Architecture Based Deployment of Large-Scale Component Based Systems: The Tool and Principles

Ling Lan¹, Gang Huang¹*, Liya Ma¹, Meng Wang¹, Hong Mei¹, Long Zhang², Ying Chen²

¹School of Electronics Engineering and Computer Science, Peking University, Beijing, 100871, China
{lanling, huanggang, maly, wangmeng}@sei.pku.edu.cn, meih@pku.edu.cn
²IBM China Research Lab, No.7, St.5, ShangDi, HaiDian District, Beijing, 100085, China
{longzh, yingch}@cn.ibm.com

Abstract. After a component based system is developed, it has to be deployed into a target environment. As the system becomes much larger and more complex and the environment becomes open and dynamic, the deployment comes to be a difficult, tiring, error-prone and time-consuming task. This paper proposes an architecture based approach to deploying large-scale component based systems into open and dynamic environments in a systematic and semi-automatic manner. It does four contributions to facilitate the deployment: Firstly, a supporting tool is developed to visualize the software architecture of the system to be deployed to help deployers understand the structure, functions and desired qualities of the system. Secondly, the tool can automatically generate the deployment information from the architecture description produced in the phase of design and this will relieve deployers of inputting hundreds or thousands of deployment elements manually. Thirdly, the tool can monitor the up-to-date resource consumptions of the machines and support to partition one system into several subsystems and deploy the subsystems onto multiple machines simultaneously. Fourthly, a set of principles are proposed for guiding the deployment with the tool. The approach, especially the tool and principles are demonstrated on J2EE (Java 2 Platform Enterprise Edition).

1 Introduction

Component-Based Software Engineering (CBSE) focuses on the development of software intensive systems from pre-fabricated and reusable components, the development of components, and system maintenance and improvement by means of component replacement and customization [4][3]. As CBSE has become a prevalent approach to building the large-scale software systems, people always pay much attention to how to develop a component-based system (CBS) in a rapid, high-quality and cost-effective way but other stages of software lifecycle, especially software deployment are neglected.

* Corresponding author.

© Springer-Verlag Berlin Heidelberg 2005
Before a CBS can operate with desired functions and qualities, it should have to be configured according to the runtime environments and installed correctly. This activity is called software deployment, which plays a key role in software lifecycle. Software deployment has been attached with more and more attentions over the past decade as rapid pervasiveness of the network and distributed systems. Kruchten [21] proposes the “4+1” view model to include logical view, process view, implementation view (previously called development view), and deployment view (previously called physical view). The deployment view describes the mapping(s) of the software to the distributed nodes. In OMG’s UML (Unified Modeling Language) [20], four kinds of graphical diagrams are defined for modeling: use case diagram, class diagram, behavior diagram and implementation diagram. The last diagram includes component diagram and deployment diagram. The deployment diagrams show the configuration of runtime processing elements and the software components, processes, and objects that execute on them. In the specification of Java 2 Enterprise Edition (J2EE) [22], the development process of J2EE applications\(^1\) is divided into three stages: component creation, assembly and deployment. During the deployment stage, the J2EE application is installed on the J2EE application servers with careful configuration and integration with the runtime environments.

Before the pervasiveness of Internet, a CBS usually ran in a closed and static environment with limited users. There were only a few simple factors to be considered in deployment. But recently, the exploding Internet makes CBS larger, more complex and the runtime environments extremely open and dynamic. On the one hand, in order to deploy the CBS into a distributed environment, the factors related to the deployment should be taken into account thoroughly and carefully. Therefore the deployers are required to understand the whole system to be deployed and sometimes even other systems already deployed in the same environment. For example, we should consider the amount of components, the profiles of every component and the dependencies among components, and so on. This will help to partition one system into several sub-systems and then distribute the sub-systems to the distributed nodes respectively. However, for the large-scale CBS, to understand the whole system to be deployed is too difficult without any high-level guidance. On the other hand, the CBS is always deployed in an open and dynamic environment. Different runtime environments have different features, such as various nodes, platforms, bandwidth and topology. These differences lead to a disastrous result that the deployment of a CBS in one environment cannot be reused in another environment. Furthermore, status varies even in a single environment, such as loads and resource consumptions. That is to say, the same system to be deployed in the same environment maybe need different deployment plans at different time. Without a comprehensive understanding of the whole system and the environment, the deployers could only accomplish the deployment according to their experiences. Such deployment may lead to serious problems: the system probably can’t work correctly; the performance might descend significantly; and the newly deployed systems may impact the other systems in the same runtime environment, preventing them from operating with desired functionalities and qualities. To sum up, to deploy a CBS in modern networks, much more complex factors have to be taken into consideration and then the deployment becomes a hard, tiring, error-prone and time-consuming task.

\(^1\) In this paper, we take the application and the system synonymous.