

Flexible Dynamic Binding in Agile Grid Development¹

Jovan Cakic, Richard F. Paige, Howard Chivers, Xiaocheng Ge,

John A. McDermid, and Jim Austin

Department of Computer Science, University of York, UK.

{cakic,paige,chive,xchge,jam,austin}@cs.york.ac.uk

Abstract

The eXGrid project focuses on the agile development of Grids, in order to more flexibly build dynamic virtual organizations. A key issue in supporting virtual organizations in Grids is a more flexible notion of dynamic binding. We report on experiments in achieving flexible dynamic binding in an agile development of a .NET-based Grid.

1. Introduction and Background

Grid computing is a model of system development based on virtual resource integration. Resources are to be delivered seamlessly and dynamically over the Internet and aim at supporting flexible collaboration and computation, in many cases on a national or international scale. However, Grid computing is still in early stages of development, and fully dynamic business interactions based on the metaphor of Virtual Organizations (VO) [Fos01] are not foreseen in the near future. Current Grid middleware does not yet provide the flexibility necessary to make sophisticated planning and scheduling decisions [Gil04] at the level required by this metaphor. Advances are needed in the areas of service discovery, composition, and orchestration.

The goals of the *eXGrid* project, which investigates agile approaches to design of Grid-based systems, include creating a high-level conceptual model of Grid-based systems, and an agile development methodology that applies the model to concrete development problems. The abstract model of Grids that we have proposed [Pai04] provides a simplified interface to Grid middleware, a flexible definition of service, and supports many of the critical usage scenarios that have substantial business value. The model was constructed by applying agile development principles [Agi04] and practices, particularly *simplicity*, and as such it is minimal. While some issues and usage scenarios may not be supported by the model, the agile methodology that we will deliver will bridge the gap between the model and concrete scenarios.

This paper introduces one of the steps – a more flexible notion of dynamic binding – that is being taken to strengthen the conceptual model, bringing it to a sufficient level of maturity so that it supports most, if not all, of the scope of Grid computing and the VO metaphor. The level of dynamic binding that we require in fully supporting the business VO metaphor goes beyond that of most existing Grids – i.e., binding a service implementation to an interface at run-time, or running a script – we support service discovery, management, and negotiation at the most flexible level at run-time.

The detailed plan of work for eXGrid includes several stages and is currently focusing on two key aspects of Grids: flexible construction of VOs; and security. The latter is reported on elsewhere [Chi05].

2. Grids vs. Service-Oriented Architectures and the Need for Dynamic Binding

A typical Service-Oriented Architecture (SOA)-based system targets a particular business process, with a predictable workflow and well-known resources. Interfaces and message formats are more or less fixed at design-time, which means that information models of such systems are stable. The Grid computational environment, however, is characterized by entity autonomy, heterogeneity, and distribution. It is an environment in which *a priori* agreements regarding engagement cannot be assumed. Trading partnerships must be dynamically selected, negotiated, procured, and monitored.

¹ Supported by EPSRC GR/S64226/01.

In this model of Grid-based systems, business processes are implemented by VOs, and change on per-job basis. In a Grid, Web Services are external resources that are mostly unknown at design-time. The resource configurations are transient, dynamic and volatile. They are anticipated to be ad-hoc as service consortia have no central location, no central control, and only the most minimal existing trust relationships [Gob04]. Therefore, a particular VO configuration can be seen as a SOA, but it has to be automatically created at run-time in response to a job request, and using currently available resources. This emphasizes the importance of mechanisms for more flexible dynamic binding and information sharing in Grid-based systems. To achieve flexible assembly of Grid components and resources requires not just a service-oriented model, but also information about functionality, availability, and interfaces of components. Thus we will need formal, expressive, and extensible ways of describing Grid components, as well as flexible mechanisms for supporting decisions based on such descriptions.

Flexible run-time binding to Web Services includes the following steps: binding to an interface at a particular access point; binding to a particular method in a selected service interface (this should use method descriptions, not names, in truly flexible run-time binding); and binding to data. The run-time binding problem thus has two aspects: definition of a Grid information model, and a mechanism to implement run-time binding based on such an information model.

In more detail, dynamic run-time binding to Web Services is a process that includes the following steps (illustrated in Fig. 1):

1. *Binding to an interface* (Web Service implementation) at a particular access point. Service discovery is performed via UDDI (Universal Description, Discovery and Integration) searches, in order to select the most appropriate service offer and to locate the corresponding Web Service access point. The access point provides a WSDL interface specification, which can be used for run-time creation of a Web Service proxy.
2. *Binding to a method* in the selected Web Service interface. To reduce interdependencies and to improve flexibility, method descriptions, not method names, should be used in the process of

dynamic binding. As a result, method discovery in Web Services interfaces is required as well. The appropriate method will be selected based on its description, which is associated with the method name in the corresponding WSDL service specification. Once the method name has been identified, a reflection mechanism can be used on the Web Service proxy to invoke the target method.

3. *Binding to data.* When the invoked method returns a reply message, it is necessary to locate the required data in it using XML tags. If returned data is not of interest, this step can be skipped.

In our model, every activity in a process workflow implemented by a Web Service will have a static binding to the Binding Workflow, which refines the activity according to its semantic specification (binding requirements). The details of that refinement are shown in Fig. 2.

The Binding Workflow includes the following components, each implemented as a Web Service:

- *Service Broker*, which is a Web Service that performs discovery and selection of service providers for Grid jobs according to obtained functional requirements and Grid management policies. It interacts with a UDDI registry and looks for services that satisfy requirements.
- *SLA Broker*, which is a Web Service that tries to establish a formal relationship with selected Web Services. It requests service offers from selected Web Services and negotiate SLAs according to give non-functional requirements.
- *Dynamic Port* performs dynamic invocation of selected Web Services.

3. Case Study in Flexible Dynamic Binding

In order to be able to investigate more flexible run-time binding mechanisms, as well as the amount and form of shared semantic information needed to support it, we are using a prototype implementation based on Microsoft .NET Framework and BizTalk Server 2004. This offers a seamless integration of technology essentials for building SOA-based distributed systems; moreover Visual Studio .NET also supports Web Services Policy Framework [Baj04].

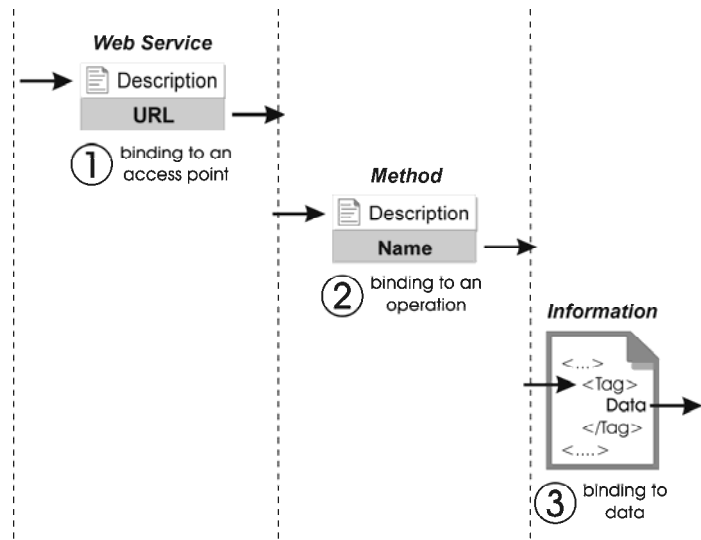


Fig.1: Steps in flexible run-time binding

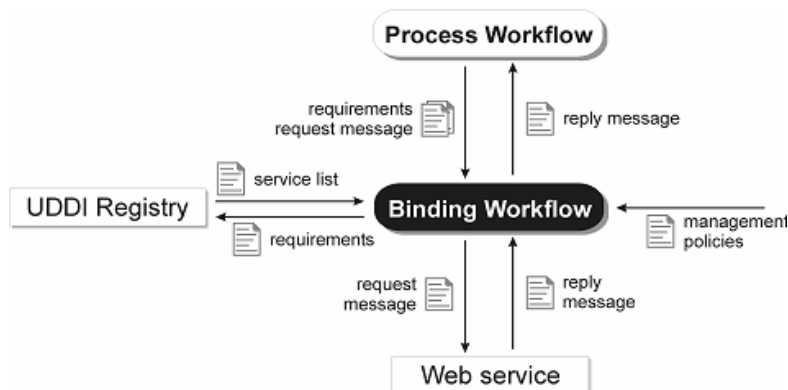


Fig.2: Refinement of activity

The Grid system under construction is called RealWizard, which will enable customers to purchase real estate properties. The Customer will be able to specify a set of requirements for property and mortgage selection according to his financial power and preferences, and the system will offer a choice of corresponding properties together with financial support. The purpose of the system is to abstract a very complex process of acquiring real estate properties by offering the complete service in just few mouse clicks. The Customer's participation is required at the beginning of a job, to set the requirements, and towards the end, when they need to select a concrete offer and make a commitment. All the other steps are automated. The development of RealWizard has three stages.

1. In the initial stage, we plan to embed metadata in WSDL 1.1 and UDDI 2.0

based specifications, which are currently implemented standards. Shared information, needed to define semantics of the metadata, will be placed in a custom-made dictionary.

- In the next stage, we plan to empower our concept by using WS-Policy Framework. This essentially means that, in addition to embedded metadata in WSDL and UDDI specifications, we will use external policy specifications associated with pre-implemented Web Services. A dictionary of shared information will be standardized according to WS-Policy as well.
- In the final stage we will investigate expressive power and complexity of Semantic Web concepts and ontologies regarding their usage for Web Services description in Grid-based systems.

A prototype of RealWizard, including some of the functionality above, has been constructed and development is continuing, exploring WS-Policy and also ontological ideas from the Semantic Web.

Engineering and its Applications 2004, CNAM, Paris, France, December 2004.

4. Conclusions

We have outlined parts of the *eXGrid* project, focusing on flexible and agile development of Grids. A more flexible dynamic binding mechanism – integrating ideas from policy management and the Semantic Web – is needed to fully support the VO metaphor. This in turn leads to additional complexity and security issues, and thus our parallel work on security will link in to this to help improve the dependability of the resulting model and Grids.

5. References

- [Agi04] The Agile Manifesto, www.agilemanifesto.org, 2004.
- [Baj04] S. Bajaj et al. *Web Services Policy Framework (WS-Policy)*, specification, <ftp://www6.software.ibm.com/software/developer/library/ws-policy.pdf>, 2004
- [Chi05] H. Chivers, R.F. Paige, and X. Ge. Agile Security using an Incremental Security Architecture. In *Proc. Extreme Programming 2005*, LNCS, Springer, June 2005.
- [Fos01] I. Foster, C. Kesselman, S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations, *Journal of Supercomputer Applications* 15(3):200-222, 2001.
- [Gil04] Y. Gil, E. Deelman, J.Blythe, C. Kesselman, H. Tangmunarunkit, Artificial Intelligence and Grids: Workflow Planning and Beyond, *IEEE Intelligent Systems* 19(1), pp. 26-33, 2004.
- [Gob04] C. Goble and D. Roure, The Semantic Grid: Myth Busting and Bridge Building, In *Proc. 16th European Conference on Artificial Intelligence (ECAI-2004)*, Valencia, Spain, 2004.
- [Pai04] R.F. Paige, J. Cakic, X. Ge, and H. Chivers. Towards Agile Re-Engineering of Dependable Grid Applications. In *Proc. International Conference on Software*