Deploying Component Software Systems

OMG-DARPA Workshop on Compositional Software Architectures
Position Paper

Dennis Heimbigner, Alexander Wolf, Richard Hall, and André van der Hoek

Department of Computer Science
University of Colorado
Boulder, CO 80309 USA
{dennis,alw,rickhall,andre}@cs.colorado.edu

1 Introduction

The configuration and deployment of software is a complex process which has largely been ignored or treated in an ad-hoc fashion. The Software Engineering Research Laboratory at the University of Colorado is attempting to address this issue by providing infrastructure to systematically enable distributed deployment and configuration management of software systems.

Deployment of software systems that are constructed through composition of pre-existing components adds a new layer of complexity over the deployment of more monolithic systems. This is because the component pieces comprising a system can and will evolve independently of whole system. On one hand, these changes can threaten the integrity and operation of the whole system. On the other hand, component evolution can also provide new and and valuable capabilities to the systems that incorporate them.

We envision that future deployment of software systems will increasingly be accomplished using the Internet, and that the components comprising a software system will be obtained from a variety of possibly independent software providers through the Internet.

Network delivery of software components has a number of advantages over other media; advantages such as timeliness and constant evolution. Networks provide a more timely method for a software producer to make software available to its consumers. As soon as a new software system version or update becomes available, customers can be given access to it The constant evolution of the content available through network distribution makes it possible to add value to software systems in a way that is not possible using other media. For example, by leveraging the timeliness of network distribution and the connectivity that it provides, it is possible for software systems and software producers to perform some management responsibilities like updating or adapting the software system.

2 Software Deployment

Software deployment is not a simple, monolithic process to be performed after a software system has been developed. Software deployment is a collection of interrelated activities that address all of the issues of interfacing a deployed software system to the ongoing usage of the consumer as well as the ongoing development efforts of the producer(s). These activities are collectively referred to as the software deployment life cycle and their relationships are depicted in figure 1. The activities are described in more detail in subsequent paragraphs.

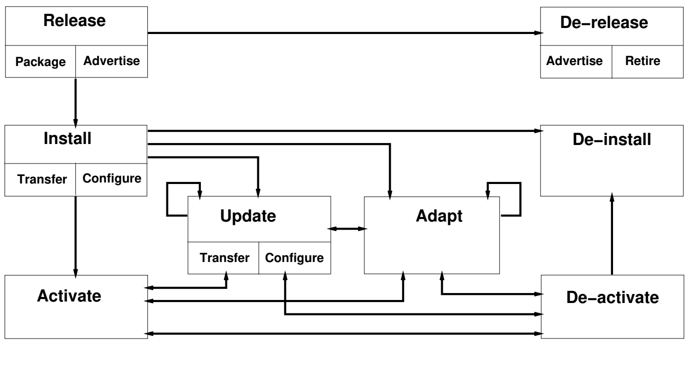

Figure 1: The Software Deployment Process.

Release: The release activity connects development to deployment. It encompasses all the activities needed to prepare and advertise a system so that it can be obtained and assembled correctly at a consumer site. Advertising includes the dissemination of sufficient information to interested parties and providing some form of access to the software system so that follow-on activities can be performed.

Installation: The installation activity covers the initial deployment of a software system onto a consumer site. It is usually the most complex of the deployment activities because it must find and assemble all the resources necessary to use a system. The installation process uses the information created by the release process. Given a released package, the installation process combines the knowledge encoded in the package with the information at a target site in order to determine how to properly configure the software system to the specific target site. Component-based software complicates this activity since the installation may need to obtain appropriate components from several Internet sites and install each one at the target site before the installation of the whole system can be completed.

Activation: Activation is the activity of starting up the components of a system that must execute in order for the system to be usable. For a complex compositional system, there will be components that must run continuously in order for the system to be usable. Examples of the latter might be various object servers and database systems needed by the system.

De-activation: De-activation, the inverse of activation, is responsible for shutting down any executing components of an installed system. De-activation may be required in order to perform other deployment activities such as update (see Section 3).

Update: The update process involves modifying a software system that has been previously installed on a consumer site. The update may be the result of the release of a new version of the software to fix a bug or add new functionality. It might also be the result of the release of a new version of a separately developed component that has been used in the composition of an already installed system. Typically the deployment life cycle includes a repeated sequence in which a system is (optionally) de-activated, an updated version of the system is installed, and then the system is reactivated.

Adapt: The adapt process involves modifying a software system that has been previously installed on a consumer site. Adapt differs from update in that updates are instigated by remote events whereas adaptations are instigated by local events. For example, if the configuration at the consumer site changes in a way that affects the deployed software system it may be necessary for the deployed software system to take some sort of corrective action.

De-installation: A software system may eventually become unneeded at a given consumer site and the system will need to be de-installed. De-installation is not necessarily a trivial process. Special attention has to be paid to shared resources such as data files and components in order to prevent dangling references to the required resource. De-installation is therefore not the process of undoing everything that was done in installation, rather it is the examination of the current state of the system and its dependencies and constraints and the removal of the specific software system artifacts that will not violate these dependencies and constraints.

De-release: Ultimately a system or component is marked as obsolete and support by the producer is withdrawn. De-release is distinct from de-installation in the sense that it makes the software no longer available for installation at consumer sites, but it does not necessarily remove it from the consumer sites that are using it. Consumers of the software system may continue to use the system without knowing that it has been marked as obsolete, but the de-release process has an obligation to notify current users that support for the software has been withdrawn.

3 The Software Dock

The Software Dock [3] is a distributed, agent-based architecture that we have developed to support the deployment life-cycle process. The Software Dock operates in a context of multiple *release sites* and *field sites*. A release site is a site in a network from which software is released. A field site is one to which software is deployed. Our architecture is strongly agent-driven, where an agent in our architecture is an executable program that performs specific tasks of the deployment process at release and field sites on behalf of producers and consumers.

Each release site and each field site has associated with it what we call a *dock*, which is an environment for hosting agents and for maintaining information about that site. A dock provides an interface through which agents can query and manipulate the information. In general, there can be many field and release docks representing the interests of the many possible participants in the deployment process.

Most of the functionality of field and release docks is embodied in the set of agents that they host at any given time. Agents register for specific events and then perform specific tasks once these events have been received. The actions performed by agents can cause other events to be generated, thus stimulating the system with additional event-action responses.

We intend the Software Dock to be strongly policy driven. For example, one policy might be to require all attempts to update a component to be screened and scheduled by a site administrator so that existing software systems are not disrupted. To support a wide variety of possible policies, the basic agents provided with the Dock are *policy-neutral*. The basic agents are combined with product specific declarative information (e.g., constraints and dependencies) both from the product site and from the target site to produce effective agents embodying specific policies.

To illustrate our use of agents in the Software Dock, consider a scenario involving the automated update of a server object in a component-based system. In order for a running server to be properly updated, a carefully orchestrated set of activities must be performed. First, the clients must be deactivated. Then the server must be deactivated. The server can then be updated. After this, the server must be reactivated. Finally, the clients must be reactivated. Each activity can be handled by an agent associated with the specific activity (e.g., deactivation) for the specific component system (e.g., the client) at a particular site (e.g., a Windows 95 platform). In effect, agents in the Software Dock abstract the dependencies and constraints imposed by the cross product of activity, component, and environment.

4 Experience Base

The Dock prototype has been successfully applied as a demonstration to the deployment of the Lockheed-Martin OLLA system, which consists of some 1700 files and 45 megabytes of data, including web pages, video, and software. Additionally, OLLA recursively depends on the University of Colorado Harvest system, and the dock is capable of installing Harvest in the process of installing OLLA.

In a related experiment, the Dock has been used in a Lockheed-Martin demonstration of their EVOLVER helpdesk system. The Dock was used to complete the cycle from problem report to updating fielded software.

Additionally, we have been tracking standardization efforts that are relevant to the deployment problem.

- Open Software Description [4] (OSD) – a proposed W3 Consortium standard created jointly by Microsoft and Marimba.

- Desktop Management Task Force (DMTF) – an industry consortium chartered to develop management standards for personal computer systems and products. An initial result of the DMTF effort was the Desktop Management Interface [1] (DMI), which creates a common interface layer to access management information on computing systems. A related effort to DMI is the Management Information Format (MIF) that is a common, hierarchical data model used in describing all aspects of computing systems, including software systems.

Where possible, we are providing commentary and critiques on these standards. The document [2], for example, provides detailed commentary on OSD.

5 Conclusion

Deploying compositional software looms as an important but little recognized problem, and it is especially critical as deployment across the Internet becomes the norm. We are developing a comprehensive deployment life-cycle process to cover all aspects of deployment. We are also developing prototype systems, notably the Software Dock, to support all phases of deployment of compositional systems.

References

[1] Desktop Management Task Force. *Desktop Management Interface Specification, Version 2.0*, 27 March 1996.

[2] Richard S. Hall, Dennis Heimbigner, and Alexander L. Wolf. Software deployment languages and schema. (http://www.cs.colorado.edu /users/rickhall/deployment/SchemaPaper/ schema.html).

[3] R.S. Hall, D.M. Heimbigner, A. van der Hoek, and A.L. Wolf. An Architecture for Post-Development Configuration Management in a Wide-Area Network. In *Proceedings of the 1997 International Conference on Distributed Computing Systems*, pages 269–278. IEEE Computer Society, May 1997.

[4] A. van Hoff, H. Partovi, and T. Thai. The open software description format. Technical report, W3C, 13 August 1997. (http://www.w3.org/TR/NOTE-OSD.html).