AUTOMATIC GRID SERVICE DEPLOYMENT

***Gabor Kecskemeti*[1]*, Gabor Terstyanszky*[2]**
1:MSc, Research Assistant and PhD student;2:PhD,Reader
1,2:University of Westminster

Abstract

This article provides an overview of the current Grid Service deployment problems and proposes a general multi layer architecture for service deployment solutions based on the previously published specialised and partially defined ones. From the architectural point of view a service could be a legacy code wrapped to a service oriented environment, a Web service or a Grid service.

1. TODAY'S GRID

1.1 Closely-Coupled Grid

Nowadays there are several possibilities to access the powers of the high throughput computing by creating virtual organisations with the help of a Grid middleware software ([1]) such as gLite, Globus([3]), Gridbus, Legion or UNICORE. These virtual organisations have a great disadvantage during their expansion phase, because all of the newly joint members has to follow the rules and policies laid down by the founders of the VO. These policies are strict rules about the site, resource and service descriptions provided by the partners. There is a usual language with predefined conditional statements to help the resource brokering and scheduling algorithms to optimise the resource usage based on cost and time from both the Grid user and the resource (or service) provider point of view. The Grid middleware based solutions are generalised enough to provide services for resources which provide processing power, some services, their storage capabilities etc.. The usual price of this general approach is the large footprint of the grid middleware.

1.2 Loosely-Coupled Grid

Before Grid middleware were developed the applications, which were pretended to use on large scale computer systems, have to be designed on a failure tolerant and effective way because these applications should utilise free computing cycles either on personal computers loosely connected to the network with usually low bandwidth solutions. A typical example for this scenario is the Seti@Home project. The users of this project can provide their computer's resources to analyse information gathered by radio telescopes throughout the world looking for signals from an extra terrestrial life form. There is a new initiative lead by the IBM called world community Grid to create a common infrastructure for projects like the Seti@Home. These projects have community support because they are dealing with problems what people are aware of thus they could become widespread. The vendors of these projects have to provide their computing applications for as much plattform as possible. These applications are developed to add the user's computer to a specially designed Grid system on a user preferred manner and platform. To exploit the idle computing cycles of the world there is an emerging demand to provide a system which can use the lonely home and office computers without overloading them like these projects but on a general approach like the previously shown Grid middleware systems.

2. NECESSITY OF THE SERVICE DEPLOYMENT

Both the closely- and loosely coupled Grid installations are quiet homogeneous today because the strict central administration. The administrators select the Grid middleware, the necessary libraries, the description conventions and etc. to be used on each site. When it is time to extend this strictly administered Grid with joining to an other Grid installation the first step of the integration is to handle the inhomogeneity. So far systems using the same middleware could suffer problems when a new job is submitted or a new service has to be deployed on the newly formed Grid etc. . Deployment issues could occur quiet frequently on all kind of Grids, the following list will provide some introduction to cases when the deployment is necessary.

2.1 Deploying new Grid services

When a system designer makes a Grid service available the service should contain some information in its deployment description about the required tasks and dependencies. These dependencies and tasks partially can be identified by an automatic agent the Grid Service Deployment Helper (GSDH). For example, sometimes the software developers have problems to identify the hidden dependencies of a new application. The GSDH can help them to publish the application with its dependencies as a Grid service using the description language available on the target Grid.

2.2 Migrating existing Grid services

In this scenario there is an existing service which can't process more requests because the site where it is executed is overloaded. The service should be deployed on a site with less load and some of the requests to the original service should be redirected to the newly deployed service. This case should be a totally automated one because the Execution Planning Services has to investigate the available sites about their willingness to accept the service which should be deployed on them. This kind of load balancing demands a simple way to checkpoint an instance of a Grid service in order to transfer it to the selected site. ([2])

2.3 Grid Systems integration

If the two joining Grid systems can be more effective when some of their services are installed on both of them to lower the communication overhead, then an automated deployment could be necessary for the services affected with this communication overhead problem. When the two systems start to cooperate network traffic monitoring is important to identify the services which are heavily used on both systems, and if all the legal statements are clarified about the transition then the automated transfer or deployment should be started. The automation here should translate between the different site description languages and should provide the deployment critical information and it has to install the proper environment on the new Grid to provide the selected service there too. ([9])

3. A DEPLOYMENT SUBSYSTEM

3.1 Requirements

There are the following requisites that a deployment system which is capable to handle all the previously described scenarios must fulfil.

- **Reliability**: Grid services are stateful services thus they have a lifecycle. During a deployment of a service other Grid services should not be affected at all, their behaviour should not change while there is an ongoing deployment and their state should not change just because the underlying Grid middleware is accepting a new Grid Service. But some service containers have to be restarted when a new service is added so the already instantiated Grid services would lose their states and their clients would suffer from service availability problems. A proper Deployment Subsystem should solve this issue. ([8])
- **Security**: [7] Trustfulness is a basic concept nowadays in the world of the Grid technologies. An installed service should not execute malicious code or defect systems on any other ways. In today's Grid this is not an issue because the administrators or the people who are responsible for service deployment can check the new Grid service before deployment and if the requirements of the software looks problematic or the person who requested the service is untrusted the service would not be deployed. In a heterogeneous Grid environment where the Grid players are not known to each other the automation of the deployment has to be aware of the future secure execution.

3.2 Proposed Architecture

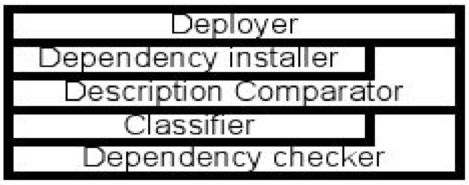

Figure 1: Overview

1. **Classifier**: [5] However a service deployment can be done without any use of a classifier the Grid should not be overloaded with requests to identify the sites which are capable to host the service. In order to lower the number of discovery requests existing site descriptions should be classified with the help of an ontology built to eliminate the differences between the heterogeneous site descriptions. Some experimental ontology have been built already by other

projects to support the Execution Planning System of Globus.

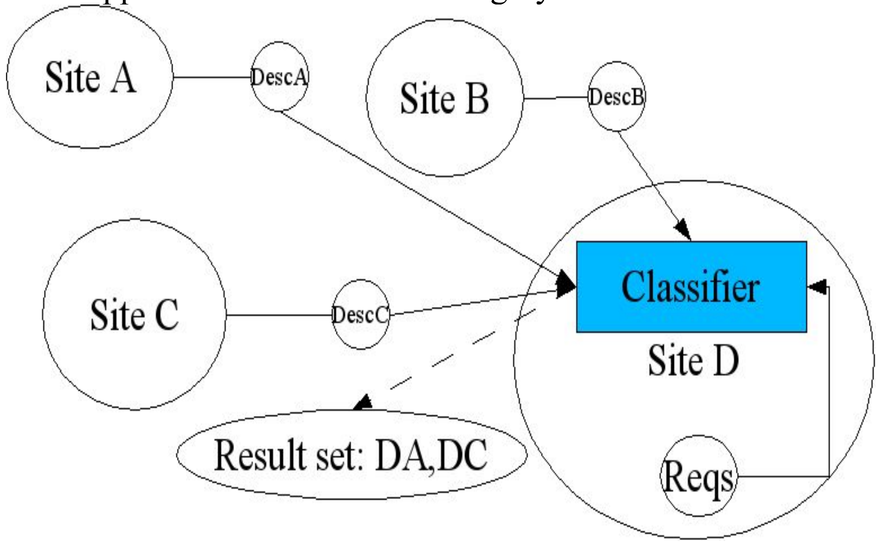

Figure 2: Classification

2. **Dependency checker**: When a service has been written the developer makes some presumptions about the execution environment. These are usually hard coded parts of the application thus it is hard to detect them. The author may assume the application is running in a strict directory structure with some files in them, or there is another usual assumption about the shared libraries present on the system. This is an optional part again because the system can use the men made descriptions so the reviewed deployment system implementations left it out. And there could be hidden dependencies which set back the results of this service for example there could be some references for other files or network connections through locally initiated communications. The dependency detector has at least two options to identify the necessary system resources, files and environment for a particular service:
   - Black box method: Run the application with some testing data which affect all the features of it to gather the runtime dependencies such as the files to be used, network connections during the execution or the environment variables need to be set up.

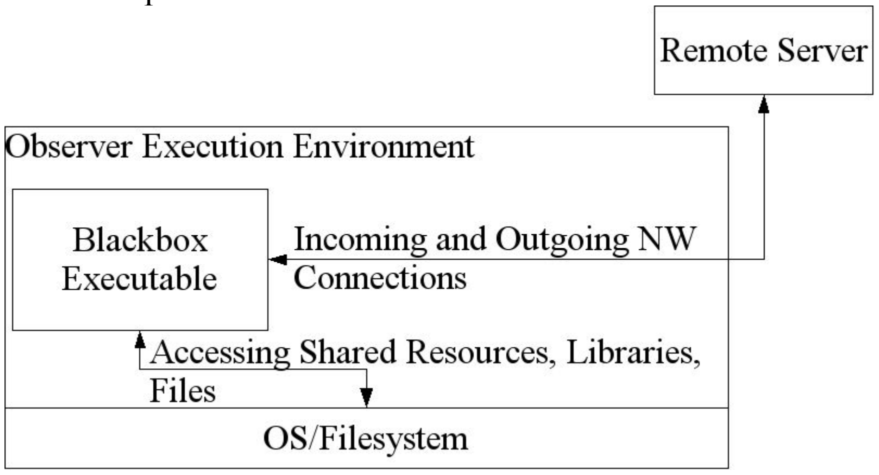
Remote Server
Observer Execution Environment
Blackbox Executable
Incoming and Outgoing NW Connections
Accessing Shared Resources, Libraries, Files
OS/Filesystem

Figure 3: Service activity monitoring

   - Code analyser: Analyse the binary (or if the source is available then them)code of the application to look for its dependencies. Some software systems has a restricted license which may prohibit disassembling the code thus this way should be an optional one.

3. **Description comparator**: [6] This is the second most important task during the deployment process. Thus, all the existing systems contain at least a partial implementation of it. After the classifier identified the sites, which might be capable to host the Grid service, and the dependency checker built the list of the service dependencies the system has to compare the minimum requirements composed by the user, who would like to use the service, and the capabilities of the possible hosting systems. It is preferable to chose a site which are most likely capable to execute the service and the deployment and execution costs times are in the scope of the user's needs. This part should provide a single site to be used because the deployment subsystem should not wait for human interaction to choose between the different results, and this is the part when all the necessary information are available to make an optimal decision.
4. **Dependency installer**: This task should prepare the sandbox (the easiest way to isolate the newly deployed executables for the security requirement) on the selected location. The sandbox should contain all the necessary libraries (these libraries can have an installer which has to be collected by the automated deployment system and used during the building of the sandbox), classes, executables and so on. If the service uses an outgoing network request to communicate with other applications and the site has a strict policy on its firewalls the system has to install a proxy at each side of the network to make the network communications through a secure and enabled channel.

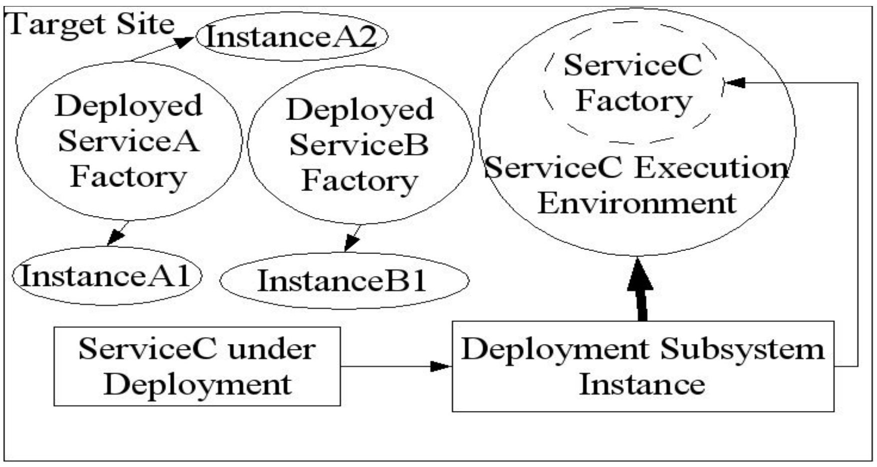
Target Site
InstanceA2
Deployed ServiceA Factory
Deployed ServiceB Factory
ServiceC Factory
ServiceC Execution Environment
InstanceA1
InstanceB1
ServiceC under Deployment
Deployment Subsystem Instance

Figure 4: Dependency installation and service deployment

5. **Deployer**: Since automatic deployment systems exist this is a cruical part to implement with fulfiling at least the reliability requirement. The deployer depends on the container the Grid middleware is running on. For example the basic Globus container has to be restarted if a new service is added which is not a preferred solution. There are existing approaches:
   - Adaptive Grid service: [9] The AGS solution can deal with the original Globus container because it extends its functionality with a new classloader to provide a sandbox for the newly deployed service and the possibility to add the implementation of the new service without restarting.
   - Tomcat based solutions: [6] A service oriented Grid middleware might have been installed on an Axis enabled Tomcat which already has a runtime deployment functionality. The only task what the system has to perform here is

to separate the execution of the newly deployed service from the other services enough to fulfil the security criterion of the Deployment Subsystem.

4. OUTLOOK

With providing this system to the public the people can easily install a future Grid middleware based software on their computer and when they are not using it they let other people do it. The person with an idle computer can make some earnings with lending his/her computer for Grid users, and of course he/she can use the underlaying Grid system for his/her purposes. For example when a user just recorded a home video about a family occasion he/she can edit and transform the video and audio streams with the power of the Grid easily. (There could be some services which are dealing with transitions on the partitions of the video and so on.)

REFERENCES

[1] Parvin Asadzadeh, Rajkumar Buyya, Chun Ling Kei, Deepa Nayar, and Srikumar Venugopal: Global grids and software toolkits: A study of four grid middleware technologies. Technical Report, 2004.
[2] R. Buyya, S. Date, Y. Mizuno-Matsumoto, S. Venugopal, and D. Abramson: Composition and on demand deployment of distributed brain activity analysis application on global grids. In New Frontiers in High- Performance Computing: Proceedings of the 10th International Conference on High Performance Computing, 2003.
[3] Ian Foster, Carl Kesselman, and Steven Tuecke: The anatomy of the grid. Intl J. Supercomputer Applications, 2001. Enabling Scalable Virtual Organizations.
[4] Robert Haas: Service Deployment in Programmable Networks. PhD thesis, Swiss Federal Institute of Technology, 2003.
[5] Gabor Kecskemeti: Comparison of Adaptive Classification Methods. Master's thesis, University of Miskolc, Department of Information Technology, 2004.
[6] Matthew Smith, Thomas Friese, and Bernd Freisleben: Hot service deployment in an ad hoc grid environment. ICSOC04, 2004.
[7] Matthew Smith, Thomas Friese, and Bernd Freisleben: Towards a serviceoriented ad hoc grid. ISPDC/HeteroPar, 2004.
[8] Ai Ting, Wang Caixia, and Xie Yong: Dynamic Grid Service Deployment, 2004.
[9] Jon B. Weissman, Seonho Kim, and Darin England: Supporting the Dynamic Grid Service Lifecycle, 2004.