N1™ Grid Service Provisioning System 4.1

Developers Concept Guide

4150 Network Circle
Santa Clara, CA 95054
U.S.A.

August 2004
# Table of Contents

1. Why NI Grid Solutions Are Important To You ................................................. 4
   1.1. Who Should Use This Book ........................................................................ 6
2. Understanding the Problem .................................................................................. 7
3. An Architectural Overview .................................................................................. 9
   3.1. Distributed Environment .......................................................................... 10
   3.2. Understanding Components and Plans ..................................................... 12
4. Getting Started ..................................................................................................... 15
   4.1. Installation .................................................................................................. 15
   4.2. Installing on Windows Systems .................................................................. 16
      4.2.1. How to Create a Scheduled Task to Optimize the Database .............. 18
      4.2.2. Installing the Remote Agent, Local Distributor, and CLI Client ........ 18
   4.3. Additional Configuration ........................................................................... 20
   4.4. Understanding the Steps to a Solution ....................................................... 23
5. Concepts and Facilities in Action ....................................................................... 24
   5.1. File distribution: Pushing configuration files ......................................... 24
      5.1.1. Create a Component .......................................................................... 25
      5.1.2. Running the Component .................................................................... 35
   5.2. Multi-tier application distribution: Silent application installation .............. 40
      5.2.1. Directory Component ........................................................................ 41
      5.2.2. File Component .................................................................................. 43
      5.2.3. Container Component ........................................................................ 46
      5.2.4. Running the Plan ................................................................................ 52
   5.3. Orchestration of a Service .......................................................................... 54
      5.3.1. Extranet Installation Components ....................................................... 55
      5.3.2. Extranet Service Plans ...................................................................... 62
6. Advanced XML Authoring .................................................................................. 68
   6.1. Dependency checking ................................................................................... 68
   6.2. Failure identification and back-out control ............................................... 69
   6.3. Conditional execution .................................................................................. 69
   6.4. Code Encapsulation .................................................................................... 70
      6.4.1. <extends> ......................................................................................... 70
      6.4.2. <execSubplan> .................................................................................. 71
   6.5. 2-Phase Installation ..................................................................................... 71
7. Glossary of terms ................................................................................................. 73
8. Resources ............................................................................................................ 77
9. Support ................................................................................................................. 77
1. Why N1 Grid Solutions Are Important To You

N1 Grid solutions are especially important today as enterprises begin to realize the benefits of Service Oriented Architecture (SOA.) With SOA enterprises build Web and XML applications that rapidly bind to one or more services in a Web infrastructure using open standard protocols over routed networks to transmit data defined in industry accepted schemas. At first the applications access data and functions from a defined set of servers at statically defined addresses and over time the applications access data and functions dynamically from the most appropriate service provider available.

From a system administrator, executive information systems technology management, software development and quality assurance perspective SOA requires a new set of skills and expertise to deliver service excellence. The old methodology depends heavily on hand-coded installation scripts, manually applied operating system patches, and procedural component tests. This methodology is not flexible, reliable, and agile enough for enterprises to succeed in an SOA environment.

The N1 Grid product family provides products for these environments and consists of:

- **N1 Grid Engine** provides the fundamental grid functionality within the N1 Grid stack, providing workload management and dynamic resource provisioning associated with workloads and according to policies. The software distributes work across a network of servers, workstations and desktops, forming a grid.

- **N1 Grid Service Provisioning System** automates application provisioning through One-Touch installation, configuration, and updating of servers in your data center.

- **N1 Grid Provisioning Server 3.1 Blades Edition** provides a highly scalable and powerful management environment for the Sun Fire B1600 Blade Platform to rapidly design, provision, and scale blade-based server farms.

- **N1 Data Platform** consolidates multiple storage network data services under a single management umbrella and utilizes innovative technology to pool storage assets to reduce complexity and administrative costs, increases utilization, and dramatically lowers Total Cost of Ownership (TCO.)

- **N1 Grid Containers** enable creation of a virtualized environment with multiple software partitions for each single operating system instance. N1 Grid Containers make server consolidation simpler, safer and more secure.
This document focuses on the N1 Grid Service Provisioning System from a developer's perspective. To understand N1 Grid Service Provisioning System consider an enterprise's goals in a SOA environment:

- Executive management needs an efficient view of service delivery against system configuration. N1 Grid Service Provisioning System maintains an audit trail for all activities and enables comparisons of configurations in two or more systems.

- System administration needs a repeatable and automated method to address the servers in a datacenter as one system, to guarantee, manage, and audit changes to a datacenter. N1 Grid Service Provisioning System enables enterprise system administrators to work with a library of automation components that allow rapid changes to server equipment, operating systems, clusters, application servers and other enterprise software.

- Software development needs a standardized method to deliver installation, orchestration, and provisioning functions for cluster and multiple system installations, rather than single machine installers as delivered today. N1 Grid Service Provisioning System enables software vendors to guarantee complex, multi-server installations are performed correctly. This enables vendors to package installation scripts, configuration steps, and provisioning tasks into object-oriented reusable components.

- Quality assurance needs a method to guarantee that the system optimizations they recommend are put into practice.

In an SOA environment service excellence means: always-on uptime, making the services that are needed available, and rapidly changing the datacenter to meet new and often unexpected needs. These goals define an agile business' datacenter. To be this agile requires N1 Grid Service Provisioning System.

N1 Grid Service Provisioning System software provides a common framework for automated application service provisioning that uses an object-oriented XML based development framework. This document shows system administrators, software developers, and quality assurance engineers how to use and extend N1 Grid Service Provisioning System to solve everyday issues of system configuration, service provisioning, and application deployment.
1.1. Who Should Use This Book

N1 Grid Service Provisioning System introduces many new concepts, facilities and terms. Sun provides documentation, videos, professional services, and sales briefings to assist you through the learning curve. This document further explains N1 Grid Service Provisioning System by taking you from the high level concepts and shows practical and useful examples of how to put the concepts into action.

Sun provides these documents to help you understand how to use the N1 Grid Service Provisioning System:

- N1 Service Provisioning System 4.1 Installation Guide
- N1 Service Provisioning System 4.1 Release Notes
- N1 Service Provisioning System 4.1 Reference Guide
- N1 Service Provisioning System 4.1 User's Guide

Access the N1 Grid Service Provisioning System 4.1 books in the N1 product distributions and on the Web at http://docs.sun.com/db/coll/1119.1

A developer license to the N1 Grid Service Provisioning System software is available at http://www.sun.com/n1/toolkit

This document was written with the following audience in mind:

- System administrators
- Software developers
- Quality Assurance engineers

This document expects that you have basic understanding and experience with service provisioning (setting up user accounts, configuring operating system parameters, and application server installation and configuration) and have working knowledge of configuration automation, including scripting. See the Resources section for books and on-line resources to help you expand your skills.
2. Understanding the Problem

For agile enterprises, change is a competitive advantage. The driving forces for change come in many forms, including the following:

- New services change frequently to address changing customer, supplier, and partner needs.

- Capacity Planning decisions change frequently to ensure optimal infrastructure utilization rates and to eliminate performance bottlenecks.

- System and application changes frequently as new system components are added to the datacenter.

- Personnel changes require system provisioning changes, including operating system and directory account provisioning and password changes.

Many enterprises use a set of configuration utilities today to manage change. These configuration utilities include management consoles, installer utilities, scripting utilities, software updating services, application optimization and load balancing solutions. The problem using configuration utilities include the following:

- Proprietary or custom scripting language knowledge is required by system administrators and engineers. Personnel turnover means lost knowledge and problems managing change over time.

- Configuration utilities are often error prone to use and may introduce security issues. For example, some configuration utilities require disclosing administrator (i.e.- root) access to a system technician.

- Configuration utilities normally have no business intelligence to know how to change datacenter configuration to optimize underutilized equipment.

- Configuration utilities are not normally compatible with other configuration utilities. In Service Oriented Architecture environments, enterprise application functions are provisioned as orchestrations of a series of individual services within one datacenter. The configuration utility that establishes a database service must be compatible with the configuration utility that installs the Java™ DataBase Connector (JDBC) to access the database and the utility that establishes the Enterprise Java Bean that holds the business logic to access the database.

- Using disparate and multiple configuration utilities in a datacenter does not provide an overall view of changes in a datacenter to management.
• A configuration utility provides no mechanism to check a change and deployment Plan prior to actually making the change to the datacenter.

• Configuration utilities are themselves software. Changes to their functionality or the scripts that drive them need to be captured for use by team members that will join your project later, and versions of their functionality or scripts must be archived to explain changes to the subsequent engineer that provides on-going maintenance support.

N1 Grid Service Provisioning System is an appropriate solution for enterprises that encounter the following situations:

• One-To-Many situations where a single change must be frequently and rapidly made a large number of systems. Section 5.1 presents a scenario where a system administrator needs to update a file on a group of servers in a datacenter.

• One-To-One situations where a change must be repeatedly made on a target environment. Section 5.2 presents a scenario where system administrator needs to install a Web server application onto a group of servers in a datacenter.

• Orchestration situations where multiple steps must be accomplished for each change in a multi-tier and multiple-system environment. Section 5.3 presents a scenario where a system administrator needs to provision a service at several levels of service integration, including a Web server, application server, and a Web application to a group of servers in a datacenter.

In the next section you will learn how the N1 Grid Service Provisioning System architecture provides a solution to these problems.
3. An Architectural Overview

N1 Grid Service Provisioning System is an object oriented, XML-based, distributed environment to solve enterprise system configuration, service provisioning, and application deployment needs. N1 Grid Service Provisioning System 4.1 provides an extensible framework and environment that at a minimum delivers these features:

- Common framework to build service provisioning automation.
- Maintains an audit log of changes over time.
- Compares the current state of target hosts with their expected state.
- Simulates a change to identify configuration problems.
- Implements a set of rules to govern automation execution.
- Notifies system administrators of problems and actions.
- Automatically manages version control.

To accomplish these features N1 Grid Service Provisioning System implements a distributed environment where object-oriented components are authored in XML scripts and orchestrated to follow execution plans for distribution, provisioning, and installation needs.

This section explains the N1 Grid Service Provisioning System architecture at a technical level for users who expect to build provisioning automation solutions. This section begins with an introduction to the distributed environment. And, then this section introduces several new terms and presents diagrams to explain the system architecture.
3.1. Distributed Environment

N1 Grid Service Provisioning System implements a distributed environment using a Master Server/Remote Agent architecture. The architecture enables a user to drive a set of Remote Agents from a central Master Server. Figure 1 shows a top-down view of how the user directs the actions of a set of Remote Agents on one or more machines.

![Diagram of the N1 Grid Service Provisioning System architecture](image)

Figure 1 - An overview of the N1 Grid Service Provisioning System architecture.

Figure 1 shows the flow from a user's browser or secure shell client to the Master Server. In the Master Server, N1 Grid Service Provisioning System provides a directory of users, hosts, and preferences. The Master Server provides browser-based authoring of provisioning solutions and automatically archives older versions.

The N1 Grid Service Provisioning System distributed architecture provides features to facilitate development and execution of provisioning solutions. For example, N1 Grid
Service Provisioning System maintains an audit trail for all activities and enables comparisons of configurations in two or more systems. The distributed environment versions each change to a component or plan automatically. Users may toggle between versions at will.

The Remote Agent runs on a target server and responds to requests it receives over a TCP/IP network connection. Remote Agents execute commands to do the actual work of provisioning a service and making changes.

Additionally, Remote Agents provide distributed access to local system resources. For example, users may drive N1 Grid Service Provisioning System to display the file system at the remote agent to a user at the Master Server. Additionally, the Master Server may compare files to understand differences between files stored at the Master Server and the files on the Remote Agent machine.

Figure 2 illustrates an optional N1 Grid Service Provisioning System mechanism to buffer large scale deployments and deployments where Remote Agents are grouped by location and other attributes.

![Figure 2 - Implementing Local Distributor tiers for scalability.](image)

Local Distributors are an optional mechanism that facilitates making changes to groups of systems by physical location and target host type. For example, an enterprise runs a group of hosts in New York, USA and a second group in Paris, France. In this example, a Local Distributor is set-up in New York to receive all the necessary files to conduct provisioning with local Remote Agents in the New York group. Additionally, in Figure 2, the Master Server connects to a second Local Distributor in Paris. In this way, the connection bandwidth between the Master Server and Remote Agents is conserved.

The Remote Agents perform service installation tasks by utilizing a set of objects. The next section will introduce you to the Service Provisioning System object model.
3.2. Understanding Components and Plans

N1 Grid Service Provisioning System provides an object model to define a set of Components and Plans for system configuration, service provisioning, and application deployment automation. In most cases you will work with Component and Plan objects using the XML scripting system and Web graphical interfaces. A good understanding of the Component and Plan object model will help you work with Component and Plan objects. Additionally, understanding the object model will prepare you should you choose to add your own custom functionality to N1 Grid Service Provisioning System.

- Components – a logical grouping of source information that defines an application and a set of instructions specifying how to handle the source information.

- Plans – the units of action that orchestrate a group of Components to accomplish a provisioning task across a set of distributed systems.

N1 Grid Service Provisioning System defines an object model to define Components and Plans programmatically and to create a relationship between them that will be used at runtime. Figure 3 shows the relationship between Component and Plan objects.

Figure 3 - Components encapsulate the data and logic needed to make a change at a host. Plans orchestrate one or more Components to provision a service in several tiers or across a set of hosts.

N1 Grid Service Provisioning System includes dozens of useful implementations of Components and Plans and it is extensible for you to author your own objects to solve your specific configuration, provisioning, and deployment needs. For example, a Component contains a BEA WebLogic Server configuration file that must be customized at each host server to include the local host name. A second Component contains a J2EE Enterprise Archive Resource (EAR) file and the logic to identify where to place the EAR file within a WebLogic Server deployment. A Plan identifies that the Component containing the configuration file to run first, and the Component that installs the EAR file to run second. In this example, Components are responsible for making changes and Plans are responsible for orchestrating the activity of Components on a set of distributed host systems.

N1 Grid Service Provisioning System 4.1 provides a set of built-in Components to use immediately. These include Components to distribute files and directories, Components
to direct BEA WebLogic Server, and Components to provision COM objects. Section 5 shows examples of built-in Components in action.

Components and Plans provide the necessary means to define a provisioning strategy and the practical code to implement provisioning automation. Plus, the system provides software vendors with a new ability to deliver a set of N1 Grid Service Provisioning System Components and Plans for their products.

Figure 4 looks at the details of the Component class. ComponentType is an abstract class that encapsulates the basic information about the Component, including its name, description and version details.

![Component Diagram](image)

**Figure 4 - The Component class provides the needed information to manage Components in the distributed environment.**

By implementing the ComponentType abstract class methods, N1 Grid Service Provisioning System knows how to visually display, control, and run the File and Container Components. Figure 4 shows the File and Container classes inheriting the ComponentType abstract classes methods. File extends ComponentType by adding methods to install and uninstall a file and to set the user, group, and permissions values for the file. Container follows the same pattern and adds methods to reference other
Components, to define a set of variables used for parameter substitution at runtime, and methods to install and uninstall the referenced Components.

The classes in Figure 4 enable Components to be used in two ways. Automation solutions may use any existing Component methods to accomplish their provisioning tasks. When a method does not exist the user may author a new class that implements the ComponentType abstract class and implement the needed method.

![Diagram](image)

Figure 5 - Classes that inherit from Plan implement methods to run a set of Components in a specified order.

Figure 5 shows that Plan is an abstract class that encapsulates a set of Components to run and the order in which they run. Plan enables users to automate multiple-tier strategies where each step in a Plan reacts to install and uninstall dependencies, processes variable substitution, and handles exceptional conditions. For example, a Plan checks to see that an application has already been installed on a target host and if the application is not installed it runs a Component that installs the application.

For many common tasks N1 Grid Service Provisioning System provides automatically generated Components and Plans, however, at times you may need to manually change a Component or Plan. Section 5 of this document provides an example and introduction to N1 Grid Service Provisioning System's XML authoring.
4. Getting Started

By reading the previous sections of this document you learned the high-level concepts and objectives of the N1 Grid Service Provisioning System. This section shows how to prepare N1 Grid Service Provisioning System to build and operate Components and Plans that solve the example scenarios presented in section 5.

4.1. Installation

Sun offers developers a free license for the N1 Grid Service Provisioning System. Download the software at http://www.sun.com/software/solutions/n1/toolkit/

The N1 Grid Service Provisioning System comes with a straightforward installation utility to facilitate installation. The software and installation scripts include the following parts:

1. **Master Server** - provides Component authoring tools and deployment capabilities to stage an N1 Grid Service Provisioning System orchestrated deployment of software to a set of Remote Agents.

2. **Local Distributor** - provides an optional middle-tier between the Master Server and Remote Agents to provide a buffer for large scale deployments and deployments where Remote Agents are grouped by location and other attributes.

3. **Remote Agent** - provides business logic and implementation code to receive a deployment and take action on a local machine.

4. **CLI Client** – provides a communication path to the Master Server to enable the execution of commands from local and remote systems.

The examples presented in this paper use the Master Server and Remote Agent. (See the *N1 Service Provisioning System Installation Guide* for instructions on installing a Local Distributor.)

The N1 Grid Service Provisioning System version 4.1 supports the following operating environments: Solaris OS, Red Hat Linux, IBM AIX, and Windows 2000 Server/Advanced Server. A comprehensive guide to system compatibility is available in the *N1 Service Provisioning System Installation Guide* found at http://docs.sun.com/db/coll/1119.1

The following section instructs you how to install the Master Server and Remote Agent on a Windows 2000 Server/Advanced Server.
4.2. Installing on Windows Systems

This chapter describes the steps to install the N1 Grid Service Provisioning System 4.1 on systems running Windows. You will install each of the applications separately by using the appropriate installation script on the product media. The installation scripts for each of the N1 Grid Service Provisioning System 4.1 applications begin by performing the same set of preparatory tasks and asking the same questions about directories and files. Each script then asks specific configuration questions about the application that it will install.

This section discusses the following topics:

• Installing the Master Server
• Installing the Remote Agent, Local Distributor, and CLI Client

Before You Begin

Review the following tasks. Complete any necessary tasks prior to installing the Master Server.

Steps

1. Insert the N1 Grid Service Provisioning System 4.1: IBM AIX, Redhat Linux, Windows 2000 Server/Advanced Server CD.

2. Use the Windows File Manager or a DOS window to access the windows directory on the CD.

3. Start the installation script for the Master Server.
   - If you are using the File Manager, double-click the cr_ms_win32_4.1.msi file.
   - If you are in a DOS window, type the name of the installation file at the prompt.

   E:\N1SPS4.1_2\windows> cr_ms_win32_4.1.msi

4. Answer the configuration questions when prompted by the installation program.

   The installation program prompts you to answer a series of configuration questions and then displays the Ready to Install screen.
5. Click Install to begin the installation.

The installation program installs the program files. When it completes, the installation program prompts you to restart the machine.

6. Restart the machine to complete the installation.

You must restart the machine to complete the installation of the N1 Grid Service Provisioning System 4.1.

7. Log in to the system.

After you log in, the installation program displays a Welcome screen.

8. Click Next to complete the installation.

---

**Note –**

The installer opens DOS windows and executes commands. Some of the commands might take several minutes to run. Do not close the DOS windows or cancel the operations. The operations complete automatically after a few minutes.

---

9. Click Finish to exit the installation program.

The Master Server is installed. Access the Master Server by using your web browser and the Web interface address you specified during the installation.

10. (Optional) Create a scheduled task to optimize the database.

To optimize the performance of your database, create a scheduled task that runs the `vacuumdb` utility daily. To create the scheduled task, follow the instructions in the next section.
4.2.1. **How to Create a Scheduled Task to Optimize the Database**

*Steps*

1. Open the Windows 2000 Scheduled Tasks Folder.
   
   You can open the Scheduled tasks folder by clicking the Start menu, then clicking Programs -> Accessories -> System Tools -> Scheduled Tasks.

2. To create a new task, right click in the folder and select New -> Scheduled Task.
3. Name the task.
4. Double click on the task to edit it.
5. In the Run field, type the following command on a single line:

   ```bash
   bash -c "/cygdrive/c/Program Files/N1 Service Provisioning System/4.1/server/bin/roxdbcmd vacuumdb -h localhost -a -z"
   ```

6. `c/Program Files/N1 Service Provisioning System/4.1` is the directory in which you installed the Master Server.

7. In the Schedule tab, configure the task to run once a day.

4.2.2. **Installing the Remote Agent, Local Distributor, and CLI Client**

Review the following steps. Complete any necessary tasks prior to installing the Remote Agent, Local Distributor, or CLI Client.

*Steps*

1. Insert the N1 Service Provisioning System 4.1: IBM AIX, Redhat Linux, Windows 2000 Server/Advanced Server CD.

2. Use the Windows File Manager or a DOS window to access the `windows` directory on the CD.
3. Start the installation script for the application you want to install.
   - If you are using the File Manager, double-click the `cr_app_win32_4.1.msi` file.
   - If you are in a DOS window, type the name of the installation file at the prompt.

```
E:\N1SPS4.1_2\windows> cr_app_win32_4.1.msi
```

4. `app` is one of the following values:
   - ra – installs the Remote Agent
   - ld – installs the Local Distributor
   - cli – installs the CLI Client

5. Answer the configuration questions when prompted by the installation program.
   The installation program prompts you to answer a series of configuration questions and then displays the Ready to Install screen.

6. Click Install to begin the installation.
   The installation program installs the program files.

7. Click Finish to exit the installation program.
4.3. Additional Configuration

With the Master Server installed on a Windows server, follow these instructions:

1. Point your browser to http://localhost:8080/admin to begin using the Web-based user interface to finish the additional configuration needed to use N1 Service Provisioning System.

2. At the sign-in page, sign-in using the default admin account. The password is admin too.

Figure 6 illustrates the browser interface of the main menu.

Follow these steps:

3. Click the Hosts link, and then the Details link.
4. Identify the host running the installed Remote Agent by entering a name for the host in the host edit field.
5. Select ra as the host type, and click the create link.
Figure 7 shows the hosts list after you populate a number of host machines.

![Image of hosts list](image.png)

**Figure 7 - Identifying Remote Agents for the Master Server.**

To conclude the N1 configuration, follow these steps:

6. Click the Users link, and then click Details.
7. Enter the appropriate values to create a new user account.
Figure 8 shows the browser interface with the fields filled-in.

This section showed you how to install the N1 Grid Service Provisioning System. Next you will learn an overview of the process to build and operate Components and Plans.
4.4. **Understanding the Steps to a Solution**

In the previous section you learned how to use the N1 Grid Service Provisioning System installation utilities to install the Master Server, a Remote Agent, and to configure the Master Server to recognize the target hosts. Figure 9 shows the steps you will take in Section 5 to create a series of N1 Grid Service Provisioning System solutions to solve real world problems.

![Diagram](image)

**Figure 9 - The steps needed to build and deploy an N1 Grid Service Provisioning System solution.**

N1 Grid Service Provisioning System provides the user interface, object model, and distributed environment to build system configuration, service provisioning, and application deployment solutions. The use case begins with the N1 Grid Service Provisioning System user building a set of Components. Each Component needs to be named, its type defined, and the source files and directories identified, and any custom tasks defined. From there the user creates a Plan to direct the deployment of the Components. Plans identify the Components to run, the order in which the Components run, set one or more variables that are input to the Components, and identify a set of target hosts that run the Remote Agent software.

The N1 Grid Service Provisioning System browser user interface provides many features to make navigation in and around Components, Plans, and configuration screens easy. The user interface expects you will continually add new Components and Plans so it provides features to let you navigate among hierarchies of Component groups. Groups of Components are grouped in directories.

Next you will see how this all gets put into action.
5. Concepts and Facilities in Action

Earlier sections of this document described goals, technology, and use cases for N1 Grid Service Provisioning System. This section moves from theory and practice. This section presents a series of scenarios to provide an immediate context to the many issues N1 Grid Service Provisioning System solves:

- One-To-Many situations where a single change must be frequently and rapidly made on multiple systems. Section 5.1 presents a scenario where a system administrator needs to update a file on a group of servers in a datacenter.

- One-To-One situations where a change must be repeatedly made on a target environment. Section 5.2 presents a scenario where system administrator needs to install a Web server application onto a group of servers in a datacenter.

- Orchestration situations where multiple steps must be accomplished for each change in a multi-tier and multiple-system environment. Section 5.3 presents a scenario where a system administrator needs to provision a service at several levels of service integration, including a Web server, application server, and a Web application to a group of servers in a datacenter.

Each topic in this section covers an objective of using N1 Grid Service Provisioning System, shows example Components and Plans to accomplish a goal, and shows how to do deployments.

5.1. File distribution: Pushing configuration files

The File Distribution scenario will show you a one-to-many situation. You will learn how to use N1 Grid Service Provisioning System to distribute a Unix configuration file to a set of target host machines. In this task you create a Component that processes a set of steps.

Consider a system administrator at a medium-sized company that recently hired 5 new network technicians. The network technicians will provide maintenance support on 20 Unix-based servers in the company data center using special administrative console software from the command shell on each server. For the network technicians to begin, the command shell needs to have a special path variable defined. In this example, an N1 Grid Service Provisioning System Component distributes a new .cshrc file to the servers.

The .cshrc file customizes work in the C shell (csh) and Tenex-extended C shell (tcsh) everytime csh or tcsh are invoked. Our special .cshrc file will modify the path of the local server.

To begin, the administrator builds a .cshrc file that contains the path information.
This .cshrc file adds a path to a special administrative console named masterservice for the five new network administrators. A Component will encapsulate the .cshrc file and administer its deployment to the target host server. To distribute the .cshrc file to the target servers requires the following tasks:

- Create a Component
- Run the auto-generated Plan

5.1.1. Create a Component

In this step you will create a new Component that will encapsulate the .cshrc file to be distributed. These steps will also create a deployment Plan to move the .cshrc file to the target host and keep an audit trail of the task so you can also rollback the changes.

The browser user interface provides a thorough and efficiently organized layout of controls to create Components. A Component is a combination of parameters, settings, script commands, and auditable history. Components either operate a step, such as copying the .cshrc file to a directory on the target machine, or contain other Components.

To begin you will learn to use the N1 Grid Service Provisioning System browser interface to create a Component. Figure 10 shows the first step to create a Component.
With the N1 Grid Service Provisioning System browser user interface opens, follow these steps:

1) Navigate to the Master Server and sign-in using the default admin account and admin password.
2) Click the Components link.
3) Enter the new Component's name. In this case enter: /files/.cshrc.
4) Select the File type from the type drop-down menu.
5) Optionally enter a description.
6) Click the create link.

These steps create a new Component in an internal database and move the Component into a state for you to enter additional details on the Component's functionality. Figure 11 shows the interface results of the steps taken so far.
Figure 11 shows the additional steps that are necessary to complete the new Component. Since this is a file Component type, the additional steps come down to identification of the .cshrc file. Other Component types require more information.

7) If you already know the host name and path to the needed file, enter those values in the appropriate fields shown in Figure 11. Otherwise, please use the Import File section.
The Import File portion of the Component view behaves like a file browser. The visual controls enable you to view files on the local machine and any available machine with a Remote Agent installed and running. Use the file browser function to defining the source host machine.

8) Click on "Select from list" and the pop-up dialog box shown in Figure 12 appears.

![Figure 12 - The import file section of the Component lets you identify the host that holds the source file to be pushed.](image)

The import file selection dialog appears in Figure 12 to define which host contains the file to be distributed in the new Component.

9) Click on a host and click "return selected host to main window."

The view displayed in Figure 13 shows the import file host name defined and the top-level directory listing.
10) Navigate to the desired file, select it, and click "check-in selected item…"

The browser-based file navigation file appears in other parts of N1 Grid Service Provisioning System, including visual controls for selecting files, directories, and applications. This is a browser-based version of what you might find at a shell prompt level of an operating system. The refresh, select from list, and up-a-level commands help you navigate through directories and files.
N1 Grid Service Provisioning System provides automatic version control of Components and Plans. N1 Grid Service Provisioning System asks you to confirm your intention before creating the Component as shown in Figure 14.

11) Click "Continue to check-in."

The check-in process usually takes a few seconds. So creating a Component initiates several internal tasks. During that time N1 Grid Service Provisioning System displays a progress bar as shown in Figure 15.

Once the new Component is checked-in, the Component detail view appears as shown in Figure 16.
The Component detail view includes data that N1 Grid Service Provisioning System automatically added when it checked-in the Component, including the following:

- **Versioning of the Component.** Each change to a Component is automatically archived into a version that may be recalled later to understand the changes.
• Type. Facilitates describing and grouping Components by function. For example, N1 Grid Service Provisioning System comes with a File type to describe Components that operate on files, as opposed to applications, operating system parameters, or database files. The File Component type knows how to copy, move and delete files and set file privileges.

• Physical or Virtual Host. The Physical Host is the machine the deployment targets. A virtual host is an endpoint that enables deployments to run concurrently and is useful for application servers that are deploying applications packaged in Enterprise Application Resources (EAR) files, and Web Application Resource (WAR) files.

• Configuration Template. Determines the rules of variable substitution when N1 Grid Service Provisioning System deploys a Component according to a Plan. Substitution is a powerful method for programming Components to vary their actions based on the deployment characteristics and local machine settings. For example, a substitution parameter such as :[Component.hostname] is replaced with the Component's definition for the hostname at deployment.

Figure 17 shows a view of the lower portion of the Component detail view, including the Component procedures section to run a Component using an automatically generated Plan. Click "default: install run" to see the Component in action.

Below the Component procedures is a set of controls to edit, delete, create a new version, and other options.

Figure 17 - The Component view provides several commands to extend and change the Component.
The browser interface makes a good effort to shield users from the relative complexity of working directly with XML Component definitions. In the example of creating a Component to distribute the `.cshrc` file, N1 Grid Service Provisioning System automatically created the XML Component definition. It may be viewed and manually edited by clicking "advanced edit." Figure 18 shows how N1 Grid Service Provisioning System presents the XML definition for the Component.

![XML Component Definition](image)

**Figure 18 - The Advanced Edit feature displays the XML definition of the Component.**

The file Component type XML declaration is straightforward. The XML defines the name of the Component, the Component type, the name of the file to distribute, and the path to where the file was found. The following is the full XML definition and then a detailed explanation of the XML elements used.
Next you will learn how each element in the XML definition of the Component works.

The `<Component>` element defines the Component name and description and the XML schema definition this XML description uses. A detailed explanation of the XML schema is found in the *N1 Service Provisioning System Reference Guide*.

The `<extends>` element defines the Component type. The "file" Component type requires no additional initialization parameters. Other more complex types use extends to pass in context parameters that initialize the object.

The `<varList>` element holds a list of variables that will be used in this Component. The `<var>` element named installName identifies the name of the .cshrc file when it is copied into the destination directory by the Remote Agent.
The `<resourceRef>` element defines the file and path where the .cshrc file was sourced.

N1 Grid Service Provisioning System comes with dozens of ready-to-use Component type objects like the "file" object you just viewed. The browser interface to the Master Server provides visual controls that create and manipulate XML Component definitions automatically. XML Component definitions are powerful when a custom installation task is needed. You will then see an example of this in the next section.

In this section you created a new Component that encapsulates the .cshrc file to be distributed. These steps automatically created install and uninstall procedures to move the cshrc file to the target host and keep an audit trail of the task so you may also rollback the changes. Next you will run the Component to see N1 Grid Service Provisioning System in action.

### 5.1.2. Running the Component

In the previous section you built a new Component. Many of the Component types automatically build deployment procedures. For example, Figure 19 shows the default install and uninstall procedures that N1 Grid Service Provisioning System created when it checked-in the cshrc Component. Running the install procedure deploys the cshrc file to a Remote Agent.

<table>
<thead>
<tr>
<th>procedure</th>
<th>actions</th>
</tr>
</thead>
<tbody>
<tr>
<td>default: install</td>
<td>run, run history</td>
</tr>
<tr>
<td>default: uninstall</td>
<td>run, run history</td>
</tr>
</tbody>
</table>

*Figure 19 - N1 Grid Service Provisioning System builds an install and uninstall procedures for us automatically.*

1) Click the run link next to the default: install procedure.

N1 Grid Service Provisioning System will display a deployment preparation view of the Component as shown in Figure 20.
Figure 20 - Prior to running Plan, N1 Grid Service Provisioning System asks deployment questions, including the location of the destination Remote Agents.

The .cshrc Component needs to identify the target Remote Agents. For the purpose of this example you will select a single host and then run the Plan.

2) Click "Select from list" in the Plan Procedures section.

A browser dialogue window will appear as shown in Figure 21.
Figure 21 - A list of hosts that you previously identified in the Hosts link.

The hosts view shown in Figure 21 contains a list of hosts, including a list of previously identified hosts from past deployments.

3) Choose one or more hosts.

4) Click "Return selected hosts to main window.

5) Click "Run-Plan (includes preflight)" and away you go!
N1 Grid Service Provisioning System shows a progress view as it marshals the resources needed to distribute the .cshrc file as shown in Figure 22. In a few seconds N1 Grid Service Provisioning System begins distributing the cshrc Component to the Remote Agent.
The cshrc Component finishes distributing the .cshrc file. The view in Figure 23 confirms the procedure finished successfully. The Remote Agent updated the .cshrc file. You may confirm that on the target host.

In this example scenario you created a new Component that encapsulates the .cshrc file to be distributed and the procedure to move the .cshrc file to the target host. The system keeps an audit trail of the task so you can also rollback the changes. Next you will show a more advanced N1 Grid Service Provisioning System installation that builds on what you have learned so far.
5.2. Multi-tier application distribution: Silent application installation

The previous scenario builds a Component to distribute a file to a target host running a Remote Agent in a one-to-many scenario. The scenario used the file Component type that is programmed to know how to copy a file to a target directory and to set the new file's properties (owner, group, and read/write/execute privileges.)

N1 Grid Service Provisioning System 4.1 comes with dozens of ready-to-use Component Types. Details on the N1 Grid Service Provisioning System built-in Component Types are found in the N1 Service Provisioning System Reference Guide. For example, a Windows COM Component type knows how to install and configure COM objects in Windows. In this scenario you will conduct a silent installation of the Sun Java System Web Server, formerly Sun One Web Server and iPlanet Web Server, using these Component types:

- Container – an object that holds one or more Components.
- File – as demonstrated in the previous scenario, the file object knows how to copy, move and delete files and set file privileges.
- Directory – similar to the file ComponentType used in the previous scenario, the directory object knows how to copy, move, and delete directories, and to set directory privileges.

Additionally, this scenario will show how to use an XML element, <execNative>, that knows how to run an application or shell script command.

In this section you will build a one-to-one solution. You will learn how to construct reusable components that may be applied to provisioning a service to multiple environments. For example, after using N1 Grid Service Provisioning System to deploy a Web Server in a test environment, the same Components and Plans may be used to install the Web Server in the Production environment.

This scenario presents a system administrator's need to install Sun Java System Web Server on a group of machines. Sun Java System Web Server comes with an installation utility that reads from a configuration file to determine several parameters, including the host name, installation path, administrative id and password, port numbers to use, and path to the html document root. Using a deployment automation solution the administrator avoids spending countless hours repeating the Sun Java System Web Server installation and configuration process. It allows the Administrator to capture the lessons learnt from previous installation experiences into the component type by way of additional dependencies in their environment.
The deployment comes in these procedures:

1) Create a Component to move the Sun Java System Web Server installation files to the target host.

2) Create a file ComponentType that configures the installer file to the target host.

3) Run the Sun Java System Web Server installer.

4) Start the Sun Java System Web Server

5.2.1. Directory Component

Sun distributes the Sun Java System Web Server (Sun Java System Web Server) installation files in a single directory that contains an installer script, an installer configuration parameters file, and the object files for the Web server. Your first task is to check-in the Sun Java System Web Server installation files into a directory Component. Creating a new directory Component should be familiar to you from the previous scenario when you created a file Component.
From the N1 Grid Service Provisioning System main menu follow these steps:

- Click the Components link
- Enter s1ws_install_dir for the name
- Choose the Directory type
- Enter your own description
• Click the create link.

After confirming the check-in, you will see a Component view shown in Figure 24. The view displays the elements of the Component, including the name, source directory, variables, and procedures.

You first saw this view in the previous scenario. The browser interface to the Master Server provides visual controls that create and manipulate XML Component definitions automatically. XML Component definitions provide customization to existing Component type objects and when implementing a task not offered by built-in Component type. Click "Advanced edit" in the Component view and optionally view and edit the XML definition for the directory Component.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by CR -->
<Component xmlns="http://www.sun.com/schema/CR"
  name="s1ws_install_dir" version="4.0"
  description="distribution of s1webserver"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  path="/Components/applications/s1webserver/subcomps"
  <extends>
    <type name="directory"></type>
  </extends>
  <varList>
    <var name="installName" default="Webserver"></var>
  </varList>
  <resourceRef>
    <resource
      name="/Components/applications/s1webserver/subcomps/
      /Components/applications/s1webserver/subcomps/s1ws_install_dir"
      version="1.0"></resource>
  </resourceRef>
</Component>
```

This Component XML definition identifies a directory containing the Sun Java System Web Server installation files. One of the contained files, install.inf, must be configured prior to running the Sun Java System Web Server installation utility. For this task you use a file Component type that is described in the next section.

### 5.2.2. File Component

Among the Sun Java System Web Server installation files is a special configuration file that instructs the installation utility on several parameters to conduct the installation. For example, the install.inf file identifies the location of the HTML root directory, the path to Java, and many other parameters. Follow these steps to create a file ComponentType:

• Click the Components link
- Enter install.inf for the name
- Choose the File type
- Enter your own description
- Click the create link

After confirming the check-in, you will see a Component view that is shown in Figure 25. The view displays the elements of the Component, including the name, source directory, variables, and procedures.

Figure 25 - The file Component type will use variable substitution to configure the parameters in the install.inf file.
This Component operates on the install.inf file by using the file Component type’s variable substitution mechanism. The Component variables section of the view shown in Figure 25 includes parameters that are substituted at runtime.

For example, the admin_sys_user parameter identifies the Sun Java System Web Server administrative user account. When this Component runs it substitutes the value for admin_sys_user in the install.inf value with the local administration user account.

Viewing the XML definition for the File Component type shows several variables.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by CR -->
<component xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" name="install.inf" version="4.0"
component.xsd" xmlns="http://www.sun.com/schema/CR"
path="/components/applications/s1webserver/subcomps">

    <extends>
        <type name="file"/>
    </extends>

    <varList>
        <var name="installName" default="install.inf"></var>
        <var name="admin_sys_user" default=":[container:admin_sys_user]"></var>
        <var name="admin_name" default=":[container:admin_name]"></var>
        <var name="full_machine_name" default=":[container:full_machine_name]"></var>
        <var name="suitespot_userid" default=":[container:suitespot_userid]"></var>
        <var name="admin_port" default=":[container:admin_port]"></var>
        <var name="admin_password" default=":[container:admin_password]"></var>
        <var name="jdk_dir" default=":[container:jdk_dir]"></var>
        <var name="http_doc_root" default=":[container:http_doc_root]"></var>
        <var name="http_port" default=":[container:http_port]"></var>
        <var name="server_root" default=":[container:server_root]"></var>
        <var name="suitespot_group" default=":[container:suitespot_group]"></var>
        <var name="start_on_boot" default=":[container:start_on_boot]"></var>
    </varList>

    <resourceRef>
        <resource name="/components/applications/s1webserver/subcomps/install.inf"
version="1.1"/></resourceRef>

</component>
```
N1 Grid Service Provisioning System automatically creates install and uninstall procedures for this Component. However, we now have more than one Component to install. Your goal is to update the install.inf file, move the Sun Java System Web Server installation directory to the target, and run the Sun Java System Web Server installation utility. To accomplish this installation you need a container Component type to orchestrate these tasks.

5.2.3. Container Component

In a service oriented world orchestration of several applications to deliver a service is essential. The N1 Grid Service Provisioning System provides orchestration features that are highlight in the container Component type that is presented in this section.

The container Component you are about to build accomplishes these steps:

1) Customize the Web server installer configuration file with parameters that are specific to the each target host

2) Copy the Sun Java System Web Server installation files to the Remote Agent

3) Run the Sun Java System Web Server installer script

4) Start the Web server

To begin, create a new container Component in N1 Grid Service Provisioning System:

- From the N1 Grid Service Provisioning System main menu click the Components link.
- Enter s1webserver_install for the name.
- Choose the Container type.
- Enter your own description.
- Click the create link.

After confirming the check-in, you will see a Component view that is shown in Figure 26.
The container Component offers these features:

- The container Component customizes the Sun Java System Web Server installer configuration file using Component variables and variable substitution.

- The container Component uses custom XML commands to run the Sun Java System Web Server installer and later to start the Web server service.

Viewing the XML definition of the container Component provides a good understanding of how these two areas are implemented in N1 Grid Service Provisioning System. Click "Advanced edit" to view the XML definition. Here is the entire XML definition and followed-by a detailed explanation of each element.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by CR -->
<Component platform="Solaris - any version"
    xmlns="http://www.sun.com/schema/CR"
    name="s1webserver_install" version="4.0">
```
<extends>
   <type name="container"></type>
</extends>

<varList>
   <var name="installPath"
       default=":[target:install_dir]"></var>
   <var name="server_root"
       default=":[target:server_root]"></var>
   <var name="suitespot_userid"
       default=":[target:suitespot_userid]"></var>
   <var name="suitespot_group"
       default=":[target:suitespot_groupid]"></var>
   <var name="admin_sys_user"
       default=":[target:admin_sys_user]"></var>
   <var name="admin_name"
       default=":[target:admin_name]"></var>
   <var name="admin_password"
       default=":[session:slws_admpw]"></var>
   <var name="full_machine_name"
       default=":[target:full_machine_name]"></var>
   <var name="http_port"
       default=":[target:http_port]"></var>
   <var name="admin_port"
       default=":[target:admin_port]"></var>
   <var name="jdk_dir"
       default=":[target:jdk_dir]"></var>
   <var name="http_doc_root"
       default=":[target:http_doc_root]"></var>
   <var name="start_on_boot"
       default=":[target:start_on_boot]"></var>
</varList>

<ComponentRefList>
   <ComponentRef name="ref1">
      <Component name="s1ws_install_dir"
         path="/Components/applications/slwebserver/subcomps"
         version="1.1"></Component>
   </ComponentRef>
   <ComponentRef name="ref2">
      <Component name="install.inf"
         path="/Components/applications/slwebserver/subcomps"
         version="1.0"></Component>
   </ComponentRef>
</ComponentRefList>

<installList>
   <installSteps name="markOnly"></installSteps>
</installList>

<uninstallList>
   <uninstallSteps name="markOnly"></uninstallSteps>
</uninstallList>

<controlList>
   <control name="run_install">
      <execNative>
         <inputText><![CDATA[
To understand the XML definition for the container Component type in more depth, the following gives a detailed explanation of each element.

```
<Component platform="Solaris - any version"
xmlns="http://www.sun.com/schema/CR"
name="slwebserver_install" version="4.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
path="/Components/applications/s1webserver"
Component.xsd">

The <Component> element defines the Component name and description and the XML schema definition this XML description uses. A detailed explanation of the XML schema is found in the N1 Service Provisioning System Reference Guide.

<extends>
  <type name="container"></type>
</extends>

The <extends> element defines the Component type. The container Component type requires no additional initialization parameters. Other more complex types use extends to pass in context parameters for the object that establish the object.

<varList>
  <var name="installPath"
    default=":[target:install_dir]"></var>
  <var name="server_root"
    default=":[target:server_root]"></var>
  <var name="suitespot_userid"
    default=":[target:suitespot_userid]"></var>
  <var name="suitespot_group"
    default=":[target:suitespot_groupid]"></var>
  <var name="admin_sys_user"
    default=":[target:admin_sys_user]"></var>
  <var name="admin_name"
    default=":[target:admin_name]"></var>
  <var name="admin_password"
    default=":[session:slws_admpw]"></var>
  <var name="full_machine_name"
    default=":[target:full_machine_name]"></var>
  <var name="http_port"
    default=":[target:http_port]"></var>
  <var name="admin_port"
    default=":[target:admin_port]"></var>
  <var name="jdk_dir"
```
The `<varList>` element identifies values that will be used later in the XML definition and when the Component is running on the Remote Agent. The `<varList>` element may also use values available from the N1 Grid Service Provisioning System environment. For example, the `:[target:install_dir]` references the installation directory on the target host. This value is dynamic as its value will only be known when a Plan deploys the Component to a host.

```xml
<ComponentRefList>
  <ComponentRef name="ref1">
    <Component name="slws_install_dir"
      path="/Components/applications/s1webserver/subcomps"
      version="1.1"></Component>
  </ComponentRef>
  <ComponentRef name="ref2">
    <Component name="install.inf"
      path="/Components/applications/s1webserver/subcomps"
      version="1.0"></Component>
  </ComponentRef>
</ComponentRefList>
```

The `<ComponentRefList>` element identifies the Components that are part of this Component. This lets the Remote Agent dynamically identify and use Components when the deployment is running.

```xml
<installList>
  <installSteps name="markOnly"></installSteps>
</installList>
```

The `<installList>` element defines a list of install step elements. Each element provides a different way to install this Component. For example, a Component could check to see if a custom Java DataBase Connector (JDBC) driver needs to be installed and provide a custom installation step.

```xml
<controlList>
  <control name="run_install">
    <execNative>
      <inputText><![CDATA[:installPath]/slws_install_dir/setup -s -f :installPath]/install.inf;]]></inputText>
      <exec cmd="sh"></exec>
    </execNative>
  </control>
</controlList>
```
The `<controlList>` element defines the detailed steps and order to complete the installation. For the Sun Java System Web Server installation the `<controlList>` defines an installation procedure named "run_install" that appears in the container Component's procedures list as shown in Figure 25. This element uses the `<execNative>` object to run the Sun Java System Web Server setup utility on the target host. When the Remote Agent implements the Plan it will process the variables list and substitute the defined variables in the install.inf file and then run the setup application.

For example, below is a portion of the install.inf file that provides configuration information to the Sun Java System Web Server installer.

```plaintext
... [core] Upgrade= False Reinstall= False AdminPort= 8888 HttpPort= 80 AdminSysUser= AdminName=: [target:admin_name] AdminPassword=: [target:s1ws_admpw] UgLdapUse= No CompVersions= 2:2:1:2:2:1:1:2: ... 
```

In the above example, the Remote Agent will use the Component variables to substitute the AdminName and AdminPassword parameters in the install.inf file.

As you learned by examining the XML definition the container Component provides two areas of function. The container Component customizes the Sun Java System Web Server installer configuration file using Component variables and variable substitution. It then uses custom XML commands to run the Sun Java System Web Server installer and later to start the Web server service.
5.2.4. Running the Plan

In the previous section you built new directory and container Components to perform a silent-install of the Sun Java System Web Server (Sun Java System Web Server.) In the container Component you used the XML authoring commands to define steps needed to customize and run the Sun Java System Web Server installer utility. To see the install method in action, click the run link next to the install procedure.

![Image of N1 Grid Service Provisioning System](image)

*Figure 27 - N1 Grid Service Provisioning System offers a view of the running Plan as the Remote Agents complete the Component tasks.*

N1 Grid Service Provisioning System shows a progress view as it marshals the resources needed to distribute the Sun Java System Web Server files as shown in Figure 27. In a few seconds N1 Grid Service Provisioning System begins distributing the installation files and Components to the Remote Agent. Figure 28 shows the view N1 Grid Service Provisioning System offers to give the user feedback on the progress of the deployment.
This scenario presented a system administrator's need to install Sun Java System Web Server on a group of machines in a one-to-one scenario whereby the same install needs to be moved from environment to environment with a few changes. The resulting set of Components leveraged the Sun Java System Web Server installer and installation configuration file to determine several installation parameters. The installation created a Component that moves the Sun Java System Web Server installation files to the target host, used a file Component type to configure the installer file to the target host, ran the Sun Java System Web Server installer, and started the Sun Java System Web Server service.

In the next section you will learn how to orchestrate a service using N1 Grid Service Provisioning System.
5.3. Orchestration of a Service

In the previous section you learned how to build an N1 Grid Service Provisioning System solution that installs and configures an application server. The solution used a set of Components and a Plan to perform a multi-step installation. In service oriented architecture (SOA) environments a service is rarely the product of a single application. Instead, services are delivered as orchestrations of many interoperating applications. For example, an extranet service to provide a supplier portal requires a Web server, application server, and database schema. In this section you will learn how N1 Grid Service Provisioning System provides a solution to orchestrate installation and configuration of a service.

To understand the problem of orchestration in more detail, consider a typical service as shown in Figure 29.

The example scenario depicted in Figure 29 shows the components of an extranet service that provides suppliers with parts ordering information at a manufacturing company. The extranet service is built with HTML files that will be served from a Web Server; A Java application packaged in an Java Web Application Resource (WAR) file; And, a database schema. The combination of these three components delivers the extranet service.

In a typical Information Technology (IT) environment the challenge is to orchestrate moving the service’s components from a software development environment to a preproduction environment, and then later to a production environment. This scenario presents an N1 Grid Service Provisioning System solution that enables a system administrator to orchestrate the service between these environments without changing the underlying code, html files, and schema.
5.3.1. Extranet Installation Components

The extranet service presented in this section is composed in three parts:

• Extranet Business Logic - Contains the business logic to handle user requests by using data stored in a database schema. This Component installs an application on a BEA WebLogic Server application server.

• Extranet Presentation - Delivers HTML content to users. This Component installs HTML files on a BEA WebLogic Web Server.

• JDBC Pool – Identifies a database schedule and Java Database Connector (JDBC) drivers to persist data and provide the business logic with an indexed method to query for data.

To understand the extranet service Components in depth, this section presents the XML definition of each of the three main Components. Additionally, some of the main Components use sub-Components that are also described.

5.3.1.1 Extranet Business Logic Component

The first Component is the Extranet Business Logic Component. This Component handles installation of a WAR file using the WebLogic ComponentType. The following is the XML definition of the Extranet Business Logic Component. After the XML definition you will find an explanation of the steps this Component takes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by N1 SPS -->
  <extends>
    <type name='container'></type>
  </extends>
  <varList>
    <var name='installPath' default="/opt/bea/user_projects/mydomain"></var>
    <var name='InstallUser' default='root'></var>
    <var name='InstallGroup' default='root'></var>
  </varList>
  <componentRefList>
    <componentRef name='ref1'>
      <component name='extranet' path="/WebLogic_Demo/BusinessLogic' version='2.0'/>
    </componentRef>
  </componentRefList>
</component>
```
The Extranet Business Logic Component makes use of two other Components: A WebLogic ComponentType that knows how to install and remove applications on BEA WebLogic Server and a file ComponentType that encapsulates the WAR file of the application.

The `<varlist>` identifies substitution variables whose values are evaluated and used when this Component runs. They enable the same Component to install on any valid target host, including development, preproduction, and production environments.
To work with BEA WebLogic Server, the Component uses a second component named extranet. It is a ComponentType of “WebLogic web application container” and contains the parameters and business logic to install and remove applications from BEA WebLogic Server. Below is the XML definition of the extranet Component:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by N1 SPS -->
    <extends>
        <type name='WebLogic web application'></type>
    </extends>
    <varList>
        <var name='wlName' default='extranet'></var>
    </varList>
    <componentRefList>
        <componentRef name='content' installMode='TOPLEVEL'>
            <type name='WebLogic web application container'></type>
            <component name='extranet_REG' path='/WebLogic_Demo/BusinessLogic' version='2.0'></component>
        </componentRef>
    </componentRefList>
</component>
```

The extranet Component instructs the Remote Agent to install the WAR file packaged in the extranet_REG Component into BEA WebLogic Server. When the Extranet Business Logic Component runs the extranet service application is installed on BEA WebLogic Server.

5.3.1.2 Extranet Presentation Component

The second of the three parts of the extranet service handles deployment and removal of the presentation layer of the extranet service. The following is the XML definition of the Extranet Presentation Component. After the XML definition you will find an explanation of the steps this Component takes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by N1 SPS -->
    <extends>
        <type name='container'></type>
    </extends>
</component>
```
The Extranet Presentation Component makes use of a second Component. The ExtranetPresentation Component encapsulates a directory loaded with HTML files. The installation passes-in parameters that identify the directory to install the HTML files and then copies all the HTML files in the directory to the designated path.
Using a Component to call another Component implements a level of indirection that makes it easy to change the target location for the HTML files using parameters.

### 5.3.1.1 JDBC Pool Component

The third of the three parts of the extranet service handles configuration of the JDBC driver and implementation of the schema in a relational database. The following is the XML definition of the Extranet Presentation Component. After the XML definition you will find an explanation of the steps this Component takes.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by N1 SPS -->
    <extends>
        <type name='container'/></type>
    </extends>
    <varList>
        <var name='installPath' default='/tmp'/></var>
        <var name='ORACLE_HOME' default='/opt/oracle'/></var>
        <var name='ORACLE_SID' default='DEMO'/></var>
        <var name='DomainName' default=':[target(:[target:adminServerHostName]):domainName]'/></var>
        <var name='ServerName' default=':[target:serverName]'/></var>
        <var name='WLS_USER' default=':[session:WL_DEFAULT_USER]'/></var>
        <var name='WLS_PASSWORD' default=':[session:WL_DEFAULT_PASSWORD]'/></var>
        <var name='ServerPort' default=':[target(:[target:adminServerHostName]):adminPort]'></var>
        <var name='WLS_URL' default=':[target(:[target:adminServerHostName]):adminHost]::[ServerPort]'></var>
        <var name='BEA_HOME' default=':[target(:[target:adminServerHostName]):wlHomeDir]'></var>
        <var name='WLS_HOME' default=':[BEA_HOME]/weblogic700' ></var>
        <var name='POOL_NAME' default='oraclePool' ></var>
        <var name='InstallUser' default='root'/></var>
        <var name='InstallGroup' default='root'/></var>
    </varList>
    <componentRefList>
        <componentRef name='ref1'>
            <argList WLS_PASSWORD=':[WLS_PASSWORD]' DomainName=':[DomainName]' InstallPath=':[installPath]' />
```
installGroup=':[InstallGroup]' POOL_NAME=':[POOL_NAME]' 
installName='create_jdbcpool.sh' WLS_URL=':[WLS_URL]' 
WLS_USER=':[WLS_USER]' ServerName=':[ServerName]' 
ORACLE_SID=':[ORACLE_SID]' installUser=':[InstallUser]' 
WLS_HOME=':[WLS_HOME]'</argList>
</component>

<component name='create_jdbcpool.sh' 
path='/WebLogic_Demo/BusinessLogic' version='4.0'></component>

<componentRef name='ref2'>
<argList WLS_PASSWORD=':[WLS_PASSWORD]' 
installPath=':[installPath]' installGroup=':[InstallGroup]' 
POOL_NAME=':[POOL_NAME]' installName='delete_jdbcpool.sh' 
WLS_URL=':[WLS_URL]' WLS_USER=':[WLS_USER]' 
ORACLE_SID=':[ORACLE_SID]' installUser=':[InstallUser]' 
WLS_HOME=':[WLS_HOME]'></argList>
<component name='delete_jdbcpool.sh' 
path='/WebLogic_Demo/BusinessLogic' version='4.0'></component>
</componentRefList>

<installList>
<installSteps name='default'>
<install blockName='default'>
<allNestedRefs></allNestedRefs>
</install>
<call blockName='Create JDBC Pool'>
<thisComponent></thisComponent>
</call>
</installSteps>
</installList>

<uninstallList>
<uninstallSteps name='default'>
<call blockName='Delete JDBC Pool'>
<thisComponent></thisComponent>
</call>
<uninstall blockName='default'>
<allNestedRefs></allNestedRefs>
</uninstall>
</uninstallSteps>
</uninstallList>

<controlList>
<control name='Create JDBC Pool' description='Creates 
JDBC Pool oraclePool'>
<execNative dir=':[installPath]'>
<exec 
cmd=':[installPath]/create_jdbcpool.sh'></exec>
</execNative>
</control>
<control name='Delete JDBC Pool' description='Delete 
JDBC Pool oraclePool'>
<execNative dir=':[installPath]'>
<exec 
cmd=':[installPath]/delete_jdbcpool.sh'></exec>
</execNative>
</control>
</controlList>
</component>
The JDBC Pool Component makes use of a second Component. The create_jdbcpool.sh Component encapsulates a Unix shell script that performs the JDBC driver configuration. Most of the XML definition for the JDBC Pool Component provides the location and configuration information that the shell script will use.

```xml
<varList>
  <var name='installPath' default='/tmp'></var>
  <var name='ORACLE_HOME' default='/opt/oracle'></var>
  <var name='ORACLE_SID' default='DEMO'></var>
  <var name='DomainName' default=':[target(:[target:adminServerHostName]):domainName]'>
  ...
</varList>
```

The <varList> enables the JDBC Pool Component to use parameters to determine installation and configuration details for the JDBC drivers. Parameters are important for reuse of this Component when installing on preproduction and production environments. For example, the above XML tells the JDBC driver which ORACLE_HOME value to use.

Here is the XML definition of the create_jdbcpool.sh Component. It copies the shell script to the target host, substitutes the passed-in parameters. Later you will see how a Plan uses the JDBC Pool Component and executes the shell script.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by N1 SPS -->
<component xmlns:xsi='http://www.w3.org/2001/XMLSchema-instance' name='create_jdbcpool.sh' version='4.1'
path='/WebLogic_Demo/BusinessLogic'>
  <extends>
    <type name='file'></type>
  </extends>
  <varList>
    <var name='installName' default='create_jdbcpool.sh'></var>
    <var name='DomainName' default=':[container:DomainName]'></var>
    <var name='WLS_URL' default=':[container:WLS_URL]'></var>
    <var name='WLS_HOME' default=':[container:WLS_HOME]'></var>
    <var name='POOL_NAME' default=':[container:POOL_NAME]'></var>
    <var name='ServerName' default=':[container:ServerName]'></var>
    <var name='ORACLE_SID' default=':[container:ORACLE_SID]'></var>
    <var name='WLS_USER' default=':[container:WLS_USER]'></var>
  </varList>
</component>
```
The above three sections presented the three main Components and their sub-Components used in the extranet service solution to deploy and remove the service. Much of the XML in the Component definitions is used to parameterize the installation. This facilitates redirecting the installation between target environments, including development, preproduction, and production environments. Next you will learn about the Plan that commands the Components.

5.3.2. Extranet Service Plans

In the previous section you learned how and why the extranet service deployment solution used three main Components and a set of sub-Components. In this section you will learn how the Components are orchestrated by the extranet service deployment Plan.

The following is the XML definition of the extranet service deployment Plan.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<!-- generated by N1 SPS -->
<executionPlan xmlns='http://www.sun.com/schema/SPS'
name='Deploy Extranet App' version='4.1' description='Deploy the n-tier Extranet application'
xmns:xsi='http://www.w3.org/2001/XMLSchema-instance'
path='/WebLogic_Demo'
xsi:schemaLocation='http://www.sun.com/schema/SPS plan.xsd'>
  <simpleSteps>
    <install blockName='default'>
      <component name='Extranet Business Logic'></component>
      </install>
    <install blockName='default'>
      <component name='Extranet Presentation'></component>
      </install>
    <install blockName='default'>
      <component name='JDBC Pool'></component>
      </install>
  </simpleSteps>
</executionPlan>
```

The deployment plan orchestrates three Components: Extranet Business Logic, Extranet Presentation, and JDBC Pool. Figure 30 shows how this Plan looks in the N1 Grid Service Provisioning System Web browser interface when you run the Plan.
1. Click the run button to begin deployment of the extranet service.

When you run the Plan, N1 Grid Service Provisioning System assembles the Components and Plans and readies the installation. Figure 31 shows the next part of the deployment.
Figure 31 - The Deploy Extranet App Plan is ready to run.

N1 Grid Service Provisioning System is now ready to do a preflight check of the Components and Plans to make certain the Master Service and Remote Agents are properly configured and the Components and Plans are complete.

2. Choose one or more target hosts.

3. Click run plan (includes preflight)
Figure 32 shows the Deploy Extranet App Plan conducting a preflight check of the Components and Plan.

![Preflight Check](image)

Figure 32 - The preflight checks to make certain all the Components are checked-in and the Remote Agents are ready.

Once the preflight check completes successfully the view changes to the browser interface shown in Figure 33.
Figure 33 - A successfully installation of the extranet service to the preproduction environment.

Figure 33 shows the browser interface as the Plan runs on the Remote Agents to deploy the extranet service to the target hosts. Upon completion, you may access the extranet service on the target hosts as shown in Figure 34.
In this section you learned how the N1 Grid Service Provisioning System orchestrates the extranet service using Components and Plans to deploy and remove the extranet service.
6. Advanced XML Authoring

The example scenarios in section 5 demonstrate N1 Grid Service Provisioning System in action automating updates and changes to a datacenter. The XML authoring system is an integral tool that enables you to author automated solutions for distribution, provisioning, and installation automation. The XML schema defines the syntax and semantics for the control structures used to model and manage applications within the N1 Grid Service Provisioning System software.

While the XML schema examples presented in section 5 showed the XML scripting language in action, there are many additional advanced XML authoring elements, including the following.

- Dependency checking
- Failure identification and back-out control
- Conditional execution
- Code encapsulation
- 2-Phase Installation

This section provides a summary of a very small portion of the advanced XML authoring topics available to you. The *N1 Service Provisioning System Reference Guide* provides in-depth coverage of all the XML schema elements available.

6.1. Dependency checking

The XML definitions of Components and Plans may incorporate XML elements to identify and handle dependencies while a Remote Agent runs a Plan. For example, when installing a new database schema a Plan may first check to make certain the database application has already been installed. The following shows an example where a Plan uses `<checkDependency>` to verify that a database Component has already been installed.

```xml
<checkDependency>
  <installedComponent name="db" installPath="." host="ptt"/>
  <!-- Dependency checks so install db schema -->
</checkDepenency>
```

In the above example, `<checkDependency>` verifies that the db component has been installed on the target host. If the component is not installed, then the step fails and execution of the element stops.

Chapter 3 of the *N1 Service Provisioning System Reference Guide* provides details on `<checkDependency>` and many other useful elements.
6.2. Failure identification and back-out control

Plans and components may incorporate try/catch/finally control blocks to identify and recover-from failures. For example, the following XML defines exception handling code to help back-out from a failed database schema installation.

```xml
<control blockName="default">
  <try>
    <block>
      <!-- Install DB schema -->
    </block>
    <catch>
      <!-- handle the error by backing out -->
      <!-- rethrow error -->
      <raise/>
    </catch>
    <finally>
      <!-- Close the db connection -->
    </finally>
  </try>
</control>
```

The `<catch>` element is used to suppress and recover from errors encountered in the `<block>` element. The `<finally>` element is used to unconditionally perform cleanup of an attempted operation, regardless of whether errors were encountered. The `<raise>` element indicates a failure condition without having to construct an artificial step to do so. In the above example, the `<raise>` element most often appears within a `<catch>` block to propagate an error condition after cleaning up.

6.3. Conditional execution

Components and Plans may incorporate logical condition checking using regular expressions and branch in different directions based on the results. The XML schema provides a set of conditional execution elements. For example, the following XML script determines if a variable with a Plan exists.

```xml
<istrue value=":\[filesystemready]">
  <!-- Install a db schema -->
</istrue>
```

If the value of the filesystemready variable is true then the enclosed command to install a database schema will run. If the value is false then the `<istrue>` element ends and processing continues at the next element.
N1 Grid Service Provisioning System provides several conditional execution commands, include the following:

<istrue value="True"/>
This evaluates to true and so the contained elements are processed.

<equals value1="apple" value2="orange"/>
This evaluates to false and processing continues to the next element.

<equals value1=":[var1]" value2=":[var2]"
This looks at the values of the var1 and var2 input variables. When the value of var1 is equal to the value of var2 the enclosed elements are processed. Otherwise, they are skipped.

The *N1 Service Provisioning System Reference Guide* provides in-depth coverage of all the conditional execution elements available.

### 6.4. Code Encapsulation

The XML schema provides advanced capabilities for instances where N1 Grid Service Provisioning System solutions must handle scenarios that are learned at runtime. Your XML authored Components and Plans may grow to the point where code encapsulation becomes worthwhile. Encapsulating frequently used XML scripts into reusable code libraries decreases the maintenance of Components and Plans overall and allows specialization of N1 Grid Service Provisioning System users.

While there are a variety of methods supported in the XML schema to achieve code encapsulation in N1 Grid Service Provisioning System this section highlights two methods: `<extends>` and `<execSubplan>`.

#### 6.4.1. `<extends>`

`<extends>` is used in a Component’s XML definition to declare a base class from which the Component can inherit methods. This is analogous to Java’s *extends* reserved word for inheriting from classes. The `<extends>` element is a child of the `<component>` element and is used to declare base component from which this component is derived. A Component that uses `<extends>` automatically inherits attributes and elements of the base component. The Component may also selectively override aspects of the inherited data. Allowances for inheritance and overrides are specified in the description of the attribute/element that is inherited.

In N1 Grid Service Provisioning System a Component is said to be an instance of the Component that it extends. It is also an instance of the Components that the base Component is an instance of.
6.4.2.  <execSubplan>

<execSubplan> permits Plans to call other Plans and pass parameters. In an N1 Grid Service Provisioning System XML authored Plan, both parameters and variables may be declared. The value of a variable is defined at the point of declaration based on the values of other variables and constants. A parameter is a special kind of variable whose value is defined by the caller.

For example, when a user initiates a Plan to run, the user enters the desired values for each parameter declared by the Plan prior to running the plan. When a Plan is invoked as the result of an <execSubplan> call, the Plan containing the <execSubplan> call is responsible for explicitly passing values for each of the parameters declared by the called Plans.

N1 Grid Service Provisioning System provides additional code encapsulation support with the <execFile>, <execJava>, and other elements. The *N1 Service Provisioning System Reference Guide* provides in-depth coverage of these.

6.5.  2-Phase Installation

In certain circumstances a Component requires a two phase installation. For example, when working with the Windows Registry a Component may first need to enter Microsoft Windows Registry Keys and then complete installation of a COM object. N1 Grid Service Provisioning System provides support for multi-phase installation using the <retarget> element.

<retarget> is a step that changes the execution target for a set of steps. For example, when working with WebLogic Server a Plan may need to restart a service. In the following example, a Plan calls a control on the admin server to "stop" the managed server, and then makes a call on the local machine to start the server.

```xml
<control name="restart">
  <varList>
    <var name="adminHostName" default=":[target:adminHostName]"/>
  </varList>
  <retarget host=":[adminHostName]">
    <varList>
      <var name="domainName" default=":[target:domainName]"/>
    </varList>
    <call blockName="stopServer">
      <argList serverName=":[serverName]" domainName=":[domainName]"/>
      <installedComponent name="ADMIN_SERVER"/>
    </call>
  </retarget>
  <call blockName="start"/>
</control>
```
Retarget steps may be nested. The host attribute is used in the <retarget> step, as well as in various component targeter elements. The value is the name of the host and may be a substitution variable reference.

The *N1 Service Provisioning System Reference Guide* provides details on `<retarget>`. 
7. Glossary of terms

Component model:

An XML representation of a component. The component model is a bill of materials of the applications that includes:

- A list of resources used by the application
- Install steps
- Uninstall steps
- Dependencies

Container:

A Component type that references other Components.

Component:

A component is a unit that is deployed to hosts in the provisioning software. It comprises a build between a component model and a specific set of resources for an application, such as a custom Java application or an infrastructure software package. A component can comprise multiple versions of an application. In this case, a separate component build represents each checked-in version of the application.

Configuration Utilities:

Configuration utilities include management consoles, installer utilities, scripting utilities, software updating services, application optimization and load balancing solutions.

Deployment:

Execution of a Plan. Running a Plan automatically distributes a Component or Components included in the Plan. The deployment follows installation instructions set in the Plan. Deployment is always preceded by a preflight. If the preflight is successful, the Plan Executor deploys/installs components to the target host.
Comparisons:

A provisioning software feature that searches for and identifies differences between hosts. There are three types of comparisons: Model to Install, Model-to-Model, Directory-to-Directory.

The install-to-install comparison examines the contents of two hosts' file systems and reports any differences.

The model-to-install comparison compares what the Master Server reports is installed on a host against what is actually installed on the host, and reports any differences.

A model-to-model comparison examines the deployment repository and history (stored on the Master Server) for two hosts, and reports any differences.

execNative:

An optional call out to custom scripts from a plan XML.

Gold server:

A reference (outside) host that contains files, directories, and other resources comprising an application you want to deploy. These resources are checked in (uploaded) to the provisioning software from the gold server.

Host:

A server connected to the provisioning software. A host can represent an individual server or a set of servers.

Host set:

A user-defined, logical grouping of hosts sharing one or more common attributes, such as physical location or functional group. A defined host set allows quick and easy application updates to all hosts in the set, as well as install-to-install comparisons between the hosts. Host sets can be nested.

Host type:

A base class of servers bound by a set of common attributes, all of which are user-defined. Hosts belonging to a host type set use particular attribute values in variable substitutions in all deployments. Host types can be used to categorize hosts into logical groupings, such as sets of applications required for the hosts, geographic locations of the hosts, or functional groups. Host types also facilitate host searches by allowing users to search for all hosts of a certain type.
Host search:

A query run on the repository that yields a list of hosts whose attributes match those specified in the query. You can use host searches to create lists of hosts that have the same host type, are running the same applications, are configured with the same subnet masks, and so on.

Local Distributor:

An engine, installed on a machine with the provisioning software, that acts as a link between other objects, such as the Master Server to Remote Agents, the Master Server to other Local Distributors, and another Local Distributor to Remote Agents. Designed to maximize bandwidth efficiency and speed, Local Distributors can also provide secure network connections for navigating restricted environments.

Master Server:

The main processing engine in the provisioning software. It is installed on a dedicated machine at a central location, where it can connect to any of the data center environments. The Master Server provides the centralized data storage, data processing, and user interfaces.

Modeling Language:

Programmatic ways to describe a series of actions, the order of the actions, dependencies, and logical checks to change, configure and provision systems. Modeling languages are different from scripting languages in that they define a set of actions that are appropriate to the environment, rather than describing the procedure as a series of one-step-after-another tasks in a script.

Plan:

An XML representation of the deployment script that references components to be deployed. A plans include:

- Install/uninstall steps.
- Start/Stop components (optional).
- Verification steps (optional)
- ExecNatives (optional)

Plan executor:

The provisioning software's deployment engine. It handles preflights and actual deployments.
Preflight:

A test "deployment" of a plan to a simulated UNIX environment that:

- Checks availability and connectivity of target hosts.
- Confirms permissions.
- Validates dependencies.

A preflight finds and reports any errors or potential errors that could affect the actual deployment. Preflights always automatically proceed a deployment; they can also be run as standalone procedures that are not automatically followed by a deployment.

Provisioning

The action of configuring a host for use by a new user or new application, including adding a new account, setting an initial password, and configuring privileges to use application data.

Remote Agent:

An engine installed on any machine in the provisioning software to which components are deployed. On that machine, the Remote Agent manages tasks such as installing software, controlling services, collecting information to send back to the Master Server.

Resources:

Files that are deployed to hosts when a plan is executed.

Resource types:

Built-in resource categories identified and supported by the provisioning software. When a resource is checked in, it must be tagged with the appropriate resource type. Resource types include files, directories, IIS Types, and COM+ components.

Snapshot:

A capture of resources stored on a Remote Agent during a deployment. The snapshot is used during comparisons between hosts (install-to-install) or between a host and the model of the host on the Master Server (model-to-install).
8. Resources

Additional knowledge, assistance, software, and documentation may be found in these resources:

- Sun offers developers a free license for the N1 Grid Service Provisioning System. Download the N1 Grid Toolkit at http://www.sun.com/software/solutions/n1/toolkit/

- Access N1 Service Provisioning System 4.1 books in the product distribution and on the Web at http://docs.sun.com/db/coll/1119.1

- Additional information on N1 Grid solutions is found at http://www.sun.com/n1

- A comprehensive guide to system compatibility is available in the install guide found at http://docs.sun.com/db/coll/1119.1

- Sun Professional Services for N1 Grid services offered are summarized at http://wwws.sun.com/service/sunps/architect/n1/index.html

9. Support

No support is provided for the N1 Grid Toolkit which includes the Developer license of N1 Grid Service Provisioning System 4.1. To be eligible for support, you must have purchased production licenses of the N1 Grid Service Provisioning System (available on the internet at http://store.sun.com).

We welcome your comments, feedback or inquiries on how to make your commercial applications N1 Grid ready at the following email N1GSPS-feedback@sun.com