The Software Dock: A Distributed, Agent-based Software Deployment System

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

Software Engineering Research Laboratory
Department of Computer Science
University of Colorado
Boulder, CO 80309 USA

{rickhall,dennis,andre,alw}@cs.colorado.edu

University of Colorado
Department of Computer Science
Technical Report CU-CS-832-97 February 1997

A version of this report to appear in
The Proceedings of the 1997 International Conference
on Distributed Computing Systems, May 1997

This work was supported in part by the Naval Research and Development Division (NRaD) and the Defense Advanced Research Projects Agency under Contract Number N66001-95-D-8656. This work was also supported in part by the Air Force Material Command, Rome Laboratory, and the Defense Advanced Research Projects Agency under Contract Number F30602-94-C-0253. 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.

ABSTRACT

Few tools exist to address the post-development activities of configuring, releasing, installing, updating, reconfiguring, and even de-installing a software system. Certainly there is no unified approach for all of these activities, and none that can take full advantage of a wide-area network. The Software Dock represents an architecture for supporting post-development activities in such a setting. It is designed as a system of loosely-coupled, cooperating, distributed components that are bound together by a wide-area messaging and event system. The components include field docks *for maintaining site-specific configuration information by consumers,* release docks *for managing the configuration and release of software systems by producers, and a variety of* agents *for automating the activities. Its mechanisms of consistent access to a site's configuration information and resources, standardized methods for making software releases available and visible, and a global event system give software producers and consumers new leverage in managing complex software systems. In this paper we describe the Software Dock architecture and discuss the use of a prototype implementation of that architecture in deploying a complex system.*

1 Introduction

Software deployment is the complex process that covers all of the activities performed after a software system has been developed. These activities include configuring, releasing, installing, updating, reconfiguring, and even de-installing a software system. For modern software systems, particularly those built by assembling independently developed components, the deployment process involves the careful coordination and interaction of multiple producers (including developers) and multiple consumers (including users) that may be geographically and organizationally dispersed.

Because traditional configuration management systems have focused on the development activity of source code control, few mechanisms are currently in place to support the software deployment activities. Consider, for example, the deployment activities of configuration and installation. It is typical that information about the sites at which a system is to be deployed is not readily available, complete, nor accurate. Even if the information were available, non-standard methods to access the information make it difficult to automatically and remotely configure the system for deployment. Tools such as Autoconf [12] and Ship [11] attempt to obtain the information on a per installation, ad hoc basis by using scripts and heuristics. The Microsoft Registry [10] stores some amount of configuration information at a site, but that information is only partially standardized. Compounding the configuration and installation problem is the fact that many development organizations do not make system dependencies an explicit part of the system's definition. This leads to installations that cannot operate, simply because all the necessary components have not been put in place.

As more systems are built using distributed component technology, such as CORBA [14] or Java [9], the "missing component" problem will become more common. The personal computing notion that software systems are self-contained, such that copies of all components needed for an installation are included on a CD-ROM, is overly simplistic. For example, the plug-ins and helper applications used with Web browsers are not themselves components of the browsers, but are independently developed and installed. Even if one could construct a monolithic installation, this approach still fails when components are shared with other systems, since the versions of those shared components may not be compatible.

Support for deployment activities other than configuration and installation is essentially non-existent. The installed system typically becomes a static entity detached from its producers and poorly understood by its consumers. For example, it is difficult for a consumer to determine how a change made to their site will affect the installed software at that site. This difficulty results from the fact that constraints and dependencies are not explicit nor is the software able to automatically adapt. Instead, as the environment changes, it becomes the consumer's burden to ensure that the system continues to function properly. As enhancements and bug fixes are released, there is no standard way for the consumer to automatically upgrade, or even locate, the installed system.

Clearly, there is a need to develop a powerful new generation of configuration management technologies that account for post-development activities. To be effective, these new technologies must

- operate in a variety of environments, ranging from a single machine to a distributed, decentralized, wide-area setting that leverages the connectivity offered by networks such as the

Internet;

- provide a way to describe site and software system configuration information;

- provide a way to manage, access, and reason about site configuration information, which includes the hardware and software environment at a site;

- provide a way to manage, access, and reason about a software system configuration, which includes dependencies and constraints inherent in the system, both for initial installation and as a way to maintain the working state of a system;

- make it possible to monitor the environment surrounding a deployed system, watch for changes in that environment, and automatically adapt the software to those changes; and

- allow consumers, in loose cooperation with software producers, to maintain their installed systems with little or no need for human intervention.

In general, then, the new configuration management technologies should automate—as an integral part of software systems themselves—the activities required for continuous support of deployed systems.

As a contribution to the new generation of configuration management technologies, we are developing the Software Dock, an architecture for supporting the software deployment process. By analogy to the hardware docking stations used with portable notebook computers, the Software Dock provides a context in which to situate a software system at a site. As with its hardware counterpart, "docking" a software system involves protocols for interrogating the local environment for its properties and adapting the software to that environment. But a significant difference from hardware docking is the fact that both the environment and the software system are malleable. For example, if a required component is not found at a site, then that component can be added dynamically to the site to satisfy the needs of the software being docked. Alternatively, a more appropriate version of the software system itself can be obtained dynamically and installed. This allows installation to become a process of negotiation between a producer and a consumer. Moreover, the mutual adaptation process can continue beyond the initial docking to provide a perpetually evolving combination of system and environment. What makes this all possible is the mobility of software (as opposed to hardware) and the connectivity of networks such as the Internet.

The Software Dock is a system of loosely-coupled, cooperating, distributed components that are bound together by a wide-area messaging and event system. The components include *field docks* for maintaining site-specific configuration information by consumers, *release docks* for managing the configuration and release of software systems by producers, and a variety of *agents* for automating the deployment process. Both the information about releases and the information about field sites are represented as hierarchies of data that, when combined, form a federated software deployment registry with a conceptually global name space. Events generated by operations on the hierarchies propagate throughout the federated registry and are received by interested agents. The agent technology enables concomitant actions to be automatically performed in response those events.

This paper describes the Software Dock architecture, an initial prototype of its components, and its application to a deployment task. To help describe the details of the technology, Section 2 provides an example deployment scenario. Section 3 introduces the overall architecture of the system and describes each of the major components, while Section 4 relates the components to the scenario. Section 5 discusses other work in the area of post-development configuration management. Finally, the status of a prototype implementation of the Software Dock architecture and directions for future work are presented in Section 6.

2 A Deployment Scenario

Lockheed Martin's Online Learning Academy (OLLA) provides a real-world example of the need for improved support for deployment activities. This example is used throughout the next two sections to illustrate the capabilities of the Software Dock, and has been used as the basis for a demonstration of our prototype.

> *OLLA is a collaborative learning environment developed by Lockheed Martin (Paoli, Pennsylvania) as part of DARPA's CAETI program. OLLA is comprised of a collection of media content (Web pages, audio, video, etc.) and a collection of scripts and programs that together occupy 45 megabytes of disk space across 1700 files. The OLLA system depends on a separate system called Disco that, in turn, depends on a Web page indexing and search system called Harvest [4]. Disco is an extension to Harvest developed by Lockheed Martin that adds a forms-based interface to the Harvest query system. Harvest, on the other hand, is an independent system developed outside of Lockheed Martin. OLLA is installed at various oversea schools and, occasionally, Lockheed Martin releases updates to the system that also need to be propagated to the schools.*

The OLLA scenario uncovers many issues that must be addressed by the various deployment activities. Initially, configuration information about the field site must be obtained to properly configure the OLLA installation. Specifically, information such as the hardware and software platforms, HTTP servers, and amount of disk space, is used to direct the installation activity. The installation activity must also ensure that Harvest and Disco are installed on the field site before OLLA is installed and then properly integrate these systems with the new OLLA installation.

Once installed, certain start-up tasks must be performed, such as using Harvest to build indexes for the OLLA content. During the remaining lifecycle of the system, updates released by Lockheed Martin will need to be installed. In addition, local changes to the field site that may affect the OLLA installation, such as reconfiguring the HTTP server, will require that OLLA's configuration be modified as well. Compounding the difficulties of this scenario is the fact that the schools have limited resources to support system administration. Therefore, the Software Dock is proposed as a next-generation configuration management system to gather all of these activities into one unified post-development framework.

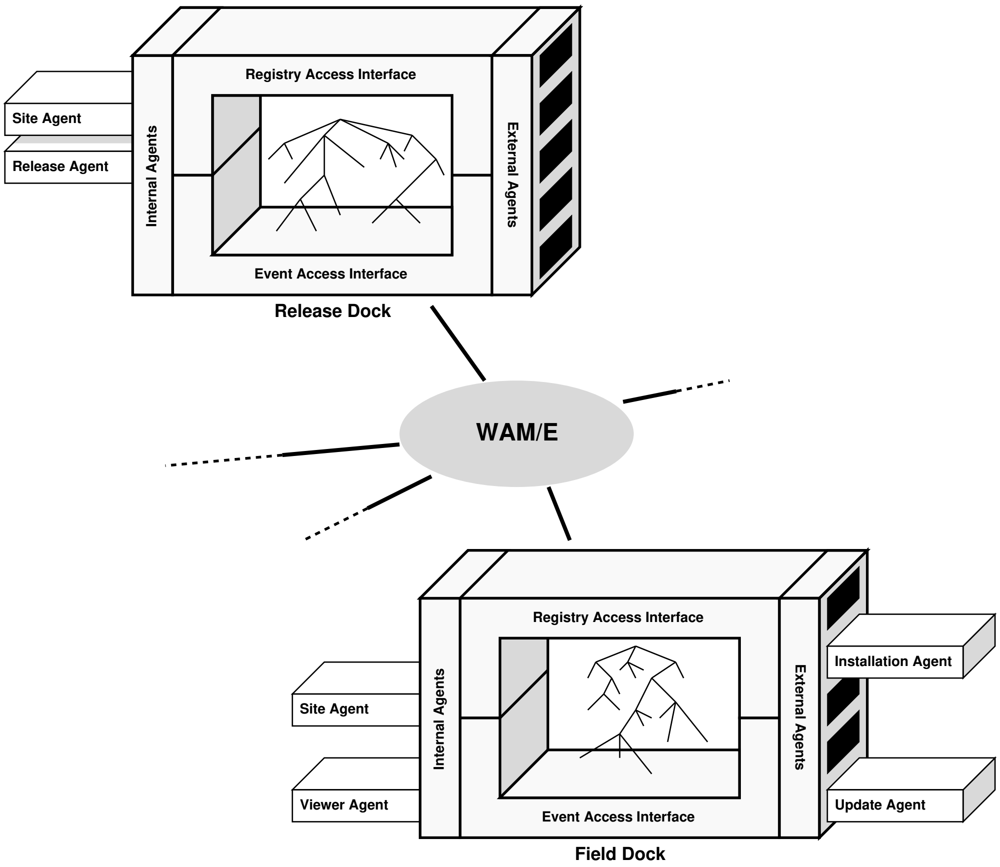

Figure 1: The Software Dock Architecture.

3 The Software Dock Architecture

This section describes the architecture of the Software Dock system. First the architecture is presented in overview and then the components are described in detail. Section 4 uses the scenario introduced in the previous section to illustrate the architectural concepts in more detail.

3.1 Overview

The Software Dock architecture consists of five primary components, as illustrated in Figure 1.

- A *field dock* is a server residing at a consumer site that maintains a registry of local configuration information. It provides interfaces for agents to access the information and to subscribe to events that are generated when information in the registry changes.

- A *release dock* is a server residing at a producer site that provides a registry of information about software releases. It provides interfaces for agents to access the information and to subscribe to events.

- A *federated deployment registry* is a conceptual aggregation of the field dock and release dock registries that provides a global name space.

- An *agent* is an executable program that performs specific tasks of the deployment process at release and field sites on behalf of producers and consumers.

- A *wide-area messaging/event system (WAM/E)* is an Internet-scale notification mechanism [16] that enables the flow of messages and events between agents located at release and field docks.

As shown in Figure 1, there can be many field and release docks representing the interests of the many possible participants in the deployment process. Tying them together is WAM/E, which provides bi-directional communication pathways. Agent technology is useful in this situation because it provides a means of dynamically distributing functionality and enabling client-side processing of events. Given the emergence of languages such as Java [9], agents can be developed independently of hardware and operating system platforms.

3.2 Federated Deployment Registry

The federated deployment registry is central to the Software Dock architecture and is formed by conjoining the registries at all release and field sites. Figure 2 depicts a conceptualization of the registry as a single, global structure and shows the portion relevant to the OLLA scenario. This structure is conceptual in the sense that it provides only a logically global name space, not a physical global implementation. The global name space enables a standard method to query the state of consumer and release sites, to subscribe for events, and to discover properties of the Software Dock environment in general.

The registry is organized as an n-ary tree. The tree model was chosen mainly for its simplicity, but also because it subsumes relevant existing models, such as the DMTF MIF format [6], the Microsoft Registry [10], the X resources model [17], and most file systems.

The schema of the registry is kept consistent across sites to ease the development of agents that access the information. Each tree node contains a name, a description, access controls, and an optional type. Additionally, each node contains a list of attribute-value pairs. The type of a node is exploited to specify the structure of the subtree under that node. For example, a tree node of type "Application" adheres to the subtree definition of the application nodes in Figure 2. The access controls on nodes are used to ensure that required subtrees and attribute-value pairs are not deleted, only extended. This allows sites to augment the information contained in the registry without disrupting the behavior of agents.

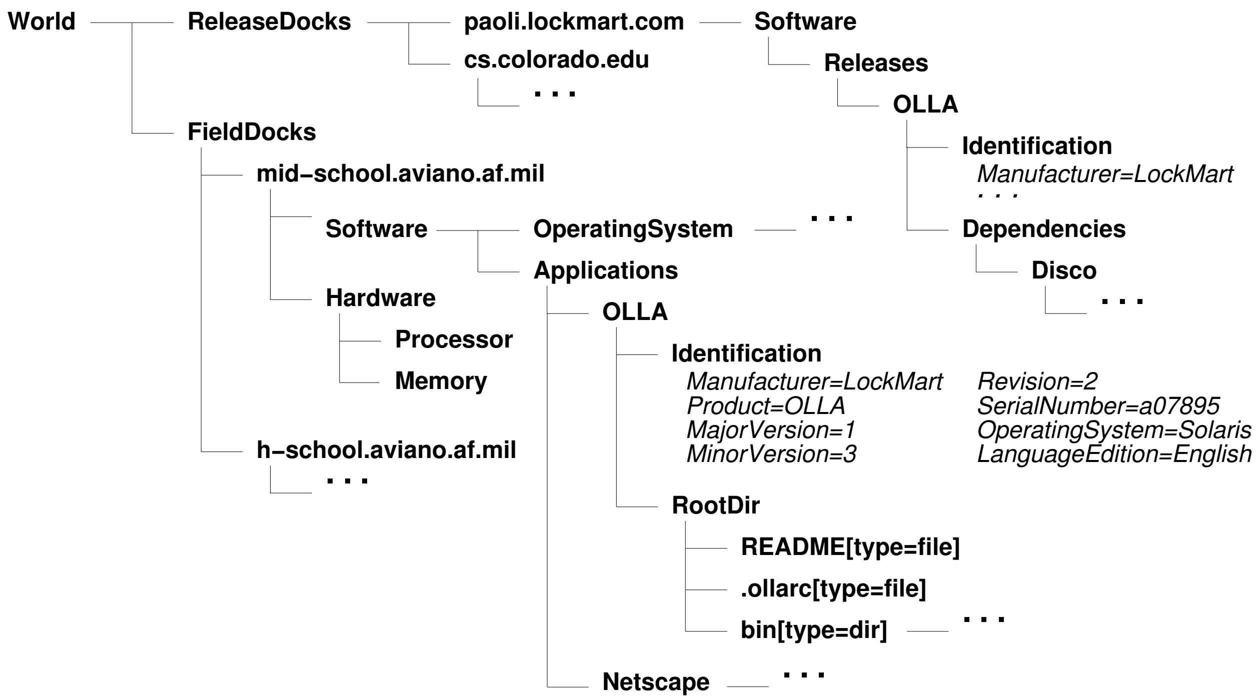

Figure 2: The Federated Deployment Registry for the OLLA Scenario. (Note: Information presented here is for illustrative purposes.)

3.3 Field Dock

A field dock typically exists one per consumer site and performs several roles on behalf of the consumer.

- It provides a registry of information about the hardware and software environment present at (or absent from) the site on which it resides through the registry interface.
- It propagates events subscribed to by agents at the local site. These events are generated in response to operations applied to the registry and events received from remote sources and are made available through the event interface.
- It provides controlled access to the site and its resources by allowing agents to add semantics to events generated in the registry.

Each of these aspects of a field dock is described below.

3.3.1 Field Dock Registry

The registry maintained by a field dock forms the basic description of the consumer site. The information contained in the registry can be thought of as a snapshot of the state of the field site. The field dock itself does not place any semantics on the content or the schema of the registry. All interpretation of semantics is performed by external entities, namely agents. When the site's state changes, an agent uses the field dock's registry interface to reflect those changes in the registry. When an agent initiates an operation on the registry, the operation generates an event that is propagated to other, interested agents.

As an example of how the registry is used, consider an agent responsible for installing OLLA. When it is downloaded to a field site, the agent queries the field dock at that site to obtain configuration information. One piece of information is a description of the operating system at a specific site such as `mid-school.aviano.af.mil`. The agent obtains this information by querying the operating system node

`/World/FieldDocks/mid-school.aviano.af.mil/Software/OperatingSystem`

which is a global path name within the federated deployment registry.[1] At that point, the agent can examine the attributes of the operating system node to determine specific operating system information.

3.3.2 Event Model

The field dock provides an interface for agents to register interest in (i.e., subscribe to) events. All events occur as a result of operations being performed on the registry. An event is defined as having a type, a name, and an attribute list. The event type is associated with the registry operation that has been performed to generate the event, such as registry tree nodes or attributes being added, deleted, or changed. The event name is derived from a path-name-like construction based on the event-generating node's path name in the registry. Finally, the attribute list by default depends on the type of the event, but can be augmented by user supplied attributes. For example, when a tree node attribute is changed, the event attributes include, by definition, the old and new values of the respective tree node attribute, but could also include any additional attributes that were specified by the instigator of the registry operation.

Referring to the OLLA scenario, when the agent responsible for installing OLLA at `mid-school.aviano.af.mil` creates an application node for the OLLA system in the registry, an event is generated. This event has the type "node added" with the name `.../Software/Applications/OLLA`. The attribute list for a "node added" event is empty by default. This event indicates to the agents that have subscribed to such events that an application called `OLLA` has been added to the site.

[1] To reduce the length of path names in this paper, assume (unless otherwise noted) that all paths are prefixed with `/World/FieldDocks/mid-school.aviano.af.mil/`, which will be denoted by a "`.../`" prefix in the text.

The process of local event delivery and propagation is the responsibility of the local field dock. Additionally, the field dock injects local events into the global event propagation mechanism, WAM/E, through a gateway agent. Locally, when an event is generated, the field dock sends the event to any local agent that has subscribed to that event. In order to subscribe to an event, an agent uses the field dock's event interface to tell the field dock the type and the name of the event in which it is interested. For example, if an agent at `mid-school.aviano.af.mil` is interested in the event of OLLA being added to that site, then it would register for an event of type "node added" with the name `.../Software/Applications/OLLA`. When such an event is generated, the interested agent receives the event notification along with any additional attributes that may have been attached.

Realistically, this interface for subscribing to events is a bit clumsy, especially if an agent is interested in a class of events, rather than a specific event. Ideally, one would like to be able to register for any event of a particular class. To enable this, the field dock event interface borrows the wildcard notation from the X resources model [17]. Two types of wildcards are provided: a single-level wildcard ("`.`") and a multiple-level wildcard ("`*`"). By using a "`.`" in the event name specification, any one value at the specified level is matched. Using "`*`" indicates matching any number of values at multiple levels. Therefore, if an agent is interested in any application being added to the site it would use the event name specification `.../Software/Applications/.` to register such an interest.

Subscribing to an event at the attribute level requires the introduction of notation by which node attributes can be used in the event name specification. When an event is generated that involves an attribute of a tree node, the attribute name, prepended with "`#`", is appended to the event name. For example, the content attribute of a file type node would be specified by `#Content`. Thus, if an agent was interested in the change of the content attribute of any file node under any application directory tree node, then it would subscribe to `.../Software/Applications/./RootDir/*/#Content`.

3.3.3 Controlled Site Access

The final function performed by the field dock is controlled access to the underlying site. All operations that can be performed on a site are directly exported in the field dock's interface or they are indirectly exported through specific agents performing a defined task in response to the occurrence of a specific event or event pattern. Examples of the former include the configuration interfaces and the event interfaces of the field dock described above. An example of the latter is an agent that resides at a site and adds an icon to the desktop whenever an application is added to the site. This agent provides an indirect interface to the user's desktop. The indirect interfaces created by the field dock's registry and event system can be quite sophisticated. The site agent, for example, creates an indirect interface to the site's file system by registering for specific events that semantically denote file operations, and then mapping these events into the file system itself. This particular example is discussed further in Section 3.5.

3.4 Release Dock

A release dock works in support of producers and resides on a designated site within a software producing organization. The architecture of the release dock shares much of its architecture and functionality with the field dock. The release dock maintains a registry of the producer's software releases that fits into the larger federated deployment registry, and provides configuration and event interfaces.

A software release in the release dock's registry is a collection of artifacts, such as executables, libraries, documentation, dependencies, and constraints. It also includes the agents responsible for all of the deployment activities, including configuring, installing, maintaining, and de-installing the software. The release dock provides an interface, in the form of an agent, to allow the insertion of new releases into its registry.

Like the field dock, the release dock generates events when operations are performed on its registry. These events are used to indicate changes in the state of releases, such as the release of a new software system, a new version of an existing system, or a patch to an existing system.

Organizations use a release dock much like current FTP sites are used for distributing software, though the release dock is more sophisticated. The release dock may provide a user interface, perhaps through a Web page, to allow consumers to browse the available releases. The release dock, however, does not distribute software releases directly. Instead, when the consumer initiates a download, the release dock sends an agent to the consumer's site. This installation agent is responsible for installing the requested software release by interacting with the consumer site's field dock to obtain the appropriate configuration information. Once the configuration information is obtained, the agent retrieves the properly configured components from its release dock and installs them at the field site.

In the example scenario, Lockheed Martin has a release dock installed at a designated site within its organization. A set of Web pages is available for consumers to browse the releases that Lockheed Martin currently has available; one particular system is OLLA. The Web pages contain detailed information about OLLA and a link to download an OLLA installation agent. The nodes in the release dock's registry represent all the information necessary to install the various configurations and versions of OLLA, including descriptions of dependencies and constraints on components available elsewhere.

3.5 Agents

Since semantic knowledge is not part of field or release docks, most of their functionality 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. Through these techniques, agents provide a large portion of the functionality in the Software Dock environment.

From the perspective of a particular site, there are two classes of agents: *internal* and *external*. Internal agents extend the functionality of a local dock and, to some extent, are trusted at that site. External agents are obtained from remote sites to perform some particular function on behalf of a remote organization. Therefore, external agents are not trusted to the same extent as internal agents.

An external agent that comes from a release dock is generically referred to as a *deployment agent*. The most common deployment agent is an agent responsible for installing a software system. This *installation agent* is typically downloaded from a release dock directly by a consumer or indirectly by another agent. The downloaded installation agent then proceeds to install the software on behalf of the consumer, using the mechanisms provided by the Software Dock. When an installation agent installs a software system, it may additionally install other deployment agents at the site to perform tasks such as updating the software when updates become available.

In the example scenario, the OLLA release includes a deployment agent to install OLLA, as well as one that resides at the field site to install updates as they become available. Disco and Harvest also have associated deployment agents that may be causally invoked due to the dependence of OLLA on Disco and Disco on Harvest.

In contrast to external agents, which mainly perform deployment activities, internal agents provide three major capabilities: viewing, abstraction, and isolation. A viewing agent provides a user interface for accessing, browsing, or manipulating a site's registry. A specific possibility could include an agent that provides a graphical interface for accessing applications installed at the site, or one that adds an entry to the Windows 95 start menu whenever an application is added to the site. To perform these tasks, an agent needs only to register with the field dock for the specific application events.

Other internal agents define abstract interfaces for operations whose implementation requires site-specific knowledge. They accomplish this by subscribing to selected events and, in response, by performing site-specific actions. In effect, they provide an extended, controlled interface to the underlying site for use by other agents, typically deployment agents. A good example of an internal agent is the field dock's site agent that creates an indirect interface to the local site's file system. The site agent registers with the field dock for events that occur under an application node's "RootDir" node (see Figure 2). The "RootDir" node is a collection of tree nodes that represent the directories and files that comprise an application. Events that occur under this tree node are semantically equivalent to the creating and updating of subdirectories and files in the site's file system. To illustrate, recall the event naming specification techniques of Section 3.3.2. The site agent registers for these events:

1. `.../Software/Applications/. (type == NODE_ADDED)`
2. `.../Software/Applications/./RootDir/* (type == NODE_ADDED)`
3. `.../Software/Applications/./RootDir/*/#Content (type == UPDATE_ATTR)`

These event specifications match events that, respectively, indicate when applications are being added to the site, when files or directories are being added to an application node, and when application files have been updated. When the site agent receives events indicating activity in the application's "RootDir" subtree, it mirrors those events into the local file system to create various application specific directories and files in the place appropriate to the field site.

In effect, the site agent allows a deployment agent to install files into the local site's file system without giving the deployment agent direct access to the local disk. Thus, internal agents support the isolation of potentially untrusted deployment agents from the local site's resources. This adds an important degree of security and access control to the whole deployment process.

Returning to the OLLA scenario, an agent responsible for installing OLLA installs the necessary OLLA system files into the deployment site's file system by first creating an application tree node in the site's registry. The installation agent then inserts files and directories into the application node's directory subtree. These actions are physically reflected in the site's file system by the site agent.

Field and release dock implementations are accompanied by a variety of predefined agents for performing specific tasks. Generic agent classes are also provided for performing simple software system installations and updates by interpreting the standard schema in both the field and release dock registries. It is also possible to create agents from scratch using the interfaces provided by the field and release dock servers.

3.6 Wide-Area Messaging/Event System

The main facilitator of interaction among components in the Software Dock architecture is event notification. As previously described, an event has a type, a name, and a list of attributes. Until this point in the paper, the discussion of events has largely been restricted to propagation within a site, only alluding to non-local propagation among sites.

WAM/E is the component responsible for propagating events across a wide-area network to sites interested in global events. Its general operation is similar to that of the local event propagation mechanism. A difference is that the subscriber to locally propagated events is an agent, while docks themselves are the subscribers to globally propagated events, which in turn propagate the events to local agents. Thus, WAM/E provides an interface by which a dock can register interest in events. It also provides an interface by which a dock can inject an event consisting of a type, a name, and an attribute list.

The architecture for our prototype consists of a single multicast message server at one site plus gateway agents at each site. The message server is responsible for keeping track of registrations from various sites. Event generation is carried out by sending the event to the multicast server. The server then matches the event against the registrations and forwards the events to those matching sites.

Registration is actually somewhat non-traditional. Some docks will register interest so that they may receive events. But docks that intend to generate events must also register with the multicast server. This is done for reasons of efficiency; there is a cost for globally multicasting an event and so generators of events need to be freed of that burden until it is known that some other dock has expressed an interest.

The gateway agents are specialized internal agents that listen to local events at a site and selectively echo events, possibly transformed, to the multicast server. Thus, there is an intentional similarity of form between local events and wide-area events.

The WAM/E mechanism described here is provisional and is the subject of further research [16]. Below, we characterize the requirements that WAM/E must eventually address.

3.6.1 Efficient Event Propagation

The single message server will be overwhelmed when the number of events and the number of sites is large. At that point, it must be replaced by a collection of cooperating servers to distribute the event propagation load.

One approach is to exploit possible locality properties. That is, it may be that the event senders and the event receivers are clustered together by frequency of event traffic or by geographic locality. In this case, servers can be connected into a hierarchy in which most events are handled within a subtree. All non-local traffic is sent on a single connection to the parent of the subtree for eventual propagation to a remote subtree. The Harvest system [4] provides a good model for this kind of approach.

Another approach might provide separate message servers for various kinds of events. Thus, all events of one type or name would go to one message server and all events of another type or name might go to a different server.

3.6.2 Event Artifact Propagation

Event artifact propagation is another consideration for the WAM/E architecture. An update notification event, for example, might contain a pointer to an updated file at the original release dock. This would require all interested recipients of the update event to contact the release dock and ask for the new file. This approach could cause undo burden on the release dock and could also cause network traffic jams as large numbers of file transfer requests converge upon the release dock shortly after it generates an announcement. Avoiding this kind of problem requires that the artifacts be replicated and cached in the network in such a way that a request for the artifact can be satisfied without going to the original source.

One possible solution is to physically include the artifact in the event, typically in the attribute list. With this solution the release dock needs to perform only one task to make a release available, namely to inject the announcement into WAM/E. Agents registered for the event would receive it and would only need to look in the attribute list to obtain the artifact. This approach, though, is best reserved for passing small artifacts to avoid slowing down the WAM/E message servers.

Another solution, which would not preclude the previous solution, involves establishing a separate artifact caching subsystem, where artifacts are cached in various places around the wide-area network. An existing solution for caching objects on wide-area networks, like Harvest, could be used for this task. In this case, the artifact would be propagated throughout the network automatically as agents retrieved the artifact from the originating release dock. This approach could be improved by creating or customizing an existing caching mechanism to cooperate with WAM/E event propagations. With a cooperative approach, it might be possible to have the artifact follow the associated events around, being cached along the way and possibly near the sites that are most likely to be interested.

3.6.3 Event Visibility and Scoping

Given the openness of WAM/E's event propagation paradigm, it is necessary to limit the scope in which an agent can register to receive events. If it were possible for an agent to easily register to receive all events it could quickly cause massive amounts of event traffic to descend upon the agent's site, causing a significant bottleneck. Some possible solutions might involve defining certain events as local, thus eliminating the possibility of someone registering for too many fine grained events. Another possible solution is to create some form of event scoping groups where only certain events are generated within the group, thus making it impossible to have an agent haphazardly register for all events.

3.6.4 Event Persistence

In order to cope with network unreliability, WAM/E must provide for controlled event persistence. Minimally, if a site is unavailable to receive events, WAM/E should queue events until such a time that they can be successfully received by the site. During this queuing process, certain events may become obsolete or unnecessary and, therefore, it may be possible to consolidate or remove them without having to maintain every event.

4 OLLA Deployment Demonstration

In this section we walk through a detailed example of an OLLA installation and update using the Software Dock prototype. This demonstration was performed using remote sites that simulated Lockheed Martin and an oversea school.

Figure 3a depicts a participating field dock and release dock at the start of the OLLA installation. In step (1) the OLLA installation agent is downloaded to the field dock. This download is initiated by either a user or another agent. Once the OLLA installation agent is in place, it starts to execute and queries the field dock's configuration registry in step (2). Using this information, the agent requests a dependency list for the proper configuration of OLLA from its release dock in step (3). Since OLLA is dependent upon Disco, the agent will check to see if this dependency is met. If not, it will make a request to the field dock to download and execute the Disco installation agent from the Disco release dock in step (4). This will result in a recursive installation of Disco, which will similarly invoke the Harvest installation agent in step (5) if Harvest is not present on the field site, since Disco is dependent upon Harvest.

Once all of the dependent systems have been installed, the OLLA installation agent retrieves the proper OLLA configuration from its release dock in step (6). In step (7) the installation agent creates an application node for OLLA in the field dock's registry. By creating a node in the registry the agent has generated an event that is received by the site agent in step (8). In response to the event the site agent creates a physical directory in the local site's file system for the new application.

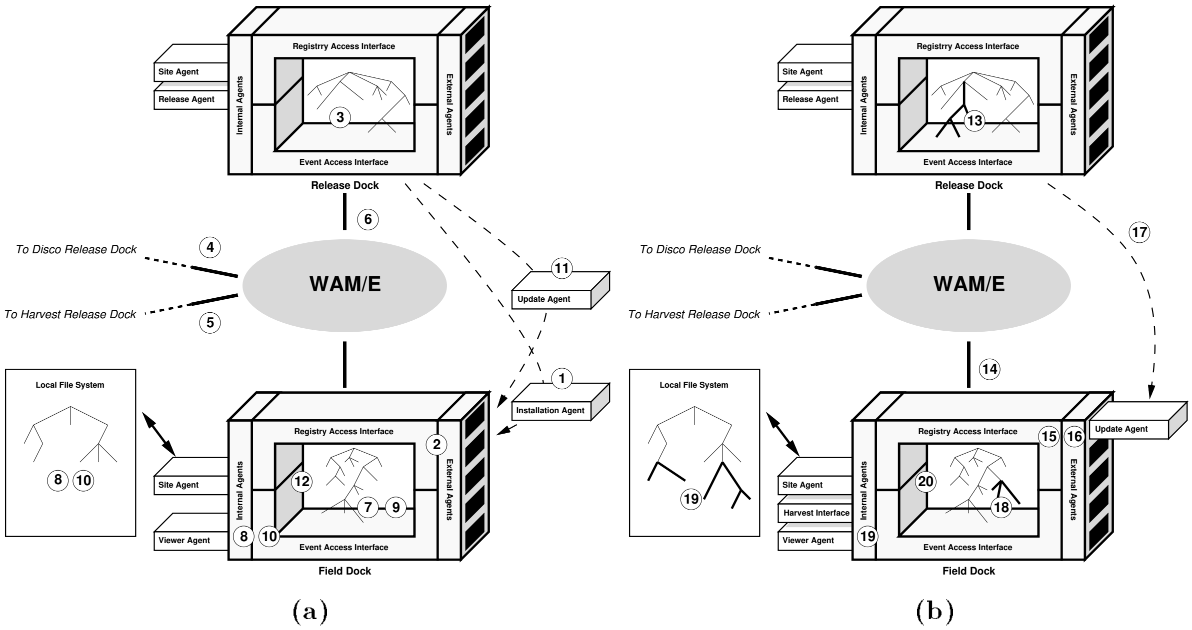

Figure 3: Installation (a) and Update (b) of the OLLA System.

In step (9), the installation agent then inserts the application specific files and directory nodes into the registry under the application node's "RootDir" node. Each of these insertions causes events that the site agent receives and then creates corresponding files and directories in the local site's file system in step (10).

A particular release artifact that the installation agent installs in step (11) is an update agent, which is an external agent responsible for obtaining and installing OLLA updates. The installation agent registers the update agent with the field dock, thus the update agent is "docked" with the field dock at the end of the installation. Before finishing the installation, the installation agent in step (12) inserts requests into the registry to create Harvest indexes for the newly installed OLLA system. As part of its installation, Harvest, which was installed as a dependent system of OLLA, added an agent to the local field dock to accept Harvest requests. The Harvest request agent responds to specific events in the registry and thus extends the indirect interface of the local site (see sections 3.3.3 and 3.5).

Figure 3b depicts the same participating field dock and release dock before an OLLA update release. In step (13) a user at the release dock's site inserts a new update for the OLLA system into the organization's release registry. The insertion of the update release causes an event to be generated and propagated through WAM/E in step (14). The update agent, which registered for update announcements when it docked at the field dock, receives the event in step (15).

The update agent queries the field dock registry in step (16) and then communicates with the originating release dock to retrieve the proper update configuration in step (17). In step (18) the update agent inserts the updated files into the proper places under OLLA's "RootDir" node and the site agent physically creates them in the file system in step (19). The final action taken by the update agent on behalf of Lockheed Martin is to request that Harvest refresh the content indexes if there were any changes in the content artifacts in step (20).

5 Related Work

Currently, there is no single system, other than the Software Dock, intended to perform the full range of software deployment activities. In this section we discuss various related systems, which fall into three broad categories: release management systems, build and installation systems, and configuration registry systems.

It should be noted that the Software Dock provides a complement to some of these other systems. In many cases, we expect to support interoperability with them, and where appropriate, adopt their solutions and standards. But it is our belief that they cannot be extended wholesale into the areas addressed by the Software Dock.

5.1 Release Management Systems

Release management is concerned with the process through which software is made available to and obtained by its users.

A number of commercial systems are available that support notions of software system release. Traditional network managers—HP's OpenView, IBM's NetView, and Sun's Solstice—were developed originally to manage hardware, and their software management support continues to lag. Tivoli's TME/10 [19, 20] most directly addresses the same issues as the Software Dock; it support configurations, dependencies, installation, and inventory. These systems assume a centralized control with a single site for configuration and releases. The Software Dock, on the other hand, supports a federated approach with multiple, autonomous consumers obtaining software systems from multiple, autonomous producers. Although internally event based, none of these systems supports the equivalent of WAM/E.

Some related but more limited systems include A+Enterprise Desktop Manager [2] and Ship [8]. They also support release management by enabling a developer to distribute their software to a specific set of users. However, these systems do not make dependency information generally available, nor do they address distribution and decentralization. Further, they lack event mechanisms and cannot support the automation of arbitrary deployment activities.

SRM (Software Release Manager) [21] was specifically designed to facilitate the distribution of software that is composed of sets of interdependent systems developed at geographically distributed and decentralized sites. Hiding the physical distribution and using the dependencies among the various systems, SRM allows retrieval of a complete software system through a Web-based interface. It does not provide any additional capabilities such as installation or update. Currently, we are using a special version of SRM as a primitive release dock in our Software Dock prototype.

5.2 Build and Installation Systems

Systems for building and installing a software system from source code provide certain features that overlap those of the Software Dock.

Surprisingly, the traditional configuration management systems (e.g., Make [7], RCS [18], ClearCase [3], and Continuus [5]) provide little support for the deployment of the systems whose configuration and construction they manage; they are almost exclusively concerned with software development at the source-code level.

The FreeBSD [15] porting system supports the FreeBSD user community by organizing freely available software into a carefully constructed hierarchy known as the "ports collection". It uses various forms of heuristics to determine a site's state and employs the results in building and installing a software package. The primary flaw in the system is that it embeds dependencies and other knowledge into Make files. Compared to the Software Dock's explicit registry, it is difficult to locate and manage dependency information about systems. Additionally, functional support is limited to only installation and de-installation.

Marimba's Castanet [13] is a content delivery system based on a client-server model of tuners and transmitters. Content, called a *channel*, is placed in a directory on a server transmitter and is made available to client tuners. Tuners that subscribe to specific channels will essentially mirror the channel directory on the local machine. Polling occurs at some per channel specified time interval and content-based, differential updating is performed to ensure that the local copy is consistent with the server copy. Limited communication for configuration is possible from the tuner to the transmitter in the form of log files and property files. There is no support for dependencies among channels, inter-channel communication, or policies other than differential updating.

NET-Install [1] enables software installation directly off the Web using Netscape plug-in technology. However, it does not have strong dependency management capabilities nor event support, and is tied to the Microsoft Windows platform, thus severely limiting its general utility.

5.3 Configuration Registry Systems

Configuration registries provides a logically centralized place for all the information needed to successfully release and deploy software systems.

GNU Autoconf [12] is a tool for producing shell scripts that automatically configure software source code build packages to adapt to many kinds of UNIX-like systems. In effect, its scripts construct much of the same information on-the-fly as that residing in a registry; in fact it could even be used to initially populate a registry. But the Autoconf approach does not scale because the information that it must calculate continues to grow over time, and this produces large and unwieldy scripts.

The Desktop Management Task Force (DMTF) has developed DMI, the Desktop Management Interface [6]. DMI consists of a format for describing management information called Management Information Format (MIF), a service provider, an API for management applications, an API for components, and a set of services for facilitating remote communication between participating objects. DMI acts as a layer of abstraction between management applications and manageable components. The Software Dock intentionally has many similarities to the DMI, and the perceived shortcomings of DMI have influenced the Software Dock architecture. For example, the DMI lacks a global event mechanism. Communication between components in the architecture is always client-server, limiting the interaction capabilities of the various components. DMI also does not address the actual tasks needed for deployment; installation, update, and other such deployment activities are not considered in its scope. Furthermore, the standard MIF schemas only support a fixed three-level hierarchy, whereas the Software Dock supports a more flexible multi-level hierarchical schema.

The Microsoft Registry is similar to DMI in many respects. It describes information about manageable components and it provides an access layer and a set of APIs for accessing this information. The Registry inspired the Software Dock registry model and it could be used as an underlying implementation. As with DMI, the registry is entirely passive and does not address the actual deployment activities. In addition, Microsoft has not taken the initiative to fully define a standard Registry schema, causing most information in the Registry to be proprietary and application specific. The Software Dock registry can be seen as combining the superior Microsoft Registry data model with the standard MIF schemas and coupling that with agent-based support for deployment functionality.

6 Conclusion

The activities of software deployment are complex and continuous and have largely been ignored by the software engineering community. Some ad hoc solutions that address specific software deployment activities have been created, but their limited applicability effectively places the burden of software deployment on the shoulders of users.

The Software Dock introduces a novel, general-purpose architecture to remedy this situation. By providing consistent access to a site's configuration information and resources, standardized methods for making software releases available and visible, and a wide-area messaging and event system, the Software Dock architecture creates new leverage for software producers and consumers to manage complex software systems. In particular, obstacles imposed by geographical distribution are overcome by making use of networks such as the Internet. Further, obstacles imposed by organizational distribution are overcome by standardizing some of the major parts of the architecture, while at the same time exhibiting enough flexibility to allow individual participants to adhere to their own standards.

The initial capabilities of the Software Dock architecture have been demonstrated with a prototype. The prototype includes versions of a provisional WAM/E, a field dock, a release dock, and a collection of agents. The prototype shows the feasibility of an agent-based architecture for performing complex software deployment activities. For example, the Software Dock is capable of performing a fully automated installation and configuration of Lockheed Martin's OLLA system, which consists of 45 megabytes of content spread across 1700 files.

Further development of the architecture and prototype will concentrate on expanding the global software deployment registry hierarchy with a typing system to allow for a more extensive global naming standardization mechanism in which different organizations can introduce their own registry node types. In addition, mechanisms to support transactional operations on portions of a registry's hierarchy and to support more sophisticated deployment activities are currently under investigation as part of the on-going effort to further enhance the Software Dock.

Acknowledgments

We thank Donald McKay, Suzanne Taylor, and Roslyn Nilson of Lockheed Martin for their help in preparing the OLLA demonstration. We also thank Antonio Carzaniga for his contributions to the implementation of the OLLA demonstration. Finally, we thank David Rosenblum for his comments on the paper.

REFERENCES

[1] NET-Install, 1996. Web address=www.twenty.com.

[2] A+Enterprise Desktop Manager, 1996. Web Address=www.amdahl.com.

[3] Atria Software, Natick, Massachusetts. *ClearCase Concepts Manual*, 1992.

[4] C. Mic Bowman, Peter B. Danzig, Darren R. Hardy, Udi Manber, and Michael F. Schwartz. The Harvest Information Discovery and Access System. In *Proceedings of the Second International World Wide Web Conference*, pages 763–771, October 1994.

[5] Continuus Software Corporation, Irvine, California. *Continuus Task Reference*, 1994.

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

[7] Stuart I. Feldman. Make – a program for maintaining computer programs. *Software – Practice and Experience*, 9:255 – 265, 1979.

[8] G. Fowler, D. Korn, H. Rao, J. Snyder, and K.-P. Vo. Configuration Management. In B. Krishnamurthy, editor, *Practical Reusable UNIX Software*, chapter 3. Wiley, New York, 1995.

[9] James Gosling, Bill Joy, and Guy Steele. *The Java Language Specification*. Addison-Wesley, 1996.

[10] Jerry Honeycutt. *Using the Windows 95 Registry*. Que Publishing, Indianapolis, IN, 1996.

[11] B. Krishnamurthy, editor. *Practical Reusable UNIX Software*, chapter 3, pages 91–120. John Wiley & Sons, Inc., New York, 1995.

[12] D. Mackenzie, R. McGrath, and N. Friedman. *Autoconf: Generating Automatic Configuration Scripts*. Free Software Foundation, Inc, April 1994.

[13] Marimba, Inc. *Castanet White Paper*, 1996. http://www.marimba.com/developer/castanet-whitepaper.html.

[14] Object Management Group. *The Common Object Request Broker: Architecture and Specification, Revision 2.0*, July 1995.

[15] The FreeBSD Documentation Project. *FreeBSD Handbook*. FreeBSD Documentation Project, 15 May 1996. Web Address=ftp.freebsd.org, Path=/pub/FreeBSD/docs/handbook.tex.

[16] D.S. Rosenblum and A.L. Wolf. A Design Framework for Internet-Scale Event Observation and Notification. Technical Report 97-06, Department of Information and Computer Science, University of California, Irvine, California, February 1997.

[17] Robert W. Scheifler and James Gettys. *X Window System*. Digital Press, 3rd edition, 1992.

[18] W.F. Tichy. RCS, A System for Version Control. *Software—Practice and Experience*, 15(7):637–654, July 1985.

[19] TME/10 Software Distribution. Web Address=www.tivoli.com, Path=/products/Courier.

[20] Tivoli Systems Inc. *Applications Management Specification*, 1995.

[21] A. van der Hoek, R.S. Hall, D.M. Heimbigner, and A.L. Wolf. Software Release Management. Technical Report CU-CS-806-96, Department of Computer Science, University of Colorado, Boulder, Colorado, August 1996.