SUN BLUEPRINTS APPROVED

[illegible]

Service Provisioning with Resource Management

Sam Antwi, Senior Architect, Sun Client Services

Sun BluePrints™ OnLine—November 2004

`http://www.sun.com/blueprints`

Sun Microsystems, Inc.
4150 Network Circle
Santa Clara, CA 95045 U.S.A.
(650) 960-1300

Part No. 817-7525-10
Revision 03, 11/11/04
Edition: November 2004

Please
Recycle

Adobe PostScript™

Service Provisioning with Resource Management

Sun customers with large deployments are increasingly looking for ways to drive out cost and complexity in the data center through a number of major, ongoing or planned IT initiatives. Some of these initiatives include server consolidation, application stacking, automated service provisioning, preprovisioned shared infrastructure services, and dynamic resource provisioning and management.

Years of over-sizing platforms (over-provisioning) and business unit IT autonomy with a *one-application-one-platform* style of deployment have left IT environments with low system utilization, wasted resources, and large, unused system *footprints* in the data center. Enterprises are looking to leverage emerging, smaller server footprints of virtualized technologies to deliver the same services, while maximizing system utilization, delaying capacity decisions, and balancing system resource requests with optimal resource provisioning.

A popular way to maximize data center resource utilization on a sustained basis lies in the ability of infrastructure environments to deliver a level of agility to service demand in which resources are dynamically provisioned to requesting service workloads. Services can be migrated to the resource availability point, achieving an overall utilization balance across the enterprise. Application stacking and service deployment or provisioning is key to realizing savings in the next-generation data center.

With these cost-savings measures, the risk for IT is the challenge to continue to maintain or exceed user expectations regarding service level commitments on performance and availability. Stacked service applications of unknown workload characteristics hosted in the same operating system image might present resource contention and other service contamination imperatives, which might unfavorably impact service delivery in regards to service level agreements.

IT needs a comprehensive, enterprise-based resource management solution born out of a network service and driven through a business policy engine in which service provisioning, resource management, and service level commitments meet operational characteristics. Driving technologies for such capacities are still evolving in the next generation data center initiatives such as Sun’s N1™ Grid software. In the meantime, IT organizations need to deploy host-specific resource management, along with service provisioning capabilities, to address expected service levels for mixed workload, stacked application environments.

Enterprise resource management features and techniques from systems vendors are meeting this need with ever-expanding system features based on the latest versions of operating systems. These include workload management, distributed resource management, and hardware and software partitioning schemes at various levels of granularity. On the Solaris™ Operating System (Solaris OS) platform, the features or techniques include dynamic reconfiguration, dynamic domaining, resource pooling, containers, and grid computing, along with their respective system attributes.

While resource management has been a feature of the Solaris OS for sometime, the Solaris™ 9 Resource Management (Solaris 9 RM) offers a more granular, elegant, and flexible solution to Solaris OS platform resource sharing and control. The Solaris 9 RM offers the best, most predictable approach to guaranteed service level commitment, even in environments where resources are contested by multiple stacked application service workloads or where maximizing system resource utilization is paramount.

Application service provisioning has automation features that offer IT organizations the ability to migrate application workloads to the point of resource availability, thus delivering service agility. Additionally, the automated nature of application service provisioning through automatic deployment offers opportunities to improve service quality through automation, as well as the ability to scale large service deployments.

Definition and Integration Scope

The Solaris 9 RM implementation example in this article is a rather simple illustration of the Solaris 9 RM software. It begins with a standard provisioning of the Solaris 9 RM in a multiservice scenario. It then progresses into an automated provisioning of the Solaris 9 RM, which is integrated into a sample automated deployment of the Sun™ ONE Web Server using the N1 Grid Service Provisioning System (N1 Grid SPS) tool. This same integration technique could be used to provision resource controls into other enterprise software (services) such as BEA WebLogic and Oracle9i® Database Server.

In particular, the manual provisioning aspect of the Solaris 9 RM in this article involves three application service workloads: Sun ONE Web Server, BEA WebLogic, and Oracle9i. The Solaris 9 RM implementation involves sharing CPU resources on a Solaris OS server. In Solaris 9 RM, a share of CPU resource is a defined as a slice or portion of the CPU allocated to an application or workload throughout its run cycle on a system. Shares are not percentages or fractions of CPU time, but rather denote the priority or relative importance of an application service relative to other application services (workload) and to the business in terms of service level guarantees.

The CPU shares are pre-allocated among three enterprise application service workloads. Then, the Solaris 9 RM software is provisioned to use its resource control algorithms to enforce the resource constraints of CPU share allocations. The CPU shares are assigned to each application service workload at the Solaris 9 RM project level. The project definitions are configured in the `/etc/project` file. User and group IDs and memberships are set up in the appropriate projects. In this case, the user accounts are the *run as* users for each of the application services (Sun ONE Web Server, BEA Worklogic, and Oracle9i).

During the automated service provisioning section of this article, automated provisioning commands for the Solaris 9 RM software are integrated into the N1 Grid SPS service provisioning models. Specifically, Solaris 9 RM provisioning or configuration commands are integrated into Sun ONE Web Server XML-based plans and components inside the N1 Grid SPS software. To help you understand the XML representations, a compilation of the name spaces that are necessary for service provisioning the Sun ONE Web Server software is provided.

Although other approaches for integrating resource management into service provisioning are possible, such as host-based or network database, provisioning-based resource management has merit for stable policy environments, as well as small scale nonpervasive implementations. In the absence of comprehensive, enterprise-wide policy-based service provisioning with integrated resource management, any method for ensuring resource controls through Solaris 9 RM can be employed.

Solaris 9 RM Provisioning Guidelines

Provisioning Solaris 9 RM manually or as an integral part of service provisioning requires planning, specification, and proper name space determination. It begins with an understanding of the workloads and the resource demands. It also includes allocation of CPU shares as resource constraints, governed by an overall set of business policies consistent with service management requirements. For the implementation example in this article, the following guidelines are provided:

- Define a Solaris 9 RM project per application or service component for each of the following: Sun ONE Web Server, BEA WebLogic, and Oracle9 servers.
- Implement a simple system CPU share resource allocation and management scheme. Within Solaris 9 RM, this resource control is called `project.cpu-share`.

- Assume multiple instances for each type of service.
- Establish Solaris OS accounts to both own and run the enterprise application services.
- Name the Solaris 9 RM projects as follows:
  - Sun ONE Web Server: `user.premiws01`
  - WebLogic Application: `user.premapp01`
  - Oracle database service: `user.premdb01`
- Allocate CPU share resource control threshold values for the three workload types (projects) as follows:
  - 50 shares for the Sun ONE Web Server service
  - 100 shares for the BEA WebLogic service
  - 150 shares for the Oracle Server9i service
- Implement Solaris 9 RM projects as host-based projects in the `/etc/project` file, and not as a central network-based project database service in LDAP.
- Avoid Solaris 9 RM resource pool definitions and allocations in this sample implementation will not be supported.
- Use the fair share scheduler (FSS) in the Solaris 9 RM and assigned as the default scheduling class.

Naming

Naming of implementation elements and their associations is very important in a service provisioning context given the arbitrary number of services, service instances, and relationships between application service components across different tiers of a service architecture. This is more so when goals for higher system utilization imperatives promise to force hosting multiple applications in a single image of the operating system.

Simplification of managing the data center begins with a structured nomenclature for administrators to easily recognize services for ongoing management. First, you must name the application or service. To recognize different service components and their instances, which might have a close affinity to one another (that is, they might deliver a particular instance or component of a particular service), the service name tends to follow the application name. Next, you must establish that defining Solaris 9 RM projects for each application or service is paramount in this integration, as workload resource request and usage of system resources will be accounted for and controlled at the Solaris 9 RM project level.

The Solaris 9 RM project naming in this article follows the `user.serviceowner` template. Project definition is registered to the Solaris OS in the `/etc/project` file. User accounts are then allocated memberships in projects. As users log in, their default or assigned membership is checked against the project file, and process workload accounting and resource control begins at that time. All subsequent application service workload-related resource requests made by that service or application owner on that system will be tracked and managed to that project.

TABLE 1 shows the application service implementation element naming for the Solaris 9 RM provisioning and resource management.

TABLE 1 Application Service Naming Description

Service or Project Name	Service Types	Service Notation	Service Instance
`Prem`	Three-tier (single instance of each tier)	`www`	`premwww01`

TABLE 2 shows the service and account naming.

TABLE 2 Service and Account Naming

Application, Service, Component, or Workload	Type	Service Instance Name	Account, Owner, User, or Group ID
`SunONEwebserver`	Web service	`premiws01`	`premiws01`
`WebLogic`	Application service	`premapp01`	`premapp01`
`OracleServer`	Data service	`premdb01`	`premdb01`

Additionally, the Solaris 9 RM project name and share allocations are depicted in TABLE 3.

TABLE 3 Project and CPU Share Allocation

Service Component, Application, or Workload	Project Name	CPU Share Allocation
Sun ONE Web Server	`user.premiws01`	50
BEA WebLogic	`user.premapp01`	100
Oracle9i server	`user.premdb01`	150

To complete the entries in the `/etc/project` file, the project name from TABLE 2 must be supplied. Then, the appropriate accounts must be added as members of that project. In this example, the application service owner accounts (already registered in `/etc/passwd` and `/etc/group`) are combined with the project information from TABLE 2.

Next, the system resource to control, in this case the CPU, is determined as the Solaris 9 RM name, `project.cpu-shares`. Finally, the action clause must be constructed, made up of the access privilege, the CPU share threshold value assigned to the particular workload or application service from TABLE 2, and the necessary action when violation of resource control occurs. Based on the implementation guidelines, the Solaris 9 RM action clause for Sun ONE Web Server would be `(privileged, 50, deny)`.

Here is a sample `/etc/project` entry for a workload or service representing a single Sun ONE Web Server instance, per the established guidelines.

```
user.premiws01:2001:SunONEwebserver
Application:premiws01::project.cpu-shares=(privileged,50,deny)
```

Standard Provisioning for the Solaris 9 RM Software

To enable the implementation of a basic Solaris 9 RM, it is necessary to perform the following steps before the systems administrator begins to define projects, assign user accounts, and allocate CPU shares.

Note – Before you set up the provisioning environment, confirm that the system is not already running the Fair Share Scheduler (FSS) as the default scheduling class.

t To Set Up the Provisioning Environment

1. **As superuser, manually enable the FSS class with the following command:**

```
# dispadmin -d FSS
```

2. **Reboot the system.**
3. **Update the** `nsswitch.conf` **host line to force the system to visit the local files for project information, instead of looking up project information in LDAP, NIS+, or other central service.**

4. **Determine and create the project-member Solaris OS accounts and groups in the `/etc/passwd`, `/etc/shadow`, and `/etc/group` files using the `useradd`(1M) and `groupadd`(1M) commands, then propagating and synchronizing those accounts into the `/etc/shadow` file with the `pwconv`(1M) command.**

   The following are sample results:

```
# cat /etc/passwd
..
sama:x:100:1::/export/home/sama:/bin/sh
premiws01:x:101:1::/export/home/premiws01:/bin/sh
premmw01:x:102:1001::/opt/app/premmw01:/bin/sh
premdb01:x:102:1001::/opt/app/premdb01:/bin/sh
```

5. **Build CPU share allocation entries for the different services or workloads in the Solaris OS 9 `/etc/project` file.**

   The allocation entries are predetermined based on workload resource allocation and control policy, in accordance with service management and business needs.

   For the sample implementation in this article, the Sun ONE Web Server entry in the `/etc/project` file is:

```
user.premiws01:2001:SunONEwebserver:premiws01::project.cpu-
shares(privileged,50,deny)
```

   The entry for BEA WebLogic instance would be:

```
user.premapp01:2002:WebLogic:premapp01::project.cpu-
shares=(privileged,100,deny)
```

   Finally, the Oracle9 instance entry would be declared as follow:

```
user.premb01:2003:OracleServer:premdb01::project.cpu-
shares=(privileged,150,deny)
```

6. **Edit the `/etc/project` file for the entries constructed in Step 4.**

   The following shows the results for the example implementation described in this article:

```
# cat /etc/project
system:0::::
user.root:1::::
noproject:2::::
default:3::::
group.staff:10::::
user.premiws01:4003:Support for web:premiws01::
project.cpu-shares=(privileged,50,deny)
user.premmw01:2001:Support for
middleware:premmw01::project.cpu-shares=(privileged,100,deny)
user.premdb01:2006:Support for db:premdb01::project.cpu-shares=
(privileged,150,deny)
```

The proper project entries in the `/etc/project` file allow the Solaris 9 RM to determine default projects for the Sun ONE Web Server, BEA WebLogic, and Oracle9i services, related to resource usage tracking and accounting. After the respective application service instances start up and run as the Solaris OS service account owners, all resource requests are tracked in the corresponding projects through the service account membership assignments in the `/etc/project` entries.

Solaris 9 RM Automatic Deployment Integration

Having established how the Solaris 9 RM is configured manually for multiservice workloads, this section contains an example of how to integrate an automated version of the above manual procedure into application provisioning involving the Sun ONE Web Server. The example uses the same implementation guidelines established earlier, except for the environment preparation steps, which in automated implementations can be addressed as a finishing step.

To integrate the Solaris 9 RM configuration commands into the N1 Grid SPS service provisioning actions, you must understand the schema and implementation of the larger context for that integration—in this case the Sun ONE Web Server in the N1 Grid SPS software. The example in this section shows how the Solaris 9 RM commands manifest inside the N1 Grid SPS, as an integral part of application service provisioning. The example demonstrates this integration only for the Sun ONE Web Server application service or component, but you can extend these ideas to any other application service or workload type.

The N1 Grid SPS software (formerly known as CenterRun) uses two core constructs called execution plans and components to model the application characteristics for service deployment and other data center operational behaviors (for example, migrations, compare deployments, remove deployments, and upgrade software deployments) being performed on an application service component. The N1 Grid SPS plans represent sequential steps for affecting a system management behavior, system management functionality, or data center task, such as an install, configuration, uninstall, upgrade, comparison, or change propagation. While plans are optional in the N1 Grid SPS, they are necessary for complex tasks. Components in the N1 Grid SPS represent the applications or software modules on which major action or behavior is performed. In the N1 Grid SPS, plans and components are represented as XML code, which can be autogenerated as part of the modeling, development, and test cycles and effectively manipulated manually through common text editors.

This article describes only the integration of the Solaris 9 RM provisioning commands into the N1 Grid SPS deployment steps for the Sun ONE Web Server. The Solaris 9 RM provisioning commands are inserted into the appropriate sections of Sun ONE Web Server XML code. In a way, the application deployment is provisioning and registering its own CPU share allocation to the environment every time and every where (host location) the Sun ONE Web Server application service is provisioned. These ideas could be extended to other operational *behaviors* on the Sun ONE Web Server such as service migrations to other hosts in the data center.

An interesting advantage for provisioning-based resource management scenarios is when a Sun ONE Web Server application service is retired from an environment. A Solaris 9 RM removal functionality (or using a modified version of the uninstall built-in procedure) inside the N1 Grid SPS would cause Sun ONE Web Server provisioning to be removed from the environment, along with the related Solaris 9 RM definitions for the Sun ONE Web Server CPU share resource controls, providing a cleanup functionality as a bonus.

N1 Grid SPS Planning and Design

First, you need a high-level design approach for the integration exercise. FIGURE 1 shows a high-level design involving the key functional elements in the N1 Grid SPS plans and components. Typically, implementing a data center service functionality with Sun's N1 Grid SPS involves a simple design framework involving plans and components. Plans represent the sequencing of functionality on components. Components represents the service software primitives, applications, or modules on which plans act to deliver a particular data center system management behavior, such as upgrading software, applying patches, deploying an application, repurposing a server, propagating file contents to other files, and comparing existing deployments.

Plans can be simple or complex, and they can call any number of other plans. Plans invoke the behavior encapsulated in applications or components (modeled separately). Components do not need plans to effect a behavior or functionality in the data center because they have built-in constructs to affect their own behaviors. This is why the N1 Grid SPS is completely object oriented, where as XML-based components have built-in behavior, procedures, or *methods* (in object speak), as well as data structures (including parameter declaration), parameter passing, and data resources.

FIGURE 1 shows how you can integrate the Solaris 9 RM into blocks of N1 Grid SPS elements, which implement the Sun ONE Web Server auto-deployment. The context is an N1 Grid SPS plan to provision or deploy the Sun ONE Web Server. Inside the plan is a call or reference to a Sun ONE Web Server component, specifically a control block (`controlList`) inside that component (for more information, refer to an N1 Grid SPS component schema definition).

The control block sets up the necessary parameters that are passed to a generic Solaris OS component that implements several *modules* of Solaris OS functionality for setting up Solaris OS accounts, provisioning the Solaris 9 RM, or even mounting file systems. In the example integration, the Solaris OS provisioning for account creation is invoked for the run owner on the Sun ONE Web Server, `premiws01` from TABLE 2.

As FIGURE 1 shows, the generic Solaris OS component is neatly de-coupled from application service components. It is a substrate component that could be leveraged across all Solaris OS-based N1 Grid SPS implementations to support any number of service behaviors at the application level, while delivering common Solaris OS functionality to application services such as BEA WebLogic or Oracle9i.

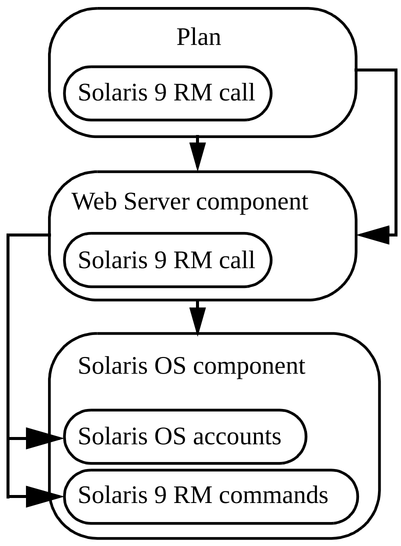

FIGURE 1 High-Level Design for Integrated Provisioning

Plan Integration

This article is not a full treatment of Sun ONE Web Server service provisioning or deployment. It focuses on how to integrate Solaris 9 RM provisioning commands inside the service deployment in general. In the XML examples that follow, detailed Sun ONE Web Server service provisioning commands that are not directly relevant to the integration tasks at hand are suppressed with “. . . . .” inside the code examples. As shown in the structure in FIGURE 1, you must begin with the Sun ONE Web Server XML plan, as shown in the following code example. The XML code is annotated with comments to help explain the details of the code.

CODE EXAMPLE 1 Sun ONE Web Server XML Plan and Call

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!-- generated by CR
-->
- <executionPlan
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
name="Application_Create_SunONE-Wbs" version="3.0"
xsi:schemaLocation="http://www.centerrun.com/schema/CR
plan.xsd" xmlns="http://www.centerrun.com/schema/CR">
- <compositeSteps> # begin a compound steps
 ......
.........
+ <inlineSubplan planName="iws60_create">

.......
.......
# component call reference to configure Solaris9 RM
- <call blockName="addToProject">
  <installedComponent name="iws60-base" />
  </call>
- <call blockName="startIplanetInstance"> #now start
the Sun ONE Web Server service.
  <installedComponent name="iws60-base" />
  </call>
......
.......
  </inlineSubplan>
  </compositeSteps>
  </executionPlan>
```

Based on this code and the N1 Grid SPS plan schema, the `AddToProject` call block invokes a *routine* in a section of the Sun ONE XML component named `iws60-base`. The routine inside the component is called a control block element inside the N1 Grid SPS. The `AddToProject` call block is a top-level call to Solaris 9 RM provisioning detail in other XML *modules*.

At the time of this call to the control block, the Sun ONE Web Server deployment (binary and configuration files) has just successfully completed on the host, hence the call to the *installed* block `iws60-base`. The call immediately after the `AddToProject` call starts up the *installed* Sun ONE Web Server instance, `StartIplanetInstance`.

Assuming a successful Solaris 9 RM provisioning to the same hosts on which the Sun ONE Web Server are deployed, control returns to this top-level plan to start up the Sun ONE Web Server with CPU share resource tracking, which can be check by using the `# prstat -J` command.

Component Integration

Inside the Sun ONE Web Server component of CODE EXAMPLE 2, there is a section of code called `AddtoProject`, which makes the necessary N1 Grid SPS and Solaris OS calls to implement Solaris 9 RM provisioning in subsequent calls. In the N1 Grid SPS, this call reference is represented in the called component as a control block (inside a `controlList`) in the appropriate section of the referenced component.

In the example implementation, the N1 Grid SPS matches the `iws60-base` plan reference (in CODE EXAMPLE 1) with the control list and control block snippet inside the `iws60-base` component in CODE EXAMPLE 2. Control list actions inside a component are performed in the N1 Grid SPS only if the referenced component has already been successfully deployed to the target hosts in the data center. In this case, there are no issues because before the call in CODE EXAMPLE 1 to provision Solaris 9 RM for the Sun ONE Web Server, there is a successful provisioning step (installation and configuration) for the Sun ONE Web Server to the target host (determined at runtime or built into the deployment). In the example implementation, the actual provisioning details of Sun ONE Web Server have been suppressed.

CODE EXAMPLE 2 shows a fragment of the XML source of the Sun ONE Web Server component (`iws60-base`)—the context for inserting the Solaris 9 RM control list reference called from the plan in CODE EXAMPLE 1. The example implementation shows other functionality of the Sun ONE Web Server deployment in CODE EXAMPLE 2 to illustrate the richness of the Sun ONE Web Server implementation. Several tags are shown before the call to the Solaris OS-specific component called `solaris services`. These tags are called elements, in this case child elements, of the component element inside the N1 Grid SPS.

The highlighted sections in the code examples show the key sections of the XML code with references to other components to issue appropriate commands for creating the Solaris OS accounts for Solaris 9 RM project membership and project creation. CODE EXAMPLE 2 shows a call to the `CreateServiceAccount` control to create the Solaris OS accounts. This call references control block named `create_os_usergroup_nc` (inside the control list) of the Solaris OS services component in CODE EXAMPLE 5. In CODE EXAMPLE 3, the `addToProject` control block prepares a list of arguments (called `argList` in the N1 Grid SPS) for the call to the Solaris OS component called `solaris services` with its own parameter list to accept and process values from the call inside `iws60-base`. The actual values of these arguments are declared in the `Varlist` section of the component or are instantiated at runtime. The values of these arguments are the values for making entries in the `/etc/project` file. These include project IDs, project names, descriptive comments, resource control definitions, and action clauses, as shown in the highlighted sections of CODE EXAMPLE 3.

CODE EXAMPLE 5 also shows how accounts are created on the target host with the appropriate call to the `useradd`(1M) and `groupadd`(1M) commands to populate the `/etc/passwd` and `/etc/group` files as appropriate. This automated version is identical to the standard or manual provisioning discussed earlier.

CODE EXAMPLE 2 Sun ONE Web Server Component Header and Account Creation

```
<?xml version="1.0" encoding="UTF-8" ?>
- <!-- generated by CR
  -->
- <component platform="Solaris - any version"
xmlns="http://www.centerrun.com/schema/CR" name="iws60-
base" version="3.0" description="iPlanet 6.0 base
component" xmlns:xsi="http://www.w3.org/2001/XMLSchemainstance"
xsi:schemaLocation="http://www.centerrun.com/schema/CR
component.xsd">
- <varList>
  <var name="SrvcName" default="premwww01" />
  .....
  </varList>
  .....
- <installList>
  ....
  </installList>
  ....
- <controlList>
- <control name="createServiceAccount" description="Create
account for
  service">
- <call blockName="create_os_usergroup_nc">
  <argList gid=":[IwsGid]" user_home_dir=":[UserHomeDir]"
  uid=":[IwsUid]"
  username=":[IwsUserName]" shell="/usr/bin/bash"
  gname=":[IwsGroupName]" />
  <systemService name="solaris services" />
  </call>
  </control>
  .....
- .....
```

CODE EXAMPLE 3 Sun ONE Web Server Component With Solaris 9 RM Calls

```
  </control>
- <control name="startIplanetInstance"
description="Start the iplanet instance">
- <execNative userToRunAs=":[IwsUserName]">
- <inputText>
- <![CDATA[ :[IWSRoot]/:[https]/start 2>/dev/null
1>/dev/null;
  ]]>
  </inputText>
  <exec cmd="sh" />
  </execNative>
  </control>
</control>
- <control name="addToProject">
- <call blockName="addToProject">
  <argList projID=":[IwsUid]"
projectConfiguration="project.cpu-shares(privileged,50,
deny)" description="Sun One Web Service"
projName="user.:[SrvcName]" userName=":[IwsUserName]" />
  <systemService name="solaris services" />
  </call>
  </control>
```

CODE EXAMPLE 4 completes the `iws60-base` component with a control block to show how the N1 Grid SPS would remove Solaris 9 RM provisioning, passing the appropriate application service name to the Solaris OS service component from a host.

CODE EXAMPLE 4 Ending Code Lines of `iws60-base` XML From CODE EXAMPLE 2

```
- <control name="removeFromProject">
- <call blockName="removeFromProject">
  <argList projName="user.:[SrvcName]" />
  <systemService name="solaris services" />
  </call>
  </control>
</controlList>
</component>
```

Solaris OS Component

A separate Solaris OS component in the Sun ONE service provisioning design is important. Decoupling Solaris OS-specific data center functionality into a separate component from the application-level components makes the solution scalable, reusable, and elegant. CODE EXAMPLE 5, CODE EXAMPLE 6, and CODE EXAMPLE 7 show the relevant fragments of the Solaris OS component. In the control list section of the `Iws60-base` component fragment in the code examples, the references to the Solaris OS underlying component provides low-level Solaris OS services to application-level components for a solution set of components, assembled to address a system management behavior in the data center.

The reference to `systemService name="solaris services"` inside the control blocks of CODE EXAMPLE 2 is the name of the Solaris OS service or component. This service or component is bundled with the N1 Grid SPS software and performs a system service for customer application-level implementations of data center behaviors with the N1 Grid SPS tool.

In the examples, the Sun ONE Web Server component calls control blocks inside the Solaris OS component to create service accounts in the Solaris OS and to provision the Solaris 9 RM for the Sun ONE Web Server, passing appropriate values for populating the project file. The Solaris OS component is the base component with a set of control blocks that might have templates for implementing Solaris OS services such as mounting file systems, removing directories, and configuring network interfaces, in addition to creating accounts and provisioning the Solaris 9 RM.

Within the `create_os_usergroup_nc` control block (inside the control list of the Solaris OS component in CODE EXAMPLE 5) is a parameter list (`paramList`) child element that declares a number of parameters to be used inside the control block to set up and provision the user and group accounts for the application services. Values for these parameters are passed from the higher-level components.

In the code examples, notice the `CDATA` section inside the `<inputText>` block inside the control blocks. The `<inputText>` block in N1 Grid SPS prepares a string of text (which might be a set of commands) to be fed to the shell with the `<exec cmd="/bin/sh" />` call, which is within the `ExecNative` call. In N1 Grid SPS, the `ExecNative` call is a directive to execute the commands (in this case represented inside the `<inputText>` element). By so doing, components and plans can execute native shell commands on hosts where the data center behavior is targeted.

In CODE EXAMPLE 5, notice the `CDATA` section inside the control blocks. In the N1 Grid SPS `CDATA` section preserves formatting, while avoiding XML parsing of the text inside `<inputText>` if these commands also contain special shell commands. The preserved string is then fed to the shell with the `<exec cmd="/bin/sh" />` call.

After completion of the Solaris OS component service call, control returns to the master plan in CODE EXAMPLE 1. At this point, the Sun ONE Web Server service starts with resource control for CPU sharing and begins to monitor the system.

CODE EXAMPLE 5 Solaris OS-Specific Commands to Set Up and Prepare for Accounts

```
- <controlList>
- <control name="create_os_usergroup_nc"
description="Create an OS group and user - no error
checking">
- <paramList>
  <param name="username" prompt="Username" />
  <param name="uid" prompt="User ID" />
  <param name="shell" default="/usr/bin/bash"
prompt="Shell" />
  <param name="user_home_dir" prompt="Home Directory for
User" />
  <param name="gname" prompt="Group Name" />
  <param name="gid" prompt="Group ID" />
  </paramList>
- <execNative userToRunAs="root">
- <inputText>
- <![CDATA[

username=:[username]
uid=:[uid]
gname=:[gname]
shell=:[shell]
home_dir=:[user_home_dir]
comment=:[username]
if [ x`grep :[uid] /etc/passwd` != 'x' -o x`grep
:[username] /etc/passwd` != 'x' ]; then
  echo "ERROR: :[uid] or :[username] already exists in
/etc/passwd";
  exit 1;
fi
if [ x`grep :[gid] /etc/group` != 'x' -o x`grep :[gname]
/etc/group` != 'x' ]; then
  echo "ERROR: :[gid] or :[gname] already exists in
/etc/group";
  exit 1;
fi
/usr/sbin/groupadd -g :[gid] :[gname];

# Set comment to something if its empty
if [ "X$comment" = "X" ] ; then
  comment="$username"
fi
```

CODE EXAMPLE 6 Solaris OS Commands to Create Accounts

```
/usr/bin/mkdir -p $home_dir;

echo /usr/sbin/useradd -c $comment -u $uid -g $gname -d
$home_dir -s $shell -m $username
/usr/sbin/useradd -c $comment -u $uid -g $gname -d
$home_dir -s $shell -m $username
/usr/bin/chown -R $username:$gname $home_dir;

  ]]>
  </inputText>
  <exec cmd="/bin/sh" />
  </execNative>
  </control>
```

CODE EXAMPLE 7 Solaris OS Commands to Populate the Project File

```
</control>
- <control name="addToProject" description="Solaris 9
Resource Management - add an entry to /etc/project">
- <paramList>
  <param name="projName" />
  <param name="projID" />
  <param name="description" />
  <param name="userName" />
  <param name="projectConfiguration" />
  </paramList>
- <execNative userToRunAs="root">
- <inputText>
- <![CDATA[
grep :[projName] /etc/project;
if [ $? != 0 ] ; then
  echo
":[projName]::[projID]::[description]::[userName]:::
[projectConfiguration]" >> /etc/project;
fi

  ]]>
  </inputText>
  <exec cmd="sh" />
  </execNative>
  </control>
```

Resource Management Conclusions

Resource management for mixed workload environment is imperative because IT organizations should be focused on driving costs out of the data center through initiatives that maximize system utilization with stacked applications, automatic deployment, and shared data center infrastructure resources, while gaining service provisioning agility, scaling, and quality. Applications deployed into these environments will contest shared resources to meet the service level expectations. Service agreements could be jeopardized if resource sharing is not controlled.

There are several options for configuring or provisioning Solaris 9 RM to control resource usage in a multiservice application environment where applications are stacked to maximize system resources with competing workloads, while delivering service commitments after moving applications to the stacked environment.

A network service-based database solution (with consideration for policy-based real-time operational threshold and service management targets) offers the ideal solution to the enterprise resource management problem. However, the application provisioning-based resource management solution described in this article represents an interim solutions for IT organizations. Both of these solutions let resource management *follow* the application workload to its point-of-service with location independence, achieving the desired effect.

This article has described how to automate and integrate Solaris 9 RM provisioning inside a larger service provisioning context. However, it also touched on the inner working of service provisioning or automated deployment (`autodeploy`) for application service behaviors or functionality inside the data center.

With the upcoming release of the Solaris 10 OS, you can expect more granular resource control. The combination of Solaris Containers and resource management will offer exciting flexibility and power in attacking service-level management for shared infrastructure environments, which has been elusive.

In addition to more granular resource control for multi-service workloads, the Solaris Containers will deliver new levels of capability and flexibility for two critical applications in IT today: security and consolidation. Being able to host multiple virtual container environments in a single image of the Solaris OS inherently offers software level isolation. Consolidators can distribute, dedicate, and host disparate application service types inside separate software partitions or containers. The application service types are isolated from faults and other operational imperatives. Additionally, the Solaris Containers support localized security profiles separate from the security configuration of the hosting Solaris OS image. Thus, they offer container-hosted applications with customizable security flexibility not seen in enterprise operating environments today.

References

The following reference material was used to create this article:

- Lawson, Stuart J. “Resource Management in the Solaris 9 Operating Environment,” Sun BluePrints Online, September 2002.

Acknowledgements

Many thanks to Stuart Lawson, Global Customer Benchmarking, and Leon Shaner, IT/ Technical Architect and Java System Architect in Sun Client Services, for reviewing this article.

Author Bio

Sam Antwi is currently a Senior Architect with Sun Client Services. Prior to joining Sun in 2000, Sam served in various Lead, Architect, Oracle DBA, and Senior Manager roles at Accenture Ltd. (formerly Andersen Consulting), Deloitte Consulting, and American Airlines Sabre Computing Group. At Sun Microsystems, Sam has interests in delivering customer solutions in Systems Architecture, N1 and utility computing, resource management, grid computing, systems performance tuning, high-end computing systems, high availability readiness services, server consolidation, capacity planning and data center operations. Sam holds a Master's degree in Computer Science from the University of Southern California.

Third-Party URLs

Third-party URLs are referenced in this document and provide additional, related information.

Note – Sun is not responsible for the availability of third-party web sites mentioned in this document. Sun does not endorse and is not responsible or liable for any content, advertising, products, or other materials that are available on or through such sites or resources. Sun will not be responsible or liable for any actual or alleged damage or loss caused by or in connection with the use of or reliance on any such content, goods, or services that are available on or through such sites or resources.

Ordering Sun Documents

The SunDocs^{SM} program provides more than 250 manuals from Sun Microsystems, Inc. If you live in the United States, Canada, Europe, or Japan, you can purchase documentation sets or individual manuals through this program.

Accessing Sun Documentation Online

The `docs.sun.com` web site enables you to access Sun technical documentation online. You can browse the `http://docs.sun.com` archive or search for a specific book title or subject.

To reference Sun BluePrints OnLine articles, visit the Sun BluePrints OnLine web site at `http://www.sun.com/blueprints/`.