Interoperable and Transparent Dynamic Deployment of Web Services for Service Oriented Grids

Michael Messig and Andrzej Goscinski

School of Engineering and Information Technology
Deakin University
Pigdons Road, Geelong
{messig,ang}@deakin.edu.au

Abstract. Dynamic deployment of Web services is a term used frequently when describing the selection and deployment of a service to a grid host. Although current grid systems (such as Globus) provide dynamic deployment, the requirements of the service being deployed are not considered. Therefore truly dynamic deployment cannot be achieved as the services deployed are restricted to the grid system used. We present a dynamic deployment mechanism as part of self configuration in a service oriented grid environment. The dynamic deployment mechanism takes the requirements of the service into consideration, including parameters such as the operating system required to execute the service, the required software libraries, any additional required software packages, price and Quality of Service (QoS) parameters.

Keywords: Autonomic Computing, Service Oriented Grids, Web Services.

1 Introduction

Grid computing has changed the way distributed systems are constructed, programmed and used. With a focus on Web services to provide interoperability, service oriented grid computing in particular have introduced a number of new problems which cannot be solved by traditional distributed systems research. One area which is of particular importance is the deployment of services.

Service deployment is the operation of configuring a service, transferring the service to a host of a service provider (which will perform execution of the service code) and publishing the service with any discovery mechanisms (such as UDDI). Although this operation may seem straightforward, a number of issues must be considered to ensure deployment is dynamic. Firstly, the requirements of the service must be taken into consideration. If a service requires a particular set of software libraries, additional applications, or a certain operating system to run on, a deployment mechanism must ensure the destination host meets these requirements. Secondly, QoS attributes must be taken into account. If a service requires specific QoS attributes such as availability, throughput, interoperability or security, a deployment facility must ensure the destination host conforms to these requirements. Finally, service deployment must take into consideration trustworthiness of the

M. Bubak et al. (Eds.): ICCS 2008, Part I, LNCS 5101, pp. 164–173, 2008.

destination service provider. The service (or clients using the service) may have a prior agreement on the level of trust required by both the service and the client. The destination host of a service provider chosen for service deployment must adequately meet any trust requirements prior to deployment.

The aim of this paper is to present our research into a dynamic deployment facility as part of an autonomic grid management system. By taking an autonomic approach to service management, we are able to provide characteristics such as self discovery and negotiation (automatic registration and discovery of services), self configuration mechanisms (dynamic deployment and migration) and self healing (automatic recovery) transparently while ensuring interoperability is achieved. In this paper we focus on our novel dynamic deployment system, one part of our overall autonomic broker environment, which ensures the requirements of the service and the grid environment are taken into consideration during deployment and provides transparent and interoperable service deployment.

2 Related Work

There are a number of grid systems which claim to support dynamic deployment. These systems however fail to address the requirements of the service, QoS and trust.

Gridbus is a grid system focused on the grid economy, allowing providers of services offered by the grid to charge for their use. The user of the grid system is able to specify some QoS parameters and query parameters such as price. The Gridbus system then finds a service which matches these requirements. Gridbus has been developed to cooperate with other grids such as Globus and exposes some of the services through a Web service interface [4]. Gridbus, although addressing QoS, does not provide any self configuration mechanisms, including dynamically deploying services in which their requirements are considered.

ProActive is a grid system which has been developed for object oriented parallel processing, mobile and distributed computing [2], and attempts to solve the problem of code reuse by introducing a grid programming and deployment framework. The aim of ProActive is to enable a simple hierarchical deployment model through the use of Java objects and focuses on scalability [2]. Although ProActive provides some self configuration mechanisms (deployment and migration) these mechanisms can only be used with ProActive Java objects. Therefore, deployment is not dynamic. The deployment mechanism is not interoperable or flexible and do not take into consideration a service or object's requirements during deployment.

The Australian BioGrid Portal [1] provides access to molecular docking applications and chemical databases. These services are used to provide data access, analysis and visualisation of molecular screening results using web based tools. The portal to the BioGrid does not offer dynamic deployment with consideration of a service's requirements. The grid requires that the client deploying the service knows the specifics of each APAC node, their network address and the software installed on each node. This is a very primitive approach which requires the users of the system to be directly involved in constructing and deploying the required services.

Although some grid systems reviewed (Globus and ProActive) provide deployment mechanisms, these are not dynamic. The metrics of the grid, properties of destination

hosts and the service's requirements are not considered. From the grid systems reviewed, there were no systems which provided dynamic deployment. In summary none of the systems surveyed provide adequate self configuration mechanisms.

3 Autonomic Grid Environment

We have previously reported on our goal of an autonomic grid environment in [8]. Autonomic characteristics have the potential to significantly improve service oriented grid environments; self discovery and negotiation, self configuration and self healing mechanisms reduce the complexity of grid systems while improving reliability, interoperability and usability. To ensure autonomic characteristics are provided for all types of grid and Web service systems, we must first identify how autonomic computing characteristics can be integrated into service oriented grids.

Autonomic characteristics can be integrated within the operating system. This however has the drawback in that grid environments are heterogeneous and many different service providers may use many different operating systems. Autonomic characteristics may be integrated within each service. This however is impractical as some of the mechanisms required by autonomic computing, such as self discovery and self healing, would require coordination between all of the services which adopt the autonomic computing characteristics.

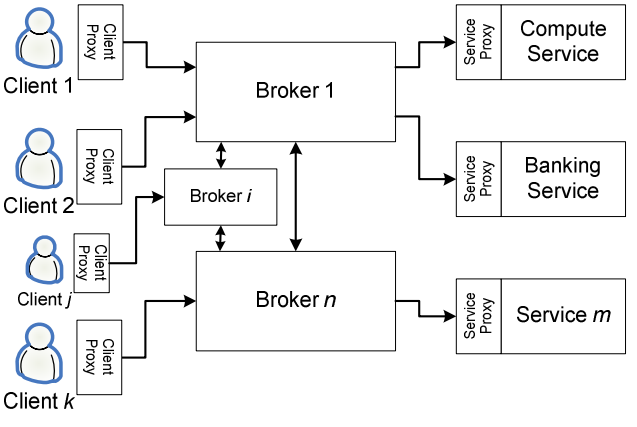

Fig. 1. Interaction between clients, brokers and services

Finally, the autonomic characteristics can be provided by an intermediary such as a broker. This approach does not have the downfalls of the previous two approaches, as a specific operating system is not required and all services do not have to include autonomic characteristics. Different grid toolkits, services and client applications can all take advantage of autonomic computing characteristics through the assistance of a broker as it is not tied to a particular software system. By introducing a broker as an intermediary, shown in Figure 1, the broker not only keeps track of client requests but

also ensures that the client's requests are fulfilled. We can provide autonomic characteristics transparently and ensure interoperability through the adoption of proxies. As grid environments are distributed and scale to extremely large distributed systems, the broker must be distributable and must be able to integrate with other broker instances. We propose the adoption of a System Management Broker (SMB) which provides self discovery and negotiation, self configuration and self healing and assesses the trustworthiness of clients and service providers (through trusted registries, trust rating authorities or other trusted SMBs), shown in Figure 2.

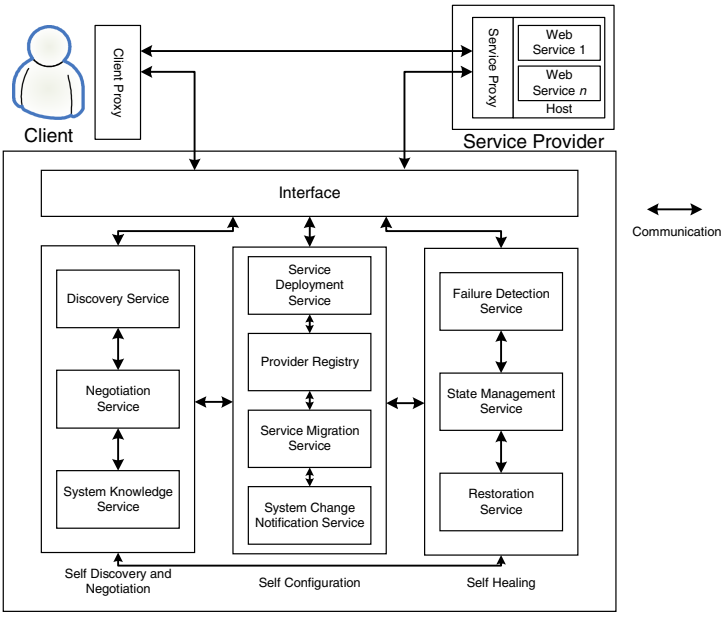

Fig. 2. SMB components

These include the interface; the self discovery and negotiation modules which contain the discovery, negotiation and system knowledge services; the self configuration module which contains the service deployment, provider registry, service migration and system change notification services; and the self healing module which contains the failure detection, state management and restoration services. Each of these modules, apart from the interface, is responsible for providing individual autonomic services. As each of the individual services within the SMB is a Web service, the functionality of each of the services can be exposed as a set of Web service methods.

To maintain transparency between clients, services and the broker, we propose two proxies. The client proxy is used by client applications to transparently discover and communicate with the SMB, while the service proxy is used by Web services and service providers to transparently discover an communicate with the SMB.

4 Dynamic Deployment

The SMB environment (as specified in Section 3) must provide dynamic deployment which takes the requirements of the service to be deployed into account when selecting a suitable destination host. Once a service is deployed (or registered) with the SMB, the service can be discovered and used by clients. To provide dynamic deployment we propose the following operations depicted in Figure 3.

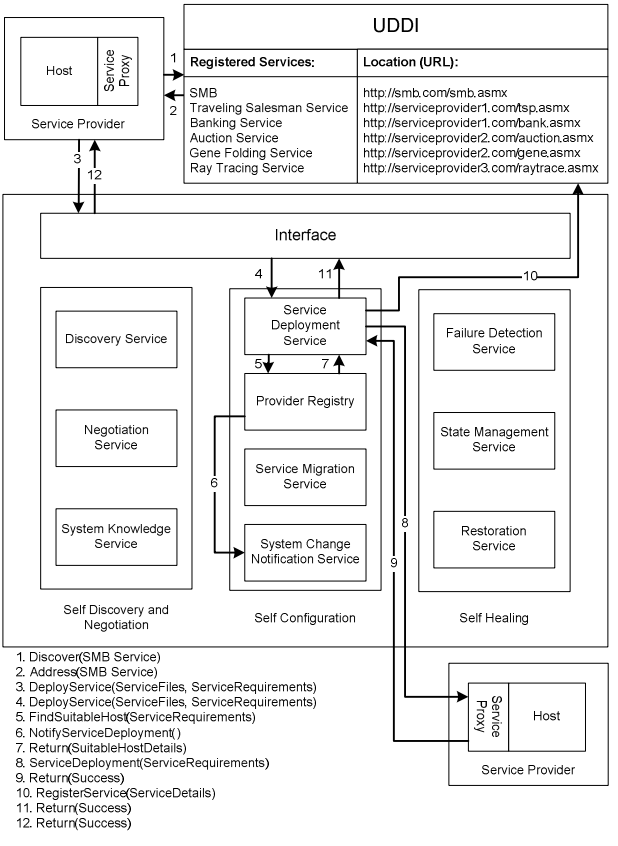

Fig. 3. Deploying and Registering a Service

The service provider must first invoke a discovery service to find an available SMB (Message 1). Once a broker is found (Message 2), the service provider invokes the SMB and requests service deployment (Message 3). The provider supplies the files required to execute the service and a document containing the details of the service; the broker is then aware of any special requirements of the service, including the operating system the service is designed for and any mandatory software packages or libraries required by the service. The details also include any QoS parameters defined by the service provider, a price for using the service, its name and a short

description. The broker's interface passes the request to the service deployment service (Message 4) which invokes the provider registry to find a suitable destination host which meets the service's requirements (Message 5). The provider registry informs the system change notification service that a change in the grid has occurred (a service has joined the grid) and a suitable host is required (Message 6). If a suitable host is found, its details are returned (Message 7). Although the first available suitable host is selected, scheduling algorithms can be used (provided by the service provider hosting the deployment service). Scheduling is an already established area of research and is not the focus of this paper. The service is then deployed by invoking the host's service proxy (Message 8), which deploys the service and returns the outcome (Message 9). The service is registered with UDDI (Message 10) and the outcome is returned (Messages 11 and 12). If unsuccessful deployment is aborted and the SMB attempts to find another suitable host. If no suitable hosts are available, the broker attempts to deploy the service with any other known SMB's (cross SMB deployment).

To be able to deploy services dynamically in a user friendly manner, we implement the Service Provider Tool for service provider registration and service deployment. The former uses the mechanisms of the service proxy to register with the SMB. Several sets of input are required, including provider's name, description and URL. Contacts at the service provider and the details of all hosts and services at the service provider can be added. Each host must be specified to ensure service deployment to these hosts can be carried out. The attributes of the host, such as all installed software, operating system and details of the host's service proxy are required. The Service Provider Tool also deploys services either dynamically using the SMB to find a suitable host or to a specified host. The service's details must be specified for deployment (name, description, version, operating system, required software and QoS attributes) and the service's binary files as a compressed (ZIP) file. The tool then utilises the service proxy and the SMB to deploy the service.

5 Experimentation of the Dynamic Deployment Facility

To demonstrate our approach is sound, we perform several experiments on a heterogeneous enterprise grid testbed. The testbed consists of 12 local Intel based nodes (intentionally older Pentium III nodes) connected via 100MB/s Ethernet. This local grid is then connected to 4 remote nodes of a remote grid (intentionally new Pentium IV nodes) via a microwave link and requires the use of a VPN.

Auction Service – The first experiment we undertake is on a commercial centric application. These applications are typically client driven, not CPU bound, and have a short execution time. To demonstrate the applicability and the benefits achieved from the SMB in a commercial environment we have developed an auction service reflective of auction services such as eBay [5]. To demonstrate the flexibility of our approach we develop several different auction service versions: a vanilla auction service which has no support from any external or third party toolkits or applications and no state management mechanisms, a WSRF auction service which utilizes the Web Service Resource Framework (WSRF) for state management, an SMB auction service which uses the SMB for state management and a WSRF with SMB service which uses both WSRF and the SMB for state management For the vanilla and SMB auction services we also study the use of mobile devices for running the client

application. This cannot be studied for the WSRF implementation as the WSRF library (WSRF.NET) does not support mobile devices. The only differentiating factor between the auction service versions (in terms of service deployment) is the size of each service. Each version of the auction service is compressed into a ZIP archive. The vanilla auction service is 1.28KB (Kilobytes) in size. The SMB auction service is 49.7KB due to the inclusion of SMB assembly (approximately 36KB uncompressed) and the UDDI library required by the SMB assembly (approximately 96KB uncompressed). The WSRF service is 822KB due to the inclusion of the WSRF.NET libraries. Finally, the combined WSRF with SMB auction service is 870KB. We deploy each of the auction services and measure the deployment times. The average resultant deployment times across ten runs for each experiment are shown in Figure 4.

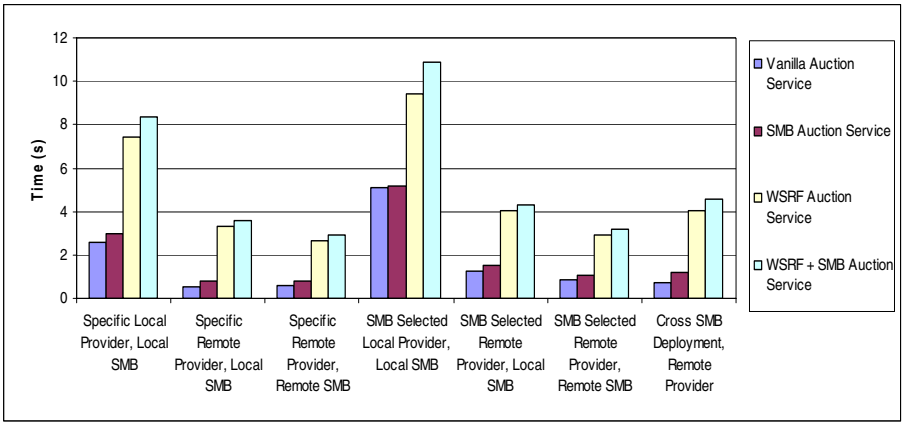

Fig. 4. Auction Service Deployment

First, we deployed to a specific host of a local service provider. The vanilla auction service performed the best in this test (and subsequently across all test scenarios), due to the service being the smallest and hence the quickest to process by the destination host. The SMB auction service performed close to the vanilla service, therefore for smaller services, a difference of approximately 50KB does not noticeably affect deployment (7.5% of deployment time). The WSRF and WSRF with SMB auction services performed on average twice as long as vanilla and SMB services (both WSRF services are larger). During deployment, both the SMB and the destination host's service proxy must receive the service files, process the SOAP, identify and process the compressed file and extract the service files on the destination.

When we look at deployment on hosts of a remote service provider (across a cross campus microwave link), the time required to deploy the services drops dramatically. When remote destinations are selected, the time difference between the larger services (WSRF and WSRF with SMB) and the smaller services (vanilla and SMB) is noticeably less than when local destination hosts are selected. The remote hosts are able to complete deployment 2.6 times faster than local hosts when the destination is selected by the SMB and 3 times faster than the local hosts when the destination host

is specified. This is attributed to the remote hosts having greater processing power and can process the messages sent by the deployment tool more efficiently.

When we used the SMB to select a destination host, the time required to is increased. On the local hosts, the SMB selection of a destination host produces a much larger overhead, an average of approximately 2.3 s (34%) when compared to static deployment. On the remote hosts, the difference between static and dynamic deployment is small (approximately 0.28 s in all cases); however the average overall increase is approximately 20%. The increase in time is expected due to the additional steps performed by the SMB in selecting a suitable and available destination host. As the remote hosts are more efficient in processing requests, additional overhead exists but it is relatively smaller than the overhead introduced on the local hosts.

Finally, we look at the results of deployment across two different SMB's. In this case, the environment is set up such that the local SMB does not have an available suitable destination host to deploy the auction services and contacts the remote SMB for service deployment. For the smaller auction services (vanilla and SMB), the difference between cross SMB deployment and SMB selected deployment is small (approximately 20 ms or 14%), however, for the larger auction (WSRF and WSRF with SMB) services the difference is greater, approximately 1.5 s or 43%. The larger auction services are slower as the service's files must be handled by an additional mediator (the local SMB) before being deployed (by the remote SMB).

We can conclude from this experiment that service deployment is affected by two factors: the size of the service and the processing capabilities of the destination host. The size of the service affects deployment in all of the tested scenarios. This is expected, as the larger the service (in size), the longer it takes to process the files to the destination host. The processing capabilities of the destination host have a big impact on the overall deployment time, as expected. Faster hosts are able to process requests more efficiently and therefore the overall time to deploy a service is quicker.

Ray Tracing – The second experiment is on a scientifically focused application. We report on the experimentation with a commonly used computational application, the Povray ray tracer [10]. The scientific focus of grid computing is primarily in high performance computing and parallel processing and the Povray application has widely been adopted as an appropriate application to test parallel processing systems [7]. We expose the latest version of the Povray application as a Web service. We then use the inbuilt mechanisms of the Povray application to split the rendering job into many distributable parts [10]. We develop a client application to invoke the Povray Web service and use the SMB environment to manage the service. We repeat the experiment ten times and record the average measurement.

As we have shown the difference between local and remote deployment in the auction service experiments, we restrict the deployment of the Povray service to local service providers. We conduct two different tests, static and dynamic deployment. As the only differentiating factor in deployment of services is the actual size of the service, as a basis for comparison, we compare the time required to deploy the Povray service with the time required to deploy the auction services. The results of the test are shown in Figure 5. A single instance of the distributed Povray Web service is approximately 777KB (KB) in size. This is compared to the Vanilla auction service, SMB auction service, WSRF auction service and the WSRF with SMB auction service which are 1.28KB, 49.7KB, 822KB and 870KB, respectively. The Povray

service takes an average of approximately 5.1 s to deploy when the destination host is specified and 6.8 s to deploy on a destination host selected by the SMB. As expected, this service is slower to deploy than the SMB auction service (49.7KB) but faster to deploy than the WSRF auction service (822KB). The additional time required to deploy the service to an SMB selected host is due to the selection process, which requires matching a suitable host to the service's requirements.

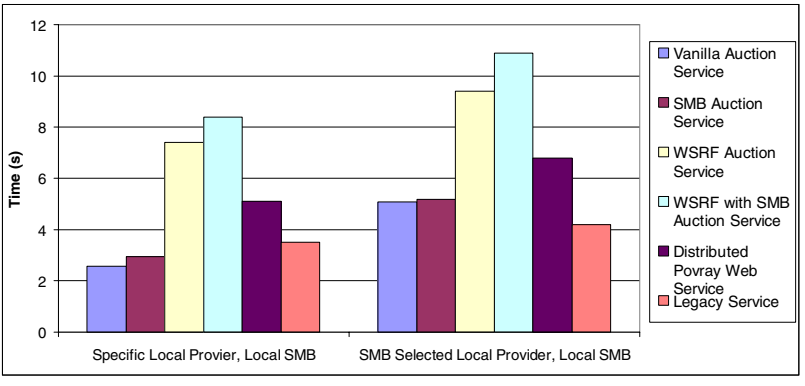

Fig. 5. Comparison of Deployment Results

Legacy Application – The final experiment we perform focuses on legacy applications. We wrap a legacy application as a Web service using the Web service Legacy Application Wrapper (WLAW) shown in [9] and use the Service Provider Tool to dynamically deploy the service. The legacy application used for these experiments is the popular bioinformatics application, hybrid-ss-min, which is a gene folding algorithm used to compute the minimum energy folding of a DNA or RNA sequence [9]. Once we have wrapped the hybrid-ss-min application as a Web service, we compare the time required to deploy the service with the services from the previous experiments. The results of the deployment tests are shown in Figure 5.

The legacy service, once compressed, is approximately 111KB and static deployment to a local host requires approximately 2.8 s. Comparing this time to the previous results, deployment is faster than the distributed Povray Web service, however not as fast as the SMB or vanilla auction services. This is expected, as the size of the legacy service is slightly larger than the SMB and the vanilla auction service, but not as large as the distributed Povray Web service. The same can be seen in deploying the legacy service dynamically. As expected, the time required to deploy a service is largely influenced by the size of the service. When deploying a service to an SMB selected host, the overall time required to deploy is larger, however the size of the service is still influential in the completion time of the deployment operation.

6 Conclusion

The aim of this paper was to demonstrate the autonomic grid management system in providing dynamic deployment. We achieved this aim through the experimentation of

three different types of applications to reflect the different uses of typical grid environments. By intentionally selecting different categories of applications we were able to successfully demonstrate transparency, usability and interoperability of the SMB in managing different applications. Through our novel autonomic grid management system we have achieved a significant improvement in service deployment. These achievements were realised through close cooperation of client and service proxies, the SMB and the Service Provider Tool. We have shown through these system components that the time required to deploy a service is relative to the service's size and the computational capabilities of the destination host. Our future work consists of rigorous testing of the SMB in a large commercial grid environment.

References

1. Smith, B., Buyya, R., Branson, K.: BioGrid: Web and Grid Services Enabled Molecular Docking for Drug Discovery. APAC Grid Program, Australian Partnership for Advanced Computing (APAC), Canberra, Australia, 2004-2006
2. Baduel, L., et al.: Programming, Composing, Deploying for the Grid, Grid Computing: Software Environments and Tools. Springer, Heidelberg (2005)
3. Beeson, B., et al.: A Portal for Grid-enabled Physics. In: Proc of the 5th IEEE/ACM Int Workshop on Grid Computing (GRID 2004), Pittsburgh, USA, November 2004. IEEE CS Press, Los Alamitos (2004)
4. Buyya, R., Venugopal, S.: The Gridbus Toolkit for Service Oriented Grid and Utility Computing: An Overview and Status Report. In: Buyya, R., Venugopal, S. (eds.) Proc of the First IEEE Int Workshop on Grid Economics and Business Models (GECON 2004), Seoul, Korea, April 23, 2004, pp. 19–36. IEEE Press, New Jersey (2004)
5. eBay, Tap into the eBay Platform (2007) (last access November 2007), `http://developer.ebay.com`
6. Foster, I.: Globus Toolkit Version 4: Software for Service Oriented Systems. In: Jin, H., Reed, D., Jiang, W. (eds.) NPC 2005. LNCS, vol. 3779, pp. 2–13. Springer, Heidelberg (2005)
7. Freisleben, B., et al.: Parallel raytracing: a case study on partitioning and scheduling on workstation clusters. In: Proc. of the 30th Hawaii Int. Conf. on System Sciences, January 1997, vol. 1, pp. 596–605 (1997)
8. Messig, M., Goscinski, A.: Autonomic System Management in Mobile Grid Environments. In: Proc of The 5th Australasian Symp on Grid Computing and e-Research (AusGrid 2007), Ballarat, Australia (2007)
9. Messig, M., Goscinski, A.: Service Migration in Autonomic Service Oriented Grids. In: Proc of the 6th Australasian Symposium on Grid Computing and e-Research (AusGrid 2008), Wollongong, Australia, CRPIT, vol. 82 (2008)
10. Persistance of Vision Pty Ltd. POV-Ray – The Persistence of Vision Raytracer (2007) (last accessed September 2007), `http://www.povray.org`
11. Zuker, M.: Mfold Web Server for Nucleic Acid Folding and Hybridization Prediction. Nucleic Acids Research 31(13), 3406–3415 (2003)