Grid Service Registry for Workflow Composition Framework

Marian Bubak[1,2], Tomasz Gubała[2,3], Michał Kapałka[1], Maciej Malawski[1,2] and Katarzyna Rycerz[1,2]

[1] Institute of Computer Science, AGH, al. Mickiewicza 30, 30-059 Kraków, Poland
[2] Academic Computer Centre – CYFRONET, Nawojki 11, 30-950 Kraków, Poland
[3] Faculty of Sciences, Section of Computational Science, University of Amsterdam
Kruislaan 403, 1098 SJ Amsterdam, The Netherlands
{bubak,malawski,kzajac}@uci.agh.edu.pl
T.Gubala@cyfronet.krakow.pl, kapalka@icslab.agh.edu.pl

Abstract. The system presented in this paper supports the user in composing the flow of distributed application from existing Grid services. The flow composition system builds workflows on an abstract level with semantic and syntactic description of services available in a Grid services registry. This paper presents concepts of an overall system architecture and it focuses on one of the two main modules of the system – the distributed Grid service registry.
Keywords: Grid workflow, workflow composition, distributed registry, ontologies, Grid programming

1 Motivation

The future Grid is often described as a geographically-distributed set of services deployed on different sites by some service providers. These *Grid services*, as described in the Open Grid Service Architecture (OGSA) specifications [7, 12], should allow the user to do computations, access some resources, get information or use external scientific devices. Usually each of them will provide only a part of functionality required by the user, so using many of them, connected in a workflow, will be required quite often. It is one of the new approaches to programming Grid applications. Unfortunately, as the number of available services grows, the task of manual workflow composition becomes very difficult and time-consuming. A mature flow composition system is needed to support the user and thus make the task feasible. This kind of system would also help in fast Grid application development (prototyping) and in finding services appropriate for completing a partial solution prepared by the user. It would also allow for automatic, on-demand creation of workflows for specified tasks or rebuilding old workflows when they cannot be run on the Grid anymore due to frequent changes of the environment. Two approaches to mapping an abstract workflow defined in terms of application components onto a set of available Grid resources are presented in [5]. In the Web Services community, the most popular registry system is the Universal Description, Discovery and Integration standard

[13]. The implementation of a distributed UDDI registry, within a full-fledged P2P environment, supported by sophisticated registry ontologies, can be found in METEOR-S [9,4]. The core specification of the UDDI standard (already in version 3.0) also introduces a new, distributed registry architecture. A simpler, but still very flexible and efficient solution for sharing any kind of information stored in XML documents can be found in JXTA Search [14]. Another example of this approach is the myGrid project [10] where the improved UDDI system has been successfully introduced in a Grid OGSA environment. To the best of our knowledge, there is still no mature distributed system for semi-automatic workflow composition on a Grid. As a proof-of-concept and for feasibility studies we have developed a prototype of such a system – an Application Flow Composer (AFC) [2]. It is based on the Common Component Architecture (CCA) and builds flows comprised of CCA components. We perceive, however, that syntactic information about components (the types of their ports) is not sufficient for efficient workflow composition. The other disadvantage of the system is its centralization, leading to lack of scalability and fault tolerance. Therefore, we have decided to design a new, Grid-enabled distributed system. It is intended to be used to compose workflows from Grid services, utilizing their semantic description.

2 Overall Structure of the System

The new system consists of two main elements: a flow composer and a distributed registry (see Fig. 1). Both the flow composer and the registry are distributed and should run in a heterogeneous, geographically-spread Grid environment. They can communicate in a standard way (e.g. via SOAP) or, if the highest efficiency is required, via a dedicated, fast protocol. This distributed topology makes both subsystems more scalable and fault-tolerant.

The roles of the flow composer and the registry can be explained with the following scenario. The user who wants to build a new Grid application consisting of Grid services, describes the initial conditions of the application workflow and sends them to flow composition unit. The flow composer divides this initial workflow document into smaller parts and tries to find appropriate solutions (in terms of Grid services) for each of them. To achieve this, it has to ask the registry for information about services that conform to a given semantic or syntactic description. After it finds all the necessary components, it combines results from all subtasks and builds the final workflow description that will be returned to the user. When multiple solutions can be applied, more than one final document is prepared. The user should choose the one that is the most appropriate. Sometimes the user's help is required during the composition process. The system might ask the user questions in order to ease and speed up the decision-making process, especially regarding semantic matchmaking which is always difficult for an artificial entity.

The flow composer architecture can be seen as a distributed set of agents, composing workflow on a user request. We use the well-known term *agent* for

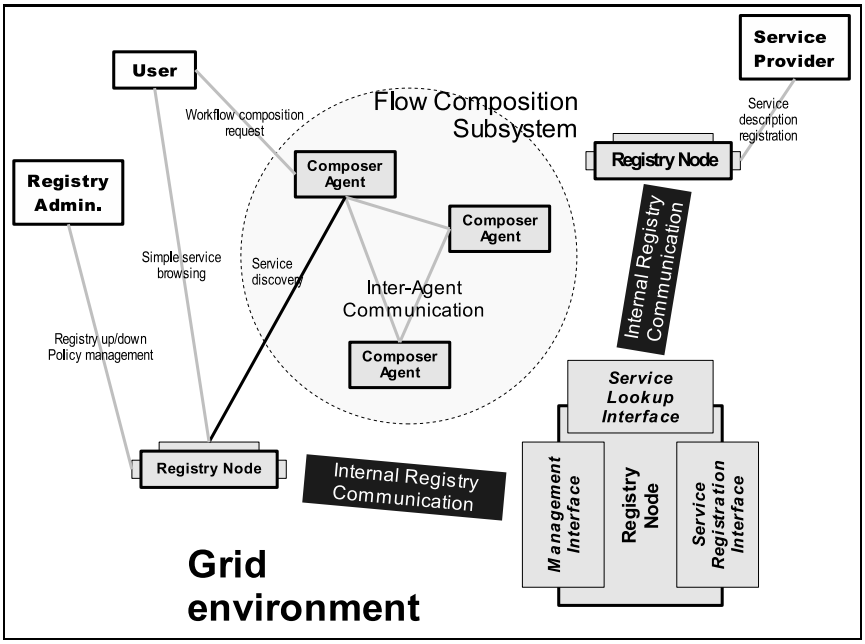

Fig. 1. General view of the system architecture

describing the components of the flow composer, because each of them operates on behalf of the user, contains its own rules of quasi-autonomous decision making and uses resources (i.e. descriptions of Grid services). This agent-based approach has several advantages. It makes the solution scalable and efficient as the agents can spread on many resources and work together on one composition problem. It also imparts some fault-tolerance on the system as a failure of one agent shouldn't stop the whole process. A more detailed description of the flow composer subsystem and the workflow composition process is given in [3].

The role of the distributed service registry in the entire framework is crucial. It is responsible for efficient delivery of data describing every available Grid service for the use of the flow composer. In this paper we describe our approach to building the registry together with requirements analysis and design.

3 Distributed Registry

3.1 Requirements

The registry should enable publication and searching for information about components of a workflow. It should be as universal as possible in order to reduce the need for multiple registries within one Grid environment. However, as we are not able to make a registry capable of storing each kind of information, we will concentrate on Grid services and their descriptions, aiming at flexibility and

extensibility. The registry should meet the following requirements which stem from demands of Grid computing and the flow composition system:

- be distributed and scalable,
- be efficient, at least as far as searching for data is concerned (the speed of updating and inserting new data into the registry isn't so crucial),
- allow for some redundancy of stored data (i.e. replication), thus lacking single points of failures,
- do not depend too much on the format of the information that can be stored, so new data formats can be introduced or the current one extended,
- be as fault-tolerant as possible,
- use technologies that are Grid-enabled,
- be simple to use.

3.2 Contents of the Registry

We want the registry to store information about Grid services. However, a simple syntactic description is not sufficient and should be extended so that every Grid service is described at least by:

- its unique identifier (within the whole Grid),
- the syntactic description of its ports and messages (e.g. WSDL),
- semantic information about the service itself and the ports, messages, tasks, data transformations, etc. it provides,
- the domain (or domains) it belongs to.

This complete description of each Grid service will be called a Grid Service Information Document (GSID). We do not specify its format, however, we assume that it will be an XML document, which should guarantee portability, flexibility and extensibility. It will also allow for using many pre-existing technological solutions, such as parsers, XML databases, etc. The *domain* field enclosed in every GSID is used by the registry's internal data distribution algorithm. All information in the registry is classified according to its domain, which can be, for example, a scientific domain (like biology, mathematics, etc). It can be a separate field, provided explicitly during service registration, or a function of the service's semantic description. It plays a very important role and thus it should be selected carefully taking into account the following requirements:

- it should be itself a simple data structure, e.g. the name of an ontology,
- it must be a part of an uncomplicated structure (like a set of ontology trees) with not too many disjoint parts (e.g. trees), to enable efficient data distribution,
- the number of domains shouldn't be too small, as it could limit the number of registry nodes, since these two quantities are strongly correlated.

The user will be able get a list of all most general domains (ie. domains that have no parents) or a list of all direct subdomains of a specific domain from the registry. This will assure that at least for the "domain" field the ontologies

will be well known to users. The list of domains can change as users add more ontologies, however the uniqueness and global consistency of domain names in the whole registry is still guaranteed.

The syntactical description of a Grid service is relatively simple. We can apply Grid/web service standards, like WSDL. The semantical description seems to be much more complicated. We have decided to use ontologies to express semantics, but the choice of notation is still to be made and many problems (e.g. whether to use shared or local ontologies) still have to be solved. Fortunately, many projects are exploring the areas of ontologies and semantic description of web/Grid services – these include OWL-S and the myGrid project [10, 11].

3.3 Architecture of the Registry

The registry should be highly distributed and scalable. To achieve that, its architecture is a structure of equal and independent registry nodes with a given communication topology. By *equal* we mean that every node works in the same way and presents to the user the same interfaces. Therefore, the complicated structure of the registry is hidden behind the interfaces and all GSID documents can be accessed by sending requests to an arbitrary registry node regardless of their distribution within the whole registry.

As long as the efficiency is not taken into account, it doesn't matter for a user which node he or she will use to access the registry through one of its interfaces. The node that receives the user's request is responsible for its proper redirection to other nodes of the registry when necessary. The algorithm of request redirection (routing) is under development. The problem is that no node can store complete information about data distribution within the whole registry as this solution wouldn't be too scalable. On the other hand, remembering only the domains of the nearest neighbors may not be sufficient and broadcasting user's requests throughout all the nodes should be strictly limited.

Every registry node will be responsible for storing GSID documents related to some specific domains (see Fig. 2). The choice of nodes responsible for given domains can be done both manually, by a system administrator, and automatically, if that is allowed by the registry's configuration. The main goal of such an approach is to store information very close to the places where the corresponding Grid services are deployed and/or most frequently used. It is reasonable to assume that every domain can be represented by at most one registry node as this solves problems of data coherency. However, data redundancy is required (see Sect. 3.1) so we have to introduce data replication between nodes, and, as efficiency remains essential, cache mechanisms should be added to the registry – this will require additional copying of data between many nodes. However, all data can be kept coherent and up-to-date in this kind of complicated structure and the concept of a single designated node for each domain, as well as other advanced techniques may be applied for that purpose. The number of registry nodes and the communication topology are dynamic so new nodes and interconnecting links can be added and removed without restarting. Of course, when

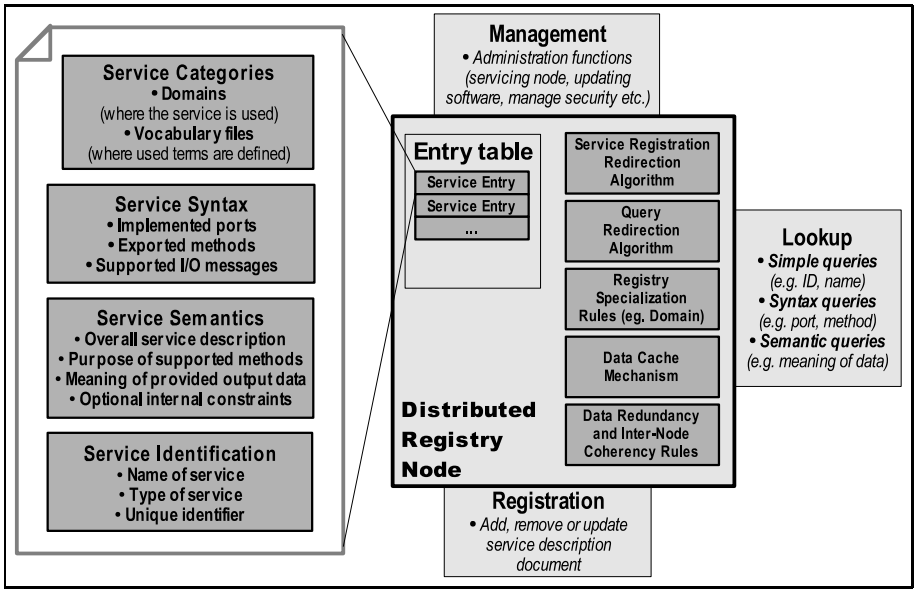

Fig. 2. Details of a single registry node

the number of nodes changes, some data has to be transferred and some time required to propagate all the required information throughout the whole registry.

3.4 Registry Interfaces

Each registry node will present the following interfaces to the outside world: a registration interface, a management interface and a lookup interface.

The registration interface is responsible for publishing GSID documents and unregistering them when needed. The registration requester may be a user (e.g. a developer of a Grid service) or any other entity that can do it automatically (e.g. searching for services and then publishing some of them). The registration scenario can be the following: the user wants to register Grid service S in domain D (placed somewhere in one of the domain trees). If the domain doesn't exist, a domain registration request has to be sent by a privileged entity. After that, service S can be registered by sending a registration request to any node of the registry. This node will take care of redirecting the request to the right place (the node responsible for domain D) at which service S will be registered.

The lookup interface allows users to search for data in the registry. An essential step is the specification of the type of queries the registry should support. As data lookup should be very fast, we'd rather not put any complicated ontological reasoning machine inside the registry. On the other hand, we cannot confine users to using only these queries that are related to syntactic information (this would be highly insufficient e.g. for our flow composer). Thus, the following types of queries should be supported by the registry:

- search by ID,
- search by syntactic information,
- search by semantic service description but using only names of ontologies and their attributes, not ontological reasoning.

In the last case, the entity that is asking a question is limited only to queries like: "find all services that have ports which are described by one of the following ontologies: O_1, O_2, O_3". The registry does not care about ontological relations; it only compares names of ontologies (strings). However, these relations might be introduced here by the entity asking the question, by properly preparing the list of ontology names that is put in the query (e.g. O_2 and O_3 may be subclasses of O_1). This approach should be efficient and flexible.

In order to make the migration of clients closer to data possible (e.g. agents of the flow composer) the registry will, in response to a query, return information on where exactly (i.e. on which node) the requested GSID document has been found. It will enable selecting the node which stores the data the user is looking for and asking next questions directly there, thus speeding up the lookup process.

4 Implementation Choices

It's now too early to make detailed implementation choices; we thus prefer to point out several requirements. First of all, as we would like the system to be portable, the use of a highly portable programming language (e.g. Java or C#) is desirable. Secondly, as the system has to be distributed, scalable and, at the same time, efficient, a non-hierarchical peer-to-peer paradigm with a lightweight underlying platform and fast communication protocols should be used. This will also allow for automatic deployment of system components in containers on nodes. For the first prototype of the system we are going to use Globus Toolkit 3.0 [8] as a hosting environment. For ontological reasoning we might use the Jena2 framework [15].

The registry should be conformable to a well-known standard to make it more universal and portable. This can be, for example, the third version of the UDDI specification that already supports distributed registries and data replication [13]. As the registry is going to be itself a Grid application, it seems obvious that it has to use a platform that is Grid-enabled. This could be one of the existing peer-to-peer environments or Grid middleware packages. We are considering using the JXTA Search system as a base and extending it with the required functionality. For an internal registry document database we investigate the possibility of using XML-dedicated database management systems, like Apache Xindice [1]

5 Conclusion and future works

This paper presents the design of a workflow composition system, with emphasis on the registry component (service). The general concept of a distributed registry is proposed basing on nodes specialized according to data domains, with

mechanisms of data replication and caching. The main interfaces of the registry are designed to meet the requirements of the flow composition system. While the main functionality of the registry system is defined, there are still many issues that need to be solved, e.g. the choice of data replication mechanisms, query routing among registry nodes and implementation technologies. Our current work is focused on these issues and on implementing the proof-of-concept prototype. The following algorithms and related protocols still have to be developed:

- request redirection algorithm,
- data replication algorithm,
- request and response caching, fault tolerance, backup servers.

Acknowledgements This work was partly funded by the European Commission, Project IST-2001-32243, CrossGrid, Project IST-2001-34808, GRIDSTART and the Polish State Committee for Scientific Research, SPUB-M 112/E-356/SPB/5.PR UE/DZ 224/2002-2004. We are grateful to Piotr Nowakowski for his comments.

References

1. Apache Xindice, The Apache XML Project, http://xml.apache.org/xindice
2. Bubak, M., Górka, K., Gubała, T., Malawski, M., Zając, K.: Component-based System for Grid Application Workflow Composition, in: Dongarra, J., et al. (Eds.): 10th European PVM/MPI Users' Group Meeting, Venice, Italy, September 29 - October 2, 2003, Proceedings. LNCS 2840, Springer 2003, pp. 611-618
3. Bubak, M., Gubała, T., Kapałka, M., Malawski, M., Rycerz, K.: Design of Distributed Grid Workflow Composition System. Cracow Grid Workshop 2003, November 2003, CYFRONET-AGH, Cracow http://www.cyfronet.krakow.pl/cgw03
4. Cardoso, J. and Sheth, A.: Semantic e-Workflow Composition. Journal of Intelligent Information Systems (JIIS), Vol. **12**, 191-225, 2003
5. Deelman, E., et al.: Mapping Abstract Complex Workflows onto Grid Environment. Journal of Grid Computing **1**, 25-39, 2003
6. Expert Group Report: Next Generation Grid(s), European Grid Research 2005-2010, June 2003
7. Foster, I., et al.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. OGSA WG, Global Grid Forum, June 2002
8. Globus Toolkit 3.0, http://www.globus.org/ogsa
9. METEOR-S Project: http://lsdis.cs.uga.edu/proj/meteor/SWP.htm
10. myGrid Project, http://mygrid.man.ac.uk/
11. OWL-S http://www.daml.org/services/owl-s/1.0/
12. Tuecke, S., et al.: Open Grid Services Infrastructure (OGSI) version 1.0, OGSI WG, Global Grid Forum, June 2003
13. UDDI 3.0 http://www.oasis-open.org/committees/uddi-spec/doc/tcspecs.htm
14. JXTA Search web page, http://search.jxta.org/
15. Jena A Semantic Web Framework for Java, http://jena.sourceforge.net/