Automatic Deployment and Interoperability of Grid Services

G. Kecskemeti, **Y. Zetuny**, G. Terstyanszky, T. Kiss, P. Kacsuk, S. Winter

Centre of Parallel Computing, University of Westminster,
115 New Cavendish Street,
London W1W 6UW United Kingdom
e-mail:gemlca-discuss@cpc.wmin.ac.uk

Abstract

The Grid Execution Management for Legacy Code Architecture (GEMLCA) enables exposing legacy applications as Grid services without re-engineering the code, or even requiring access to the source files. The integration of current GT3 and GT4 based GEMLCA implementations with the P-GRADE Grid portal allows the creation, execution and visualisation of complex Grid workflows composed of legacy and non-legacy components. However, the deployment of legacy codes and mapping their execution to Grid resources is currently done manually. This paper outlines how GEMLCA can be extended with automatic service deployment, brokering, and information system support. A conceptual architecture for an Automatic Deployment Service (ADS) and for an x-Service Interoperability Layer (XSILA) are introduced explaining how these mechanisms support desired features in future releases of GEMLCA.

1. Legacy Code Services for the Grid

The Grid requires special Grid enabled applications capable of utilising the underlying middleware and infrastructure. Most Grid projects so far have either developed new applications from scratch, or significantly re-engineered existing ones in order to be run on their platforms. This practice is appropriate in this context, where the applications are mainly aimed at proving the concept of the underlying architecture. However, as the Grid becomes stable and commonplace in both scientific and industrial settings, a demand will be created for porting a vast legacy of applications onto the new platform. Companies and institutions can ill afford to throw such applications away for the sake of a new technology, and there is a clear business imperative for them to be migrated onto the Grid with the least possible effort and cost. Grid computing is now progressing to a point where reliable Grid middleware and higher level tools will be offered to support the creation of production level Grids. A high-level Grid toolkit should definitely include components for turning legacy applications into Grid services.

The Grid Execution Management for Legacy Code Architecture (GEMLCA) [1] enables legacy code programs written in any source language (Fortran, C, Java, etc.) to be easily deployed as a Grid Service without significant user effort. GEMLCA does not require any modification of, or even access to, the original source code. A user-level understanding, describing the necessary input and output parameters and environmental values such as the number of processors or the job manager required, is all that is needed to port the legacy application binary onto the Grid.

In order to offer a user friendly application environment, and support the creation of complex Grid applications from building blocks, GEMLCA is integrated with the workflow oriented P-GRADE Grid portal [2]. Using the integrated GEMLCA – P-GRADE portal solution users can create complex Grid workflows from legacy and non-legacy components, map them to the available Grid resources, execute the workflows, and visualise and monitor their execution.

A drawback of the current solution is the static mapping of legacy components onto resources. Before creating the workflow the legacy application has to be deployed on the target site, and during workflow creation, but prior to its submission, the user has to specify the resource where the component will be executed. It would desirable to allocate resources dynamically at run-time, or to automatically deploy a legacy component on a different site in order to achieve better performance.

Figure 1 illustrates how GEMLCA can be extended with these functionalities. Instead of mapping the execution of workflow components statically to the different Grid sites, the abstract workflow graph created by the user is passed to a resource broker together with quality of

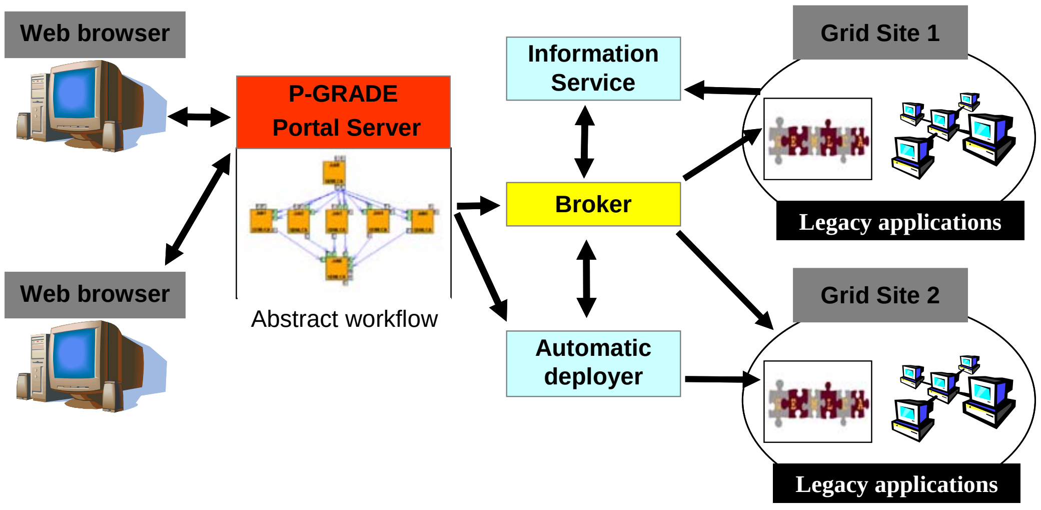

Figure 1 GEMLCA with Brokering, Information System and Automatic Deployment Support

service (QoS) requirements. The broker contacts an information service and tries to map different components of the workflow to different resources and pre-deployed services. If user QoS requirements cannot be fulfilled with the currently deployed services, or if the required service is not deployed on any of the resources, the broker contacts the automatic deployment service in order to deploy the code on a different site. As the sites can belong to different Grids with different middleware, policy and security standards, the deployer service should resolve these interoperability problems.

Unfortunately no currently existing information system, resource broker or deployment service can be directly used and integrated with GEMLCA to solve these problems. Significant research, extension and improvement of existing solutions are necessary. In this paper we concentrate on a subset of this complex architecture and propose a solution for the Automatic Deployment Service (ADS) and for an x-Service Interoperability Layer (XSILA).

2. Related Work

There are several research efforts aiming at automating the transformation of legacy code into a Grid Service. These approaches are either *invasive* or *non-invasive*. Both approaches are valid in different circumstances, depending on factors such as the granularity of the code, the assumed users and application area.

In the invasive approach, it is typically assumed that an application programmer, such as a biologist or chemist with some programming background but no Grid-specific knowledge, would like to build Grid enabled applications using specific software libraries. These libraries need to be wrapped using tightly-coupled code-wrapping technology that exposes low level functionality. Most of these solutions are based on the principles outlined in [3] and use Java wrapping in order to generate stubs automatically. One prominent example is represented by the work of researchers at University of Cardiff [4]. This solution is based on the semi-automatic conversion of program code into Java using Java Native Interface (JNI). After wrapping the native application with the Java-C Automatic Wrapper (JACAW), or the Simplified Wrapper and Interface Generator (SWIG), the MEdiation of Data and Legacy Code Interface tool (MEDLI) is used for data mapping to make the code available as part of a Grid workflow using Triana.

A different approach is represented by the non-invasive solutions, like GEMLCA. This method is relatively coarse-grained, in that the application does not allow visibility of low-level functionalities. The legacy code is provided as a black-box with specified input and output parameters and environmental requirements. Only the executable is available, and required, in this case, together with a user-level understanding of the application. This scenario is very common in both scientific and business applications when:

- the source code is not available,
- the program is poorly documented and/or the necessary expertise to do any modifications has long left the organisation,
- the application has to be ported onto the Grid within the shortest possible time and smallest effort and cost ,
- the functionalities are offered to partner organisations but the source is not.

Other non-invasive approaches are described in [5] and [6]. Although both solutions are similar in aims with GEMLCA, they have limited prototype implementations supporting only OGSI (Open Grid Services Infrastructure) type GRID middleware at the moment. GEMLCA, implemented both on top of GT3 and GT4, offers a more comprehensive solution, since it includes portal and workflow access, security solutions incorporating authentication, authorisation and security delegation mechanisms.
The aim of GEMLCA, similarly to the Triana-based solution of Cardiff University, is to provide a comprehensive and user-friendly environment for legacy code deployment and execution. This includes not only the core legacy transformation functionality, but also supporting the end-users with several tools like Grid portal, workflow engine, brokering or the automatic deployment facility presented in this paper.

3. Grid Execution Management for Legacy Code Architecture

The Grid Execution Management for Legacy Code Architecture (GEMLCA) enables legacy code programs written in any source language (Fortran, C, Java, etc.) to be easily deployed as a Grid Service without significant user effort. GEMLCA represents a general architecture for deploying legacy applications as Grid services without re-engineering the code or even requiring access to the source files. The high-level GEMLCA conceptual architecture is represented on Figure 2.
As shown in the figure, there are four basic components in the architecture:
The **Compute Server** is a single or multiple processor computing system on which several legacy codes are already implemented and available. The goal of GEMLCA is to turn these legacy codes into Grid services that can be accessed by Grid users.
The **Grid Host Environment** implements a service-oriented OGSA-based Grid layer, such as GT3 or GT4. This layer is a pre-requisite for connecting the Compute Server into an OGSA-built Grid.
The **GEMLCA Resource** layer provides a set of Grid services which expose legacy codes as Grid services.
The fourth component is the **GEMLCA Client** that can be installed on any client machine through which a user would like to access the GEMLCA resources.
The novelty of the GEMLCA concept is that it requires minimal effort from both Compute Server administrators and end-users of the Grid. The Compute Server administrator should install the GEMLCA Resource layer on top of an available OGSA layer (GT3/GT4). It is also their task to deploy existing legacy applications on the Compute Servers as Grid services, and to make them accessible for the whole Grid community. End-users do not have to do any installation or deployment work if a GEMLCA portal is available for the Grid and they only need those legacy code services that were previously deployed by the Compute Server administrators. In such a case end-users can immediately use all these legacy code services - provided they have access to the GEMLCA Grid resources. If they would like to deploy legacy code services on GEMLCA Grid resources they can do so, but these services cannot be accessed by other Grid users. As a last resort, if no GEMLCA portal is available for the Grid, a user must install the GEMLCA Client on their client machine. However, since it requires some IT skills to do this, it is recommended that a GEMLCA portal is installed on every Grid where GEMLCA Grid resources are deployed.
The deployment of a new legacy code service in current GEMLCA implementations assumes that the legacy application is already deployed and runs in its native environment on a Compute Server. The deployment with GEMLCA means to expose this legacy application as a Grid service. It is the task of the GEMLCA Resource layer to present the legacy application as a Grid service to the user, to

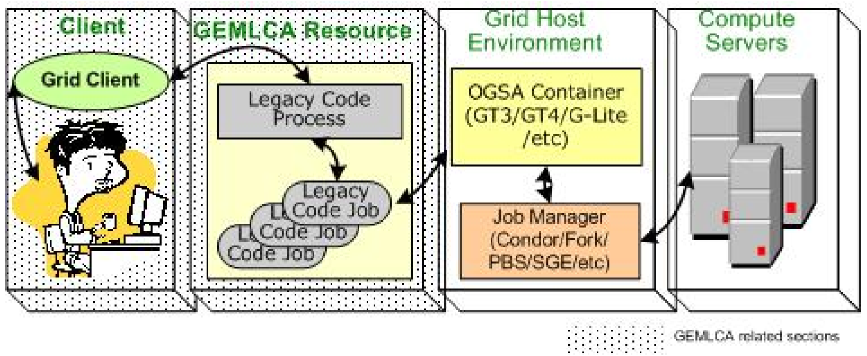

Figure 2 GEMLA Conceptual Architecture

communicate with the Grid client and to hide the legacy nature of the application. To expose a legacy code as a Grid service with GEMLCA requires only a user-level understanding of the legacy application, i.e., to know what the parameters of the legacy code are and what kind of environment is needed to run the code (e.g. multiprocessor environment with ‘n’ processors). The execution environment and the parameter set for the legacy application is described in an XML-based Legacy Code Interface Description (LCID) file that should be stored in a pre-defined location. This file is used by the GEMLCA Resource layer to handle the legacy application as a Grid service.

GEMLCA provides the capability to convert legacy codes into Grid services just by describing the legacy parameters and environment values in the XML-based LCID file. However, an end-user without specialist computing skills still requires a user-friendly Web interface (portal) to access the GEMLCA functionalities: to deploy, execute and retrieve results from legacy applications. Instead of developing a new custom Grid portal, GEMLCA was integrated with the workflow-oriented P-GRADE Grid portal extending its functionalities with new portlets.

Following this integration, end-users can easily construct workflow applications built from legacy code services running on different GEMLCA Grid resources. The workflow manager of the portal contacts the selected GEMLCA resources, passes them the actual parameter values of the legacy code, and then it is the task of the GEMLCA Resource to execute the legacy code with these actual parameter values. The other important task of the GEMLCA Resource is to deliver the results of the legacy code service back to the portal.

4. Automatic Deployment Service in GEMLCA

In the current GEMLCA architecture legacy code services are deployed and mapped manually to Grid resources at workflow construction time. As a pre-requisite to extending GEMLCA with QoS based brokering and load-balancing capabilities, services have to be automatically deployed or migrated from one site to another. This section describes the challenges faced when deploying services, and proposes a general architecture for an Automatic Deployment Service.

4.1 Deployment Scenarios

There are several research efforts identifying and implementing solutions for scenarios when automatic deployment of services is important [7]. Each scenario can be derived from the following two basic cases:

1. *Deploying new Grid services.* This scenario means the deployment of a new Grid service onto a target site by the service developer. Dependencies have to be detected and resolved by the automatic service deployment tool, and the service container has to be prepared accordingly in order to prevent misbehaviour.
2. *Migrating existing Grid services.* This scenario occurs when migrating an already deployed Grid service to a different site where a dependency description is available. However, even within the same Grid, this description could be in a different format than is required, depending on the selected service container. An automated deployment tool should provide a transformation between different dependency descriptions. Where the description is not appropriate, dependencies have to be investigated like in the previous scenario.

Based on these two basic scenarios the following examples illustrate where automatic service deployment is important in a Grid environment:

- *Automatic selection services.* An already deployed service can’t process any more request as its hosting container is overloaded. The service has to be migrated to a site with lower load, and some of its requests have to be redirected to the newly deployed service.
- *Grid systems integration.* Joining different Grids can be more efficient when some services are installed on both of them. Migration of a service in this situation may result in lower communication overhead. In this case a translation is needed between the different site description languages, and deployment specific information has to be provided. Following this, the system has to install the proper environment on the Grid receiving the service in order to carry out the migration.
- *Refining existing services.* Some services (usually data retrieval solutions) provide very generic information to their users, irrelevant to their real, usually restricted, needs. In this case users have to filter this information in order to retrieve what is

relevant for them. To avoid high network traffic this filtering can be implemented and deployed as a new service on the site where the general service resides.

4.2 Deployment Service Architecture

In order to support the previously described scenarios a layered deployment service architecture has been identified. Figure 3 shows this architecture, and illustrates how it is utilised when migrating an already deployed service to a target site. The migration process and the tasks of the different layers of the architecture are the following:

1. The Grid sites register themselves in an information system. The registration contains basic site descriptions.
2. In order to be migrated from site A to an appropriate target site, the service contacts the Automatic Deployment Service.
3. The deployment service queries the information system in order to access site descriptions, and also generates the description of the service to be migrated. The classifier module [8] tests the description of the service against the site descriptions, and generates a set of sites that are the most capable of hosting the service. All the descriptions, with the help of ontologies, are transformed into a meta-description suitable for classification [9]. Following this, the dependency checker investigates the capabilities of the selected sites. The capabilities should be identified with a black box method as the source code is not available in GEMLCA. In a black-box method, dependencies are detected using an observer execution environment. The service uses generic test data that affects all of its features in order to gather runtime dependencies, such as the files accessed, network connections used, or environment variables needed to be set up. The generated descriptions are stored in the information system for further use.
4. Based on the information received from the dependency checker the comparator prepares some metrics (cost and time requirements of the deployment based on the descriptions), and selects the site with the lowest deployment cost (Site B in our example) [10].
5. In order to make *Site B* compatible with *Site A* from the service's point of view, the dependency installer prepares several installation scripts and environment configuration files/setup scripts. These scripts have to take care of all third party software necessary for the service. The established network connections have to be simulated with a proxy. This proxy has to be prepared on both sites.
6. The deployer prepares a sandbox [11] on *SiteB* in order to separate the execution of the service from others. The sandboxing technique used can be various; e.g. a basic *chroot*ed environment, some Java security model based solution, or a virtualisation technique (Xen, VirtualPC, VMware). The deployer interfaces with the actual sandboxing technique to create a new sandbox, and then the installation scripts, created in the previous step, are executed in it.
7. The deployer notifies *SiteA*, and negotiates the transfer of the service between the sites. The negotiator can detect the available and accessible transfer services on each site. It also has the capability to act as an intermediate layer between the source and the destination, if it is necessary. The service has to be registered with the new host environment in an execution environment specific way without restarting it (the state information of services should

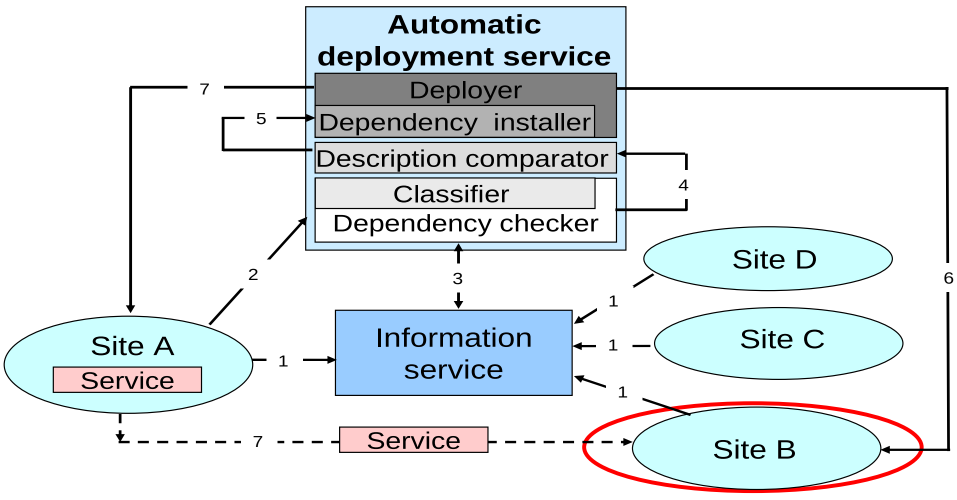

Figure 3 Automatic Deployment Service Architecture

not be modified) [12]).

After the transfer is completed between the two sites the service becomes available on the new site.

5. x-Service Interoperability

The construction and operation of interoperable services running on large-scale Grids lead to great challenges. The highly distributed nature of services within a Virtual Organisation (VO), spanning many different management and security domains, raises both policy and security issues. Current GEMLCA implementations utilise GSI [13] as a standard Grid security solution. Despite the fact that GSI provides important security features it has several problems, which lead to scalability and flexibility limitations, particularly in the authorisation and policy management aspects. These limitations have direct impact on the interoperability of Grid services in multi-domain Grid environments [14]. The aim of our Grid services interoperability research is to build on existing policy and security solutions and standards that are managed independently by different Grid sites, and to develop an architecture that is capable of bridging isolated Grids in a flexible, scalable and dynamic manner. As a result of this work, GEMLCA is significantly extended to enable the deployment, creation, invocation and management of Grid services between multi-domain Grid environments, thus enabling a dynamic integration of different Grid sites.

5.1 Policy & Security Interoperability Scenarios

The policy and security interoperability challenges analysis and investigation are focusing on three major topics: how to support multiple security implementations; how to allow dynamic creation of services; and how to establish trust domains dynamically. Two major scenarios can be derived from these topics:

1. *Deployment and migration of Grid services*. This is an extension of the deployment scenario described in 4.1 by adding extra capabilities to the Automatic Deployment Service taking local policy and security solutions into consideration.
2. *Invocation of Grid services*. The invocation of a Grid service follows its deployment on the target site as a result of either a new Grid service deployment, or a migration of an existing service.

Prior to the deployment and the invocation of the Grid service, policy and security analysis and mapping should be performed in order to decide whether it is feasible to deploy the Grid service onto a target node in a Grid within a different security domain. Based on the scenario analysis, three categories have to be taken into consideration when performing the mapping procedure:

1. *The integration category* is concerned with integrating existing security architectures and models across multiple platforms and hosting environments. Since every Grid is likely to manage security policies, authentication credentials and identities within its own security domain, there is a need in the previously mentioned interoperability scenarios to define a global mechanism that translates access rules and policies from one domain to another. Therefore, local security implementations are independent from each other, while mapping is still possible between these implementations in a global manner.
2. *The interoperability category* is defined in terms of policy. This means that each party is able to specify any policy in order to engage in a secure conversation. Policies expressed by different parties can be made mutually comprehensible. Once policy requirements of each domain are managed, authorisation problems can be solved:
   - managing different kinds of authorisation mechanisms,
   - controlling access between Grid domains (such as enabling the ADS to deploy a Grid service into a different domain, or submitting jobs between domains),
   - allowing the Grid environment to grow and shrink dynamically by adding/removing access rights to services and resources.

   Furthermore, there is a need to manage and translate privacy rules and preferences between different domains.
3. *The trust category* is concerned with establishing trust between different parties. This is a complex problem in a Grid environment due to the need to support dynamic, user-controlled deployment and management of Grid services. In the interoperability scenarios, the trust category is divided into two forms of relationships: direct/mutual trust relationship (invocation), and indirect trust relationship (deployment) that is achieved through an intermediary service, such as the ADS.

5.2 Interoperability Service Architecture

The analysis of the above scenarios demonstrates the need for an interoperability bridge that converts one interface into another one in a dynamic manner. This interface can imply multiple meanings, depending on the context, like security interface (WS-Security) [15], policy interface (WS-Policy, WS-PolicyAttachment) [16] etc. A general interoperability architecture, the x-Service Interoperability Layer (XSILA), has been specified in order to handle interoperability issues between Grid clients and Grid services when they are in different domains. “x” refers to any kind of Grid or Web service in this context. The ADS, as described is section 4, is limited to the deployment of a Grid service within one Grid, and cannot span multiple domains. Extending the ADS with XSILA enables the automatic deployment of a Grid service into different domains. XSILA serves as a bridge between the different Grids, and makes the deployment to a different domain transparent for the ADS by redirecting the communication between the ADS and the services though XSILA, as illustrated on Figure 4. The architecture is composed of five layers:

1. *Negotiator Layer* - collects interoperability properties, such as access mechanisms, policies, and security mechanisms of the involved domains.
2. *Analyzer Layer* - analyses the properties collected by the negotiator layer, defines the differences between domains, and prepares a list of interoperability requirements based on these differences.
3. *Classifier Layer* - classifies the interoperability requirements into interoperability classes. It utilizes a mapping engine to create correlation between the demands of each domain.
4. *Dispatcher Layer* - uses the mapping produced by the classifier layer to spawn a Bridge Service that contains the generated mappings. Each dispatched bridge includes a unique identifier which is then can be used by a client to access the service.
5. *Bridge Layer* - encompasses one or more Bridge Services that are spawned by the Dispatcher Layer. Each Bridge Service is intended to resolve a particular interoperability problem. The Bridge service is discarded once a communication is no longer required.

6. Conclusion and Further Work

Deploying legacy applications on the Grid without reengineering the code is crucial for the wider scientific and industrial take-up of Grid technology. GEMLCA provides a general solution in order to convert legacy applications as black-boxes into OGSA compatible Grid services, without any significant user effort.

Current GEMLCA implementations fulfil this objective, and the integrated GEMLCA - P-GRADE Portal solution offers a user friendly Web interface and workflow support on top of this. However, GEMLCA should be further developed and extended with additional features, like information system support, brokering, load balancing or automatic deployment and migration of services, in order to offer a more comprehensive solution for Grid users.

This paper presented an Automatic Deployment Service Architecture that enables the automatic deployment and migration of GEMLCA Grid services to different sites within the same Grid domain. The combination of this architecture with the x-Service Interoperability Layer

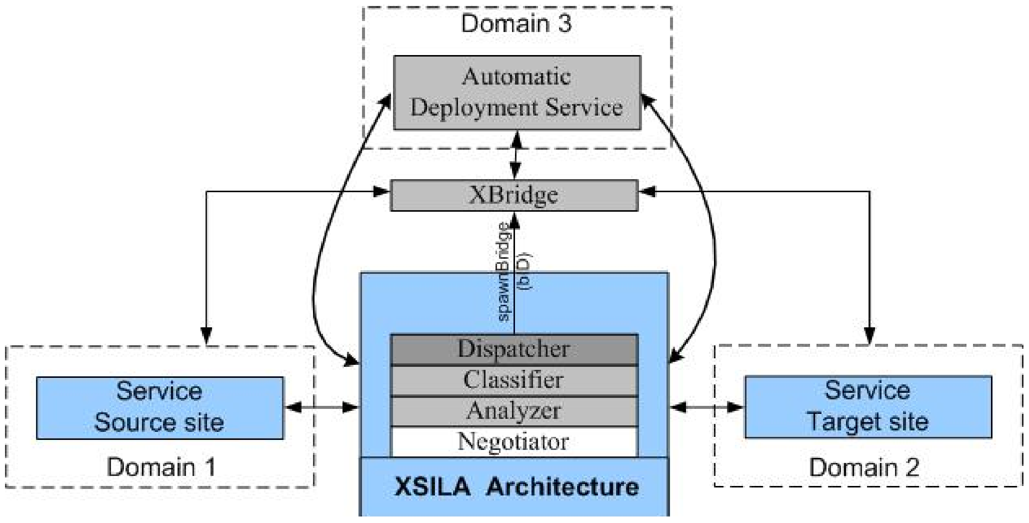

Figure 4 ADS and the x-Service Interoperability Layer

extends deployment and migration capabilities to different domains. Adding these features to GEMLCA enables service developers to deploy their services automatically on the target site, or to migrate the service to a different site, spanning multiple Grid domains when required, if execution is more efficient there.
The implementation of these architectures and their integration with GEMLCA is currently work in progress. Also, the investigation has already started how it could be integrated and extended with existing information system and brokering solutions in order to realise the full GEMLCA-based Grid presented in Figure 1 of this paper.

References

[1] T. Delaitre, A.Goyeneche, T.Kiss, G.Z. Terstyanszky, S.C. Winter, P. Kacsuk, D. Igbe, P. Maselino, K. Sajadah, N. Weingarten: Experiences with Publishing and Executing Parallel Legacy Code using an OGSI Grid Service, Conf. Proc. of the UK E-Science All Hands Meeting, pp. 999-1002, ISBN 1-904425-21, 31st August - 3rd September 2004, Nottingham, UK

[2] Cs. Nemeth, G. Dozsa, R. Lovas, P. Kacsuk, “The P-GRADE Grid portal”, In: Computational Science and Its Applications - ICCSA 2004: International Conference, Assisi, Italy, 2004, LNCS 3044, pp. 10-19.

[3] D. Kuebler, W. Eibach, “Adapting legacy applications as Web services, IBM Developer Works, http://www-106.ibm.com/developerworks/webservices/

[4] Y. Huang, I. Taylor, D. W. Walker, ”Wrapping Legacy Codes for Grid-Based Applications”, Proceedings of the 17th International Parallel and Distributed Processing Symposium, workshop on Java for HPC), 22-26 April 2003, Nice, France. ISBN 0-7695-1926-1

[5] B. Balis, M. Bubak, and M. Wegiel, ”A Solution for Adapting Legacy Code as Web Services, in “Component Models and Systems for Grid Applications” edited by V. Getov and T. Kiellmann, Springer, 2005, pp 57-75, ISBN 0-387-23351-2.

[6] D. Gannon, S. Krishnan, A. Slominski, G. Kandaswamy, L. Fang, “Building Applications from a Web Service based Component Architecture, in “Component Models and Systems for Grid Applications” edited by V. Getov and T. Kiellmann, Springer, 2005, pp 3-17, ISBN 0-387-23351-2.

[7] J. B. Weissman, S Kim, D. England. Supporting the Dynamic Grid Service Lifecycle, Technical Report, University of Minnesota, 2004,

[8] Mike James : Classification Algorithms, Wiley, 1985, ISBN: 0-471-84799-2

[9] M. Cannataro, C. Comito: A Data Mining Ontology for Grid Programming, Conf. Proc of the 1st Workshop on Semantics in Peer-to-Peer and Grid Computing at the Twelfth International World Wide Web Conference, 20 May 2003, Budapest, Hungary

[10] P. Watson, C. Fowler. An Architecture for the Dynamic Deployment of Web Services on a Grid or the Internet, Technical Report, University of Newcastle, February, 2005.

[11] M. Smith, T. Friese, B. Freisleben. Towards a service-oriented ad hoc grid, Conf. Proc of the ISPDC/HeteroPar Conference, 2004

[12] A. Ting, W. Caixia, X. Yong. Dynamic Grid Service Deployment, Technical Report, March, 2004

[13] V. Welch et al.: Security for Grid Services” Twelfth International Symposium on High Performance Distributed Computing (HPDC-12), IEEE Press, June 2003.

[14] L. Cornwall, J. Jensen, D. P. Kesley: Security in multi-domain Grid environments, ZDNet White Papers, Springer, April 2004.

[15] B. Atkinson et al.: Specification: Web Services Security (WS-Security), http://www-106.ibm.com/developerworks/webservices/library/ws-secure

[16] D. Box et al.: WS-Policy - Web Services Policy Framework, Joint specification by BEA Systems, IBM, and Microsoft, May, 2003