A Characterization Framework for Software Deployment Technologies

Antonio Carzaniga[†‡], Alfonso Fuggetta[‡], Richard S. Hall[†], Dennis Heimbigner[†], André van der Hoek[†], and Alexander L. Wolf[†]

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

[‡]Dip. di Elettronica e Informazione
Politecnico di Milano
20133 Milano, Italy
fuggetta@elet.polimi.it

University of Colorado
Department of Computer Science
Technical Report CU-CS-857-98 April 1998

© 1998 Antonio Carzaniga, Alfonso Fuggetta, Richard S. Hall, Dennis Heimbigner, André van der Hoek, and Alexander L. Wolf

ABSTRACT

Software applications are no longer stand-alone systems. They are increasingly the result of integrating heterogeneous collections of components, both executable and data, possibly dispersed over a computer network. Different components can be provided by different producers and they can be part of different systems at the same time. Moreover, components can change rapidly and independently, making it difficult to manage the whole system in a consistent way. Under these circumstances, a crucial step of the software life cycle is deployment*—that is, the activities related to the release, installation, activation, deactivation, update, and removal of components, as well as whole systems.*

This paper presents a framework for characterizing technologies that are intended to support software deployment. The framework highlights four primary factors concerning the technologies: process coverage; process changeability; interprocess coordination; and site, product, and deployment policy abstraction. A variety of existing technologies are surveyed and assessed against the framework. Finally, we discuss promising research directions in software deployment.

This work was 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.

1 Introduction

Software applications are no longer stand-alone systems. They are increasingly the result of integrating heterogeneous collections of components, both executable and data, possibly dispersed over a computer network. Consider the relatively simple case of web browsers, which are in fact shells into which a variety of applications, such as document viewers and collaboration tools, can be fit to create the effect of an integrated system. Such "systems of systems" are created by integrating components from different development organizations having different release schedules and different goals.

Software producers in this context no longer distribute complete systems. They must therefore find a way to deal with greater uncertainty in the environment within which their systems will operate. For example, before they can guarantee a successful installation, producers must be able to determine what components are available at a given site, as well as the configuration of those components. Additionally, since the components are produced by multiple organizations, they must be able to anticipate or react to updates to components that are not under their control. These issues are further magnified when the scale of the Internet—in terms of the vast numbers of producers and consumers, as well as the great distances involved—is taken into consideration.

Clearly, this is a daunting task that creates new challenges in the areas of release, installation, activation, deactivation, update, and removal of components. These activities constitute a large and complex process that we refer to as *software deployment*. The growing complexity of software systems mandates that software deployment activities be given special attention.

Recently, a number of new technologies have begun to emerge to address the deployment problem. Typical features offered by these technologies include system and configuration description, package and installation construction, automatic update delivery, and various network management capabilities. One emerging technology that has a particularly strong relationship to software deployment is content delivery. Systems such Castanet, PointCast, and recent enhancements to Microsoft and Netscape web browsers, make it possible to simplify and automate the transfer of raw data from one site to another. While software deployment includes a content delivery activity, in which the system to be deployed must be physically moved from the producer site to the consumer site, this is just one step that makes up the larger deployment process.

Despite the proliferation of deployment technologies, we have no clear understanding of the issues related to Internet-scale deployment, nor a way to characterize the suitability of these technologies to address those issues. Thus, the purpose of this paper is threefold. First, we analyze and characterize the challenges and issues of software deployment. Second, we develop a framework for characterizing existing and proposed deployment technologies. This framework highlights four primary factors that characterize the maturity of the technologies: process coverage; process changeability; interprocess coordination; and site, product, and deployment policy abstraction. Third, we provide a survey of a variety of systems and an assessment of them against the framework.

The main conclusion that can be drawn from this work is that no single existing system is yet able to comprehensively and coherently cover the full range of deployment activities in a way that factors out critical properties and characteristics of the entities involved in the deployment process. Moreover, it is not clear how different deployment technologies could be integrated to satisfy these requirements. Our framework and the results of our characterization suggest areas where attention should be focused to guide future research and development activities in software deployment.

The paper is organized as follows. In the next section we present a motivating example to illuminate the issues in software deployment. From this example, we derive basic concepts and requirements. Section 3 presents our characterization framework. A number of representative

existing technologies are then characterized against that framework in sections 4 and 5. We conclude with some thoughts about future research directions suggested by the characterization.

2 Characterizing Software Deployment

Informally, the term *software deployment* refers to all the activities that make a software system available for use. While this definition is reasonable and intuitively clear, the creation of a characterization framework for software deployment technologies requires a more precise and comprehensive understanding of the nature and characteristics of these activities. This section explores the multiple facets of software deployment from several different viewpoints. In particular, we first introduce a motivating scenario that highlights the role of software deployment.[1] We then study and discuss software deployment from a structural viewpoint by identifying its basic constituent activities. Finally, we present requirements and constraints that must be taken into account in the development of the characterization framework.

2.1 A Motivating Scenario

A medium-size organization uses an information system that operates over a wide-area computer network. The organization is hierarchically structured into a number of branches. Each branch has a local-area network with a branch server, several workstations, and personal computers. Every branch server is connected to a central server that resides at the organization's headquarters. The organization manages approximately thirty servers and a few thousand PCs and workstations. The headquarters as well as every single branch is connected to the Internet.

The organization's computing environment is host to a number of different applications. The central server and the branch servers manage the main database of the organization. PCs and workstations run database client applications, Web servers, mail systems, and common stand-alone tools for office automation (e.g., word processors and spreadsheets).

Each branch of the organization needs to consult a common set of files containing prices and product information. These files are administered by the organization's headquarters. However, to avoid traffic overhead, these files are replicated at every branch so that the requests can be handled locally. The files are updated once per day.

The organization acquires hardware and software components from a number of different providers. Some of the providers are connected to the Internet and offer electronic commerce facilities that provide on-line catalogs, negotiation of product features, and electronic purchase transactions. Some providers also offer on-line maintainance and automatic update services so that updates can be automatically deployed by the provider, either in response to bug reports or to release new versions of products.

The configuration of the information system, both hardware and software, undergoes regular change. The organization's headquarters is responsible for maintaining the hardware and software configuration of systems and services that are shared among branches (e.g., the main database and the mail servers). Each branch is given freedom to customize its local environment to meet the specific needs of its business. Specifically, each branch is responsible for the configuration of branch-specific systems and information, such as database client applications, stand-alone tools, and user profiles. There may be applications installed on every machine in a branch, as well as common branch-wide applications that can be shared through a server.

[1] This example is inspired by the Digital Equipment Corporation's Project Gabriel [25].

The obvious requirement of the organization is to minimize the overall cost of software management and deployment, where the cost function is determined by the loss or the down time of functionality due to deployment activities, the additional labor spent in those activities, and the amount of computational and communication resources used for deployment. The organization policies also impose strict requirements over the deployment activities. For example, interference caused by deployment activities during business hours are not tolerated. Moreover, for security purposes, the organization must have full control over every deployment activity. This is achieved by means of strict authorization and test procedures.

2.2 Basic Terminology

We envision a network of computers partitioned into *sites*, where each site hosts a set of *resources*. In most cases, a site refers to a single computer. In general, however, a site may be a strongly coupled set of computers administered identically. A *software system* is a coherent collection of artifacts, such as executable files, source code, data files, and documentation, that are needed at a site to offer some functionality to end users. We assume that software systems evolve over time, and that a *version* of a system refers both to time-ordered revisions as well as to platform-specific and/or functional variants.

A *resource* is anything needed to enable the use of a software system at a site. Examples include IP port numbers, memory, disk space, as well as other systems. Some resources (e.g., data files) may be sharable, while others may be used by only one system at a time (e.g, IP port numbers).

Deploying a software system involves the transfer or copy of its constituents from a *producer site* to one or more *consumer sites*, which are the targets of the deployment process. Once deployed, a software system is *available for use* at the consumer site.

2.3 The Software Deployment Process

It should be evident from the motivating scenario presented above that software deployment is a complex process in its own right. The deployment process consists of several interrelated activities, as depicted in Figure 1. Arrows in the figure indicate possible transitions between activities for some particular deployed system, suggesting a life-cycle model for a system in the field. No assumption is made, however, about the location where activities are carried out; they can occur at the producer site (e.g., releasing an update) or at the consumer site (e.g., removing a component), or at both (e.g., configuring a component).

Although we can identify a set of distinct and key activities that typically constitute a generic deployment process, we cannot precisely define the particular practices and procedures embodied in each activity. They heavily depend on the nature of the software being released, and on the characteristics and requirements of the producers and consumers. Therefore, Figure 1 should be interpreted as a reasonable process that has to be customized and enriched according to specific requirements of the deployment activity being observed. In the remainder of this section we briefly discuss the general characteristics of this generic software deployment process.

Release. The release activity is the interface of the deployment process with the development process. It encompasses all the operations needed to prepare a system for assembly and transfer to the consumer site. Thus, the release activity must determine the resources required by a software system to correctly operate at the consumer site. It must also collect the information that is necessary for carrying out subsequent activities of the deployment process. This information may

be derived from a variety of sources including the development process itself and human knowledge about the system structure and operation.

The release activity includes *packaging* the system so that it can be transferred to the consumer site. This package must contain the system components, a description of the system, including its requirements and the dependencies on other external components, the deployment procedures, and all the information that is relevant for the management of the system at the consumer site.

Another step in the release activity is *advertising*, i.e., the set of operations that are needed to disseminate appropriate information to interested parties about the characteristics of the system being released.

Install. The installation activity covers the initial insertion of a system into a consumer site. Usually, it is the most complex of the deployment activities because it deals with the proper assembly of all the resources needed to use a system. It is also currently the activity best supported by specialized tools.

Installation involves two distinct sub-activities. The first one is the *transfer* of the product from the producer site to the consumer site. The second one consists of all the *configuration* operations that are necessary to make the system ready for activation on the consumer site.

Activate. Activation is the activity of starting up the executable components of a system. For a simple system, activation involves establishing some form of command (or clickable graphical icon) for executing the binary component. For a complex system, it might be necessary to start servers or daemons before the software system itself is activated.

Note that any of the activities of the deployment process might require the activation of other systems in support of that activity, which in turn might require recursive deployment of those support systems. For example, if a system has been packaged as an archive file, the installation procedure must be able to activate the archive tool to extract the pieces of the system to be installed. If the archive tool is not available, then a recursive deployment of the archive tool would be required.

Deactivate. Deactivation is the inverse of activation, and refers to the activity of shutting down any executing components of an installed system. Deactivation is often required before other deployment activities, such as update, can commence.

Update. The process of updating a version of a system is a special case of installation. However, it is usually less complex because it can often rely on the fact that many of the needed resources have already been obtained during the installation process. Typically, the deployment life cycle includes an iteration where a system is deactivated, a new version is installed, and then the system reactivated. For some systems, deactivation may not be necessary and update can be performed while a previous version is still active. Similar to installation, update also includes the transfer and configuration of the components needed to complete the operation.

Adapt. The adaptation activity, like the update activity, involves modifying a software system that has been previously installed. Adaptation differs from update in that the update activity is initiated by remote events, such as a software producer releasing an update, whereas adaptations are initiated by local events, such as a change in the environment of the consumer site. An adaptation activity may be initiated to take corrective action to maintain the operational correctness of the deployed software system.

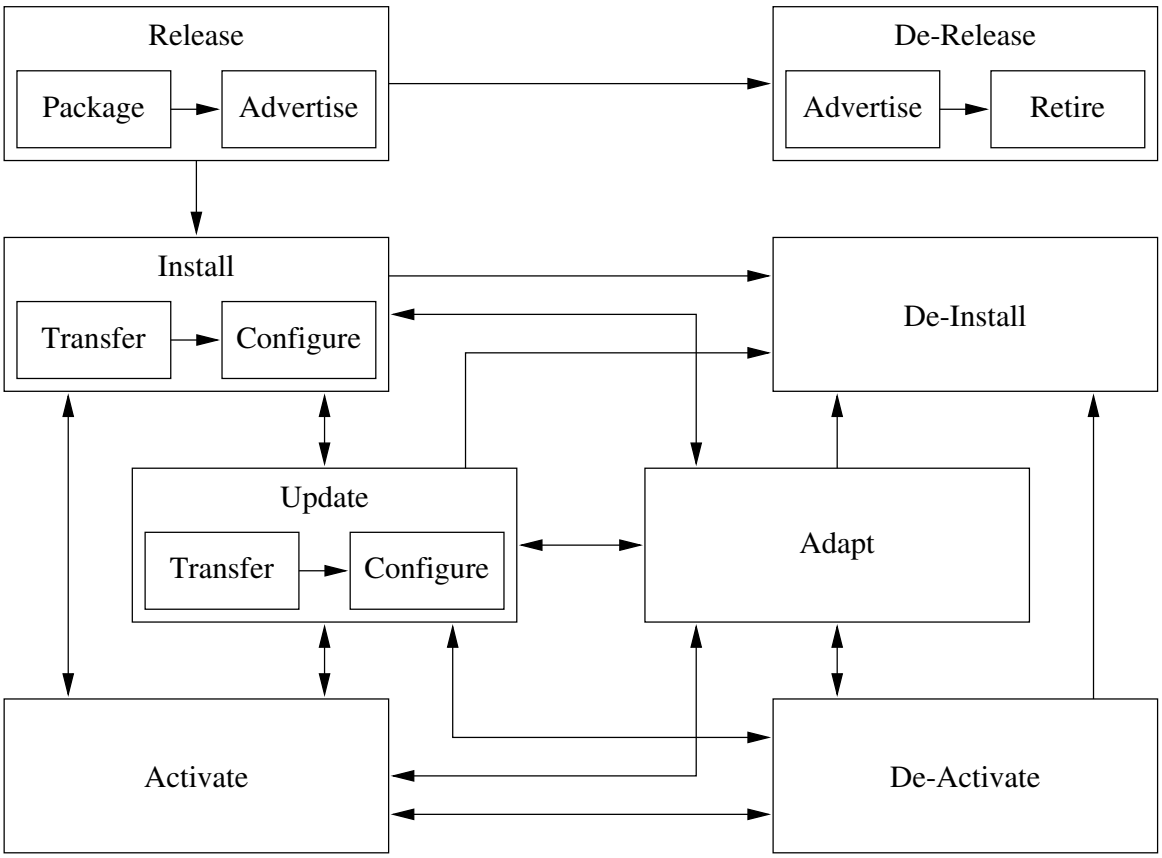


Figure 1: Activities of the Software Deployment Process.

Deinstall. At some point a system as a whole is no longer required at a given consumer site and can be removed. Of course, deinstallation presumes the system is also deactivated. The deinstallation activity possibly involves some reconfiguration of other systems in addition to the removal of the deinstalled system's files.

There is a critical issue of properly deinstalling systems that were installed as a result of deploying the system. In particular, subsequent dependencies may arise that must be taken into account by the deinstallation activity.

Derelease (Retire). Ultimately, a system is marked as obsolete and support by the producer is withdrawn. As with deinstallation, care must be taken to ensure that the withdrawal will not cause difficulties. This requires that the withdrawal be advertised to all known consumers of the system.

2.4 Software Deployment Issues

The scenario presented in Section 2.1 illustrates common problems encountered in deploying software systems. The terminology defined in sections 2.2 and 2.3 makes it possible to refer in a consistent and comprehensive way to the different activities that constitute the software deployment process. We now outline the general issues that characterize software deployment.

2.4.1 Change Management for Installed and Running Systems

The evolution of a computerized system has a significant impact on software deployment activities. Evolution is both natural and inevitable. For instance, hardware components, such as new network interfaces and video cards, may be added to a computer to achieve better performance or support new functionality. In turn, these replacements may induce changes to installed software components, such as drivers or other operating system modules. Similarly, the deployment of new application features or the need for improved performance may require the installation or update of entirely new software systems. Another very common activity is the adaptation of the run-time configuration of running components, and of logical components or common environments, such as user profiles, shared directory structures, and common disk workspaces.

2.4.2 Dependencies among Components

As a consequence of modular design and reuse practices, any non-trivial software system will consist of multiple components exhibiting various interdependencies. A dependency is any kind of "use" relationship that holds among software components. Dependencies significantly increase the complexity of every step of the deployment process. For instance, in the case of a client-server application, the client configuration depends on the server configuration and location, and vice versa. Stand-alone applications show a number of dependencies as well, introduced by the existence of shared components. For example, a presentation tool may require a graphical editor that may already be installed as part of a word processor. Another kind of dependency is introduced when components rely on other components for their installation, activation, initialization of data, update, or recovery from failures or exceptions. A common example is an installation-time dependency on archiving and decompression tools, such as *zip*.

2.4.3 Coordination

The need for coordination emerges quite clearly from the example of Section 2.1. In that scenario, client-server applications are deployed and used on different sites, possibly managed in separate administrative domains. It is usually the case that some deployment activities on the server require the execution of some other actions on the client. The execution of these activities must be coordinated to make sure, for instance, that all the clients are deactivated before the server is updated or reconfigured. As another example, deployment activities must be scheduled so that they do not interfere with the normal business tasks. This may require some procedures to be deferred to off-peak times, or to interrupt activities that cannot be completed in the alloted time.

2.4.4 Large-scale Content Delivery

In the example presented in Section 2.1, a major problem is the transfer of data files from the organization's headquarters to each branch in a way that minimizes transmission costs. In general, one important step of the deployment process is the transfer of information between different locations of a network. We refer to this activity as *content delivery*. It requires special attention in the face of various scale issues: size of components, number of sites, and variable quality of transmission service.

2.4.5 Managing Heterogeneous Platforms

The connectivity offered by computer networks and the use of standard communication protocols have made possible the development of large-scale, heterogeneous, distributed applications. In the scenario of Section 2.1, we have mainframes, workstations and servers, and personal computers, all possibly running different operating systems. The coexistence and the interoperability of heterogeneous platforms pose new challenges for software deployment. At the minimum, the software that supports deployment has to be ported to every platform. Moreover, the platform type becomes a new variable that has to be taken into account when dealing with configurations and dependencies. For example, consider an application using a library. If the application is built using dynamic linking, each platform-specific version of the application will depend on the corresponding platform-specific version of the library. On a platform that does not provide dynamic linking, that same application may not require any shared library at all.

2.4.6 Deployment Process Changeability

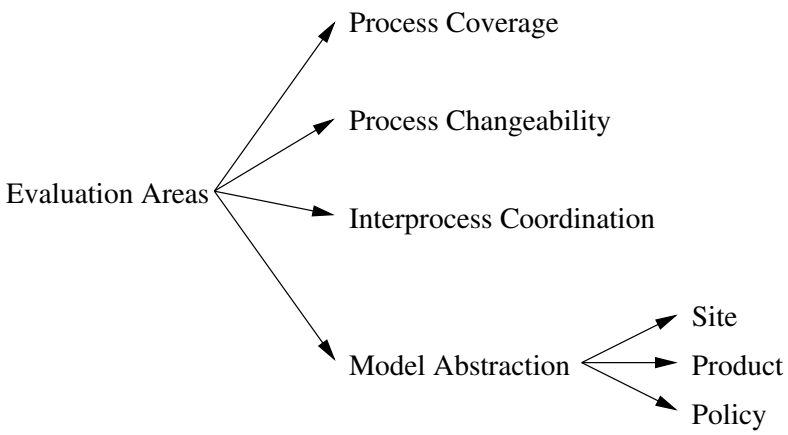
An important requirement for the software deployment process is the provision of adequate mechanisms to support its change to meet new or unexpected requirements. In general, most deployment activities execute at the consumer site, make use of system resources, and often require privileged access to system components. Therefore, it is essential for administrators to be able to tailor these activities in a flexible and effective way. For example, a security policy may indicate that any software system must undergo a standard inspection before it is installed (e.g., checked for viruses). In specific cases (e.g., mission-critical applications), it may be desirable to enforce a stronger policy requiring a preliminary trial installation in a safe environment.

2.4.7 Integration with the Internet

The advent of the Internet has created a virtual, worldwide marketplace. Producers can advertise their products by making technical specifications and requirements (e.g., dependencies) available on-line. Customers are able to evaluate the impact of the installation of a new product and can exploit the Internet to provide producers with feedback, comments, suggestions, bug reports, usage patterns, and new requirements. In response, producers can deliver on-line support and automatic deployment of upgrades. Therefore, the deployment process and technology must be tightly integrated with the Internet and must be able to exploit the features offered by this innovative infrastructure.

2.4.8 Security: Privacy, Authentication, and Integrity

The Internet is an insecure network. This fairly obvious observation introduces a variety of issues and concerns. Specifically, there are three aspects of computer security that are critical with respect to software deployment: privacy, authentication, and integrity. In the scenario of Section 2.1, critical information (i.e., product information, including prices) was transmitted from the headquarters over an insecure wide-area network. Clearly, in this case, the organization is concerned with privacy—that is, they want to transfer the files to every branch in such a way that nobody outside the organization can read them. As a second example, we have the headquarters accessing, for management purposes, some resources at each branch. Here, reliable authentication procedures must be in place so that only the authorized managers can gain privileged access to the machines at a branch. Finally, the organization interacts with many software providers and establishes automatic update procedures. Even if the transfer of software is carried out in a secure way,



]

Figure 2: Areas of Characterization.

there might still be security concerns related to the installation of software in the "production" environment. In particular, it is important to guarantee the integrity of the organization's data against the execution of malicious or incorrect procedures that may cause corruption or loss of data during installation or update.

3 Characterization Framework

Having delineated the basic terminology and issues associated with software deployment, we now present a characterization framework for deployment technologies. The framework divides the characterization into four primary, and largely orthogonal categories, as depicted in Figure 2. Each is explained below. A detailed discussion of the characteristics of deployment technologies that lead to different ratings in each category is left to the sampling of available technologies in sections 4 and 5.

3.1 Process Coverage

Figure 1 illustrates the richness and complexity of the software deployment process. Each box in the figure represents a significant activity requiring specialized support from a deployment system. This leads to our first characterization criterion, *process coverage*, which is the degree to which a given deployment system covers each of the constituent activities of the process.

We recognize three values in the characterization of process coverage: no support for the given activity, minimal support, and full support. When a technology does not explicitly recognize an activity as part of its process, we say that there is no support. A minimal support for an activity is provided by a technology that has at least a "hook"—that is, an access point for the user to supply an implementation for that activity—but does not itself implement the activity. Full support is provided by a technology that implements at least a default version of the activity and precisely defines the interface to the activity so that it can be fully integrated into the overall process.

3.2 Process Changeability

The definition of a particular deployment process for a given product and a given consumer site can be a difficult task. Sometimes the developer of the process, typically someone at the producer

site, cannot fully anticipate all the requirements for the process. For example, a consumer may want to insert steps into the process that are intended to perform specialized tests of the status of the deployment. This suggests our second characterization criterion, *process changeability*, which is an indication of whether the deployment process can be changed after definition.

Here we are concentrating, not on the normal evolution of the process to correct mistakes or respond to modifications to the product, but instead on unanticipated and ad hoc changes typically performed by consumers. An ability to change the process implies an explicit, manipulable representation of the process.

3.3 Interprocess Coordination

Coordination support is required when various deployment activities associated with different systems, possibly distributed over a network, have to cooperate and synchronize. This situation can arise, for example, when installing a composite system that requires and triggers the deployment of the subsystems upon which it depends. Another common example is found in client-server systems, where an update to a running server requires coordinated deactivation and reactivation of the clients.

As far as coordination is concerned, we examine the ability of a deployment technology to handle distributed and composite systems. In particular, we examine whether or not a technology supports synchronization and data exchange among different activities, and whether this support is extended over a wide-area network.

3.4 Site, Product, and Policy Abstraction

An especially useful way to characterize deployment technologies is to consider the relative difficulty of defining a particular deployment activity from the perspective of the creator of that activity. A deployment activity can be seen as a procedure for controlling execution and resource allocation. In principle, this procedure should be provided for each combination of product, consumer site, and set of execution policy constraints.[2] The degree to which information about the product, site, and policy can be abstracted out of the procedure allows the procedure to be used in a wider range of situations. This reduces the effort required to define a deployment process and leads to our fourth characterization criterion, *model abstraction*.

We start by considering the "worst" case for deployment, where one would need $Si \times Pr \times Po$ different deployment procedures for each deployment activity, where Si is the number of consumer sites, Pr is the number of products, and Po is the number of possible policy constraints imposed on a deployment activity. Informally, this case would be similar to having a deployment system that required separate scripts (e.g., "Make files") for each activity, for every product, at every consumer site, and with every kind of execution policy. Clearly this can lead to a large number of such scripts and the consequent high cost of their individual development.

The idea here is to examine the extent to which the different deployment technologies are able to factor out site, product, and policy information from the deployment procedures for each activity. This factoring results in generic models of the three kinds of deployment information. The information can then be used as parameters to the procedures, rather than having to be "hard wired". Thus, the deployment procedures themselves become generic, reducing the total number

[2]Recall that a product can consist of multiple components, each of which may be independently developed. Similarly, a consumer site can consist of multiple, heterogeneous machines, each of which may be managed by independent organizations.

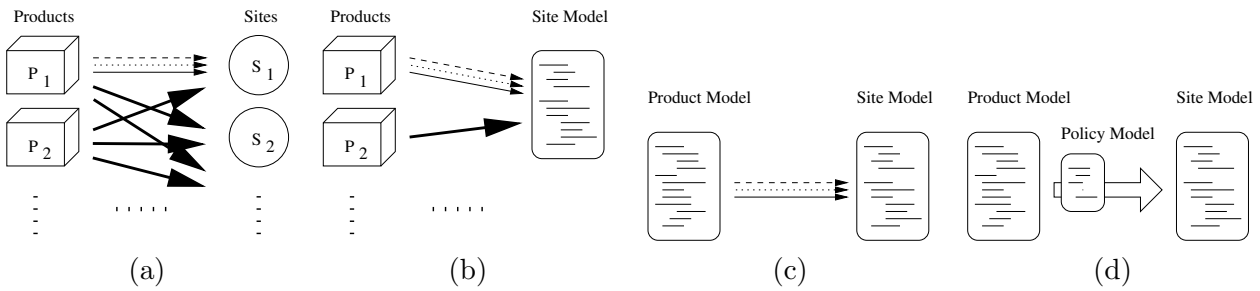


Figure 3: An Example of Site, Product, and Policy Factoring.

of deployment procedures that must be defined. In the best case, this reduces to one procedure for each activity discussed in Section 2.3.

Figure 3 shows a scenario in which one deployment activity—say, installation—must be defined for m products on n sites with three policies. The three policies in this example are: *confirm* (dashed arrow), requiring a confirmation by the consumer before taking any action; *notify* (dotted arrow), informing the consumer of every step taken; and *automatic* (solid arrow), performing every action silently. To simplify the picture somewhat, bold arrows are used as a shorthand for the three separate arrows.

Moving from left to right in the figure, we see a progressive reduction in the number of specialized deployment procedures. Figure 3(a) depicts the "worst" case in which an installation procedure, in the absence of any information abstraction, must be defined for each of the $3mn$ product, site, policy combinations. Developing a site model improves the situation, with $3m$ procedures parameterized by site information. Having a product model further reduces the number of required installation procedures to 3, one for each of the policies. Finally, in Figure 3(d) we see the best case, only one installation procedure that accepts a policy parameter in addition to site and product parameters.

The site model, the product model, and the policy model define a conceptual architecture that we use together with the deployment process model to characterize commercial and research deployment technologies in sections 4 and 5. It turns out that none of the technologies that we examined is designed to provide all of the abstractions, although elements of each can be found in many. A prototype that explicitly refers to this conceptual architecture, implementing site and product models, is presented in [6].

To characterize the ability of deployment technologies to abstract site and product information, we examine the following issues:

- *completeness*—the ability to capture all the necessary information that can be associated with a site or product;
- *orthogonality*—the ability to separately model different aspects of a site or product, and the ability to combine independent site or product attributes;
- *availability*—the ability to manipulate the information from deployment procedures; and
- *openness*—the compliance to well-known standards, and interoperability with other technologies.

Because of the importance of model abstraction in characterizing deployment technologies, we now explore site, product, and policy models in greater depth.

3.4.1 Site Model

A site model is a standardized way of describing or abstracting a consumer site's resources and configuration. A site model for a single computer would contain information such as the machine type, the operating system, the available hardware resources, and the available software resources. The site model enables all consumer sites to be treated in the same manner, since standard methods would then exist to query the site's configuration and to access required resources for performing deployment activities. This gives the producer the ability to ignore consumer-site anomalies. In this respect, the site model specifically addresses the issue of heterogeneity discussed in Section 2.4.5.

Mechanisms such as GNU's Autoconf [13] and the Microsoft Registry [8] demonstrate how this factoring can be achieved, but in two rather different ways. Autoconf is used to produce a single program, *configure*, that dynamically computes a site abstraction. The Registry, in contrast, is a passive repository containing the site abstraction. In both cases, the deployment process, or more accurately the installation activity, is significantly simplified, since a producer can construct installation scripts that are parameterized by common information available from the site model.

3.4.2 Product Model

The product model, though similar in purpose to the site model, does not abstract the same type of information. The product model is used to create a full description of the constraints and dependencies of a system to be deployed, such that all deployable systems can be reasoned about in a consistent manner by a particular deployment procedure. The product model should include information such as producer contact information, subsystem dependency specifications, the set of constituent files, and documentation. As a counterpart to the process specification, the product model should describe the state of a product throughout the deployment life cycle. As long as a product is made up of a single component with minimal and/or standard system requirements, achieving product factorization is fairly straightforward. But for more complex systems composed of multiple, distributed, and heterogeneous components, the complexity increases significantly.

Similar to the site model, a product model should define a standard schema together with some access and query functions so that deployment activities can use and possibly transform the product information. It is often the case that the product information is integrated into the site information once a system is installed. This integration of the site and product models constitutes the basis for managing configurations, as discussed in Section 2.4.1.

3.4.3 Policy Model

We conceive of a deployment policy as a particular way of tailoring the execution of a deployment activity. For example, consider the installation of a product that depends upon other products. In such a case, a sequence of integrity and compatibility checks should be performed. A strict policy for performing these checks might require that they all be done before the actual installation takes place, while a looser policy might delay the checks until the dependent products are actually needed. Another example of alternative policies for the same activity concerns whether updates should be pushed or pulled. Under both the push and pull policies, the installation activities are essentially the same, differing only in when and how updates are triggered.

The policy model should include information describing such things as scheduling, ordering, preferences, and security control. Unlike many aspects of site and product models, however, it is not obvious how policy information can be abstracted into a generic schema. Within our characterization framework, therefore, we examine only the policy issues related to parameters for scheduling

and control of resource usage.

4 Current Technologies

A wide variety of technologies already exist to support various parts of the deployment process, at varying degrees of sophistication, and using a number of different approaches. Below we discuss many of these by giving a summary description, and then characterizing them with respect to the framework presented in the previous section. We do not describe every available technology, but instead group them into five classes. Each class is then described by presenting the most relevant features among the union of the features of all the technologies that belong to the class. Although the technologies that we analyze here are a significant sampling of the current deployment technologies, this section should not be regarded as a complete survey nor as a critique. It is rather an example of the practical application of the characterization framework.

Summarizing, the purpose of this section is threefold. First, it describes the technologies, highlighting their features in relation to software deployment. Second, it provides an initial classification of a significant number of deployment technologies. Third, for each class of deployment technology, it shows how to apply our framework so that others may use it.

The results of our sampling of technologies are collected into two tables. Table 1 presents a characterization of the technologies in terms of process coverage. Table 2 assesses their support for changeability, coordination, and model abstraction. In the tables, a filled circle ("•") indicates full support, an empty circle ("○") indicates minimal support, and the absence of a circle indicates no support at all.

4.1 Installers

PC-Install 6 with Internet Extensions [21], InstallShield 5 [11], and netDeploy 3 [20] are representative of the class of deployment technologies referred to as *installers*. The primary focus of installers is to package a stand-alone software system into a self-installing archive that is able to be distributed via physical media or networks. Most installers also perform some form of deinstallation by undoing the changes made during installation.

Installers, as a class of deployment systems, generally have some limited notion of a product model. This model varies among installers, but at the very least includes a list of the component files that comprise the system to be installed, as well as version and platform information. In addition to a product model, most installers also have a consumer site model. The consumer site model generally consists of site configuration information, the user environment, and the file system. The site model, however, tends to be either platform specific or otherwise severely limited. Finally, the process by which installation occurs tends to be very rigid and cannot be easily customized.

Given the connectivity and popularity of the Internet, it is reasonable to expect installers to increasingly branch into related deployment activities. For example, netDeploy 3 already supports the update deployment activity to a limited extent.

4.2 Package Managers

Almost every modern operating system comes with a suite of utilities that assist system administrators in installing, updating, and generally managing software. Examples of such utilities include Linux RedHat's RPM [3, 1], HP-UX SD commands [7], and SUN Solaris *pkg* commands [24]. These deployment technologies are based on the concept of *package*, and on a site repository that stores

	System	release		install		act	de-act	update		adapt	de-inst	de-rel
		pack	adv	trans	conf			trans	conf			
Installers	PC-Install 6	●			○						●	
	InstallShield 5	●			●						●	
	netDeploy 3	●		○	○	○		○	○		●	
Package managers	RPM	●		○	●			○	●	○	●	
	HP-UX SD	●			●				●	○	●	
	SUN *pkg*	●			●				●	○	●	
Application management	TME-10			●	●	●	●	●	●	○	●	
	SystemView			●	●	●	●	●	●	○	●	
	OpenView			●	●	●	●	●	●	○	●	
	Platinum			●	●	●	●	●	●	○	●	
	EDM				●	●	●		●		●	
Standards	MIF											
	AMS											
	Autoconf				○				○			
Content delivery	Castanet		○	●				●				
	PointCast		●	●								
	Rsync			●				●				
	Rdist			●								

Table 1: Deployment Process Coverage in Current Technologies.

	System	changeability	coordination	model abstraction		
				site	product	policy
Installers	PC-Install 6				○	
	InstallShield 5	○			○	
	netDeploy 3				○	○
Package managers	RPM	○		○	●	○
	HP-UX SD	○		○	●	○
	SUN *pkg*	○		○	●	○
Application management	TME-10	●	○	●	●	
	SystemView	●	○	●	●	
	OpenView	●	○	●	●	
	Platinum	●	○	●	●	
	EDM	●	○	●	●	
Standards	MIF			●	○	
	AMS			●	●	
	Autoconf				○	
Content delivery	Castanet					
	PointCast					
	Rsync					
	Rdist					

Table 2: Changeability, Coordination, and Model Abstraction in Current Technologies.

information representing the state of each installed package. A package is an archive that contains the files that constitute a system together with some meta-data describing the system. The main functionalities provided by package managers are to create a package, install (i.e., unpack and configure) a package, query a repository or package, verify the integrity of an installed package, update a package, and deinstall a package.

The package meta-data are usually supplied by the software producer. They are then used by the package manager to create the package on the producer side. Once the package is installed, the information from the package is stored in the consumer site repository. The procedures of the package manager are given access to this repository to query and possibly even update the information.

A substantial portion of the deployment process is covered by package managers. Besides supporting the packaging activity on the producer site, package managers more or less support every deployment activity on the consumer site except for activation and deactivation. A limited form of content delivery is supported by package managers incorporating file transfer facilities that allow transparent access to "remote" package files.

In terms of model abstraction, the site model inherent in package managers is simply the collection of packages and a repository, but the product model is more sophisticated, since it provides a rich set of attributes in the package meta-data. Package managers are the only deployment technology that provide some sort of primitive policy model. In particular, a small set of parameters can be supplied to the deployment procedures to modify their behavior. Typically, however, the parameters only control whether some predefined internal action is taken or avoided.

There are two major limitations to package managers as currently available. First, they are targeted to a single machine or, at best, to a set of machines that share a network file system. Thus, there is little support for large-scale deployment and for distributed systems that require coordination. Second, because their site abstraction is primarily static—that is, the site model does not accommodate any run-time information—package managers are not in a position to provide support for activation and deactivation.

4.3 Application Management Systems

TME-10 [26] from Tivoli, SystemView [23] from IBM, OpenView [9] from Hewlett Packard, System Management tools [22] from Platinum, and EDM [19] from Novadigm are representative of so-called network or application management systems.[3] Their original purpose was to support the management of corporate LANs. They are capable of detecting hardware failures and network disruptions and reporting them to some operations center for examination. Recently, they have ventured beyond hardware and have begun addressing the problems of software management, including some parts of the deployment process. It is evident that this class of deployment technology is targeted towards medium- or large-scale organizations that usually play both the roles of producer and consumer.

Application management systems generally have a centralized architecture. There is usually a logically centralized "producer", which is a designated central administration site for all officially approved releases. Correspondingly, all the management and deployment activities are typically controlled by a central management station.

With respect process coverage, application management systems essentially support all of the

[3]This kind of high-end technology is difficult to characterize due to the lack of publicly available technical documentation. Thus, we base our discussion on descriptions supplied by the technology providers themselves and, therefore, are unable to directly verify their claims.

deployment life cycle activities except the producer-side activity of release. They make use of an explicit definition of the deployment activities that includes the ability to execute sub-installations on multiple hosts and to coordinate the installation of distributed systems. They support activation, deactivation, and monitoring of applications with the possibility to set up call-back diagnostic routines that respond to anomalies or exceptions generated by the monitoring facility. All the supported deployment activities, including content delivery, can be programmed and coordinated in a distributed environment.

Application management systems are all based on repositories, usually centralized, that store deployment meta-data. The meta-data contain information on product packages (see Section 4.2), site configurations, and deployment process specifications. In some cases the repository coincides with the one provided with a package manager. For example, the HP OpenView system is a sophisticated extension of the HP-UX SD package manager. The site information includes the static and dynamic configuration of all the machines belonging to the site, including their hardware, software, and various pieces of run-time information. An interesting capability of application management systems is *inventory*. In particular, they are capable of scanning a consumer site and determining the set of installed systems. This information is then brought back to the repository. Finally, deployment activities can be scheduled and optimized according to the hierarchical topology of the site.

4.4 System Description Standards

It should be evident that deployment technologies critically depend on various information structures. As a consequence, the two key factors for the applicability and interoperability of all deployment technologies are the completeness and standardization of those structures.

To this end, a number of organizations have proposed various standards to describe software and hardware components for purposes of deployment and management. The Desktop Management Task Force (DMTF) is the major organizational force here and is proposing a standard called the Management Information Format (MIF) [4, 2]. The standard specifies the description of various hardware and software properties. Another effort has been made by IEEE with its POSIX standard for software administration [10]. Tivoli's Application Management Specification (AMS) [27] is derived from the DMI. It specifically targets the description of application software systems. The Simple Network Management Protocol (SNMP) [15] defines a standard for schemas of information, primarily for hardware components of networks. To some extent, GNU Autoconf [13] falls into this category as well by providing a consumer site abstraction and various techniques to determine the consumer site configuration.

In terms of our characterization framework, the standards are used to specify both the site and the product models. Because their primary objective is to model software and hardware components in changing network environments, they are flexible enough to incorporate new elements and structures. They subsume an articulated deployment process and provide an access point to each deployment activity.

4.5 Delivery of Content

Marimba's Castanet [14], PointCast, Rsync [28], and Rdist [17] directly implement the content delivery activity. In this class of technology, the information being deployed is simply transferred from one or more information servers to a number of client sites.

Depending on the particular technology, the information that is transferred can be an application, a data file, or some kind of news bulletin. In any case, the data sent by the server is opaque to

	System	release		install		act	de-act	update		adapt	de-inst	de-rel
		pack	adv	trans	conf			trans	conf			
Research	ArchShell					●	●	○	○	○		
	SW Dock	●	●	●	●			●	●			
	SAA						●	●	○	○		

Table 3: Deployment Process Coverage in Research Technologies.

	System	changeability	coordination	model abstraction		
				site	product	policy
Research	ArchShell				●	
	SW Dock	○	○	●	●	
	SAA				●	

Table 4: Changeability, Coordination, and Model Abstraction in Research Technologies.

the client—that is, there is no configuration or meta-data in the information flow and, other than for some particular functionalities, the data are not interpreted by the client at the consumer site. One functionality that does perform interpretation is present in delivery technologies providing automatic update of the client, wherein the producer can push an update using the same channel as it uses to deliver the content.

From the point of view of our models, these systems simply support the transfer activity for either install or update and they do not have any model abstractions. However, it is worth noting that they have adopted quite different technologies for carrying out the data transfer function, with an effort to make it scalable and efficient, especially in a low-bandwidth or costly network environment.

5 Research Approaches

The previous section discussed mature software deployment technologies that are currently available. We now present three technologies that are still research efforts and, as such, are still in their infancy. These new approaches are important because they are addressing some of the most complex issues in software deployment. In particular, ArchShell [16] and the Software Architecture Assistant (SAA) [18] are addressing the dynamic aspects of the adapt and update activities. The Software Dock [6] is concerned with integrating support for the whole deployment life cycle into a single, systematic framework. Tables 3 and 4 summarize our characterization of these research technologies according to our framework.

5.1 Configurable Distributed Systems

ArchShell and the Software Architecture Assistant are two technologies that are representative of a new, emerging class of research that is concerned with supporting configurable distributed systems (CDS). The goal of CDS research is to allow the reconfiguration of a system after it has been deployed. In particular, component evolution needs to be administered in such a way that consistency of the deployed system is guaranteed, even in cases where it is required that a system continues executing while the evolution takes place. Both the adapt and update activities are addressed by CDS technologies.

To support the adapt activity, specialized programming constructs and libraries have been developed that allow a system to reconfigure itself when its environment changes. Typical changes that are supported are related to replication and migration of services as network connections become unreliable or disappear. Changes within the installation environment of a system, such as changes in the location of system files or changes to the hardware configuration, are not supported at this time.

Update is supported by CDS technologies through product models that are captured within formally defined architecture description languages. Both ArchShell and the Software Architecture Assistant use the product model to dynamically insert and remove components in a running system. They are also capable of changing the topology of a product—that is, changing connections among components.

5.2 Software Dock

The Software Dock [6] is a distributed, agent-based framework to support the entire software deployment life cycle. The Software Dock addresses the software deployment life cycle by defining a rigorous and thorough schema to describe both products and sites [5]. The data encompassed by the schema is housed in registries at the producer and consumer sites.

The schema for describing software systems capture semantic information such as constraints, subsystem dependencies, component relationships, and distributed coordination. The schema for describing consumer sites captures the current state or configuration of the site where a software system is to be deployed. The consumer site schema includes information such as operating system and platform information, available resources, and existing software systems and their configurations.

The information provided by the schema definitions is accessible through the registry servers at the producer and consumer sites. This information is combined with agent technology, where agents are used as generic process interpreters of the semantic descriptions. In other words, agents realize and perform specific deployment activities. A collection of generic agents perform standard deployment tasks, while customized agents can be created to perform specific tasks. The agents can coordinate their activities and, thus, support coordinated software deployment.

6 Conclusions

In this paper we discussed the challenge of software deployment, a part of the larger software life cycle that has been to date given short shrift by the software engineering community. The process underlying software deployment is complex and must be better understood in order to produce improved, more automated approaches to the whole deployment life cycle.

To further our understanding of deployment, we have defined some terminology, and then introduced a three-part model that provides a framework for characterizing software deployment technologies. This model consists of a deployment process specification and a conceptual architecture consisting of product, site, and policy abstractions. Using this model, we have surveyed a set of representative technologies that are either currently available or under development by researchers.

The result of this exercise suggests several lines of future research. First, the model itself should be deepened to give better characterizations of the problems surrounding deployment. In particular, a better understanding in the area of policy and coordination modeling has to be achieved. Second, we can find no instance of a deployment technology that supports every deployment activity. In fact, there is no evidence to suggest that the deployment technologies can be easily integrated to

cover the whole spectrum of activities. Third, we notice that existing deployment technologies fail to scale in various respects. In particular, we found two major classes of technologies. High-end technologies provide sufficient site and product abstraction, as well as broad process support. They are best suited to large-scale deployment, but are "closed" systems that do not allow integration with other network services, nor are they easily adapted to new situations. In contrast, low-end technologies are light weight, providing good support for both abstraction and process dimensions, but only for a single or tightly-coupled set of systems and sites. They offer little or no support for coordination and large-scale deployment.

The main contributions of the paper are therefore the characterization framework and the (initial) assessment of available and research software deployment technologies. Certainly, since software deployment is a relative new and unexplored domain, this work will need further extensions and revisions, as new insights and results are achieved by the research community. Nevertheless, this paper provides a solid foundation for directing future research and development efforts.

References

[1] E.C. Bailey. *Maximum RPM.* Red Hat Software, Inc., February 1997.

[2] Desktop Management Task Force Inc. *Desktop Management Interface Specification*, March 1996. http://www.dmtf.org/tech/specs.html.

[3] M. Ewing and E. Troan. The RPM Packaging System. In *Proceedings of the First Conference on Freely Redistributable Software*, Cambridge, MA, USA, February 1996. Free Software Foundation.

[4] Desktop Management Task Force. Software Standard Groups Definition, November 1995. http://hplbwww.hpl.hp.com/people/arp/dmtf/ver2.htm.

[5] R.S. Hall, D. Heimbigner, and A.L. Wolf. Software Deployment Languages and Schema. Technical Report CU-SERL-204-08, Department of Computer Science, University of Colorado, December 1997.

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

[7] Hewlett-Packard Company. *HP-UX Release 10.10 Manual*, November 95.

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

[9] HP OpenView, 1998. http://www.hp.com/openview/index.html.

[10] IEEE Standard for Information Technology. *Portable Operating Interface System–Part 2: Administration (POSIX 1387.2)*, 1995.

[11] InstallShield, 1997. http://www.installshield.com.

[12] B. Kantor and P. Lapsley. Network News Transfer Protocol, A Proposed Standard for the Stream-Based Transmission of News. RFC 977, February 1986.

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

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

[15] K. McCloghrie and M. Rose. Management Information Base for Network Management of TCP/IP-based Internets. RFC 1156, May 1990.

[16] N. Medvidovic. ADLs and Dynamic Architecture Changes. In L. Vidal, A. Finkelstein, G. Spanoudakis, and A.L. Wolf, editors, *Joint Proceedings of the SIGSOFT '96 Workshops*, pages 24–27, New York, New York, 1996. ACM Press.

[17] D. Nachbar. When Network File Systems Aren't Enough: Automatic Software Distribution Revisited. In *Proceedings of the USENIX 1986 Summer Technical Conference*, pages 159–171, Atlanta, GA, June 1986. USENIX Association.

[18] K. Ng, J. Kramer, and J. Magee. A CASE Tool for Software Architecture Design. *Journal of Automated Software Engineering*, 3(3/4):261–284, August 1996.

[19] Novadigm's EDM, 1998. http://www.novadigm.com/c1.htm.

[20] OpenWEB netDeploy, 1997. http://www.osa.com.

[21] PC-Install, 1996. http://www.twenty.com.

[22] PLATINUM System Management, 1998.
http://www.platinum.com/products/sys_mgmt.htm.

[23] RS/6000 - System Management, 1998.
http://www.austin.ibm.com/resource/aix_resource/Pubs/redbooks/ooksc16.html.

[24] SUN Microsystems. *SunOS 5.5 Manual*, September 1992.

[25] O.H. Tallman. Project Gabriel: Automated Deployment in a Large Commercial Network. *Digital Technical Journal*, 7(2):52–70, October 1995.
http://www.europe.digital.com/.i/info/DTJI05/DTJI05SC.TXT.

[26] TME/10 Software Distribution.
http://www.tivoli.com/products/Courier.

[27] Tivoli Systems Inc. *Applications Management Specification*, 1995.
http://www.tivoli.com/products/tech_info/AMS/AMS.html.

[28] A. Tridgell and P. Mackerras. The Rsync Algorithm. Technical Report TR-CS-96-05, Department of Computer Science, The Australian National University, Canberra, Australia, June 1996.

A Additional information

This appendix provides more detailed information on some of the more significant and popular systems supporting installation and content delivery. Obviously, given the limitation in space and the fast rate of change in the field, this presentation does not attempt to be exhaustive. Rather, the purpose is to provide some additional justification to the structure and organization of the framework proposed in this paper.

A.1 Installers

PC-Install 6 with Internet Extensions PC-Install 6 with Internet Extensions is a Windows-based installation product. PC-Install supports the creation of installation packages that can be automatically installed from physical media or from a network via Web-browser plug-in technology.

PC-Install uses a product model to describe the software system to be installed. The software system model provided by PC-Install includes the ability to describe conditional components depending on consumer site configuration and optional software system components. The component files associated with the system are then gathered into an archive and accompany the semantic description.

PC-Install provides a consumer site model, though it is very platform specific. Access to information such as CPU, OS version, disk space, memory, video, sound card and modem communication port is provided. It is also possible to interact with the desktop and registry environment provided by Windows, locate and check for files, and modify site configuration files.

PC-Install creates a script, using the description of the software system to be installed and the information provided by the consumer site model. The script executes locally on the consumer site and an engine interprets the actions specified in the script. The creation of a PC-Install package is simplified by an easy-to-use graphical interface and the result is a fully graphical, automated installation package. PC-Install can also perform a deinstallation of the installed software system.

InstallShield 5 InstallShield5 is a Windows-based installation product. InstallShield5 supports the creation of installation packages that can be automatically installed from physical media. The resulting installation package can also be downloaded from networks.

InstallShield5 is very similar in features and function to PC-Install with Internet Extensions, with the exception that it does not support direct installation from a network. An additional system, called InstallFromTheWeb, provides direct network installation capabilities.

netDeploy 3 netDeploy is a multi-platform installation and update product. It takes a different approach to deployment than other products. netDeploy specifically targets installing and updating software systems directly from wide-area networks.

netDeploy is divided into two parts, a packer and a launcher. The packer creates complete file sets to be installed. The package can embed actual files or contain URL pointers to files. The packages created by the packer are not as sophisticated as other installers, but it does enable installation of dependent software packages. The launcher portion of netDeploy is a user tool in the form of a Web-browser plug-in that handles the installation, validation, and activation of the deployed software system. The launcher maintains a cache of installed software packages and performs automatic updates of the file sets when they change at the producer site.

A.2 Content Delivery

PointCast and ZIP delivery PointCast and ZIP Delivery provide news multi-casting services that are an evolution from the Internet News system [12]. A consumer can determine which data they wants to receive by subscribing to a number of "channels", possibly from different producers. The subscription, or some configuration on the consumer site, determines how often or in response to which events the channel is to be updated. Unlike Internet News, only unidirectional communication is possible.

Castanet The same publish/subscribe protocol described above is adopted by Castanet. Castanet is content delivery system that has some additional features to deal with applications other than news. A Castanet channel is in essence a set of files. On a regular basis, depending on the configuration of the channel at the consumer site, an update for that channel is pulled. Castanet enables the producer to customize the channel with a channel, which is an application that manages communication with the consumer.

Web Channels Recent versions of Netscape and Microsoft Web browsers introduce new features for content delivery. These facilities convey the idea of an information "push" that allows the user to configure a set of links mimicking a subscription to an information channel. Nevertheless, subscriptions remain local and the browser simply polls and possibly reloads Web pages periodically.

Rsync Rsync is a file synchronizer that supports file update in an efficient manner. The Rsync operation involves two machines: the source machine, which has the files or the new versions, and the target machine, which has the set of files to be deployed. Rsync can be invoked in the same way by either the source or the target. It transfers only the portions of a file that differ on the consumer site, thus performing an incremental update at a fine granularity.

Rdist Rdist is a tool that allows efficient distribution of files from a source machine to a large number of target machines over a network. Rdist introduces multi-casting at the network and transport layer.