AUTHORIZED DOCUMENTATION

Administrator Reference

Novell®
PlateSpin® Orchestrate

2.5

January 27, 2011

www.novell.com

Legal Notices

Novell, Inc. makes no representations or warranties with respect to the contents or use of this documentation, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to revise this publication and to make changes to its content, at any time, without obligation to notify any person or entity of such revisions or changes.

Further, Novell, Inc. makes no representations or warranties with respect to any software, and specifically disclaims any express or implied warranties of merchantability or fitness for any particular purpose. Further, Novell, Inc. reserves the right to make changes to any and all parts of Novell software, at any time, without any obligation to notify any person or entity of such changes.

Any products or technical information provided under this Agreement may be subject to U.S. export controls and the trade laws of other countries. You agree to comply with all export control regulations and to obtain any required licenses or classification to export, re-export or import deliverables. You agree not to export or re-export to entities on the current U.S. export exclusion lists or to any embargoed or terrorist countries as specified in the U.S. export laws. You agree to not use deliverables for prohibited nuclear, missile, or chemical biological weaponry end uses. See the Novell International Trade Services Web page (http://www.novell.com/info/exports/) for more information on exporting Novell software. Novell assumes no responsibility for your failure to obtain any necessary export approvals.

Novell, Inc.
404 Wyman Street, Suite 500
Waltham, MA 02451
U.S.A.
www.novell.com

Online Documentation: To access the latest online documentation for this and other Novell products, see the Novell Documentation Web page (http://www.novell.com/documentation).

Novell Trademarks

For Novell trademarks, see the Novell Trademark and Service Mark list (http://www.novell.com/company/legal/trademarks/tmlist.html).

Third-Party Materials

All third-party trademarks are the property of their respective owners.

Contents

About This Guide

This *Administration Guide* introduces the processes you can use with PlateSpin Orchestrate 2.5, including the applied use of the PlateSpin Orchestrate Development Client and various command line tools. The guide provides an introductory overview of PlateSpin Orchestrate and explains how it administers and manages work on the resources of the data center. The guide is organized as follows:

- Chapter 1, "Basic PlateSpin Orchestrate Concepts," on page 7
- Chapter 2, "Server Discovery and Multicasting," on page 25
- Chapter 3, "PlateSpin Orchestrate and LDAP Authentication," on page 27
- Chapter 4, "Increasing the Kernel ARP Threshold Value on the Orchestrate Server," on page 31
- Appendix A, "PlateSpin Orchestrate Security," on page 33
- Appendix B, "Changing Orchestrate Server Default Parameters and Values," on page 39
- Appendix C, "Documentation Updates," on page 41

For reference information about the Orchestrate Server or the Orchestrate Development Client, see the *PlateSpin Orchestrate 2.5 Development Client Reference*. For information about the Orchestrate Command Line Tools, see *PlateSpin Orchestrate 2.5 Command Line Reference*.

Audience

This book is intended for data center managers and IT or Operations administrators. It assumes that users of the product have the following background:

- General understanding of network operating environments and systems architecture.
- Knowledge of basic UNIX shell commands and text editors.

Feedback

We want to hear your comments and suggestions about this manual and the other documentation included with this product. Please use the User Comments feature at the bottom of each page of the online documentation, or go to www.novell.com/documentation/feedback.html (http://www.novell.com/documentation/feedback.html) and enter your comments there.

Additional Documentation

In addition to this Administration Guide, PlateSpin Orchestrate 2.5 includes the following additional guides that contain valuable information about the product:

- *PlateSpin Orchestrate 2.5 Getting Started Reference*
- *PlateSpin Orchestrate 2.5 Installation and Configuration Guide*
- *PlateSpin Orchestrate 2.5 Upgrade Guide*
- *PlateSpin Orchestrate 2.5 VM Client Guide and Reference*
- *PlateSpin Orchestrate 2.5 Development Client Reference*
- *PlateSpin Orchestrate 2.5 Virtual Machine Management Guide*

- *PlateSpin Orchestrate 2.5 High Availability Configuration Guide*
- *PlateSpin Orchestrate 2.5 Command Line Reference*
- *PlateSpin Orchestrate 2.5 Server Portal Reference*
- *PlateSpin Orchestrate 2.5 Developer Guide and Reference*

Basic PlateSpin Orchestrate Concepts

1

This section contains the followings information:

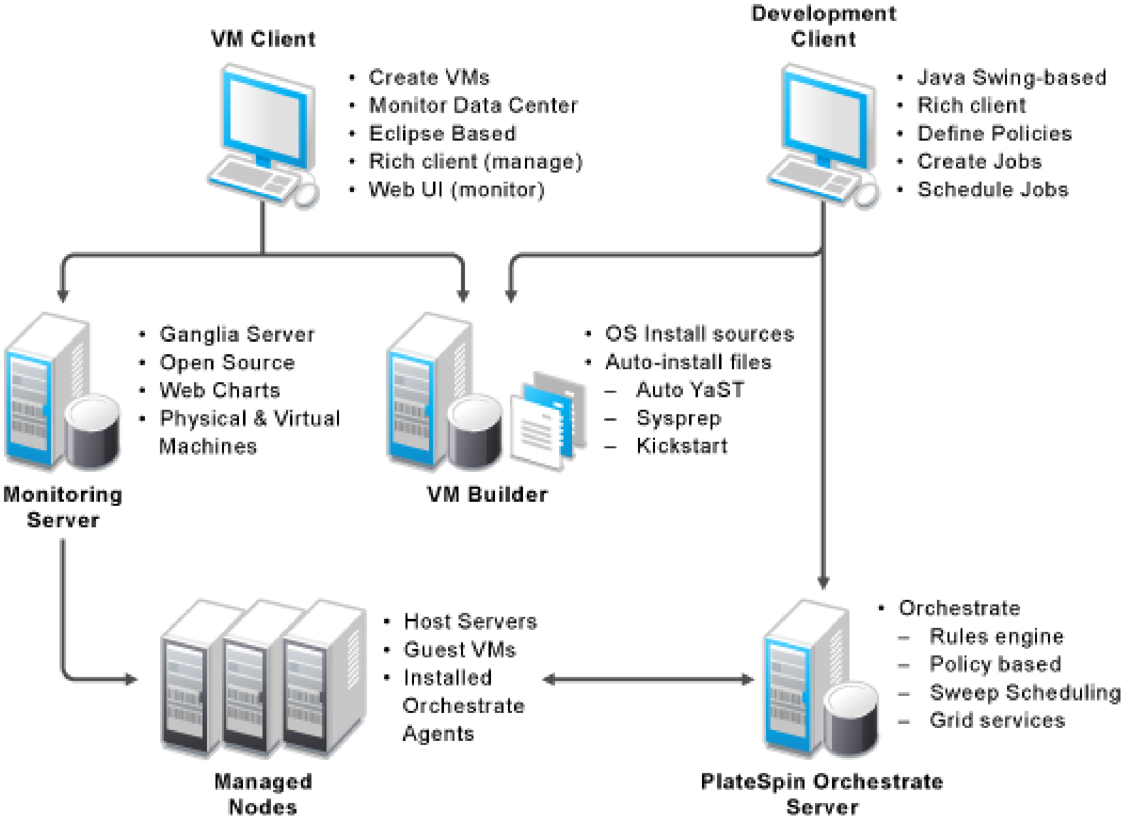
- Section 1.1, “Understanding PlateSpin Orchestrate Architecture,” on page 7
- Section 1.2, “Understanding PlateSpin Orchestrate Functionality,” on page 18

1.1 Understanding PlateSpin Orchestrate Architecture

PlateSpin Orchestrate from Novell is an advanced datacenter management solution designed to manage all network resources. It provides the infrastructure that manages group of ten, one hundred, or thousands of physical or virtual resources.

PlateSpin Orchestrate is equally apt at performing a number of distributed processing problems. From high performance computing, the breaking down of work into lots of small chunks that can be processed in parallel through distributed job scheduling. The following figure shows the product’s high-level architecture:

Figure 1-1 PlateSpin Orchestrate Architecture



This section contains information about the following topics:

- Section 1.1.1, "The PlateSpin Orchestrate Agent," on page 8
- Section 1.1.2, "The Resource Monitor," on page 9
- Section 1.1.3, "Entity Types and Managers," on page 10
- Section 1.1.4, "Jobs," on page 12
- Section 1.1.5, "Constraint-Based Job Scheduling," on page 16
- Section 1.1.6, "Understanding PlateSpin Orchestrate API Interfaces," on page 16

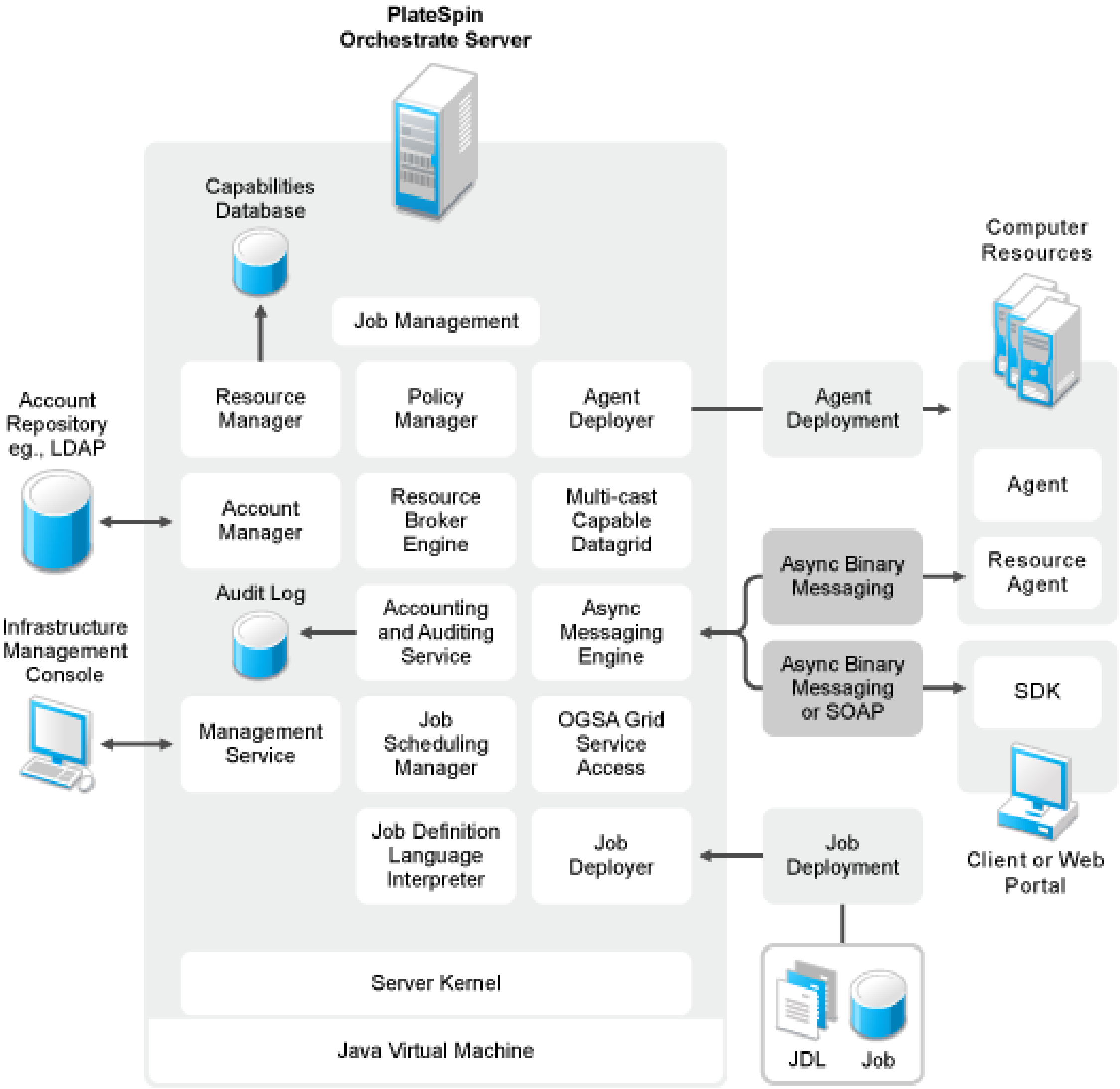
1.1.1 The PlateSpin Orchestrate Agent

Agents are installed on all managed resources as part of the product deployment. The agent connects every managed resource to its configured server and advertises to the PlateSpin Orchestrate Server that the resource is available for tasks. This persistent and auto-reestablishing connection is important because it provides a message bus for the distribution of work, collection of information about the resource, per-job messaging, health checks, and resource failover control.

After resources are enabled, PlateSpin Orchestrate can discover, access, and store detailed abstracted information—called "facts"—about every resource. Managed resources, referred to as "nodes," are addressable members of the Orchestrate Server "grid" (also sometimes called the "matrix"). When integrated into the grid, nodes can be deployed, monitored, and managed by the Orchestrate Server, as discussed in Section 1.2, "Understanding PlateSpin Orchestrate Functionality," on page 18.

An overview of the PlateSpin Orchestrate grid architecture is illustrated in the figure below, much of which is explained in this guide:

Figure 1-2 *PlateSpin Orchestrate Server Architecture*



For additional information about job architecture, see "Job Architecture" in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

1.1.2 The Resource Monitor

PlateSpin Orchestrate enables you to monitor your system computing resources using the built-in Resource Monitor. To open the Resource Monitor in the Development Client, see "Monitoring Server Resources" in the *PlateSpin Orchestrate Administration Guide*.

1.1.3 Entity Types and Managers

The following entities are some of key components involved in the Orchestrate Server:

- "Resources" on page 10
- "Users" on page 10
- "Job Definitions" on page 10
- "Job Instances" on page 11
- "Policies" on page 11
- "Facts" on page 11
- "Constraints" on page 12
- "Groups" on page 12
- "VM: Hosts, Images, and Instances" on page 12
- "Templates" on page 12

Resources

All managed resources, which are called nodes, have an agent with a socket connection to the Orchestrate Server. All resource use is metered, controlled, and audited by the Orchestrate Server. Policies govern the use of resources.

PlateSpin Orchestrate allocates resources by reacting as load is increased on a resource. As soon as we go above a threshold that was set in a policy, a new resource is allocated and consequently the load on that resource drops to an acceptable rate.

You can also write and jobs that perform cost accounting to account for the cost of a resource up through the job hierarchy, periodically, about every 20 seconds. For more information, see "Auditing and Accounting Jobs" on page 15.

A collection of jobs, all under the same hierarchy, can cooperate with each other so that when one job offers to give up a resource it is reallocated to another similar priority job. Similarly, when a higher priority job becomes overloaded and is waiting on a resource, the system "steals" a resource from a lower priority job, thus increasing load on the low priority job and allocating it to the higher priority job. This process satisfies the policy, which specifies that a higher priority job must complete at the expense of a low priority job.

Users

PlateSpin Orchestrate users must authenticate to access the system. Access and use of system resources are governed by policies. For more information, see "The User Object" in the *PlateSpin Orchestrate 2.5 Development Client Reference*.

Job Definitions

A job definition is described in the embedded enhanced Python script that you create as a job developer. Each job instance runs a job that is defined by the Job Definition Language (JDL). Job definitions might also contain usage policies. For more information, see "Job Class" in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

Job Instances

Jobs are instantiated at runtime from job definitions that inherit policies from the entire context of the job (such as users, job definitions, resources, or groups). For more information, see “JobInfo” in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

Policies

Policies are XML documents that contain various constraints and static fact assignments that govern how jobs run in the PlateSpin Orchestrate environment.

Policies are used to enforce quotas, job queuing, resource restrictions, permissions, and other job parameters. Policies can be associated with any PlateSpin Orchestrate object. For more information, see Section 1.2.2, “Policy-Based Management,” on page 19 and “Policies” in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

Facts

Facts represent the state of any object in the PlateSpin Orchestrate grid. They can be discovered through a job or they can be explicitly set.

Facts control the behavior a job (or joblet) when it’s executing. Facts also detect and return information about that job in various UIs and server functions. For example, a job description that is set through its policy and has a specified value might do absolutely nothing except return immediately after network latency.

The XML fact element defines a fact to be stored in the Grid object’s fact namespace. The name, type and value of the fact are specified as attributes. For list or array fact types, the element tag defines list or array members. For dictionary fact types, the dict tag defines dictionary members.

Facts can also be created and modified in JDL and in the Java Client SDK. For more information, see “Using Facts in Job Scripts” in the “Job Development Concepts” section of the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

There are three basic types of facts:

- **Static:** Facts that require you to set a value. For example, in a policy, you might set a value to be False. Static facts can be modified through policies.
- **Dynamic:** Facts produced by the PlateSpin Orchestrate system itself. Policies cannot override dynamic facts. They are read only and their value is determined by the PlateSpin Orchestrate Server itself.
- **Computed:** Facts derived from a value, like that generated from the cell of a spreadsheet. Computed facts have some kind of logic behind them which derive their values. For more information, see “Computed Facts” in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

See the example, `/opt/novell/zenworks/zos/server/examples/allTypes.policy`. This example policy has an XML representation for all the fact types. For a comprehsive list of facts and fact junctions used in PlateSpin Orchestrate.s, see “Grid Object Facts and Fact Junctions” in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

Constraints

The constraint element of a policy can define the selection and allocation of Grid objects (such as resources) in a job. The required type attribute of a constraint defines the selection of the resource type.

For example, in order for the PlateSpin Orchestrate to choose resources for a job, it uses a “resource” constraint type. A resource constraint consists of Boolean logic that executes against facts in the system. Based upon this evaluation, Orchestrate considers only resources that match the criteria that have been defined in constraints.

For more information about the types of constraints and how they are applied in jobs, see “The Role of Policy Constraints in Job Operation” in the “Job Development Concepts” section of the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

Groups

Resources, users, job definitions and virtual machines (VM) are managed in groups with group policies that are inherited by members of the group.

VM: Hosts, Images, and Instances

A virtual machine host is a resource that is able to run guest operating systems. Attributes (facts) associated with the VM host control its limitations and functionality within the Orchestrate Server. A VM image is a resource image that can be cloned and/or provisioned. A VM instance represents a running copy of a VM image.

Templates

Templates are images that are meant to be cloned (copied) prior to provisioning the new copy. For more information, see “Creating a Template from a VM” in the *PlateSpin Orchestrate 2.5 VM Client Guide and Reference*.

1.1.4 Jobs

The Orchestrate Server manages all nodes by administering jobs (and the functional control of jobs at the resource level by using joblets), which control the properties (facts) associated with every resource. In other words, jobs are units of functionality that dispatch data center tasks to resources on the network such as management, migration, monitoring, load balancing, etc.

PlateSpin Orchestrate provides a unique job development, debugging, and deployment environment that expands with the demands of growing data centers.

As a job developer, your task is to develop jobs to perform a wide array of work that can be deployed and managed by PlateSpin Orchestrate.

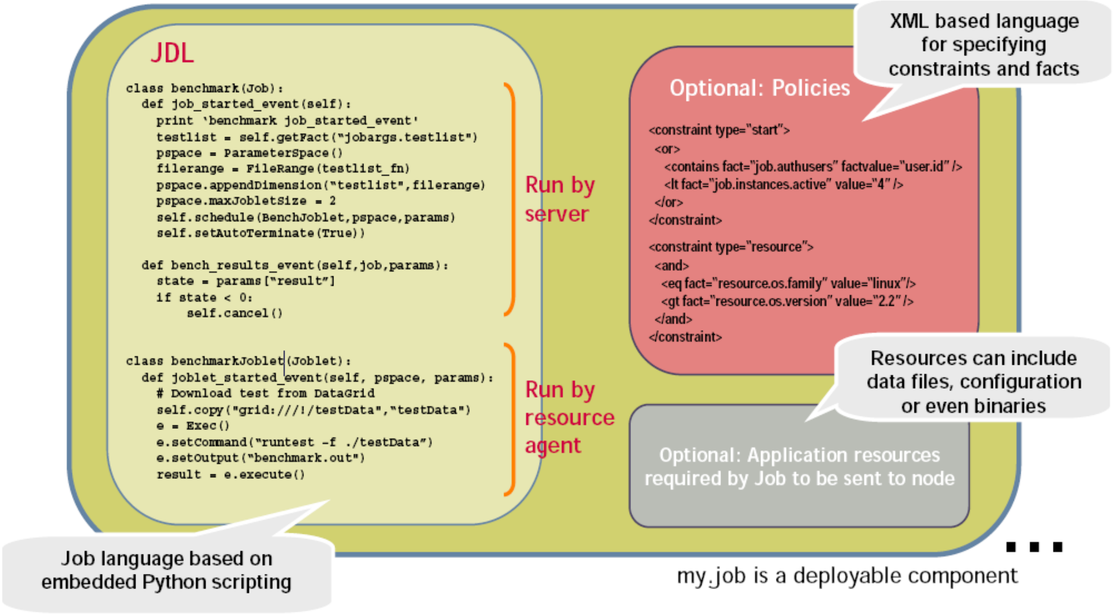
Jobs, which run on the Orchestrate Server, can provide functions within the PlateSpin Orchestrate environment that might last from seconds to months. Job and joblet code exist in the same script file and are identified by the `.jdl` extension. The `.jdl` script contains only one job definition and zero or more joblet definitions. A `.jdl` script can have only one Job subclass. As for naming conventions, the Job subclass name does not have to match the `.jdl` filename; however, the `.jdl` filename is the defined job name, so the `.jdl` filename must match the `.job` filename that contains the `.jdl` script. For example, the job files (`demoIterator.jdl` and `demoIterator.policy`) included in the `demoIterator` example job are packaged into the archive file named `demoIterator.job`, so in this case, the name of the job is `demoIterator`.

A job file also might have policies associated with it to define and control the job's behavior and to define certain constraints to restrict its execution. A `.jdl` script that is accompanied by a policy file is typically packaged in a job archive file (`.job`). Because a `.job` file is physically equivalent to a Java archive file (`.jar`), you can use the JDK JAR tool to create the job archive.

Multiple job archives can be delivered as a management pack in a service archive file (SAR) identified with the .sar extension. Typically, a group of related files are delivered this way. For example, the Xen30 management pack is a SAR.

As shown in the following illustration, jobs include all of the code, policy, and data elements necessary to execute specific, predetermined tasks administered either through the PlateSpin Orchestrate Development Client, or from the zos command line tool.

Figure 1-3 *Components of a Job (*`my.job`*,)*



JDL
Run by server
Run by resource agent
Optional: Policies
XML based language for specifying constraints and facts
Resources can include data files, configuration or even binaries
Optional: Application resources required by Job to be sent to node
Job language based on embedded Python scripting
my.job is a deployable component
...

```
class benchmark(Job):
  def job_started_event(self):
    print 'benchmark job_started_event'
    testlist = self.getFact("jobargs.testlist")
    pspace = ParameterSpace()
    filerange = FileRange(testlist_fn)
    pspace.appendDimension("testlist",filerange)
    pspace.maxJobletSize = 2
    self.schedule(BenchJoblet,pspace,params)
    self.setAutoTerminate(True))

  def bench_results_event(self,job,params):
    state = params["result"]
    if state < 0:
        self.cancel()


class benchmarkJoblet(Joblet):
  def joblet_started_event(self, pspace, params):
    # Download test from DataGrid
    self.copy("grid:///!/testData","testData")
    e = Exec()
    e.setCommand("runtest -f ./testData")
    e.setOutput("benchmark.out")
    result = e.execute()
```

```
<constraint type="start">
 <or>
   <contains fact="job.authusers" factvalue="user.id" />
   <lt fact="job.instances.active" value="4" />
 </or>
</constraint>

<constraint type="resource">
 <and>
  <eq fact="resource.os.family" value="linux"/>
  <gt fact="resource.os.version" value="2.2" />
 </and>
</constraint>
```

Because each job has specific, predefined elements, jobs can be scripted and delivered to any agent, which ultimately can lead to automating almost any datacenter task. Jobs provide the following functionality:

- "Controlling Process Flow" on page 14
- "Parallel Processing" on page 14
- "Managing the Cluster Life Cycle" on page 14
- "Discovery Jobs" on page 14
- "System Jobs" on page 15
- "Provisioning Jobs" on page 15
- "Auditing and Accounting Jobs" on page 15

For more information, see "Job Development Concepts" in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference* and the following JDL job class definitions in the same guide:

- "Job"
- "JobInfo"

Controlling Process Flow

Jobs can written to control all operations and processes of managed resources. Through jobs, the Orchestrate Server manages resources to perform work. Automated jobs (written in JDL), are broken down into joblets, which are distributed among multiple resources.

Parallel Processing

By managing many small joblets, the Orchestrate Server can enhance system performance and maximize resource use.

Managing the Cluster Life Cycle

Jobs can detect demand and monitor health of system resources, then modify clusters automatically to maximize system performance and provide failover services.

Discovery Jobs

Some jobs provide inspection of resources to more effectively management assets. These jobs enable all agents to periodically report basic resource facts and performance metrics. In essence, these metrics are stored as facts consisting of a key word and typed-value pairs like the following example:

```
resource.loadaverage=4.563, type=float
```

Jobs can poll resources and automatically trigger other jobs if resource performance values reach certain levels.

The system job scheduler is used to run resource discovery jobs to augment resource facts as demands change on resources. This can be done on a routine, scheduled basis or whenever new resources are provisioned, new software is installed, bandwidth changes occur, OS patches are deployed, or other events occur that might impact the system.

Consequently, resource facts form a capabilities database for the entire system. Jobs can be written that apply constraints to facts in policies, thus providing very granular control of all resources as required. All active resources are searchable and records are retained for all off-line resources.

The following `osInfo.job` example shows how a job sets operating system facts for specific resources:

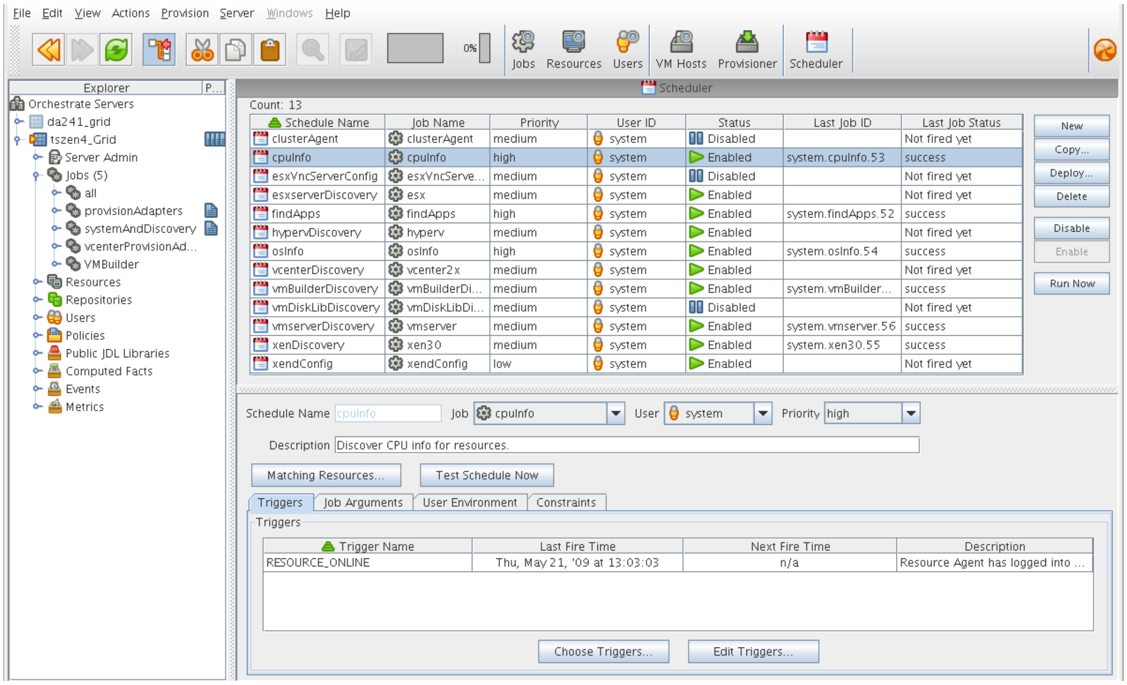
```
resource.cpu.mhz (integer) e.g., "800" (in Mhz)
    resource.cpy.vendor (string) e.g. "GenuineIntel"
    resource.cpu.model (string) e.g. "Pentium III"
    resource.cpu.family (string) e.g. "i686"
```

`osInfo.job` is packaged as a single cross-platform job and includes the Python-based JDL and a policy to set the timeout. It is run each time a new resource appears and once every 24 hours to ensure validity of the resources. For a more detailed review of this example, see "osInfo.job" in "Job Development Concepts" in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

System Jobs

Jobs can be scheduled to to periodically trigger specific system resources based on specific time constraints or events. As shown in the following figure, PlateSpin Orchestrate provides a built-in job scheduler that enables you or system administrators to flexibly deploy and run jobs.

Figure 1-4 *The Job Scheduler*



Count: 13

Schedule Name	Job Name	Priority	User ID	Status	Last Job ID	Last Job Status
clusterAgent	clusterAgent	medium	system	Disabled		Not fired yet
cpuInfo	cpuInfo	high	system	Enabled	system.cpuInfo.53	success
esxVncServerConfig	esxVncServe...	medium	system	Disabled		Not fired yet
esxserverDiscovery	esx	medium	system	Enabled		Not fired yet
findApps	findApps	high	system	Enabled	system.findApps.52	success
hypervDiscovery	hyperv	medium	system	Enabled		Not fired yet
osInfo	osInfo	high	system	Enabled	system.osInfo.54	success
vcenterDiscovery	vcenter2x	medium	system	Enabled		Not fired yet
vmBuilderDiscovery	vmBuilderDi...	medium	system	Enabled	system.vmBuilder...	success
vmDiskLibDiscovery	vmDiskLibDi...	medium	system	Disabled		Not fired yet
vmserverDiscovery	vmserver	medium	system	Enabled	system.vmserver.56	success
xenDiscovery	xen30	medium	system	Enabled	system.xen30.55	success
xendConfig	xendConfig	low	system	Enabled		Not fired yet

Schedule Name: cpuInfo Job: cpuInfo User: system Priority: high

Description: Discover CPU info for resources.

Triggers

Trigger Name	Last Fire Time	Next Fire Time	Description
RESOURCE_ONLINE	Thu, May 21, '09 at 13:03:03	n/a	Resource Agent has logged into ...

For more information, see “How Constraints Are Used” in “Job Development Concepts” in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference* and “The PlateSpin Orchestrate Job Scheduler” in the *PlateSpin Orchestrate 2.5 Development Client Reference*. See also “Job Scheduling” and “Job” in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

Provisioning Jobs

Jobs also drive provisioning for virtual machines (VMs) and physical machines, such as blade servers. Provisioning adapter jobs for various VM hypervisors are deployed and organized into appropriate job groups for management convenience.

The provisioning jobs included in PlateSpin Orchestrate are used for interacting with VM hosts and repositories for VM life cycle management and for cloning, moving VMs, and other management tasks. These jobs are called “provisioning adapters” and are members of the job group called “provisionAdapters.”

For more information, see the *PlateSpin Orchestrate 2.5 Virtual Machine Management Guide* and Section 1.2.1, “Resource Virtualization,” on page 18 of this guide.

Auditing and Accounting Jobs

You can create PlateSpin Orchestrate jobs that perform reporting, auditing, and costing functions inside your data center. Your jobs can aggregate cost accounting for assigned resources and perform resource audit trails.

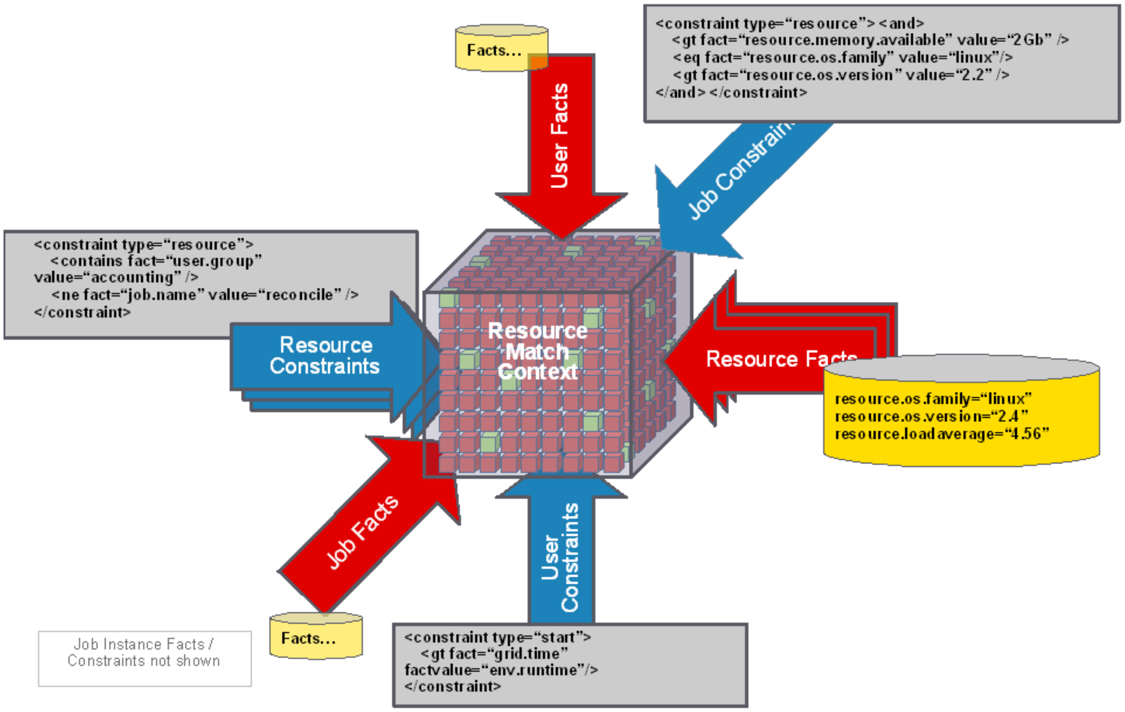
1.1.5 Constraint-Based Job Scheduling

The Orchestrate Server is a “broker” that can distribute jobs to every “partner” agent on the grid. Based on assigned policies, jobs have priorities and are executed based on the following contexts:

- User Constraints
- User Facts
- Job Constraints
- Job Facts
- Job Instance
- Resource User Constraints
- Resource Facts
- Groups

Each object in a job context contains the following elements:

Figure 1-5 *Constraint-Based Resource Brokering*



```
<constraint type=“resource”> <and>
   <gt fact=“resource.memory.available” value=“2Gb” />
   <eq fact=“resource.os.family” value=“linux”/>
   <gt fact=“resource.os.version” value=“2.2” />
</and> </constraint>
```

```
<constraint type=“resource”>
   <contains fact=“user.group”
value=“accounting” />
   <ne fact=“job.name” value=“reconcile” />
</constraint>
```

```
resource.os.family=“linux”
resource.os.version=“2.4”
resource.loadaverage=“4.56”
```

```
<constraint type=“start”>
   <gt fact=“grid.time”
factvalue=“env.runtime”/>
</constraint>
```

For more information, see “Scheduling with Constraints”in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

1.1.6 Understanding PlateSpin Orchestrate API Interfaces

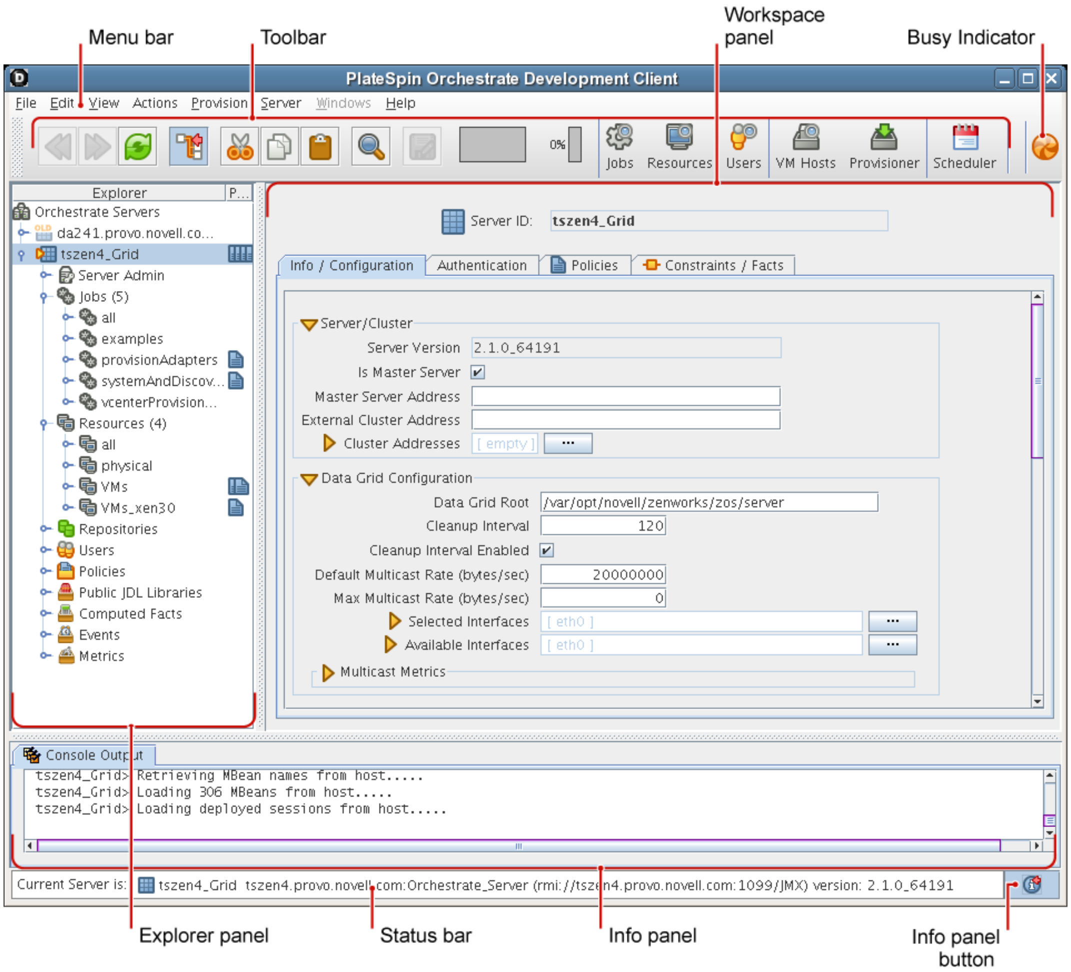
There are three API interfaces available to the Orchestrate Server:

- **Orchestrate Server Management Interface:** The PlateSpin Orchestrate Server, written entirely in Java using the JMX (Java MBean) interface for management, leverages this API for the PlateSpin Orchestrate Development Client. The Development Client is a robust desktop

GUI designed for administrators to apply, manage, and monitor usage-based policies on all infrastructure resources. The Development Client also provides at-a-glance grid health and capacity checks.

For more information, see the *PlateSpin Orchestrate 2.5 Development Client Reference*.

Figure 1-6 *PlateSpin Orchestrate Development Client*



Server ID: tszen4_Grid

Server Version 2.1.0_64191
Is Master Server ☑
Master Server Address
External Cluster Address
Cluster Addresses [empty]

Data Grid Root /var/opt/novell/zenworks/zos/server
Cleanup Interval 120
Cleanup Interval Enabled ☑
Default Multicast Rate (bytes/sec) 20000000
Max Multicast Rate (bytes/sec) 0
Selected Interfaces [eth0]
Available Interfaces [eth0]

```
tszen4_Grid> Retrieving MBean names from host.....
tszen4_Grid> Loading 306 MBeans from host.....
tszen4_Grid> Loading deployed sessions from host.....
```

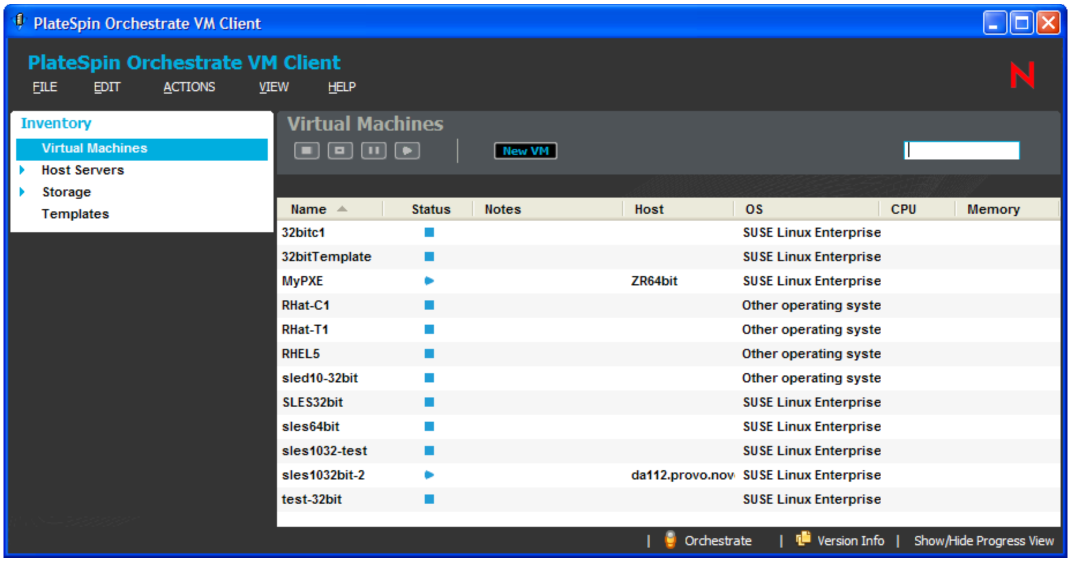
Current Server is: tszen4_Grid tszen4.provo.novell.com:Orchestrate_Server (rmi://tszen4.provo.novell.com:1099/JMX) version: 2.1.0_64191

- **Job Interface:** Includes a customizable/replaceable Web application and the zosadmin command line tool. The Web-based Server Portal built with this API provides a universal job viewer from which job logs and progress can be monitored. The job interface is accessible via a Java API or CLI. A subset is also available as a Web Service. The default PlateSpin Orchestrate Server Portal leverages this API. It can be customized or alternative J2EE* application can be written.
- **PlateSpin Orchestrate Monitoring System:** Monitors all aspects of the data center through an open source, Eclipse-based interrface referred to as the PlateSpin Orchestrate VM Client.This interface operates in conjunction with the Orchestrate Server and monitors the following objects:
    - Deployed jobs that teach PlateSpin Orchestrate and provide the control logic that PlateSpin Orchestrate runs when performing its management tasks.

- Users and Groups
- Virtual Machines

For more information, see the *PlateSpin Orchestrate 2.5 VM Client Guide and Reference*.

Figure 1-7 *The PlateSpin Orchestrate VM Client Interface*



Name	Status	Notes	Host	OS	CPU	Memory
32bitc1	■			SUSE Linux Enterprise		
32bitTemplate	■			SUSE Linux Enterprise		
MyPXE	▶		ZR64bit	SUSE Linux Enterprise		
RHat-C1	■			Other operating syste		
RHat-T1	■			Other operating syste		
RHEL5	■			Other operating syste		
sled10-32bit	■			Other operating syste		
SLES32bit	■			SUSE Linux Enterprise		
sles64bit	■			SUSE Linux Enterprise		
sles1032-test	■			SUSE Linux Enterprise		
sles1032bit-2	▶		da112.provo.nov	SUSE Linux Enterprise		
test-32bit	■			SUSE Linux Enterprise		

1.2 Understanding PlateSpin Orchestrate Functionality

- Section 1.2.1, "Resource Virtualization," on page 18
- Section 1.2.2, "Policy-Based Management," on page 19
- Section 1.2.3, "Grid Object Visualization," on page 20
- Section 1.2.4, "Understanding Job Semantics," on page 20
- Section 1.2.5, "Distributed Messaging and Failover," on page 21
- Section 1.2.6, "Web-Based User Interaction," on page 22

1.2.1 Resource Virtualization

Host machines or test targets managed by the Orchestrate Server form nodes on the grid (sometimes referred to as the matrix). All resources are virtualized for access by maintaining a capabilities database containing extensive information (facts) for each managed resource.

This information is automatically polled and obtained from each resource periodically or when it first comes online. The extent of the resource information the system can gather is customizeable and highly extensible, controlled by the jobs you create and deploy.

Past releases of PlateSpin Orchestrate used some SBLIM CIM providers to build Xen VMs. The xen provisioning adapter in this release has been re-implemented to use the vm-install binary directly. This deprecates the VM Builder CIM providers and pattern.

With this new functionality, any Xen host with access to the vm-install binary (`/usr/sbin/vm-install`) can build VMs.The binary is packaged in the `vm-install` RPM on SLES 11. This binary was packaged in the `xen-tools` RPM on SLES10. Both RPMs are contained on the SLES media, and unless you explicitly omit them during the installation, they are available as part of a default XEN installation.

For more information, see "Creating a Xen VM" in the *PlateSpin Orchestrate 2.5 VM Client Guide and Reference*.

1.2.2 Policy-Based Management

Policies are aggregations of facts and constraints that are used to enforce quotas, job queuing, resource restrictions, permissions, and other user and resource functions. Policies can be set on all objects and are inherited, which facilitates implementation within related resources.

Facts, which might be static, dynamic or computed for complex logic, are used when jobs or test scenarios require resources in order to select a resource that exactly matches the requirements of the test, and to control the access and assignment of resources to particular jobs, users, projects, etc. through policies.This abstraction keeps the infrastructure fluid and allows for easy resource substitution.

Of course, direct named access is also possible. An example of a policy that constrains the selection of a resource for a particular job or test is shown in the sample below. Although resource constraints can be applied at the policy level, they can also be described by the job itself or even dynamically composed at runtime.

```
<policy>
  <constraint type="resource">
    <and>
      <eq fact="resource.os.family" value="Linux"/>
      <gt fact="resource.os.version" value="2.2" />
    <and>
  </constraint>
</policy>
```

An example of a policy that constrains the start of a job or test because too many tests are already in progress is shown in the following sample:

```
<policy>
  <!-- Constrains the job to limit the number of running jobs to a
defined value but exempt certain users from this limit. All jobs
that attempt to exceed the limit are qued until the running jobs
count decreases and the constraint passes. -->

<constraint type="start" reason="Too busy">
    <or>
      <lt fact="job.instances.active" value="5"/>
      <eq fact="user.name" value="canary" />
    </or>
  </constraint>
</policy>
```

For more information about policies and constraints, see "Policies" in the "Job Development Concepts" section of the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

1.2.3 Grid Object Visualization

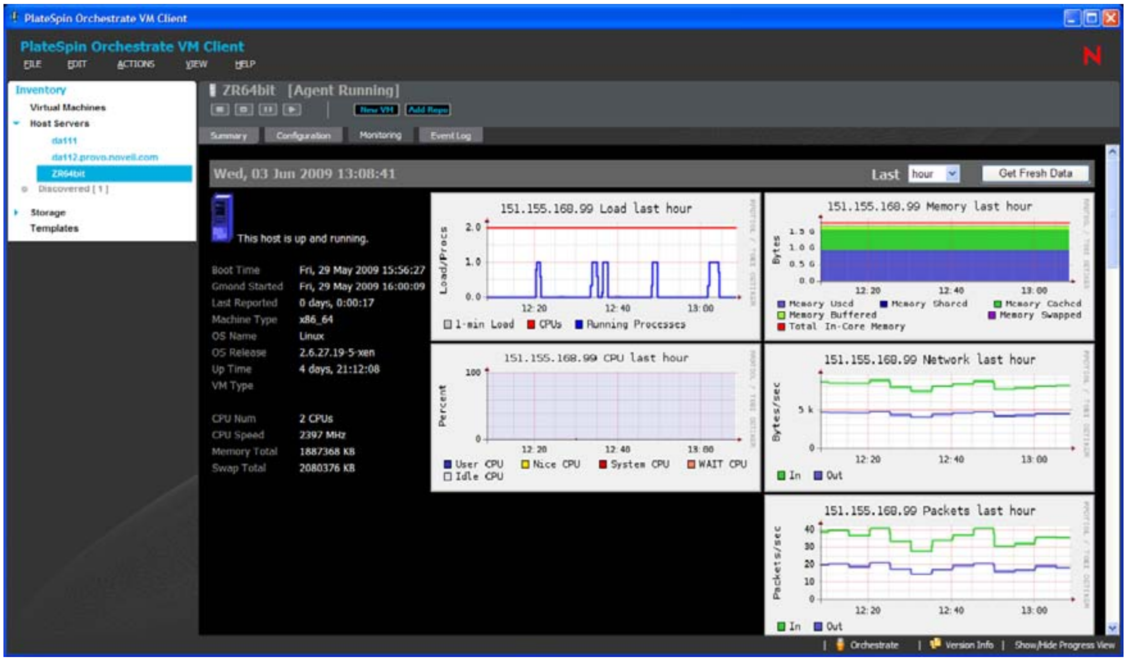
One of the greatest strengths of the PlateSpin Orchestrate solution is the ability to manage and visualize the entire grid. This is performed through the PlateSpin Orchestrate Development Clientand the PlateSpin Orchestrate VM Monitoring System.

The desktop Development Client is a Java application that has broad platform support and provides job, resource, and user views of activity as well as access to the historical audit database system, cost accounting, and other graphing features.

The Development Client also applies policies that govern the use of shared infrastructure or simply create logical grouping of nodes on the grid. For more information about the PlateSpin Orchestrate Development Client, see the *PlateSpin Orchestrate 2.5 Development Client Reference*.

The PlateSpine Orchestrate VM Monitoring System provides robust graphical monitoring of all managed virtual resources managed on the grid.

Figure 1-8 *PlateSpin Orchestrate Monitoring in the VM Client*



ZR64bit [Agent Running]

Wed, 03 Jun 2009 13:08:41

This host is up and running.

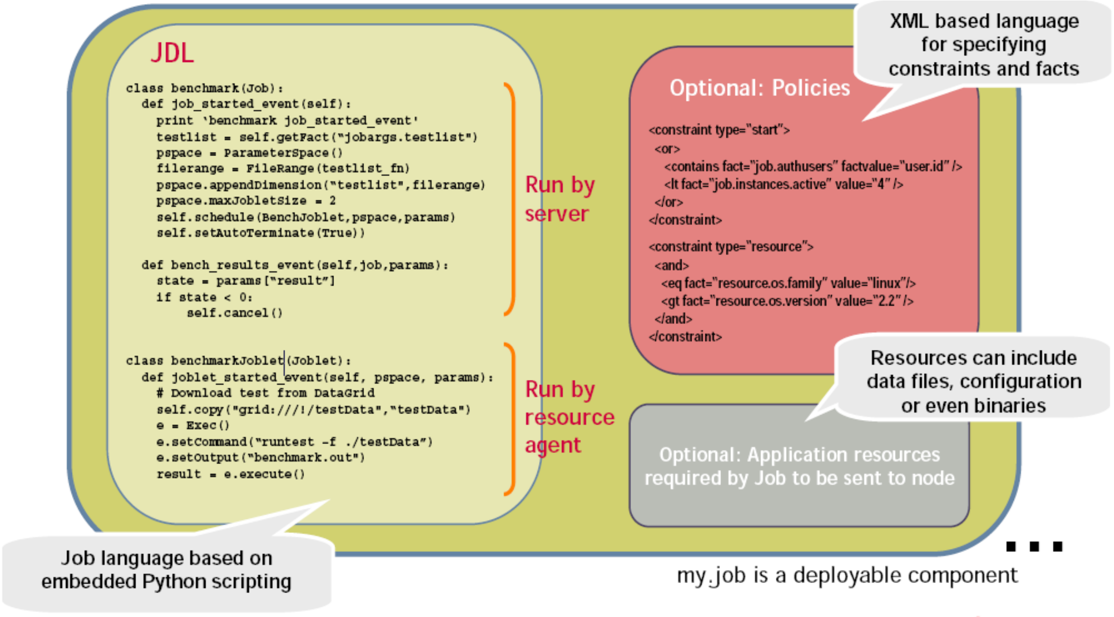
Boot Time	Fri, 29 May 2009 15:56:27
Gmond Started	Fri, 29 May 2009 16:00:09
Last Reported	0 days, 0:00:17
Machine Type	x86_64
OS Name	Linux
OS Release	2.6.27.19-5-xen
Up Time	4 days, 21:12:08
VM Type	
CPU Num	2 CPUs
CPU Speed	2397 MHz
Memory Total	1887368 KB
Swap Total	2080376 KB

For more information, see the *PlateSpin Orchestrate 2.5 VM Client Guide and Reference*.

1.2.4 Understanding Job Semantics

As mentioned earlier, PlateSpin Orchestrate runs jobs. A job is a container that can encapsulate several components including the Python-based logic for controlling the job life cycle (such as a test) through logic that accompanies any remote activity, task-related resources such as configuration files, binaries and any policies that should be associated with the job, as illustrated below.

Figure 1-9 Components of a Job



```
class benchmark(Job):
  def job_started_event(self):
    print 'benchmark job_started_event'
    testlist = self.getFact("jobargs.testlist")
    pspace = ParameterSpace()
    filerange = FileRange(testlist_fn)
    pspace.appendDimension("testlist",filerange)
    pspace.maxJobletSize = 2
    self.schedule(BenchJoblet,pspace,params)
    self.setAutoTerminate(True))

  def bench_results_event(self,job,params):
    state = params["result"]
    if state < 0:
        self.cancel()


class benchmarkJoblet(Joblet):
  def joblet_started_event(self, pspace, params):
    # Download test from DataGrid
    self.copy("grid:///!/testData","testData")
    e = Exec()
    e.setCommand("runtest -f ./testData")
    e.setOutput("benchmark.out")
    result = e.execute()
```

```
<constraint type="start">
 <or>
   <contains fact="job.authusers" factvalue="user.id" />
   <lt fact="job.instances.active" value="4" />
 </or>
</constraint>

<constraint type="resource">
 <and>
  <eq fact="resource.os.family" value="linux"/>
  <gt fact="resource.os.version" value="2.2" />
 </and>
</constraint>
```

Resources can include data files, configuration or even binaries
Optional: Application resources required by Job to be sent to node
Job language based on embedded Python scripting
my.job is a deployable component
...

Workflows

Jobs can also invoke other jobs, creating hierarchies. Because of the communication between the job client (either a user/user client application or another job) it is easy to create complex workflows composed of discrete and separately versioned components.

When a job is executed and an instance is created, the class that extends job is run on the server and as that logic requests resources, the class(es) that extend the joblet are automatically shipped to the requested resource to manage the remote task. The communication mechanism between these distributed components manifests itself as event method calls on the corresponding piece.

For more information, see "Workflow Job Example" in "Job Development Concepts", and "Job State Transition Events", or "Communicating Through Job Events" in "Job Architecture" in the *PlateSpin Orchestrate 2.5 Developer Guide and Reference*.

1.2.5 Distributed Messaging and Failover

A job has control over all aspects of its failover semantics, which can be specified separately for conditions such as the loss of a resource, failure of an individual joblet, or joblet timeout.

The failover/health check mechanisms leverage the same communications mechanism that is available to job and joblet logic. Specifically, when a job is started and resources are employed, a message interface is established among all the components as shown in Figure 1-10 on page 22.

Optionally, a communication channel can also be kept open to the initiating client. This client communication channel can be closed and reopened later based on jobid. Messages can be sent with the command

```
sendEvent(foo_event, params, ...)
```

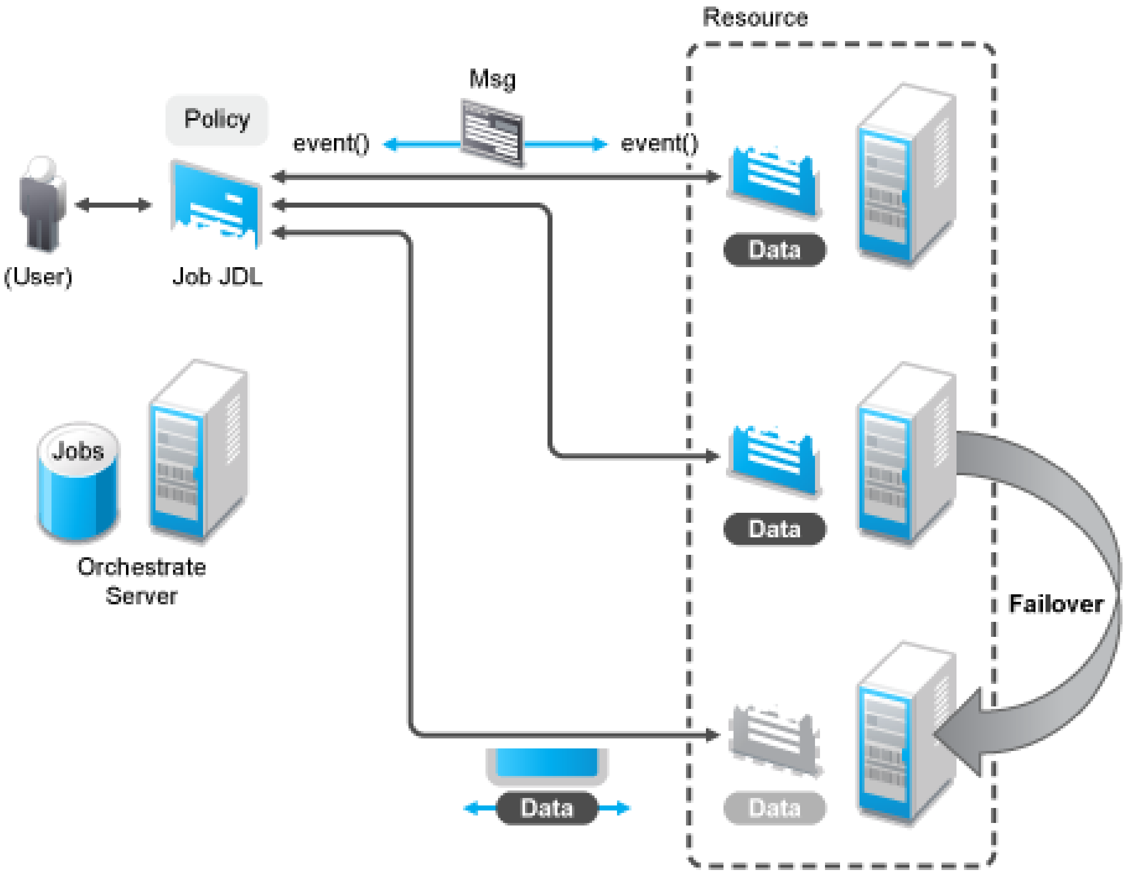
and received at the other end as a method invocation

```
def foo_event(self, params)
```

If a job allows it, a failure in any joblet causes the Orchestrate Server to automatically find an alternative resource, copy over the joblet JDL code, and reestablish the communication connection. A job also can listen for such conditions simply by defining a method for one of the internally generated events, such as def joblet_failure_event(...).

Such failover allows, for example, for a large set of regression tests to be run (perhaps in parallel) and for a resource to die in the middle of the tests without the test run being rendered invalid. The figure below shows how job logic is distributed and failover achieved:

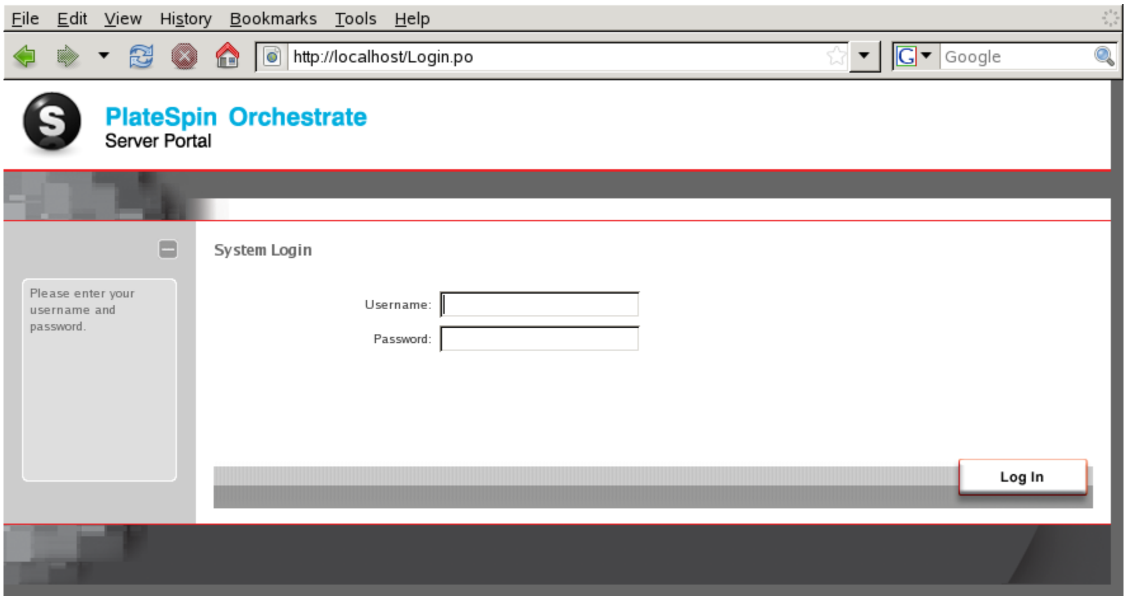
Figure 1-10 *A Job in Action*



1.2.6 Web-Based User Interaction

PlateSpin Orchestrate ships a universal job monitoring and submission interface as a Web application that natively runs on the Orchestrate Server. This application is written to the PlateSpin Orchestrate job management API and can be customized or replaced with alternative rendering as required.The figure belows shows an example of this interface, called the *Server Portal*.

Figure 1-11 PlateSpin Orchestrate Server Portal



http://localhost/Login.po

System Login

Please enter your username and password.

Username:

Password:

For more information, see the *PlateSpin Orchestrate 2.5 Server Portal Reference*.

Server Discovery and Multicasting 2

The PlateSpin Orchestrate Server, Orchestrate Agent, and other Orchestrate tools use IP multicast messages to locate servers and to announce when servers are started or shut down. If multicasting is not supported in your existing network environment, all PlateSpin Orchestrate components allow a specific machine to be specified instead of using multicast discovery. Multicast support is not required to run the PlateSpin Orchestrate Server, the Orchestrate Agent, or any Orchestrate tools.

This section includes the following multicast information:

- Section 2.1, "Multicast Troubleshooting," on page 25
- Section 2.2, "Multicast Routes," on page 25
- Section 2.3, "Multi-homed Hosts," on page 26
- Section 2.4, "Multiple Subnets," on page 26
- Section 2.5, "Datagrid and Multicasting," on page 26
- Section 2.6, "Datagrid Multicast Interface Selection," on page 26

2.1 Multicast Troubleshooting

An exhaustive tutorial of IP multicasting and troubleshooting is beyond the scope of this document. If you are having problems with multicast discovery, ensure that your operating system is configured to provide IP multicasting support. Most modern versions of Linux* and Windows* provide this support, however it might be disabled. Multicast discovery does not work unless IP multicasting is enabled by the operating system. Routing misconfiguration on the system can also lead to problems with multicast discovery.

2.2 Multicast Routes

A common problem with multicasting, particularly on Linux, is the lack of a default route or multicast network route. Most systems are configured to have at least a "default" route, and on such systems, multicast messages use the default route like any other network traffic. Systems do not necessarily require a default route. Multicasting might not function correctly on systems that lack a default route. Attempts to send messages on such systems fail with a `Network Unreachable` message because the operating system is unable to determine the correct network interface on which to send the message.

The quick solution is to add a default route on such systems. In some environments, however, it might not make sense to add a default route. In such cases, another solution is to add a network route for the 224.0.0.0/4 block representing the multicast IP address space. On Linux, for example, issue the following command as the `root` user:

```
route add -net 224.0.0.0 netmask 240.0.0.0 dev eth0
```

This command tells the system to send all multicast datagrams to the `eth0` network card by default. Substitute `eth0` with a different interface name if applicable.

2.3 Multi-homed Hosts

A multi-homed host is a machine with more than one network interface configured. This can be anything from a Linux system being used as a network router to a laptop computer with both an active Ethernet connection and an active wireless connection. If there are two or more network interfaces active at the same time (even if only one is actually being used) the system is “multi-homed.”

Some operating systems like Linux provide only very rudimentary routing as a default part of the operating system. They rely on external routing software like “mrouted” to support full multicast routing. As a result, problems might arise in multi-homed machines because outgoing multicast messages are sent on only one interface in the absence of more sophisticated routing software. It is possible that the interface chosen by the operating system is incorrect. The Orchestrate Server and its associated tools make a best effort to ensure that discovery queries and announcements are sent on all available interfaces. It should not be necessary to run an external routing program with the current Orchestrate Server.

2.4 Multiple Subnets

By default, Orchestrate Server and its associated tools are configured to allow multicast messages to pass through up to two gateways. This allows discovery to work in multi-subnet environments, provided that the network routers on your network are properly configured to perform multicast routing. Consult the vendor’s documentation for information on configuring multicast routing on your network routers.

2.5 Datagrid and Multicasting

The PlateSpin Orchestrate datagrid facility provides a multicast-based file distribution service that allows large multi-gigabyte files to be simultaneously delivered to a large number of recipient machines while using far less network bandwidth than would be used by copying the file individually to each node. This service is available only in network environments that support IP multicasting. Aside from the file multicast service, all other features of the datagrid use normal unicast network operations and do not require multicast support. The routing and troubleshooting pointers provided above for network discovery also apply to datagrid multicasting. In addition, due to the potentially large bandwidth used by file transfers, you might want to limit the set of interfaces on which files are multicasted.

2.6 Datagrid Multicast Interface Selection

On multi-homed servers, the datagrid multicast service sends outbound control and data packets on all available interfaces on the system. This allows datagrid multicasting to work “out of the box” with multi-homed servers. This behavior might not be optimal if you require multicasting of files only to a subset of the available interfaces. You can instruct the datagrid to multicast only to the desired interfaces by selecting the correct interfaces from the Orchestrate Development Client on the Info/Configuration tab under *Data Grid Configuration*. Restricting the set of datagrid multicast interfaces prevents large amounts of file data from being sent to uninterested subnets.

PlateSpin Orchestrate and LDAP Authentication

3

Although PlateSpin Orchestrate has its own user database and authentication mechanism, it also allows integration with an existing Lightweight Directory Access Protocol (LDAP) system for authenticating user credentials.

This section includes the following information:

- Section 3.1, "What is LDAP?," on page 27
- Section 3.2, "Understanding LDAP Structure," on page 27
- Section 3.3, "How PlateSpin Orchestrate Uses an LDAP Entry to Authenticate," on page 29

3.1 What is LDAP?

At a high level, LDAP is a protocol designed to allow quick, efficient searches of directory services. Built around Internet technologies, LDAP makes it possible to easily update and query directory services over standard TCP/IP connections, and it includes many powerful features, including security, access control, data replication and support for Unicode.

LDAP is based on the Directory Access Protocol (DAP), which was designed for communication between directory servers and clients compliant to the X.500 standard. However, DAP can be difficult to implement and use, and is not suitable for use with Web applications. LDAP is a simpler, faster alternative, offering much of the same basic functionality without the performance overhead and deployment difficulties of DAP.

Because LDAP is built for a networked world, it is based on a client-server model.The system consists of one (or more) LDAP servers, which host the public directory service, and multiple clients, which connect to the server to perform queries and retrieve results. LDAP clients are built into most common address book applications, including e-mail clients like Microsoft Outlook and Qualcomm Eudora; however, since LDAP-compliant directories can store a diverse range of data (not just names and phone numbers), LDAP clients are also increasingly making an appearance in other applications. LDAP support is included in PlateSpin Orchestrate to allow it to integrate with existing user authentication mechanisms that are being used in many data centers.

There are many similarities between the Internet Domain Name System (DNS) model and LDAP: both are global directories that can be split across multiple hosts, both have built-in redundancy and replication features, and both include referral capabilities that make it possible to retrieve data that is not available locally from other hosts in the system.

3.2 Understanding LDAP Structure

An LDAP directory is usually structured hierarchically as a tree of nodes (the LDAP directory tree is sometimes referred to as the Directory Information Tree, or DIT). Each node represents a record, or "entry" in the LDAP database.

This section includes the following information:

- Section 3.2.1, “The Distinguished Name,” on page 28
- Section 3.2.2, “The Relative Distinguished Name,” on page 28

3.2.1 The Distinguished Name

An LDAP entry consists of numerous attribute-value pairs. It is uniquely identified by what is known as a “distinguished name” (DN).

To draw a parallel with a relational database management system (RDBMS), an LDAP entry is analogous to a record, its attributes are the fields of that record, and a DN is a primary key that uniquely identifies each record.

Consider the following example of an LDAP entry:

```
dn: mail=joe@novell.com, dc=novell, dc=com
objectclass: inetOrgPerson
cn: Joe
sn: Somebody
mail: joe@novell.com
telephoneNumber: 1 234 567 8912
```

This is an entry for a single person, Joe Somebody, who works at Novell. The components of the entry – name, email address, telephone number – are split into attribute-value pairs, with the entire record identified by a unique DN (the first line of the entry). Some of these attributes are required and some are optional, depending on the object class being used for the entry; however, the entire set of data constitutes a single entry, or node, on the LDAP directory tree.

3.2.2 The Relative Distinguished Name

Every entry in the directory tree has a “relative distinguished name” (RDN) consisting of one or more attribute-value pairs. An RDN must be unique at that level in the directory hierarchy. In the example above, for instance, the following are all valid RDNs for the entry:

```
cn=Joe
```

or

```
cn=Joe+sn=Somebody
```

or

```
cn=Joe+sn=Somebody+telephoneNumber=12345678912
```

or

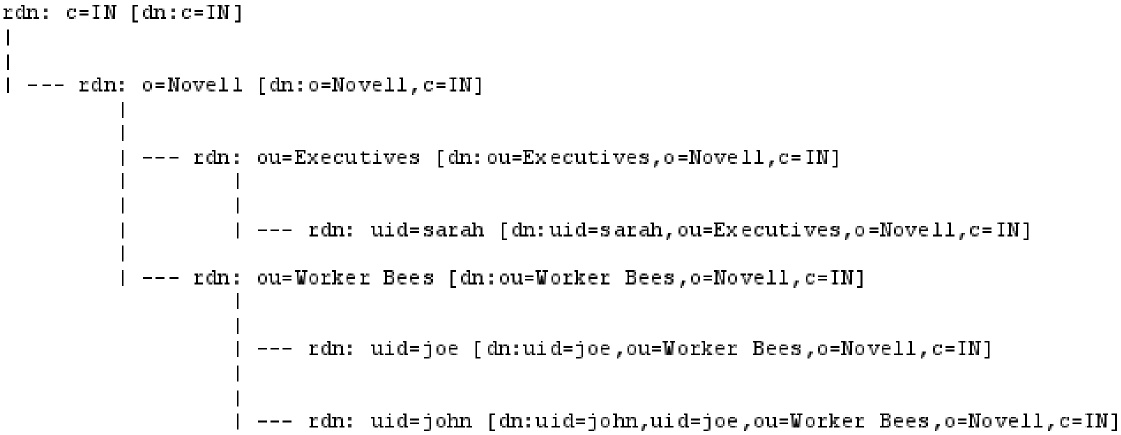
```
mail=joe@novell.com
```

There are no set rules regarding which attributes of a particular entry should be used for the RDN; the LDAP model leaves this decision to the directory designer, specifying only that the RDN of an entry must be such that it can uniquely identify that entry at that level in the DIT.

Because RDNs exist for every entry in the tree, the DN for any entry is formed by sequentially appending the RDNs of all the nodes between that entry and the root entry. In this way, you can use the DN to easily locate any node in the directory tree, regardless of its location or depth in the hierarchy.

For example, consider the following LDAP directory:

Figure 3-1 *Sample LDAP Directory*



```
rdn: c=IN [dn:c=IN]
|
|
| --- rdn: o=Novell [dn:o=Novell,c=IN]
         |
         |
         | --- rdn: ou=Executives [dn:ou=Executives,o=Novell,c=IN]
         |         |
         |         |
         |         | --- rdn: uid=sarah [dn:uid=sarah,ou=Executives,o=Novell,c=IN]
         |
         | --- rdn: ou=Worker Bees [dn:ou=Worker Bees,o=Novell,c=IN]
                   |
                   |
                   | --- rdn: uid=joe [dn:uid=joe,ou=Worker Bees,o=Novell,c=IN]
                   |
                   |
                   | --- rdn: uid=john [dn:uid=john,uid=joe,ou=Worker Bees,o=Novell,c=IN]
```

To identify the node belonging to Joe Somebody (the DN for Joe Somebody's entry) you would add all the RDNs between that entry and the root of the tree:

```
uid=joe,ou=Worker Bees,o=Novell,c=IN
```

In a similar manner, the DN for the node belonging to Sarah would be

```
uid=sarah,ou=Executives,o=Novell,c=IN
```

while the DN for the Novell node would be

```
o=Novell,c=IN
```

Because LDAP entries are arranged in a hierarchical tree, and because each node on the tree can be uniquely identified by a DN, the LDAP model lends itself to sophisticated queries and powerful search filters.

3.3 How PlateSpin Orchestrate Uses an LDAP Entry to Authenticate

PlateSpin Orchestrate uses only one attribute of a given LDAP user: its group membership. For example, if the following settings were already configured in PlateSpin Orchestrate,

```
BaseDN 'dc=domain,dc=novell,dc=com'
UserAttribute 'uid'
UserPrefix 'ou=Users'
```

you could further configure PlateSpin Orchestrate to identify users belonging to an LDAP group using the setting `LDAP:groupnocase:administrators`.

You would do this by specifying a filter in PlateSpin Orchestrate using these settings:

```
GroupFilter 'memberUid=${USER_NAME}'
GroupPrefix 'ou=Groups'
GroupAttribute 'cn'
```

Applying these settings would let authenticated users belonging to the “administrators” LDAP group be added to the “administrators” user group in PlateSpin Orchestrate (and so allow them to log in to the Development Client, for example).

For information on configuring these settings in PlateSpin Orchestrate, see “The Orchestrate Server Authentication Page” in the *PlateSpin Orchestrate 2.5 Development Client Reference*.

NOTE: Depending upon your selection at the *Server Type* drop down list on the *Enable LDAP* subpanel of the *Authentication* page of the Orchestrate Development Client, the configuration fields change to reflect the relevant settings. (One server type is *Active Directory Service*, the other is *Generic LDAP Directory Service*.)

The general concepts for LDAP authentication discussed above also apply to Active Directory authentication.

4 Increasing the Kernel ARP Threshold Value on the Orchestrate Server

Testing has shown that PlateSpin Orchestrate grids that have more than 1210 VMs (each with an installed Orchestrate Agent) and 124 Resource objects, the Orchestrate Server (installed on SLES 10 SP3) in the grid loses connection to the Audit Database Server or other devices installed on other machines. The failure is manifest following the ping command when the following error is displayed:

```
connect : No buffer space is available
```

To correct this problem, it is necessary to increase the kernel ARP threshold on the Orchestrate Server (SLES 10 SP3). This section contains the following information to help you perform this increase:

- Section 4.1, “Threshold Definitions,” on page 31
- Section 4.2, “Determining the Current Kernel Threshold Value,” on page 31
- Section 4.3, “Changing the Current Kernel Threshold Value,” on page 32

4.1 Threshold Definitions

The following definitions for kernel ARP levels is referenced from the Linux ARP man page (http://linux.die.net/man/7/arp):

- **gc_thresh1:** The minimum number of entries to keep in the ARP cache. The garbage collector will not run if there are fewer than this number of entries in the cache. Defaults to 128.
- **gc_thresh2:** The soft maximum number of entries to keep in the ARP cache. The garbage collector will allow the number of entries to exceed this for 5 seconds before collection will be performed. Defaults to 512.
- **gc_thresh3:** The hard maximum number of entries to keep in the ARP cache. The garbage collector will always run if there are more than this number of entries in the cache. Defaults to 1024.

4.2 Determining the Current Kernel Threshold Value

To determine the current kernel threshold value on the Orchestrate Server, use the following commands to determine the values for each threshold:

- `# cat /proc/sys/net/ipv4/neigh/default/gc_thresh1`

  The command might result in a displayed value like this:

  ```
  # cat /proc/sys/net/ipv4/neigh/default/gc_thresh1
  128
  ```

- `# cat /proc/sys/net/ipv4/neigh/default/gc_thresh2 512`

  The command might result in a displayed value like this:

  ```
  # cat /proc/sys/net/ipv4/neigh/default/gc_thresh2
  512
  ```

- `# cat /proc/sys/net/ipv4/neigh/default/gc_thresh3`

  The command might result in a displayed value like this:

  ```
  # cat /proc/sys/net/ipv4/neigh/default/gc_thresh3
  1024
  ```

You can also use this command:

```
#sysctl -A|grep ipv4|grep default |grep gc_thresh
```

The result is a listing similar to the following:

```
net.ipv4.neigh.default.gc_thresh3 = 1024
net.ipv4.neigh.default.gc_thresh2 = 512
net.ipv4.neigh.default.gc_thresh1 = 128
```

4.3 Changing the Current Kernel Threshold Value

When you know the current threshold values, you can change them using one of two methods:

- Section 4.3.1, "Editing the /etc/sysctl.conf File," on page 32
- Section 4.3.2, "Making Live Changes to the Threshold Values," on page 32

4.3.1 Editing the /etc/sysctl.conf File

1. Open `/etc/sysctl.conf` in a text editor.
2. Add the following lines to the `.conf` file:

   ```
   net.ipv4.neigh.default.gc_thresh1 = 256

   net.ipv4.neigh.default.gc_thresh2 = 1024

   net.ipv4.neigh.default.gc_thresh3 = 2048
   ```
3. Reboot the server.

4.3.2 Making Live Changes to the Threshold Values

To make changes to a given threshold on the Orchestrate Server you can run a command for each threshold that you want to change, for example:

```
# echo '256' > /proc/sys/net/ipv4/neigh/default/gc_thresh1
```

After you run the command, perform a `/etc/init.d/network restart` command to restart the Orchestrate Server and put the changes in place.

NOTE: This method of changing the threshold values is volatile: if you reboot the SLES server, the changes are lost.

PlateSpin Orchestrate Security

A

This section explains various security issues related to PlateSpin Orchestrate:

A.1 User and Administrator Password Hashing Methods

All passwords stored in PlateSpin Orchestrate are hashed using Secure Hash Algorithm-1 (SHA-1). However, user passwords are no longer hashed when sent from the client to the server. Instead, the plain text password entered by the user is sent over an encrypted authentication connection to the server to obtain a unique per-session credential issued by the server. This allows the server to “plug in” to alternative user directories such as Active Directory or OpenLDAP. Agent credentials are still stored, singly hashed, on the disk on the agent machine. The first pass hashing prevents “user friendly” passwords entered by administrators from being compromised by storing them on the agent machines. The server’s password database (for agents and for users not using an alternative user directory) stores all passwords in a double-hashed form to prevent a stolen password database from being used to obtain passwords.

WARNING: The zosadmin command line and the PlateSpin Orchestrate Development Client do not use SSL encryption, nor do they support TLS/SSL, so they should only be used over a secure network.

All agent and client connections support TLS encryption. This includes the zos command line and the PlateSpin Orchestrate Agent.

A.2 User and Agent Password Authentication

The PlateSpin Orchestrate Server stores all user and agent passwords in its data store as double-hashed strings. User clients such as the zos command send the plain text password over a TLS encrypted authentication connection to obtain a randomly generated per-session credential issued by the server. This session credential is retained by the client, either in memory or in a temporary disk file for the duration of the session.

It is not possible to obtain the user’s password from the session credential, however. It should be protected to prevent unauthorized users from taking over the session. Agents send a singly hashed password as their login credential, which is in turn hashed once more on the server to authenticate new agent connections. Upon authentication, agents receive the same type of session credential as user clients.

Singly-hashed password strings are used as a special case for agents, because agents typically must store their plain text credentials to disk to allow the agents to start up on host or VM reboot. The use of a once hashed version of the password on the agent prevents administrators from compromising “user friendly” text passwords by storing them unhashed on agents. The use of single hashing on the agents and double hashing on the server database prevents stolen credential data from being used to obtain actual user or administrator-entered passwords

A.3 Password Protection

You should take measures to protect the passwords and credentials on both the PlateSpin Orchestrate Server and PlateSpin Orchestrate Agents by ensuring that only the user account of the PlateSpin Orchestrate Server (currently `root` or `Administrator`, by default) has access to the `/store` and `/tls` directories on the server, so that general users are prevented from obtaining the password. On agents, allow only the agent users (normally `root` or `Administrator`) to have access to the `agent.properties` file, which contains the agent’s authentication credential.

Currently, PlateSpin Orchestrate restricts file access on the server, but we recommend that you disallow shell accounts on server machines for general users as a precaution.

For users, none of the Novell-provided client utilities stores the user-entered password to disk in either plain text or hashed form. However, temporary once-per-session credentials are stored to the disk in the users `$HOME/.novell/zos/client` directory. Theft of this session credential could allow someone else to take over that user session, but not to steal the user’s password. Users can protect their logged-in session by making sure the permissions either on their home directory or on the `~/.novell/zos/client` directory are set to forbid both read and write access by other users.

PlateSpin Orchestrate Agents use the same authentication protocol and password hashing as users (agent passwords are stored to disk in hashed form, not plain text) with the exception that agent passwords are not salted, allowing agents to be renamed by the server. Because agent passwords are not salted, we recommend that you generate and use random non-mnemonic strings for agent passwords.

For PlateSpin Orchestrate 2.0 and later, administrators can enhance security when configuring new agents by setting the `zos.agent.password` property to the asterisk character (*). This causes the agent to automatically generate a new random credential not based on any easily guessable plain text word. When the new agent is “accepted” by the administrator, the newly generated credential is stored by the server. This is the default behavior when the PlateSpin Orchestrate Agent is first installed.

n addition, the `zos.agent.password` property can be set to a plain text password in `agent.properties`. If this is done, the agent automatically replaces the plain text password with the hashed version when it next starts. This allows administrators to more easily set up an initial password for agents.

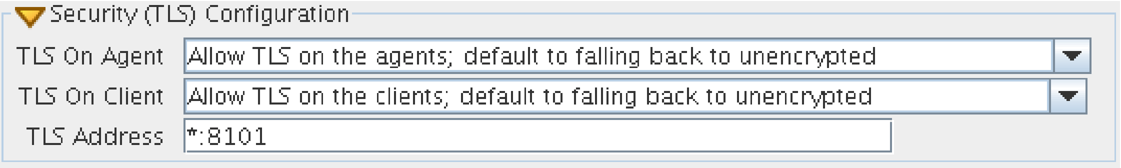
A.4 TLS Encryption

- Section A.4.1, “Setting TLS Options,” on page 35
- Section A.4.2, “Updating tje TLS Server Certificate,” on page 36

A.4.1 Setting TLS Options

PlateSpin Orchestrate uses Transport Layer Security (TLS) to provide encryption for both user and agent connections. Both the PlateSpin Orchestrate Agent and the PlateSpin Orchestrate Clients use TLS to initiate their connections to the Orchestrate Server, and then the server specifies whether to "fall back" to plain text or continue the session fully encrypted.Although you can manually configure the agent and clients to either always require TLS encryption or to fully disable TLS encryption, we recommended that you leave the agents and clients in their default configuration, and then use the PlateSpin Orchestrate Development Client on the server to specify the default behavior. This is the purpose of the TLS Options section on the main server tab of the Orchestrate Development Client.

Figure A-1 *TLS Options in the PlateSpin Orchestrate Development Client*



Security (TLS) Configuration

TLS On Agent: Allow TLS on the agents; default to falling back to unencrypted

TLS On Client: Allow TLS on the clients; default to falling back to unencrypted

TLS Address: *:8101

Here, there are 4 levels that you can set separately for both agent connections and user/client connections:

- **Forbid TLS for (agents/clients):** This option is to fully disable and prohibit TLS encryption altogether. This is the least secure option and is therefore usually not the desirable choice, but it could be required in countries that restrict encryption or in low security environments where performance is more critical than security.
- **Allow TLS on the (agents/clients); default to falling back to unencrypted:** This option (the factory default for both agents and clients) is to allow TLS encryption if the agent or client explicitly requests it, but to default to falling back to plain text after authentication.

  NOTE: Authentication always occurs over SSL, regardless of settings.

- **Allow TLS on the (agents/clients); default to TLS encrypted if not configured encrypted:**

  This option is similar to the second option. Agents/clients may specify whether or not to use TLS, but if they use the default of "server specified," the server defaults to using TLS.
- **Make TLS mandatory on the (agents/clients):** This option is the most secure, locked down option. It requires TLS at all times, and fails connections if the agent or the client tries to specify plain text.

In addition to these settings for TLS configuration, there are files that need to be protected on both the server and on the client/agent. For more information, search for the *TLS Certificate Installation On PlateSpin Orchestrate* article at the *Novell Cool Solutions Community (http://www.novell.com/communities/coolsolutions/)*.

A.4.2 Updating tje TLS Server Certificate

Understanding Transport Layer Security (TLS) encryption is particularly important if you reinstall the server and have an old server certificate in either your agent or client user profile similar to `ssh` shared keys. If you have an old certificate, you need to either manually replace it or delete it and allow the client or agent to download the new one from the server using one of the following procedures:

- **For the Agent:** The TLS certificate is in `<agentdir>/tls/server.pem`. Deleting this certificate will cause the agent, by default, to log a minor warning message and download a new one the next time it tries to connect to the server. This is technically not secure, since the server could be an impersonator. If security is required for this small window of time, then the real server's `<serverdir>/<instancedir>/tls/cert.pem` can be copied to the above server.pem file.
- **For the Client:** The easiest way to update the certificate from the command line tools is to simply answer "yes" both times when prompted about the out-of date certificate. This is, again, not 100% secure, but is suitable for most situations. For absolute security, hand copy the server's cert.pem (see above) to `~/.novell/zos/client/tls/<serverIPAddr:Port>.pem`.
- **For Java SDK clients:** Follow the manual copy technique above to replace the certificate. If the local network is fairly trustworthy, you can also delete the above `~/.novell/.../*.pem` files, which will cause the client to auto-download a new certificate.

A.5 Security for Administrative Services

The PlateSpin Orchestrate Development Client and the zosadmin command line tool are clients to the MBean and RMI servers. PlateSpin Orchestrate does not provide encryption for these administrative services, so you should be careful to use them only in a secure environment.

When the user logs in using either `zosadmin login` or the Orchestrate Development Client, the user's password is sent to the server, and then the server issues a per-session credential to be used for further operations. The user's cleartext password is never stored to disk; however, it is currently sent "over the wire" in plain text form. For this reason, the administrative clients should only be used in a secure, trusted environment.

The zosadmin client stores the session credential obtained from a `zosadmin login` request in a temporary file for use by subsequent operations. This credential cannot be used to obtain the user's password, but it could be used to take over the user's current session until it times out or expires. For this reason, the files in the user's `.novell/zoc/` directory should be configured to disallow access by other users.

A.6 Plain Text Visibility of Sensitive Information

The following table outlines where sensitive information might be visible as plain text:

***Table A-1** Locations Where Sensitive Information Might Be Stored As Plain Text*

Information	Storage Location	Visibility Issue
Audit Database configuration	ZOS properties store	Contains plain text information including user/ password for allowing PlateSpin Orchestrate to log into the Audit Database for logging. You should use a non-privileged database account for logging.

Changing Orchestrate Server Default Parameters and Values

B

The following table provides the current default values for some key performance parameters of the PlateSpin Orchestrate Server. Although the server is fine-tuned by default for optimal performance at normal loads, if you want to perform hundreds of provisioning actions simultaneously you can change some of the default settings for increased server performance in such a scenario.

Table B-1 *Default Parameters of the PlateSpin Orchestrate Server*

Parameter Name	Shipping Default Value	Changing or Displaying the Parameter Configuration
File Descriptors Limit	2048	(Optional) You can change the value as shown below: `/etc/init.d/novell-zos-server: ulimit -n <new_limit>`

Java Heap Space	2048 MB	Change the Java heap space from the default by changing the start parameters in `/etc/init.d/novell-zosserver` by editing the line
		`$ZOS_BIN start -d $ZOS_CONFIG > /dev/null`
		Change the line as follows:
		`$ZOS_BIN start --jvmargs=-Xmx4096m -d $ZOS_CONFIG > /dev/null`

PermGen Space	512 MB	Increase the PermGen space size from the default by changing the start parameters in `/etc/init.d/novell-zosserver` by editing the line
		`$ZOS_BIN start -d $ZOS_CONFIG > /dev/null`
		Change the line as follows:
		`$ZOS_BIN start --jvmargs=-Xmx4096m --jvmargs=-XX:MaxPermSize=<new_value_in MB> -d $ZOS_CONFIG > /dev/null`

Audit Queue Size Max	200	Increase the value of this parameter by using the following command: `zosadmin set --mbean="local:facility=audit" --attr=QueueSizeMax --type=Integer --value=1000`

Parameter Name	Shipping Default Value	Changing or Displaying the Parameter Configuration
MaxRunJobWaitTimeout	120000	You can change the value of this parameter as shown below: `zosadmin set --mbean="local:facility=broker" --attr=MaxRunJobWaitTimeout --type=Integer --value=<`*`time_in_milliseconds`*`>`
MatchingResourcesCheckinterval	30000	Increase the value of this parameter by using the following command: `zosadmin set --mbean="local:facility=broker" --attr=MatchingResourcesCheckInterval --type=Integer --value=600000`

Kernel ARP Threshold Values	◆ thresh1 = 128
	◆ thresh2 = 512
	◆ thresh3 = 1024
		Set these values higher than the default. For example:
		`cat /proc/sys/net/ipv4/neigh/default/gc_thresh1 = 256`
		`cat /proc/sys/net/ipv4/neigh/default/gc_thresh2 = 1024`
		`cat /proc/sys/net/ipv4/neigh/default/gc_thresh3 = 2048`

Job Limits: ◆ Soft Top Level Job Limit ◆ Max Queued Jobs ◆ Absolut Max Active Jobs	`matrix.maxtopjobs` = 200 `matrix.maxqueued` = 300 `matrix.maxactive` = 400	Change the Grid Object default values in the Development Client as follows: `matrix.maxtopjobs` = 600 `matrix.maxqueued` = 700 `matrix.maxactive` = 1000

Documentation Updates

C

This section contains information about documentation content changes that were made in this PlateSpin Orchestrate Development Client Reference after the initial release of PlateSpin Orchestrate 2.5. The changes are listed according to the date they were published.

The documentation for this product is provided on the Web in two formats: HTML and PDF. The HTML and PDF documentation are both kept up-to-date with the changes listed in this section.

If you need to know whether a copy of the PDF documentation that you are using is the most recent, the PDF document includes a publication date on the title page.

The documentation was updated on the following dates:

- Section C.1, "January 27, 2011," on page 41

C.1 January 27, 2011

Updates were made to the following sections:

Location	Update Description
Section 1.2.1, "Resource Virtualization," on page 18	Removed reference to the deprecated VM Builder utility and replaced it with information about the vm-install binary.