Deploying Component–based Applications: Tools and Techniques

Abbas Heydarnoori

School of Computer Science, University of Waterloo, Canada
aheydarnoori@uwaterloo.ca

Summary. Software deployment comprises activities for placing an already developed application into its operational environment and making it ready for use. For complex component-based applications that constitute many heterogeneous components with various hardware and software requirements, this deployment process can become one of the most burning challenges. In this situation, it is difficult to manually identify a valid deployment configuration that satisfies all constraints. Thus, automated tools and techniques are required to do the complex process of software deployment. To address this requirement, a variety of tools and techniques that support different activities of the deployment process have been introduced in both industry and academia. This paper aims to provide an overview of these tools and techniques.

Key words: Software Components, Software Deployment, Deployment Life Cycle.

1 Introduction

Software deployment is a complex process that covers all post-development activities required to place an application into its target environment and make it available for use [1]. Along with significant advances in software development technologies in recent years, *component-based software development* (*CBSD*) has also gained a lot of attention in both industry and academia [2]. CBSD is a paradigm advancing a view of constructing software from reusable building blocks named *components*. According to Szyperski [3], a software component is a unit of composition with contractually specified interfaces and explicit context dependencies that can be deployed independently and is subject to composition by third parties. With respect to this definition, it is possible to have complex software systems that consist of a large number of heterogeneous components. In these applications, various components of the application often have different hardware and software requirements and hence they may provide their functionality only when those requirements are satisfied. Furthermore, software systems may undergo numerous updates during their lifetime and the components that comprise those systems are also more likely to be developed and released independently by third parties. Under these circumstances, finding a valid deployment configuration for a component-based application can be a challenging

task and it might be effectively impossible to manually find a valid deployment configuration. This situation becomes even more complex in heterogeneous, resource-constrained, distributed, and mobile computing platforms that demand highly efficient software deployment configurations [4]. Thus, support for automated software deployment process becomes crucial.

To address this requirement, software deployment process has gained a lot of attention both in industry and research community in recent years and it is possible to find a large number of tools, technologies, techniques, and papers that address various aspects of the software deployment process from different perspectives. This paper aims to provide a survey of the existing software deployment tools and techniques in both industry and academia.

This paper is organized as follows. Section 2 considers some of the existing deployment processes in literature and proposes a generic deployment process. Section 3 provides a survey of software deployment tools in industry. Section 4 surveys the research-based techniques for the deployment of component-based applications. Finally, Section 5 concludes the paper.

2 Software Deployment Process

Software deployment is a sequence of related activities for placing a developed application into its target environment and making the application ready for use. Although this definition of software deployment is reasonable and clear, different sequences of activities have been mentioned in literature for this process [5, 6, 7, 8, 9, 10, 11]. This section has a look at some of the proposed deployment processes in literature and introduces a generic deployment process that covers the activities of all of them. This generic deployment process is then used in the rest of this paper to characterize different software deployment tools and techniques.

OMG Deployment and Configuration Specification (*OMG D&C Specification*) [5] outlines the following steps in the deployment process: *packaging* the application components; *installation* which involves populating a repository with the application components; *configuring* the functionality of the installed application in the repository; *planning* the deployment; *preparing* the target environment by moving the application components from the repository to the specified hosts; and *launching* the application.

Caspian [6] defines a five-step process for the deployment of component-based applications into distributed environments: *acquisition* of the application from its producer; *planning* where and how different components of the application should be installed in the target environment, resulting in a deployment plan; *installing* the application into its target environment according to its deployment plan; *configuring* it; and finally *executing* it.

Liu and Smith [7] in their LTS framework for component deployment specify the following activities for the deployment process: *shipping* the system from the development site; *installation* of the system at the deployment site; *reconfiguring*

the system at the deployment site in response to changes; and actually *executing* the system.

The *Open Service Gateway initiative* (*OSGi*) [8] is an independent consortium of more than eighty companies launched in 1999 with the aim of developing a platform for the deployment of services over the wide-area networks to local networks and devices. In the deployment model of OSGi, a deployment process includes only the *install*, *update*, and *uninstall* activities.

Carzaniga et al. [9] in *Software Dock* research project define software deployment as an evolving collection of interrelated activities that are done on either the producer side or consumer side. The producer side activities include *release* and *retire*; the consumer side activities constitute *install*, *activate* (*launch*), *deactivate*, *reconfigure*, *adapt*, *update*, and *uninstall* (*remove*). Although this is one of the most comprehensive definitions of software deployment in literature, it lacks two activities in our view: *acquire* and *plan*. Acquisition is actually performed as part of the Software Dock's *install* activity and it is not explicitly introduced as a separate task. However, in many cases, the software systems are first acquired by the software consumers and then they are deployed on their operational environments. Moreover, as mentioned earlier, for large, distributed, component-based applications with many constraints and requirements, it is first required to plan how to install the components of the software system into its operational environment and then perform the actual installation. Hence, it is required to explicitly have a planning stage during the software deployment process. Finally, in the Software Dock deployment process, both of the *reconfigure* and *adapt* activities change the configuration of the deployed software system. The *reconfigure* activity assumes that it starts from a valid configuration of a system and then transforms it into another valid configuration. The *adapt* activity assumes it starts from an invalid configuration and then transforms it to a valid configuration. However, since both of these activities perform the same task which is changing the configuration of a software system into a valid configuration, we propose the *configure* activity rather than both of the Software Dock's *adapt* and *reconfigure* activities.

With respect to the above discussion, we propose a generic deployment process with ten activities that covers the activities of all the deployment processes mentioned in this section (Table 1). The description of these activities are as follows: (1) *release:* packages, prepares, and advertises a system for deployment into operational environments; (2) *acquire:* during this activity, the components of the application are acquired from the software producer and are put in a repository; (3) *plan:* given the specifications of the component-based application, the target environment, and user-defined constraints regarding this deployment, this activity determines where and how different components of the application will be executed in the target environment, resulting in a deployment plan; (4) *install:* this activity uses the deployment plan generated in the previous activity to install the application into its operational environment; (5) *configure:* this activity changes the configuration of an already installed software system; (6) *activate:* actually launches the application; (7) *update:* modifies a previously installed system and deploys a new, previously unavailable configuration of that system or updates the components of the system with newer

releases of those components; (8) *deactivate:* shuts down executing components of an activated system; (9) *uninstall:* completely removes the software system from its operational environment; and (10) *retire:* makes a system release unavailable.

Table 1 compares the activities of different deployment processes and shows how they are mapped to the activities of the generic deployment process. However, it should be noted that the particular practices and procedures being done in each activity highly depend on the characteristics of the software system being deployed, its operational environment, and on the requirements of software developers and users. Therefore, the generic deployment process presented in this section can be customized for specific deployment requirements. For instance, Hnetynka and Murphy in [10] adapt the OMG D&C Specification for the deployment of Java-based components into embedded systems such as mobile phones and PDAs. As another example, Coupaye and Estublier in [11] present a deployment process for the deployment of complex applications into large companies.

3 Software Deployment Technologies in Industry

A variety of technologies exist in industry to support various activities of the software deployment process. This section surveys these technologies and classifies them into six major groups. Table 2 represents these groups and their corresponding example tools. This table also characterizes these deployment technologies in terms of their support for the activities of the generic software deployment process. In this table, • represents complete support, ○ indicates partial support, and no circle means no support.

3.1 User-Driven Installers

There are many programs that are used to install and uninstall software systems from a single machine. Examples include InstallShield [12], InstallAnywhere [13],

Table 1. Mapping the activities of existing deployment processes to the activities of the generic software deployment process

		Deployment Processes in Literature				
		OMG D&C	*LTS*	*OSGi*	*Software Dock*	*Caspian*
Generic Software Deployment Process	*Release*	Packaging	Shipping	-	Release	-
	Acquire	Installation	-	-	-	Acquiring
	Plan	Planning	-	-	-	Planning
	Install	Preparation	Installation	Install	Install	Installation
	Configure	Configuration	Reconfiguration	-	Adapt & Reconfigure	Configuration
	Activate	Launch	Execution	-	Activate	Execution
	Update	-	-	Update	Update	-
	Deactivate	-	-	-	Deactivate	-
	Uninstall	-	-	Uninstall	Uninstall	-
	Retire	-	-	-	Retire	-

Table 2. Comparison of different industry-based deployment technologies in terms of their support for the activities of the generic software deployment process

Deployment Technology	**Example Tools**	**Generic Software Deployment Process**									
		Release	*Acquire*	*Plan*	*Install*	*Configure*	*Activate*	*Update*	*Deactivate*	*Uninstall*	*Retire*
User-Driven Installers	InstallShield, InstallAnywhere, Setup Factory	●			●	○		○		●	
Package Managers	Linux RPM, Fedora yum, Debian `Dpkg`	●		○	●	○		●		●	
Web-based Deployment Tools	Java Web Start, Windows Update, Microsoft ClickOnce	●	●		●	○	●	●			
Systems Management Tools	Microsoft SMS, IBM TME, Altiris			●	●	●	●	●	●	●	
Remote Sessions	Citrix, PowerTCP, SSh						●				
Publish/Subscribe Tools	TIBCO Rendezvous, IBM Gryphon, Sun JMS	●	●		●			●			

and Setup Factory [14]. These tools are typically not more than a compression tool with a user-friendly interface (e.g., a wizard). In these tools, different files of software systems are compressed into self-installing archives and are delivered to users. Then, users themselves use these tools to uncompress those archives on their intended machines. Users can also use those tools to uninstall the software systems by undoing the changes they have made during the installation. Many installers may also support some sort of configuration by which users can add or remove some functionalities from the installed software systems.

There are a number of limitations associated with user-driven installers. First, they are targeted to a single machine and it is typically impossible to use them for distributed platforms. Also, users themselves have to administer their software systems. This can have several difficulties such as: it is error prone, it is impossible to always rely on users, it might be difficult to enforce it, and finally it is hard to monitor.

3.2 Package Managers

Modern operating systems often come with package managers to assist in installing, uninstalling, and updating software systems. Linux RPM [15], Fedora yum [16], and Debian `Dpkg` [17] are representatives of this category of tools. All these tools are based on the concept of *package* and a *repository* that keeps information about the state of all installed packages. Each package comprises an archive of files to be deployed along with some metadata describing the software package such as its version.

The main goal of all package managers is to install packages in such a way that the correct dependencies among them are preserved. Package managers offer several functionalities such as creating a package, installing/uninstalling/updating a package,

verifying the integrity of an installed package, querying the repository, and checking dependencies among installed packages. However, they do not take into account the execution phase of the deployment process and the package's content may not be executable code at all. Another issue of package managers is that they are targeted to a single machine and do not support distributed systems or large scale deployments. In addition, they are also user-driven and may pose the problems mentioned for user-driven installers.

3.3 Web-based Deployment Tools

Web-based deployment tools try to use the connectivity and popularity characteristics of the Internet during the software deployment process. In these tools, it is not required to install or update the software system on every single host separately. Instead, the software application is deployed only to a single Web server. Then, client machines (users) connect to this server to download the application files or updates automatically. Representatives of these tools are Java Web Start [18], Microsoft Windows Update [19], and Microsoft ClickOnce [20]. However, one of the major limitations of these tools is that they are useless when there is no Internet connectivity.

3.4 Systems Management Tools

The term *systems management* is typically used to describe a set of capabilities (e.g., tools, procedures, policies, etc.) that enable organizations to more easily support their hardware and software resources. Systems management tools usually have a centralized architecture. In these tools, the IT administrator performs operations from a centralized location which is applied automatically to many systems in the organization. Therefore, the IT administrator is able to deploy, configure, manage, and maintain a large number of hardware and software systems from his own computer. Examples of these tools are Microsoft Systems Management Server [21], IBM Tivoli Management Environment [22], and Altiris Deployment Solution [23]. Systems management tools are all based on centralized repositories that keep deployment metadata such as client configurations and software packages. Moreover, they all support an inventory of hardware and software resources.

Systems management tools support many of the software deployment activities. In addition, all of the supported deployment activities can be done in a distributed environment as well. However, it is obvious that these tools are suitable for medium to large organizations. The issues associated with all these tools are that they are often heavy and complicated systems, they all require reliable networks, they are all based on complete administration control, and they are not viable for mobile devices.

3.5 Remote Sessions

Citrix [24], PowerTCP [25], and SSh [26] fall in this category of deployment tools. In this category, software systems are deployed to a single server machine. Then,

each client initiates a session on that server and invokes desired software systems on it. Application state is kept entirely on the server. Therefore, these tools only support the execution activity of the deployment process.

The advantages of these tools are that they reduce the inconsistencies in deployed clients when the functionality is extended and it is not required to deploy the same application to several machines. The disadvantages are server load, under-utilized client resources, and consumption of network bandwidth.

3.6 Publish/Subscribe Tools

TIBCO Rendezvous [27], IBM Gryphon [28], and Sun Java Message Service [29] are examples of this kind of tools. In this class of tools, users express their interests ("subscribe") in certain kinds of events on a server, such as installing a new application or updating installed applications. Then, whenever these events happen on that server, they will be applied ("publish") automatically to the subscribed machines. This method is an efficient approach for the distribution of data from a source machine to a large number of target machines over a network.

The limitation of this class of tools is that the users themselves have to subscribe for the deployment of applications. Furthermore, these tools might not be efficient in costly and low-bandwidth networks.

4 Software Deployment Techniques in Research

Section 3 provided an overview of software deployment technologies in industry. However, The deployment of component-based applications has been the subject of extensive research in recent years. This section considers some of the deployment techniques proposed in the research community and classifies them into eight major deployment approaches. Table 3 represents these deployment approaches and their corresponding example techniques. This table further compares these deployment techniques in terms of their support for the activities of the generic software deployment process. However, this classification is not necessarily complete and it could be extended in the future. Furthermore, these deployment approaches are not completely disjoint and the same deployment technique might fall in two or more different deployment approaches. For instance, DAnCE [30] is a deployment technique that is both QoS-aware and model-driven. However, this categorization represents different directions of interest in research-based deployment techniques.

4.1 QoS-Aware Deployment

It is typically possible to deploy and configure a large and complex component-based application into its target environment in many different ways, specifically when the target environment is a distributed environment. Obviously, some of these deployment configurations are better than others in terms of some QoS (Quality of Service)

Table 3. Comparison of different research-based deployment approaches in terms of their support for the activities of the generic software deployment process

Deployment Approach	**Example Techniques**	**Generic Software Deployment Process**									
		Release	*Acquire*	*Plan*	*Install*	*Configure*	*Activate*	*Update*	*Deactivate*	*Uninstall*	*Retire*
QoS-Aware Deployment	DeSi			●							
	MAL			●	○	○		○			
	Caspian			●							
Architecture-Driven Deployment	Prism-DE		○	●	●		●	●			
	Olan	●	●		●	●	●				
Model-Driven Deployment	OMG D&C		●	●	●	●	●	○			
	Deployment Factory		●	●	●	●	●	○			
	DAnCE		●	●	●	●	●	○	●	○	
Agent-based Deployment	Software Dock	●	●		●	●		●			
	TACOMA	●	●		●			●			
Grid Deployment	Globus Toolkit			●	●	●	●				
	ORYA		●	○	●					●	
Hot Deployment	OpenRec		●	○	●	●	●	●	●	●	
	MagicBeans		●	○	●	●	●	●	●	●	
AI Planning-based Deployment	Sekitei			●							
	CANS		●	●	●	○		○		○	
Formal Frameworks	LTS	●	●	○	●	○	●	●		●	
	Conceptual				●						

attributes such as efficiency, availability, reliability, and fault tolerance. Thus, the deployment configuration has significant impacts on the system behavior and it is necessary to consider the issues related to the quality of a deployment. To this aim, a number of research approaches have been proposed in literature that address this aspect of software deployment. Examples include *DeSi* [31], *MAL* [32], and *Caspian* [6]. DeSi provides a deployment technique for maximizing the *availability* of systems defined as the ratio of the number of successfully completed inter-component interactions in the system to the total number of attempted interactions over a period of time. MAL is a system that enables the deployment of QoS-aware applications into ubiquitous environments. Ubiquitous systems such as Internet are those that can be instantiated and accessed anytime, anywhere, and by using any computing devices. Caspian provides a deployment planning approach in which the application being deployed and its target operational environment are modeled by graphs. Then, deployment is defined as the mapping of the application graph to the target environment graph in such a way that the desired QoS parameter is optimized. This deployment problem is solved for the QoS parameters *maximum reliability* and *minimum cost* in [33] and [34] respectively.

4.2 Architecture-Driven Deployment

The software architecture research community has also addressed configuration and deployment issues for component-based applications ([4], [35], [36], [37], [38]). A software architecture represents high-level abstractions for structure, behavior, and key properties of a software system. Software architectures are usually specified in terms of *components* that define computational units, *connectors* that define types of interactions among components, and the *configuration* that describes the topologies of components and connectors. For this purpose, *Architecture Description Languages* or *ADLs* have been developed to describe software systems in terms of their architectural elements.

Prism-DE [4] and *Olan* [35] are examples of deployment techniques that employ the concepts of software architecture during the process of software deployment. In these techniques, ADLs are used to specify valid deployment configurations. Then, during the process of deployment, candidate deployment configurations are checked against those valid deployment configurations.

4.3 Model-Driven Deployment

Model-Driven Architecture (*MDA*) [39] proposed by OMG is an approach for software development based on models in which systems are built via transformations of models. MDA defines two levels of models: *Platform-Independent Model* (*PIM*) and *Platform-Specific Model* (*PSM*). Developers begin with creating a PIM, then they transform the model step by step to a more platform-specific model until the desired level of specificity is approached. In the case of software deployment, the MDA approach starts with a platform-independent model of the target environment and the transformations finish with specific deployment configurations for the considered component-based applications [40]. Model-driven deployment has gained a lot of attention in recent years ([5], [30], [40], [41], [42], [43]). In particular, *OMG D&C Specification* [5] follows MDA concepts to provide a unified technique for the deployment of component-based applications into distributed environments. The OMG D&C Specification defines three platform-independent models: the *component model*, the *target model*, and the *execution model*. To use these platform-independent models with a specific component model, they have to be transformed to platform-dependent models, capturing the specifics of the concrete platform [44]. An example of this transformation to the *CORBA Component Model* (*CCM*) can be found in [5]. However, Hnetynka in [40] mentions that OMG's approach fails in building a single environment for the unified deployment of component-based applications and it leads to several deployment environments for different component models. To address this issue, he examines several common component models (e.g., *COM*, *CCM*, *EJB*, *SOFA*, and *Fractal*) and ADLs (e.g., *Wright* and *ACME*) to identify the set of features that are missing in the OMG's D&C component model. Based on this study, he proposes a unified deployment component model and introduces *Deployment Factory* (*DF*) as a model-driven unified deployment environment. As another example of model-driven deployment approaches, *DAnCE* [30] is a QoS-enabled middleware

that conforms to the OMG D&C Specification. DAnCE enables application deployers to deploy component assemblies of distributed real-time and embedded (DRE) systems.

4.4 Agent-based Deployment

A *mobile agent* is defined as an object that migrates through many hosts in a heterogeneous network, under its own control, to perform tasks using resources of those hosts [45]. A *Mobile Agent System* (*MAS*) is defined as a computational framework that implements the mobile agent paradigm. This framework provides services and primitives that help the implementation, communication, and migration of software agents. *Software Dock* [46] and *TACOMA* [47] are two MAS examples that use mobile agents for the purpose of software deployment. Software Dock is a deployment framework that supports cooperations among software producers themselves and between software producers and software consumers. To perform the software deployment activities, it employs mobile agents that traverse between software producers and consumers. Similarly, TACOMA is a deployment framework based on mobile agents for installing and updating software components in a distributed environment such as Internet.

4.5 Grid Deployment

A *computational grid* is defined as a set of efficient computing resources that are managed by a middleware that gives transparent access to resources wherever they are located on the network [48]. A computational grid can include many heterogeneous resources (e.g., computational nodes with various architectures and operating systems), networks with different performance properties, storage resources of different sizes, and so on. To take advantage of the computational power of grids, the application deployment must be as automated as possible while taking into account application constraints (e.g., CPU, Memory, etc.) and/or user constraints to prevent the user from directly dealing with a large number of hosts and their heterogeneity within a grid. Therefore, deployment of component-based applications into computational grids has been the subject of extensive research ([48], [49], [50], [51]).

4.6 Hot Deployment

In autonomic environments, a software system can automatically adapt its runtime behavior with respect to the configuration of the drastically changing execution environment and user requirements [52]. In this context, it is required to dynamically install, update, configure, uninstall, and replace software components without affecting the reliable behavior of the system or other constituent components. In doing so, a number of research projects address the ability of dynamically deploying software

components during the program runtime, referred to as the *hot deployment* [7]. *OpenRec* [53] is a software architecture effort focusing on how to design dynamically reconfigurable systems. The *MagicBeans* platform [54] supports self-assembling systems of plug-in components that allow applications to be constructed and reconfigured dynamically at runtime. Reference [55] presents a generic architecture for the design and deployment of self-managing and self-configuring components in autonomic environments. Some other work on dynamic reconfiguration and hot deployment are [56], [57], [58], [59], [60], and [61].

4.7 AI Planning-based Deployment

Planning the deployment of component-based applications into network resources has also gained attention in the AI research community. Two reasons have been mentioned for this [62]: (1) the requirement to satisfy the qualitative (e.g., reliability) and quantitative (e.g., disk quota) constraints; and (2) the fact that software deployment may involve selecting among compatible components as well as insertion of auxiliary components. Because of these reasons, AI planning-based techniques have been introduced for the purpose of software deployment. *Sekitei* [63] provides AI planning techniques for deploying components into resource-constrained distributed networks. *CANS* planner [64] finds optimal deployment of components along network paths. *Pegasus* [65] is a planning architecture for building grid applications. *Ninja* planner [66] makes directed acyclic graph (DAG) structured applications by using the available components in the network. *Panda* [67] has a database of predefined plan templates and simply instantiates a suitable template based on the programmer-provided rules that decide whether or not a component can be instantiated on a network resource.

4.8 Formal Deployment Frameworks

There are few works that provide platform-independent formal frameworks for the deployment of component-based applications. In these frameworks, different activities of the software deployment process are defined formally in a platform-independent manner that are suitable for derivation of theoretical results. For example, they can give deployment tool developers a theoretical basis to implement systems with well-defined behavior. Examples of these frameworks are *LTS* [7] and *conceptual foundation* [68]. LTS provides formalisms for almost all the activities of the software deployment process. The conceptual foundation only proposes conditions under which various software installation strategies are safe and successful.

5 Conclusions

An application can provide its expected functionality only when it is deployed and configured correctly in its operational environment. As a result, software deployment

is a critical task that takes place after the development of an application. Inspired by the work by Carzaniga et al. [9], this paper defined software deployment as a process comprising ten activities related to releasing, acquiring, planning, installing, configuring, activating, updating, deactivating, uninstalling, and retiring software systems. However, for many modern component-based applications, the deployment process is complicated by the dimensions along which a system can be configured. Thus, a variety of tools and techniques have been introduced in both industry and academia to address this problem. This paper surveyed a set of representatives of these tools and techniques, and assessed them in terms of their support for the activities of the deployment process. This assessment indicated that there is no deployment tool or technique that can support the full range of deployment process activities. This suggests the deployment of component-based applications as an open problem that requires further research by the research community.

References

1. D. Heimbigner, R.S. Hall, and A.L. Wolf. A framework for analyzing configurations of deployable software systems. In *ICECCS*, 1999.
2. Ivica Crnkovic, Brahim Hnich, Torsten Jonsson, and Zeynep Kiziltan. Specification, implementation, and deployment of components. *Commun. ACM*, 45(10):35–40, 2002. ISSN 0001-0782.
3. C. Szyperski. *Component Software - Beyond Object-Oriented Programming*. Addison-Wesley, 1999.
4. M. Mikic-Rakic and N. Medvidovic. Architecture-level support for software component deployment in resource constrained environments. In *CD*, LNCS 2370, 2002.
5. Deployment and configuration of component-based distributed applications specification. `http://www.omg.org/docs/ptc/04-05-15.pdf`.
6. A. Heydarnoori. Caspian: A QoS-aware deployment approach for channel-based component-based applications. Technical Report CS-2006-39, David R. Cheriton School of Computer Science, University of Waterloo, 2006.
7. Y.D. Liu and S.F. Smith. A formal framework for component deployment. In *OOPSLA*, 2006.
8. OSGi Alliance. `http://www.osgi.org/`.
9. A. Carzaniga, A. Fuggetta, R.S. Hall, A.V.D. Hoek, D. Heimbigner, and A.L. Wolf. A characterization framework for software deployment technologies. Technical Report Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, 1998.
10. Petr Hnetynka and John Murphy. Deployment of Java-based components in embedded environment. In *IADIS Applied Computing*, 2007.
11. T. Coupaye and J. Estublier. Foundations of enterprise software deployment. In *CSMR*, 2000.
12. InstallShield Developer. `http://www.installshield.com/isd/`.
13. Zero G software deployment and lifecycle management solutions. `http://www.zerog.com/`.
14. Setup factory. `http://www.indigorose.com/suf/`.
15. RPM package manager. `http://www.rpm.org/`.

16. Yum: Yellow dog updater. http://linux.duke.edu/projects/yum/.
17. Package maintenance system for Debian. http://packages.debian.org/dpkg/.
18. Java web start technology. http://java.sun.com/products/javawebstart.
19. Microsoft windows update. http://update.microsoft.com.
20. ClickOnce: Deploy and update your smart client projects using a central server. http://msdn.microsoft.com/msdnmag/issues/04/05/clickonce/.
21. Systems management server home. http://www.microsoft.com/smserver/.
22. IBM Tivoli software. http://www.tivoli.com/.
23. Altiris deployment solution. http://www.altiris.com/.
24. Citrix. http://www.citrix.com/.
25. PowerTCP. http://www.dart.com/powertcp/.
26. Secure shell (SSH). http://www.ssh.com/.
27. TIBCO Rendezvous. http://www.tibco.com/software/messaging/.
28. IBM Gryphon. http://www.research.ibm.com/distributedmessaging/.
29. Java message service (JMS). http://java.sun.com/products/jms/.
30. G. Deng, J. Balasubramanian, W. Otte, D. Schmidt, and A. Gokhale. DAnCE: A QoS-enabled component deployment and configuration engine. In *CD*, LNCS 3798, 2005.
31. M. Mikic-Rakic, S. Malek, and N. Medvidovic. Improving availability in large, distributed component-based systems via redeployment. In *CD*, LNCS 3798, 2005.
32. D. Wichadakul and K. Nahrstedt. A translation system for enabling flexible and efficient deplyoment of QoS-aware applications in ubiquitous environments. In *CD*, LNCS 2370, 2002.
33. A. Heydarnoori and F. Mavaddat. Reliable deployment of component-based applications into distributed environments. In *ITNG*, 2006.
34. A. Heydarnoori, F. Mavaddat, and F. Arbab. Deploying loosely coupled, component-based applications into distributed environments. In *ECBS*, 2006.
35. R. Balter, L. Bellissard, F. Boyer, M. Riveill, and J. Y. Vion-Dury. Architecturing and configuring distributed application with Olan. In *Middleware*, 1998.
36. V. Quema and E. Cecchet. The role of software architecture in configuring middleware: The ScalAgent experience. In *OPODIS*, LNCS 3144, 2003.
37. V. Quema and et al. Hierarchical, and scalable deployment of component-based applications. In *CD*, LNCS 3083, 2004.
38. J. Matevska-Meyer, W. Hasselbring, and R. H. Reussner. Software architecture description supporting component deployment and system runtime reconfiguration. In *WCOP*, 2004.
39. OMG model driven architecture. http://www.omg.org/mda/.
40. P. Hnetynka. A model-driven environment for component deployment. In *SERA*, 2005.
41. A. Hoffmann and B. Neubauer. Deployment and configuration of distributed systems. In *SAM*, LNCS 3319, 2004.
42. N. Belkhatir, P. Cunin, V. Lestideau, and H. Sali. An OO framework for configuration of deployable large component based software products. In *OOPSLA ECOOSE Workshop*, 2001.
43. S. Jansen and S. Brinkkemper. Modelling deployment using feature descriptions and state models for component-based software product families. In *CD*, LNCS 3798, 2005.
44. L. Bulej and T. Bures. Using connectors for deployment of heterogeneous applications in the context of OMG D&C Specification. In *INTEROP-ESA*, 2005.
45. D. Rus, R. Gray, and D. Kotz. Transportable information agents. In *AAMAS*, 1997.

46. R. S. Hall, D. Heimbigner, and A. L. Wolf. A cooperative approach to support software deployment using the software dock. In *ICSE*, 1999.
47. N. P. Sudmann and D. Johansen. Software deployment using mobile agents. In *CD*, LNCS 2370, 2002.
48. S. Lacour, C. Perez, and T. Priol. A software architecture for automatic deployment of CORBA components using grid technologies. In *DECOR*, 2004.
49. S. Lacour, C. Perez, and T. Priol. Deploying CORBA components on a computational grid: General principles and early experiments using the Globus Toolkit. In *CD*, LNCS 3083, 2004.
50. V. Lestideau and N. Belkhatir. Providing highly automated and generic means for software deployment process. In *EWSPT*, 2003.
51. P. Brebner and W. Emmerich. Deployment of infrastructure and services in the open grid services architecture (OGSA). In *CD*, LNCS 3798, 2005.
52. R. Murch. *Autonomic Computing*. Prentice Hall, 2004.
53. J. Hillman and I. Warren. An open framework for dynamic reconfiguration. In *ICSE*, 2004.
54. R. Chatley, S. Eisenbach, and J. Magee. Magicbeans: A platform for deploying plugin components. In *CD*, LNCS 3083, 2004.
55. E. Patouni and N. Alonistioti. A framework for the deployment of self-managing and self-configuring components in autonomic environments. In *WoWMoM*, 2006.
56. A. Akkerman, A. Totok, and V. Karamcheti. Infrastructure for automatic dynamic deployment of J2EE applications in distributed environments. In *CD*, LNCS 3798, 2005.
57. H. Cervantes and R. S. Hall. Autonomous adaptation to dynamic availability using a service-oriented component model. In *ICSE*, 2004.
58. M. W. Hicks, J. T. Moore, and S. Nettles. Dynamic software updating. In *PLDI*, 2001.
59. H. Liu and M. Parashar. A component-based programming framework for autonomic applications. In *ICAC*, 2004.
60. S. R. Mitchell. *Dynamic Configuration of Distributed Multimedia Components*. PhD thesis, University of London, 2000.
61. J. Paula, A. Almeida, M. Wegdam, M. V. Sinderen, and L. Nieuwenhuis. Transparent dynamic reconfiguration for CORBA. In *DOA*, 2001.
62. T. Kichkaylo and V. Karamcheti. Optimal resource-aware deployment planning for component-based distributed applications. In *HPDC*, 2004.
63. V. Kichkaylo, A. Ivan, and V. Karamcheti. Constrained component deployment in wide-area networks using AI planning techniques. In *IPDPS*, 2003.
64. X. Fu and V. Karamcheti. Planning for network-aware paths. In *DAIS*, LNCS 2893, 2003.
65. J. Blythe, E. Deelman, Y. Gil, C. Kesselman, A. Agarwal, G. Mehta, and K. Vahi. The role of planning in grid computing. In *ICAPS*, 2003.
66. S. Gribble and et al. The Ninja architecture for robust internet-scale systems and services. *Computer Networks*, 35(4):473–497, 2001.
67. P. Reiher, R. Guy, M. Yavis, and A. Rudenko. Automated planning for open architectures. In *OpenArch*, 2000.
68. A. Parrish, B. Dixon, and D. Cordes. A conceptual foundation for component-based software deployment. *The Journal of Systems and Software*, 57(3):193–200, 2001.