Pp. 33-46 of the *Proceedings of LISA '02: Sixteenth Systems Administration Conference* (Berkeley, CA: USENIX Association, 2002).

Application Aware Management of Internet Data Center Software

Alain Mayer – CenterRun, Inc.

ABSTRACT

We have built a comprehensive solution to address the management aspects of deployment and analysis of applications in Internet Data Centers. Our work was motivated by the high total cost of ownership of operating such centers, largely due to the variety of applications and their distinctive management requirements. We have chosen an approach that encapsulates application specific knowledge (is *application aware*) and deployed it in a number of corporate Internet Data Centers. Operations staff found substantial cost reduction in managing applications using our approach.

Introduction

A corporate Internet Data Center consists mostly of Web server farms, application server farms, and database servers. Frequently there is a heterogeneous server environment running Windows 2000, Solaris, Linux, and AIX platforms, often due to corporate mergers and acquisitions. Acquired third-party software can include (1) Web servers such as Apache, iPlanet (SunONE), Microsoft IIS, (2) application servers, such as BEA WebLogic, IBM WebSphere, Microsoft MTS, iPlanet (SUNOne), and (3) database servers such as Oracle and Microsoft SQL servers. In addition, enterprises have a large quantity of in-house developed software (J2EE applications, ASP/JSP pages, COM(+) components, etc.) which need to be deployed and configured on top of the third party software.

There has been excellent prior work on infrastructure deployment and management (see the pioneering paper [TH98] and references therein, such as [R97], or books, such as [LH02], [B00]). These solutions (and some of their tools, e.g., JumpStart, KickStart, Ghost, SUP, etc.) mostly focus on more generic, lower level aspects of the infrastructure, such as OS and standard network servers (DNS, mail, etc.). We advocate building on these existing solutions and at the same time creating new management technologies, which are *application aware*. By this we mean that instead of operating only on the level of files and directories, such solutions capture knowledge about an application such as its configuration methods, requirements, and more. Operations personnel can use such knowledge to define methods to deploy, install, upgrade, and start applications. Once defined, these methods can be executed repeatedly and reliably through a "push" method or through a "pull" method (what might be done today using tools such as "rdist" or "cfengine" [CF02], respectively).

It is important to note that the approach of application-aware management technologies extends well beyond the realm of the Web applications in Internet Data Centers. However, for the sake of concreteness and because of the importance of Web applications, this paper focuses on our efforts to create solutions for Internet Data Centers.

In the next section, we introduce Application Management as an emerging and important field of system administration and then motivate our approach of application-aware management. Subsequently, we present the required building blocks of our approach and shows how they interact with each other to form a comprehensive solution. The next section shows some of the technologies needed to realize the building blocks. After that, we quantify some of the benefits seen by operations people using application-aware technology and finally conclude.

Importance of Web Application Management

In every industry, business is moving online. After migrating business information to databases in the eighties and setting up e-commerce applications in the nineties, organizations of all sizes are relying on web applications for core business operations. With the advent of Web services, this trend will only be reinforced.

Application management today operates on file, directory, and configuration parameter levels. As most operations managers can attest, application changes are frequently hectic ordeals, involving custom-built scripts (sometimes written only moments before they are first used), best guesses about hardware and software configurations, remote locations, late hours, and overworked staff. Changes can involve complex combinations of commercial software, in-house applications, and custom scripts. Operations staff is responsible for understanding the requirements for all these components, deploying them quickly and flawlessly, and remembering what they have done at a detailed level.

Apart from the actual deployment, application management includes other substantial challenges. It is often important to detect what has changed and by whom in deployed applications (e.g., the administrator on duty at the time of an emergency made a quick fix to some Web server configuration file/database, which is not consistent with the overall policy). Similarly, operations staff needs to quickly pinpoint deployment and configuration differences among two servers. Severe reliability problems often lead operations staff to undo and rollback application deployments (we note that not every application can be cleanly rolled-back due to its possible side effects, such as changing the OS state).

Cost of Managing Web Applications

Below are the summary points of a representative Internet Data Center environment. The associated cost of managing application deployments and analysis in such an environment is discussed later.

- 120 servers, 26 applications
- Applications are all running on top of Apache Web servers and either Weblogic or WebSphere application servers.
- Collecting application changes and deploying them once a week.
- IBM consulting project to document manual change processes did not result in any improvements in quality or in cost.
- Costly, time-consuming errors, such as running out of disk space during an application deployment or forgetting to add required database tables during an n-tier application update.
- Pain Manifestation: ‘‘My team of seven spent 16 hours on one WebSphere deployment’’

Motivating Application Awareness

Here we present some typical workflows involving applications both on the UNIX and Windows platform.

J2EE Applications

J2EE application servers, such as Weblogic, WebSphere, and iPlanet (SunOne) implement their own ‘‘logical topology’’ on top of the network of physical server machines. A ‘‘server instance’’ is a software component that makes one or more J2EE applications available on a physical machine. Each physical machine may host one or more server instances. Each product has its own way of creating, grouping, and managing its server instances.

J2EE applications (e.g., Enterprise Java Beans, EJB’s) implement the core business logic (second tier in an n-tier architecture) of most Web-enabled applications. These applications are typically packaged in archive formats, such as EAR (Enterprise Archive) or WAR (Web Archive). These archives contain configuration information in XML. Each vendor extends the basic XML configuration, affecting the way the J2EE application is deployed to the application server. Operations personnel often get these archives from the development organization and might have to open the archive to edit configuration values. The deployment of these applications to a server instance always has to be effected via the responsible administrative console server. Details of the deployment commands differ among vendors and even among versions from the same vendor. For example, the configuration information for each J2EE application running on WebSphere is stored in a centralized database, which is read and written by the administrative console during each deployment, upgrade, or other change.

Closely coupled with J2EE business logic are the Web applications, the first tier in an n-tier architecture. These applications are deployed onto Web servers (Apache, iPlanet, etc.) and contain JSP pages, static content, and more. They are often deployed together with the second tier as a single logical software component, forming cross-machine dependencies. Furthermore, the Web servers need to be configured to connect to the appropriate application server instances.

Windows Applications

On the Windows platform, IIS is the dominant Web server platform and MTS is the dominant platform for the business logic applications. Deployment of applications (ASP pages, ISAPI filters, etc.) onto IIS (versions 5.x) require each configuration to be reflected in the ‘‘metabase,’’ a registry like database resident on each Windows 2000 server. Any configuration information about IIS itself also has to be reflected in the metabase. On the Windows .Net server and IIS version 6.x, the metabase is realized as an XML file. Business logic and transactional components are typically packaged as COM or COM+ components. Deployments onto MTS require the registration of these components with the Windows registry database. Any Web or business logic application might require additional manipulation of the registry database. Within the .Net framework, applications are packaged as assemblies, which have yet another deployment philosophy.

Database Dependencies

Most applications require access to database servers running Oracle, SQL, or similar. While deploying and configuring such servers might not be a frequent operation (relative to changes to applications), the deployment of an application typically does require some manipulation of the database (e.g., tables, stored procedures).

Need for Application Awareness

Given the description of Internet Data Center software above and the illustration of Figure 1, it becomes evident that deploying and managing applications requires a lot of application specific knowledge. This is unlike basic infrastructure management of UNIX-based machines, which has been pioneered and described in [TH98] and the corresponding tools for UNIX or Windows (JumpStart, Ghost, isconf [TH98], cfengine [CF02], etc.). There is an unmet need for technology and solutions to automate the deployment and management process beyond host and OS management, and beyond pushing or pulling files and directories of applications.

Application Aware Management

Application aware management requires technology that can (1) capture knowledge about an application, such as its installation, registration, configuration methods and its dependency management and (2) automate processes based on this knowledge. In the following, we present some of the key components of such an approach, building upon known technologies of [TH98], such as central version control, Gold Server, and basic server (host and OS) set-up tools. See also Figure &2, which summarizes the building blocks described below, some of the building blocks introduced in [TH98], and their dependencies. Some of the building blocks in [TH98] have been merged under "Host&OS" management, which can be argued is a pre-condition for application management as a whole. The building blocks above the fat line were introduced in [TH98]; we created the blocks below that line. Lines between building blocks denote that the block below depends on the block above being realized in the system. The system we built includes all building blocks except the "Host&OS" block.

- An **Application Model** is a data-driven representation of an application (including executables and configuration files, content, etc.) and associated methods to deploy, configure, and analyze the application. A model has to be reusable, so that it can be applied to different server environments (e.g., staging vs production), which might require different deployment or configuration choices. Rather than capturing "hard-coded" configuration settings of the application, the model contains variables for such settings which the user can instantiate during the deployment process.
- The **Model Builder** automates the creation of application models. It captures deployed applications from a Baseline Server (e.g., a machine in the QA environment), checks them into a central version-controlled repository ("Gold Server" approach), and at the same time creates a base model of the application. The model builder associates the type of the captured application (e.g., J2EE application on WebSphere, Web application on IIS) with an appropriate base model, including methods for deployment, analysis, and discovery (see subsequent building blocks). The model builder then allows customizing the base model by adding dependencies and configuration customizations (see subsequent building blocks) to the model. An illustration of workflow enabled by this building block is shown later.
- The **Deployment Manager** automates the deployment process of an application end-to-end. For each checked-in and modeled application, this process assures that the application will be correctly installed on the desired server machines. In other words, the deployment management module forms the runtime system for the modeled methods.
- The **Configuration Manager** determines the desired configuration of an application according

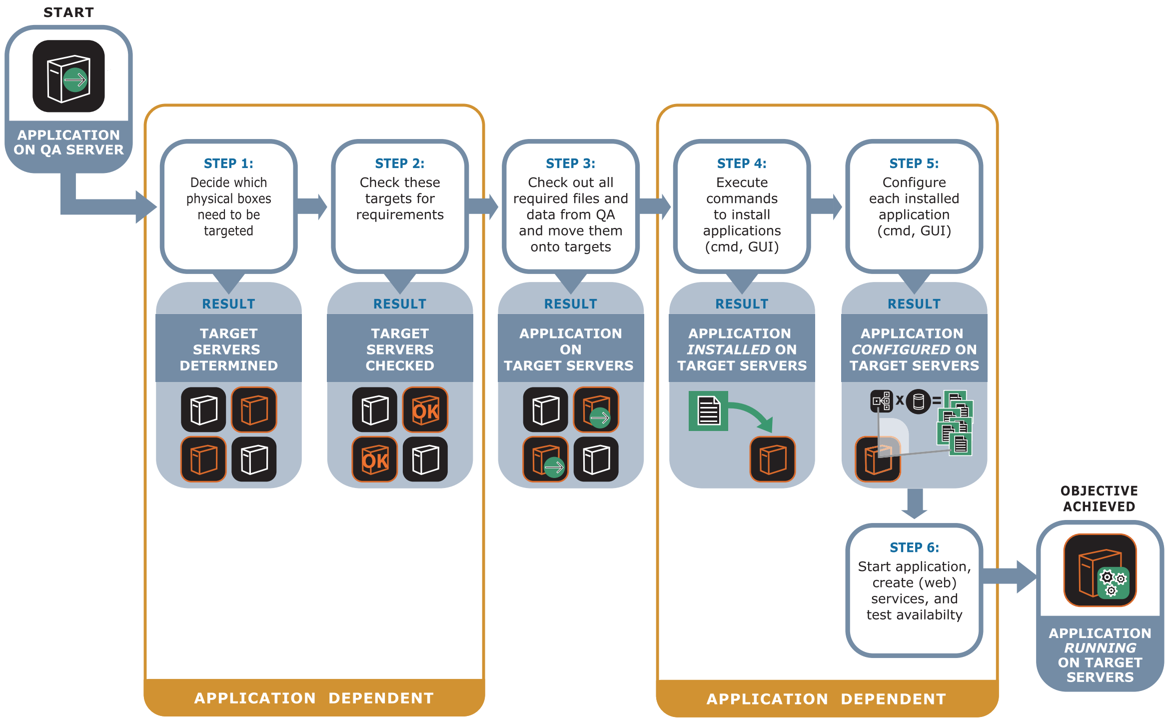

Figure 1: The deployment and configuration of web applications is a complex process, where most steps are application dependent, meaning they differ from application to application ("QA" = quality assurance).

to the environment (e.g., number of CPUs, database connector, and thread-pool of the target server). It then generates the configuration by setting values in appropriate text files, XML files, or modeled methods of the application. In this way, the configuration manager can even write to database-like structures on the target servers, such as Windows registry, IIS metabase, WebSphere data stores, etc.

- The **Dependency Manager** ensures that all modeled requirements for a successful deployment are met (including across different machines). This occurs before the deployment manager makes any changes to the target server.
- **Automated Analysis** pinpoints configuration, version, and other differences between data center servers. This detects the "out-of-band" changes, that are difficult to suppress in most Internet data centers, such as when an operations person changes a configuration value not using any sanctioned tools, but rather by hand, for example during an emergency server recovery procedure.
- **Application Discovery** collects information on what applications are already deployed on which server in the Internet data center. The application model guides the discovery process, as it knows what features indicate the presence of an application on a server. See [MDEGGH00] for a good introduction to model-driven application discovery.

Figure 3 summarizes how the above components map onto the deployment and analysis process.

Figure 4 illustrates how these technologies fit into a system solution (which we call "CenterRun"). The architecture consists of a master server and remote agents. This solution offers a centralized console to the operations personnel. From this console (command line or Web GUI), applications can be captured from Baseline Servers, application models can be created, and deployments can be executed as captured in the application model. Every action is logged and archived. Installed applications can be analyzed and compared across servers. Below we describe the workflow as offered by the centralized console. We use two concrete and very different applications, a J2EE application on WebLogic (following the sample application "petstore," one of Sun's Java blueprint applications) and an n-tier Web application on Windows (following the Microsoft sample application "FMStocks," see http://www.fmstocks.com).

Workflows Enabled by Application Awareness

Workflow 1: Deploy a J2EE Application on WebLogic

1. The Model Builder captures the J2EE application on a Baseline Server:

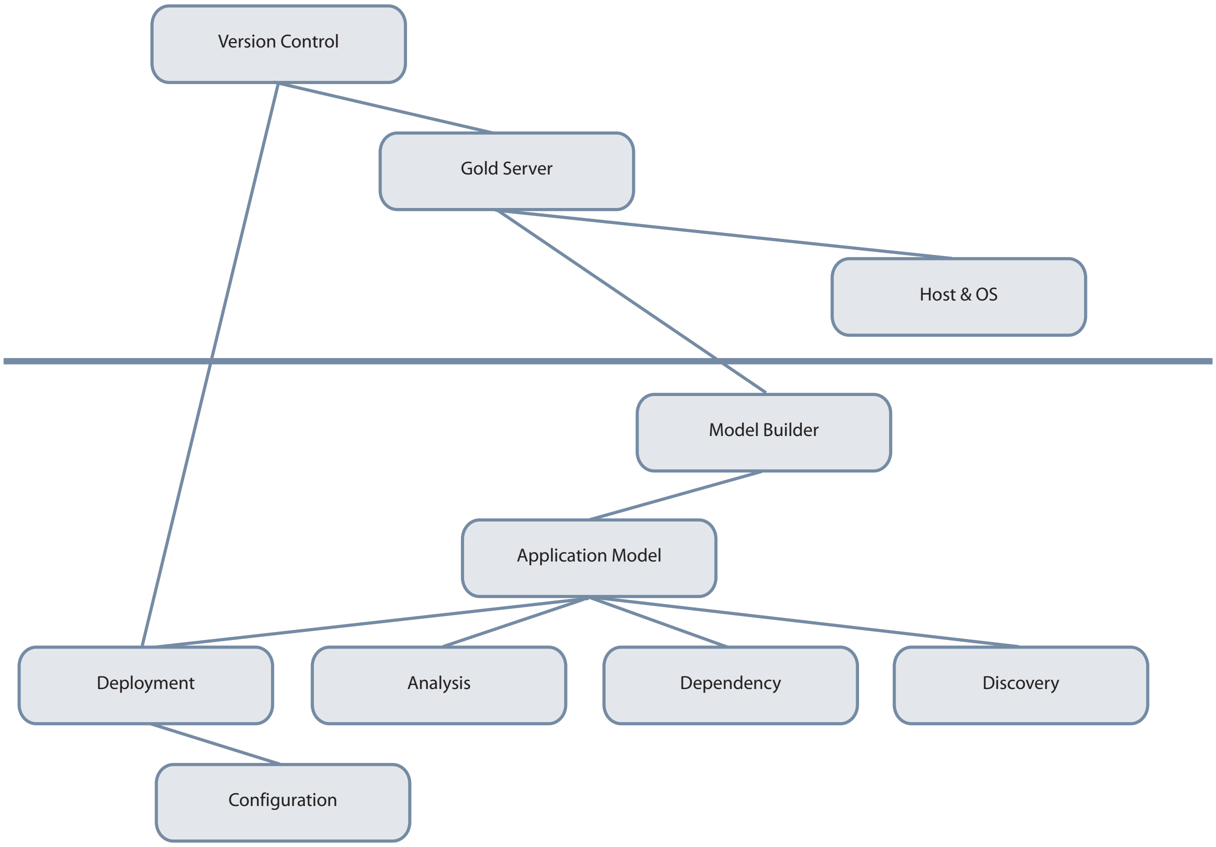

Figure 2: Building blocks for application management.

a. Recognizes the application as a J2EE application on Weblogic.
b. Creates an application model, capturing and describing all the relevant files and archives, such as EAR and WAR.
c. Checks these resources into the master server's repository, where they are versioned.
d. Captures the relevant configuration information from the Weblogic XML file, "config.xml."
e. Uses predefined models to add all the necessary methods to install and uninstall the application to the newly created application model.

2. The Dependency Engine executes checks, such as whether Weblogic 6.0 or higher is actually installed and running, and whether the corresponding Web servers are configured to connect the Weblogic server instances. It does so by querying the remote agent on the target servers.
3. The Deployment Engine parses the modeled methods for the J2EE application. It then understands which server is the target of the deployment (server hosting the administrative console) and which command-line calls need to be made to the administrative console. It transmits the resources to the agent on the target and has that agent execute the command-line calls.
4. The Configuration Engine determines the configuration values of the J2EE application. For example, the path of the application's home directory on the WebLogic administrative console depends on the WebLogic domain. This install path is modeled as a variable. The Configuration Engine generates the value for this variable by examining the configuration state of the previously installed J2EE server instance.

It is worthwhile noting that Step 1 in the above workflow is typically executed once for each application. As a result, the application and the model is stored and version-controlled on the master server. Steps 2 and 4 are typically executed many times for many different WebLogic target servers. Also, it is very likely that different people within the operations staff execute Step 1 and Steps 2-4. No knowledge about Step 1 is required to kick off the sequence of automated Steps 2-4.

Workflow 2: Compare Two Deployed Instances of the Above J2EE Application

1. The Analysis Engine parses the analysis methods of the model for the J2EE application, which guide it to identify all the relevant configuration settings of the application in the config.xml files of both servers. It then transforms these two files into two new, smaller XML files, containing only this relevant information.

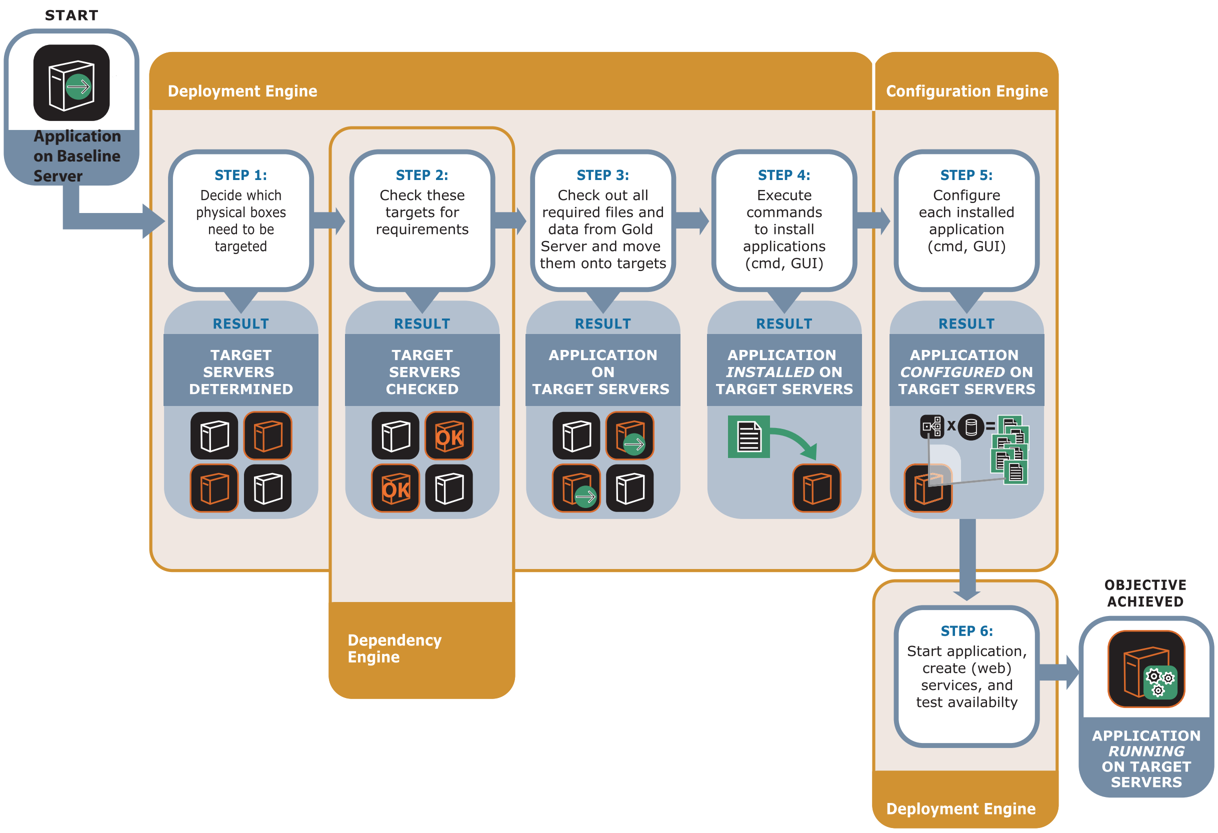

Figure 3: Application aware components can automate the configuration and deployment process from end-to-end, using appropriate application-specific knowledge for each step.

2. The Analysis Engine parses this captured information, compares the two XML files to each other and then presents any differences in a structured way to the user (e.g., the full name of the Weblogic parameter is presented with each differing configuration value).

It is worthwhile noting that the steps in the above workflow use the methods created in Step 1 of Workflow 1. The operations person does not need any knowledge about Step 1 of Workflow 1 in order to execute the above two steps.

Workflow 3: Deploy an N-Tier Web Application on Windows, Consisting of an IIS Virtual Directory, COM+ Components, and a Database

1. Guided by the user, the model builder captures the Web application on a Baseline Server:
   a. Recognizes the application as a Web application on IIS;
   b. Creates an application model of the virtual directory, capturing and describing all the relevant content, ASP pages, and ISAPI filters. Figure 5 shows the screen, which lets a user select a virtual directory from the Baseline Server machine (which in Figure 5 is called "*win_qa*").
   c. Creates an application model for the COM+ components.
   d. Creates an application model for the SQL scripts.
   e. Checks the resources of b), c) and d) into the master server's repository, where they are versioned.
   f. Captures the relevant configuration information from the IIS metabase database and the COM+ catalog and stores them in XML format.
   g. Uses predefined models for each resource type (virtual directory, COM+ component and SQL script) to add to the newly created model all the necessary steps to install and uninstall the application. Figure 6 shows the resources of the completely checked-in FMStocks application. The selection of the virtual directory in Figure 5 resulted in two resources, the virtual directory tree (*FMStocks*) containing content, ASP pages, etc. and the corresponding configuration data (*FMStocks.xml*), which is a resource containing the relevant part of the IIS metabase in XML format. Each resource is listed together with its type. As we will see in the next section, this type determines how the application model is built.
2. The Dependency Engine executes checks, such as whether IIS 5.1 is actually installed and running on the target server or a global ISAPI filter implementing application security is registered

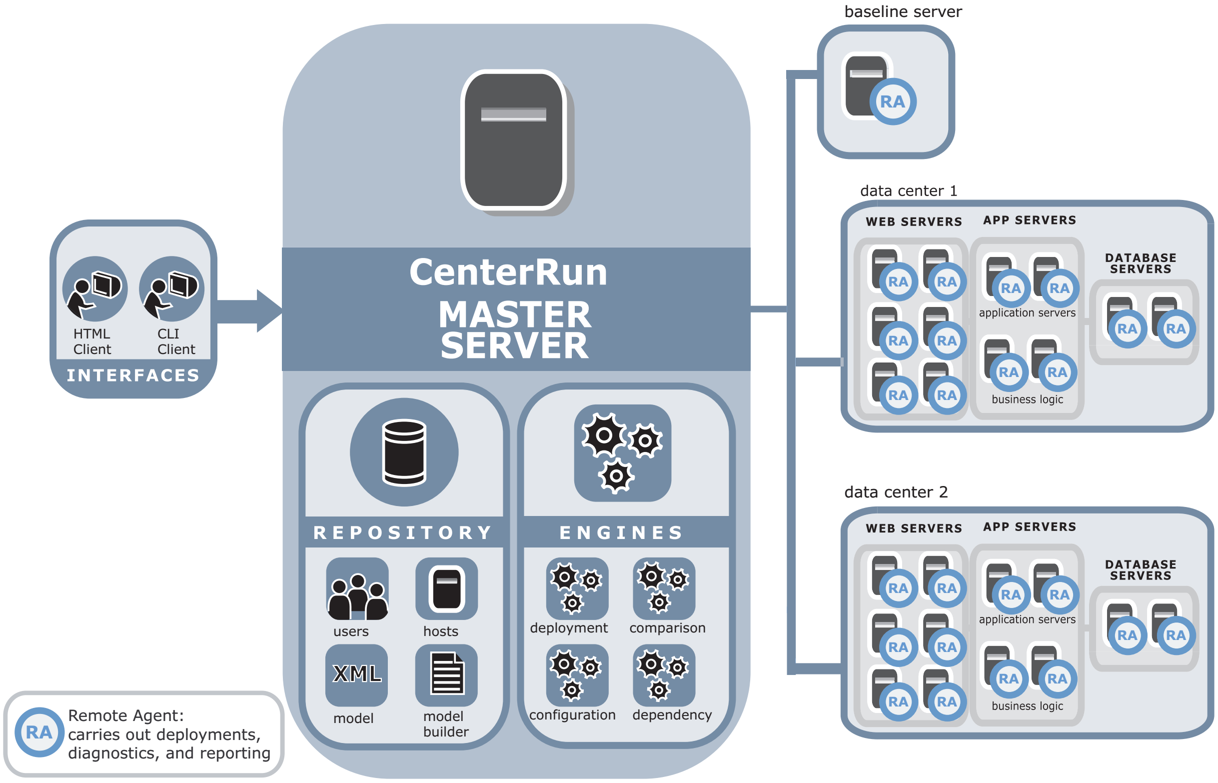

Figure 4: The Master Server (our version of a Gold Server) connects to Remote Agents in different data centers that reside on each managed server.

in the metabase. It does so by querying the remote agent on the target server.

3. The Deployment Engine parses the modeled methods for the IIS Web application. It then understands how to install the files and ISAPI filters on the target IIS server and which IIS services to shut down at the beginning and restart at the end of the deployment. It also understands how to register the COM+ components and how and on which server to run the SQL scripts. It transmits the resources to the agent on the target server and has that agent execute the installation by making calls into the ADSI API for IIS and COM+ API. The agent also runs the SQL scripts with the appropriate SQL server as target.
4. The Configuration Engine inserts all the captured configuration data from the XML files into the metabase of IIS on the target server and into the COM+ catalog. It executes the configuration by making calls into the ADSI and COM+ APIs.

Workflow 4: Analyze the Above N-Tier Web Application on Windows for "Out-of-band Changes"

1. The Analysis Engine parses the analysis methods of the model for the IIS virtual directory, which guide it to do the following: (1) identify all the relevant configuration settings of the given application in the metabase of the IIS server and then extract these settings into an XML file; (2) capture all file metadata from the relevant virtual directories and all local ISAPI filter version and configuration data of the given application.
2. The Analysis Engine parses the analysis methods of the model for the COM+ components, which guide it to query the COM+ catalog and capture the resulting configuration settings.
3. The Analysis Engine parses this captured information, compares it to the corresponding information in the master server's repository (where the information relevant at the time of the last deployment is stored), and then presents any

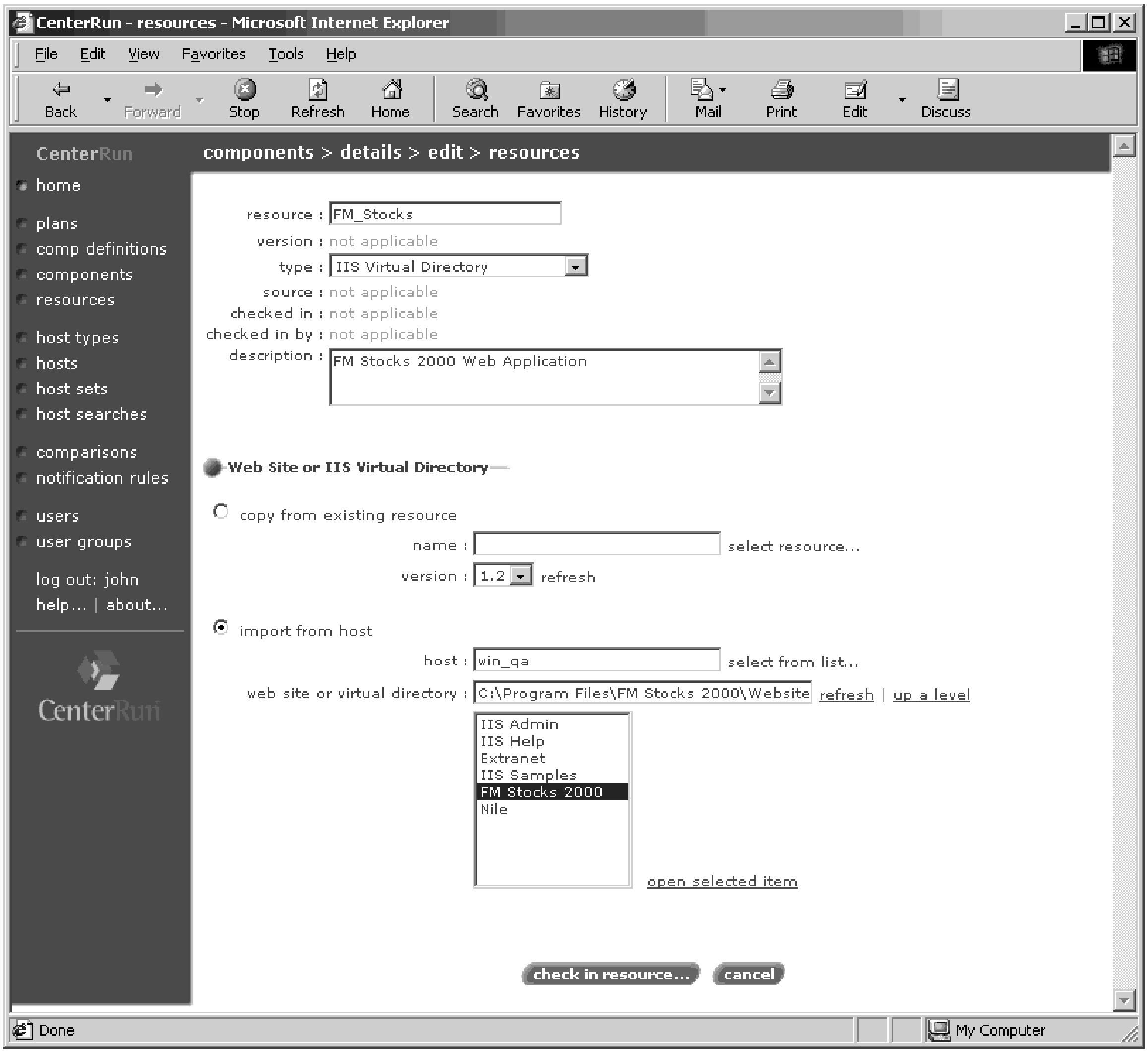

Figure 5: Capture of the IIS virtual directory for FMStocks.

differences in a structured way to the user (e.g., the full name of the metabase field is presented with each difference in the Web application configuration).

Application Aware Technology

In this section, we discuss some of the technology behind the building blocks and functions presented in the last section.

Application Model and Infrastructure

A model needs to capture all aspects of an application: the software features (directories, files, binaries, content, etc.) and the execution steps to deploy, configure, discover, and analyze the application. The model also needs to express the relationships among objects of interest, such as grouping (e.g., software features into an application, target servers into clusters) and dependencies of one application on other applications being deployed, or dependencies on OS environments on the target server. Consequently, the model is responsible for all of the capturing, storing, and manipulating of application knowledge in the system. At the same time, for a user (system administrator, operations personnel), the model is simply the means to an end, which is the automation of processes. That is why we decided that for common cases, the modeling task should be done by the system. The workflow, introduced in the previous section, allows a user to simply pick resources from a Baseline Server, group them into a modeled and checked-in application and then deploy such application with the "click of a button." In order to achieve such a workflow, we put the following infrastructure in place:

- A **Component** is a modeled application object, consisting of all the resources and methods

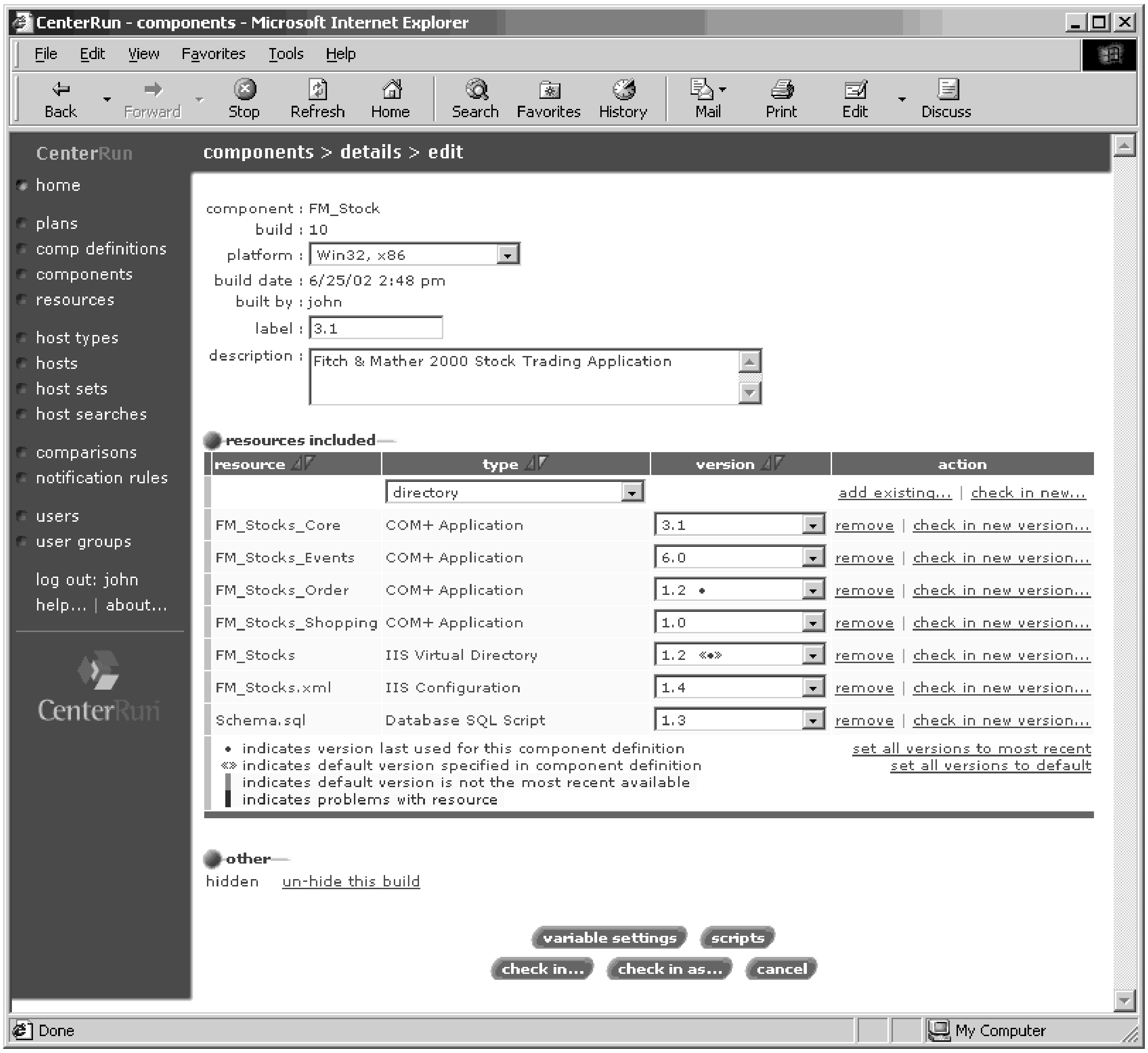

Figure 6: FMStocks application is checked-in and modeled.

(install, uninstall, discover, analyze) of an application.

- A **Resource Type** is a first class object in the system. Each time a user checks a resource into the repository, a *resource type name* (*COM_PLUS*, *WAR_WebLogic*, *IIS_Web_Site*, etc.) is associated with the resource. The resource type associates the corresponding type name with a set of behaviors for deployment, configuration, discovery, and analysis.
- A **Resource Handler** defines the actual behavior. It captures the methods to install, uninstall, discover, and analyze resources of a given type. A *resource handler* is implemented as a *component* itself. These *components* ("system components") come bundled with the system, as opposed to application components built by the user. A *system component* might have its own resources, which are implementations (scripts, Java classes) of modeled methods for the supported *resource type*. Its methods are accessible by other components being deployed onto the same target server and when calling these methods parameters can be passed. Consequently, the *resource type* really associates a *resource type name* with a *resource handler*. These *system components* are deployed onto target servers transparently to the user, so that they are available to application components at the time of a user initiated deployment or analysis.
- The **Model Language** defines the syntax in which each component is expressed. We chose XML, which is sufficiently readable so that advanced users can manipulate the model directly if needed for customization, rather than just using the model builder.

In the following we illustrate these concepts with some model fragments.

First, we show a fragment of the handler (see Listing 1) for resources of type COM+. This is a *system component*, bundled with the system and thus never has to be manipulated by the user. As depicted below, it consists of a few Windows Scripting Host scripts, which are bundled, in a resource called "ComPlusScripts." Those scripts are deployed to each target server ahead of time, transparently to the user. The fragment only shows one modeled method, which is the "install" method, responsible for installing any resource of type COM+. This method takes an input parameter (*rsrcDescription*), which determines the COM+ object for the handler. The method then invokes one of the WSH scripts (*Complus.wsf*) via the cscript command shell, passing the input parameter as argument.

Next, we show an *application component* fragment, which corresponds to the FMStocks sample application, mentioned in the previous section. The model builder generates this *component* during the workflow steps 1a-1g, as described in the previous section.

The (complete) component contains a reference to each resource selected by the user in Steps 1b-1d of the workflow. The first two resources (*FMStocks2000* and *FMStocks2000.xml*) are the result of the user selecting the virtual directory. The first resource represents all the content and ASP files, whereas the second resource represents the configuration data of the virtual directory on IIS, as it was set on the Baseline Server. The model builder transparently exports all relevant configuration data from the IIS metabase on the Baseline Server into the *FMStocks2000.xml* file. The third resource represents a COM+ resource. The model builder transparently exports the COM+ object from the Baseline Server into an MSI (Windows Installer) file.

The *installList* XML block (see Listing 2) represents the installation method. The *deployResources* tag

```
<?xml version="1.0" encoding="UTF-8" ?>
- <component name="complusHandler" description="Installs com+ objects">
   - <resourceList defaultInstallPath=":[install_path]">
        <resource installName="complus" resourceName="ComPlusScripts" />
     </resourceList>
 - - <controlList>
        - <control name="install" description="Installs a com+ object">
            - <paramList>
                 <param name="rsrcDescription" />
              </paramList>
            - <execNative>
                - <exec cmd="cscript">
                     <arg value="/Job:comPlusInstallApp" />
                     <arg value=":[install_path]\complus\ComPlus.wsf" />
                     <arg value="":[rsrcDescription]"" />
                  </exec>
              </execNative>
          </control>
      </controlList>
  </component>
```

Listing 1: Fragment of the handler for COM+ resources.

causes for each resource a call to the appropriate resource handler installation method. For the COM+ resource, this causes the installation method in the previous model fragment to be called, with parameters set to appropriate values obtained when the resource was captured. The *installList* XML block contains one step before and one step after the *deployResources* step. These steps stop and re-start IIS on the target server. They are calls to another *system component*, which models Windows services. The model builder generates these calls explicitly, rather than implicitly through *deployResources*. These two steps are not attached to the installation of a single resource, but rather represent global behavior of the deployment of an n-tier Windows application.

In the description above, we introduced our modeling infrastructure. We largely created our own solution. Modeling frameworks in the application management space do exist, among them the following two:

- **CIM**: The common information model [BSTWW00] is a hierarchical, object-oriented modeling paradigm with relationship capabilities. The core schema of CIM contains objects modeling both basic notions of system management and their relationships. There is an extension schema for applications, containing objects modeling (1) basic notions of software products, features, elements, and actions on these objects (2) their relationships. CIM has been used to model a large part of the Windows platform (the CIM derived model is called WMI) and the Solaris platform (see Solaris WBEM at [DMTF]). CIM is also being investigated for run-time application management in [KKS01].
- **JSR77** ([JSR77]): A Java Specification Request, which proposes a standard management model for exposing and accessing the management information, operations, and parameters of the Java 2 Platform, Enterprise Edition components. This proposal covers modeling the basic J2EE notions of J2EEServer, J2EEModule, EJBModule, WebModule, etc., their relationships, and event management. The proposal also discusses mappings of this (specific) model into (the more generic) CIM. JSR77 encapsulates a lot of knowledge about the J2EE world. As pointed out in [FK02], JSR77 lacks expressiveness for describing runtime entities and for representing versions and dependencies.

We found that the above models, while inspiring in many ways were not a good fit for us for an initial implementation. While CIM is a very expressive and extensible modeling framework, it comes with a relatively complex implementation and representation price. It also lacks two key ingredients for us: parameter passing and configuration management. Parameter passing appears to be required for implementing the notion of *resource handler*. Configuration management addresses the issue that the same application will be deployed with different configuration settings depending on the environment and that a model must therefore allow such values to be generated dynamically at deployment time. See the subsequent section for more details. Furthermore, it is important that the model can be conveniently represented to advanced users, who desire to customize the deployment or analysis behavior. CIM does not offer a simple solution for this.

Configuration Management

We decided that configuration management is an area in which we needed to innovate. We designed an extension to our model, which allows the inclusion of variables, and an algorithm (see also Figure 7) which runs at deploy time to instantiate these variables. We also allowed users to replace actual values with

```
<?xml version="1.0" encoding="UTF-8" ?>
<component name="fmstocks" description="FMStocks sample application">
  - -<resourceList defaultInstallPath=":[install_path]">
          <resource installName="FMStocks2000" resourceName="FMStocks2000" />
          <resource installName="FMStocks.xml" resourceName="FMStocks.xml" />
          <resource installName="FMStocks2000Core.msi"
                resourceName="FMStocks2000Core.msi" />
      </resourceList>
   - <installList>
        - <controlService actionName="stop" componentName="servicesHandler">
             - <argList>
                    <arg name="serviceName" value="IISADMIN" />
               </argList>
          </controlService>
          <deployResources />
        - <controlService actionName="start" componentName="servicesHandler">
             - <argList>
                     <arg name="serviceName" value="W3SVC" />
               </argList>
          </controlService>
      </installList>
</component>
```

Listing 2: The installation method.

variables in configuration files associated with an application. The algorithm determines the configuration data by looking at least at the following *environment characteristics*:

- The characteristics of the server machine (IP address, hostname, network connectivity, processors, etc.).
- The characteristics of the operating system of the server machine (OS type and version, patch level, etc.).
- The characteristics of other software already on the server.
- The characteristics chosen by a user (system administrator) at run-time of the deployment process.

The algorithm reads the model as its first input. Depending on the content of the model, the algorithm then determines the environment characteristics by collecting the relevant data from each target server, from models of other applications provisioned on the target server, and from user generated input. The algorithm uses the collected information to generate values for settings within configuration files of the application and to transform the methods of the model into concrete execution steps, to be executed on each server to install and deploy the application, taking into account all the characteristics collected as described above.

Let us revisit the *petstore* application introduced earlier (see Listing 3). Assume that *petstore* is to be deployed onto WebLogic 6.1. The path of petstore's home directory on the Weblogic administrative console depends on the Weblogic domain of the J2EE server, onto which petstore is deployed. Below is a fragment of an *application component* modeling this fact. The *varList* XML block contains declarations of variables used in this component. The *domain_dir* variable is used to hold the value of the home directory path. The declaration allows defining a default value of a variable. In this case, the default value is a path with a fixed prefix (/opt/bea/wlserver6.1/config) and a suffix, which refers to another variable *domain_name* in another *application component* called *WL61Server*. That component models the WebLogic server instance on the same target server and that variable contains the domain name of the server. The algorithm above instantiates the variable *domain_dir* by examining the configuration state of the installed J2EE WebLogic server.

Deployment Management

The Deployment Manager handles all aspects of the deployment of a modeled application; including efficient WAN distribution of potentially large amounts of data and executing the modeled deployment methods on each target server. This is why we decided to use a local cache (*local distributor*) on LANs, so that a single master and console can control multiple data centers. Another issue is security of the communication. However, none of these issues relate to the main theme of this paper, application awareness.

From an implementation point of view, it is worthwhile to note that JSR 88 (see [JSR88]) is a proposed standard API for the deployment of J2EE

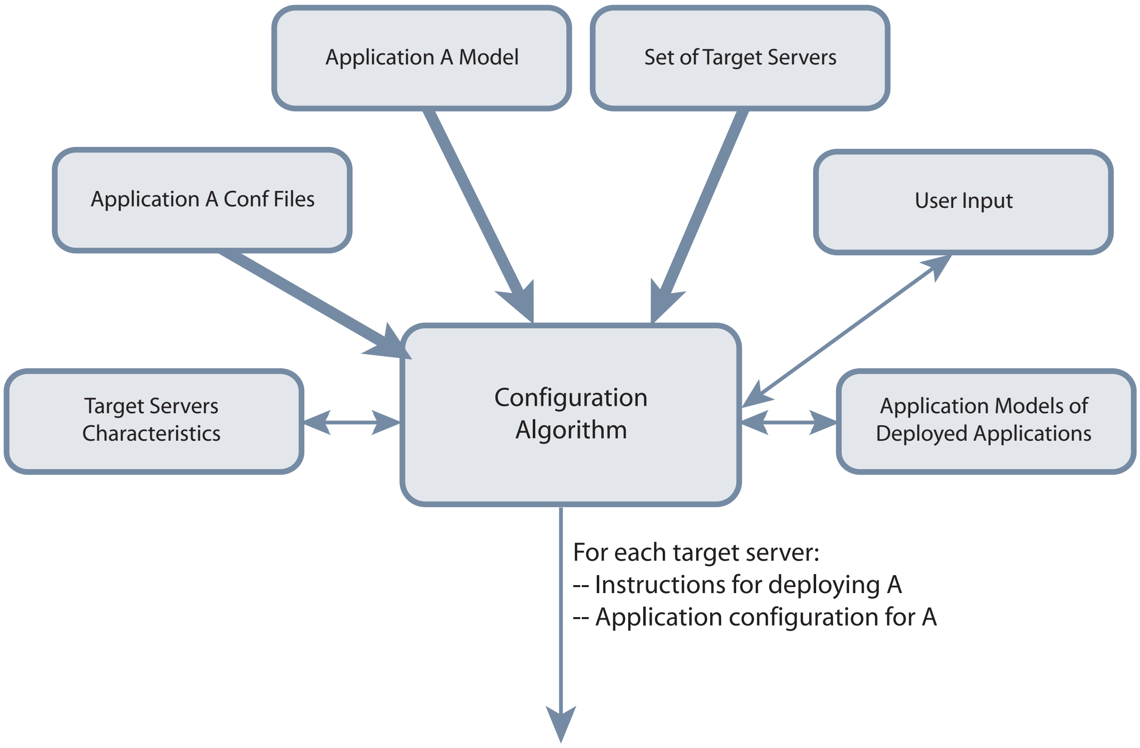

Figure 7: Configuration management algorithm.

applications. The specification aims at defining the contracts that enable solutions from multiple providers to configure and deploy applications on any J2EE platform (e.g., Weblogic, WeSphere, iPlanet, etc.). We need to do deployment across platforms (UNIX, Windows) and well beyond the scope of J2EE applications; still, standardization of this form is a welcome simplification for management tools like ours.

Dependency Management

Dependency information is expressed within our model as relationships among applications. For example, a J2EE application might only be deployable onto WebSphere V4 or newer; WebSphere itself might require the JDK x.y already installed, which in turn requires a certain patch-level of the Solaris OS. On the Windows side, an IIS-based application might require IIS V5 or newer, etc. In CIM, relationships are expressed as objects themselves. We decided to model relationships simply as object attributes, which is a simpler implementation at the cost of some flexibility (e.g., modeling a relationship both directions) and extensibility (e.g., adding a relationship without changing the object in the relationship itself).

An application can consist of several software features. These features might get installed on separate machines (Web server, app server, console for app server). Also yet another set of machines (e.g., database server) might require configuration changes. Our model captures these inter-machine dependencies in the deployment methods and their target servers. Such dependencies are a simplified form of the ones considered in [EK01]. Our approach therefore could greatly benefit from integration with a solution, such as presented in that paper.

Automated Analysis

The remote agent does the analysis of the installed application with parsers and tools, which have application knowledge. For example, in order to analyze the configuration state of an IIS / WebSphere / Weblogic application, the remote agent needs to (1) determine which values are relevant in the corresponding configuration store (metabase / centralized database / XML store) and (2) export these values to the master server. Analyzing Apache Web server configuration data requires a parser understanding the "httpd.conf" file.

We implement analysis in a very analogous fashion to deployment. Each *resource handler* has a method, which models the steps to obtain the appropriate configuration data. Analysis consequently implements the runtime environment to that model aspect, which results in the master server obtaining all the relevant data in a well-defined XML format, ready to be analyzed.

Cost Benefits

The organization depicted in the case study of the second section has been spending the following dollar amounts on application management per year:

- $450K in staff cost for writing deployment scripts, documentation on deployment methods and best practices, etc.
- $1.1M in staff cost for executing manual changes, deployments, etc. on the servers.
- $500K in staff cost for emergency response to application failures, deployment failures, etc.

→ This organization spends roughly **$2,050K on total cost of ownership** or **$17K on each of their 120 servers**.

The organization has now adopted our technology in their extranet production environment. Using our technology, they were able to automate the application deployment process from end-to-end. System reliability has improved, both due to shorter maintenance windows and due to less unplanned down time. They were able to free up resources dedicated to application deployment, lowering operating costs. Specifically, they have experienced the following benefits:

1. Drastically reduced time to deploy applications (Apache, WebSphere, and Weblogic base infrastructure and J2EE applications residing on this base infrastructure) through our automation technology (early indications show the quantitative gain to amount to an 80% reduction).
2. Eliminated most errors that typically occurred during deployment (where the remaining issues are mostly around process issues among the operations staff).
3. Significantly reduced firefighting and server rebuilds by using our analysis technology to track "out-of-band" changes (i.e., changes to installed applications made ad-hoc by operations personnel).
4. Increased number of applications supported while eliminating reliance on external consultants.

```
<varList>
     <var name="domain_dir"
          default="/opt/bea/wlserver6.1/config/:[component:WL61Server:domain_name]" />
</varList>

<resourceList defaultInstallPath=":[domain_dir]">
     <resource installName="petstore.ear" resourceName="wl61-petstore/petStore61.ear"
           installPath="applications" />
</resourceList>
```

Listing 3: Enhanced *petstore* application.

5. Leveraged reusability of the technology to consistently deploy applications across five different environments.
6. Automatically extracted application builds from Rational ClearCase into our version-controlled repository, reducing errors associated with application builds.

As a result, they could reduce the management costs to the following amounts:

- $50K in staff cost for authoring and fine-tuning models for their applications.
- $260K in staff cost for running deployments, upgrades etc.
- $190K in staff cost for using analysis tools to investigate failures, etc.

→ **Cost of ownership has dropped to roughly $500K** per year, a 75 percent drop from the "before picture," **totaling $4.2K per server**.

Conclusions

In this paper, we have presented our management approach to deployment and analysis of Web applications. We have argued that an effective solution needs to be application aware and have shown what technology makes up such a solution. Finally, we have described some quantitative implications on the total cost of ownership (TCO) of an Internet Data Center. We believe that application aware management is a new space, with a lot of potential for creating new and exciting technology and solutions.

Acknowledgements

We would like to thank Steve Traugott (Infrastructure Architect par excellence!) for many inspiring discussions. We also acknowledge Avishai Wool (of Lumeta) and Charles Beadnall (of Verisign) for feedback on an earlier draft. Alexander Keller (of IBM) and Steve Traugott were great "shepherds" whose work substantially improved this paper.

Author Information

Alain Mayer has a PhD in Computer Science from Columbia University. After over four years at Bell Labs, Lucent Technologies, he has joined the start-up world. He is currently Chief Technology Officer at CenterRun, Inc, where he guides the research and development of data center management software. He can be reached at alain@centerrun.com .

References

[B00] Burgess, M., *Principles of Network and System Administration*, Wiley, 2000.

[BSTWW00] Bumpus, W., et al., *Common Information Model (CIM)*, Wiley, 2000.

[CF02] cfengine Web site, http://www.cfengine.org .

[DMTF] *The Distributed Management Task Force*, http://www.dmtf.org .

[EK01] Ensel, C. and A. Keller, "Managing Application Service Dependencies with XML and the Resource Description Framework," *Seventh International IFIP/IEEE Symposium on Integrated Management (IM 2001)*, IEEE Press, May, 2001.

[FK02] Frey, G. and R. Kauzleben, "Configuration and Change Management of Java Components Using WBEM and JMX," JavaOne Conference Talk, 2002.

[JSR77] *The Java 2 Platform, Enterprise Edition Management Specification*, http://jcp.org/jsr/detail/077.jsp .

[JSR88] *J2EE Application Deployment Specification*, http://jcp.org/jsr/detail/088.jsp .

[KKS01] Keller, A., H. Kreger, and K. Schopmeyer, "Towards a CIM Schema for RunTime Application Management," *Proceedings of the Twelfth IFIP/IEEE International Workshop on Distributed Systems: Operations and Management (DSOM 2001)*, October, 2001.

[LH02] Limoncelli, T. and C. Hogan, *The Practice of System and Network Administration*, Addison-Wesley, 2002.

[MDEGGH00] Machiraju, V., M. Dekhil, K. Wurster, P. Garg, M. Griss, and J. Holland, "Towards Generic Application Auto-discovery," *IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2000.

[R97] Rudorfer, G., "Managing PC Operating Systems with a Revision Control System," *Eleventh Systems Administration Conference, LISA XI*, 1997.

[TH98] Traugott, S. and J. Huddleston, "Bootstrapping an Infrastructure," *Twelfth System Administration Conference, LISA XII*, 1998.