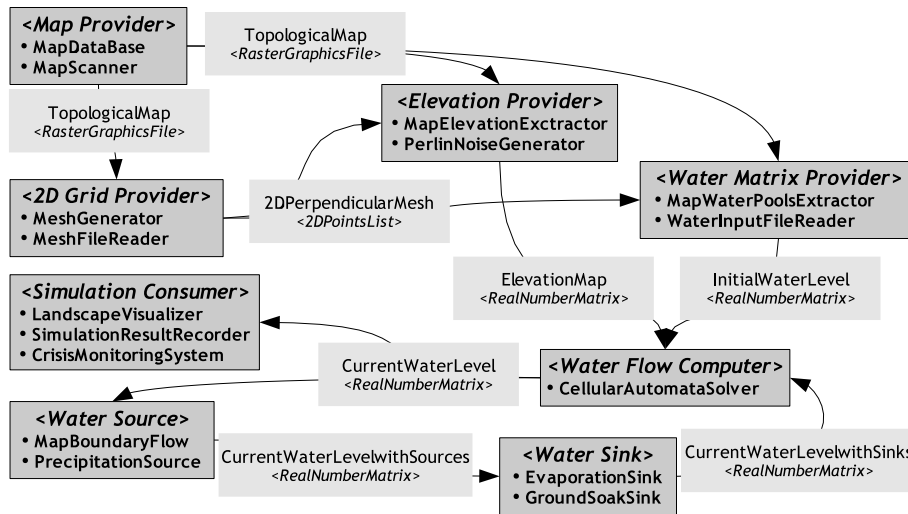


Fig. 2. Workflow of the testing application

of connections in the workflow graph. Every connection expresses one dependency of a Grid service – the need for an input message. Searching for a provider of a particular message may involve more than one registry query step. This happens when there are no providers of data with the exact desired meaning yet the ontology reasoning engine is able to name some different classes with similar meanings.

The tests were conducted using the Xindice XML database implementation, a Tomcat server to host the XML database endpoint and Globus Toolkit 3.2 to provide an execution environment for the Grid Registry service. The three Linux machines used to run the test were: a dual Pentium 4 2.4 GHz computer as the first Amsterdam node, a dual Xeon 2.4 GHz node in Kraków and a slightly slower dual Pentium III 1.0 GHz machine acting as the third node, also located in Amsterdam. Every single test run was repeated ten times in very similar circumstances to exclude anomalies and every single time value was obtained using a clock working with a precision of 1 ms.

The first test showed how the system performs on various levels of distribution. On every step the same workflow was composed, however the information about the required services was dispersed on a different number of nodes. The results may be seen in Tab. 1 (**distributed** columns). The first sample was produced with both the composer and the registry parts located on the same machine, communicating locally with each other (the leftmost column). All the required services were available on the same, local node of the registry. The second sample is a result of execution distributed between two Grid sites with the composer module and one node in Amsterdam while another part of the registry was in Kraków. The second node contained critical information regarding the composition process to make distant query redirection inevitable. In the third step we divided the registry even further: two of its nodes were placed as before and the third node was located in Amsterdam but running on another computer. The registry was deliberately configured in the way that every query for the third node had to be redirected through the node located in Kraków to force it to cross the double distance twice (this was a round trip).

Table 1

Results of local and distributed composition test

Number of nodes	1 node	2 dist.	2 local	3 dist.	3 local
Registry initiation	41%	40%	37%	39%	29%
Inference	17%	16%	15%	16%	12%
Resolution	15%	17%	23%	18%	38%
Retrieval	6%	6%	5%	6%	4%
Internal comp.	21%	21%	20%	21%	17%
Total, <i>ms</i>	7871	8151	8979	8420	11257

In order to show exactly how the topological distribution of nodes influences computation time the second test was conducted with exactly the same setup of the distributed registry content and nodes. Yet, this time all the nodes of the registry were located on the same testing machine (dual Pentium 2.4 GHz CPU). In such an environment every connection, query redirection and result retrieval had to be done using the same serialization mechanisms and communication protocols (each node of the registry was hosted by a different Grid service container instance). It should be noted that no information was sent remotely and that all communication was performed using the localhost network interface. The results are summarized in Tab. 1 for comparison (**local** columns).

The registry connection initialization time is the delay of establishing a connection between the flow composer and a node of the registry Grid service. The query status check time is the summarized time of processing of all queries issued in the course of a single test run. There were eleven queries, some computed locally, some redirected to distant nodes (in case of distributed tests). The query result retrieval time is the overall time spent on retrieving the results for all queries. Its small value indicates that the result is already transferred back to the query origin node at the moment the client is able to acquire it; the retrieval operation is therefore completely local. The inference time is the summarized time used to initialize the inference engine, load the proper OWL vocabulary document and conduct ontology reasoning.

The results obtained in these tests are optimistic. The registry connection initialization time is significant, but it is a one-off operation and it may be neglected for larger and more complex workflows. The inference engine proved to be effective enough to perform any kind of inference operation in a reasonable time. Furthermore, the detailed results (not shown here for legibility reasons) show that the performance of reasoning over the same vocabulary increases with time and subsequent operations always tend to be substantially faster than first or second ones. Obviously, the overall time increases with the level of registry distribution but the result is still obtained, even when queries have to cross thousands of kilometers between distant Grid sites – that is due to very fast Internet connections between modern scientific laboratories.

4. Summary and Future Work

The objective of the development was to create a simple, yet operational system. The prototype is already able to build valid and semantically correct workflows and therefore fulfills the main functional requirements. The feasibility of the approach was tested as

described in Sect. 3, proving that the concept, algorithms and architecture are correct and that they satisfy all the essential requirements as well.

As a further improvement we will use the topology information provided by the Grid Registry to migrate composition agents into the proximity of information sources. From our previous study [8] we know that connection between the flow composer and the registry tend to be very intensively used. The tests performed on a system distributed over large distances show that communication delays introduced by the distribution of registry nodes are substantial even in the case of a relatively simple test application workflow. To reduce this cost it seems advisable to move the workflow computation near the information source. Subsequently, we plan to further develop workflow notation to support runtime conditions and alternative processing branches.

The current implementation of the Grid Registry supports searching for documents with syntactic, semantic and text-based queries; adding and removing documents; query redirection mechanisms and resolving queries to multiple nodes. The main disadvantage of the current Grid Registry is that all its configuration has to be performed manually. We intend to implement dynamic, automatic configuration of registry nodes, with support for discovery of new nodes. There is also a need to provide synchronization of data stored in various parts of the registry as a basis for fault tolerance and data recovery mechanisms.

Acknowledgements We are grateful to Prof. P.M.A. Sloot for valuable discussions and to Piotr Nowakowski for helpful comments. This work was partly funded by the European Commission in the framework of IST projects CrossGrid, GRIDSTART and K-WfGrid as well as by the Polish State Committee for Scientific Research, SPUB-M 112/E-356/SPB/5.PR UE/DZ 224/2002-2004.

References

- [1] G. Alonso, F. Casati, H. Kuno, V. Machiraju, *Web Services*, Springer-Verlag Berlin Heidelberg, 2004
- [2] D. de Roure, N.R. Jennings, N. Shadbolt, *The Semantic Grid: A future e-Science infrastructure*, *Grid Computing - Making the Global Infrastructure a Reality*, John Wiley and Sons Ltd., 2003, 437–470
- [3] T. Berners-Lee, J. Hendler, O. Lassila, *The Semantic Web*, *Scientific American* 284 5 (2001) 34–43
- [4] R. Stevens, R. McEntire, C.A. Goble, M. Greenwood, J. Zhao, A. Wipat, P. Li, *myGrid and the drug discovery process*, *Drug Discovery Today: BIOSILICO*, 2 4 (2004) 140-148
- [5] H. Zhuge, *Semantics, Resource and Grid*, *Elsevier FGCS* 20 1 (2004) 1–5
- [6] B. Chandrasekaran, R. Jorn, V. Josephson, R. Benjamins, *What Are Ontologies, and Why Do We Need Them?*, *IEEE Intelligent Systems* 14 1 (1999) 20–26
- [7] K. Sycara, M. Paolucci, A. Ankolekar, N. Srinivasan, *Automated Discovery, Interaction and Composition of Semantic Web services*, *Journal of Web Semantics* 1 1 (2003) 27–46.
- [8] M. Bubak, K. Górká, T. Gubała, M. Malawski, K. Zajęc, *Component-based System for Grid Application Workflow Composition*, 10th European PVM/MPI Users' Group Meeting (proc.), Springer LNCS 2840 (2003) 611–618
- [9] M. Bubak, T. Gubała, M. Kapalka, M. Malawski, K. Rycerz, *Grid Service Registry for Workflow Composition Framework*, *Int. Conf. on Computational Science – ICCS 2004 (proc.)*, Springer LNCS 3038 (2004) 34–41
- [10] A. Ankolekar et al., *DAML-S: Semantic Markup for Web Services*, 1st Int. Semantic Web Working Symposium (SWWS), Stanford University (2001) 411–430
- [11] S. McIlraith, T.C. Son, *Adapting ConGolog for Programming the Semantic Web*, *Working Notes of The 5th Int. Symp. on Logical Formalization of Commonsense Reasoning (2001)* 195–202
- [12] D. McDermott, *Estimated-Regression Planning for Interactions with Web Services*, 6th Int. Conf. on Artificial Intelligence Planning Systems (proc.), AAAI (2002) 204–211

- [13] J. Peer, A PDDL based Tool for Automatic Web Service Composition, 2nd Workshop on Principles and Practice of Semantic Web Reasoning (PPSWR 2004) at the 20th International Conference on Logic Programming (proc.), (2004)
- [14] Web-Ontology Working Group, OWL Web Ontology Language, W3C Recommendation documents set, 10 Feb 2004,
- [15] K. Czajkowski, D.F. Ferguson, I. Foster, J. Frey, S. Graham, I. Sedukhin, D. Snelling, S. Tuecke, W. Vambenepe, The WS-Resource Framework 1.0, May 2004
- [16] A. Slominski, G. von Laszewski, Scientific Workflows Survey <http://www.extreme.indiana.edu/swf-survey/>
- [17] J. Kim, Y. Gil, M. Spraragen, A Knowledge-Based Approach to Interactive Workflow Composition, Workshop on Planning and Scheduling for Web and Grid Services at ICAPS04, Whistler, Canada, 2004 (to appear)
- [18] U. Kuter, E. Sirin, D. Nau, B. Parsia, J. Hendler, Information gathering during planning for web service composition, Workshop on Planning and Scheduling for Web and Grid Services at ICAPS04, Whistler, Canada, 2004 (to appear)