Software Deployment
Extending Configuration Management Support into the Field

André van der Hoek, Richard S. Hall, Antonio Carzaniga,
Dennis Heimbigner, and Alexander L. Wolf
University of Colorado

Traditionally, configuration management has only addressed the needs of the software development process. Once a software system leaves the development environment and enters the field, however, there still is a significant role for configuration management. Activities such as release, installation, activation, update, adaptation, deactivation, and de-release constitute the "deployment lifecycle"; these activities need careful coordination and planning in their own right. This article discusses the dimensions of software deployment, argues why current solutions are not sufficient, and presents two research systems that specifically address software deployment.

The management of software systems after they have been deployed is an emerging problem that manifests itself in numerous places. Software on the Mars Pathfinder rover Sojourner was regularly changed by mission control to give it new behavior in the exploration of Mars. During the Persian Gulf war, software on Patriot missiles stationed in Israel and Saudi Arabia was updated with U.S.-implemented patches to increase their ability to intercept hostile Iraqi missiles. Also, software for on-board military aircraft computers is continually adapted to compensate for the various missiles carried on each mission.

All these activities are part of the software deployment process, which is defined as

> The *delivery*, *assembly*, and *maintenance* of a particular *version* of a *software system* at a *site*.

The elements of the above definition can be illustrated in more detail using the Mars Pathfinder as an example. The *site* in this case is the rover, whereas the *software system* is the software that controls the movement of the rover over the planet as well as the operation of its measuring instruments. The *delivery* is the transmission from Earth of new code or new data to the Pathfinder, whereas the *assembly* process ensures consistency in the inclusion of new code or data in the system already present at the rover. This results in multiple *versions* of the software system to be present at the rover, which need to be *maintained* by ground control.

At first sight, the software deployment processes for the Mars Pathfinder, Patriot missiles, and military aircraft seem vastly different. However, a significant amount of commonality exists among these and many other deployment processes. This article examines this commonality. We first define the software deployment lifecycle, which consists of the series of activities normally carried out during the deployment of a software system. We then examine some existing solutions and demonstrate why these solutions are not sufficient to solve all deployment problems. We conclude with a brief look at two research prototypes we have been constructing that provide a radically different approach to software deployment.

This work was supported in part by the Air Force Materiel Command, Rome Laboratory, and Defense Advanced Research Projects Agency (DARPA) under contract number F30602-94-C-0253, and in part by DARPA and the Office of Naval Research (ONR) under ONR contract number N66001-95-D-8656, delivery order 0001. The content of the information does not necessarily reflect the position or the policy of the U.S. government and no official endorsement should be inferred.

Software Deployment Lifecycle

A software system's general deployment process is composed of a variety of subprocesses or activities. Figure 1 lists these activities and organizes them into an overall deployment lifecycle. Following is a more detailed discussion of each activity.

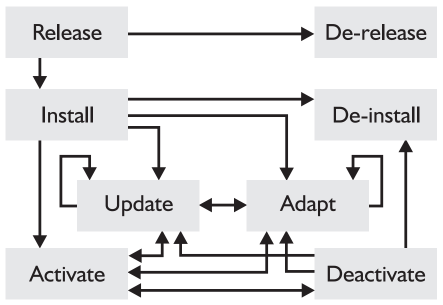
Release

The release process is the interface between the development process and the deployment process. It encompasses all activities needed to prepare and advertise a system so that it can be assembled correctly at another site. The result of this activity is a package that contains the system components, the systems dependencies and constraints, and information needed for the other deployment steps. In addition, this package is advertised.

Install

The installation activity covers the initial insertion of a system onto a site. It is usually the most complex of the deployment activities because all the necessary resources must be found and assembled. In the installation process, the package created in the release process is used, then the encoded knowledge is interpreted and the target site is examined to determine how to properly configure the software system.

Figure 1. *Software deployment lifecycle.*



		Release		Install		Activate	Update		Adapt	Deactivate	De-install	De-release
		Package	Advertise	Transfer	Configure		Transfer	Reconfigure				
Content Delivery	Castanet		o	•			•					
	PointCast		•	•								
	Rsync			•			•					
	Rdist			•								
Install and Update	NetInstall	o		o	o						•	
	NetDeploy	o		o	o		o	o			•	
	InstallShield	o			•						•	
	RPM	•		o	•		o	•	o		•	
	HP-UX SD	•			•			•	o		•	
Standards	MIF											
	AMS											
	Autoconf				o			o				
Network Mgt.	TME-10			•	•	•	•	•	o	•	•	
	Platinum			•	•	•	•	•	o	•	•	
	Nebula			•	•	•	•	•		•	•	
	Novadigm				•	•		•		•	•	

Table 1. *Evaluation of software deployment lifecycle support coverage. The release, install, and update activities have been split into two subactivities to better highlight system features. "o" indicates some support, "•" is better-than-average support.*

Activate
Activation refers to starting up those components of a system that must execute for the system to be usable. For large systems in particular, activation can be especially complex because it involves the start-up of servers, clients, database systems, etc.

Deactivate
Deactivation is the inverse of activation, and refers to shutting down any executing components of a system. Deactivation is often required to perform other deployment activities, e.g., a software system may need to be deactivated before an update can be performed.

Update
The update process involves modifying a software system that has been previously installed on a site. An update is a new version of a software system that fixes a bug or adds new functionality. Updates are normally less complex than installations because many of the needed resources have already been obtained during the installation process. Although a system is usually deactivated before an update, this is not always the case. For some systems, there is a stringent requirement that they continue to operate while being updated. In such cases, the update process can be quite complex.

Adapt
The adapt process involves modifying a software system that has been previously installed at a 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 of a site changes in a way that affects the deployed system, it may be necessary for the deployed software system to take corrective action.

De-install
At some point, a system as a whole is no longer required at a site and can 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 libraries to prevent dangling references.

De-release
Ultimately, a system may be marked obsolete, and support by its producer is withdrawn. De-release is distinct from de-installation in the sense that the software system becomes unavailable for further installation at sites, but it is not removed from sites that are using the software.

Classification
The recent emergence of Internet-based deployment systems has created a renewed interest in software deployment. However, solutions for software deployment problems have been around for decades, and many deployment systems exist that support one or more activities of the deployment lifecycle. These existing systems can be categorized into some combination of the four classes discussed below.

Content Delivery
In this class of systems and technologies, the information being deployed is merely transferred from one or more sources to a number of receiving sites. No customization is carried out once this information has been placed at a site. In essence, content delivery systems provide a replication mechanism between source and target sites.

System Install and Update
These systems deal with the localization of a software system to the environment

provided by a site. Both an initial localization and incremental updates are supported. Most deployment systems fall into this class.

Standardization

Because of the large amount of information that needs to be managed during deployment of a software system, it is not surprising that some efforts for standardization have taken place. Standards generally focus on creating a standard template to describe software systems and deployment sites. These templates are used by deployment systems to manage the deployment process.

Network Management

In the past, these systems have dealt with managing hardware systems in a heterogeneous network. Over time, they have grown to include some deployment activities, such as installation, update, and de-installation. The systems in this class often operate in a centralized notion; they assume "dumb" target sites.

Current Solutions

We have examined a representative set of systems from each of the above classes with respect to coverage of the deployment lifecycle. The results of this evaluation can be found in Table 1. However, such a simple examination is not sufficient to fully characterize deployment systems. We also have to look at several other characteristics of deployment systems.

We strongly emphasize the importance of the changeability and parameterization of the deployment process embodied in deployment systems. This requires deployment systems to operate at a certain level of abstraction. The primary abstractions are the target site, the system to be deployed, and the process. As more powerful modeling capabilities for these abstractions are included in a deployment system, two benefits arise. First, simple systems can be deployed using generic processes. Second, more complex software systems can be successfully deployed. For example, content delivery systems often do not model the software system to be deployed, whereas system installers have knowledge about the composition of a software system. Therefore, updates done by content delivery systems are often based on an algorithmic difference, but updates done by systems installers are mostly based on a firsthand understanding of the components that need to be changed.

Another important aspect to examine is support for the coordination of distributed, cooperating software systems. These systems contribute to the growing complexity of software deployment because their architectures have inherently complex, unreliable relationships and dependencies. Such architectures require special support to deploy successfully because coordination among servers, peers, and clients may be necessary. These coordination issues complicate the activities of the software deployment lifecycle.

Table 2 presents the results of the second part of our examination. The table reflects the ability of each examined deployment system to model, change, and parameterize the target site, the system to be deployed, and its embodied process. It also presents each system's ability to support the deployment of distributed, coordinated systems.

		Site Abstraction	System Abstraction	Process Abstraction	Coordination
Content Delivery	Castanet				
	PointCast				
	Rsync				
	Rdist				
Install and Update	NetInstall		o		
	NetDeploy		o		
	InstallShield		o		
	RPM	o	●	o	
	HP-UX SD	o	●	o	
Standards	MIF	●	o		
	AMS	●	●		
	Autoconf		o		
Network Mgt.	TME-10	●	●		●
	Platinum	●			●
	Nebula	●	●		●
	Novadigm	●	●		●

Table 2. *Evaluation of abstraction and coordination capabilities.* "o" *indicates some support,* "●" *is better-than-average support.*

University of Colorado Approach

As Tables 1 and 2 show, none of the existing deployment systems support the entire software deployment lifecycle. More important, support for appropriate abstraction and coordination is lacking in most systems. What is needed is an all-encompassing, highly parameterized, unifying approach to software deployment. Because it is unlikely that ad hoc combinations of existing systems would yield the desired result, the Software Engineering Research Laboratory (SERL) at the University of Colorado at Boulder, funded by the Defense Advanced Research Projects Agency (DARPA)-sponsored Evolutionary Design of Complex Software (EDCS) program is researching software deployment systems. Two prototype systems have been created, each of which is briefly described below.

SRM – A Software Release Manager

Software Release Manager (SRM) focuses on the release activity of the deployment process. It supports the release of systems of systems from multiple, geographically distributed organizations. In particular, SRM tracks dependency information to automate and optimize the retrieval of components. Both developers and users of software systems are supported by SRM. Developers are supported by a simple release process that hides distribution. Users are supported by a simple retrieval process that allows the retrieval, via the Web, of a system of systems in a single step as a single package.

New Software Engineering Mailing List

The Software Engineering Research Laboratory at the University of Colorado at Boulder invites you to subscribe to a new, noncommercial mailing list for the software engineering community: SEWORLD@cs.colorado.edu.

SEWORLD will serve as a central place for relevant announcements of software engineering conferences, workshops, symposiums, special journal issues, calls for papers, research and educational systems, etc.

The list is moderated to avoid junk E-mail, duplication, and other misuses. In addition, all E-mail addresses are registered privately to the list, are not published, and will not be given to anyone requesting them.

Simple instructions on how to subscribe or contribute to SEWORLD are on the SEWORLD Web site at

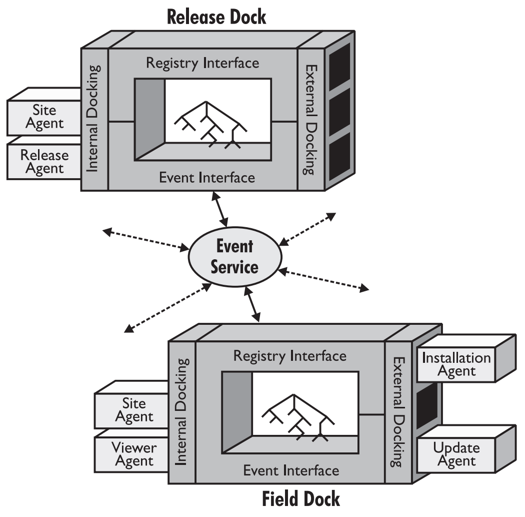
http://www.cs.colorado.edu/serl/seworld

SRM is freely available and supported on most UNIX platforms. It has been used extensively by our research laboratory to release our software and is currently being deployed to the approximately 50 participants in the EDCS program. This will provide the EDCS program with a single location from which all technology created by EDCS can be browsed and retrieved, despite that the technology is created by organizations spread over the United States.

Software Dock – A Software Deployment Architecture

Extending the ideas of SRM, the Software Dock constructs an architecture that supports *all* of the activities in the software deployment process. The Software Dock relies not only on a standard available *release dock* at a producer site (similar to SRM), but also on a standard available *field dock* at a target site (see Figure 2). Between these docks, *agents* accompany software systems as they are being deployed. The agents represent the various deployment activities and operate on the semantic data (available in the registries of the release and field docks) to properly deploy a software system. Standard agents are available that model most common deployment activities. These agents can be parameterized to carry out more specific deployment activities as required.

Figure 2. *Software Dock architecture.*



Central to the Software Dock approach is that all dimensions are customizable. In particular, the release dock is the abstraction for a software system to be deployed, the field dock is the abstraction for the target site, and the agents are the abstraction for the deployment process activities.

A prototype of the Software Dock is currently being implemented. Previous incarnations have been created, evaluated, and constructed based on a collaboration with Lockheed Martin, wherein an existing 6,000-line Perl-based installation script was reduced to a small installation agent derived from a generic installation template.

Conclusion

Although software deployment is an ever-present activity as software systems are being developed, structured support for the deployment process has been remarkably lacking until now. Various deployment systems have been created, but no system comes close to providing a single solution that can "do it all." Based on a well-defined deployment lifecycle, we have highlighted some of the complex issues, partial solutions, and research to define and build the software deployment systems of tomorrow. ◆

Further Reading

Space limitations do not allow us to treat all issues in as much depth as they deserve. Please visit the following Web sites for further information on deployment issues, existing deployment solutions, and the prototypes discussed in this article.

- SRM http://www.cs.colorado.edu/serl/cm/SRM.html.
- Software Dock http://www.cs.colorado.edu/serl/cm/dock.html.
- The Configuration Management Yellow Pages http://www.cs.colorado.edu/users/andre/configuration_management.html.
- The Software Deployment Information Clearinghouse http://www.cs.colorado.edu/users/rickhall/deployment/.
- SERL http://www.cs.colorado.edu/serl.

About the Authors

André van der Hoek is a computer science doctoral candidate at the University of Colorado at Boulder. He has a bachelor's degree and a master's degree in business-oriented computer science from

the Erasmus University Rotterdam, the Netherlands. His research interests include configuration management, software architecture, and distributed systems. He is a member of the program committee of the Eighth International Symposium on System Configuration Management.

Richard Hall is a computer science doctoral candidate at the University of Colorado at Boulder. He has a bachelor's degree in computer engineering from the University of Michigan and a master's degree in computer engineering from the University of Colorado at Boulder. His research interests include software deployment and distributed systems. He currently works on a distributed, agent-based framework to support software deployment.

Antonio Carzaniga has a laurea degree in electronic engineering from Politecnico di Milano and a master's degree in information technology from CEFRIEL, Milano, Italy. He was a junior researcher with

CEFRIEL before entering the doctorate program in software engineering at Politecnico di Milano. Currently, he is a visiting research assistant in the computer science department of the University of Colorado at Boulder. His interests include software process as well as generic distributed systems technology. His current research concerns scalable event observation and notification mechanisms.

Dennis Heimbigner has a bachelor's degree in mathematics from the California Institute of Technology and a master's degree and a doctorate in computer science from the University of Southern California. He is a former member of the technical staff for TRW Defense and Space System Group in Los Angeles, Calif. He is currently a research associate and assistant professor in the computer science department of the University of Colorado at Boulder.

Alexander Wolf is a faculty member in the computer science department of the University of Colorado at Boulder. Previously, he was employed at AT&T Bell Laboratories. His research interest is the discovery of principles and development of technologies to support the engineering of large, complex software systems. He has published papers on software engineering environments and tools, software process, software architecture, and configuration management. He is vice chairman of the Association for Computing Machinery Special Interest Group on Software Engineering.

Software Engineering Research Laboratory
Department of Computer Science
University of Colorado
Boulder, CO 80309
Voice: 303-492-5263
Fax: 303-492-2844
E-mail: {andre, rickhall, carzanig, dennis, alw}@cs.colorado.edu

Coming Events

Second Workshop on Software Architectures in Product Line Acquisitions

Dates: June 8-10, 1998
Location: Salem, Mass., Hawthorne Hotel
Subject: Adoption of an architecture-driven approach to acquiring a line of software-intensive products.
Call for Position Papers: Submissions due: March 6, l998. Notification of acceptance: April 1, 1998. For submission guidelines, visit the Production Line Issues Action Team Web site.
Sponsor: Product Line Issues Action Team
Contact: Edward Addy, NASA/WVU Software Research Laboratory
Voice: 304-367-8353
Fax: 304-367-8211
E-mail: eaddy@wvu.edu
Internet: http://columbia.ivv.nasa.gov:6600/pliat

Relationship of DoD Architecture-Driven Standards to Product Line Acquisition Business Model

Dates: March 17-18, 1998
Location: Burlington, Mass.
Sponsor: System Resources Corporation
Subject: Meeting participants will discuss the applicability of Department of Defense (DoD) architecture initiatives to software architecture-based acquisitions of a product line and produce a short point paper discussing how product line acquisition organizations can effectively apply DoD standards in the acquisition of a family of software-intensive systems.
Contact: Harry Joiner
E-mail: hjoiner@world.std.com

International Information Technology Quality Conference

Dates: April 13-17, 1998
Location: Orlando, Fla.
Theme: "Providing Proven Solutions for the New Millenium"
Keynote Speakers: Phillip Crosby, Tom DeMarco, William Perry, Howard Rubin
Sponsor: Quality Assurance Institute
Contact: 407-363-1111
Fax: 407-363-1112
Internet: http://www.qaiusa.com