Towards Architecture Model based Deployment for Dynamic Grid Services

Gang Huang, Meng Wang, Liya Ma, Ling Lan, Tiancheng Liu, Hong Mei
School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China.
E-mail: {huanggang, wangmeng, maly, lanling, liutch} @ sei.pku.edu.cn; meih@pku.edu.cn

Abstract

The deployment of grid services should make the services, including those to be deployed and those already deployed, operate with desired functionalities and qualities. The critical challenge in the deployment is that many technical and non-technical factors have to be taken into account, such as performance, reliability, utilization, operating cost, incomes, and so on. Since the factors change continuously, some deployed services may have to be re-deployed for guaranteeing their functionalities and qualities. This position paper presents an approach to the deployment and re-deployment of grid services based on software architecture models. In this approach, all services in a grid consist in a software architecture, which represents the services, their relationships and other factors in a global, understandable and easy-to-use way. To demonstrate the approach, a visual tool for deploying services onto a set of popular grid infrastructures, including J2EE application servers and BPEL engines, with the help of software architectures is developed.

1. Introduction

As computing power, network bandwidth and storage capacity have increased dramatically over the past decade, Internet cannot enable the users to access the resources in a flexible, efficient and reliable way with current application models, such as WWW, FTP, etc. As a result, Grid computing has emerged, which enables the virtualization of distributed computing and data resources to create a single system image and grants users and applications seamless access to vast IT capabilities [10].

Distinguished by traditional distributed computing, Grid focuses on the frequent and continuous changeability of the users, user requirements, resources and resource providers. In order to enable flexible, secure, coordinated resource sharing among dynamic collections of individuals, institutions and resources, Grid defines the concept of Virtual Organization (VO). A VO is a collection of resource providers and consumers with a set of clearly and carefully defined sharing rules, such as what is shared, who is allowed to share, the conditions under which sharing occurs, etc [4].

More technically, grid computing is based on an open set of standards and protocols, such as Open Grid Services Architecture (OGSA) and Web Services, which enable resources accessible as services across heterogeneous, geographically dispersed environments [6]. In that sense, the users can solve their problems through requesting proper services, while the resource providers can share their resources as services. In order to deal with user requests in desired qualities and utilize the resources efficiently, the VO has to properly install all services onto resource providers and carefully define their collaborating relationships in terms of sharing rules. Such work that makes the services ready to be used is called deployment[1].

The deployment is a very important activity for a VO because it is responsible for introducing new services to meet new user requirements, removing useless services, changing already deployed services to keep up with changing environments and so on. From the perspective of the information system lifecycle, the service deployment mediates between the service development and service operation. Due to the extremely open and dynamic natures of Grid, the service developer can only make some assumptions instead of precisely predict the operating environments. Similarly, the operating environment cannot support any service because of some limitations. Moreover, the VO may have some special sharing rules in terms of its purpose, scope, size, duration, structure, community and sociology.

To make the above assumptions, limitations and sharing rules consistent is the key to assure the deployed services of desired functionality with desired qualities. For a service deployer, he/she has to investigate the static and dynamic details of operating environments, understand the functionality and desired quality of the services to be deployed, consider the sharing rules of the VO, etc. For examples, shared resources may range from programs,

[1] Note that, "Grid Deployment" refers the activity to build up a Grid, while "Deployment in Grid" refers the activity to install a service in an existing Grid. This paper focuses on "Deployment in Grid".

files, and data to computers, sensors, and networks; the collaborating relationships among services may range from client-server, n-tiers to peer-to-peer; the services may be controlled at difference sophisticated and precise levels, including fine-grained and multi-stakeholder access controls, delegation, and local or global policies; a service may support single user or multi-user in the performance-sensitive or cost-sensitive way. Taking so many factors into account simultaneously makes the deployment very challenging. Furthermore, since many factors may change continuously and even some factors may be available only when the service runs for a while, the service already deployed has to be deployed again (called re-deploy) to cope with the ever-changing factors.

From the above discussion, we argue that the key of the service deployment is how to represent the knowledge derived from the service development, the information collected from the operating environments and the sharing rules of the VO in a uniform, understandable and easy-to-use way. This paper presents an approach to the deployment and re-deployment of grid services based on software architecture models. In this approach, all services in a grid consist in a software architecture, which represents the services, their relationships and other factors in a uniform, understandable and easy-to-use way. To demonstrate this approach, we develop a prototype of the visual tool for deploying services onto a set of popular grid-enabled infrastructures, including J2EE (Java 2 Platform Enterprise Edition) application server [23], BPEL (Business Process Execution Language) engine [2] and Globus [9], with the help of software architectures. Finally, a detailed case study of deploying services with the consideration of reliability is discussed to illustrate the complex and difficulty of the service deployment and the applicability of the tool.

The rest of the paper is organized as follows: section 2 explains the idea of introducing software architecture into service deployment; section 3 presents the framework of architecture based service deployment; section 4 illustrates the visual supporting tool and section 5 introduces some related work; the last section concludes this paper and identifies the future work.

2. Approach Overview

Since its first literately identification and discussion [19], software architecture becomes an important subfield of software engineering, receiving increased attention from both academic and industrial community. It describes the gross structure of a software system with a collection of components, connectors and constraints [21]. In general, software architecture acts as a bridge between requirements and implementation and provides a blueprint for system construction and composition. It helps to understand large systems, support reuse at both component and architecture level, indicate the major components to be developed and their relationships and constraints, expose changeability of the system, verify and validate the target system at a high level and so on [7].

Due to the success of software architecture in the development, some researchers propose to maintain and evolve software systems with the help of software architecture [28]. Particularly, we propose a framework to make software architecture an entity at runtime, called Runtime Software Architecture (RSA) [12]. RSA can immediately capture changes of the runtime system so as to keep itself up-to-date, and ensure that changes made on RSA will immediately lead to corresponding changes of the runtime system. In other words, the runtime system can be maintained and evolved online via RSA. This framework has already been implemented in PKUAS, a reflective J2EE application server [16].

Recall the challenges of grid service deployment, it is a natural and feasible approach to introducing software architectures into the deployment. As shown in Fig. 1, both software architectures equipped with plentiful knowledge produced in the development and software architectures representing runtime information of operating environments are applied into the deployment. As a result, software architecture plays a centric role in the whole software lifecycle. When a deployed service does not work well, the functionalities or qualities of the software architecture will be damaged or decreased. Such

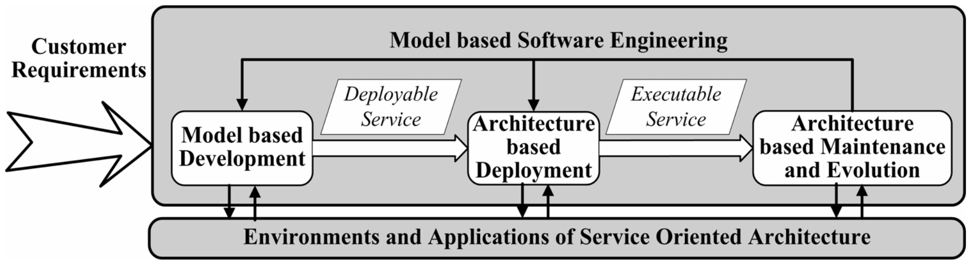
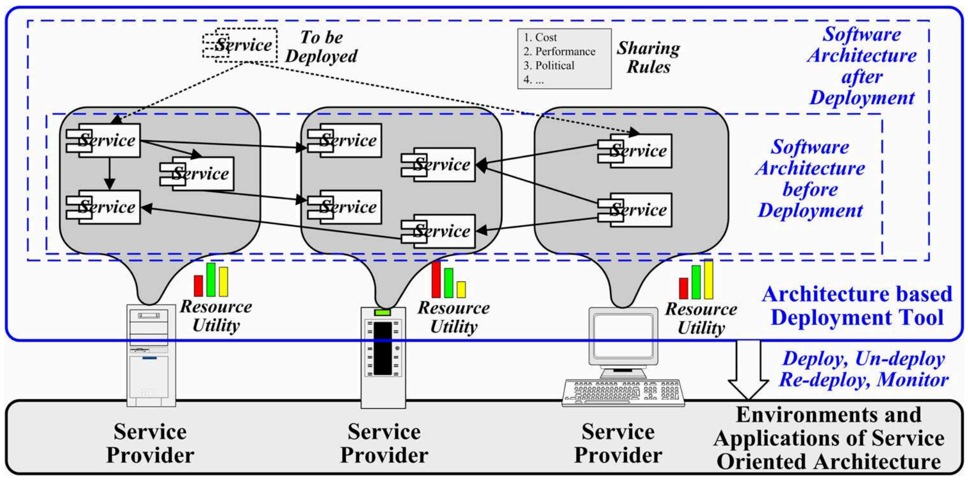

Customer Requirements
Model based Software Engineering
Deployable Service
Executable Service
Model based Development
Architecture based Deployment
Architecture based Maintenance and Evolution
Environments and Applications of Service Oriented Architecture

Fig. 1 Software architectures in the development, deployment and maintenance



Service
To be Deployed
1. Cost
2. Performance
3. Political
4. ...
Sharing Rules
Software Architecture after Deployment
Software Architecture before Deployment
Service
Resource Utility
Architecture based Deployment Tool
Deploy, Un-deploy Re-deploy, Monitor
Service Provider
Service Provider
Service Provider
Environments and Applications of Service Oriented Architecture

Fig. 2 Software Architecture Model based Service Deployment in a Virtual Organization

case can be discovered and repaired with the help of architecture based software maintenance. Then, once a service is deployed, it will keep available until being un-deployed.

The core idea of the architecture based deployment is shown in Fig. 2. All services deployed in a VO form a huge software architecture. The service to be deployed has a small software architecture. The service deployment is to ensure that the small software architecture merges into the huge one. Obviously, the service to be deployed has implicit interactions with existing services because, for examples, it will share resources and even cause resource competition with existing services and the sharing rules of the VO may force the service to be or not to be deployed in the give nodes. In terms of the philosophy of meeting new customer requirements with existing IT capabilities, the service to be deployed may implemented by reusing some existing services. That means the service to be deployed may have explicit interactions with existing services. These implicit and explicit interactions have to be considered thoroughly and carefully in the deployment. Otherwise, the service to be deployed may not work well and the functionalities and qualities of the existing services may be damaged or decreased.

All activities in this approach are done with software architectures, such as understanding the service to be deployed, selecting the sharing rules, evaluating the capabilities of service providers, monitoring the working status of the services, and deploying, un-deploying or re-deploying the services.

3. The Framework

There are some key issues in the architecture model based service deployment approach, as shown in Fig. 3, such as the representation, including a formal description language and a set of visual notations, of SA in the deployment, the transformation from the models in the development into architecture models, the mechanisms for deploying, un-deploying, re-deploying and monitoring the

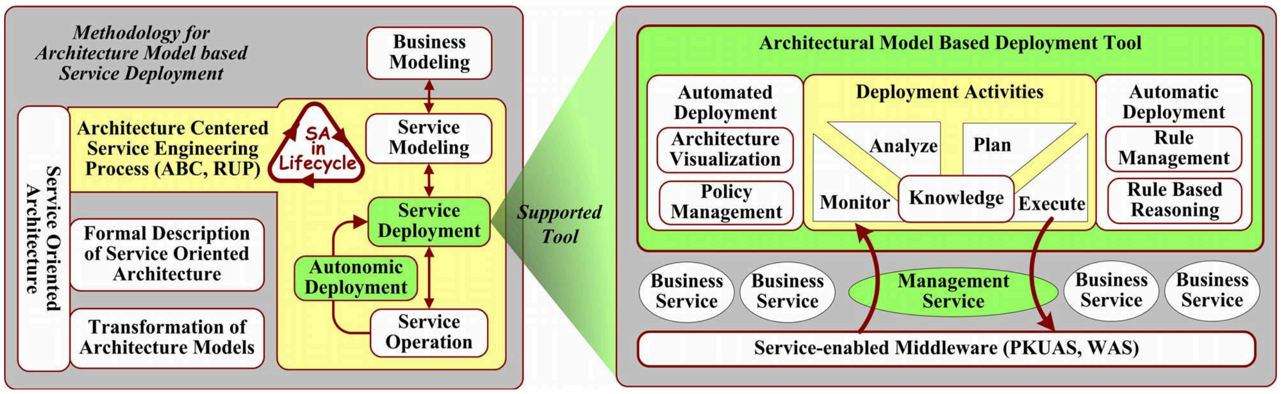


Fig. 3. Technical Framework for Software Architecture Model based Service Deployment

services, the methods for defining and managing sharing rules and evaluating the capabilities of service providers, and so on.

3.1 Architecture Centric Software Engineering

As mentioned previously, the introduction of software architecture into the deployment results in an architecture centric software engineering model. Here, we will give more details about the process with the illustration of ABC (Architecture Based Component Composition). ABC is a software reuse methodology that supports to build a software system with pre-fabricated components under the guide of software architecture [17], as shown in Fig. 4.

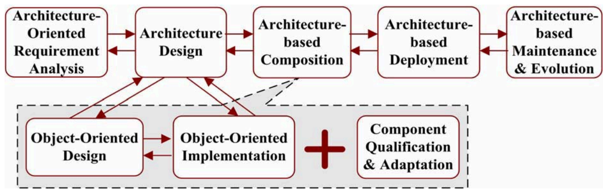

Architecture-Oriented Requirement Analysis
Architecture Design
Architecture-based Composition
Architecture-based Deployment
Architecture-based Maintenance & Evolution
Object-Oriented Design
Object-Oriented Implementation
Component Qualification & Adaptation

Fig. 4 . ABC process model.

In the requirement analysis, there is no actual software architecture but only the requirement specifications of the system to be developed, which are structured in the way similar to software architecture. The requirements specification consists of a set of component specifications, connector specifications and constraint specifications and will be used as the basis for software architecting. In the architecture design, requirements specifications are refined and some overall design decisions are made. After architecture design, the components, connectors and constraints in the reusable assets repository will be selected, qualified and adapted to implement the target system. However, there are still some elements unable to be implemented by reusable assets. These elements have to be implemented by hand in object-oriented languages or other ones.

Component-based systems are usually implemented and executed with the help of some common middleware, such as J2EE/EJB [24][23]. Before the implementation of the system being executed, it must be deployed into the middleware platform. In the deployment phase, software architecture should be complemented with some information so that middleware can install and execute the system correctly. Typically, the information includes declaration of required resources, security realm and roles, component names for runtime binding, and so on.

In some sense, the development of a system in ABC can be considered as a series of automated refinement and transformation of architecture models. Software architecture in maintenance and evolution has the most accurate and complete details of the final system. Then, it helps the deployer to get more information that cannot be obtained before runtime.

3.2 Architecture Description Language and Its Transformation for Deployment

The deployment view consists of the SA of the application and server capabilities. Developer has to input the needed information so that application can be installed and executed correctly. In fact, most of the information can be obtained from the architecture models in the design and composition phases and some can be automatically generated.

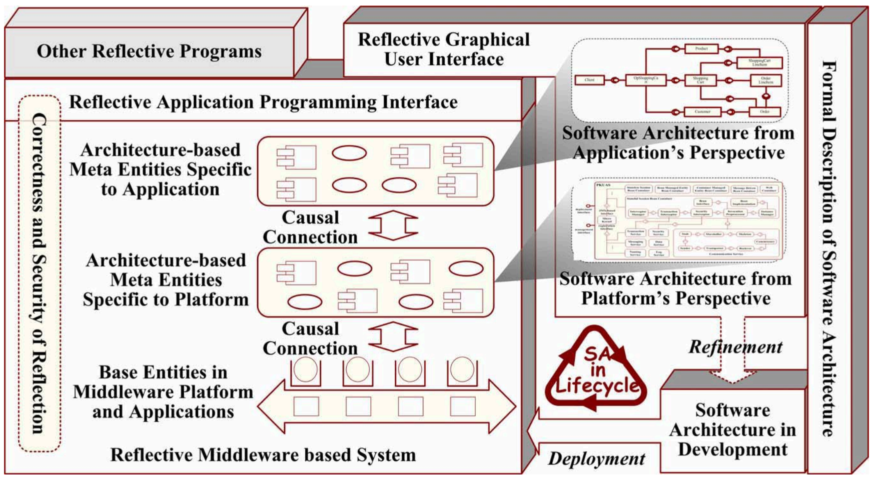
Architecture Description Language (ADL) is proposed to provide formal notations for development and analysis of software architectures [21]. ADL is the key to obtain the design and composition information. As shown in Tab. 1, the architecture model description using ABC/ADL [17] helps to generate almost all information needed in the J2EE deployment descriptor [24].

Tab. 1 The mappings between ABC/ADL elements and J2EE deployment descriptor elements

ABC/ADL Elements	J2EE Deployment Descriptor Elements
Name of *ComponentDef*	<ejb-name> and <jndi-name> in <module>
Name of the provide player of *ComponentDef*	<home> and <remote> or <local-home> and <local> in <session> or <entity>
Name of the request player of *ComponentDef*	<home> and <remote> in <ejb-ref>; <local-home> and <local> in <ejb-local-ref>
Attributes of *ComponentDef*	<env-entry>, <resource-ref>, <cmp-field> and <primkey-field>
Properties of *ComponentDef and AspectDef*	<ejb-class>, <session-type>, <persistence-type>, <prim-key-class>, <transaction-type>, <reentrant>, <security-role-ref>, <security-role>, <method-permission>

Furthermore, we employ ADL as the formal tool to describe software architectures in the deployment so as to perform some automated consistency and completeness checking. In order to describe the characteristics of service operating environments, we extend ABC/ADL to model communication functions as complex connectors and other common services as aspects.

For example, BPEL supports complex interactions among multiple services in sequential or parallel orders. Such complex interactions are scattered into a set of ADL fragments describing simple interactions between two components in traditional ADLs. Furthermore, the



Other Reflective Programs
Reflective Graphical User Interface
Reflective Application Programming Interface
Correctness and Security of Reflection
Architecture-based Meta Entities Specific to Application
Causal Connection
Architecture-based Meta Entities Specific to Platform
Causal Connection
Base Entities in Middleware Platform and Applications
Reflective Middleware based System
Software Architecture from Application's Perspective
Software Architecture from Platform's Perspective
Refinement
SA in Lifecycle
Software Architecture in Development
Deployment
Formal Description of Software Architecture

Fig. 5. Architecture-based Reflection in PKUAS

enforcement of the complex interaction is supposed to be implemented by the codes scattered into the interacting components. But now, such complex interactions can be implemented as BPEL processes which are isolated from the interacting components and executed by an independent BPEL engine. In ABC/ADL, such complex interactions are modeled as processes of a complex connector in the architecting phase. In the composition phase, the BPEL processes can be automatically generated after the involved components are implemented by services. In the deployment, the BPEL processes can be deployed independent of other components.

3.3 Software Architecture Based Maintenance and Evolution

Compared to software architectures in the development, software architectures introduced into the maintenance and evolution are available at runtime and provide more concrete views of the runtime system with much more information. We call such software architecture RSA (runtime software architecture) [12]. As shown in Fig. 5, RSA can be supported by reflective middleware, which is demonstrated on PKUAS, a reflective J2EE application server [16]. PKUAS is a J2EE-compliant application server which is the platform including J2SE, common services and one or both of Web Container and EJB Container. It provides all functionalities required by J2EE v1.3 [24] and EJB v2.0 [23]. PKUAS also implements a prototype of SOAP stack and BPEL engine.

The states and behaviors of middleware platform and applications can be observed and adapted from the perspectives of the platform RSA and application RSA respectively. The platform RSA represents the implementation of middleware platform as components and connectors. Middleware applications are invisible or represented as the attributes of some components. For example, J2EE application server consists of containers and services and the J2EE application consists of EJBs or Servlets. In the platform RSA, the containers and services are represented as components; their interactions or dependencies are represented as connectors; and the EJBs or Servlets are represented as the attributes of the containers. On the other hand, the application RSA represents middleware application as components and connectors. Middleware platform are typically represented as constraints or attributes of components and connectors. For example, J2EE security and transaction services are represented as the security and transaction constraints on the EJBs or Servlets.

4. Supporting Tool

CADTool is an assembly and deployment tool, which is based on software architecture, for J2EE applications deployed on PKUAS. It facilitates developers to visually pack as well as assemble components. More importantly, based on the software architecture, CADTool extracts most needed information in the deployment from the architecture models in the development.

Fig. 6 shows the case of deploying JPS with CADTool. The "deploy" panel shows the software architecture of Java Pet Store (JPS). JPS is one of the sample applications for J2EE Blueprints, demonstrating how to use the capabilities of the J2EE platform to develop flexible, scalable, cross-platform e-business applications. CADTool can facilitate the deployment in the following functionalities:

- Visualization of architecture models in the

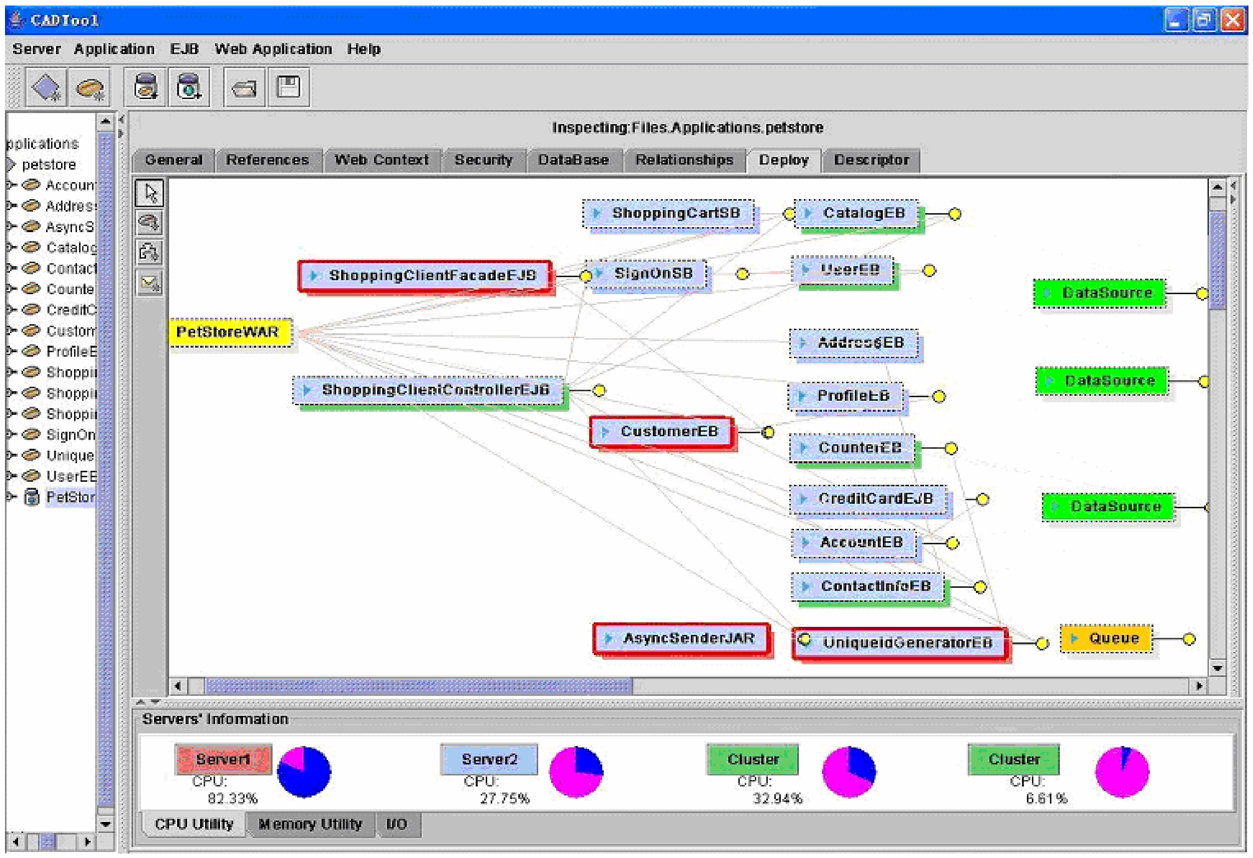


Fig. 6. Deploying JPS with CADTool

development: CADTool reuses the graphical elements of ABCTool, which supports architecture modeling with ABC/ADL in a visual way [17]. If the deployable package contains the architecture description in ABC/ADL, CADTool can display the syntax and semantics information produced in the development. If the deployable package also contains the layout description of the architecture, CADTool can display the architecture in the same layout as that in the development, which helps to understand the intention of the designers. If the deployable package does not contain the two descriptions, CADTool can automatically construct the architecture from the individual deployable components. However, the last case is not desired because the architecture lacks enough information derived from the development.

- Visualization of servers and their capabilities: Based on reflective mechanisms of PKUAS, CADTool can automatically collect and display the servers' information, such as CPU utilization, memory utilization, throughout, etc. These information is useful to determine which components should be deployed into which servers. They also help to investigate whether the deployment works well. For example, the *CatalogEJB* consumes much CPU time. If the component is deployed into the *Server1*, the CPU utilization of the *Server1* may exceed 90% and the *Server1* becomes unstable and easy to crash. Then, it'd better un-deploy the *CatalogEJB* in the *Server1* and re-deploy it into the cluster.
- Drag-and-drop deployment of components: With the above two visual elements, a component can be easily deployed into a server just through dragging the component and dropping it on the target server or vice versa. In traditional deployment tools, the deployer has to connect to a given server, load the components to be deployed into the server, and repeat the work again for another server. In Fig. 6, the VO has four servers, two of them are single servers and the other two form a cluster. The *ShoppingClientFacadeEJB*, *AsyncSenderJAR,* *UniqueidGeneratorEB* and *CustomerEJB* are deployed into the *Server1*.

- Automatic calculation of deployment factors: There are many successful case studies on the quantitative and qualitative evaluations of the given architecture models. However, some factors may be wrongly predicted in the design phase and should be re-evaluated in the deployment. Specially, some factors may be only available after the services running for a period, such as the response time and throughput. That means the deployment may not meet the requirements related to these factors. Then the services have to be re-deployed with the actual factors. Currently, CADTool can automatically calculate the response time, throughput and reliability of a given use case.

5. Related Work

In practice, the operating environment of Grid consists of the traditional operating systems, like UNIX, Linux and Windows, and a set of Grid-enabled middleware [6], such as Web Services, J2EE (Java 2 Platform Enterprise Edition), .NET, CORBA (Common Object Request Broker Architecture), Globus and Service Domain. To the best of our knowledge, almost all products of Gird-enabled middleware provide private deployment solutions. None of them helps the deployer to understand and analyze the functionality and desired qualities of the service to be deployed. None of them provide the runtime states of all resources to evaluate the best resource providers to install the services. Few of them provide the topology of the resource providers in the VO and their capability details. For examples, to deploy a service into Globus, the most famous Grid-enabled middleware, the deployer has to write a deployment descriptor and use ANT, a popular script interpreter, to generate the executable package and deploy the package into Globus runtime [9].

The traditional deployment tool in J2EE supports to deploy an application into any local or remote application servers [24]. But when one application server is being operated, no other servers can be operated in the same deployment tool. Though many J2EE application server providers have claimed that their products support Grid computing, their deployment solutions cannot deal with the above challenges efficiently. For instance, Oracle 10g has claimed as the first middleware for Grid [11]. The deployment tool of Oracle 10g application server eases the work to deploy an application into multiple servers. But in fact, it is primarily based on cluster technology. In a cluster, only the client requests are un-predictable and changeable, while in a Grid, few can be predicted and everything is changeable.

Dearle et al. propose a framework for constraint-based deployment and automatic management of distributed applications [4]. In this framework, a purely declarative and descriptive architectural description language, named Deladas, is used to describe a deployment goal. To satisfy the goal, an automatic deployment and management engine (ADME) tries to generate a configuration, which describes which components are deployed in which hosts. After the initial deployment, the ADME will monitor the deployed application to check whether the deployment satisfies the original goal and re-deploy the application if necessary. This approach has the similar philosophy to our approach on the role of software architecture in the deployment. However, this approach ignores the plentiful knowledge derived from the development and the runtime states of hosts. Without such knowledge, it is very difficult to generate the proper configuration in a manual or automated way.

Rakic et al. propose the DeSi environment to support flexible and tailorable specification, manipulation, visualization, and (re)estimation of deployment architectures for large-scale, highly distributed systems [20]. DeSi studies deeply on how to take the availability into account in the deployment, including defining a formal foundation and investigating six algorithms to automatically generate the deployment plan. However, in DeSi, the formal specification of the deployed application has to be written by hand and some values in the specification are difficult to retrieve without the support of runtime environments. On the other hand, the formal specification can be automatically generated in CADTool with the plentiful knowledge derived from the development and runtime states of hosts. In our opinion, the work of DeSi can improve the reliability calculation of CADTool, which is under development. Moreover, DeSi only takes the availability into account while CADTool tries to facilitate the tradeoff between multiple qualities.

6. Conclusion and Future Work

In this paper, an architecture model based approach to grid service deployment is proposed. With the help of software architectures, it's easy to deploy services under the guidance of their structures and achieve the desired qualities and best resource utilization according to the statistics of the runtime environment. Some key techniques are discussed in the more technical way when services are implemented by J2EE applications. We demonstrate this approach in PKUAS, a reflective J2EE application server. A graphical assembly and deployment tool, called CADTool, is also built to assist the deployment.

This paper just investigates the importance and challenge of the service deployment. The deployment tool only visualizes the factors to be considered in an automated way. The most critical work, that is, investigating the factors and determining the trade-offs, is performed by human. In that sense, the major contribution of this paper is to demonstrate the complexity of the service deployment. Then, the future work will focus on how to decrease the complexity through some automations, such as more factors can be automatically calculated, the trade-off among the factors can be automatically done, and even the architecture of the deployed services can be automatically re-constructed to achieve the best trade-offs.

Acknowledgement

This effort is sponsored by the National Key Basic Research and Development Program (973) under Grant No. 2002CB31200003; the National Natural Science Foundation of China under Grant No. 60233010, 60125206; the National High-Tech Research and Development Plan of China (863) under Grant No. 2001AA113060; the Major Project of Science and Technology Research of Ministry Of Education P.R.C. under Grant No. 0214; and the IBM-University Joint Study Program.

References

[1] .NET Framework Home, http://msdn.microsoft.com/ netframework/.
[2] Andrews, T., F. Curbera, etc, Business Process Execution Language, V1.1,May 2003
[3] Connor, J. Building e-business resiliency through autonomic computing. October 2002.
[4] Dearle, A, Kirby, GNC, McCarthy, A. In: Proc. International Conference on Autonomic Computing (ICAC-04), New York, USA, Kephart, JO, Parashar, M (eds), pp 300-301. IEEE Computer Society. 2004.
[5] Foster, I., C. Kesselman and S. Tuecke, The Anatomy of the Grid: Enabling Scalable Virtual Organizations. International Journal of High Performance Computing Applications, 2001, 15 (3), pp. 200-222.
[6] Foster, I., C. Kesselman, J. M. Nick and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. June 2002.
[7] Garlan, D., Software Architecture: A Roadmap, The Future of Software Engineering 2000, Proceedings of 22nd International Conference on Software Engineering, ACM Press, 2000, 91-101.
[8] Homepage of Catalysis, http://www. catalysis.org/.
[9] Homepage of Globus, http://www.globus.org.
[10] Homepage of IBM Grid, http://www.ibm.com/grid.
[11] Homepage of Oracle 10g. http://www.oracle.com/ appserver.
[12] Huang, G., H. Mei, Q.X. Wang. Towards Software Architecture at Runtime. ACM SIGSOFT Software Engineering Notes, Vol. 28, No. 2, March 2003.
[13] IBM. Autonomic Computing: IBM's Perspective on the State of Information Technology, http://www.ibm.com/ research/autonomic, 2001.
[14] Jacobson, I., J. Rumbaugh, and G. Booch. The Unified Software Development Process. Object Technology Series. Addison-Wesley, 1999.
[15] Kephart, J.O. and Chess, D.M. The Vision of Autonomic Computing. IEEE Computer, January 2003, pp. 41-50.
[16] Mei, H. and G. Huang. PKUAS: An Architecture-based Reflective Component Operating Platform, invited paper, 10th IEEE International Workshop on Future Trends of Distributed Computing Systems, 26-28 May 2004, Suzhou, China.
[17] Mei, H., J.C. Chang, F.Q. Yang. Software Component Composition based on ADL and Middleware, Science in China(F), 44(2), 136-151, 2001.
[18] OMG, The Common Object Request Broker: Architecture and Specification, Version 3.0, June 2002.
[19] Perry D. and A. Wolf, Foundations for the Study of Software Architecture, ACM SIGSOFT Software Engineering Notes, 1992, 17(4): 40-52.
[20] Rakic, M.M., S. Malek, N. Beckman and N. Medvidovic, A Tailorable Environment for Assessing the Quality of Deployment Architectures in Highly Distributed Settings, 2nd International Working Conference on Component Deployment, Edinburgh, UK, 2004.
[21] Shaw, M. and D. Garlan, Software Architecture: Perspectives on an Emerging Discipline, Prentice Hall, 1996.
[22] Soley, R. and the OMG Staff Strategy Group, Model Driven Architecture: OMG White Paper, Draft 3.2, http://www.omg.org/mda, Nov 27th, 2000.
[23] SUN Microsystems, Enterprise JavaBeans Specification, Version 2.0, Final Release, 2001.
[24] SUN Microsystems, Java 2 Platform Enterprise Edition Management Specification, v1.0, 2002.
[25] SUN Microsystems, Java 2 Platform Enterprise Edition Specification, Version 1.3, 2001.
[26] SUN Microsystems, Java Management Extensions Instrumentation and Agent Specification, v1.1, 2002.
[27] UML Resource Page. http://www.omg.org/uml/.
[28] Van Deursen, A. 2002. Software Architecture Recovery and Modeling: [WCRE 2001 discussion forum report]. ACM SIGAPP Applied Computing Review, 10(1): 4-7.