A Cooperative Approach to Support Software Deployment Using the Software Dock

Richard S. Hall, Dennis Heimbigner, Alexander L. Wolf
Software Engineering Research Laboratory
University of Colorado
Boulder, Colorado 80309-0430 USA
[rickhall,dennis,alw]@cs.colorado.edu

ABSTRACT
Software deployment is an evolving collection of interrelated processes such as release, install, adapt, reconfigure, update, activate, deactivate, remove, and retire. The connectivity of large networks, such as the Internet, is affecting how software deployment is performed. It is necessary to introduce new software deployment technologies that leverage this connectivity. The Software Dock framework creates a distributed, agent-based deployment framework to support the ongoing cooperation and negotiation among software producers themselves and among software producers and software consumers. This deployment framework is enabled by the use of a standardized deployment schema for describing software systems, called the Deployable Software Description (DSD) format. The Software Dock also employs agents to traverse between software producers and consumers in order to perform software deployment activities by interpreting the descriptions of software systems. The Software Dock infrastructure allows software producers to offer their customers high-level deployment services that were previously not possible.

Keywords
Software deployment, Java, mobile agents, configuration management

1 INTRODUCTION

The connectivity of large networks, such as the Internet, is affecting how software deployment is being performed. The simple notion of providing a complete installation procedure for a software system on a CD-ROM is giving way to a more sophisticated notion of ongoing cooperation and negotiation among software producers and consumers. This connectivity and cooperation allows software producers to offer their customers high-level deployment services that were previously not possible. In the past, only software system installation was widely supported, but already support for the update process is becoming common. Support for other software deployment processes, though, such as release, adapt, activate, deactivate, remove, and retire [see Section 2] is still virtually non-existent.

New software deployment technologies are necessary if software producers are expected to accept more responsibility for the long-term operation of their software systems. In order to support software deployment, new deployment technologies must:

- operate on a variety of platforms and network environments, ranging from single sites to the entire Internet,
- provide a semantic model for describing a wide range of software systems in order to facilitate some level of software deployment process automation,
- provide a semantic model of target sites for deployment in order to describe the context in which deployment processes occur, and
- provide decentralized control for both software producers and consumers.

The Software Dock research project addresses many of these concerns. The Software Dock is a system of loosely coupled, cooperating, distributed components. The Software Dock supports software producers by providing the release dock that acts as a repository of software system releases. At the heart of the release dock is a standard semantic schema for describing the deployment requirements of software systems. The field dock component of the Software Dock supports the consumer by providing an interface to the consumer's resources, configuration, and deployed software systems. The Software Dock employs agents that travel from release docks to field docks in order to perform specific software deployment tasks while docked at a field dock. The agents perform their tasks by interpreting the semantic descriptions of both the software systems and the target consumer site. A wide-area event system connects release docks to field docks and enables asynchronous, bi-directional connectivity.

The purpose of this paper is to discuss how the Software Dock project supports software deployment processes. This is accomplished by first introducing the processes that

ICSE '99 Los Angeles CA

comprise software deployment in Section 2. Section 3 provides a high-level introduction to the Software Dock, a framework for software deployment, while Section 4 describes the Deployable Software Description (DSD) format, a critical piece of the Software Dock project used to describe the deployment requirements of software systems. Section 5 discusses specific deployment process support through the use of agents. Section 6 discusses security as it relates to the deployment and the Software Dock specifically, while Section 7 discusses related work. Lastly, current status and future work are discussed in Sections 8 and 9, respectively, followed by the conclusion.

2 SOFTWARE DEPLOYMENT LIFE CYCLE

In the past, software deployment was largely defined as the installation of a software system; a view of software deployment that is simplistic and incomplete. Software deployment is actually a collection of interrelated activities that form the software deployment life cycle. This life cycle, as defined by this research, is an evolving collection of processes that include release, retire, install, activate, deactivate, reconfigure, update, adapt, and remove.

The processes of the software deployment life cycle are performed on either the software producer or consumer side; the processes for each side are described below.

Producer-side Processes

The producer-side of the life cycle consists of two processes, ***release*** and ***retire***. The release process is the bridge between development and deployment. It encompasses all of the activities needed to package, prepare, provide, and advertise a system for deployment to consumer sites. The release package that is created contains the physical artifacts that comprise a given software system and also a description of the deployment requirements for the software system. As modifications or updates are made to the software system, the software producer must repeat the release process to create an updated release package.

When a software producer is no longer able or willing to support a given software system, it must perform the retire process. This process withdraws support for a software system or a given configuration of a software system. The retire process is distinct from the consumer-side remove process; retiring a software system makes it unavailable for future deployment, but it does not necessarily affect consumer sites where the retired software system is currently deployed. Consumers of the software system may continue to use the software without knowing that it has been retired, but the retire process should attempt to notify current users that support for the software system is withdrawn.

Consumer-side Processes

The ***install*** process is the initial deployment activity performed by a consumer. The install process must configure and assemble all of the resources necessary to use a given software system. The install process uses the package created in the release process above. For a specific package, the install process interprets the encoded knowledge and then examines the target consumer site in order to determine how to properly configure the software system for the specific site. Once installation is complete, the deployed software system is ready for use and is ready for other deployment activities.

After a software system is installed, the ***activate*** and ***deactivate*** processes allow the consumer to actually use the software system. The activate process is responsible for making a deployed software system executable or usable. For a simple tool, activation involves establishing some form of command or click-able graphical icon for executing the tool binary. In a client/server system, for example, multiple components may need to execute in parallel. The deactivate process is the inverse of the activate process. It is responsible for shutting down any executing components of an activated software system.

Throughout the lifetime a software system is installed at a consumer site, it is not a static entity with respect to software deployment. Instead, the ***reconfigure***, ***update***, and ***adapt*** processes are responsible for changing and maintaining the deployed software system configuration. These processes may occur in any order and any number of times.

The update process modifies a previously installed software system. Update deploys a new, previously unavailable configuration of a software system. An update becomes necessary when a software producer makes a change to the description of a deployed software system. The changes to the software system's description may denote a new version, a content update, or simply a description update.

The reconfigure process, like install, also modifies a previously installed software system, but its purpose is to select a different configuration of a deployed software system from its existing description.

The purpose of the adapt process is to maintain the consistency of the currently selected configuration of a deployed software system. The adapt process must monitor changes at the consumer site and respond to those changes in order to maintain consistency in the deployed software system. Adaptation becomes necessary when a change is made to the local consumer site that affects the deployed software system. For example, when a required software system file is deleted or corrupted, the adapt process determines the affected file and replaces it.

Once a software system is no longer required at a consumer site, the ***remove*** process is performed. The remove process must undo all of the changes to the consumer site that were caused by previous deployment activities for a given software system. The remove process must pay special attention to shared resources such as data files and libraries in order to prevent dangling references to a required resource. As a result, the remove process must examine the current state of the consumer site, its dependencies, and constraints, and then remove the software system in such a way as to not violate these dependencies and constraints.

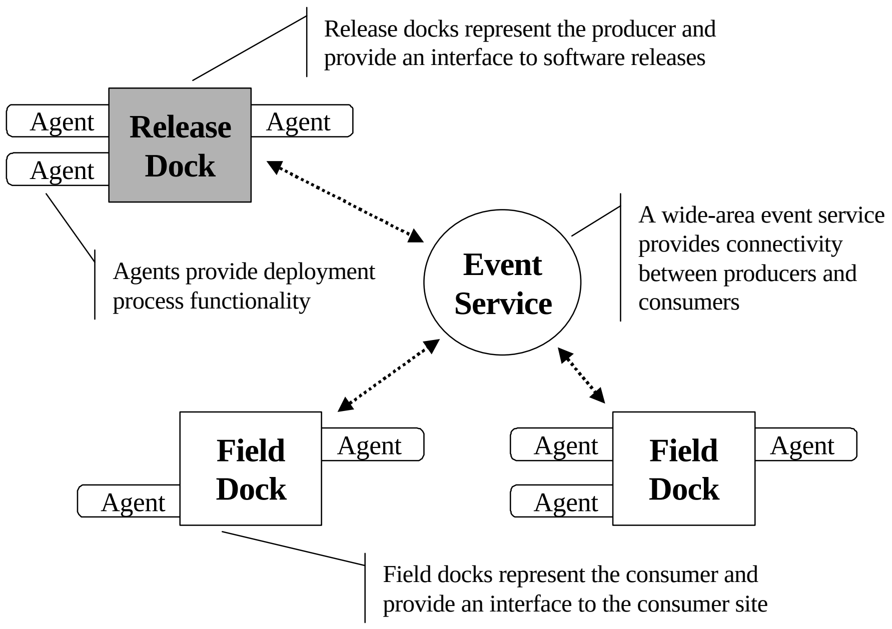

Release docks represent the producer and provide an interface to software releases
Agent
Release Dock
Agent
Agent
Agents provide deployment process functionality
Event Service
A wide-area event service provides connectivity between producers and consumers
Field Dock
Agent
Agent
Agent
Field Dock
Agent
Agent
Field docks represent the consumer and provide an interface to the consumer site

Figure 1: Software Dock Architecture

3 SOFTWARE DOCK ARCHITECTURE

The Software Dock research project, originally described in [9], addresses support for software deployment processes by creating a framework that enables cooperation among software producers themselves and between software producers and software consumers. The Software Dock architecture [see Figure 1] defines components that represent these two main participants in the software deployment problem space. The *release dock* represents the software producer and the *field dock* represents the software consumer. In addition to these components the Software Dock employs *agents* to perform specific deployment process functionality and a *wide-area event system* to provide connectivity between the release docks and the field docks.

In the Software Dock architecture, the release dock is a server residing within a software producing organization. The purpose of the release dock is to serve as a release repository for the software systems that the software producer provides. The release dock provides a Web-based release mechanism that is not wholly unlike the release mechanisms that are currently in use; it provides a browser-accessible means for software consumers to browse and select software for deployment.

The release dock, though, is more sophisticated than most current release mechanisms. Within the release dock, each software release is described using a standard deployment schema; the details of standard schema description for software systems are presented in Section 4. Each software release is accompanied with generic agents that perform software deployment processes by interpreting the description of the software release. The release dock provides a programmatic interface for agents to access its services and content. Finally, the release dock generates events as changes are made to the software releases that it manages. Agents associated with deployed software systems can subscribe for these events to receive notifications about specific release-side occurrences, such as the release of an update.

The field dock is a server residing at a software consumer site. The purpose of the field dock is to serve as an interface to the consumer site. This interface provides information about the state of the consumer site's resources and configuration; this information provides the context into which software systems from a release dock are deployed. Agents that accompany software releases "dock" themselves at the target consumer site's field dock. The interface provided by the field dock is the only interface available to an agent at the underlying consumer site. This interface includes capabilities to query and examine the resources and configuration of the consumer site; examples of each might include installed software systems and the operating system configuration.

The release dock and the field dock are very similar components. Each is a server where agents can "dock" and perform activities. Each manages a standardized, hierarchical registry of information that records the configuration or the contents of its respective sites and creates a common namespace within the framework. The registry model used in each is that of nested collections of attribute-value pairs, where the nested collections form a hierarchy. Any change to a registry generates an event that agents may receive in order to perform subsequent activities. The registry of the release dock mostly provides a list of available software releases, whereas the registry of the field dock performs the valuable role of providing access to consumer-side information.

Consumer-side information is critical in performing nearly any software deployment process. In the past, software deployment was complicated by the fact that consumer-side information was not available in any standardized fashion. The field dock registry addresses this issue by creating a detailed, standardized, hierarchical schema for describing the state of a particular consumer site. By standardizing the information available within a consumer organization, the field dock creates a common software deployment namespace for accessing consumer-side properties, such as operating system and computing platform. This information, when combined with the description of a software system, is used to perform specific software deployment processes.

Agents implement the actual software deployment process functionality. When the installation of a software system is requested on a given consumer site, initially only an agent responsible for installing the specific software system and the description of the specific software system are loaded onto the consumer site from the originating release dock. The installation agent docks at the local field dock and uses the description of the software system and the consumer site state information provided by the field dock to configure the selected software system. When the agent has configured the software system for the specific target consumer site, it requests from its release dock the precise set of artifacts that correspond to the software system configuration.

The installation agent may request other agents from its release dock to come and dock at the local field dock. These other agents are responsible for other deployment activities such as update, adapt, reconfigure, and remove. Each agent performs its associated process by interpreting the information of the software system description and the consumer site configuration.

The wide-area event service [2] in the Software Dock architecture provides a means of connectivity between software producers and consumers for “push”-style capabilities. Agents that are docked at remote field docks can subscribe for events from release docks and can then perform actions in response to those events, such as performing an update. Direct communication between agents and release docks is provided by standard protocols over the Internet. Both forms of connectivity combine to provide the software producer and consumer the opportunity to cooperate in their pursuit of software deployment process support.

4 DEPLOYABLE SOFTWARE DESCRIPTION FORMAT

In order to automate or simplify software deployment processes it is necessary to have some form of deployment knowledge about the software system being deployed. One approach to this requirement is the use of a standardized language or schema for describing a software system; this is the approach adopted by the Software Dock research project. In such a language or schema approach it is common to model software systems as collections of properties, where semantic information is mapped into standardized properties and values. This approach is also used in [4], [6], [10], [20], and [22].

Minimally five classes of semantic information have been identified [7] that must be described by the software system model. These classes of semantic information are:

- **Configuration** – describes relationships inherent in the software system, such as revisions and variants, and describes resources provided by the software system, such as deployment-related interfaces and services.
- **Assertions** – describe constraints on consumer-side properties that must be true otherwise the specific deployment process fails, such as supported hardware platforms or operating systems.
- **Dependencies** – describe constraints on consumer-side properties where a resolution is possible if the constraint is not true, such as installing dependent subsystems or reconfiguring operating system parameters.
- **Artifacts** – describe the actual physical artifacts that comprise the software system.
- **Activities** – describe any specialized activities that are outside of the purview of standard software deployment processes.

The Software Dock project has defined the Deployable Software Description (DSD) format to address these needs. The DSD is a critical piece of the Software Dock research project that enables the creation of generic deployment process definitions.

DSD provides a standard schema for describing a software system family. In this usage, a family is defined as all revisions and variants of a specific software system. The software system family was chosen as the unit of description, rather than a single revision, variant, or some combination, because it provides flexibility when specifying dependencies, enables description reuse, and provides characteristics, such as extending revision lifetime, that are necessary in component-based development.

A DSD family description is broken into multiple elements that address the five semantic classes of information described above. The sections of a DSD family description are identification, imported properties, system properties, property composition, assertions, dependencies, artifacts, interfaces, notifications, services, and activities. Some of these sections map directly onto the five semantic classes of information, others, such as system properties, property composition, interfaces, and notifications, combine to map onto the configuration class of semantic information.

A DSD family description is a simple, hierarchical schema that is built around the notion of properties of the described software system. For example, a typical property of a software system is version number. By defining such a property in a family description it is possible to organize the other pieces of the family description, such as assertions, dependencies, and artifacts, with respect to a given version number. Other examples of software system properties are performance variants and optional capabilities. Once the properties of a software system are defined then the property composition section is used to describe the relationships among properties. For example, one property may exclude another property or it may require secondary property selections. Therefore, composition rules describe valid configurations for the described software system.

The remaining DSD family description sections are guarded by arbitrary boolean property expressions that indicate whether a specific schema element is applicable to a specific configuration. The property expression guards are expressions over software system properties, consumer site properties, or both.

The following examples depict portions of a DSD description that describes a software system that has optional online help documentation. To describe the optional online help documentation, it is necessary to create a software system property to represent the online documentation:

```
Property {
  Name = “Online Help”
  Type = “Boolean”
  Description = “Include online help.”
  … }
```

The above property definition creates a boolean property of the software system that is used for determining whether the online help documentation is applicable to a given configuration of the software system.

Also consider that the described software system only supports the Solaris™ and Window 95™ operating systems. To guarantee that these constraints are true an assertion is created:

```
Assertion {
  Condition = “($OS$ == ‘Solaris’) ||
              ($OS$ == ‘Win95’)”
  Description = “Test for supported
                 operating system.”
  … }
```

This assertion tests the target consumer site's operating system properties by using the standard namespace that is created by the field dock registry. In the above assertion example, the variable `$OS$` is actually shorthand introduced for brevity; the actual variable is the standard field dock registry path expression of :

`$/Local/Software/OperatingSystem/Name$`.

The artifacts that comprise the online help documentation must also be described:

```
Artifacts {
  Guard = “($Online Help$ == true)”
  Artifact {
    Guard = “($OS$ == ‘Solaris’)”
    SourceName = “help.html”
    Source = “/proj/doc”
    DestinationName = “help.html”
    Destination = “doc”
    Mutable = false
    Signature = “a4ca443b8902d3410ec832”
    Type = “DOCUMENTATION”
    … }
  Artifact {
    Guard = “($OS$ == ‘Win95’)”
    SourceName = “help.hlp”
    Source = “/proj/doc”
    DestinationName = “help.hlp”
    Destination = “doc”
    Mutable = false
    Signature = “9283cd2378102f1a3b12ee”
    Type = “DOCUMENTATION”
    … } }
```

The artifacts are described by nesting them in an artifact collection. The above artifact collection is guarded by a property expression that tests the applicability of the artifact collection with respect to a specific configuration; in this case, the artifact collection is only applicable if the “Online Help” property of the software system is true. The actual online help documentation artifacts are described within the artifact collection, each of which is guarded by a property expression that tests for a specific consumer site operating system value. The end result is that the proper artifact is installed with respect to the target consumer site and the selected configuration of the software system.

As a note, software system properties are arbitrary names; they have no meaning within DSD. Therefore, a property such as “version” has no special significance in DSD as it might in other configuration management disciplines. One result of this approach is that properties can be used to organize a software system in a variety of ways. For some examples, properties can be mapped to the traditional configuration management view of versions, the components in the software system architecture, or the features or capabilities of the software system.

5 SOFTWARE DOCK PROCESSES

In the prototype Software Dock framework, agents define the software deployment processes. In general, the other components in the Software Dock architecture are passive elements, such as data and interfaces. Agents, on the other hand, are active since they perform the functionality of the software deployment life cycle processes. The Software Dock framework enables the creation of a collection of generic agents that perform many of the standard software deployment processes, such as install, update, adapt, reconfigure, and remove. These generic agents, although useful in many cases, may not be sufficient for every case and therefore are also useful as base classes for the creation of other, more specialized deployment agents.

All agents perform their deployment processes by encoding some functionality that is then parameterized by the information provided in the DSD specifications and the consumer site descriptions. In this fashion, a single agent definition is used for any software system described using DSD and at any consumer site that has a field dock. The remainder of this section describes the generic deployment process algorithm that all current deployment agents perform and then describes each specific deployment process in more detail.

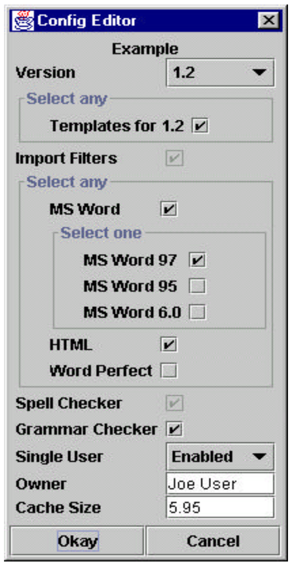
Generic Deployment Process Definition
As described in Section 4, DSD models a software system based on properties and the proper configuration of those properties. A result of this approach led to the discovery of an abstract deployment process algorithm.

Most software deployment processes can be characterized as the transformation of one software system configuration to another based on the set of property values for a given software system configuration. A valid set of software system property values represents a particular valid configuration of a software system. Given a new set of valid property values, a deployment process simply transforms its current configuration to the new configuration by performing differential processing over the applicable schema elements of the DSD specification. The applicable schema elements for a software release are computable via the guard conditions that are dispersed throughout the DSD specification. Differential processing of the applicable schema elements creates a new software system configuration that corresponds to the desired software configuration. For a common example, if the version of a software system is changed from “1.0” to “1.1,” then all of the artifacts associated with version “1.0” are removed, the artifacts associated with version “1.1” are added, and any common artifacts are left untouched.

The install, update, reconfigure, adapt, and remove software deployment processes all follow this general, abstract algorithm.

Specific Deployment Process Definitions
The software deployment processes vary from each other in small, but important ways. Each specific deployment process is described below. There is an interesting, implicit issue with respect to all of the deployment process implementations described below. All of the agents manipulate the DSD specification of a given software release in isolation of the software system itself. This means that an agent needs only the specification of a software release to perform a large portion of its tasks. As a result, an agent is much more efficient, especially in the area of transfer time, since by manipulating the schema description first, the agent only requests exactly what it needs to finish its task. This is possible since the release dock works in cooperation with the agents to perform the deployment processes.

Install Process
The install agent deploys a new configuration of a software release to a consumer site. The install agent differs from the other software deployment process agents since it is not associated with an existing software release configuration. The install agent performs its task by first retrieving the current DSD specification for the software family for which it is responsible. The install agent queries the local field dock and the user to determine the configuration of the software release to install [see Figure 2]. Once a configuration is determined the install agent only needs to perform the actions associated with all of the applicable schema elements for the selected configuration, such as testing assertions, resolving dependencies, and retrieving artifacts. Once the install process is complete, the install agent is no longer needed and therefore it removes itself. Multiple install requests are always handled by separate install agents and therefore always install another configuration of the associated software release. If a software release is unable to have multiple installations at a site, it is necessary to add an assertion to the DSD specification that tests for this condition. Currently the install process is always invoked either directly or indirectly by a specific user request to install a software release; therefore the install process is always “pull” oriented.



Example
Version 1.2
Select any
Templates for 1.2 ✓
Import Filters ✓
Select any
MS Word ✓
Select one
MS Word 97 ✓
MS Word 95 ☐
MS Word 6.0 ☐
HTML ✓
Word Perfect ☐
Spell Checker ✓
Grammar Checker ✓
Single User Enabled
Owner Joe User
Cache Size 5.95

Figure 2: Configuration Editor

Update Process
The update agent deploys a new, previously unavailable configuration of a deployed software release, thus eliminating the previously deployed configuration. The newly available software release configuration is provided in an updated DSD specification for the software release. The update agent must retrieve the new DSD specification from its release dock in order to perform the update. The update agent must account for the existing deployed software release by performing differential processing on the applicable schema elements for the existing and updated software release configurations. Differential processing requires the undoing of schema elements corresponding to the prior configuration and performing the associated actions of the schema elements for the updated configuration. Any schema elements that are shared among configurations are left untouched. A specific update agent always handles the update process for a specific deployed software release. The update process is either specifically directed by the “push” of a new configuration, such as a new version, or it may be undirected in the case of a “pull” update where a new configuration is discovered or specifically selected by the user. An update is not always the result of a change to the currently selected configuration; a content-only update is also possible. In such a scenario, the update does not change the selected configuration of the software system,

only the content of the current configuration. This is typical in many software systems that use a “channel” or content delivery model. Finally, an update may not actually update the deployed software release at all; an update may simply provide a new, more accurate DSD specification for the deployed software release.

Reconfigure Process
The reconfigure agent changes the current configuration of a deployed software release, thus eliminating the previously deployed configuration. The reconfigure agent differs from the update agent because it does not retrieve a new DSD specification from its release dock even if one exists; therefore the reconfiguration agent cannot perform an update. The reconfigure agent only manipulates the existing DSD specification of the deployed software release with which it is associated. The reconfigure agent determines the new software release configuration much like the install agent. Once a new configuration is chosen from the existing DSD specification, the reconfigure agent performs differential processing on the applicable schema elements much like the update agent. A specific reconfigure agent always handles the reconfigure process for a specific deployed software release. Currently the reconfigure agent operates in “pull” mode.

Adapt Process
The adapt agent maintains the consistency of a currently deployed software release configuration in the context of the consumer site. The adapt agent does not change the software release configuration at all, it enforces it. When invoked, the adapt agent uses the existing DSD specification for its associated software release to verify that the deployed software release matches its description. It does this by determining the applicable schema elements for the deployed software release configuration and then testing them to make sure that they are still valid. If any discrepancies are discovered, the adapt agent simply performs the default processing of the invalid schema elements in order to correct the problem. A specific adapt agent always handles the adapt process for a specific deployed software release. Currently the adapt agent operates in “pull” mode. The adapt agent is easily extended to operate in “push” mode where consumer-side events, such as file deletions, automatically instigate the adapt process.

Remove Process
The remove agent is responsible for removing a deployed configuration from a consumer site. The remove agent must ensure that no constraints are violated by the removal of the software system. For example, if other deployed software systems depend on the software system that is being removed, the remove must fail. A specific remove agent always handles the remove process for a specific deployed software release. One remove request may cause multiple remove requests to other remove agents in the case of dependent software releases. Currently the remove agent operates in “pull” mode.

6 SECURITY
Security has an impact on the Software Dock research, but has not been a primary research issue. Despite this fact, this issue has not been summarily excluded in the solution discussed thus far.

Mobile agents cause a large security concern because they come from unknown sources. In order to address some of the security concerns in the Software Dock, agents operate in the Java Virtual Machine (JVM) sandbox. The field dock is the only local interface that an agent has to perform its tasks. To extend the interface provided to agents, the field dock uses a capability approach. The capability approach provided by the field dock allows access to certain restricted operations, such as controlled access to the disk. Currently, the JVM does not support a true capability approach, but this functionality is expected in the 2.0 release of Java. Regardless, all current agents are implemented as though this approach was in effect; thus there is a relatively simple transition when support for the capability-based security approach is released. In addition, this approach can be extended to adopt a mechanism by which agents can become trusted. In such a scenario, trusted agents may be provided with even more sensitive capabilities.

7 RELATED WORK
Software deployment intersects a number of related technologies; this section only covers the most important of these. For more detailed information on related technologies refer to [3] and [8].

The DSD schema created for the Software Dock project is not a unique attempt to create a standard schema for describing software systems. A handful of related technologies are also trying to address the same issue with similar approaches. Traditional configuration management modeling approaches, such as Adele [6] and PCL [22], have influenced DSD, particularly in the area of configuration selection. These traditional approaches, though, are more general configuration modeling languages that do not address software deployment. Nor do these approaches attempt to create a standard schema for any specific task, rather the modeling language is their primary contribution.

A recent, high-profile effort to create a standard software deployment schema is called the Open Software Description (OSD) format [10]. This effort is a collaboration between Microsoft and Marimba to create a schema for describing software systems for “push” technologies. OSD is immature and merely allows for the description of multiple coarse-grain variants of a single revision of a software system; dependent software systems may also be specified. The descriptive information includes some identification information and pointers to archives where the physical artifacts are found. The resulting description is too simplistic to perform any significant software deployment process automation.

The Desktop Management Task Force (DMTF) has created the Management Information Format (MIF) [4]. It is a modeling language for describing various computing system elements. DMTF formed a specific working group to create a standard schema in MIF for describing software systems [4]. An extension to the Software MIF was created by Tivoli and is called the Application Management Specification (AMS) [20]. Since AMS is a superset of MIF, only AMS is discussed here. AMS is more mature than OSD. AMS describes a single revision of a single variant of a software system in great detail. Software system composition, constraints, dependencies, identification, support, and artifacts are some of the elements that AMS describes. AMS is not intended, though, to automate all of the software deployment processes. Instead, AMS describes a semi-static configuration of a software system that is to be installed and monitored at a consumer site; the notion of manipulating internal software system properties like revisions or variants is not directly supported. It is also assumed that there is no cooperation between software producers and software consumers; rather, there is a centralized "administration" authority that is responsible for maintaining the state of deployed software systems.

The Defense Information Infrastructure Common Operating Environment (DII COE) [13] is a Department of Defense effort to restrict the set of components used to build their software systems. The COE supports, among other things, a standard means for packaging components for delivery and installation. These packages are called *segments* [14], where each segment is a separate, installable entity. The DII COE segment describes the constraints, dependencies, and artifacts of a software system. High-level software deployment process support is provided in the form of scripts, though all deployment activities are not directly supported. Like other approaches, the deployed software system configurations are largely static entities that do not change and cannot be manipulated. The support provided is intended for a centralized administration authority and there is no release-side support.

Other approaches, such as GNU Autoconf [16], try to resolve consumer site description by using scripts and heuristics to directly examine the state of a site, but these methods are not always accurate and they are not rich enough to support deployment process automation. The Microsoft Registry [11], is a hierarchical registry of consumer site information for the Windows platform. The schema used in this registry is only partially standardized and even the standardized portions are not sufficient to semantically describe software systems for deployment.

The Redhat Package Manager (RPM) [1] is a tool for the Linux user community that provides many software deployment features. RPM packages contain the software system to be deployed and a semantic description of the software system; this description includes constraints, dependencies, artifacts, and activities in the form of scripts. The granularity of an RPM package is a single revision and a single variant. As a result, only limited forms of configuration selection are supported. RPM does not have a notion of a "release-side" and therefore is only able to request and manipulate complete packages. Also, RPM is intended for single-site deployment and provides no support for multi-site deployment or management.

A host of install utilities exist in the commercial world, such as InstallShield [12]. These systems typically work well for installation, but only address a handful of deployment processes, such as reconfigure and remove, in a limited form. Recent install utilities, such as netDeploy [19] and PC-Install with Internet Extensions [23], are starting to leverage the connectivity of the Internet. Some of these utilities are addressing the update process as well. In general, most of these solutions do not provide reasonable software system description capabilities. The deployment information is not declarative and is not rich enough for software deployment process automation.

Another class of commercial and research utilities exist to support artifact update; these systems include Castanet [17], NSBD [15], and rsync [21]. In most of these systems, there is little if any support for other software deployment processes. These solutions provide only a very simple model for describing software systems, in most cases a software system is merely considered to be a collection of files.

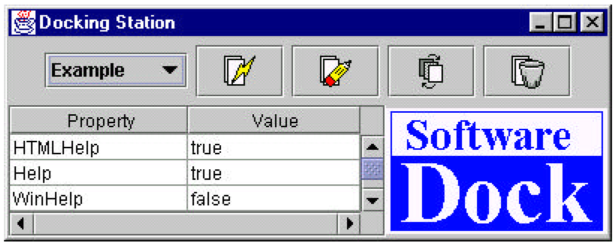
8 CURRENT STATUS

A prototype of the Software Dock deployment framework exists. The Software Dock prototype is implemented entirely in Java and uses Voyager [18] from ObjectSpace as an inter-process communication mechanism and a mobile agent enabling technology. A related research project at the University of Colorado, called SIENA [2], provides a wide-area event.

An evolving definition of the DSD also exists. The current definition of the DSD contains the main elements to support gross software deployment behavior.

The current implementation of the Software Dock infrastructure includes elements for both the release-side and the consumer-side. A release dock implementation exists to house the various software system releases that a software producer has available. The creation of release packages for the release dock is supported by a schema editing tool. This simple schema editor provides a way to create and edit DSD descriptions of software systems and automates some tasks, such as the entry of software artifacts into the DSD description. The schema editor is also used to submit new or updated software release specifications to the local release dock so that they can be made available for deployment. The submission of a release to the release dock automatically generates a set of HTML pages for the new release that consumers can browse and use to initiate installation.

The consumer-side the field dock describes various aspects of the consumer site, such as platform, operating system, memory, and resources. The field dock also provides a



Property	Value
HTMLHelp	true
Help	true
WinHelp	false

Figure 3: Docking Station Support Tool

place for agents to "dock" and perform software deployment related tasks by providing an interface to the underlying consumer site. To further support the consumer-side, a tool, called a docking station [see Figure 3], has been created that provides an interface to the software systems that have been deployed at the consumer site. The docking station provides an interface to the deployment processes that can be performed on the locally deployed software systems. The docking station is used to request updates, reconfigures, adapts, and removes.

A collection of generic agents exists to interpret the DSD software system descriptions in order to perform specific software deployment processes. These generic agents include install, reconfigure, update, adapt, and remove. Each of these agents is fully parameterized by the DSD software description. All agents generically perform the configuration and selection process and then check assertions, resolve subsystem dependencies, and request and retrieve physical artifacts. The end result is support for the release and deployment of configurable content software systems.

The current implementation was used in a demonstration to describe a Web-based software system called the Online Learning Academy (OLLA) created by a division of Lockheed Martin. OLLA consists of 45 megabytes in over 1700 files. OLLA is comprised of two dependent subsystems called Disco and Harvest. The software deployment processes of release, install, reconfigure, update, adapt, and remove have all been initially demonstrated using the generic agents described in this paper along with the DSD description of all three software systems.

Experiments were also conducted to verify the feasibility of the Software Dock. These experiments compared the Software Dock prototype to an existing deployment solution for a specific software system. A DSD specification for versions 1.1.6 and 1.1.7 of the Java Development Kit (JDK) by Sun Microsystems was created in order to compare the Software Dock deployment processes to the standard InstallShield self-extracting distribution archive for the Microsoft Windows platform. Time to completion was the dimension for comparison; Table 1 summarizes the results of the experiments.

In these experiments the Software Dock prototype performed better in most cases, even though it is dynamically creating release packages for each operation. In two of the experiments, reconfigure (remove) and update, the InstallShield process is not actually performing the equivalent actions and therefore direct comparison is difficult.

9 FUTURE WORK

The current implementation of the Software Dock concentrates on the one-to-one aspects of the software producer/consumer relationship. There is no inherent limitation in the Software Dock framework for supporting other aspects of the software producer/consumer relationship. The most obvious scenario is that of the administrator role at a consumer site.

In order to support an administrator role, a new collection of "remote" agents will be created. These remote agents will behave much like the current agents, except that they will also be parameterized by consumer site names. With such a capability, an administrator is able to specify that an activity, such as install or update, should occur on a specific site or a specific set of sites.

To further support the administrator role, a new server, called the interdock, will be introduced. An interdock server contains a global view of the consumer organization, such as site domains and global services. With the interdock, administration tasks are simplified and more complicated deployment scenarios are addressable, such as those of distributed, coordinated software systems.

In addition, the DSD will continue to be extended and expanded. Support for administration policies will be enhanced. Arbitrary dependency specification, rather than just subsystem dependencies, will also be researched. Lastly, better support for specialized deployment activities will be further investigated.

10 CONCLUSIONS

Software deployment is not a single process, such as install, but rather it is a collection of interrelated processes that are performed after a software system has been developed and made available to consumers. Support for software deployment by software producers was neglected until recently. Large network environments, such as the Internet, offer connectivity that enables software producers to offer high-level software deployment services to their customers, services that were previously not possible. By combining the connectivity of large networks with the deployment technologies described in this paper, the Software Dock creates a cooperative framework that supports software deployment.

The Software Dock supports software deployment processes by introducing components that represent software

	Software Dock	InstallShield
Install	172.0s	168.0s
Remove	36.7s	80.0s
Reconfig (remove)	40.3s	90.0s
Reconfig (add)	113.3s	284.3s
Update	187.3s	149.6s

Table 1: Software Dock Comparison Experiments

producers and consumers, release docks and field docks, respectively. The definition and use of a standard schema for describing software systems is central to the Software Dock framework, and it provides, in a declarative form, all of the knowledge necessary to perform software deployment processes. Finally, agents are employed to embody the actual functionality of the deployment processes. The agents realize the deployment process functionality in a generic fashion by interpreting the declarative schema description of the software system.

ACKNOWLEDGMENTS
This material is based upon work sponsored by the Air Force Materiel Command, Rome Laboratory, and the Defense Advanced Research Projects Agency under Contract Numbers F30602-94-C-0253 and F30602-98-2-0163. The content of the information does not necessarily reflect the position or the policy of the Government and no official endorsement should be inferred.

REFERENCES

1. E. C. Bailey. "Maximum RPM," Red Hat Software, Inc., ISBN: 1-888172-78-9, Feb. 1997.
2. A. Carzaniga, D. S. Rosenblum, and A. L. Wolf. "Design of a Scalable Event Notification Service: Interface and Architecture," Technical Report, Dept. of Computer Science, University of Colorado, 1998.
3. A. Carzaniga, A. Fuggetta, R. S. Hall, A. van der Hoek, D. Heimbigner, A. L. Wolf. "A Characterization Framework for Software Deployment Technologies," Technical Report CU-CS-857-98, Dept. of Computer Science, University of Colorado, April 1998.
4. Desktop Management Task Force, "Enabling your product for manageability with MIF files," Nov. 1994.
5. Desktop Management Task Force, "Software Standard Groups Definition, Version 2.0," Mar. 27, 1996. http://www.dmtf.org/tech/apps.html.
6. J. Estublier and R. Casallas. "The Adele Configuration Manager," Configuration Management, Wiley, 1994, pp. 99-134.
7. R. S. Hall, D. Heimbigner, and A. L. Wolf. "Requirements for Software Deployment Languages and Schema," Proc. of the 1998 Int'l Workshop on Software Configuration Management, July 1998.
8. R. S. Hall, D. Heimbigner, and A. L. Wolf. "Evaluating Software Deployment Languages and Schema," Proc. of the 1998 Int'l Conf. on Software Maintenance, IEEE Computing Society, Nov. 1998.
9. R. S. Hall, D. Heimbigner, A. van der Hoek, A. L. Wolf. "An architecture for Post-Development Configuration Management in a Wide-Area Network," Proc. of the 1997 Int'l Conf. on Distributed Configurable Systems, IEEE Computing Society, May 1997, pp. 269-278.
10. A. van Hoff, H. Partovi, T. Thai. "The Open Software Description Format (OSD)," Microsoft Corp. and Marimba, Inc., 1997. http://www.w3.org/TR/NOTE-OSD.html.
11. J. Honeycutt. "Using the Windows 95 Registry," Que Publishing, Indianapolis, IN, 1996.
12. InstallShield Corp. InstallShield, 1998. http://www.installshield.com.
13. Joint Interoperability and Engineering Organization. "Defense Information Infrastructure Common Operating Environment Baseline Specificiations," Version 3.0, Defense Information Systems Agency, CM-400-25-05, Oct. 31 1996. http://spider.osfl.disa.mil/cm/baseline/base_line3/baselin3.pdf
14. Joint Interoperability and Engineering Organization. "How to Segment Guide," Version 4.0, Defense Information Systems Agency, Dec. 30 1996. http://spider.osfl.disa.mil/cm/how_to/howtoseg.pdf.
15. Lucent Technologies. Not So Bad Distribution (NSBD), 1998. http://www.bell-labs.com/project/nsbd/.
16. D. Mackenzie, R. McGrath, and N. Friedman. "Autoconf: Generating Automatic Configuration Scripts," Free Software Foundation, Inc, April 1994.
17. Marimba, Inc. "Castanet Product Family," 1998. http://www.marimba.com/datasheets/castanet-3_0-ds.html.
18. ObjectSpace, Inc. Voyager, 1998. http://www.objectspace.com.
19. Open Software Associates. OpenWEB netDeploy, 1998. http://www.osa.com.
20. Tivoli Systems. "Applications Management Specification," Version 2.0, Nov. 5 1997. http://www.tivoli.com/o_products/html/body_ams_spec.html.
21. A. Tridgell and P. Mackerras. "The rsync algorithm," Technical Report TR-CS-96-05, June 1996. http://cs.anu.edu.au/techreports/1996/index.html.
22. E. Tryggeseth, B. Gulla, R. Conradi. "Modeling Systems with Variability using the PROTEUS Configuration Language," Proceedings of the 1995 International Symposium on System Configuration Management, Springer, 1995, pp. 216-240.
23. Twenty Twenty Software. PC-Install with Internet Extensions, 1998. http://www.twenty.com.