Automating Scientific Experiments on the Semantic Grid

Shalil Majithia, David W. Walker, and W. Alex Gray

Cardiff School of Computer Science, Cardiff University,
Queens Building, Newport Road, Cardiff, UK
{shalil.majithia, david.w.walker, w.a.gray}@cs.cardiff.ac.uk

Abstract. We present a framework to facilitate automated synthesis of scientific experiment workflows in Semantic Grids based on high-level goal specification. Our framework has two main features which distinguish it from other work in this area. First, we propose a dynamic and adaptive mechanism for automating the construction of experiment workflows. Second, we distinguish between different levels of abstraction of loosely coupled experiment workflows to facilitate reuse and sharing of experiments. We illustrate our framework using a real world application scenario in the physics domain involving the detection of gravitational waves from astrophysical sources.

1 Introduction

Grid computing is emerging as a key enabling infrastructure for "flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions, and resources" [38,39]. Scientific communities, including biology and physics, are embracing Grid technologies to manage and process large datasets, execute and share scientific experiments and simulations. Additionally, applications to process data intensive scientific experiments are no longer developed as monolithic code; instead autonomous services are combined together as required into a workflow to process the data [3,4]. In this context, service-oriented Grids can provide significant new capabilities to scientists by facilitating the composition and execution of experiments from a large pool of available services. Current manual composition techniques [12,13,14,15] are of limited value in dynamic and complex service-rich environments like the Grid for several reasons. First, the potential number of suitable services in Grid environments can be extremely large. It would be unrealistic to expect a user to be able to locate a suitable service. Second, the user is expected to ensure that the data being passed to services are semantically and syntactically correct. Finally, it is possible there may be no single service which can provide the functionality required. To resolve these problems and facilitate automated composition, mechanisms are needed to specify and manipulate rich descriptions of available resources. One promising approach is the Semantic Grid [40,41,42]. The Semantic Grid concept represents the use of Semantic Web technologies in Grid computing and promises the seamless interoperability of autonomous, heterogeneous, distributed applications across the

Grid by enabling the automation of many tasks for example discovery, selection, composition, and execution of workflows. Current automated composition techniques [4,6,17,31,7,8,9,10,11,18,33,34,35,36,37] typically make assumptions that render them of limited value in complex dynamic environments, for example they do not address issues like autonomy, adaptation, reuse, and sharing of workflows. This implies a need for an autonomic approach to workflow composition. In such an approach, the service composition process should be able to dynamically configure services in a way that can satisfy a service request. Autonomic models are suited for Grid environments where services are constantly added, modified or removed. By facilitating autonomic service composition, scientists can:

- Easily and efficiently discover a service that is more suitable to their needs.
- Focus on the conceptual basis of their experiments rather than understand the low level details of locating services.
- Easily create and share high quality complex workflows with varying levels of abstraction.

In this paper, we present a novel framework for the autonomic composition of experiment workflows in Semantic Grids. The framework presented in this paper has a number of important features. First, we distinguish between different levels of abstraction of loosely coupled workflow representations. Second, the framework uses a dynamic fault tolerant composition algorithm which adapts to the available resources. Third, the framework allows the user to specify and refine a high level objective. As a consequence, we propose that our framework:

- has a higher probability of successfully generating executable workflows compared to existing automated composition algorithms.
- allows users to share workflows with different levels of granularity.
- allows users to specify and refine high level objectives.
- is domain independent.

The remainder of this paper is structured as follows. First we review related research in the area of workflow composition in service-oriented architectures (Section 2). Second, we present an overview of the framework (Section 3). Third, we examine the main components of the framework (Sections 4 - 7). Fourth, we present an example application and evaluate our framework (Section 8). Finally, we conclude by discussing future directions.

2 Background

Workflow composition in Grids and more generally, in Service-Oriented Architectures has been the focus of much research recently, although the composition problem has been actively investigated in the database and agent communities [23-26]. Workflow composition is an active field of research and we review the major frameworks that are most closely related to our research. Workflow composition can be broadly classified into three categories: manual, semi-automated and automated composition. Manual composition frameworks [12, 13, 15, 27] expect the user to generate workflow scripts either graphically or through a text editor, which are then submitted to a workflow execution engine. Triana [12, 13] is used in astrophysics community and provides a graphical user interface, allowing the user to select the service required from a toolbox and "drag-and-drop" onto a canvas. The services are retrieved from UDDI using a simple keyword search. Additionally, Triana allows the composition of services with locally available tools. The composed graph can then be distributed over a P2P or Grid network for execution. BPWS4J [15] provides an Eclipse plug-in to allow the user to compose a graph at the XML level. This composed graph, along with a WSDL document for the composite service, is submitted to the execution engine. Self-Serve [27] allows the user to build the workflow by locating the services required by using the service builder. The service builder interacts with UDDI to retrieve service meta-data. The composed graph, an annotated state chart, is then executed using a P2P based execution model. An interesting feature of this system is the use of a service container that aggregates services offering the same functionality. At run time, the service container selects the actual service based on membership modes and a scoring service. These systems have several drawbacks. First, the discovery and selection of services is un-scalable as the number of services increases. Second, they require the user to have low-level knowledge, e.g. in the case of BPWS4J, the user is expected to set up a workflow at the XML level. Although Triana provides a graphical drag and drop interface, this is not feasible for a large workflow. Third, if the service is no longer available, the execution will fail, although in the case of Self-Serve, the service container would probably substitute another functionally-equivalent service. Also, recent research [19] has 'extended' BPWS4J to allow runtime selection and substitution of services.

Semi-automated composition techniques [4, 6, 17, 31] are a step forward in the sense that they make 'semantic suggestions' for service selection during the composition process; the user still needs to select the service required from a shortlist of the appropriate services and link them up in the order desired. The myGrid project [43] is one of the first Semantic Grid projects and aims to provide a problem-solving workbench for biologists. Specifically, myGrid allows scientists to compose, store and execute workflows. The scientist browses a registry and selects a workflow template. This template is then submitted to an enactment engine which then either asks the user to select actual instances of the workflow components or automatically selects services based on user's personal preferences. The myGrid project is also investigating other related issues like provenance. Sirin et. al. [17] propose a system that provides service choices which are semantically compatible at each stage. The generated workflow is then executed. Cardoso and Sheth [4] propose a framework which provides assistance to the user by recommending a service meeting the user's needs. This is done by matching the user-specified Service Template (ST) with the Service Object (SO). Chen et. al. [6] outline a knowledge-based framework which provides advice as the user constructs a workflow. The system allows the user to store workflows, hence facilitating reuse. Although these systems solve some of the problems of manual composition frameworks, they are still un-scalable as the user is expected to browse a registry to select the appropriate template or service. The filtering process may provide numerous matching services for the user to select from. For example in the myGrid project, the user is expected to select the actual instance of the service desired, although an automated workflow harmonization mechanism is envisaged. Additionally, there are few, if any, fault-handling mechanisms built into these systems. For example in Cardoso and Sheth, if a ST fails to match an SO, composition will fail. Similarly in Sirin et. al., Chen et. al., and myGrid project the composed workflow is sent to the enactment engine for execution. At this stage, if a service becomes unavailable, workflow execution will fail. Finally, except for Chen et al., there is no support for generating workflows of differing levels of granularity.

Automated composition techniques [7-11, 18, 33-37] automate the entire composition process by using AI planning or similar technology. McIlraith and Son [10] address the automated composition problem by proposing an agent-based Web services composition framework which uses generic procedures and semantically marked up services to guide composition. The Agent Broker acts as a gateway to Web services and is responsible for selecting and invoking the services. The framework assumes the existence of a generic procedure. In the absence of one, composition cannot proceed. Additionally, if the Agent Broker cannot match a service, execution will fail. The SWORD toolkit [11] automates service composition by using rule-based service descriptions. The user specifies the initial and final state facts. Based on this, the planner attempts to put together a chain of services that can satisfy these requirements. The user is expected to be able to specify the state facts. More importantly, there is no automated service discovery mechanism built in. As a result, composition will fail if the service required cannot be found. Also, composition is based on specific implementations of services; this makes it difficult to share workflows as the service may no longer be available. An important component of automated composition is the discovery of the services required. Research in this area has focused on the use of DAML-S to describe service capabilities. A matchmaker engine then compares the DAML-S description of the service requested with those of the services advertised [20, 21]. This work is extended by Sycara et. al. [16] to propose a preliminary discovery and execution framework based on AI planning technology, the DAML-S Matchmaker and the DAML-S Virtual Machine. The framework assumes a composition plan is available and hence concentrates on identification of suitable services and execution of the plan. Additionally, the framework does not incorporate the use of a workflow repository. Hence, a workflow has to be recomputed each time a request is received. Also, the framework does not distinguish between executable and non-executable workflows; all workflows are based on implemented services available. As a result, the workflows are not reusable or cannot be shared as there is no guarantee that any component service will be available in the future. Sheshagiri et. al. [8] propose a framework with two key features: use of DAML-S to describe service capabilities and a planner to generate simple workflow graphs using a backward chaining algorithm. A very similar framework is proposed by Wu et al., [7] that uses SHOP2, an HTN planning system, to generate workflows. The Pegasus project [9, 33-37] used to generate workflows in high energy physics applications proposes two workflow generation systems. The Concrete Workflow Generator (CWG) maps an abstract workflow onto an executable workflow. The second system (ACWG) uses AI planning technology to generate workflows in Grid environments. An interesting feature of this work is the distinction between abstract and concrete workflows. These frameworks suffer from the same drawbacks: there is no fault handling mechanism - if a service implementation is not available, execution will fail. Additionally, the frameworks proposed do not include a workflow repository which would make it possible to reuse previously constructed workflows. Further, there is no distinction between abstract and concrete workflows. This makes it difficult to share workflows. Although the Pegasus system does make this distinction, it does not provide mechanisms to expose the abstract and the concrete workflows as services. Finally, Koehler and Srivastava [29] point out several problems with current AI planning technology when applied to the service composition problem. Laukkanen and Helin [28] put forward a model which includes simple fault handling mechanisms. The model assumes that a workflow already exists in a repository and the user selects the workflow required. This workflow is then validated to ensure that all the participating services are available. If a service is not available, a matchmaker is used to retrieve similar services. Interestingly, if a single alternative cannot be found, the composer agent attempts to put together a simple chain of service which can provide the functionality. The workflow is then rewritten to include these new services and their network locations and sent to the execution engine. The drawbacks of this model are as follows: First, the model assumes a workflow is available. There is no mechanism for the user to create one. This is a severe drawback as it completely side-steps the issue of composition. Second, workflows are stored with the network locations of the services encoded within the workflow. This means that when a workflow is reused and all the participating services are available, there is no option for the user to select alternative implementations of the service based on changed optimization criteria. The IRS-II [18] framework provides an infrastructure which can be used to publish, compose and execute Semantic Web services. The system distinguishes between a task (generic description of the task to be solved) and a Problem Solving Method (an abstract, implementation-independent description of how to accomplish a task). A task may correspond to several PSMs. The user selects a task to be performed. A broker locates an appropriate PSM (which can satisfy the task) and then invokes the corresponding services. Although the system proposes an interesting distinction between a task and a PSM, this increase in the number of artifacts required merely enhances the probability that workflow composition will fail. First, a service developer has to manually associate a PSM to a service. This becomes un-scalable as the number of PSMs and services increase. Second, composition and/or execution will fail if a task, PSM or a service is not available. There appear to be no fault-handling mechanisms to tackle such a scenario.

In this section, we reviewed briefly several service composition frameworks, some of which are used to generate experiment workflows on the Grid. We identified gaps in existing systems; lack of an autonomic composition algorithm, and non-distinction between different abstract levels of workflows.

3 Framework Overview

In this section, we outline the general requirements for the framework and present the proposed architecture. In order to successfully generate scientific workflows, an automated workflow composition framework must adhere to certain basic requirements.

- High degree of fault-tolerance: The framework must be able to handle common fault scenarios. There is no guarantee in a Grid environment that a particular service will be available at some particular time. In this case, there should be intelligent mechanisms to discover and invoke another service, or compose services which provide the same functionality. Similarly, in case a rulebase is not available for a particular objective, there must be mechanisms to chain services to achieve the objective. We define fault tolerance to mean the non-availability of the service and exclude other possible faults, for e.g. a service returns an invalid response, at this time.
- Workflow granularity: The framework should support mechanisms to allow users to generate workflows of varying levels of granularity. For example, abstract and concrete workflows. Abstract workflows specify the workflow without referring to any specific service implementation. Hence all services are referred to by their logical names. A concrete workflow specifies the actual names and network locations of the services participating in the workflow. An abstract workflow allows users to share workflows without reference to any specific service implementation. This is particularly useful as there is no guarantee of the availability of any service in Grids. On the other hand, a concrete workflow could be useful for provenance purposes. Additionally, workflows of differing levels of granularity should be loosely coupled, i.e. with minimum interdependence.
- Specify and refine high-level objectives: The framework should support mechanisms to allow users to specify and dynamically refine a high-level objective which is then translated into a workflow. It should be possible to carry out "what-if" analysis in an efficient manner with only the changed sub-graphs being recomposed.
- User-specified optimization criteria: The framework should provide a mechanism which allows users to specify the workflow composition/execution optimization criteria. For example, the user may want to minimize the total runtime of the workflow or minimize the use of expensive resources.
- Scalable: The framework should be scalable to a large number of services.
- Domain independent: The framework should be as generic as possible in order to allow its use within any domain.

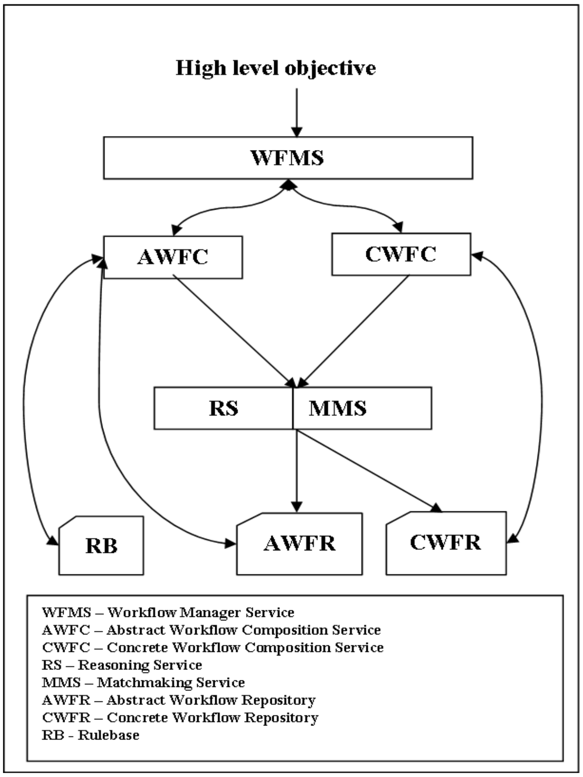
High level objective
WFMS
AWFC
CWFC
RS
MMS
RB
AWFR
CWFR
WFMS – Workflow Manager Service
AWFC – Abstract Workflow Composition Service
CWFC – Concrete Workflow Composition Service
RS – Reasoning Service
MMS – Matchmaking Service
AWFR – Abstract Workflow Repository
CWFR – Concrete Workflow Repository
RB - Rulebase

Fig. 1. The Framework Architecture.

- Other important requirements include security and ability to record and reason over provenance.

Our proposed framework provides the following:

- Support for dynamic fault-tolerant, adaptive workflow composition mechanism
- Support for generating, storing, and reusing workflows with differing levels of granularity.
- Support for specification and refinement of high-level user objectives.

The framework architecture is shown in Figure 1. Everything is a service. Services are described and invoked based on their descriptions. The key bootstrap operation is the location of the Workflow Manager Service (WFMS). The WFMS puts together the framework required based on available services and user-specified preferences. The framework consists of two core and five supporting services: Abstract Workflow Composer (AWFC), Concrete Workflow Composer (CWFC), Reasoning Service (RS), Matchmaker Service (MMS), Abstract Workflow Repository (AWFR), Concrete Workflow Repository (CWFR) and the

Rulebase (RB). The WFMS coordinates the entire process and manages the flow of messages between the components. The AWFC is a service which accepts an incoming user-specified high-level goal and transforms this into an abstract workflow. At this level, all tasks and their inputs and outputs are referred to by their logical names. The AWFC will typically query the AWFR to ascertain if the same request has been processed previously. If so, the abstract workflow will be returned to the Manager Service. If not, a request will be made to the Reasoning Service to retrieve a process template from the Rulebase which can satisfy the request. If a process template is not available, an attempt will be made to retrieve a combination of tasks that provide the same functionality based on the inputs, outputs, preconditions and effects. The same process will apply to all constituent services. An abstract workflow is generated and returned to the Manager Service. The CWFC Service accepts an abstract workflow and attempts to match the individual tasks with available instances of actually deployed services. If the matching process is successful, an executable graph is generated and returned to the Manager Service. If not, a request will be made to the Reasoning Service and the Matchmaker Service to retrieve a combination of services that can provide the required functionality. The AWFR and CWFR Services are domain specific services which wrap and provide access to the workflow repositories. The repositories store abstract and concrete workflows respectively. Sections 4 to 7 describe each of these components in more detail.

4 Abstract Workflow Composition

As introduced above, the AWFC Service generates an abstract workflow from a high-level objective. The AWFC uses a dynamic adaptable algorithm (Fig. 2) which tries to build an abstract workflow by using at least three different sources of information.

- AWF Repository. The AWF Repository stores semantically annotated descriptions of services and workflows. The AWFC queries the AWFR Service to ascertain if the same workflow has been processed previously. This is done by semantically matching the workflow name by referring to an ontology. If a match is found, the AWF is returned.
- Rulebase. If no match is found, the AWFC queries a rule base to retrieve a rule which can provide the template to achieve this goal. Given this rule, the AWFC again queries the AWFR, but this time an attempt is made to semantically match the inputs and outputs required to those of the abstract workflows in the repository. If an AWF is matched, this is returned. If the matching does not succeed, the AWFC analyzes and splits the rule into component rules. This is done recursively until an atomic rule is reached. The AWFC will then query the AWFR to ascertain if any AWFs match the rule.
- Chaining Services. The AWFC uses the Reasoning Service (see Section 6) to create a chain of services that when put together can fulfil the user objective.

```
AWFC(name){
   retrieveFromAWFR(name);
   if (!found) {
      retrieve rule from rulebase;
      retrieveFromAWFR(input,output);
      if (!found) {
         split rule into smallest rule;
         for (each rule) {
            get rule output;
            get rule input;
            retrieveFromAWFR(input,output);
            if (!found) {
               chain services(input,output);
            }
         concatenate;
         }
      }
   }
   return AWF
}
```

Fig. 2. AWFC algorithm

5 Concrete Workflow Composition

The CWFC uses a dynamic adaptable algorithm to match each services in the abstract workflow with an instance of an executable service available on the network at that time. It does this in two different ways:

– Matchmaking: Each abstract service is passed to the Matchmaker Service (see Section 7) which then attempts to retrieve a semantically matching service available on the network. Depending on the degree of the match, the CWFC Service may accept the service. The degree of match may vary from exact to plug-in.
– Chaining Services: The CWFC uses the Reasoning Service (see Section 6) to chain together services which can provide the same functionality. It is possible that a service implementation may not be available. In this case, the AWFC is invoked through the WFMS, and asked to provide alternative AWFs for that particular sub-graph. This provides a high degree of fault tolerance.

Matching is done in two stages. First, a semantic match is made. If this is successful, the number and the types of the input and output messages are compared: if they are the same, the match is successful, else an attempt is made to chain services starting from the output desired.

6 Reasoning Service

The Reasoning Service provides support to the two core composer services. Specifically, this service provides a back-tracking algorithm to produce a chain of services which can provide the output required using the given input. More specifically, the following steps are performed:

- For each service available, find a service that matches the output of the service requested. Let one such service be Sn.
- Ascertain the input of Sn. Find a service that can generate the input for Sn. Let this service be Sn-1.
- This process is iterated until the input of the service Sn-x matches the input of the service requested.
- Formulate the workflow which specifies the order of execution of the components S1 to Sn.

An interesting feature of our chaining algorithm is that it uses the Matchmaking Service to match the inputs and outputs of the services. As a result, it is not essential that matching be exact; plug-in matches are acceptable. This makes our chaining algorithm more robust and with a higher rate of success than similar other algorithms.

7 Matchmaking Service

The Matchmaking Service (MMS) provides an intelligent matchmaking mechanism. Its core functionality is to match two services based on the inputs, outputs, pre-conditions and effects (IOPEs). The Matchmaking Service is used by the Reasoning Service to match inputs and outputs of the service that it is chaining. Additionally, the MMS is used by the CWFC Service to match abstract services to concrete services. Our algorithm is based on that proposed by Paolucci et. al. [20]. However, only exact and plug-in matches, as defined by Paolucci et al., are calculated as there is no guarantee that a subsumes match would be able to provide the required functionality. As a consequence, we would expect the algorithm to be more suitable and efficient for use in automated composition frameworks.

8 Implementation and Evaluation

In this section, we first describe the implementation of our framework. We then illustrate our approach by using a scenario from the physics domain and provide some preliminary results of performance evaluation experiments.

All the main components are implemented as Web services using the Axis server to expose them. We use existing Semantic Web tools, like RDF, OWLS, and reasoning mechanisms to provide the basic representation and reasoning

```
<profileHierarchy:SignalProcessing rdf:ID="FFT">
  <profile:input>
    <profile:ParameterDescription rdf:ID="FFTInput">
      <profile:restrictedTo rdf:resource="Concepts.owl#VectorType"/>
    </profile:ParameterDescription>
  </profile:input>
  <profile:output>
    <profile:ParameterDescription rdf:ID="FFTOutput">
      <profile:restrictedTo rdf:resource="Concepts.owl#ComplexSpectrum"/>
    </profile:ParameterDescription>
  </profile:output>
</profileHierarchy:SignalProcessing>
```

Fig. 3. Snippet of OWL-S Profile for FFT

technology. Services are described using OWL-S. The Reasoning and Matchmaking Services use DQL/JTP server to carry our subsumption reasoning. The repositories are currently implemented as flat files containing the OWL-S profiles and process models (see figure 3). The WFMS forms the core of the framework. It locates and coordinates all the other components required to form the framework. The AWFC Service generates an abstract workflow as a DAML-S Process Model. This abstract workflow is passed to the CWFC Service, which then matches all the services taking part in the workflow with available implementations. Finally, a BPEL4WS document containing the network locations and protocol details of all the tasks is generated and this can now be submitted to any BPEL4WS engine for execution. To illustrate the main features of our framework, we present the following scenario.

Compact binary stars orbiting each other in a close orbit are among the most powerful sources of gravitational waves. As the orbital radius decreases, a characteristic chirp waveform is produced whose amplitude and frequency increase with time until eventually the two bodies merge together. Laser interferometric detectors such as GEO600, LIGO, VIRGO should be able to detect the waves from the last few minutes before collision. Interferometric gravitational wave detectors are inherently broadband, sensitive to radiation coming from almost all directions in the sky, and output data at rates in excess of several megabytes per second. Consequently, searching for gravitational wave signals buried in the detector output and identifying the sources is a challenging task. To analyze this data, thousands of templates, representing the theoretical knowledge of relativistic binary systems, are correlated with the signal using fast correlation by a technique known as matched filtering. This implementation uses approximately 50 separate services connected together to form a workflow. The AWFC generates a DAML-S Process model referring to the services by their logical names, for example FFT, Conjugate, Scaler, OneSide, and Wave. These are then matched to actual implementations available on the network.

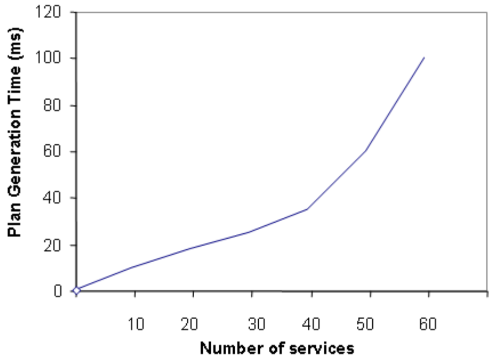

Fig. 4. Plan Generation Times.

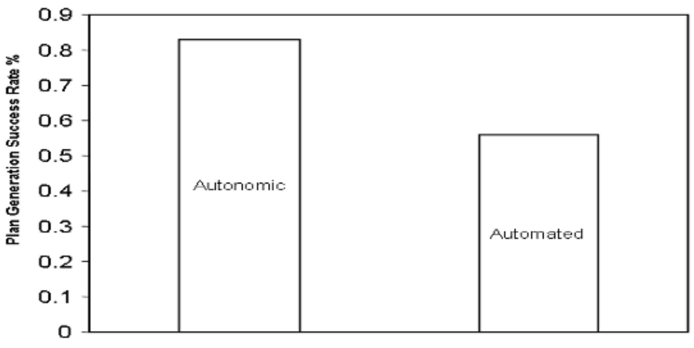

Fig. 5. Plan Generation Success Rates.

The purpose of our experiments is two-fold: to assess plan generation times and success rates. We present here preliminary results. We ran our experiments on a Windows machine with a 1.4GHz Intel processor and 512Kb RAM. We first evaluate the time for generating concrete workflows (see Figure 4). There is an exponential relationship between the number of services in a plan and the time required to match and compose an executable graph. We also looked at the workflow generation success rates(see Figure 5). We compared our framework with a simple automated back-tracking composition algorithm. We used the same process ontologies and composition requests for both of these techniques. Based on 50 requests, the results show that our framework had a success rate of 80% while the back-tracking algorithm had a success rate of about 60%. This confirms our expectation that our autonomic framework has a higher probability of successfully generating workflows. Further experiments are currently being done to ascertain the range over which this results apply.

9 Conclusion

In this paper, we propose a framework to facilitate automated composition of scientific experiment workflows in Semantic Grids made up of a Manager Service and other supporting services, including an abstract and a concrete workflow generator. We have described these components in detail and outlined their interactions. We have also described our implementation and its use within the physics domain. The important features of our approach are: an autonomic workflow generation algorithm and distinction between different levels of abstraction of the workflow in order to allow reuse and sharing. These features ensure that our framework has a higher possibility of successfully generating workflows compared to other similar algorithms. Future work involves the use our model in other application domains. We also plan to integrate user-specified optimization criteria to guide the generation of workflow. Finally, we intend to investigate the use of user specified policies to guide decisions which need to made when composing services.

References

1. Casati, F., Shan, M., and Georgakopoulos, D. 2001. E-Services - Guest editorial. The VLDB Journal. 10(1):1.
2. Tsalgatidou, A and Pilioura, T. 2002. An Overview of Standards and Related Technology in Web services. Distributed and Parallel Databases. 12(3).
3. Fensel, D., Bussler, C., Ding, Y., and Omelayenko, B. 2002. The Web Service Modeling Framework WSMF. Electronic Commerce Research and Applications.1(2).
4. Cardoso, J. and Sheth, A. 2002. Semantic e-Workflow Composition. Technical Report, LSDIS Lab, Computer Science, University of Georgia.
5. Paolucci, M., Sycara, K., and Takahiro Kawamura. 2003. Delivering Semantic Web Services. In Proc. Of the Twelfth World Wide Web Conference.
6. Chen, L, Shadbolt, N.R, Goble, C, Tao, F., Cox, S.J., Puleston, C., and Smart, P. 2003. Towards a Knowledge-based Approach to Semantic Service Composition. 2nd International Semantic Web Conference
7. Wu, D., Sirin, E., Hendler, J., Nau, D., and Parsia, B. 2003. Automatic Web Services Composition Using SHOP2. Twelfth World Wide Web Conference.
8. Sheshagiri, M., desJardins, M., and Finin, T. 2003. A Planner for Composing Service Described in DAML-S. Workshop on Planning for Web Services, International Conference on Automated Planning and Scheduling.
9. Deelman, E., J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, Cavanaugh, R. and Koranda, S. Mapping Abstract Complex Workflows onto Grid Environments. Journal of Grid Computing. Vol. 1, 2003
10. McIlraith, S. and Son, T.C. 2002. Adapting golog for composition of semantic web services. In Proc. of the 8th International Conference on Knowledge Representation and Reasoning (KR '02), Toulouse, France.
11. Ponnekanti, S. R., and Fox, A. 2002. SWORD: A Developer Toolkit for Web Service Composition. In Proc. Of the Eleventh International World Wide Web Conference, Honolulu.
12. Taylor, I., Shields, M., Wang, I., and Philp, R: Grid Enabling Applications Using Triana, Workshop on Grid Applications and Programming Tools, June 25, 2003, Seattle. In conjunction with GGF8

13. Taylor, I., Shields, M., Wang, I., and Philp, R: Distributed P2P Computing within Triana: A Galaxy Visualization Test Case. To be published in the IPDPS 2003 Conference, April 2003
14. Mayer, A., McGough, S., Furmento, N., Lee, W., Newhouse,S., and Darlington, J: ICENI Dataflow and Workflow: Composition and Scheduling in Space and Time In UK e-Science All Hands Meeting, p. 627-634, Nottingham, UK, Sep. 2003
15. IBM Alphaworks, BPWS4J, http://www.alphaworks.ibm.com/tech/bpws4j [12.01.2004]
16. Sycara, K., Paolucci, M., Ankolekar, A., and Srinivasan, N.: Automated Discovery, Interaction and Composition of Semantic Web Services, Journal of Web Semantics, Volume 1, Issue 1, December 2003
17. Sirin, E., Hendler, J., and Parsia, B: Semi-automatic composition of web services using semantic descriptions. In Web Services: Modeling, Architecture and Infrastructure Workshop in conjunction with ICEIS 2003
18. Motta, E., Domingue, J., Cabral, L. and Gaspari, M: IRS-II: A Framework and Infrastructure for Semantic Web Services. 2nd International Semantic Web Conference (ISWC2003) 20-23 October 2003, Sundial Resort, Sanibel Island, Florida, USA
19. Mandell, D.J., and McIlraith, S.A: A Bottom-Up Approach to Automating Web Service Discovery, Customization, and Semantic Translation. In The Proceedings of the Twelfth International World Wide Web Conference Workshop on E-Services and the Semantic Web (ESSW '03). Budapest, 2003.
20. Paolucci, M., Kawamura, T., Payne, T., Sycara, K: Semantic Matching of Web Services Capabilities. Proceedings of the 1st International Semantic Web Conference (ISWC), pp. 333-347, 2002.
21. Lei Li and Ian Horrocks. A software framework for matchmaking based on semantic web technology. In Proc. of the Twelfth International World Wide Web Conference (WWW 2003), pages 331-339. ACM, 2003.
22. Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web, Scientific American, May, 2001.
23. Parent, C. and Spaccapietra, S: Issues and Approaches of Database Integration. Communications of the ACM 41(5): 166-178.
24. Kashyap, V. and A. Sheth: Semantic Heterogeneity in Global Information Systems: The Role of Metadata, Context and Ontologies, Academic Press.
25. Preece, A.D., K.Y. Hui, Gray, W.A., Marti, P., Bench-Capon, T.J.M., Jones, D.M., and Cui, Z: The KRAFT Architecture for Knowledge Fusion and Transformation. 19th SGES International Conference on Knowledge-based Systesm and Applied Artificial Intelligence (ES'99) , Springer, Berlin
26. Bayardo, R.J., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh and D. Woelk: InfoSleuth: Agent-Based Semantic Integration of Information in Open and Dynamic Environments. Proceedings of the ACM SIGMOD International Conference on Management of Data, ACM Press, New York. pp. 195-206.
27. Benatallah, B., Sheng, Q.Z., and Dumas, M.: The Self-Serv Environment for Web Services Composition, Jan/Feb, 2003, IEEE Internet Computing. Vol 7 No 1. pp 40-48.
28. Laukkanen, M., and Helin, H: Composing Workflows of Semantic Web Services.In AAMAS Workshop on Web Services and Agent-Based Engineering, 2003.

29. Koehler, J., and Srivastava, B: Web Service Composition: Current Solutions and Open Problems. ICAPS 2003 Workshop on Planning for Web Services, pages 28 - 35.
30. Medjahed, B., Bouguettaya, A., and Elmagarmid A: Composing Web Services on the Semantic Web. The VLDB Journal, Special Issue on the Semantic Web, Volume 12, Number 4, November 2003.
31. Stevens, R.D., Robinson, A.J., and Goble, C.A: myGrid: Personalised Bioinformatics on the Information Grid. Bioinformatics Vol. 19 Suppl. 1 2003, (Eleventh International Conference on Intelligent Systems for Molecular Biology)
32. Business Integration, http://www.bijonline.com/default.asp [12.01.2004]
33. Blythe, J., Deelman, E., Gil, Y., and Kesselman C: Transparent Grid Computing: a Knowledge-Based Approach. 15th Innovative Applications of Artificial Intelligence Conference (IAAI 2003), 2003.
34. Blythe, J., Deelman, E., Gil, Y., Kesselman, C., Agarwal, A., Mehta, G., and Vahi, K: The Role of Planning in Grid Computing, 13th International Conference on Automated Planning and Scheduling, 2003.
35. Blythe, J., Deelman, E., and Gil, Y: Planning for workflow construction and maintenance on the Grid. ICAPS 2003 Workshop on Planning for Web Services.
36. Deelman, E., Blythe, J., Gil, Y., and Kesselman, C: Pegasus: Planning for Execution in Grids., GriPhyN technical report 2002-20, 2002
37. Deelman, E., Blythe, J., Gil, Y., Kesselman C., Mehta, G., Vahi, K., Koranda, S., Lazzarini, A., Papa, M.A: From Metadata to Execution on the Grid Pegasus and the Pulsar Search. , GriPhyN technical report 2003-15
38. Foster, I. and Kesselman, C. eds., The Grid: Blueprint for a New Computing Infrastructure, Morgan Kaufmann, San Francisco, 1999.
39. Foster, I. et al., "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration," tech. report, Globus Project;
40. De Roure, D., Jenning, N.R., and Shadbolt, N., Research Agenda for the Semantic Grid: A Future e-Science Infrastructure. http://www.semanticgrid.org
41. The Grid: An Application of the Semantic Web by Carole Goble and David De Roure, 2002, www.semanticgrid.org
42. Semantic Web and Grid Computing by Carole Goble and David De Roure, 2002, www.semanticgrid.org
43. Stevens,R., Robinson, A., and Goble, C.A., myGrid: Personalised Bioinformatics on the Information Grid in proceedings of 11th International Conference on Intelligent Systems in Molecular Biology, 29th June–3rd July 2003, Brisbane, Australia, published Bioinformatics Vol. 19 Suppl. 1 2003, i302-i304