HP Service Activator

System Integrator's Overview

Edition: V51-1A

**for Microsoft Windows® Server 2008 R2, HP-UX 11i v3,
Red Hat Enterprise Linux 5.4 and Solaris 10 operating systems**

Manufacturing Part Number: None

July 2, 2010

Legal Notices

Warranty.

Hewlett-Packard makes no warranty of any kind with regard to this manual, including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Hewlett-Packard shall not be held liable for errors contained herein or direct, indirect, special, incidental or consequential damages in connection with the furnishing, performance, or use of this material.

A copy of the specific warranty terms applicable to your Hewlett-Packard product can be obtained from your local Sales and Service Office.

Restricted Rights Legend.

Use, duplication or disclosure by the U.S. Government is subject to restrictions as set forth in subparagraph (c)(1)(ii) of the Rights in Technical Data and Computer Software clause in DFARS 252.227-7013.

Hewlett-Packard Company
United States of America

Rights for non-DOD U.S. Government Departments and Agencies are as set forth in FAR 52.227-19(c)(1,2).

Copyright Notices.

Trademark Notices.

Java™ is a U.S. trademark of Sun Microsystems, Inc.

Linux is a U.S. registered trademark of Linus Torvalds

Microsoft® is a U.S. registered trademark of Microsoft Corporation.

Oracle® is a registered U.S. trademark of Oracle Corporation, Redwood City, California.

UNIX® is a registered trademark of the Open Group.

Windows® and MS Windows® are U.S. registered trademarks of Microsoft Corporation.

All other product names are the property of their respective trademark or service mark holders and are hereby acknowledged.

HP Service Activator contains Westhawk's Java SNMP stack.

Document id: p158-pd001206

Contents

Contents

Contents

In This Guide

This guide provides an overview of the HP Service Activator product, including its architecture and components, and provides information to help plan and design Service Activated-based solutions.

Audience

The audience for this guide is the Systems Integrator (SI) who will plan and deliver solutions, particularly SIs with architect roles. It is not intended for the end user. Implementors will need additional detail from the manuals for the different components of tools within the Service Activator product.The SI is expected to have some or all of the following background:

- Understanding and working knowledge of:
  - UNIX® commands
  - Windows® system administration
- Familiarity with Java™ and XML
- Understanding of security issues
- Understanding of the customer's problem domain

Conventions

The following typographical conventions are used in this guide.

Font	What the Font Represents	Example
Italic	Book or manual titles, and manpage names	Refer to *HP Service Activator, Workflows and the Workflow Manager* and the *Javadocs* for more information
	Provides emphasis, introduces a new term	You *must* follow these steps.
	Identifies a variable or parameter	Run the command: `InventoryBuilder` <*sourceFiles*> The *assigned_criteria* parameter returns an ACSE response.
	Location descriptor	*`$JBOSS_DEPLOY`*
`Computer`	Text and items on the computer screen	The system replies: `Press Enter`
	Command names	Use the `InventoryBuilder` command
	Method names	The `get_all_replies()` method does the following…
	File and directory names	Edit the file *`$ACTIVATOR_ETC`*`/config/mwfm.xml`
	Window/dialog box names	In the `Test and Track` dialog…
`Computer Bold`	Text that you must type	At the prompt, type: **`ls -l`**
Keycap	Keyboard keys	Press **Return**
`[Button]`	Buttons on the user interface	Click `[Delete]`. Click the `[Apply]` button.
`Menu Items`	A menu name followed by a colon (`:`) means that you select the menu, then the item. When followed by an arrow (`->`), a cascading menu follows.	Select `Locate:Objects->by Comment`

Install Location Descriptors

The following names are used to define install locations throughout this guide.

Descriptor	What the Descriptor Represents
$ACTIVATOR_OPT	The base install location of Service Activator. The UNIX® location is `/opt/OV/ServiceActivator` The Windows® location is `<install drive>:\HP\OpenView\ServiceActivator`
$ACTIVATOR_ETC	The install location of specific Service Activator files. The UNIX location is `/etc/opt/OV/ServiceActivator` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\etc`
$ACTIVATOR_VAR	The install location of specific Service Activator files. The UNIX location is `/var/opt/OV/ServiceActivator` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\var`
$ACTIVATOR_BIN	The install location of specific Service Activator files. The UNIX location is `/opt/OV/ServiceActivator/bin` The Windows location is `<install drive>:\HP\OpenView\ServiceActivator\bin`
$ACTIVATOR_THIRD_PARTY	The location for new Java™ components such as workflow nodes and modules. The UNIX location is `/opt/OV/ServiceActivator/3rd-party` The Windows location is `<install drive>:\HP\OpenView\Service Activator\3rd-party` Customized inventory files are stored in the following location: *$ACTIVATOR_THIRD_PARTY*`/inventory`
$JBOSS_HOME	The install location for JBoss. The UNIX location is `/opt/HP/jboss` The Windows location is `<install drive>:\HP\jboss`
$JBOSS_DEPLOY	The install location of the Service Activator J2EE components. The UNIX location is `/opt/HP/jboss/server/default/deploy` The Windows location is `<install drive>:\HP\jboss\server\default\deploy`
$JBOSS_ACTIVATOR	More specific location of Service Activator UI components deployed in JBoss: *$JBOSS_DEPLOY*`/hpovact.sar/activator.war`

1 Introducing HP Service Activator

HP Service Activator is a customizable product that performs tasks to activate services offered by providers of converged IT and network communications services. It can perform any activation on elements of any infrastructure comprising network elements and IT servers that can be configured through command or request interfaces, whether they use command lines, web services, or any other protocol. The product is typically deployed to perform highly repetitive activations where automation brings a significant advantage in terms of cost saving, speedup and ensuring correct activation, but it is possible to use Service Activator to automate any process that requires the execution of a sequence of automated command interactions.

The core of HP Service Activator is a generic workflow engine. To build a solution for activation of services in a specific domain the core must be supplemented with customized data models, workflows and plug-ins for interaction with the elements in the provider's environment. Workflows must implement the activation processes needed to add, modify or terminate services for each of the provider's customers. The product includes tools to assist the SI in the customization process.

This overview manual introduces the way Service Activator works in a deployed solution as well as the tool set and the customization process that the SI must go through in a solution delivery project.

Positioning an HP Service Activator Solution

Described here is the positioning of HP Service Activator for use as an activation system in fulfillment solutions. It should be noted that this positioning is not exclusive. Due to the flexibility of Service Activator workflows and plug-ins, it is possible to customize solutions also for other domains, notably test and diagnostics, also known as network troubleshooting applications.

Positioning in the Provider's Environment

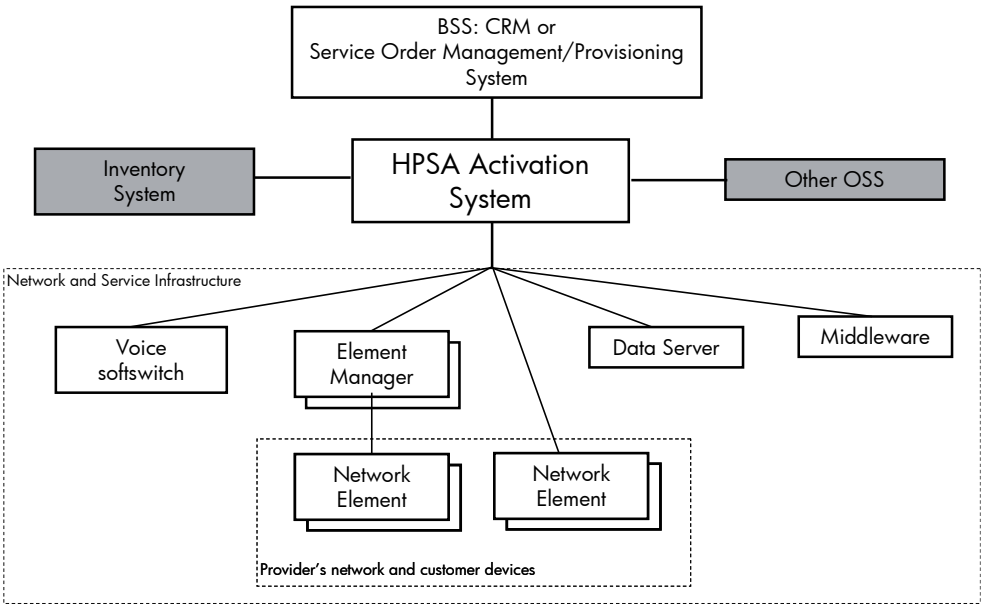
An activation system built with HP Service Activator will typically be part of a complete service fulfillment solution driven by requests from a CRM (Customer Relationship Management) system. The fulfillment solution may comprise also order management (OM) and resource inventory, and it may be integrated with other Operation and Business Support Systems (OSS and BSS) outside of fulfillment.

The Service Activator-based activation system typically has a built-in data repository commonly referred to as inventory: resource inventory, which is used to allocate resources in the network and keep data about devices which are activated, and service inventory which records services and their parameters as they are activated. Alternatively, if required, the activation system can access the inventory data in an external inventory system.

The positioning of the activation system between a BSS system, which drives it with requests for service activation, and the network and service infrastructure on which services are activated is shown in Figure 1-1. In this diagram the flow of control is from north to south. The network and service infrastructure may contain many different kinds of network elements and data servers. The latter can belong to a service infrastructure, for example GSM or IMS: HLR, HSS, servers for voice mail, SMS, MMS, etc. They can also be related to Internet data services: DHCP, RADIUS,

email, etc. In general any server which needs to be activated with information about a new customer/subscriber can be part of the infrastructure that is known to the Service Activator system.

Figure 1-1 Positioning of HP Service Activator Activation System

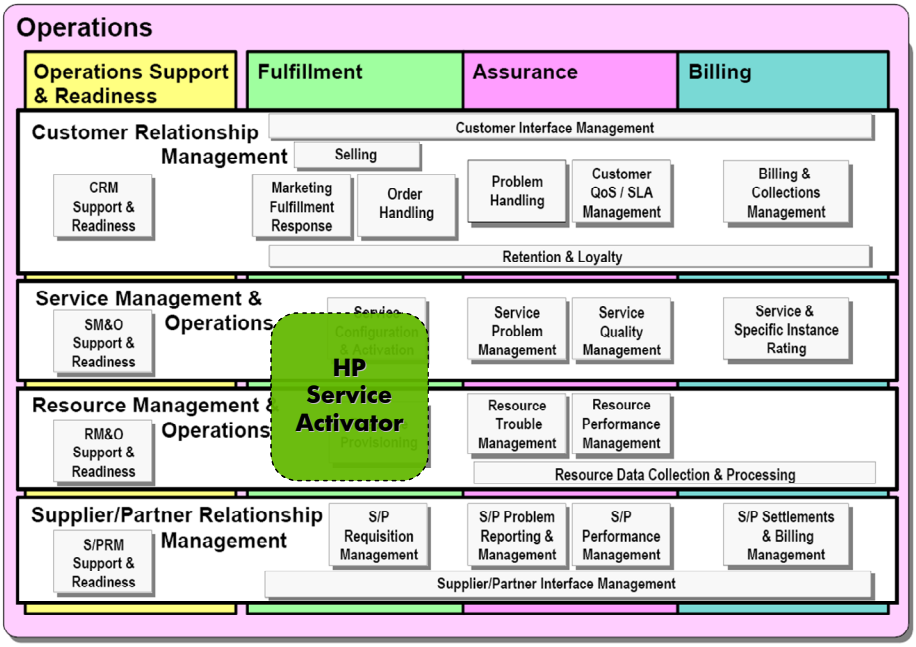


It is typical, but not mandatory that Service Activator is driven by requests forwarded by an external BSS system, as discussed above. It is also possible to implement a stand-alone system where requests to start workflows are entered directly from the user interface on HP Service Activator.

Positioning in TMF NGOSS Maps

For the reader who is familar with TMF's (NGOSS) eTOM map of the processes which are executed by a communications service provider to plan, deliver and maintain services, HP Service Activator can be characterized by its position on the map as shown in Figure 1-2. Service Activator helps to automate the processes designated in eTOM as 'Service Configuration & Activation' and in particular 'Resource Provisioning'.

Figure 1-2 HP Service Activator in the eTOM Map

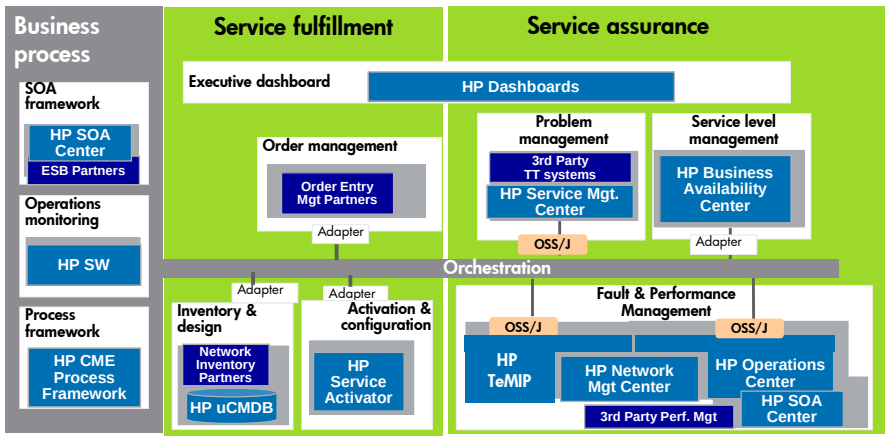


HP Service Activator can also be positioned on TMF's emerging Telecom Application Map (TAM). TAM describes a number of so-called applications (with considerable overlap), grouped in domains and sub-domains, from which a complete OSS can be constructed. The functions of Service Activator fall mainly in the sub-domains of Resource Inventory Management and Resource Order Management, but also in Service Order Management.

Positioning in HP Integrated NGOSS Solutions

HP offers integrated NGOSS solutions which use a number of HP's own products along with appropriate partner products. The products that are used are depicted on a map which is similar to both eTOM and TAM, as shown in Figure 1-3. Here HP Service Activator appears together with other OSS products, integration adapters and business process management tools which may be integrated to deliver a complete solution.

Figure 1-3 Service Activator in HP NGOSS Solutions



HP Service Activator Component Architecture

In Figure 1-1 there is a single box 'HPSA Activation System'. This box has a number of interfaces. Here we open the box and consider the components inside it. The various interfaces into the box are handled by different internal components. Figure 1-4 shows the major components: the workflow manager, the resource manager, and the web server, all of which run on the J2EE platform JBoss. The web server (Apache Tomcat) is actually part of JBoss, whereas the other components are HP Service Activator additions to the platform. For further discussion of the platform and how Service Activator can scale by using a cluster of servers (Note: not based on JBoss clustering), see chapter 6.

This section only scrapes the surface. For more information about the components and the associated development tools, read the next chapter and then go to the dedicated manuals as needed.

The workflow manager and the resource manager together make up the workflow engine of Service Activator. The workflow engine does the process work of the activation system by executing workflow jobs.

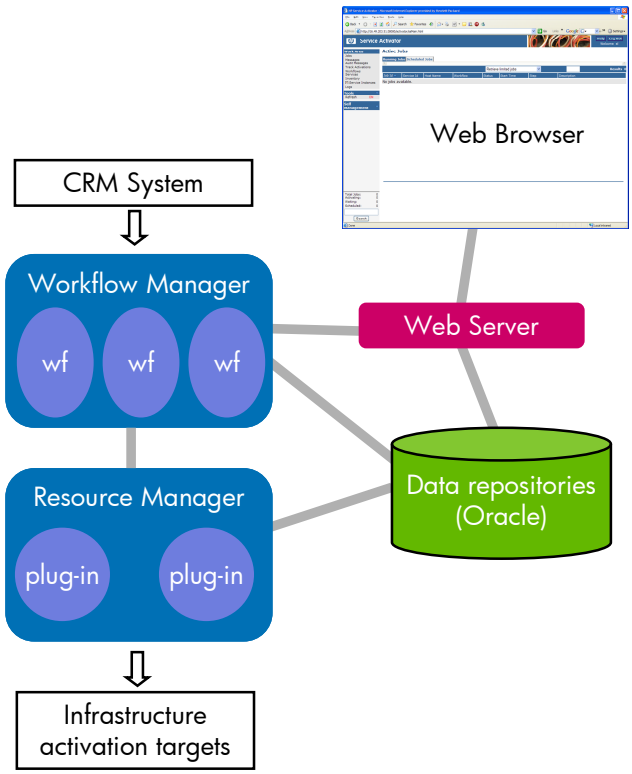
Workflows

A workflow is a definition of an executable process. The definition is at a detailed algorithmic level, suitable for control of interactions with activation targets. HP Service Activator workflows should not be confused with business process workflows as supported by languages like BPEL. It is possible, however, to implement processes which interact with human operators as well as external system and have significant duration in Service Activator. Workflows are composed from a set of action primitives known as workflow nodes. They are executed by the workflow manager, one node at a time. The crucial step in an activation workflow is the Activate node. This node executes an activation task which is where the actual interaction with activation targets take place.

Plug-ins and activation tasks

Activation tasks consist of one or more atomic tasks; each atomic task interacts with a specific activation target and is implemented as an element of the plug-in for that target, for example a specific type of network element (vendor and model) or IT server, for example LDAP, or a voice softswitch. A plug-in is a pluggable component which is managed by the resource manager. A task that consists of more than one atomic task is called a compound task. A compound task is executed as a transaction: the atomic tasks are executed sequentially, and if one of them fails, any atomic tasks that have already finished successfully will be undone by executing their "undo" parts, so that the net effect of the compound task will be nil. The undo part of each atomic task restores the state of the target as it was before the atomic task was executed.

Figure 1-4 HP Service Activator Components and Interfaces



HP Service Activator includes a few generic plug-ins, which are not for specific devices or target systems, but for a certain of type of interface, such as a command line interface (CLI) or HTTP message exchange interface. When a generic plug-in is used, it will have pre-implemented atomic tasks which handle the communications protocol , but must be customized with additional control information that determines the specific commands or messages that are exchanged. In these cases a plug-in dedicated to specific type of activation target comprises the generic plug-in plus the additional customization. Such plug-ins occur frequently.

Plug-ins may be reusable from one solution to another, and a list of plug-ins that have been built is maintained as a "plug-in library". However, plug-ins are typically designed to fit the requirements of a particular provider's solution. To limit the implementation effort they are not generally designed to be able to control all features of the target, and may therefore require additional work when reused for a different project.

Solution Data Repositories (Inventory)

It will often be necessary as steps of activation processes implemented with workflows to access data describing resources in the provider's infrastructure as well as the services (instances) that have been activated. Service Activator has the capability to model and maintain repositories of data that is needed by the solution, commonly referred to as resource and service inventory data. There is also a user interface for working with inventory data.

Northbound interface

The native northbound interface of the workflow manager is a Java RMI interface which supports a range of methods to inspect and control the state of the workflow manager. In particular it is possible to request the workflow manager to start a workflow job, specified by name and with parameters. The RMI is normally not used directly by an external requester, but all other NBIs are based on it.

Solution Packages

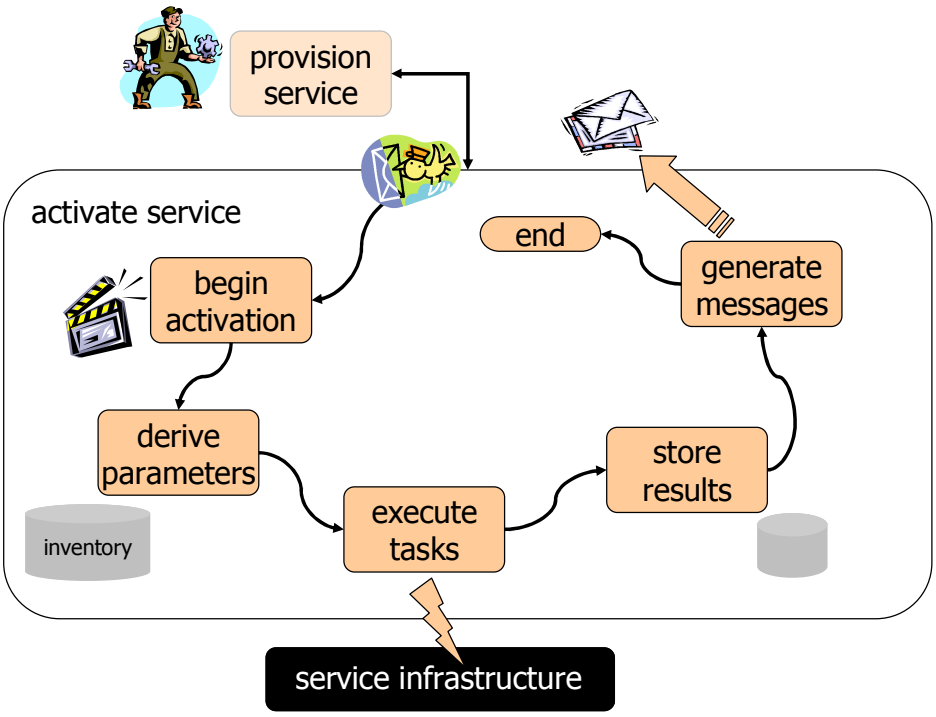
For frequently occurring types of activation solutions pre-customized packages of plug-ins, workflows and more, are maintained and made available for delivery projects. These packages allow rapid delivery of solutions. Depending on the provider's environment some customization will still be required, to adapt to the provider's specific network architecture, processes and service definitions, to develop any additional plug-ins needed for targets not already supported, and to integrate with other systems.

Solution packages exist for layer 2 (Metro Ethernet) and layer 3 (IP) VPN services and for residential services delivered over IP networks (Internet access, IPTV, etc.).

A Typical Workflow

Workflows for activation of services typically follow a pattern as depicted in Figure 1-5.

Figure 1-5 Typical Service Activation Workflow



In an actual workflow, each box as shown in Figure 1-5, will require several workflow nodes, the number depending on the complexity of the details of the process. At the high level, as shown, the process is typically as follows:

- A customer order is entered into a CRM system, where it is validated, approved and forwarded to the next level of processing. In a simple case it can go directly to Service Activation. In a more complex case, an order management process may be needed to decompose an order which is for a bundle of services and to separate the request for automatic activation, which can be handled by HP Service Activator, from other tasks which must be performed manually or passed on to partners of the provider.
- The activation system based on HP Service Activator receives the request for activation of a service, maps to the appropriate workflow and starts an instance of the workflow as a job.

- The workflow job inspects the parameter values from the request and calculates or gathers additional necessary parameter values.
  - For example, if a service type is specified as ‘Gold’ or ‘Silver’, the corresponding technical values for bandwidth and similar parameters will be looked up. For a complex technology, the derivatioin can also be complex.
  - Resources needed to satisfy the request, such as an access port, for example on a DSLAM, where customer premises equipment can be connected, are allocated from resource inventory. Resource identifiers will also be activation parameters. Allocated resources are recorded in resource inventory.
  - Parameter values, for example selection of a device or port, can also be obtained from a dialog with an operator.
- When all parameters are ready, the necessary activations are executed, ideally as a single task, which may be compound.
  - If an atomic task within a compound task fails, any preceeding atomic tasks are automatically rolled back. This logic is not visible in the workflow.
- If the activation was successful, the newly activated service will be recorded in service inventory. If it failed, any reserved resources are again released.
- A summary of the action of the workflow, including service id, main parameter values, success or failure, is recorded in HP Service Activator’s audit trail.
- A response message to be returned to the requester is prepared. Success or failure indication and any relevant parameters that were derived by the workflow are included in the message. The message is forwarded according to the protocol of the northbound interface. This may be done by a sender module or by the web service interface after the workflow terminates.
- The response message is processed by the order handling system or CRM system. This is outside the scope of the activation system.

In actual implementations process logic is often divided over several workflows. A parent workflow can start instances of child workflows and synchronize with their completion. An architecture is recommended where the northbound communication to receive and respond to incoming request messages is separated from activation details that will differ for each type of request. This topic will be discussed in chapter 5.

HP Service Activator Documentation

This manual provides an introduction and overview:

- Chapter 2 describes in more detail each of the components of HP Service Activator that were introduced above along with the tool(s) for customizing the corresponding part of a solution.
- Chapter 3 walks through a simple (not real) solution containing components of all the kinds introduced in the first chapters.
- Chapter 4 provides a brief guide for planning a solution delivery project and identifies the topics to be covered in the analysis phase which may be needed as a pre-sales activity to scope the delivery project.
- Chapter 5 discusses some design topics for the different components of a solution.
- Chapter 6 describes how HP Service Activator can be deployed on a cluster of JBoss platforms for scalability and high availability.
- Chapter 7 describes the use of roles to control access and customize the user interface for different groups of users.
- Chapter 8 describes the Common Network Resource Model.

- Chapter 9 describes the Web Server Designer, a tool to generate a web service servlet dedicated to a solution.
- Chapter 10 describes the integration of HP Service Activator and HP NNMi.
- Chapter 11 describes the integration of HP Service Activator and HP Network Automation (NA).
- Chapter 12 gives some useful hints about testing and debugging.
- Chapter 13 gives advice about configuring an HP Service Activator system.
- Chapter 14 gives an overview of the process of localizing an HP Service Activator solution.

For further familiarization with HP Service Activator it will be useful to study some of the example material that is provided as part of the installable kit. A simple example is introduced in chapter 2. A more thorough example - using management of layer 2 VPN services as use case - complete with inventory data model, workflows and plug-in code is fully documented in a separate manual, *Putting Service Activator to Work: A Sample Service Scenario for VPLS*.

For an actual customization and solution delivery project, detailed manuals are provided for installation on each supported operating system platform and for the major components of HP Service Activator and the associated customization tools:

- *HP Service Activator, Workflows and the Workflow Manager* provides all the information needed to understand workflows in detail, including descriptions of the workflow node and workflow manager module libraries, how to extend the libraries with new nodes and modules, and the workflow designer tool.
- *HP Service Activator, Inventory Subsystem* explains how to customize a solution data repository model with definitions of each entity class (database table) and the user interface to present the data model. This manual also describes the Inventory Builder, the tool used to process inventory definitions.
- *HP Service Activator, Developing Plug-Ins and Compound Tasks* explains the concepts of plug-ins, atomic tasks and compound tasks, and how to use the Service Builder tool to build plug-ins and customize compound tasks.
- *HP Service Activator, Solution Separation and the Deployment Manager* explains how customized solutions can be managed on a target system: installed, inspected, removed. The Deployment Manager is the tool to use for managing solutions.

There is one manual describing the HP Service Activator and its user interface in a generic way for users and system administrators: *HP Service Activator, User's and Administrator's Guide*. You should read that manual as an introduction and to understand how you can supply solution specific information.

2 Solution Components and Tools

In chapter 1 the components of HP Service Activator were introduced: workflow manager, resource manager, data repositories. Correspondingly, a customized solution will contain different components: workflows implementing activation processes, plug-ins implementing target interactions, and definitions for the required data model. On top of the workflows it will generally be necessary to add a northbound interface for receiving and responding to service activation requests. Depending on specific requirements it may also be necessary to integrate with other OSS/BSS systems.

Service Activator comes with a set of tools which make the process of customization a joy. They will be introduced in this chapter and each one is thoroughly documented in one of the manuals listed at the end of chapter 1. Here is an overview of this chapter:

- The first section, “Database Repositories”, describes the key role that is played by the Oracle database to hold a number of repositories that are used by HP Service Activator.
- The second section, “Solution Data Repositories ”, introduces the capabilities of HP Service Activator to manage repositories of resource and service data, also known as *inventory*, including the tree-structured inventory user interface. The tools in this area are the Inventory Builder and the Inventory Tree Designer.
- The third section, “Plug-Ins and Activation Tasks”, introduces the concepts of HP Service Activator plug-ins, including plug-ins that are developed as new Java source code by the solution integrator as well as generic plug-ins that can be customized. The tool in this area is the Service Builder.
- The fourth section, “Workflows”, explains the capabilities and architecture of the workflow manager and introduces the construction of workflows. The tool for building workflows is the Workflow Designer.
- The fifth section, “User Interface and Roles”, introduces the user interface of HP Service Activator.
- The sixth section, “Interfaces for Integration”, describes how an HP Service Activator solution is integrated with other OSS/BSS systems, primarily how to construct a northbound interface for receiving and responding to service activation requests. There is no tool dedicated to building NBIs.
- The seventh section, “Integration with Other HP NGOSS Products: NNMi, NA, uCMDB”, introduces the components which are present in HP Service Activator to enable integration with the mentioned products.
- The final section, “Solution Deployment”, describes how to organize, package and deploy all the components of a solution. The Deployment Manager is the tool that is used to deploy and manage one or more customized solutions on the HP Service Activator platform.

The next chapter, “An Example Solution: Intro_Example”, walks through a small example solution which does not activate devices in a real environment, but includes examples of all the solution components that have been discussed in sections of this chpater as material for study.

Database Repositories

Service Activator uses several data repositories, all stored in tables of one or more Oracle databases, for different purposes. The first five repositories listed here are held in predefined tables that are created when Service Activator is installed.

- Static Repository - customized items deployed on the system: workflows, plug-ins, compound tasks, inventory presentation tree definitions; the static repository for atomic and compound tasks is also called the task repository,
- Workflow Job Repository - state information about running workflow jobs and about cluster nodes
- Auxiliary Repository - temporary data for use under workflow control
- Audit and Message Repository - audit trail collected from running workflow jobs and modifications of inventory data, messages shown to operators
- Statistics Repository - server usage and workflow job statistics
- Solution Data Repository (Inventory) - the data model defined by the SI, primarily intended but not restricted to store data pertaining to infrastructure resources (network elements, servers, applications, etc.).

By default, all the repositories belong to a single Oracle user which is defined when HP Service Activator is installed. Additional inventory repositories can be held in other Oracle databases. To control the use of Oracle servers and of physical disk space it is possible to assign the different repositories to multiple databases and/or to organize the tables in different tablespaces.

Solution Data Repositories (Inventory)

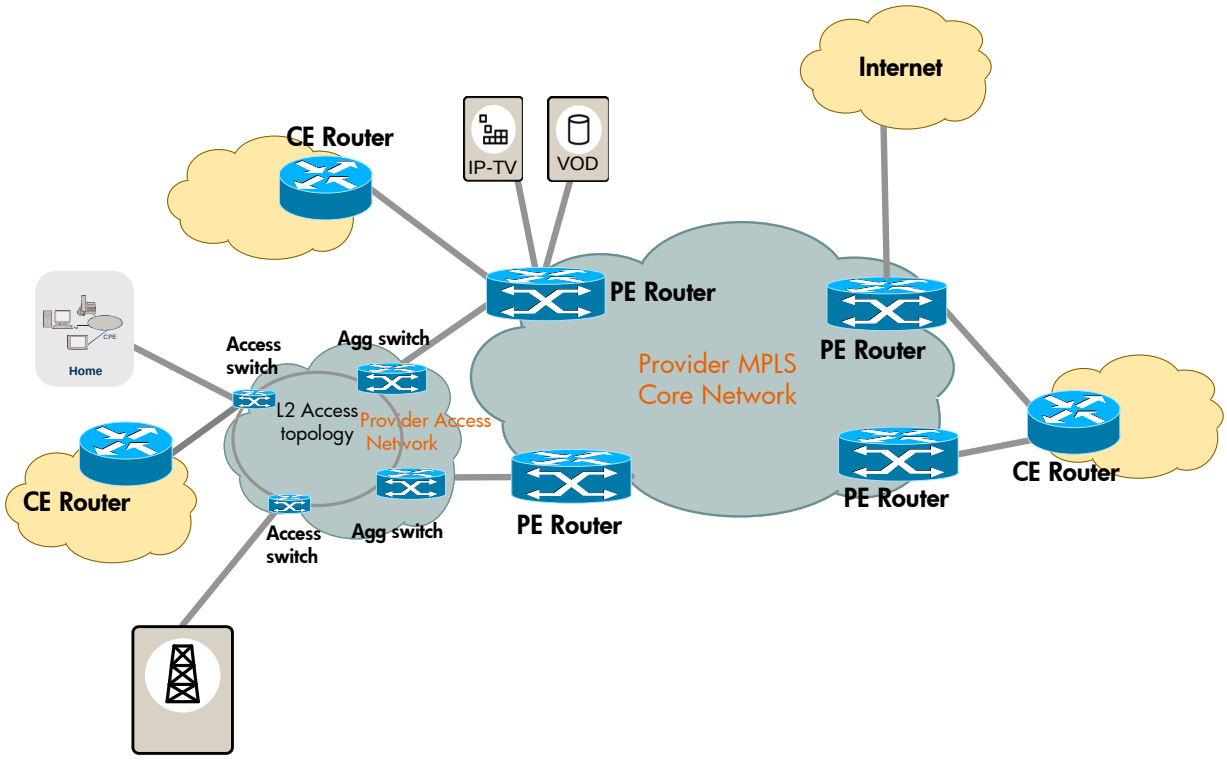
Repositories of data representing resources and services, or indeed anything that is needed, can be incorporated in an HP Service Activator solution. Such repositories can be accessed from workflows as well as from the user interface. Data repositories are commonly referred to as resource inventory and service inventory, but these concepts and the distinction between them are not implied by the tools that are used.

The HP Service Activator core product includes a Common Network Resource Model (CNRM) which is suitable for often managed new generation networks based on IP, Ethernet and MPLS technology. The network architecture that the CNRM is intended for is depicted in Figure 2-1.

Such networks may be used for a number of purposes, implying the CNRM can be used in several different activation solutions:

- to provide corporate VPN services;
- to carry the traffic between (residential and business) customer sites and provider platforms for a range of services: Internet access, VoIP, IPTV;
- to carry other provider traffic, for example between different platforms in the provider's service network such as mobile backhaul from BST to BSC.

Figure 2-1 Network Architecture Modelled by Common Network Resource Model



The CNRM can be used, possibly with extensions and adaptations, for solutions in the NGN space. For solutions in other spaces a different model must be built as needed. For example, a solution to activate mobile services will need to model the various servers involved, like HLR, etc., which must receive information about subscribers, but the solution has no need to know the radio and backbone transmission networks.

In general the data model must contain the entities that are needed by the activation processes. It must be defined in terms of entity classes. The solution designer may define any desired entity classes.

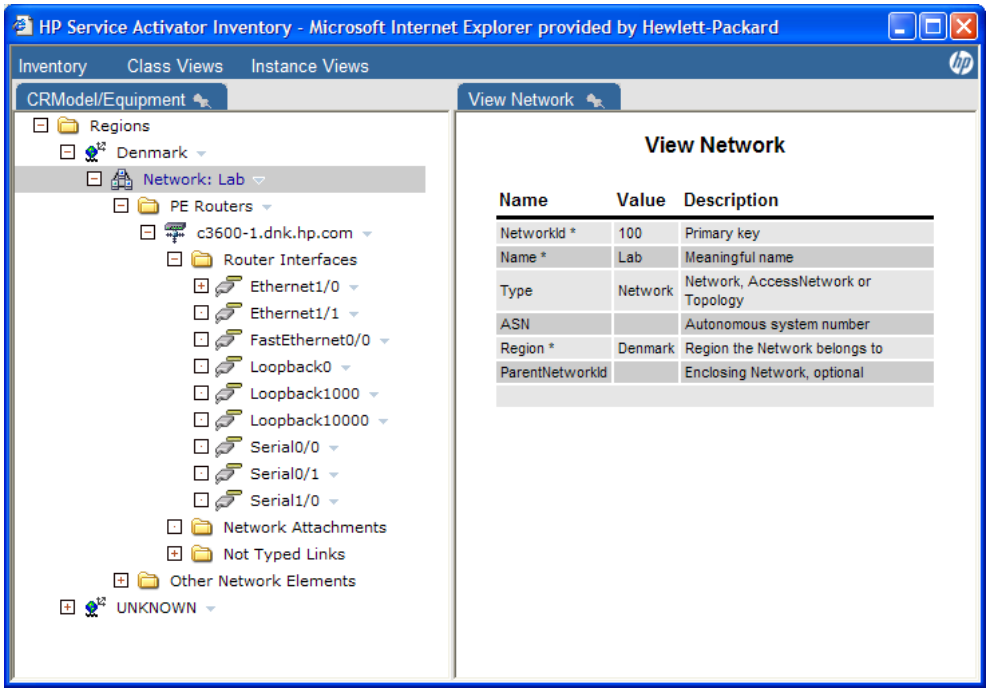
The data for each entity class will be stored in a separate table in the underlying database, and data entities can be accessed through Java bean objects from workflows. On the user interface different forms are used to create, display and edit entities of different classes.

Entity class definitions, known as “resource definitions” or “resource bean definitions” (the name does not imply that every entity must represent a resource), are given as compact XML-formatted files, one file per entity class. The definition format has a number of powerful features to allow the definition of fields, search keys, entity relationships, etc. Relationships between different entities are represented with foreign keys, i.e. inter-table pointers. Inheritance can be used to define an entity class which extends and modifies an existing entity class.

Deploying a data model from its definition in terms of entity classes is done with a tool, the Inventory Builder. The Inventory Builder reads the definitions and generates all the code that is needed for deployment: SQL statements that will create the database tables with indexes etc., Java code for beans to access the data for each entity class, and Java Server Pages (JSPs implemented with Struts) for forms to access the data from the user interface. The beans are used by the JSPs as well as by workflows. A number of workflow nodes are available in the built-in node library to create, query, reserve, release, update and delete repository data by means of the beans. The customizer will be aware that the beans and JSPs are created and deployed, but does not normally need to look into them.

HP Service Activator's user interface for accessing data in the repositories, known as the inventory user interface, is based on an explorer-style expandable tree structure where data entities are associated with branches of the tree. Entity relationships are used to define the child branches which appear when a branch is expanded. An example screenshot from the CNRM is shown in Figure 2-2. There is a graphical tool, the Inventory Tree Designer, to build tree definitions, branch by branch. A solution can include one or more tree definitions.

Figure 2-2 Inventory User Interface Screenshot



For a description of the inventory user interface, see the chapter "Inventory User Interface" in *HP Service Activator, User's and Administrator's Guide*.

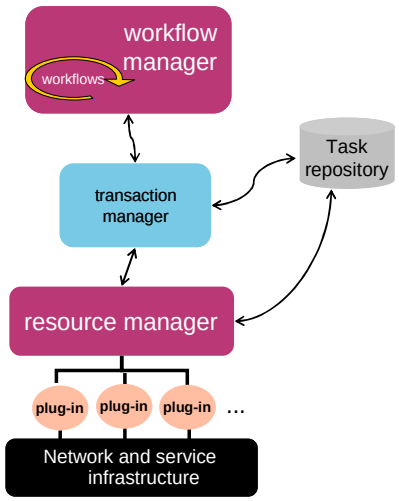
To build a working acquaintance with resource definitions, refer to example material introduced in chapter 3. For a full description of the details of resource definitions, tree definitions and the tools to manage them, refer to *HP Service Activator, Inventory Subsystem*.

For some solutions it may be necessary to integrate HP Service Activator with an external inventory. Approaches to address such a requirement are discussed in chapter 4.

Plug-Ins and Activation Tasks

In activation processes controlled by HP Service Activator workflows, interactions with target devices or systems, possibly through element managers, take place in Activate nodes. Activate nodes execute activation tasks. Three components of HP Service Activator are involved in the execution of an activation task: the workflow manager, the transaction manager and the resource manager, as shown in Figure 2-3, which is slightly more detailed than Figure 1-4 in chapter 1 in that the transaction manager has been added here. The task can be a single atomic task, or it can be a compound task, i.e. a list of atomic tasks.

Figure 2-3 Components Involved in Target Interactions



When an activation task is executed the workflow manager will pass the task information including all parameters from the activation node in the workflow job to the transaction manager. The transaction manager will retrieve the definition of the requested task from the task repository and, if it is a compound task, take control of the sequencing of atomic tasks and of rollback if a failure occurs. Parameters of a compound task are mapped to parameters for each atomic task. Each atomic task invokation with parameters is passed to the resource manager which will retrieve the corresponding plug-in Java class method from the task repository and execute it. During rollback, atomic tasks will be invoked in undo mode. Atomic tasks are responsible for communication with target devices or systems and pass to them for execution all appropriate commands or messages.

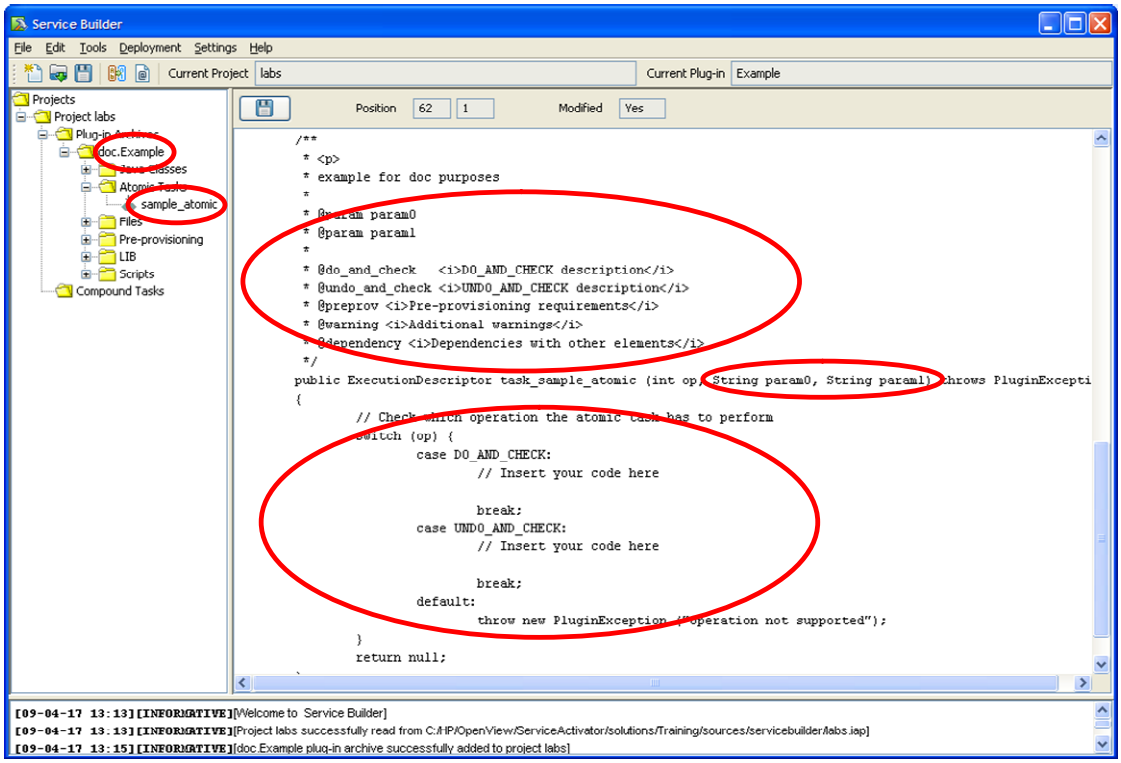
The transaction manager will return to the activation node in the workflow result information indicating whether the activation succeeded or failed; more detail on how this result is obtained is given below under "Plug-In Development".

For each solution one or more plug-ins will be needed to interact with the relevant targets. In many cases it will be possible to use one of the generic plug-ins, in other cases it may be possible to reuse a dedicated plug-in from a previous solution. Otherwise it will be necessary to develop a new dedicated plug-in.

Plug-In Development

Development of a new plug-in entails the writing of the atomic tasks as Java methods. The Service Builder tool is a graphical development environment that supports this work. Figure 2-4 gives an idea of how the Service Builder works and what must be added to implement a plug-in. Before the time of the screenshot a new plug-in called `doc.Example` has been defined in a popup form, then an atomic task called `sample_atomic` has been initiated using another popup form. The code that is shown (including the prefix `task_` for the method name) has then been generated by Service Builder with placeholders for comments that will be used to generate plug-in documentation as Javadoc, and for the code to implement the actions of the atomic tasks in the do and undo modes, respectively. The `sample_atomic` task takes two parameters named `param0` and `param1` (thus defined in the popup screen, any names can be used), the runtime values for which must be supplied by the invoking Activate node in the workflow that initiates execution of the task.

Figure 2-4 Atomic Task in Service Builder



The do mode (case `DO_AND_CHECK`) of an atomic task must always be implemented with appropriate Java code. The undo mode must also be implemented if it shall be possible to use the atomic task in a compound task in any sequential position other than the very last one. If there is no such requirement, implementation of the undo case can be omitted.

An atomic task, when executed in the do mode, may fail, when for some reason it is not possible to complete the necessary interactions with the targets. If possible, the atomic task must fail cleanly, the target should be left in the same state as when execution of the task began. As Figure 2-4 shows, an atomic task must return an `ExecutionDescriptor`. The `ExecutionDescriptor` makes it possible to distinguish between success and failure, and between clean failure and "dirty" failure, i.e. the second order failure, when a change that was made to the target cannot be removed. The `ExecutionDescriptor` also includes a descriptive text field. The information collected from all atomic tasks that are executed in the forward path (do mode) of an activation task is passed back to the calling workflow and may subsequently be processed in the continuation of the workflow. Result information from atomic tasks executed in the rollback path (undo mode) is not made available to the workflow.

The Service Builder packages a plug-in as a single file in an archive format known as a plug-in archive identified by the file name suffix `.par`. To make use of existing Java libraries in a plug-in, the `.jar` files can be included in the LIB folder of the plug-in archive. Plug-in archives can be deployed into the task repository of an HP Service Activator system.

Compound tasks do not belong to plug-ins. Typically a compound task will include atomic tasks from more than one plug-in. The Service Builder can also be used to build static compound tasks, i.e. compound tasks with fixed member atomic tasks. An XML-formatted file which describes the compound task is produced for distribution.

The Service Builder can also deploy plug-ins and compound tasks into HP Service Activator's task repository from the distribution files (`.par` and `.xml`, respectively).

Customizing Control Templates for Generic Plug-ins

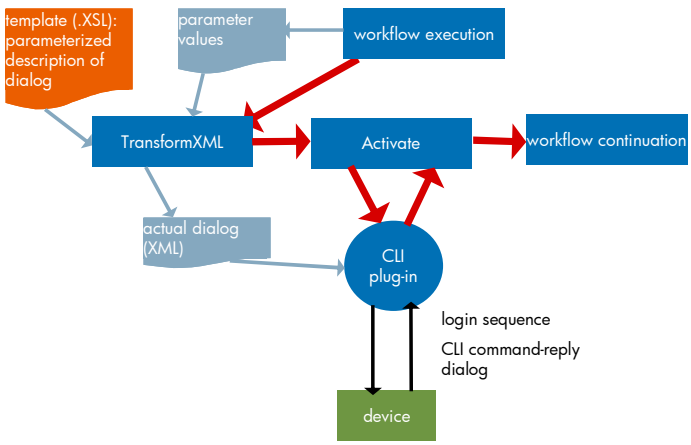
Two generic plug-ins are supplied as part of the HP Service Activator core product: the generic CLI plug-in and the generic HTTP plug-in. These plug-ins implement communication protocols for command line interface communication and for HTTP, respectively. But they do not include information about commands or HTTP messages to send or what to expect as responses. The controlling information must be prepared by the calling workflow and supplied in the form of task parameters.

The CLI plug-in is controlled by an XML-formatted dialog control document. A dialog, for example with a complex router device, may comprise a large number of commands, and so the control document may be quite large. The control document will contain information about how to establish a session (log in) with the device, a sequence of command-response exchanges, and finally how to terminate the session. The specification for each command response exchange includes the exact command line to be sent, the expected response (next prompt) and patterns that allow recognition of possible error responses. Note that even a long dialog with many commands is executed as a single atomic task. In order to ensure clean failure of the task when an error response is received for a command, the CLI plug-in supports rollback of the command sequence (sub atomic task rollback). Command sequence rollback uses rollback commands that can be specified for each command-response exchange that is part of the sequence.

The HTTPPost atomic task supported by the generic HTTP plug-in makes a single HTTP Post call to a specified target. It supports features such as HTTPS with exchange of certificates and use of a proxy. The main parameter to achieve the intended effect is the HTTP Post message body, typically in SOAP format. This plug-in can be used when all necessary message formats are well known and simple enough that it is convenient to prepare templates for them and decode the responses by parsing XML in the workflow.

The generic CLI and HTTP plug-ins are quite different. But in both cases, there is a main parameter which takes the form of a document, the command sequence dialog control document and the HTTP message body, respectively. And in both cases the document will typically include a number of strings whose values must be sourced from variables of the workflow job that invokes the atomic task, representing items such as: user name and password for authentication of the session/request by the target, names of devices, ports, interfaces and other objects that exist and must be manipulated on the target. The way to handle this situation is to prepare a template document with replaceable placeholders for variable values, and to use a workflow node to substitute the actual values for the placeholders to obtain the final document to submit to the plug-in. The resulting workflow logic will be as shown in Figure 2-4, where the TransformXML node is used to achieve the parameter value substitution. For simple cases the ComposeMessage node can also be used.

Figure 2-5 Parameter Value Substitution for Generic Plug-in



In summary, when a generic plug-in is used, the target specific interaction control information will be found, not in the plug-in, but in the control document templates.

Workflows

Workflows are programs which define the activation processes of an HP Service Activator solution. An execution of a workflow is called a job. The workflow manager is the operating system for workflow jobs: it starts new jobs when called upon to do it, it executes the workflow nodes of running jobs in the proper sequence, it persists and safeguards the states of running jobs, and it manages the sharing of a number of execution threads by all concurrent jobs. Refer to chapter 6 for a description of how the workflow engine can scale by deploying the workflow manager on a cluster of HP Service Activator servers.

As programs, workflows consist of a number of directionally connected nodes. Each node performs a basic action. The workflow manager executes one node at a time:

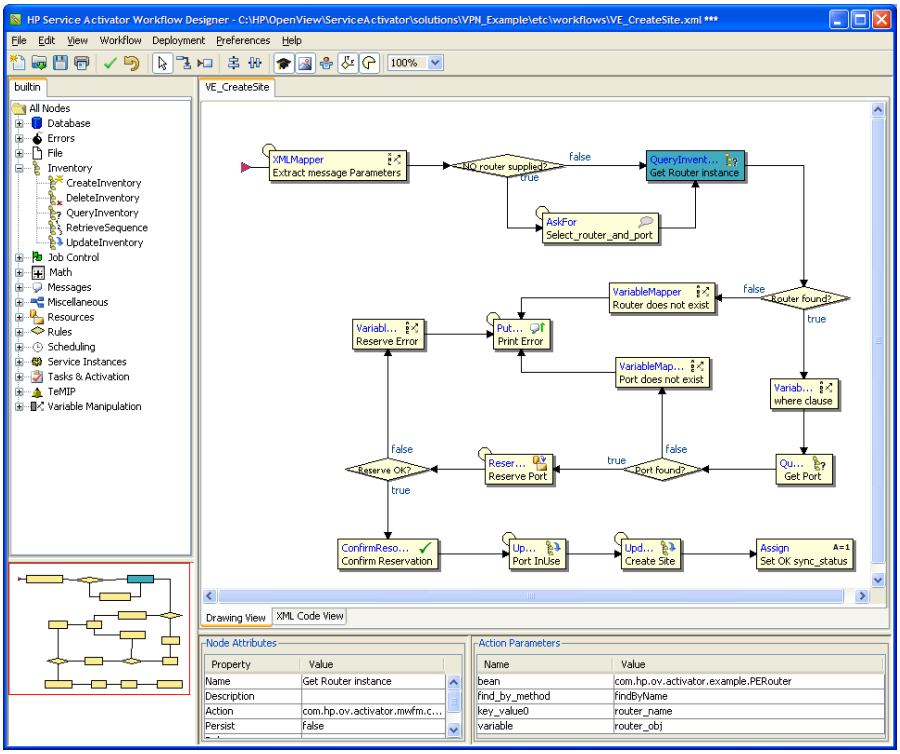
- select a ready job
- run the next node for the selected job
- save the state of the job to the workflow job repository

Persisting the states of workflow jobs between node executions makes the workflow engine robust. It allows HP Service Activator to stop and restart at any time, whether intended or unintended, and workflow jobs will just continue from the point where they were persisted. In practice it is not necessary to persist the state after every single node, therefore it is possible per workflow node to disable saving of the job state to optimize on processing time.

The state of a workflow job consists primarily of its case packet: the values of a set of variables, some common ones are predefined for all workflows, others are added as part of the workflow definition.

A library of workflow nodes are available that can be used to compose workflows. Workflows are composed with the Workflow Designer, a tool with a user interface that displays a workflow as a directed graph, with nodes connected by arrows to represent their algorithmic sequence. Two special types of nodes allow the introduction of branching in the sequencing logic, based on values of variables: rule nodes, for two-way branches, and switch nodes, for n-way branches. Figure 2-6 is a snapshot of the Workflow Designer, with the library from which nodes can be picked on the left and the workflow chart with nodes, arrows and branches on the right.

Figure 2-6 Workflow Designer



The built-in node library covers the functions which are generally needed in activation workflows. Nodes range from very powerful ones which accomplish significant tasks, like transformations of XML documents, to simple ones, like comparison of two values or simple string manipulation. Functions accomplished with nodes include:

- inventory operations: create, query, update, delete repository data using beans
- reserve and release resources
- execute activation tasks, through the transaction manager
- spawn child workflow jobs
- wait to get data from outside the workflow job: from an operator through a popup dialog, from another workflow job, or from an external source via a programmatic interface
- send message (or email) to an external system
- various transformations of documents, typically XML formatted, for parsing of incoming request messages and preparation of messages to be sent
- post messages to be shown to operators, process log entries for troubleshooting purposes and audit trail records
- liaison functions for integration with other relevant HP products, TeMIP and uCMDB

Data manipulated by a workflow is held in variables that are passed to and from workflow nodes as input and output parameters. Typically, the request message whose receipt triggers execution of a workflow is passed into the workflow job in a preinitialized variable, the first nodes of the workflow will extract field values from the message into other, simpler variables. Some of these values will be used as keys for retrieving data from repositories, and retrieved data will be placed in further variables, etc., until all the parameters that are needed for the actual activation are held in a set of variables, from where they are passed to the activation task through an activation node.

A number of types are supported for variables, reflecting those of the underlying Java language: String, Boolean, Integer, Float, Object. Objects can be beans, maps, arrays and vectors.

The input and output parameters for each node in a workflow are specified as constants or mapped to workflow variables by means of the Workflow Designer.

Workflow Structure

Workflow jobs can spawn child jobs to execute other workflows. The parent may initialize variables of the child and may also synchronize with the child to retrieve result values back into its own variables. A single workflow job is restricted to a single string (thread) of node executions. Parallel algorithms can be realized by spawning multiple child jobs to work concurrently, for example to optimize activation of multiple independent devices.

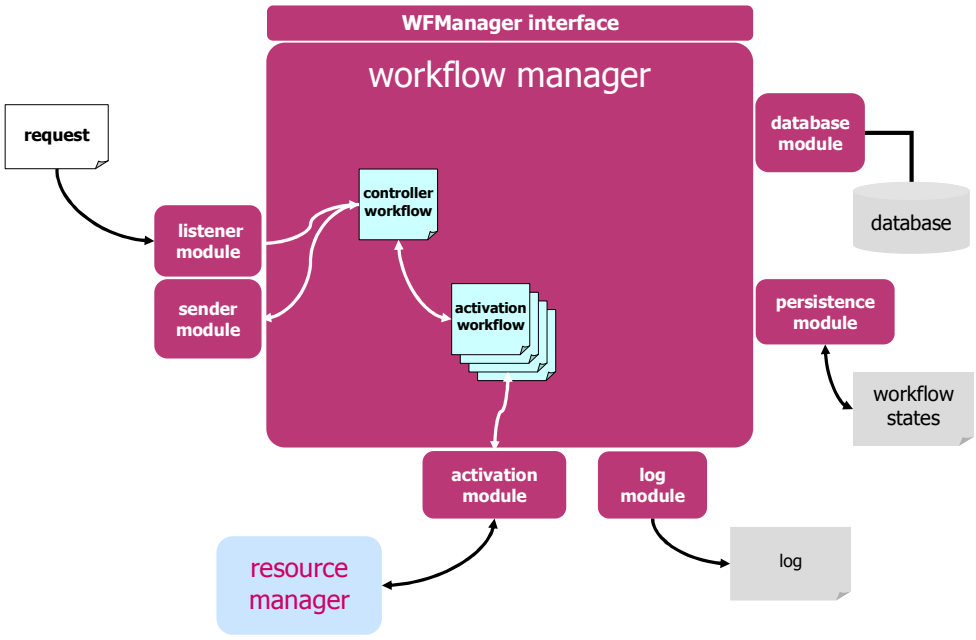
Multiple workflows working in parent-child relationships can also be used to architect complex processes in layers. A good practice is to use one workflow, designated as the *controller* workflow, to deal with northbound communication and process orchestration, and use dedicated workflows, spawned from the controller, to deal with service and device specific details.

Workflow Manager Architecture

The workflow manager has a core which is concerned with the execution of workflows, one node at a time, as described above. Other functions of the workflow manager are implemented as pluggable *workflow manager modules*. Modules are used for all functions which interface the workflow manager to its environment, like synchronizing with external sources of input, sending messages and emails, accessing the database, interacting with the transaction manager to execute activation tasks, etc., as illustrated in Figure 2-7. All tasks which involve wait points are generally performed by modules which run in separate threads from nodes of active jobs, to avoid depleting and potentially even deadlocking the power of the core engine.

Other examples of functions performed by workflow manager modules are management of scheduled workflows, authentication of users and RMI connections, writing of logs and audit trails, job distribution within a cluster (see chapter 6), and collection of statistics.

Figure 2-7 Workflow Manager Architecture



Algorithms residing in workflow manager modules can easily be replaced, because modules are pluggable. In some cases, like authentication, several modules are provided in the product distribution, making the choice configurable.

Each workflow node is implemented as a Java class, extending a base node class with specializations for process nodes, rule nodes and switch nodes, and similarly with modules. As with nodes there is a built-in library of modules, providing a wide range of generic capabilities. The libraries can be extended by adding new nodes and modules. Extensions can be provided in solution packages (see below), or they can be implemented as customizations for a delivery project.

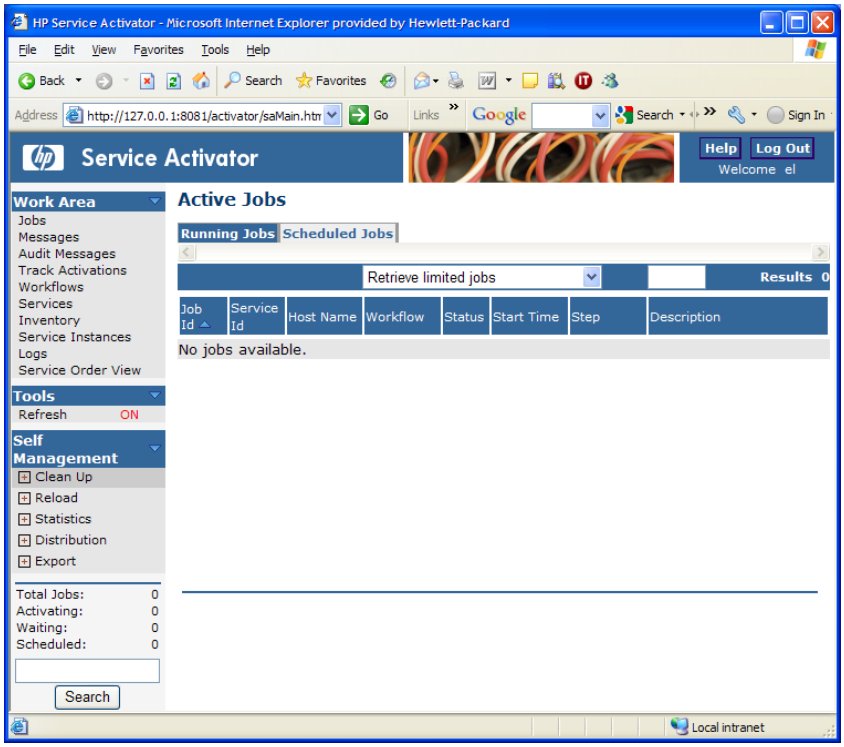
User Interface and Roles

HP Service Activator supports a web browser based user interface, which does not rely on any components to be installed on the client side. It is implemented as a collection of web pages which are executed in the web server component of the HP Service Activator platform (Apache Tomcat in JBoss). The pages are implemented in multiple technologies: plain HTML, Java Server Pages and Java Server Faces.

The inventory user interface that was mentioned under "Solution Data Repositories (Inventory)" above is launchable from the main window of the HP Service Activator user interface as a separate window. A special feature that can be customized in the inventory UI is an operation to start a workflow job with input parameters that can be sourced from entities in the repository or entered by the user.

Other functions are available within the working area of the main window, selected from menus in the left hand side. Figure 2-8 shows the menu with the active jobs list (empty) selected and displayed in the working area. The Work Area and Tools menus can be customized; the ones shown here are the installation defaults.

Figure 2-8 User Interface Screenshot



The user interface allows operators to monitor activity within the workflow engine, including active activation transactions. Information about active jobs can be filtered by values of key characteristic variables of the job, such as order id (identifier of the request that triggered the job), service id (identifier of the service being activated) and type and state of the workflow.

Some HP Service Activator solutions require operators to interact with workflow jobs through the user interface. There are also a number of housekeeping functions (Self Management) for the system administrator. The menu and other capabilities that each user gets via the user interface is controlled through roles that are established through log-in authentication. Only the admin role has access to the Self Management functions, other users may be specialized to interact with certain workflows, but not with others. The data that users may view and edit via the inventory user interface can also be controlled by roles.

The texts that are displayed on the user interface are organized as resource bundles and can be translated and localized by the system integrator.

For a general description of HP Service Activator's user interface, see *HP Service Activator, User's and Administrator's Guide.*

Interfaces for Integration

The main integration of HP Service Activator for a solution is normally the northbound integration to the CRM / Order Management system which is the source of activation requests. Northbound integration may be realized in different ways depending on the requirements.

The "classical" northbound interface for HP Service Activator uses workflow manager modules for listening to request messages and sending response messages. This approach is illustrated in Figure 2-7. Listener and sender modules, which can receive and send messages in any format, are provided in the built-in workflow manager module library for TCP sockets and for JMS. With this approach, receiving and sending are done on separate connections and are asynchronous. Such decoupling is desired whenever workflow jobs can have non-neglible duration, to avoid occupying connection resources while workflows are running.

Sender and listener modules can also be used to implement peer interfaces to other OSS systems, with one-way notifications or two-way interactions as required.

An alternative to listener and sender modules is to use a web service interface, so that requests are received by the web server component of the platform (JBossWS in Apache Tomcat) and handled by a servlet which can start workflows. A tool is provided, the Web Service Designer, which can generate the servlet that will expose a collection of workflows as web service methods that allow them to be run conveniently by a client. A WSDL document defining the exposed interface for import to a client system can be generated.

The native generic RMI for the workflow manager is also available in a web service version. With this interface the method to start a workflow job can be called by forwarding a SOAP formatted request specifying the name of the workflow to run and initial values for workflow variables as parameters. A web service as described in the preceeding paragraph is just a convenient specialization of the capabilities of this interface.

The section "Northbound Interface" in chapter 5 elaborates on the two approaches to northbound integration that have been outlined here.

Integration with Other HP NGOSS Products: NNMi, NA, uCMDB

An important aspect of HP Service Activator is the support for solutions which include also HP NGOSS products such as NNMi, NA and UCMDB, and where the products work seemlessly together and mutually enhance each other's capabilities.

HP Service Activator V5.1 includes support for interworking with NNMi V9, UCMDB V8.0 and NA V7.6.

The HP Service Activator product kit includes a number of components which serve to bind the products together, mostly in the form of hooks such as workflow nodes and plug-ins dedicated to interaction with the other products in order to allow the building of integrated solutions, but also readily usable capabilities such as UI crosslaunch with single sign-on and workflows which can load data into the Common Network Resource Model from NNMi.

Chapter 10 is about integration with NNMi, and chapter 11 about integration with NA. These two chapters have similar organization, discussing the positioning of the product concerned to set the scene for integration, the potential benefits of integration, the integration capabilities that are available out-of-the-box, the components (hooks) that facilitate customized integration, and how to customize and configure an integrated solution.

There is not a chapter about integration with UCMDB. The components for UCMDB integration are a number of workflow nodes (Create/Delete/Query/Update UCMDBCIsandRelations) which can be used to access data in a UCMDB data repository with similar power to those nodes which are used to access the native inventory of HP Service Activator for the case where the inventory for a solution will consist of data managed with UCMDB.

Solution Deployment

An HP Service Activator solution comprises different customized parts that are separate from and added on top of the core product framework: inventory data model, workflows, plug-ins, etc. All these parts have source files, mostly different XML documents, which are prepared using the appropriate specialized tools. To become operational on a running HP Service Activator system, all the customized parts of a solution must be deployed into the static repository and file structures where they are accessed at runtime.

Deployment can be done with the specialized tools: the Workflow Designer can deploy workflows, the Inventory Builder can deploy resource definitions, etc. An entire solution can be deployed in a single operation with the *Deployment Manager*. The Deployment Manager is an umbrella for the specialized tools, it will use each one of them as needed.

In order to be managed with the Deployment Manager the source files for a solution must be arranged in a directory hierarchy which obeys a certain structure. The Deployment Manager can create the structure, and it can import entire solutions from zip or tar archives of the structure. You are not forced to keep your source files in a solution hierarchy, but it is strongly recommended to name and use a solution structure.

See chapter 3 for illustrations of the solution directory structure and screenshots of the Deployment Manager.

It is possible to have multiple independent solutions on a single HP Service Activator platform. All sources for each solution must include a short name of the solution, which is used to keep the deployed solutions separate, and must obey some naming conventions to avoid name clashes, for example of database tables. The Deployment Manager can deploy and undeploy each solution in turn.

The Deployment Manager is particularly useful with clustered platforms, where HP Service Activator is running symmetrically on multiple servers. It will ensure that a solution is identically deployed on all servers within the cluster. The Deployment Manager can also manage versions of solutions, where some of the source files are updated.

3 An Example Solution: Intro_Example

This section walks you briskly through a very simple example solution, too simple to be realistic, which has components of all the types that have been described. A larger example with more substance, yet still only an example for study, not activating real services, is described in a separate manual, *Putting Service Activator to Work: A Sample Service Scenario for VPLS*. Both examples are found as zip files in the `$ACTIVATOR_OPT/examples` directory when HP Service Activator has been installed, named `Intro_Example.zip` and `VPN_Example.zip`, respectively.

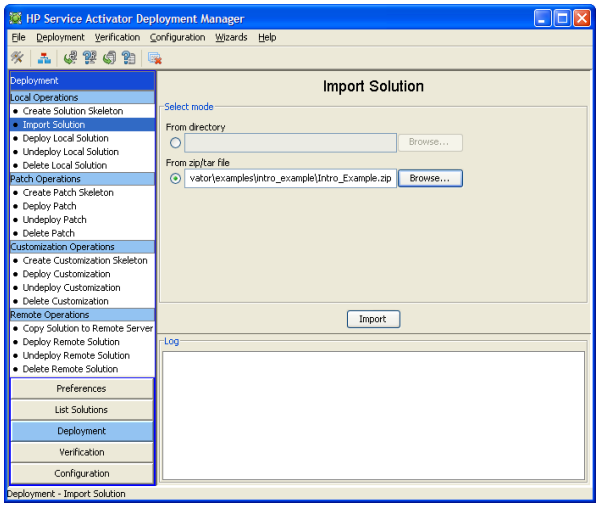
The Intro_Example is our subject here. The service that it activates is a point-to-point connection terminating on two switches assumed to be customer facing. The switches with the ports where the connections terminate are modelled as resources. There are workflows to create and delete connections, working through a plug-in that activates the service by configuring the endpoint switches. It doesn't really do that, it only writes some log entries. The Intro_Example uses the socket listener and sender modules to implement its northbound request interface. The listener module receives request messages and then starts the controller workflow.

Throughout this section you will be instructed to use the different HP Service Activator tools to investigate the solution components. This assumes you have installed HP Service Activator. After installation on a Windows platform you can launch the tools, the documentation and the user interface (in a browser) from the `start -> all programs -> HP Service Activator` menu, or from desktop icons. For other platforms, consult the installation guide in question.

Contents of the Intro_Example

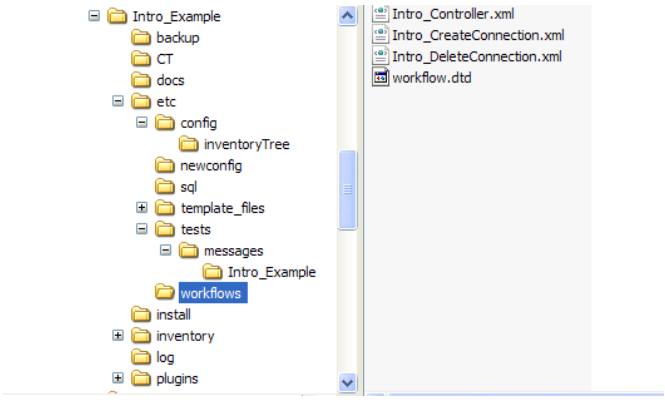
To access the source files of the Intro_Example, launch the Deployment Manager, select `Deployment -> Import Solution`, and browse to the zip file, as shown in Figure 3-1. Click `[Import]`.

Figure 3-1 Deployment Manager, Import Solution



The zip file contents will be unpacked into dedicated directories for the different solution components, according to the conventions of the Deployment Manager, under `$ACTIVATOR_OPT/solutions/Intro_Example`, see Figure 3-2. We will reference the component directories with their names relative to this solution root directory.

Figure 3-2 Intro_Example Solution Directories



Browsing the various component directories, you will find:

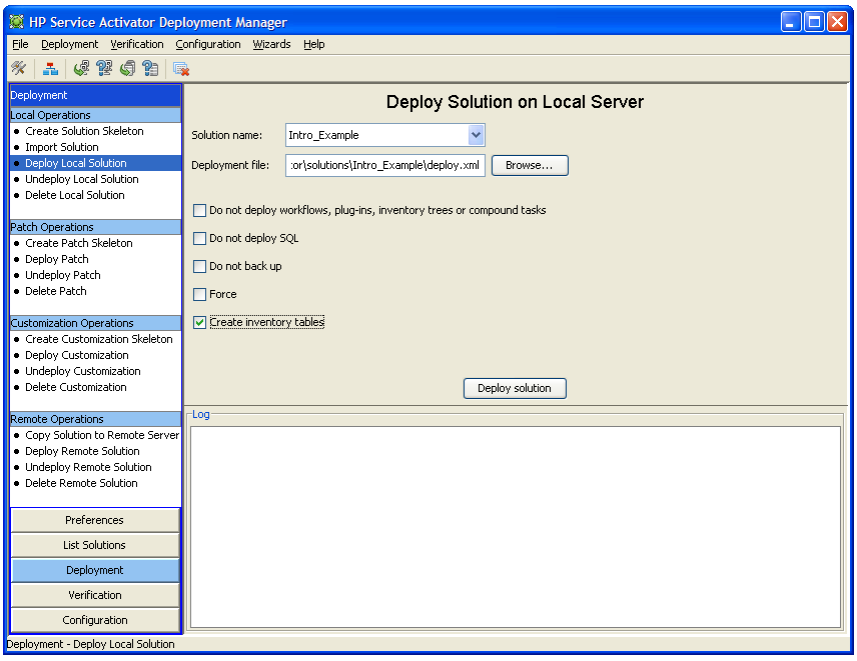
- In `plugins`, the plug-in archive file `Intro.Switch.par`. This is the plug-in for switch devices in deployable plug-in archive format. It has two atomic tasks:
  - `createConnectionEndPoint`, intended to create a unilateral connection from a customer facing port on one switch to a port on another switch; both switches and ports are identified by parameters
  - `deleteConnectionEndPoint`, intended to delete a unilateral connection from a customer facing port on one switch to a port on another switch; again both switches and ports are identified by parameters
- In `CT`, deployable definitions of two compound tasks that are invoked from the workflows: `Intro.CreateConnection` and `Intro.DeleteConnection`. Both tasks take parameters specifying two endpoints, each given by a switch and a port on that switch. The first task creates a bilateral connection as two unileateral connections between the two endpoints. The second task similarly deletes a bilateral connection.
- In `etc/workflows`, three workflows:
  - `Intro_Controller`, controller workflow which is started by the socket listener. It parses the request message, extracts the fields (tagged elements) of the message to variables, and uses the value of the *action* field as the name of a workflow to run as a child job; it must be one of the other two workflows.
  - `Intro_CreateConnection`, workflow for creating a connection, specified by variables initialized from the controller.
  - `Intro_DeleteConnection`, workflow for deleting a connection, similarly.
- In `inventory`, resource definitions for two resource data entity classes, `IntroSwitch` and `IntroPort`, where a port belongs to a switch identified by a foreign key on the port entity.
- In `etc/sql`, scripts to populate the resource inventory with a few switches and their ports.
- In `etc/config/inventoryTree/intro-Tree.xml`, the definition of a simple tree in which switch and port data is made accessible on the inventory user interface.

- In `etc/tests/message/Intro_Example`, sample request messages (`intro-Message-1.xml` and `intro-Message-2.xml`) with format definition in `etc/config/intro-Message.dtd`. You can use HP Service Activator's regular user interface to select one of these messages and inject it for processing (send it to the port configured for the socket listener module). This feature must be enabled in the configuration file for the user interface (*`$JBOSS_ACTIVATOR`*`/WEB-INF/web.xml`) by setting the value of the parameter named `tests` to **`true`**. Do this before or after you deploy the Intro_Example solution, but before you start HP Service Activator.
- In `etc/template_files/Intro_Example`, templates for response messages to be sent after processing of a request (`ERROR_intro.template` and `OK_intro.template`) and for an information pane to be shown to an operator.
- In `newconfig/mwfm.xml`, a fragment to be copied into the configuration file for the workflow manager (`$ACTIVATOR_ETC/mwfm.xml`) to configure the socket receiver and sender modules. Do this before or after you deploy the Intro_Example solution (peruse the long configuration file in an editor to find a proper place for the fragment), but before you start HP Service Activator.

Deploying the Example

Once you have unpacked the solution zip file as described above, use the Deployment Manager to deploy the complete solution. First you must configure the Deployment Manager to be able to access the Oracle database; select `Preferences -> Configure Database Connection`, enter the username and password for your HP Service Activator installation, and click `[OK]`. Then select `Deployment -> Deploy Local Solution` and then in the drop-down list labelled `Solution name:`, which shows all undeployed solutions present (unpacked under `$ACTIVATOR_OPT/solutions`) on the server, select `Intro_Example`. Check the `Create Inventory Tables` field and click `[Deploy solution]`.

Figure 3-3 Deployment Manager, Deploy Solution



The Deployment Manager will then deploy all components of the solution into HP Service Activator's internal directories and static repository (database tables).

Make sure you have changed the configuration files for the workflow manager (`mwfm.xml`) and for HP Service Activator's user interface (`web.xml`) as described above under "Contents of the Intro_Example". In the workflow manager configuration file you should also enable the auditor module: search for `auditor`, uncomment the `<Module>` element for the `DBAuditModule`; set the parameter `store_audit` to have value `"true"`. This module is used to write audit trail records, and the workflows in the example generate such records.

When this has been done, you must start HP Service Activator (restart, if it is already running) to be able to run the solution.

Examining Components of the Intro_Example Solution

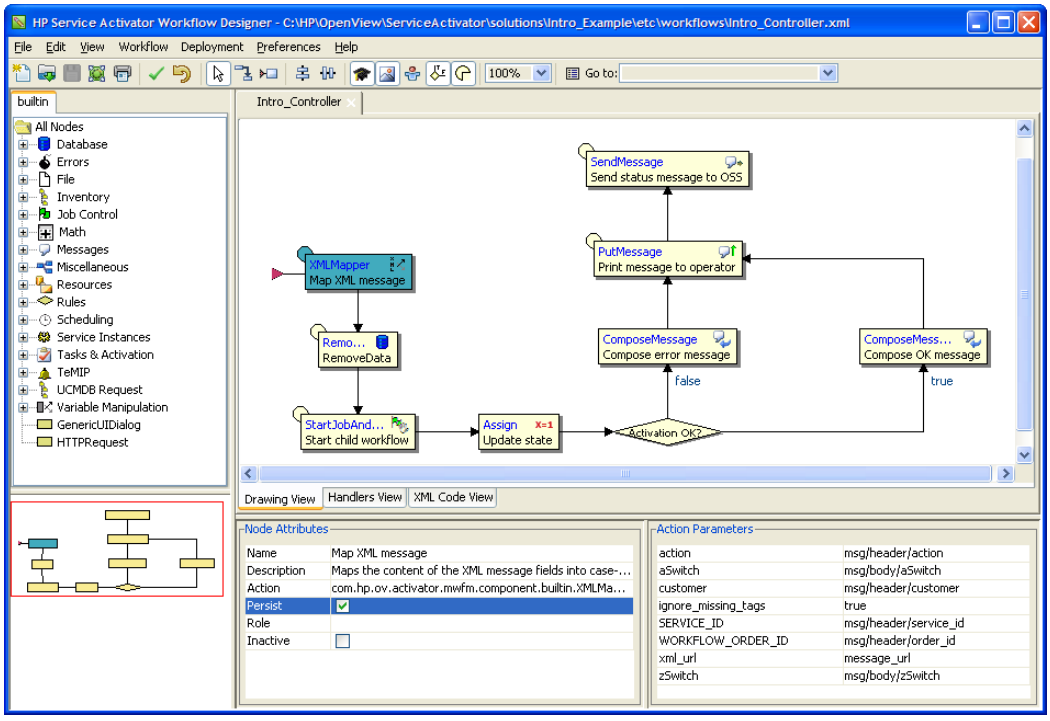
We will now take a closer look at the components of the Intro_Example solution.

The **plug-in** can be imported from the archive file `Intro.Switch.par` into the Service Builder and studied at the source code level, if desired. You will need to create a Service Builder project as a workspace to import the plug-in. We will not go into the details here, refer to *HP Service Activator, Developing Plug-Ins and Compound Tasks*. The plug-in is documented with Javadoc, which you can study to understand the atomic tasks, their parameters and actions. To access the Javadoc you can extract the contents of the archive with Winzip, go to the `doc` directory, and double-click on the html file to open it in a browser.

The **compound task** definition files can also be imported and studied in the Service Builder which has dedicated functions for editing compound tasks. Alternatively, the xml files, which are quite straightforward, can be studied in a text editor. Be sure to understand the sequence of parameters of each compound task.

To study the **workflows**, launch the Workflow Designer, and open the workflow definition files. The initial location of the file open window will be `$ACTIVATOR_ETC/workflows`, you will need to navigate to the solution's workflow directory to open the files. With the Intro_Controller workflow open, the Workflow Designer appears as in Figure 3-4.

Figure 3-4 Intro_Controller Workflow in Workflow Designer



The workflows are simple. It is possible to understand the algorithms by inspecting them, node by node. Some clues are given here:

In Figure 3-4 the first node of the controller workflow, an XML mapper node, is selected. It extracts from the request message, which has been received by the listener module and passed to the workflow job, several values identified by XPath-like tag sequences to workflow variables. You can decipher this from the Action Parameters shown in the lower right corner (variable names in Name column, tag sequences in Value column). Take a closer look at one of the sample messages (in `etc/tests/messages/Intro_Example`) to understand how the tag sequences relate to the message format. You may note the use of the variables WORKFLOW_ORDER_ID and SERVICE_ID to hold the identifiers for the order and the service, respectively. These predefined variables are used by convention to make it possible in a general way to display information from running workflow jobs. The same is true for WORKFLOW_STATE, whose value is intended to give the observer an idea of how far the workflow job has progressed whenever it is inspected. The values of these predefined variables are automatically inherited to child workflows.

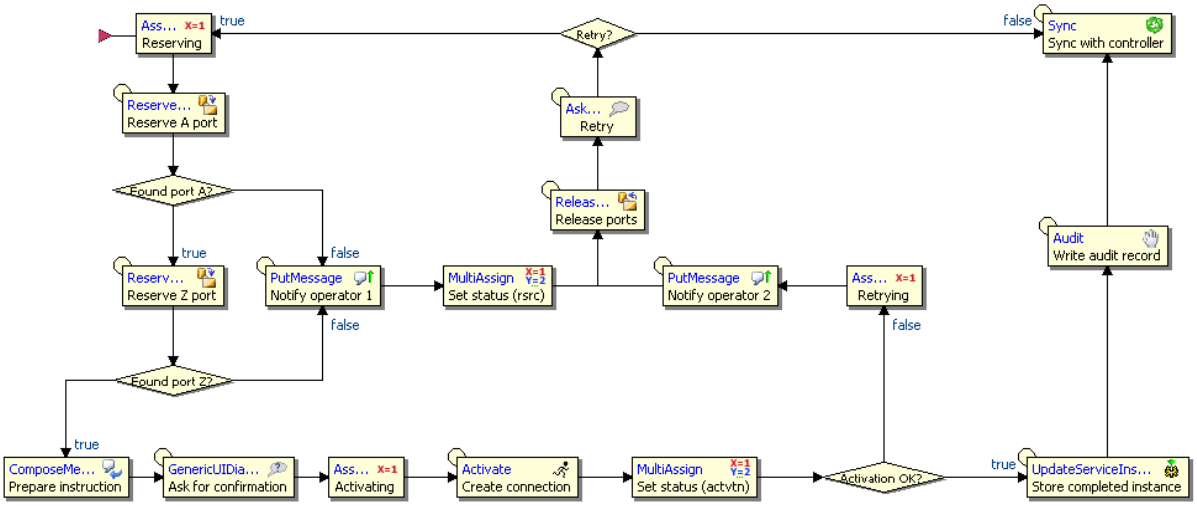
The controller workflow starts one of the other workflows to do the actual work that was requested. The name of the workflow is extracted literally from the request message. In a more complete implementation it would be appropriate to perform a database lookup to validate the request.

The child workflow in its final Sync node (true for both child workflows) returns a status that is inspected by the controller to determine if the child completed its task successfully.

The ComposeMessage nodes in the controller substitute variable values for placeholders in templates to create response messages which the controller will send back to the originator of the request. The PutMessage node posts the response message into the message area of the user interface, so you can easily see it when the workflow has run. Inspect the message template files (in `etc/template_files/Intro_Example`) to understand how the value substitution works.

The Intro_CreateConnection workflow (shown as screenshot in Figure 3-5) begins by allocating and reserving ports for the A and Z ends of the connection on switch devices which are specified by values that originate from the request message and have been passed from the controller workflow. This is done with ReserveResource nodes. Then follows the activation which invokes the compound task that will configure the switches. The workflow has looping logic to repeat the attempt after a failure. The GenericUIDialog andAskFor nodes synchronize with an operator; the one in the Retry loop will ask the operator to decide whether a retry is appropriate. The node labelled “Ask for confirmation” posts an information panel with a description of the connection that will be created, and waits for the operator to read the information and perform the required cabling before allowing the workflow to proceed, but does not take any input from the operator.

Figure 3-5 Intro_CreateConnection Workflow

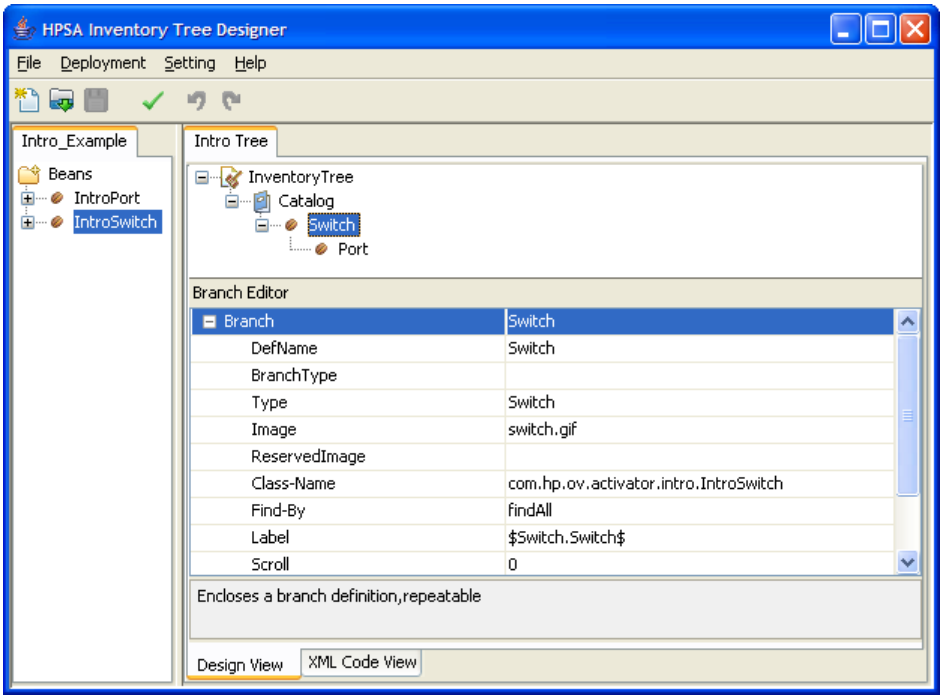


After a successful activation information about the newly created service is recorded as a simple service instance. Service instances are generic and predefined, the system integrator does not need to prepare a definition; they are stored in a database table that is created at installation time, not by the Inventory Builder. A service instance includes a unique identifier (database key) and contains the names and values of a number of additional variables, identified by name. This predefined capability can be used if services are simple and you have no special requirements for a service data repository, its structure or presentation. Finally, the sync node at the end of the workflow passes information back to the *operation_status* and *operation_description* variables of the controller workflow where they are used to determine the choice and contents of the response message.

When you have understood the process to create a service, the Intro_DeleteConnection workflow will be straightforward. It is supplied with the service identifier, retrieves all additional information about the service to be deleted from the service instance and unravels everything that was done during creation.

The **resource definitions** for resource data entity classes can be processed with the Inventory Builder. But to inspect these XML files you must use an editor. During deployment some instances of switch and port data were populated into the tables defined by the resource definitions. After you have launched HP Service Activator's main user interface (when you are running the solution, see below) you can view this data in the inventory user interface which is controlled by the **tree definition** that was also deployed, from file `/etc/config/inventoryTree/intro-Tree.xml`. The InventoryTreeDesigner tool can be used to inspect the tree definition itself as shown in Figure 3-6.

Figure 3-6 Intro Tree in Inventory Tree Designer



Running the Intro_Example Solution Workflows

With HP Service Activator running, launch the user interface in a browser and log in; if you have not configured an authentication module, you will not need to type a password.

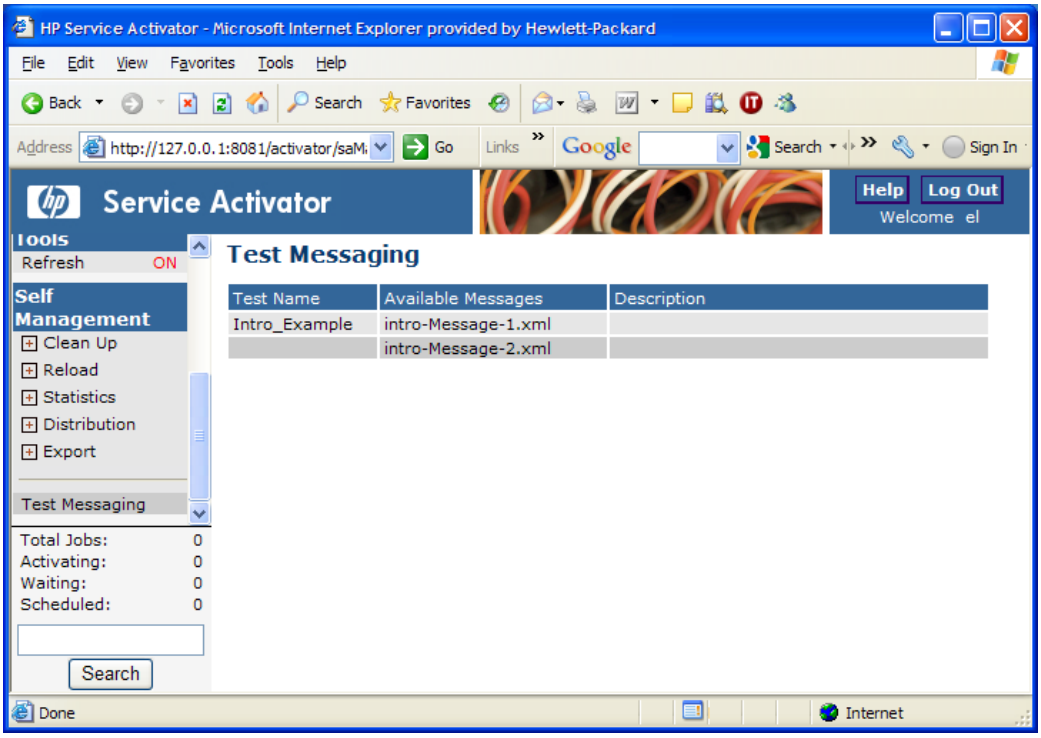
Inspect inventory. In the left hand menu of the main user interface window click `Inventory`, and the inventory user interface opens in a dedicated window. In this window select `Instance Views -> Intro Example/Intro Tree`. Then expand the tree and click in the sensitive area of a switch or port branch (or right-click to get an operations menu) to view the forms that display the field value data for the entity associated with the branch. Compare the data you see in the view forms to the fields defined in the resource definitions. If you inspect the tree after you have created a connection you will notice the ports that have been reserved for the activated service are marked with an **r**.

Make request to run workflow. You can select and inject one of the sample request messages to run the workflows. In the left hand menu, open the Self Management part at the bottom by clicking the little triangle. Then select `Test Messaging`. A list of the two sample messages will appear in the work area as shown in Figure 3-7. The first message contains a request to create a connection, the second message a request to delete the same.

Now right-click on one of the two message names, and then click `Start Test` in the single-item pop-up menu that appears. The message will be sent to the listener module, which will start the controller workflow. The resulting workflow job will read and process the message. The controller job will start a child job to do the requested work.

As you know from examining the workflows, they contain AskFor nodes that synchronize with an operator. To perform this interaction, select `Jobs` at the top of the left hand menu. In the job list that appears, select the tab labeled `Intro` (the tab label is specified by a parameter of the AskFor node). You will see a list of jobs waiting for interaction. Unless you have injected more than one request message, there will be just one job in the list. Right-click on it, and select `Interact with job` in the pop-up menu that appears. Study the pop-up window and click `[Submit]` to close it and signal to the workflow job to proceed.

Figure 3-7 User Interface with Test Messages



Inspect effects of workflow. The simplest way to study the result of running the workflow is to view the request response message. The sender module will attempt to send it, but since no application is waiting for it, the sending will fail. It is also posted in the message list on the user interface, and you can view it there. Select `Messages` in the left hand menu to see the message display. The posted messages will appear under the tab `Intro` (the tab is specified by a parameter of the PutMessage node in the controller workflow). Compare the message you see to the template from which it was generated.

When a request has been successfully it is documented with an audit record which you can inspect in the Audit Record view (select `Audit Messages` in the left hand menu).

When a service has been created you can inspect the service instance record that has been stored to hold information about it. In the left hand menu, click the '+' to expand `Service Instances`, then click `Open Instance`. A special window will then open; click the '+' to expand the list of service instances, and click on the service identifier of the branch line to view the details.

Another effect you can study is the warning messages written by the atomic tasks in the log of the resource manager. Select `Logs` in in the left hand menu, then select the `RESMGR` tab at the top of the work area, and then the `resmgr_active` file name at the bottom. Scroll down the file to see the yellow entries (these entries are written as warnings, hence the yellow color, so they will be easy to spot). If you study the Java source code of the plug-in you can find the statements that log the warnings. The other entries are written by the resource manager itself. Try to make out what they document. You can also view the log of the workflow manager (`Logs` -> `WWFM` -> `mwfm_active`); here you will see error entries in red color written by the sender module when it fails to send the request response message.

Finally you can inspect the inventory user interface to find the ports that are reserved and used when a connection service is created.

4 Solution Planning and Analysis

This chapter gives an overview of the activities that must be undertaken in a project to build a solution based on HP Service Activator and explains the aspects of a solution that must be analyzed early on to establish a solid base for scoping and planning a delivery project.

Activities in a Project to Build a Solution

Some solution delivery projects will start with a solution that exists more or less off the shelf, in the form of a solution package or a similar solution that has been delivered previously. Other projects will start from scratch or with very little in the form of reusable customization. Even when a rich solution package is used, it must generally be expected that the customer has requirements which are not met with the existing solution, so that the package must be enhanced or changed.

In general a solution delivery project can be broken into these phases, which are quite generic:

- **Analysis:** determine and understand all requirements in sufficient detail to develop a general description of the solution and plan the work to build it. The amount of work in the subsequent phases will depend heavily on the scope of the services that must be activated, the number of activation targets and complexity of the interfaces to control them, as well as the complexity of other process and integration requirements.

  If a solution package or another existing solution is used as starting point, the analysis work will take the form of gap analysis, where the required capabilities are compared to what is already supported.

- **Design:** drawing board work to identify and determine full details of all components to be built, so that they can be described and the ideas for building each one developed. Components of a solution will include:

  — Northbound interface to receive orders from CRM system or other system which is the source of orders.

  — Workflows to execute the processes to configure and control activation of services.

  — Activation tasks to interact with target devices and systems for activation of services.

  — Operational model for resource repository. You can build a data model of the resources needed for the solution using HP Service Activator's inventory capability. You will then need to design a way load data into the resource repository when the solution is installed and keep the repository regularly synchronized with the network; the source of the data can be an existing resource inventory system, or it can be the network devices directly. Alternatively, if there is a requriement to interface directly with an existing external inventory system and you are building a solution from scratch, then you can build such an interface using an API supported by that system.

  — Data model for resource and service data repositories.

- **Implementation and testing:** use HP Service Activator tools to build all components according to design; test them as units and as a complete solution. In this phase the interworking of the components with external systems and devices must also be tested. Also

latest in this phase documentation and plans for use of the solution must be prepared along with training of the user's of the solution.

- Install the solution in the production environment, perform appropriate tests to ensure it works properly, train the users, commission and hand over for production use.

Analysis

The following information must be collected, clarified, documented and understood as part of the analysis of an activation solution:

- Full definitions of the services to be activated including the underlying technology, the packaging and organization of the services, the detailed features, and the intended evolution of the services.
- The tasks involved in activating services. Understand all activation targets that are involved in each service, the protocols for interacting with them, and the command-response or message exchange interactions that must take place in the activation processes.

  In order to fully understand activation tasks it may be necessary to study the network architecture to identify devices with different roles. This can be the background for understanding which devices need to be activated for a specific service. For example, when end customer's equipment is attached to the provider's network through some access device such as a DSLAM or access switch, where there is a port or interface dedicated to the customer then it will typically be necessary to reserve the port or interface in the resource data model and also to perform some action to enable the customer's traffic across the access port. Whether it will also be necessary to enable functions per customer on devices further towards the core of the network, such as aggregation devices and MPLS edge devices, will vary depending on the service and the network architecture.
- How activation processes are initiated. Normally these processes are triggered by XML messages received via a northbound interface from a an order management or CRM system. Various flavors of XML interfaces, including web services, are possible. Alternatively, in some solutions, there can be a requirement to support a user interface for entry of orders directly into HP Service Activator.
- How the solution will fit into the provider's overall OSS/BSS environment, apart from order entry. Which other systems will it interface to, what data will be exchanged? At what point in the activation processes will there be interactions with other systems?
- Requirements for integrating with an external inventory system, initially or at a later stage. The provider may enforce as a policy that all data of a given type, for example resource inventory, is maintained in a single inventory system, and that other systems, including the activation system, must retrieve and update the data by interfacing to that inventory system. Such a requirement is fundamental to the way HP Service Activator must be customized and must be made clear very early in the project.
- Requirements for maintaining resource and service data in HP Service Activator's data repositories. Such a requirement may be explicitly stated, or it may be implicit that the data will be needed by activation processes. It is generally easier to customize HP Service Activator to maintain resource and service data in its own repositories than to integrate with an external inventory system.
- A very important point to decide and understand is how data describing physical and logical resources that must be held in HP Service Activator's resource data repository shall be loaded and held synchronized with the actual state of the resources. One possible way is to include workflows that can upload device details from devices in the network. Another possible way is to refresh data from an external system that is already synchronized with the network.

- Volumes of activation orders to be processed and of data to be managed. Number of users performing web interactions with the solution and nature of interactions. This is information is needed to determine the processing capacity that is needed.
- Requirements for the time allowed to process orders. The analysis of the implication of this requirement combined with volume of orders will involve the activation targets and their ability to execute the implied interactions in a timely fashion. HP Service Activator is rarely the bottleneck in a solution.
- High availability requirement. This requirement together transaction volume may justify / necessitate deployment of the solution on a cluster of servers.
- User interface requirements, for example: entering orders, monitoring activity, browsing/editing data in HP Service Activator repositories. If several user functions shall be available, there may be requirements to use of roles and privileges to differentiate between users.

  A special aspect of user interface is localization of messages from English to local language.
- Audit trail requirement: what information is required to be stored as historical information about completed orders.

With all requirement information collected and organized it is possible to characterize the solution and to identify and estimate work items for the delivery project.

5 Solution Design

This chapter provides information that is intended to help you as system integrator in the solution design phase, after you have collected complete information in the analysis, before you embark on implementation of the components of your solution using the HP Service Activator tools. You will also need to consult the manuals for the individual components and tools, as listed at the end of chapter 1. When you consider tasks that require Java programming, you must consider in addition to the PDF-formatted manuals also the Javadoc which a range of interfaces and classes and which can be accessed when HP Service Activator has been installed by selecting (on Windows) `start -> all programs -> HP Service Activator -> Docs -> JavaDoc`. On Unix systems the location is `/opt/OV/ServiceActivator/docs/javadoc`. An example page is shown in Figure 5-1.

We explain here some best practices, discuss some possible ways to solve common problems, and discuss some considerations that will have an impact on how the operating solution will be experienced by operators.

Solution Labelling

In order to be able to manage and separate different solutions, which may also be different parts of a large solution, it is strongly recommended that you use the source file organization supported by the Deployment Manager for all the source files of your solution(s) and that you use a well-chosen short string to uniquely label all the parts of the solution, as follows:

- As the name space part of the name of plug-ins and compound tasks.
- As the value of the <Solution> element in definitions of resource beans and inventory trees (required for deployment).
- As prefix for database table names for resource beans.
- As the solution name in the workflow settings of each workflow and also as prefix for the name of each workflow.

Plug-Ins

Customized Use of Generic Plug-ins

As described in chapter 2 if your activation target is controlled by a command line dialog you will not need to construct a new plug-in. Instead you will customize control document templates to be used with the generic CLI plug-in. Similarly if your activation target supports HTTP and it is convenient to prepare template files for each message you will need to send, then you can use the generic HTTP plug-in. In both of these cases you will need to read the documentation for the generic plug-in you are using. It will be found as Javadoc.

A question that will come up for command-line interfaces is how many commands to include in a single command document template, so that it is executed as an atomic task. Since there may be significant overhead in establishing a session and committing changes on the target device you should strive to accomplish the complete activation of all aspects of a service within a single

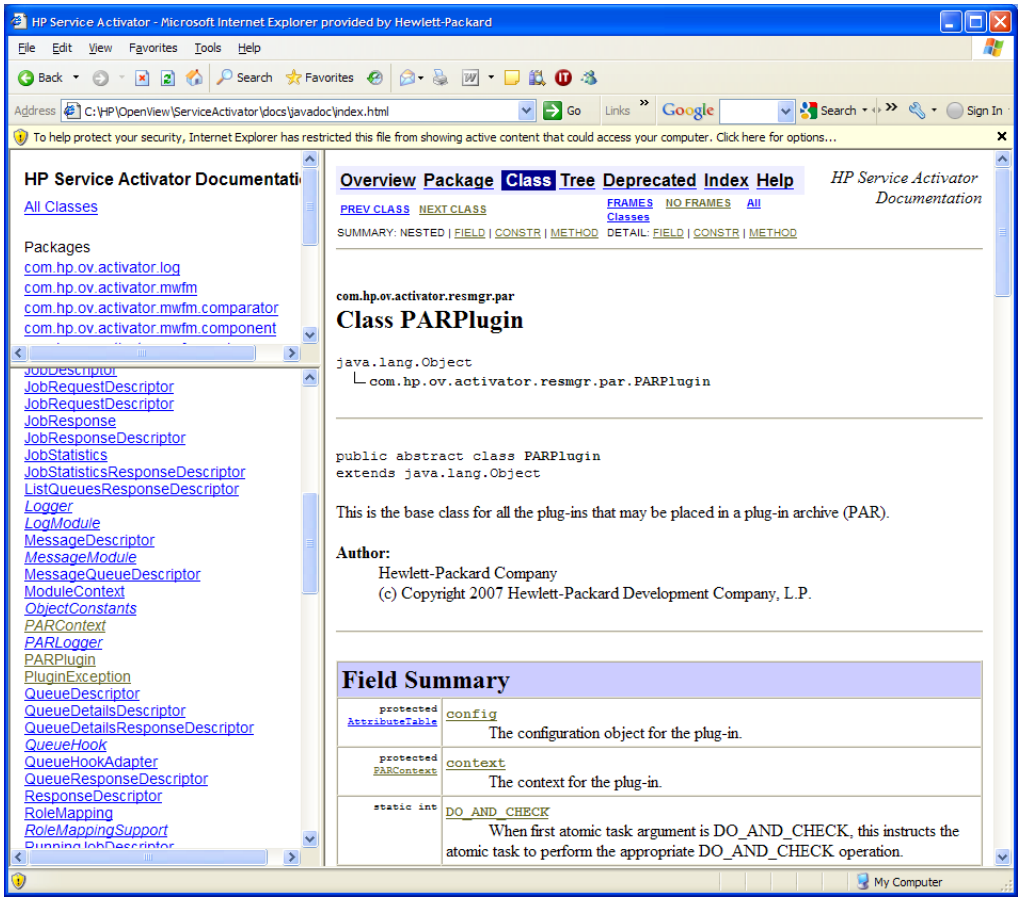
atomic task. Combinations of multiple smaller files can be accomplished with include-files and the XSL call-template construct. The result may well be large control documents containing many commands.

Creating New Plug-ins

When the generic plug-ins are not sufficient, you will need to create your own plug-in(s). Then refer to *HP Service Activator, Developing Plug-Ins and Compound Tasks* which gives you the details about plug-in programming using the Service Builder. Some general points relevant for plug-in design are briefly stated here:

All plug-ins extend a base class called PARPlugin. For detailed information, refer to the Javadoc (see Figure 5-1). From class PARPlugin all plug-ins have available a context field with a set of callable methods.

Figure 5-1 Javadoc for Class PARPlugin



In principle every atomic task must be implemented for both the do and undo modes. However, if you don't use compound tasks, the undo mode will never be used, so you can skip it. Even if compound tasks are used, this is also true for an atomic task that will not be used in any compound task or only be in the last sequential position, because then the atomic task will never be called in the undo mode.

As you design the implementation of undo mode you may need information about the state of activation targets before the do mode of the atomic task was executed, so that you can restore it. Such information may have been passed as one or more parameters to the atomic task, or you may have retrieved it from the activation target before its state is changed (in do mode). You can use

the methods of the TransactionStateSaver (or AtomicTaskStateSaver) interface which are inherited by the context to save the information for later retrieval, should the atomic task be called in the undo mode as part of unrolling the transaction. However, don't overdo it; it may not always be meaningful to attempt to do a perfect job in the undo mode: in a task intended to destroy a service a best effort approach may be the right one. When a service can only be partially destroyed, your best choice may be to design the destroy method to be idempotent, i.e. record the partially destroyed state and finish the job when the destroy method is called a second time.

If you write significant amounts of Java code, note that you can conveniently use the Service Builder in combination with a more powerful Java development environment, for example Eclipse.

Target Locking

A question you will confront when you create a new plug-in is target locking. Some targets will not be able to engage in multiple sessions concurrently, and even if the target does not have a problem, you may want to eliminate concurrency at the session level. The resource manager supports locking of a target. To determine when two calls to any atomic tasks in the plug-in address the same target, the resource manager looks at the first arguments (param0, param1, ...). You should think about this aspect when you define the arguments for the atomic tasks in your plug-in. You must define for the entire plug-in (not per atomic task) how many parameters to use for logging, with a minimum of one, and these parameters should then have the same meaning for all atomic tasks in the plug-in.

As an example, the first parameter may be the IP address/hostname of an element manager and the second parameter may identify a device in the communication with the element manager. If you will accept multiple concurrent sessions to the element manager, but you want to prevent two workflow jobs from configuring the same device concurrently, the number of locking arguments for the plug-in shall be two. It would then be natural to use the third argument to identify the function to perform and subsequent arguments to be function-specific.

As an alternative to locking on a set of argument values when one atomic task is actively using them, you can set a limit higher than one on the number of concurrent activations that you will allow.

Plug-ins for Web Services and Corba Interfaces

A frequently occurring case is an activation target which exposes a web service NBI described in WSDL form. The recommended approach to build a plug-in for such a target is to use the JBossWS utility WSConsume which is available as part of the HP Service Activator installation to generate from the WDSL a Java class where the web service methods are callable as stubs and then incorporate the generated class into the plug-in as an auxiliary Java class. The simplest approach to complete the plug-in is then to wrap each web service method as an atomic task method, but you may also want to call multiple methods, possibly with looping and branching, within a single atomic task.

A very similar process will apply to a Corba interface defined in the form of IDL. You can generate a class with stubs that can call the Corba methods and then wrap them as atomic task methods. Use this link for further information on Java IDL technology: http://java.sun.com/j2se/1.5.0/docs/guide/idl/index.html.

User Interface and Roles

The system integrator of an activation solution must understand the user community, how it can be divided into users with different roles, and the requirements each role has for viewing data, monitoring activity and interacting with processes. For more information on roles and what you can restrict and organize by means of them, see chapter 7.

HP Service Activator's main user interface window has functions to display information about ongoing activity as well as messages and audit trails produced by the workflow jobs and user

activity in the past. Such information is shown in the various windows available in the Work Area menu. As system integrator you should be familiar with how these windows work, and you should plan how you intend your users will use them. Such a plan will guide you to decide the details of the information which you can control by customization. To become familiar with the user interface, read the pertinent chapters in "*HP Service Activator, User's and Administrator's Guide*" and work the UI on a running system.

You can customize the items that appear in the Work Area and Tools parts of the left hand menu of the main UI window; see "*HP Service Activator, User's and Administrator's Guide*" for more information.

Four common case-packet variables are present in all workflows intended to hold

1) an identifier of the order being processed (WORKFLOW_ORDER_ID),

2) an identifier of the service being activated (SERVICE_ID),

3) the type of the workflow (WORKFLOW_TYPE), and

4) the state of the workflow (WORKFLOW_STATE).

The values of these variables can be shown in several of the lists (jobs, messages, audit records, including the service order view) available on the user interface. They can also be used to search for jobs and other related items. You should use them properly in your workflows to ensure that meaningful information is shown in these UI views. Use the type to classify your workflows, first of all by solution, and within each solution as you see fit. When a workflow passes through several phases the state can be used to indicate how far along a job is, particular when it is waiting for something time consuming such as operator input, an activation task, or other external interaction. The values you assign to these variables can be chosen freely.

General recommendations are:

- Use messages shown in the Message window (generated by PutMessage nodes in workflows) for debugging during development, and when the solution is in operation only for well thought out diagnostic information that will make the user aware of problems that have occurred and point to causes that the user can repair.
- Use audit records generated explicitly using the Audit node in workflows to create an accurate audit trail. In addition to a number of standard fields of audit records you can add any needed information from case-packet variables, to properly document what has been accomplished by your workflow. Audit records should always be written at the end of a workflow.

All strings shown on the user interface are collected in resource bundles to allow localization. See Appendix A for more information.

UI Integration with Other Applications

Part of the integration of HP Service Activator with other HP NGOSS products such as NNMi and NA is GUI integration involving cross launching from one the GUI of one application to that of another.

One aspect of GUI interworking is the use of HP's Light Weight Single Sign On to eliminate the need for the user to log in to each application.

Another aspect is to configure one system to know the relevant URLs for contextual views of a launched application. In order to be able to set up cross launch capability from some other application to HP Service Activator views you will need to know the relevant URL formats. They are described here in the following.

The general format is

```
http://<hostname>:<port>/activator/<activator-subsystem>?<parameters>
```

where <hostname> must be an IP address or name of the servier hosting Service Activator, and <port> is the port number configured at system installation for access to HP Service Activator's web server (default is 8080).

<activator-subsystem> defines the view you want to launch. Values are:

- views as shown in Work Area jsp/<jsp-file-name>
- Service Order View jsf/sov/serviceOrderView.jsf?<parameters>
- Inventory UI OpenInventoryFG.do?<parameters>

Look in the menu configuration file *$ACTIVATOR_ETC*/config/menu.xml for <jsp-file-name> for the Work Area views.

<parameters> take the standard form for a URL: <name>=<value>, with multiple occurrences separated by &

For service order view the following parameters can be used to specify the filter and the tab to select: jobId, serviceId, orderId, type, state, resultsTab (value shall be one of: jobsTab, messagesTab, auditTab, transactionsTab). Example:

```
http://<host-name>:<port>/activator/jsf/sov/serviceOrderView.jsf?
serviceId=site001&resultsTab=auditTab
```

For inventory UI the context must be specified as a specific instance branch to be selected using the following parameters; all are mandatory except operation:

- solution name of solution the tree belongs to
- view name of tree definition
- cl =true (to indicate cross launch)
- operation to invoke an operation on the branch
- branchPath sequence of branch names from tree definition leading to the desired branch, separated by '|' characters
- pk primary key of the instance to be selected

Example (from CNRM):

```
http://<host-name>:<port>/activator/OpenInventoryFG.do?
solution=CRModel&view=Equipment&cl=true&operation=view&
branchPath=Catalog|Region|Network|PEs|PE&pk=100
```

A Note on Workflow Start Role Attribute

Most workflows do not get started from the workflows list on the user interface, but driven from an external message, for example through the socket listener, or as child jobs of other workflow jobs. To prevent users from accidentally starting such a workflow you can set the start (or default) role attribute in the workflow definition.

Viewing Jobs During Activation

Workflow jobs engaged in activation transactions (while executing an Activate node) will be visible on the activation queue in the Active Jobs view by users who have the 'internal' role. This role is normally not assigned to human users, but you can modify this behavior by setting the role attribute on the Activate node (using the Workflow Designer). The activation queue tab will then be visible to users with the specified role, and they will be able to select 'View Activation' from the right-click menu to launch the Transaction Details window. In order to allow the activation module to synchronize with the job when an activation transaction has finished, you must also assign the role you specify for the Activate node to the system user.

You will not want users to be able to actually interact with a job during an activation transaction (that would allow the workflow job to continue as if the activation had completed). It can be prevented by configuring the user interface to hide the 'Interact with Job' item from the right-click menu of jobs on the activation queue.

Encrypted Passwords

One of the data items you may need to model for an activation target is a password that will be used to authenticate HP Service Activator when it establishes a session with the target. You will want to avoid to store the password in clear text in resource inventory, log files, etc., so you will need to be able to encrypt the password after it is entered, for example on the inventory UI, using an encryption scheme that allows you to decrypt it again before transmission to the target, where it may be encrypted again using the key appropriate for that particular connection.

You can configure fields of resource beans for the inventory to be treated as passwords that must be encrypted upon entry. There are a number of ways to decrypt passwords (encrypted strings) when needed: by a workflow node (Decrypt), by a context method for use in workflow manager modules, by a context method for use in plug-ins, by an attribute in a control document for the CLI plug-in.

Data Models

To create a data model for resource and service inventory you must apply entity relationship analysis to the data you need to represent in order to determine entity classes and the foreign keys you need to represent relationships.

The inventory subsystem, i.e. the Inventory Builder and the beans and JSPs it generates, support many features that you should be aware of:

- a range of field data classes, including passwords as described above, with several control attributes
- inheritance, which lets you define resource superclasses and then subclasses by adding fields to a superclass
- flexible definition of keys that can be used to retrieve desired entity instances by resulting findBy methods, including foreign keys which represent relationships between different entity classes
- use of entity relationships to define keys comprising fields from inrter-related entity classes (database joins)

For inventory trees and forms presented on the inventory there is also a rich set of features to control the structure and appearance of the trees and the operations that can be performed by users with different roles.

Consult the inventory manual, *HP Service Activator, Inventory Subsystem*, for detailed information.

External Inventory Integration

You may need to integrate your solution with a preexisting inventory system.

If the inventory is held as Oracle data with known table definitions, and the HP Service Activator system can access the data at the SQL level, you can integrate by creating resource definitions that match the existing data, then use the resulting beans in your workflows to access the external inventory.

If the inventory is accessible only through a higher level API, an approach you can take is to design and build a number of workflow nodes to access the data in a way that will be convenient to

use from your workflows: create, query, update, etc., as needed. Such nodes will typically use a tailor-made workflow module to maintain a session with the external inventory.

A set of nodes providing a liaison to HP's uCMDB product are available in the built-in library.

Workflow Processes

Controller Workflow Pattern

The section "A Typical Workflow" in chapter 1 and the example in chapter 2 give you some general ideas of what you can do with workflows. The pattern that you see in the Intro example with a controller workflow that handles the northbound communication and separate workflows to handle each type of request (in this case, create and delete service) is often useful. This pattern is a refinement of the single process that was described as "a typical workflow".

Workflow Structure: Before - Activate - After

When you have complex activations involving several targets it is generally recommended to use compound tasks in order to simplify the error recovery logic in the workflow. A workflow with an activation task should have a clear before-and-after structure: before the activation only preparations, like retrieval, calculation and selection of parameters for the activation, possibly including resource reservations take place; after the activation come service inventory updates, output of informative messages, writing of audit records. If the activation fails, it should fail cleanly, and the updates that need to be undone or changed should be minimized.

Business Processes

Although workflows are primarily intended to implement activations, it is also possible using interactions with operators and possibly external systems to implement business processes of some complexity and duration.

Starting Workflow from Inventory UI

Although it is generally expected that most workflow processes are initiated by receipt of request messages from a northbound system, it is also possible to start workflow jobs from the Inventory UI. You can use this feature for example if you have some processes which are not service oriented requests, for example device configuration or inventory maintenance. You can also provide an emergency route to initiate service activations, for situations where the northbound system is out of operation. It is possible to initialize case-packet variables of the workflow job that is started with values that are sourced from inventory or entered by the user.

Workflow Job Persistence

In a workflow definition you determine for each node whether the state of a job executing the workflow shall be stored ("persisted") in the database after execution of the node. Should the workflow manager be restarted while the workflow is running, the job will be restarted at the last point of persistence. It is not necessary to persist after every single node; there is a performance cost associated with storing the data. Nodes which only change state internally in the workflow manager generally do not require persistence; they can be reexecuted after a restart, which will in reality happen very rarely, or never.

Business Calendars

A dedicated set of workflow nodes - `IsTimeIncluded`, `GetNextIncludedTime`, `GetBusinessHoursAfterDuration`, `GetTimeRangesOfBusinessDay`, `GetCalendarTimezone` - allows workflows to consult a business calendar defined for a solution with weekly working days and hours per day as well as annual holidays to avoid undesired activity

outside of business hours. Business calendars are created and edited from the user interface. It is configurable whether the editor shows time of day in 12 hour (am/pm) or 24 hour format.

Considerations for Custom Workflow Nodes

You can extend the library of built-in workflow nodes by implementing custom nodes. The same is true for workflow manager modules. Information about these topics is found in *HP Service Activator, Workflows and the Workflow Manager*.

There is a golden rule to observe: a workflow node must never occupy a workflow thread when it is waiting for an external event.

Northbound Interface

In most cases an HP Service Activator solution will need to be integrated northbound with an Order Management or CRM system which will be the originator of requests to create, modify or terminate services. The activation system will then respond to each request with one or more messages to acknowledge receipt and to indicate progress and completion of the requested process.

The general assumption is that the activation system is implemented as a collection of workflows that can accomplish a set of tasks which the originator or client system will request. In other words, each request from the client to perform a task can be translated to: run a specific workflow with some case-packet variables initialized with parameter values supplied by the client as values of elements in the message.

Two approaches to implementing the northbound integration were introduced in the section "Interfaces for Integration" in chapter 2:

- using listener and sender modules of the workflow manager
- using web services implemented with servlets

The choice of which approach to take will depend on circumstances and specific requirements.

Using Listener and Sender Modules

Two pairs of listener and sender modules are available: the socket module pair and the JMS module pair. The interworking between modules and workflows follows the same pattern in both cases. An instance of the listener module is configured to listen at an access point of the underlying communication service for an incoming message and start a named workflow each time a message arrives. The message is stored in a temporary database entry, and access to it is provided to the workflow job through one initialized case-packet variable. In both cases, socket and JMS, a message is sent back to the client by using the SendMessage workflow node with two parameters: one identifying the sender module instance, another holding or identifying the message.

In the case of the socket listener and sender modules the underlying communication service is TCP. The access points are ports, and the messages are exchanged as raw TCP messages. In the JMS case the underlying communication service is a MOM (message-oriented middleware) supporting a JMS interface. The access point are JMS destinations, and messages are exchanged as MOM/JMS messages. See Appendix C for more information on using JMS.

With the listener/sender module approach parsing of incoming request messages is left to the workflow job that is started by the listener module. It is recommended to use a controller workflow to handle the parsing (at least of the general message fields such as order identifier, customer identifier, service identifier and request type which are not specific to the type of request) and determine the appropriate workflow to handle the specific request.

The Intro example described in chapter 3 follows this recommendation with one controller workflow and a separate workflow for each request type. The example shows how to implement parsing of request messages with the XMLMapper node and how to use templates of response

messages where values of case-packet variables are substituted for placeholders. This pattern represents a recommended best practice.

If you have multiple solutions with unrelated request message streams and formats, it is recommended to use a separate instance of the listener module for each solution, each one listening on a separate port.

Using Web Service Servlets

The second approach uses servlets deployed in JBoss, more specifically they are deployed with JBossWS in Apache Tomcat which is part of the JBoss platform that HP Service Activator runs on. The servlet for a service can be generated with the Web Service Designer tool which is provided as part of the HP Service Activator core product. This tool and its use is described in chapter 8. When the generated servlet has been deployed, a description of its interface in WSDL can be extracted from JBossWS on the running platform - see chapter 8 for more detail - and imported in a client system to support generation or configuration of the necessary interface component on the client side.

In this approach a set of web service methods are exposed, where each method will run a specific workflow. Incoming messages will be parsed by the servlet and parameters of the method are assigned as initial values to case-packet variables of the workflow job.

The method may just start the workflow job and return the job identifier to the calling web service client, or it may wait for the job to complete, and return a response message including final values of case-packet variables. In the latter case the call-return communication synchronizes the caller with the activation system. In the former case the activation job may run asynchronously with continued activity on the client side, and will typically need to return one or messages to inform the client of progress and final status of the activation job. Such messages can be implemented according to the sender module approach as described above.

Synchronous calls are not recommended when there may be many jobs with long or unpredictable duration. Servlets are not fault tolerant; the relationship between a running job and the client that requested it will be lost in the case of server host failure.

With this approach there is no obvious need for a controller workflow. Compared to the first approach the servlet takes on the role of the controller workflow.

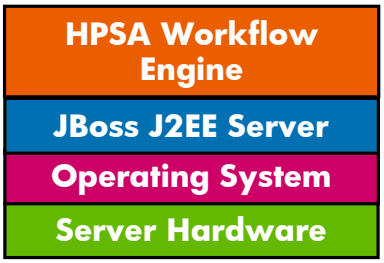
6 HP Service Activator Platform

As described in chapter 1 the core component of an HP Service Activator production system is a workflow engine. The workflow engine comprises the workflow, transaction and resource managers.

The workflow engine runs in the framework of JBoss (version 4.2.2). JBoss is a J2EE platform which can run on a range of hardware and operating system platforms. HP Service Activator is specifically supported on HP-UX on HP Itanium processors, on Sun Solaris on Sparc processors, on Red Hat Enterprise Linux on x86-64 processors, and on Windows 2008 Servers on any hardware platform. In addition to JBoss, the operating system, and the processor hardware, HP Service Activator requires access to an Oracle server via Oracle net.

The complete stack comprising the workflow engine on top of the required platform is shown in Figure 6-1.

Figure 6-1 HP Service Activator Platform Stack



Cluster Platform

As described above, HP Service Activator can be deployed on a range of different hardware, from small and inexpensive to large, very powerful processing systems. From V5, HP Service Activator can also be deployed on clusters of separate servers, interconnected within a segment of a local area network. The term *cluster node* (or just *node*) is used to designate a server in a cluster.

Cluster configurations provide three important benefits:

- Extreme scalability: the workload of the HP Service Activator solution can be distributed over all the nodes in a cluster; when a node is added, the total processing power is enhanced.
- High availability: if one of the nodes in a cluster suffers a failure and ceases to function or is taken out operation, all running workflow jobs will continute to run and are redistributed over the remaining nodes (one or more).
- Easy configuration of server nodes on standby sites for disaster recovery.

The unit of work for an HP Service Activator system is a workflow job. A cluster node is selected to run a workflow job when the job is started. The job will remain on the selected node until it completes, except when a node failure occurs. All the processing of the workflow job, including

activation transactions down to atomic tasks takes place on the same node. But when a job starts a child job, a different node may well be selected for the child job.

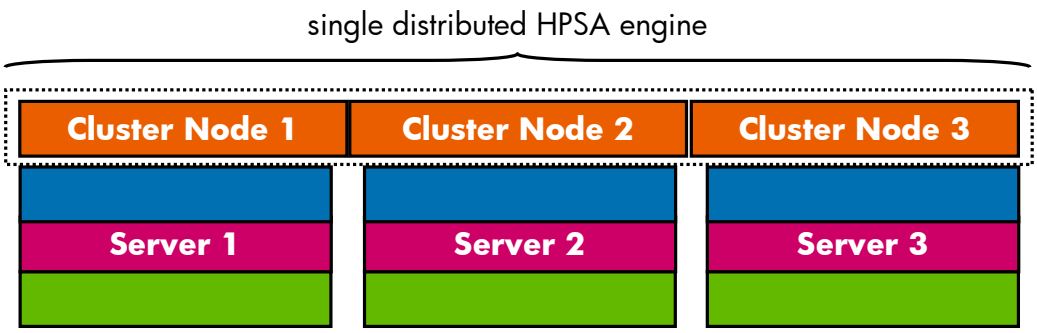
At any point in time when an HP Service Activator production system is running, there will be a number of jobs running on each of the nodes. The distribution of work will change when jobs terminate, and when new jobs are started. The system administrator can view the running jobs and their assignment to cluster nodes.

Workflow jobs are executed as a sequence of workflow nodes (note: a different meaning of the word node!) determined by the control logic of the workflow, with branching controlled by rule nodes. As the execution of each workflow node completes, the state of the workflow job may be persisted, allowing execution of the job to be restarted from the next node. After a node which has no external impact, persistence may be omitted to enhance performance. Workflow nodes which have external impact are those which perform activation tasks, update the database (inventory or audit trail), post messages to queues, send messages to other systems, or perform any other action which causes something to change outside of the workflow engine itself. The persistence property as described here allows a workflow job to be restarted, if the cluster node it runs on suffers a failure or is halted, while the job is running. Restarting can take place without any startup costs on any node in the cluster.

To allow completely symmetric assignment of workflow jobs to cluster nodes, all the information that is needed to run a job is held in database repositories that can be accessed by all the nodes. This includes the static information: workflows, compound and atomic tasks (plug-ins), inventory tree definitions, as well as the dynamic states of workflow jobs, including values of case packet variables, and of active compound transactions, including locks on activation targets.

As a result of the symmetry and the single database, a cluster of workflow engines will appear and behave like a single distributed engine as illustrated in Figure 6-2 for a cluster comprising 3 nodes. Operators and external systems can interact with a running workflow through the web service interface of any node in the cluster.

Figure 6-2 HP Service Activator Distributed Workflow Engine



Given the equal access to persisted workflows from all nodes in the cluster, it could be considered to monitor the relative loads on differents nodes and migrate jobs between nodes to balance the load. There would be a performance penalty, since a job that remains on the same node benefits from having its state cached in the workflow engine. Assuming most workflows are short lived, the benefit would not be great. In HP Service Activator V5 workflow jobs only migrate upon node failure.

In the event of a node failure, it will be detected by the other nodes in the cluster. One of them will then automatically assume the role of redistribution manager. The redistribution manager will restart all workflow jobs that were active on the failed node. For each job a new cluster node is chosen to execute it. In this way the load of the failed node will be shared over all the remaining functional nodes.

Detection of cluster node failure is done by means of keep alive timers that are monitored by keep alive modules of the workflow managers on all active nodes in the cluster. The timers are also held in the shared database.

Cluster Installation and Setup

For details on how to install and configure an HP Service Activator cluster, please refer to the Installation Guide. Only the main principles are mentioned here as part of the explanation of the clustering concept:

- HP Service Activator must be installed on each node in the cluster. This will also install JBoss on each node. HP Service Activator clustering does not build on JBoss clustering. The JBoss servers are independent and not aware of each other.
- When HP Service Activator is installed on the first node, the database tables used by the workflow manager and other parts of the HP Service Activator platform (as opposed to the tables belonging to the customized data model of a solution to be added on top) shall be created. When HP Service Activator is installed on subsequent nodes, the same database user shall be specified, but the tables shall not be created again. An additional entry (row) will then be created in the table (CLUSTERNODELIST) that describes the nodes of the cluster.
- HP Service Activator must be configured on each of the nodes in the cluster. For example, the workflow manager is configured in file mwfm.xml. The configuration shall normally be identical on all nodes of the cluster.
- Nodes can be installed on different sites. Each site is either a primary site or a standby site for disaster recovery. The name and type of the site must be given for each node where HP Service Activator is installed.
- Except on a Windows server a virtual IP address can be defined for the server. The virtual address must be associated with an interface on the server. The virtual address is useful in the internal load balancing scenario described below.

Workflow Load Distribution

The important control decision that determines the distribution of load across the cluster is the selection of the node where each new workflow job is started.

Except for child workflows the initiative to start a workflow activity always originates outside of HP Service Activator, typically in a CRM or Order Management system. The external component which makes requests for workflow jobs to be run can also be a catalog-driven workflow controller delivered by HP as part of a solution. Service orders may also be entered into an order portal GUI, which can translate orders directly to requests to run workflows.

Once a workflow job has been started in response to an external request, it may spawn child jobs. The complexity of the workflows that are started by an external initiator will vary depending on the solution. In complex cases, the initial workflow will spawn many additional jobs.

Different mechanisms can be implemented to interface the external control component to HP Service Activator's workflow manager, but in the end the request to run a workflow job is always made by a call on the API of the workflow manager: startJob (asynchronous mode) or startAndWaitForJob (synchronous mode).

NOTE The case when a user starts a workflow from the main UI or from the inventory UI is not special. The request is passed via the JBoss web server through a call on the workflow manager's API.

The actual API call may be implemented in different ways. It can be made directly by an external controlling component using either the RMI or SOAP version of the API. Alternatively it can be made by a workflow manager module, such as the socket listener, or by a web service NBI

dedicated to an activation solution, i.e. a specific set of workflows exposed as web service methods.

Regardless of these technical variants, the more significant difference has to do with the selection of the cluster node where the job will run, as this is the decision that will affect load balancing in the cluster. The selection is made by a module of the workflow manager which receives the API call, known as the *distribution module*. Three different distribution modules implementing different selection algorithms (see below, Internal Load Balancing) are available in the HP Service Activator V5 kit; the workflow manager must be configured to know which distribution module to use. When the distribution module has selected the node to process a workflow job, it transfers the workflow job to that node.

The workflow manager can run in "stand-alone mode" (as all versions before V5) with no active distribution module.

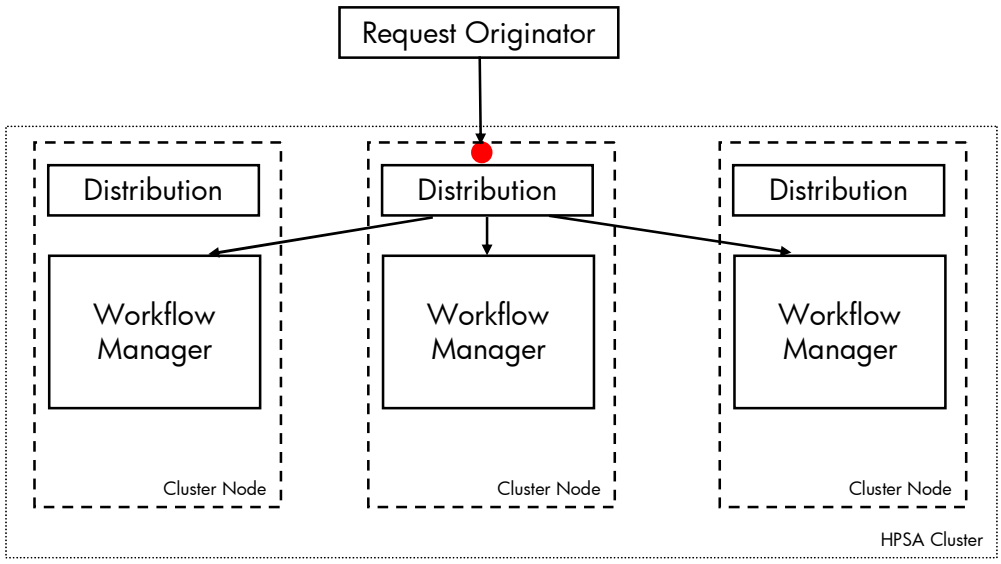
As for other workflow manager modules it is possible to extend HP Service Activator with additional node selection algorithms by implementing new distribution modules.

All distribution modules allow the node selection algorithm to be bypassed for workflows that are started by a call on the external API.

Internal Load Balancing, Virtual IP Address

Internal load balancing is the intended normal distribution model, where HP Service Activator itself takes care of load balancing. In this mode it does not matter which cluster node receives the request to start a workflow job. The originator of the request therefore does not require any load balancing capability. It is enough to know a single node which can receive all the calls and distribute them across the cluster, as illustrated in Figure 6-3.

Figure 6-3 Internal Load Balancing



The HP Service Activator server address known by the request originator should be the virtual IP address of the node designated to receive the requests. In case this node suffers a failure its virtual IP address will migrate ("fail over") to one of the remaining servers. This may take up to a minute. The request originator may then simply continue to forward requests to the same address, and they will be transparently received and processing initiated by the failover server.

By default the virtual IP address will automatically move back to the node it belongs when that node again becomes available. You can configure the keep alive workflow manager module to

overrule this behaviour, if you prefer to manually decide from the UI when the virtual IP address shall move back.

Four distribution modules, supporting different algorithms, are available as part of the HP Service Activator standard product distribution. The workflow manager can be configured to use any one of these. The four algorithms are:

- Plain round robin.
- Weighted round robin, where each cluster node can be assigned a different weight according to its processing power.
- Selection of the node which has the shortest queue of workflow jobs waiting for a processing thread in the workflow manager (a measure of the load at the time when the decision is made).
- Selection is based on the value of a pre-initialized case-packet variable of the job to be started.

When a workflow job spawns a child job, the load balancing algorithm will again be applied, so a family of jobs with a common ancestor are not tied to the same cluster node.

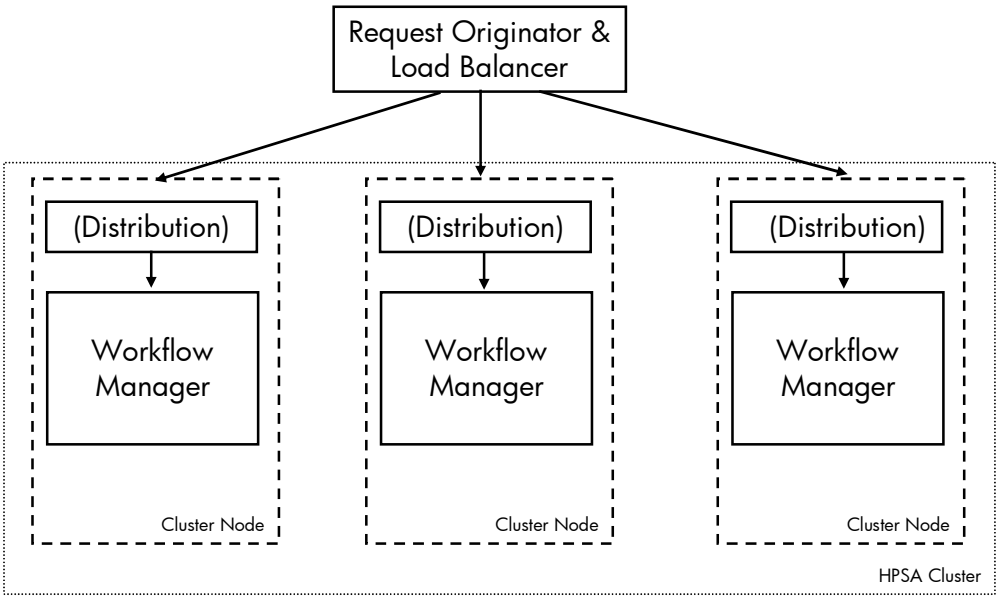
External Load Balancing

In the external load balancing model, the cluster node that receives the external API call to start a workflow job also runs the job. To achieve this behavior the active distribution module must be configured to run externally requested workflows locally, i.e. the node selection algorithm is not used. This means the external originator of the requests to run workflows will determine the distribution of processing across the cluster.

This mode of operation is supported to allow the use of request originating systems which already support a desired load balancing algorithm. Such systems will need to know all the cluster nodes and to maintain information about them, which requests have been dispatched to each one, etc., in order to make load balancing decisions.

When a job is started on a cluster node where the distribution function is disabled, any child jobs will run on the same cluster node. This is a logical consequence of the balancing logic being implemented externally. However, when jobs are redistributed after a node failure, they will still be distributed as opposed to all being executed by the node which acts as redistribution manager.

Figure 6-4 External Load Balancing



Standby Sites for Disaster Recovery

For the case of mission critical solutions where servers must be in standby mode at a secondary disaster protection site, HP Service Activator cluster configuration allows cluster nodes to be associated with sites, and sites can be categorized as primary or standby. The category is actually a modifiable state of a site. Only the nodes at the primary site need to be running in normal operation, but the standby site(s) is also included in the installed and configured system and is visible from the user interface.

The Oracle database server that HP Service Activator uses may also be configured with primary and standby sites and use Oracle Data Guard to automatically maintain an up-to-date copy of the database at the standby site(s). Alternatively independent Oracler servers can be used at the sites and backups of the database can be regularly transferred to the standby site(s).

In the event of a disaster, i.e. the servers at the primary site become unavailable, the server(s) at the disaster site must be started, the system administrator must connect to it (one of them) and change the state of the site from standby to primary. Northbound systems and users must then make sure to access the (a) new primary server. A configuration change may be needed, because the secondary site may have different host names and IP addresses from the original primary site.

The procedure for switchback to the original primary site must be similar. The secondary site must be deactivated, the original primary site restarted, the states of the sites must be swapped; this can be done from the system administrator user interface, and then the system will again be ready for normal operation.

Managing an HP Service Activator Cluster

User Interface Functions for Cluster Nodes

HP Service Activator's system administrator interface (see *HP Service Activator, User's and Administrator's Guide*) has features for managing a cluster:

- The operator can view how jobs currently running are distributed across the nodes of the cluster.
- The operator can set the state of a site to primary or standby.
- Statistics reports can be produced about the load of the nodes over a specified period of time.
- Nodes can be taken in and out of operation.

The last feature is known on the interface as locking and unlocking of a node. When a node is locked, it will no longer be a candidate to run new workflow jobs. The load on the node will then drain as current jobs run to completion. When there is no load left the node can be taken of operation for hardware or software maintenance.

Cluster nodes can also be suspended: all jobs are immediately suspended with saved state so they can be resumed later. Suspension is primarily intended to be applied to a complete cluster to allow backup of a frozen database. No use case is intended for suspending a single node.

Synchronizing Time on Cluster Nodes

In order to allow proper cooperation between the workflow engines in a cluster, the date and time on all the servers must be kept synchronized using a tool such as NTP (Network Time Protocol).

7 Roles, Privileges and Authentication

Some Service Activator solutions are black boxes with very few users. Others may have a larger user community. The inventory subsystem, in particular, may allow users to perform operations on many different types of entities. It is possible, therefore, to assign privileges to perform operations from the user interface with any desired granularity.

Privileges in Service Activator are granted to roles. When a user logs in to Service Activator, an authentication takes place based on user name and password, and a list of roles is established that the user will have during the session. The user is then permitted to perform any operation for which the privilege has been granted to one of the roles in the list.

NOTE Authentication must be configured for the workflow manager, otherwise it will not occur, and users will not get any roles. It is not enabled at installation time.

The user interface operations that may require privileges fall in four areas:

- Functions available as menu items in HP Service Activator's main UI window. The menu falls in two parts, the Work Area and the Self Management menu. The Self Management menu as a whole is only available to users who have the - predefined - system administrator role, normally named 'admin'. For the work area a required role can be assigned to each menu item.
- Workflow related operations, also performed from the main UI window: starting and stopping workflow jobs, inspecting the state of a workflow job, interacting with a workflow job, viewing messages posted on queues by workflow jobs.
- Inventory operations performed from the inventory UI window: view a tree; expand a branch in a tree to see more branches; create, view, edit or delete a resource instance.
- Deploying plug-ins using the Service Builder tools. In this case the user does not log in to Service Activator's web-based UI, but the Service Builder invokes methods to deploy plug-ins on the resource manager, and these methods may require authorization using the user's identity as established by the operating system.

As customizer of a Service Activator solution you must define the roles that shall exist for the system, and you must assign to roles the privileges to perform specific operations. Typically you will then leave it to the system administrator to define users and assign roles to them. You must decide which operations shall require specific privileges. If it is not necessary to distinguish between different categories of users, you can simplify your task by leaving many operations available to any authenticated user. You generally do this by omitting to configure a privilege for each operation (for example to interact with a workflow or perform an operation on the inventory UI). You must also decide the granularity of the roles you define. If you make few large roles, each with wide privileges, you will simplify the task of the system administrator. If you make many small roles, each with limited privileges, you enable the system administrator to assign privileges to user groups with fine granularity.

The first section in this chapter introduces the system user and the predefined system administrator role.

The second section gives an overview, with references to additional information, of how the privileges to perform the operations in the four mentioned areas are assigned to roles. This is done in different ways, using different tools, for each of the four areas.

The third section gives an overview of the different modules you can choose from to authenticate users and establish their roles at log-in, and how to configure these modules.

The fourth section describes how users can be grouped in teams, and finally there is a section that describes HP Service Activator's support for HP's Light-Weight Single Sign On framework.

Management of users, roles and privileges is done by the system administrator using the User Management view available from the Self Management menu area of HP Service Activator's main UI, as described in *HP Service Activator, User's and Administrator's Guide*.

System User and Predefined Roles

There are two predefined roles, 'admin' and 'internal'.

The 'admin' role, as described above, is the system administrator role with privileges to perform system administrator functions. It is possible, but not recommended to change the name of the role by editing the configuration file for the workflow manager, `mwfm.xml`, see "Setting the Workflow Manager Parameters" in *HP Service Activator, Workflows and the Workflow Manager*.

The 'internal' role exists for technical implementation reasons. This role has the privilege to interwork with workflow jobs which are sleeping or waiting for activations or other software internal interactions. It is not intended to be assigned to human users.

One special user, the *system user*, is created with username and password during installation (system configuration) of HP Service Activator. It exists primarily for technical implementation reasons and has all roles, including the predefined ones 'admin' and 'internal'. Software components (for example workflow manager modules) which need to perform 'internal' interactions with workflow jobs will take on the identity of the system user.

It is not possible from the user interface to change the name or password of the system user or to revoke any roles from it.

The system user is not intended to be used by a system administrator during normal operation, but must be used to create at least the first ordinary user once authentication has been enabled.

NOTE If a non-native authenticator module is used (see "Authentication and Assigning Roles to Users" below), the predefined roles and the system user are not automatically created during installation. They must then be explicitly created in the appropriate environment (operating system) before authentication is configured.

Assigning Privileges to Roles

The privileges to perform operations related to workflows and inventory can be defined and assigned to roles with a fine granularity as part of the customization of workflows and inventory.

Roles and also their relationships to inventory UI privileges can be described in an XML-formatted file. The schema for the document is found in file `$ACTIVATOR_ETC/config/ummData.xsd`. If you have defined the roles and relationships in the User Management view so that they are recorded in Service Activator's static repository in the Oracle database, you can use the UMMData script to export them to a file and include it in a solution to be deployed with the Deployment Manager. Refer to the manual for the Deployment Manager for information on how to include the roles file in a deployable solution.

For each of the four areas of potentially privileged operations, the information on how to define the operations with privileges and assign them to roles is found in different manuals, as outlined here:

User Interface

Privileges to view and select menu items in the work area of HP Service Activator's main UI are assigned directly to roles. Configuration of the UI menu is described in *HP Service Activator, User's and Administrator's Guide*.

Workflows

For operations related to workflows, privileges are also assigned directly to roles, see "Setting Roles" in *HP Service Activator, Workflows and the Workflow Manager*. In the same manual, specifically for information about message and request queues, see "Queues", and for information about how to make these queues permanent using the <Permanent-Queue> tag, see "Setting the Workflow Manager Parameters".

Inventory

With respect to inventory operations, see "Inventory Tree Definitions" in *HP Service Activator, Inventory Subsystem*, for information about defining the privileges, known as operation types, branch types and tree definitions. The assignment of these privileges to roles is done as part of User Management in HP Service Activator's main UI.

Deploying Plug-ins

For deploying plug-ins, there is only one privilege, it is preassigned to a role named "deployer". The role is only required if you have enabled authentication of plug-in deployment. For more information, see "Configuring Authentication or Authorization" in *HP Service Activator, Developing Plug-Ins and Compound Tasks*.

Authentication and Assigning Roles to Users

User authentication in Service Activator is the process of validating that a supplied (user name, password) pair is valid, determining the identity of the user, and retrieving the list of the user's roles. It is done by a module of the workflow manager, known as an *authenticator module*.

On a virgin system, after installation of Service Activator, authentication is disabled. You enable it, like other functions performed by workflow manager modules, by editing a specification of the module you want to use - its name, Java class, and configuration parameters, into the configuration file for the workflow manager, `mwfm.xml`.

You have four authenticator modules to choose from, HP Service Activator's native recommended module, the `DatabaseAdvancedAuthModule`, and one for each of the four supported operating systems, Windows, HP-UX, Solaris and Red Hat Linux. For special requirements it is possible to build additional authenticator modules.

With the native authenticator module the user information is held in Service Activator's own database, and all user administration functions are done through the User Management view of Service Activator's main UI.

The other authenticator modules delegate authentication to the operating system, validating Service Activator user credentials against the same database that the operating system uses for its own users. These modules then retrieve all user groups that the validated user belongs to and maps them to roles using the role mapping file. By default, the role mapping file is absent, and each group name is taken directly as a role name.

You will understand how to construct role mappings from this example, which shows the contents of a role mapping file that maps the two groups `activ_users` and `activ_oper` to the same Service Activator role, operator, and a third group `root` to the admin role.

```
<?xml version="1.0" encoding="utf-8" ?>
<!DOCTYPE RoleMappings SYSTEM "role_mappings.dtd">
<RoleMappings>
```

```
    <Role>
       <Name>operator</Name>
       <Mapping>activ_users</Mapping>
       <Mapping>activ_oper</Mapping>
    </Role>
    <Role>
       <Name>admin</Name>
       <Mapping>root</Mapping>
    </Role>
 </RoleMappings>
```

The role mapping file must be named and placed as: `$ACTIVATOR_ETC/config/role_mappings.xml`. After you create or change it, restart Service Activator to allow it to take effect.

NOTE If you write your own authenticator module - see "Writing New Authenticator Module" in *HP Service Activator, Workflows and the Workflow Manager* - you must consider if your module needs to support role mapping.

When you use one of the operating system based authenticator modules, you manage Service Activator users by the same tools that you use to manage users of the operating system, including the creation of dedicated groups to implement the roles you have defined for Service Activator.

Organizing Users in Teams

For use on systems with many users there is a concept of teams. You can divide the user community in teams and appoint one or more administrators for each team. A team administrator can access the user management functions in the Self Management menu, but is restricted to manage the members of his/her own team.

Each user will belong to exactly one team. Roles must then be assigned to teams, and the team roles limit what roles can be assigned to the users in the team. Initially there is a default team which has all roles.

NOTE The team feature is only available when configured on the native authentication module.

Light Weight Single Sign On

LWSSO is a framework that a number of HP NGOSS (and other) products adhere to. Within this framework a user can log in to establish a session with one product and may then execute a cross launch action that will bring up a GUI of another product - in a separate window or in a frame of the window from which the action was launched - without having to log in to the second product. LWSSO is based on tranferring encrypted cookies between the product servers, via the user's browser client. The cookie will contain the user's identity (user name), but not the password. The password is only authenticated when the initial session is established, and the authentication mechanisms (and users' passwords) may be different on the various products that cooperate within the framework.

HP Service Activator supports LWSSO. It allows cross launch to and from other products which also support LWSSO. NNMi and UCMDB fall in this category, NA does not. When launching an NA window from Service Activator, the user will be confronted with NA's log in dialog, but only once in a session.

The use of LWSSO must be configured for Service Activator at installation time. At this time a number of parameters concerning the encryption method, etc., must be entered. See the Installation Guide for details.

Some special considerations for the use of LWSSO must be noted: Functions which require access to the user's password are not possible when Service Activator has been cross launched using

LWSSO: user management, use of stored filters and searches. Time skew between cooperating servers must not exceed 15 minutes.

8 Common Network Resource Model

The Common Network Resource Model (CNRM) is a data model appropriate for a class of commonly managed new generation networks based on IP, Ethernet and MPLS technology. The model is included in the HP Service Activator core product packaged as a deployable solution which can be used as a basis for a customized solution or in fact by multiple solutions which will manage different services over the same network. The CNRM consists of resource definitions, tree definitions for the inventory user interface and functions implemented as workflows which can be launched from the inventory UI.

The model matches well the types of networks which are typically managed with the HP BTO NNM and NA products. It is eminently suitable for solutions where HP Service Activator is integrated with these products (as discussed in chapters 10 and 11). The CNRM can be populated by uploading data representing the network and its equipment. Data upload can achieved through integration with NNMi.

NOTE The modelling of classical transmission technologies (SDH, SONET, T1) for layer 1 transport is not covered by the CNRM, but must be added to the model if needed. With respect to the MPLS core network, the model is only concerned with edge devices. Additions will be needed to manage routing of label switched paths across core (P) routers.

The CNRM was introduced in the section "Solution Data Repositories (Inventory)" (starting on p. 18). The introduction included a diagram of the network architecture that CNRM is suitable for and listed some services that such networks are used for: corporate VPN services; traffic between customer sites and (provider or third party) platforms providing a range of services (Internet access, VoIP, IPTV); interconnection of elements of provider infrastructure (mobile backhaul).

The network architecture as depicted in Figure 2-1 includes an MPLS core network and L2 (Ethernet) access networks. L2 and L3 VPN technologies are used to structure traffic across the MPLS core network. Traffic across the L2 access network normally uses provider bridging, i.e. S-VLAN tags are used to identify the traffic for specific services. PE routers can also have interfaces to legacy access networks (ATM, frame relay, SDH, TDM), but the model does not cover these networks.

Adapting the CNRM for a Solution

For some solutions, those which are not concerned with networks that resemble Figure 2-1, the CNRM will be not be useful. For those solutions a different model must be created according to the requirements.

In many cases, when the network and services of the service provider requiring an activation solution, do resemble Figure 2-1, extensions or adaptations may be needed to match precisely the provider's network architecture and the data required to manage the equipment and services involved in a solution.

An important case where the CNRM is used with some extensions is the solution package for L2 and L3 VPN management. In fact the model is a generalization of the one that was used in the VPN SP before it was included in the core product.

The CNRM is present in the core product in the form of resource and tree definition source files, so it is possible to modify the model in any way required for a solution. Even in cases where requirements dictate a significantly different model, it may be useful to take the CRM as a starting point and work with differences and modifications rather than elaborating a complete model from scratch.

For a project where the CNRM can be used with modest modifications, considerations of support and maintenance of the solution over time suggest that the solution should be constructed in such a way that such evolution of the CNRM as may occur in future product releases may be incorporated smoothly. It is recommended, therefore, to strive to maintain the resource bean definitions of the CNRM unchanged and implement any modifications in specializations of the classes of the CNRM.

When defining a specialization you can use the <ParentField> element to define for the specialized class properties to override the properties of the superclass (parent) fields, such as: name, label, list of values, visibility, modifiability and formatting.

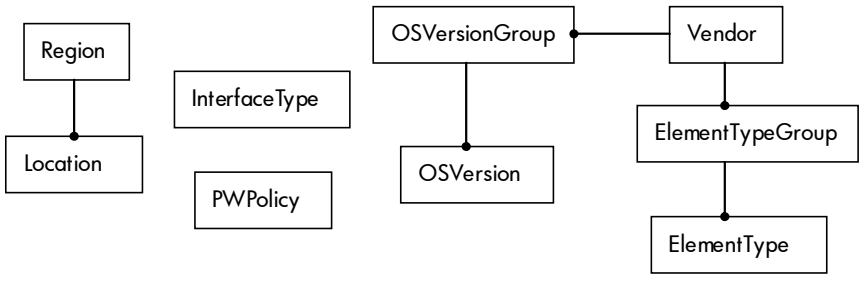
Model Configuration Data

A number of (resource bean) object class are associated with the CNRM but do not model elements of a network. These classes configure the model by defining valid values for fields of the proper network model classes. They are shown with entity relationships in Figure 8-1. Each region object defines a valid value for the region property of a network, each ElementType object defines a valid for the ElementType property of a network element, etc.

An important aspect of this is that a solution may contain mappings and templates for network elements and their interfaces which depend on interface type, OS version and element type. The configured property values will be used as indexes for lookup of the proper mapping or template to apply for each network element or interface. The groupings of OSVersions and ElementTypes make it possible to associate mappings and templates with groups rather than individual OSVersions and ElementTypes without losing accuracy of the model data.

The CR Model solution itself contain a table, RouterTemplate, which is indexed by Vendor, OSVersionGroup and ElementTypeGroup and identifies a template for the dialog to query a network element for its interfaces and a Java class to parse the response.

Figure 8-1 Model Configuration Data Classes



There are very few fields on these objects (beans): the main one for each is simply the name which defines a valid value of the corresponding property. In addition there are the foreign keys representing the entity relationships and description fields.

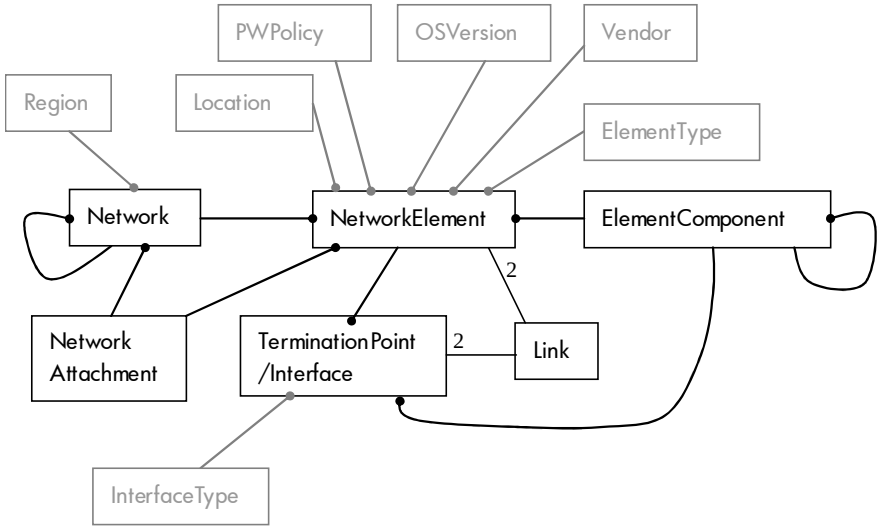
The PWPolicy table also falls in the category of model configuration. It contains username and passwords for a group of devices. This is useful when the same username and passwords are used for several devices and must be regularly updated. Instead of updating them as properties of each network element object, they will be updated on the PWPolicy object.

Object Classes of the CNRM

This section describes the (resource bean) object classes of the actual CNRM. The object classes and their entity relationships are depicted in Figure 8-2. The fields on the objects which have values constrained by the model configuration data classes are indicated by showing relationships to greyed-out model configuration objects.

- All of a provider's resources are placed in one or more regions.
- Each region will contain one or more networks; networks can also be hierarchically nested.
- A region contains a number of locations where equipment can be placed; locations are not nested within networks. This allows two or more networks to meet at a common location, where network elements belonging to different networks can be colocated.
- The central object class is NetworkElement (NE). An NE has a location and belongs to a network.
- An NE can be decomposed into ElementComponents, on which there may be interfaces which serve as termination points for traffic.
- NEs can be connected by links which terminate on termination points.
- NEs are of specific element types provided by vendors. The operational characteristics (command set) of the NE will depend on the firmware (OSVersion); the firmware is also provided by the equipment vendor.

Figure 8-2 Common Network Resource Model Objects



The model is topological in the sense that those links between network elements which are needed for activation purposes, because the devices at their endpoints must be configured for services, are represented by objects. But the topology does not need to be complete to support service activation: links between devices where the endpoints are not configured for individual services do not need to be represented. Unless such devices and their links are also added, the model cannot be used to generate a complete topological map of the provider's network.

The properties of the object classes, including the fields, are described in the following sections.

The following conventions apply to primary key and foreign key fields:

- Every object class has a single field primary key; hence foreign keys are also single fields.

- For the real resource objects (not system parameters) the primary key is always called '<class name>Id', e.g. NetworkId. Foreign keys referencing them have the same or a derived name, which includes the 'Id', sometimes preceded by an underscore. These primary keys are generated from sequences to automatically ensure uniqueness. Most of these objects also have a 'Name' field intended to be human meaningful. The convention for device interfaces is to follow the vendor's standard naming scheme that will also be used in device commands.

An 'M' in the type column indicates the field is mandatory.

The remaining subsections within this section describe each entity bean class of the model in turn. For each class there is a general description of the meaning of the class and a table which described the fields of the bean.

Bean Class Network

This object class is used to represent a collection of NEs and the links between them.

In general networks may be nested to any depth. A network which has nested subnetworks may also have direct member NEs (not members of any of the subnetworks).

The network object has a type field which can be used to distinguish different types of subnetwork, for example with respect to nesting. The intended meaning of the predefined values is as follows (use Figure 2-1 as architectural reference):

network — A set of PE devices (NEs) and the CE devices that are attached to them; also used for collections of unattached or unmanaged CE devices. No nesting.

access network — A collection of aggregation devices (NEs) and the simple access topologies (nested) attached to them. Aggregation devices will be linked by aggregation trunks to PE devices in a network. The relationship of the PE device to the access network is modelled with a NetworkAttachment object. S-VLAN tags assigned to attachment circuits over an access network must be unique on an access network. If several access networks are attached to the same PE device, uniqueness must extend to the union of such access networks (complete flow domain).

topology — A collection of access devices (NEs) connected in a simple ring or string topology within an access network. Nested within an access network. Aggregation switches may also be part of the ring or string, but in the model they will belong directly to the access network with a NetworkAttachment to represent that topologically they are part of the ring or string.

Other network types may be defined for other network structures, in particular if a solution uses nesting of networks.

Table 8-1 Fields of Network Bean

Name	Type	Description
NetworkId	String	Primary key
Name	String M	User friendly name
Type	enumeration: Network, AccessNetwork, Topology	See text above
ASN	String	Autonomous system number
Region	String	Foreign key, represents relationship to Region object

Name	Type	Description
ParentNetworkId	String	Foreign key, represents relationship to enclosing Network object

Bean Class NetworkElement

This object class represents individual network elements. Its fields hold the information needed when management communication takes place directly from HP Service Activator to each NE (using CLI). Some modification may be needed if the network elements are managed through an element manager. In simple cases it may suffice to use the Management_IP field to hold the address of the element manager. In more complex case it may be necessary to introduce an addtional object class to model the element manager and establish a relationship between the NetworkElement and the ElementManager.

NEs are implemented as reservable beans to allow mutual exclusion zones to be implemented easily in workflows: the NE is reserved over the section of the workflow where exclusive access is required to the model of the NE and its components, or to communicate with the NE. NEs are not expected to be reserved in their entirety for subscriber services.

NOTE The NE class can be specialized for a solution, the subclass will then inherit the reservability property. Do not set the maxCount attribute on the subclass.

NEs play different roles in the network: PE router, CE router, aggregation switch, access switch. The role is represented by the value of the Role field. The VPN SP uses specializations of the Network class for the different roles.

When comparing the Table 8-2 to the inventory UI forms for the NetworkElement bean beware that some of the fileds have labels which deviate from the values shown in the Name column.

Table 8-2 Fields of NetworkElement Bean

Name	Type	Description
NetworkElementId	String	Primary key
NetworkId	String	Foreign key, represents relationship to Network object that the NE belongs to
Name	String	Human meaningful name of the device
Description	String	Additional user information about the device
Location	String	Constrained by configured Locations
IP	String	Primary IP address of the device
Management_IP	String	IP address used for management communication with the device
ManagementInterface	enumeration: telnet, ssh	Protocol used for management communication with the device
PWPolicyEnabled	boolean	True if PWPolicy is used

Name	Type	Description
PWPolicy	String	References object defining username and password information to use for the device
UsernameEnabled	boolean	Deprecating, replaced with PWPolicy
Username	String	
Password	String (password)	
EnablePassword	String (password)	
Vendor	String	Constrained by configured Vendors
OSVersion	String	Constrained by configured OSVersions
ElementType	String	Constrained by configured ElementTypes
SerialNumber	String	Serial number of the device (inventory information)
Role	enumeration: PERouter, CERouter, AggregationSwitch, AccessSwitch	See text above
AdminState	enumeration: Up, Down, Unknown, Reserved	Administrative state, semantics defined by solution
LifeCycleState	enumeration: Planned, Preconfigured, Accessible, Ready	Life cycle state, semantics defined by solution
ROCommunity	String	SNMP read-only community string
RWCommunity	String	SNMP read-write community string
NNMi UUID	String	Universally unique identifier of corresponding NNM object
NNMi ID	String	Local identifier of corresponding NNM object
NNMi Last Update	Date	Time the object was last updated/refreshed from NNMi.

Bean Class NetworkAttachment

This object class models an n:m relationship between NetworkElements and Networks. An object of the class exists to represent that the NE is attached through a link to the network, which is not the one it belongs to, but may be a subordinate one, e.g. the NE can be an aggregation switch belonging to an access network and the network an access topology.

Table 8-3 Fields of NetworkAttachment Bean

Name	Type	Description
NetworkElementId	String	Identifies the NE
NetworkId	String	Identifies the Network the NE is attached to

Bean Class ElementComponent

This object class allows hierarchical decomposition of an NE, typically into racks, slots, cards, ports, in a generic way without restriction to specific component types.

The ComponentType field is used to indicate the type of component, determining its level in the containment hierarchy. The predefined values for this field include Slot and Port as well as Controller. An ElementComponent with ComponentType Slot represents a slot as well as the card that it holds. The value Controller is used for a port on a controller card which supports multiplexing of channels and has the ability to create channelized interfaces on a subset of the channels.

The capability to create a channelized interface on a Controller port is included in the Common Resources solution and can be invoked from the inventory UI.

The EC is reservable for the same reason that is described for NetworkElement above.

Table 8-4 Fields of ElementComponent Bean

Name	Type	Description
ElementComponentId	String	Primary key
NE_Id	String M	Foreign key, represents relationship to NE object that the component belongs to
ParentEC_Id	String	Foreign key, represents relationship to enclosing element component
Name	String M	User meaningful name of the component, vendor naming convention applies; typically the type of port/interface, rack number, slot number within rack, and port number on card will be included (when applicable)
Description	String	Additional user information about the component
State	enumeration: Up, Down, Unknown, Added, Removed	State, semantics defined by solution
ECType	enumeration: Slot, Port, Controller M	See text above

Name	Type	Description
Type	String	For a given ComponentType, the Type further characterizes the component. Type of slot: what cards can it holds; type of card: for example, SDH linecard, Ethernet linecard; type of port: Ethernet, GigEth, SDH with layer rate - STM-1 etc., E1, etc. By vendor convention this information may also be part of Name field. This field may be informative or may have semantics according to application.
ComponentNumber	String	Slot number, port number, etc. Not expected to be globally unique, only within the parent component. By vendor convention this information may also be part of Name field. This field may be informative or may have semantics according to application.
Capacity	int	Indicates number of units of capacity on the component. Used with ComponentType Port or Controller in situations where ports reside on a daughter cards, which are not modelled as separate ElementComponents, and the daughter card may comprise multiple physical ports or is preconfigured with multiple separate termination points (E1s within STM-1); indicates number of physical or logical ports

Bean Class TerminationPoint

Termination points reside on ports. Depending on the type of the port there can be many termination points on a port: for example VLAN tagged sub-interfaces on an Ethernet port or E1 channels on an SDH port. TerminationPoint fields are listed in Table 8-5.

Termination points often support switch/router interfaces, which can be configured for service. Interfaces are modelled as a specialization of TerminationPoint. See the section below

Table 8-5 Fields of TerminationPoint Bean

Name	Type	Description
TerminationPointId	String	Primary key
Name	String M	Number of the slot, vendor conventions apply

Name	Type	Description
NE_Id	String M	Foreign key, represents relationship to NE object that the termination point belongs to
EC_Id	String	Foreign key, represents relationship to element component (port) object that the termination point belongs to
State	enum: Up, Down, Unknown	Semantics of these values defined by VPN SP

Bean Class Interface

This specialization of TerminationPoint covers all the technology and application details that are added to generic termination points when they are used as switch/router interfaces for an NGN application.

An interface is a resource that can be reserved for a service.

A workflow to upload information about ElementComponents and interfaces from an NE, parse it and create the implied objects in the model is included with the CNRM solution and can be invoked from the inventory UI.

Likewise a workflow to create aggregated interfaces from interfaces on multiple ports is included with the CNRM and can be invoked from the inventory UI.

Table 8-6 Fields of Interface Bean

Name	Type	Description
TerminationPointId	String	Primary key
Type	String M	Type of interface
ParentIf	String	For a subinterface, represents relationship to parent interface
IPAddr	String	IP address assigned to the interface
Subtype	String	Subtype of interface, more specific than type (in VPN SP set to indicate how the interface is used for a service)
Encapsulation	String	For Ethernet interface: none or Eternet-dot1q For serial interface: FR, HDLC or PPP
Description	String	Uploaded from equipment
IfIndex	String	SNMP identification index
ActivationState	enumeratiuon: Activated, Failed, Undefined, Ready	Semantics of these values defined by VPN SP
UsageState	enumeration: Available,	Semantics of these values defined by VPN SP

Name	Type	Description
	SubIfPresent, Uplink, Reserved, InBundle, Trunk, ASBRLink, SwitchPort	
VLANId	String	Tag for traffic belonging to the (sub-) interface
VLANMode	String	
DLCI	String	For a frame relay interface, DLCI of the traffic
Timeslots	String	For channelized interface: which time slots
NumberOfSlots	String	For channelized interface: number of timeslots
Bandwidth	String	Bandwidth of interface (bps)
LMIType	String	For FR: Cisco, ansi or q933a
IntfType	String	For FR: dte/dce interface type
BundleKey	String	Alphanumeric name of an aggregate interface; identical on aggregate and its members
BundleId	String	Numeric identifier of an aggregate interface
NNMi_UUID	String	Universally unique identifier of corresponding NNM object
NNMi_ID	String	Local identifier of corresponding NNM object
NNMi_LastUpdate	Date	Time the object was last updated/refreshed from NNMi.

Bean Class Link

In a classical layer transport network model, a connection in one (server) layer of a network is used as a link in the topology of the next higher (client) layer. In the common network resource model the link property is the more interesting one. Algorithms will generally configure link endpoints to allow certain traffic (typically by VLAN tag) rather than explicitly create connections by setting up cross-connections. Cross-connections occur in the server (transmission) layers that are beyond the scope of the model.

A link connects two termination points, on two NEs.

Links are typically used as trunks, i.e. traffic for many different customer services identified by different VLAN tags may pass the same link.

Link fields are specified in Table 8-7.

Table 8-7 Fields of Link Bean

Name	Type	Description
LinkId	String	Identifies the link
Name	String	
NE1	String	Foreign key, represents relationship to NE object at endpoint 1
TP1	String	Foreign key, represents relationship to termination point object at endpoint 1
NE2	String	Foreign key, represents relationship to NE object at endpoint 2
TP2	String	Foreign key, represents relationship to termination point object at endpoint 2
Type	enumeration: access trunk, aggregation trunk	Used to distinguish different types of links
NNMi_UUID	String	Universally unique identifier of corresponding NNM object
NNMi_LastUpdateDate	String	Time the object was last updated/refreshed from NNMi.

User Interface and Launchable Functions for the CNRM

The CNRM can be populated with data either directly from the network elements or from NMMi, if the latter is present and set up to interwork with HP Service Activator. Dataload from NNMi comprises network elements, interfaces and links, direct dataload is for one network element at a time and comprises only the interfaces on each device.

The dataload process can be repeated regularly to update the model with new entities.

The process of loading data from NNMi has two phases. First all the data available from NNMi, within the specified scope, is loaded as intermediate entities. Network elements and interfaces which are equal to ones already existing in the CNRM are dropped at this point, and those which are new, i.e. have no counterpart already in the CNRM are created in the CNRM.

The second phase is manual and is concerned with links and with network elements and interface which already exist in the CNRM, but with one or more fields values being different from the one on the intermediate entity. These entities can be inspected and validated in a special tree on the inventory user interface. When there is a field value difference the user can select to update the value as determined in the first phase or keep the old value. Each link entity can be edited and accepted or rejected.

During the first phase extensive configuration data in the forms of mapping tables and enrichment descriptions., is applied to the data that is received in NNMi in order to represent according to the CNRM. The scope of the dataload, defined as a number of network elements identified by host name or IP addess, is part of this configuration. All those network elements, their interfaces and the links which interconnect them are then loaded in one process.

The CNRM solution pack includes definitions of three trees for the inventory user interface:

- Equipment, the tree that shows network with regions, network elements, interfaces and links. There are some operation privileges associated with this tree: nnm_operation_type allows UI crossload from Service Activator to NNMi, na_operation_type allows crossload from HP Service Activator to NA.
- Parameters, the tree where CNRM is configured: vendors, element types, information about NA and NNMi system that HP Service Activator shall interwork with, as well as information for the dataload function and for accessing NNMi and NA.
- NNMiDataload, the tree where data loaded from NMMi is validated in a temporary intermedate form by the user before it is accepted into resource inventory.

The NNMi dataload processes are controlled through workflows which are invoked from the root of the Equipment tree.

Alternatively, there is a workflow which can upload interfaces directly from a network element, without involving NNMi. This workflow is invoked from the network element branch in the Equipment tree. You can add a control workflow to traverse the network and invoke the interface upload from several network elements in a single process invoked at a higher level.

An additional function, implemented as a workflow invokable from the Equipment tree, is creation of channelized or aggregate interfaces on network elements which include controllers supporting such interfaces. Channelized interfaces can be created on STM-1 and E1 ports. Aggregate interfaces can be based on channels (64KB or E1) or on Ethernet interfaces.

Details about the functions that can be invoked from the interface UI are given in *HP Service Activator, User's and Administrator's Guide*.

9 Web Service Designer

The Web Service Designer is a tool dedicated to generate specialized servlets exposing the capability to run HP Service Activator workflows as web service methods. Each servlet will include a set of convenient methods mapping to the set of workflows which are required for integration with a specific type of client. The servlets can be deployed very simply with JBossWS on the HP Service Activator platform.

HP Service Activator also has a generic web service interface supporting a subset of the complete API of the workflow manager whose primary form is RMI. Compared to the generic interface the servlets generated with the Web Service Designer are specialized to contain dedicated methods to use for a particular application; they avoid exposing the concept of workflows and generic methods to control them. As system integrator you will control the names of the methods and their parameters.

Each web service is defined in an XML document, the *web service definition* file, which is created and edited with the tool. Consider this file a source file in the same way as workflow, resource (bean) and tree definition files. It should not be confused with a WSDL file (in W3C standardized web service definition language), which is used for external definition of a web service interface. The file produced by the tool is in a private format, the schema for which is in file *$ACTIVATOR_ETC*`/config/wsd-config.xsd`. It is concerned with both the definition of the interface and its mapping to workflows.

With the tool you can create and edit the web service definition; you can generate the servlet Java class, you can compile the class and build the web-application archive (.war file) containing the servlet, and you can deploy it on the HP Service Activator platform.

Servlets generated with the Web Service Designer will use SOAP over HTTP as transport, and the style of communication will be rpc (remote procedure call).

Defining a Web Service

The web service call to run an HP Service Activator workflow can be synchronous or asynchronous. This choice is made for each method; you can combine synchronous and asynchronous methods in the same servlet. In both cases the call will start the workflow as a job. In the synchronous case it then waits for the workflow job to complete and is able to return final values of case-packet variables as result information to the caller. In the asynchronous case the web service method does not wait for the job to complete; the only return information is the job id. A separate mechanism will then be needed for providing progress and result information to the caller; refer to the section “Northbound Interface” in chapter 5 for further discussion of this topic.

Before you use the tool you must prepare the following:

- a list of the workflows you want to be runnable as web service methods; for each one you must name the method you want to expose (typically the name will be similar to the name of the workflow)
- for each runnable workflow the list of case-packet variables that must be initializable from values of parameters of the web service

- for each synchronous method the list of case-packet variables whose final values must be returned to the caller
- for each of the input or output parameters a name for the parameter (the default choice will be the name of the case-packet variable) and a type; the type must agree with the type of the associated case-packet variable in the workflow (the tool does not read the types from workflow definitions; you must select them)
- for each method with multiple output parameters a decision on how to package the parameters in the result message (see below for more information)

With the tool you must then define the methods, each one named and mapped to a workflow and with a list of input and output parameters. From these definitions the tool will generate a Java class with your methods annotated to be callable as web service methods. The call parameters and result type of the generated methods will be translated to input and output messages to be exchanged between the caller and the HP Service Activator platform.

The input parameters will be combined as named elements (like members of a bean) of the input message.

The result returned from a servlet method must be a single object. If there is exactly one output parameter, the type of the result message will be the type of the output parameter. If there are multiple output parameters, you can choose to package them as a record (bean) with the parameters as named fields or as a hashmap with an entry for each parameter, in which case the parameter name is passed as the key. The latter form is more generic: the parameter names will not appear in the external definition (WSDL) of the interface, only as data in the messages.

You can use all the types supported for workflow case-packet variables for input and output parameters: String, Integer, Long, Boolean, Double, Float, Data, Object, List, Map, Set, Bean. Along with List and Set collection types you must also define a subtype, i.e. the type of the elements. Along with a Map parameter you must define two subtypes, for the keys and values, respectively. For Object and Bean you must provide a properly annotated Java class (library) that implements the class in question. Bean classes generated by the inventory builder can be used.

If you want to use the same multi-field response from several methods, you can define a bean class to represent the response. Then you will not need to list the bean members as output parameters for each method, and you will avoid to have several response bean classes generated with the same members but different names (derived from the name of the method).

There are two ways to authenticate the caller of the web service methods: either it is done globally, once per session, by the JBossWS authentication handler using username and password supplied in the SOAP header when the session is opened, or it is done by each method. When the latter mechanism is selected, username and password will be added as two parameters to each method.

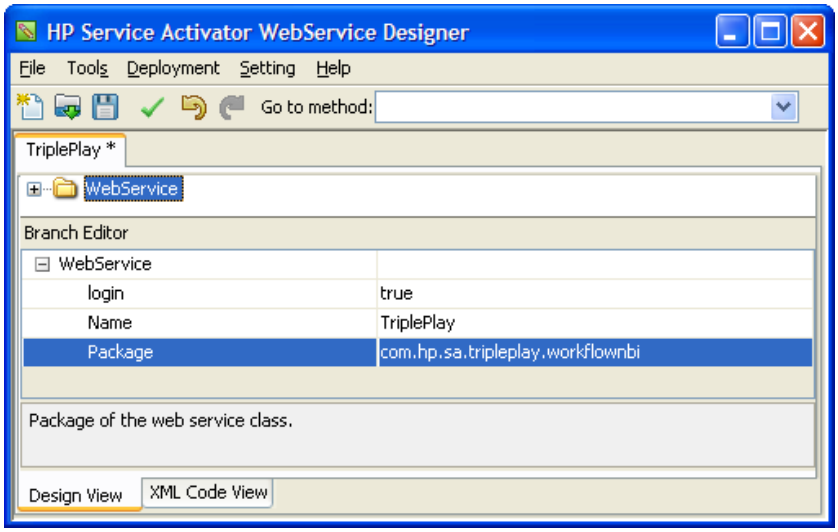
Web Service Designer Tool

The Web Service Designer is a stand-alone tool in the family of HP Service Activator design time tools. It is basically a UI tool for editing a web service definition with additional capabilities to generate servlet Java classes, build web archives and deploy them. All the functions except editing are also available as command line functions, and the tool is integrated with the Deployment Manager for deploying web services even across a multi-node cluster platform.

As the tool is quite simple, it has no dedicated manual; it is fully described in this chapter.

Figure 9-1 shows the user interface of the Web Service Designer, where the definition of an example web service called TriplePlay is open. The root of the web service tree is selected, allowing to edit the global properties of the web service.

Figure 9-1 **Web Service Designer, Global properties**



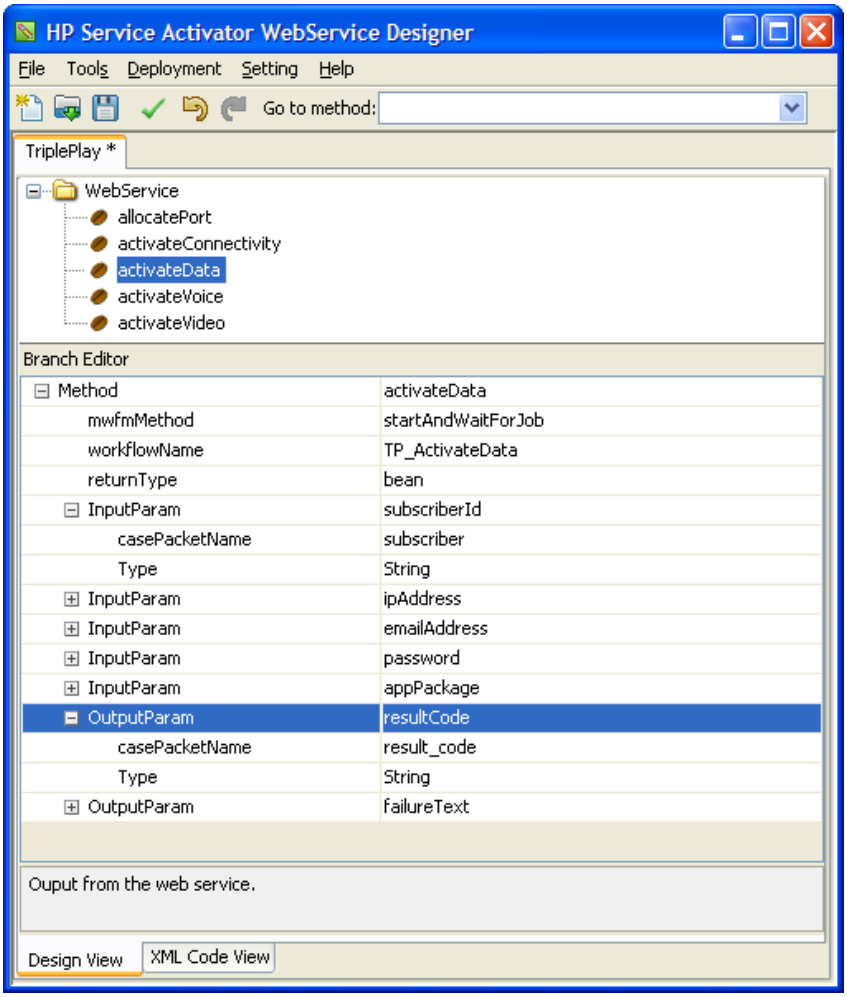
For the login property select true, if you want each method to login separately, or false, if you want session based authentication. In the first case each method will automatically get two parameters, username and password, before any explicitly specified parameters.

The Name property defines the name of the servlet. You must also define a package name for the Java class to be generated. The name of the servlet class will be appended to the package name you give.

Right-click in the upper frame on the tree root (labelled WebService) to bring up a menu where you can add or paste a method to the servlet (prepare for pasting by selecting the copy operation from the right-click menu of an existing method).

Figure 9-2 shows the Web Service Designer UI, where the servlet has been expanded to show its methods, and one of them (activateData) is selected, so that its properties and parameters can be edited. You can right-click on a method in the upper frame to get a menu comprising the following operations: copy the method, delete the method, move up and move down.

Figure 9-2 Web Service Designer, Method properties and parameters



To add an input or output parameter to the selected method, right-click on the root of the lower frame (labelled Method) and select the appropriate item in the menu that appears; the same applies to the mwfmMethod and workflowName properties of the method, which in Figure 9-2 have already been added.

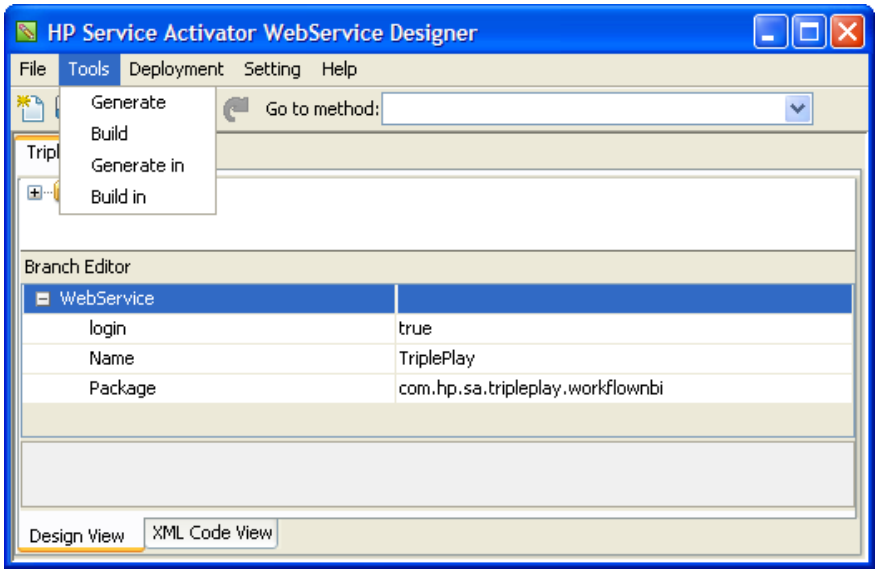
The method properties are as follows:

Method	the name of the method is editable
mwfmMethod	selectable as startJob (asynchronous, applies if omitted) or startAndWaitForJob (synchronous)
workflowName	name of the workflow to run (Method is used if omitted)
returnType	selectable as none, all (hashmap) or bean (record)

For each input or output parameter you can enter its name, add the name of the associated case-packet variable and select its type. If you omit the name of the variable it is assumed to be the same as the parameter name.

When you have finished defining the servlet and its methods, you can generate the code and build the web application archive. Select each step from the Tools menu which is shown in Figure 9-3.

Figure 9-3 Web Service Designer Tools menu



If your generated code will need to make reference to other Java classes, for example resource beans, you can place such classes in libraries (.jar files) and add those libraries into the process in the Web Service Designer through the 'Default Libraries...' operation that you can launch from the Setting menu.

To build and deploy the servlet, first select Generate from the Tools menu. This operation will generate the Java code for the servlet. It will generate a sequence of directories, according to the Package defined, starting with classes, in the directory where the web service definition file is located, normally <Solution>/etc/web-services. The operation also generates a deployment descriptor file (web.xml), defining the mapping from URL to servlet code. This file is placed in the web subdirectory, which will be a sibling to the classes directory. If you want to place the classes and web directory in a different location (from that of the web service definition file), choose 'Generate in' from the Tools menu and browse to the desired location.

Next select Build. This operation will compile the Java code with the specified libraries in the classpath and place the resulting class file along with you have specified, in a web archive, packaged as a .war file, in the war directory. Again, if you do not want the generated war directory to be place in the same parent directory as the web service definition file, use 'Build in' from the Tools and browse to the desired location.

Finally, if you also want to deploy the web service, select the Deploy operation in the Deployment menu. This function will retrieve the .war file from the war directory and copy it to the *$JBOSS_DEPLOY/hpovact.sar* directory, whereby it will be activated when JBoss is restarted (it is not hot deployed in a subdirectory of *$JBOSS_DEPLOY*).

To include the web service servlet in a deployed solution, you must include either the definition file and the library files or the generated .war file in the solution distribution archive and include in the deployment descriptor (deploy.xml) file a description of how to (generate and) deploy the servlet.

Extracting WSDL Definition

When a servlet has been deployed with JBossWS you can view it in the list of deployed services as shown in Figure 9-4. Use this URL in your web browser:

```
http://HPSA host:HPSA port/jbossws/services
```

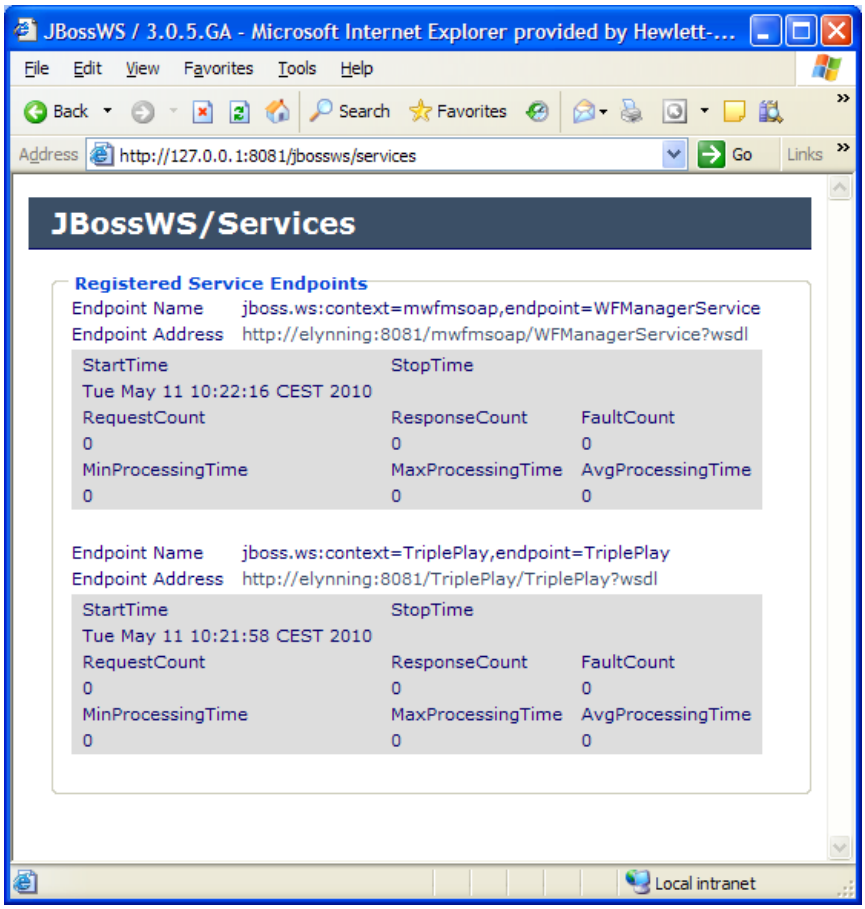
The table will have an entry for each deployed web service. In Figure 9-4 two services are shown: the generic web service for the API of the workflow manager and the dedicated TriplePlay service.

To obtain the WSDL for a service, just click on its Endpoint Address (http:// ..... ?wsdl). JBossWS will generate and return the WSDL document. You can save it from the browser to a file. You will need to edit the file to insert the proper XML header:

```
<?xml version="1.0" encoding="UTF-8"?>
```

You can study the WSDL document to observe the definitions of your web service methods as operations with input and output messages, and you can use it as external documentation of the interface supported by the servlet. Several software products used as client systems will be able to import the WSDL document and use it directly to drive communication with the HP Service Activator server or to generate client side artifacts for integration.

Figure 9-4 JBossWS Deployed Services List



10 Integration with NNMi

This chapter covers the integration of HP Service Activator with the HP NGOSS product Network Automation.

The components dedicated to integration of HP Service Activator with NNMi are licensed for use as the *NNMi Liaison*. To use these components the RTU (right to use) must be purchased in addition to the license for the HP Service Activator core product.

Positioning of NNMi

NNMi enables a customer to monitor and manage all the devices in an IP network from a single point of control; a graphical web browser based user interface offers the user topological and tabular views of the network and its status, and is launch pad for control functions.

NNMi interworks with devices by means of the SNMP protocol. The product has built-in knowledge of a large number of devices through their SNMP MIBs.

NNMi can automatically discover and build a model of a network as it is.

Fault reports from devices are received as SNMP traps and correlated to perform root cause and service impact analysis.

Summary of Benefits of Integration with NNMi

Overall, instead of two separate solutions NNMi and Service Activator will appear and behave as a well-integrated single solution for network and service management including service activation.

The capabilities of NNMi are enhanced in ways which will significantly assist operators in focusing on high priority incidents to meet committed SLAs and provide better visibility of the state of important services.

- Service Activator workflows to activate services by configuring network elements can also enrich NNMi objects with service and customer related information. This will enable prioritization of network faults based on evalauation of service importance. You get new possibilities to group interfaces based on the type of service that is using each interface, or the customer that is using the interface.
- For example device interfaces can be grouped by service type or customer.
- Enhanced synchronization of NNMi data model with the network, as Service Activator may request NNMi to rediscover the state of a device when a service has been activated on it.
- Service Activator workflows to perform diagnostic analysis and corrective actions can be initiated from NNMi. This will enrich the scope of control actions an NNMi operator can take when faults have been discovered (fully automated ?)
- GUI crosslaunch from NNMi views to Service Activator inventory and service order views allows an operator to view Service Activator data and activity related to known objects.
- Not only the network, but also the activation system based on Service Activator can be monitored with NNMi

Through the integration with NNMi the Service Activator solution will benefit greatly from availability of consolidated information about the network and its state:

- The resource inventory data model of the network can be populated by loading data already discovered by NNMi. Likewise the data model can be synchronized with the NMMi data model on an ongoing basis.
- Service Activator workflows can obtain real-time device status from NNMi.
- GUI crosslaunch from a Service Activator inventory view into NNMi topology view and status lists. For example from Service Activator resource inventory objects such as network or subnetwork, launch a topological view. Or from Service Activator service inventory customer object, launch an NNMi interface group view of all the device interfaces carrying the customer's traffic.

Readily Available Capabilities with NNMi

The capabilities that are available "out of the box" without customizing items like Service Activator workflows, modules and data models include populating and synchronizing the Common Network Resource Model and mutual GUI crosslaunch.

Loading and Synchronizing of the CNRM

The workflow for this purpose and the definition of the inventory UI tree from which it can be launched is provided as part of the Common Network Resource Model deployable solution pack which is described in chapter 8.

The first time the dataloader is used it will populate the CNRM data model with network elements, interfaces and links. On subsequent runs it will update the CNRM data model to include newly discovered objects.

UI Cross Launch from Service Activator to NNMi

Crosslaunch from Service Activator to NNMi generally requires JSP customization and incorporation in an inventory tree or main UI menu (see below). Some cross launch functions are precustomized as part of CNRM:

- from every interface branch in the CNRM equipment tree the NNMi interface form is launchable
- from every network element branch in the CNRM equipment tree the NNMi L2 and L3 neighbor views are launchable

UI Cross Launch from NNMi to Service Activator

Cross launch from NNMi to Service Activator inventory UI is simple to configure in NNMi. It will be a menu item associated with the relevant NNMi object type, for example node or interface. The substance of the menu item is the URL to launch (see "Summary of Techniques for Configuring Integration on NNMi" below).

Components for Customized Integration with NNMi

The components that are available on Service Activator for use as a basis for customized integration with NNMi include a plug-in to automate configuration of NNMi as part of service activation, three workflow manager modules and three workflow nodes. The nodes serve essentially to invoke the functions provided by the modules.

NNMi conf plug-in

The plug-in can create an interface group view on NNMi.

SNMP trap module

This is a general purpose module, not restricted to integration with NNMi, which can issue an SNMP trap.

NNMi module

This module acts as a web service client towards NNMi. It supports all needed calls from Service Activator to the NNMi web application, as used by the workflow nodes and the data load module. It can create annotations on NNMi objects and pull data from NNMi.

Data load module

This module performs the task of loading and synchronizing the CNRM data model from/with NNMi using the functions of the NNMi module to communicate with NNMi.

Workflow Nodes

Three workflow nodes are available that can be used in solution workflows:

- GetBeans retrieves objects from NNMi's network data model
- RediscoverHost requests NNMi to rediscover a specific network element
- UpdateCustomAttributes sets values of custom attributes on an NNMi data object

For details, consult *HP Service Activator, Workflows and the Workflow Manager*.

Summary of Techniques for Configuring Integration on NNMi

To configure nodes and interfaces for cross launch of Service Activator views: define for a type of NNMi object a URL action to launch an Service Activator view. The action may use one or more ext properties of the NNMi installation (for example hostname and port of HPSA server) and custom attribute values of the object, such as primary key of the HPSA counterpart object) to construct the URL. The URL action will typically include a filter to enable it when necessary custom attributes are set. Note that Service Activator has a script that can be used to set ext properties on an NNMi installation. Consult NNMi online help for URL Actions for specific detailed information.

To create action scripts to launch Service Activator workflows you can install and use the mwfm command line tool.

Customizing and Configuring Service Activator to Work with NNMi

Configure the workflow manager to use the NNMi modules, and in the Parameters tree for the CNRM configure the basic parameters to connect to the NNMi system and the function to data load from NNMi as described in *HP Service Activator, User's and Administrator's Guide*.

Write workflows to perform functions that shall be invoked as actions from NNMi; there are no special integration concerns for the workflow as such; integration is done by the invoking script on the NNMi side.

Use NNMi workflow nodes in activation workflows to retrieve infornation from NNM, create annotations on NNMi objects, and invoke host rediscovery.

Create JSPs (with Struts actions) to implement cross load functions in addition to the ones already available (see above); for this you will need to understand NNMi URLs. As an example, study the UI part of the CRModel solution, the Struts-config and JSPs.

11 Integration with NA

This chapter covers the integration of HP Service Activator with the HP NGOSS product Network Automation.

The components dedicated to integration of Service Activator with NA are licensed for use as the *NA Liaison.* To use these components the RTU (right to use) must be purchased in addition to the license for the Service Activator core product.

Positioning of NA

NA maintains router configurations and enforces configuration policies.

NA connects to devices and configures them. NA also allows other system to establish connections to devices and intercepts them to monitor and log the configuration changes. NA typically establishes a baseline configuration for each device according to its role in the network.

NA enforces compliance of policies for configuring devices; this involves monitoring, comparing configurations to policies applicable to a group that a device belongs to, detection of violations, alerting.

NA manages and downloads firmware versions and patches to all devices in a network and performs backup and restoring of device configurations.

NA is generally not concerned with configuring specific interfaces on service provider devices for individual customer services.

Summary of Benefits of Integration with NA

By combining configuration management, setting up baseline configuration of devices done with NA, and service activation, setup of customer service specific configuration done with Service Activator, the complete device/service lifecycle is managed. NA is used heavily in the buildup of network and service infrastructure, specifically to configure routing protocols, MPLS forwarding, core network interfaces, BGP peering, ASN, general security settings, ACLs, passwords, QoS for the service provider's own traffic and other aspects which are general and not specific to individual customers' VPNs. Once the infrastructure for a service offering is in place, Service Activator will automate flow-through configuration of devices for all aspects of customer specific services with minimal operator involvement and elimination of the requirement for operators to master the commands needed to configure services.

Service Activator workflows can be used to control and automate processes consisting of several NA actions, for example to run test scripts, draw conclusions and take remedial actions.

NA can provide connectivity to devices that Service Activator can use including tunnelling to devices that are not easily reachable, thus eliminating the need to configure Service Activator with device specific usernames, passwords. etc., and ensuring that all device interactions to activate services are logged together with all other device interactions.

NA can ensure integrity of services configured on devices: when configuring the device Service Activator will also configure NA to monitor the setup, leveraging NA's ability to enforce policy compliance.

Also in the process of configuring a device for a specific customer service, Service Activator can instruct NA to backup the device configuration, after all changes have been made, and annotate the backup with a comment stating it was requested by Service Activator due to the specific service that was configured.

As a minimum, even if the functions of NA and Service Activator are not combined to obtain these benefits, some coordination is necessary to avoid counterproductive effects such as NA detecting and "repairing" by undoing the service specific device configurations set up by Service Activator as violations of policy.

Readily Available Capabilities with NA

NA, with its management of connections across the network and into shielded network domains can be used as a proxy for network elements to provide connectivity for Service Activator to establish command sessions with network elements.

The NA user interface can be accessed by cross launch from the CNRM tree of the Service Activator inventory user interface. Note that NA does not support Single Sign On.

The platform for running Service Activator and NA can be shared, including hardware, operating system, and Oracle. The two applications both include JBoss (different versions !), and it will be necessary to configure one or the other JBoss instance to use non-fault port numbers for overlapping functions, such as web service access, etc.

Service Activator Components for Customized Integration with NA

A large number of workflow nodes are available, allowing Service Activator workflow to exercise a range of the capabilities of NA, including:

- take backup snapshot of device
- run scripts of different types
- create/delete devices and device groups, and manage group membership
- manage associations between device groups and rules, conditions and policies
- retrieve different types of information from NA to Service Activator

These nodes interwork with NA by invoking its web service interface through a dedicated module.

Summary of Techniques for Configuring Integration on NA

Where NA functions involve the running of scripts, it will be possible in a script to use the mwfm command line tool to run a Service Activator workflow.

Cross launching of Service Activator UI from NA is not possible.

Customizing and Configuring Service Activator to Work with NA

When Service Activator activates a service by configuring one or more devices with specific constructs, for example Virtual Router Forwarding tables, the activation workflow can be extended to set up NA to monitor the integrity of those constructs. The rule that defines remedial actions when the policy has been violated must be created on NA in advance and may involve such actions as creating trouble tickets, sending email, even executing automatic repair. NA is generally geared to manage groups of devices, not individual services, so it will be necessary to define a dedicated "group" per service instance and add the affected device to that group. Then the policy which knows the command pattern for the construct for the service can be defined for the group and

associated with the action rule. The activation workflow can accomplish all of that by means of the workflow nodes for NA integration.

In general workflows can be written to perform any tasks which can be accomplished through execution of NA actions or scripts, using the workflow nodes that are available. Such workflows can be integrated in flow-through activation systems, or they can be launchable from the inventory UI.

12 Development Hints

This chapter contains miscellaneous useful information that will be useful during development, including testing and debugging.

Configuring Database Credentials

The Workflow Designer, Deployment Manager, Tree Deployer, Tree Designer and the Inventory Builder all require the user name and password for the system database to deploy data. To avoid having to enter the credentials repeatedly, it is possible to create a configuration file with this information that is read by the tools during start up. The file must be named dbAccess.cfg and be placed in the directory *$ACTIVATOR_ETC*/`config`. An example file exists in this directory and is named `dbAccess_example.cfg`.

NOTE This file must not be present in a production environment.

Configuring Injection of Request Messages for Test

If you use the socket listener to receive incoming request messages, a message injector can be configured as part of the System Administrator menu in the navigation pane of the main UI. Files that you wish to be able to inject must be placed in *$ACTIVATOR_ETC*/`templates_files/<your solution>`.

In the UI configuration file *$JBOSS_ACTIVATOR*/`WEB-INF/web.xml` configure these two parameters, then restart HP Service Activator.

tests	True enables injection of messages to a socket (CRM simulation). Default is false.
socketListener_port	The socket port to which tests messages will be injected. This port must match the configuration of the socket listener module which is to receive the messages.

A menu item named `Test Messaging` will appear at the botton. When you select it a view listing available files for injecting are shown. Right-click on the one you want and select `Start Test`. This will inject the message into the socket listener which will should start your workflow.

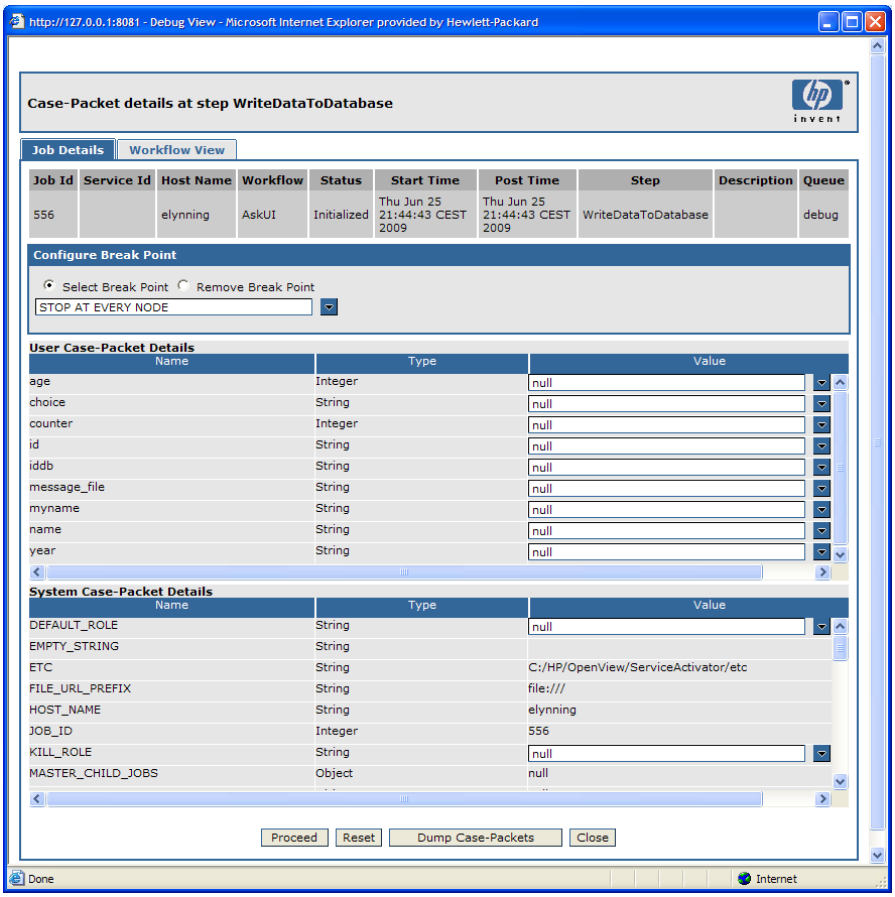
Workflow Testing and Debugging

To avoid having to inject messages to start workflows, you can start your workflow from the Workflows view available in the Work Area of the navigation pane of the main UI. For this to work, your workflow must be independent of pre-initialized variables. For initial testing you can start the workflow in debug mode and initialize variables manually (see below). When you are confident with the workflow, make the northbound integration and test the interaction with a northbound system to start the workflow.

There are several ways to trace what is happening when you run your new and unproven workflows.

- You can output trace messages that you can read in the Messages view. A number of messages will be generated automatically by the workflow manager when the workflow contains or does something invalid.
- You can study the log file produced by the workflow manager and resource manager. You can include entries generated from your own workflow (Log node) or plug-in (context method). Such log entries can be directed to dedicated files.
- You can dump the case-packet of your workflow to a file (WriteCasePacket node).
- You can place extra AskFor nodes to interact with your workflow job, thus controlling its progress.
- If you set the `tests` variable in the UI configuration file (see above), you can start your workflow in debug mode from the Workflows view. It will then interact with you from a special queue, `Debug`, in the Active Jobs view when it reaches a breakpoint node, initially the first node, before the breakpoint node is executed. In the debug interaction window you can select single-step or set the next breakpoint, and you can read and edit values of case-packet variables, for example to initialize them as discussed above. You can view the current breakpoint in a flowchart (workflow) view, and you can dump the case-packet to a file. An example of the debug window is shown in Figure 12-1.

Figure 12-1 Debug Interaction Window



13 System Configuration

Most of the configurable aspects of HP Service Activator solutions are covered in detail elsewhere:

- The resource manager has its own configuration file (`resmgr.xml`) which is described in detail in *HP Service Activator, Developing Plug-Ins and Compound Tasks*
- The workflow manager has its own configuration file (`mwfm.xml`) which is described in detail in *HP Service Activator, Workflows and the Workflow Manager*. Note that some important workflow manager modules are not configured to be active upon installation, you must change the configuration file to use these modules: authentication, audit, socket listener/sender, statistics collection
- The user interface is configured has a main configuration file (web.xml) and separate files for solution specific modifications to the default Work Area menu. Both aspect are described in *HP Service Activator, User's and Administrator's Guide*.
- Appendix B contains a list with descriptions of all configuration files

A couple of topics deserve special mention here.

Number of Threads and Memory Usage

The amount of parallel activity that can take place in an HP Service Activator system is configured for the workflow manager (in file `mwfm.xml`). The maximum number of workflow jobs that can exist simultaneously is determined by the parameter `Max-Work-List-Length`. When this number is reached, attempts to start additional jobs will fail. A typically much smaller number of operating system threads will be used to execute the workflows, determined by the (global) parameter `Max-Threads`. The assignment of threads to active jobs is managed by the Work Manager module.

When a job is waiting on a queue or for an activation, it will not occupy a workflow job thread. However, to control the amount of parallel activation activity, the Activation Manager module (normally named activator), manages a private pool of so-called activation threads. The maximum number of concurrent activations will be limited by the (module) parameter `max_threads`. When all these threads are in use, additional workflow jobs attempting to start activations will be queued. Activations do not occupy operating system threads in the workflow manager, but they do in the resource manager. That is the real significance of the `max_threads` parameter.

There are two JBoss files with parameters that configure numbers of threads:

- *`$JBOSS_DEPLOY`*`/hpovact-ra-ds.xml`, parameter `max-pool-size`
- *`$JBOSS_DEPLOY`*`/hpovact.sar/hpoavct-EJBs.jar/META-INF/jboss.xml`, parameter: `MaximumSize` in `container-pool-conf`.

Both of these parameters should be set 10 threads higher than `max_threads` for the Activation Manager module.

The memory consumption of the workflow engine, i.e. workflow manager plus resource manager, will depend on the number and size of deployed workflows and plug-ins, but even more on the number of workflow threads and activation threads along with the sizes of their case-packets and

the variables of the plug-ins. If you set the maximum numbers for the threads very high, out-of-memory exceptions may occur at run-time. You should make sure that your system testing goes to the limits that you have configured.

Instructions for increasing the memory size of the JBoss process that includes the workflow engine are found in *HP Service Activator, Installation Guide*.

Database Connections

Database connections are needed by a number of different components of the HP Service Activator system. Here is an overview of where the numbers of connections are configured for different purposes:

- For the workflow manager's access to system repositories and the default database module for workflows: *$JBOSS_DEPLOY*`/mwfm-default-ds.xml`
- For the resource manager for purposes of locking and managing transactions: *$JBOSS_DEPLOY*`/resmgr-default-ds.xml`
- For JBoss to access the task repository: *$JBOSS_DEPLOY*`/hpovact-oracle-ds.xml`
- For JBoss to support the UI except inventory UI: *$JBOSS_DEPLOY*`/hpovact-ui-ds.xml`
- For JBoss to support the inventory UI: *$JBOSS_DEPLOY*`/hpovact-inventory-ds.xml`

14 Localization

This chapter describes the different resource bundles that must be translated to localize an HP Service Activator solution.

In general the Java resource bundles you must translate are files with names ending in `_en.properties`. You must make a copy of each resource bundle file in the same directory as the original file, where you replace `_en` in the file name with the appropriate abbreviation for the locale, like `_jp` or `_dk`.

Then you must translate the contents of each file to the language of the locale. The files must be saved encoded in the ISO 8859-1 character set with appropriate escape sequences to represent characters that do not have 8-bit codes; the Java utility `native2ascii` may be helpful to convert from a UTF character set to ISO 8859-1.

NOTE When the same English word or phrase appears in property files for several parts of the UI, make sure to translate it consistently. On the other hand, a property may be used in several places within a single part of the UI, and you should make sure that the translation you choose is appropriate in all of those places.

After you modify resource bundles, restart HP Service Activator for the new resources to become available.

Localizing the Main UI Window and most views

The basic resource property bundles are found in *`$ACTIVATOR_ETC`*`/nls`.

There are a couple of small exceptions: column headers for the Logs view, non-default column headers for the Active Jobs view, and non-default entries in the Work Area menu are not defined in the resource property bundles. The “non-default” items occur only when the UI has been configured in special ways; follow the reference at the beginning of chapter 13 to understand the options.

If you want to localize the column header for Logs views, do it directly in file *`$JBOSS_ACTIVATOR`*`/xsl/saLogs.xslt`.

If you define non-default columns for the Active Jobs view, you must define them exactly as you want them to appear in file *`$JBOSS_ACTIVATOR`*`/WEB-INF/web.xml`.

If you introduce new items in the Work Area menu or change the label on existing ones, you must define the labels exactly as you want them to appear in the solution menu file.

Localizing the Service Order View and more

Three parts of the UI are implemented with Java Server Faces: the Service Order View, the job counters at the bottom of the navigation pane, and the Debug window. The resource property bundles for these parts are found in *`$JBOSS_ACTIVATOR`*`/WEB-INF/classes/jsf-resources`. When you add support for a new locale, you must also add that locale in file *`$JBOSS_ACTIVATOR`*`/WEB-INF/classes/faces-config/locales.xml`.

Localizing the User Management UI

Files to localize for the User Management UI are:
$JBOSS_ACTIVATOR/WEB-INF/classes/umm.properties and
$JBOSS_ACTIVATOR/WEB-INF/classes/com/hp/ov/activator/mwfm/umm/*.properties

Localizing Inventory UI

There are several parts to localize for the Inventory UI relating to inventory resources and tree definitions, as described in a dedicated chapter in *HP Service Activator, Inventory Subsystem*.

Custom UI Files in Solution Source Hierarchy

In the solution source hierarchy you build, files to be deployed for the UI should be placed under the UI directory, which is a direct child directory of the main solution directory (in Figure 3-2 there is no UI directory, but it should be at same level as etc, inventory, plugins). The paths used under the UI directory should equal the target paths relative to *$JBOSS_ACTIVATOR*.

Appendix A Scripts

This table contains locations and descriptions of the scripts that are available in Service Activator. Unless otherwise indicated, these files are located in the $ACTIVATOR_BIN directory. In general the scripts have help options to explain usage, or do so when called without or with incorrect parameters.

Table A-1 Service Activator Scripts

Script	Description
`/etc/init.d/activator`	Starts/stops the Service Activator processes (Solaris).
`/sbin/init.d/activator`	Starts/stops the Service Activator processes (HP-UX).
`ActivatorConfig[.bat]`	Configures Service Activator for a specific environment.
`AssignNonRoot`	Configures Service Activator to run as non-root, only on UNIX
`CatchSocketSenderMessages[.bat]`	Listens for messages on a given port and prints those messages to stdout. This script is typically used for testing and demonstration of the SocketSenderModule of the Workflow Manager. By default, it listens on port 4099, but takes a single parameter to specify the port.
`checkLicence.[bat]`	Checks the status of the HP OpenView AutoPass license and prints out debug information.
`CleanLogs[.bat]`	Deletes all but the active logs
`crypt[.bat]`	Encrypts or decrypts a password for local use, to avoid storing unencrypted passwords in the workflow manager configuration file.
`DataSourceConfiguration[.bat]`	Assists in management of data source configurations.
`dc[.bat]`	Starts Data Collector, command line tool for gathering information about the Service Activator components (Workflow Manager, Resource Manager, JBoss).
`DeleteCompleteTransactions[.bat]`	Cleans up saved completed activation

Script	Description
	transactions.
`deploymentmanager[.bat]`	Invokes the Deployment Manager executable.
`designer[.bat]`	Runs the Workflow Designer tool
`generateEncryptedPassword[.bat]`	Utility to generate an encrypted password. This can be used when an additional data source file has to be created.
`generateMD5[.bat]`	Calculates MD5 checksum for a file.
`InventoryBuilder[.bat]`	Runs the InventoryBuilder tool.
`InventoryTreeDeployer[.bat]`	Runs the InventoryTreeDeployer tool.
`InventoryTreeDesigner[.bat]`	Runs the InventoryTreeDesigner tool.
`modifySystemPassword[.bat]`	Utility to update the system user password.
`mwfmtool[.bat]`	A command line tool for performing workflow engine tasks such as starting workflows and viewing posted messages. If this script is executed without any parameters, it will display a list of all the tasks that can be performed.
`NNMExtProperties[.bat]`	Creates ext properties on NNMi.
`remove.serviceactivator`	Uninstalls Service Activator on UNIX.
`servicebuilder[.bat]`	Invokes the Service Builder executable, either the command line (if arguments are passed) or the GUI (if no arguments are passed).
`TestAtomicTask[.bat]`	Starts an atomic task for testing purposes.
`UMMData[.bat]`	Imports/export roles with inventory UI privileges from/to file.
`updateLicence.[bat]`	The script lets you update your trial or existing licence for HP Service Activator.
`ViewTransactionState[.bat]`	Displays the different states of a completed transaction.
`WebServiceDesigner[.bat]`	Runs the WebServiceDesigner tool.

Some files with examples of scripts for use with Oracle are provided in the directory *$ACTIVATOR_OPT*`/examples/database/oracle10` as listed in the following table.

Table A-2 Example Files for Oracle

File	Description
`create_ovdb_ora.sh`	Example of files used to create an Oracle 10g database instance.
`listener.ora`	Example of an Oracle listener configuration file used to enable remote database access.
`tnsnames.ora`	Example of an Oracle local naming parameters file used to define aliases for referencing Oracle databases.

Appendix B Configuration Files

The following table identifies and describes the configuration files that are provided with Service Activator. Unless otherwise indicated, these files are located in *$ACTIVATOR_ETC*/config.

Table B-1 Service Activator Configuration Files

File	Description
CompoundTask.dtd	Document type definition (DTD) for compound task files created by Service Builder.
designer.xml	Configuration file for the Workflow Designer.
inventoryTree.dtd	DTD for inventory tree definition file.
mwfm.xml (mwfm.dtd)	Configuration file for the Workflow Manager and associated DTD.
par.dtd	DTD for the MANIFEST/par.xml file found in the Plug-in Archive (PAR).
resmgr.xml (resmgr.dtd)	Configuration file for the Resource Manager and associated DTD.
role_mappings.xml (role_mappings.dtd)	Definition file and associated DTD for configuring role mappings. The role mappings definition file is optional.
service_builder.xml	Configuration file for Service Builder.
$ACTIVATOR_ETC/ workflows/workflow.dtd	DTD for workflow definition.
$JBOSS_DEPLOY/ hpovact.sar/META-INF/ jboss-service.xml	Contains the classpath for the Service Activator J2EE components deployed in JBoss.
$JBOSS_DEPLOY/ hpovact.sar/activator.war/ WEB-INF/web.xml	Configuration file for the UI and servlets. For additional information, see the following references: Chapter **Error! Reference source not found.**, the section "**Error! Reference source not found.**" in this manual; Chapter 4, the section "Adding a Data Source for Inventory UI" and Chapter 5, the beginning, in *HP Service Activator, Inventory Subsystem*; Chapter 4, the section "AskFor" in *HP Service Activator, Workflows and the Workflow Manager*.

File	Description
$JBOSS_DEPLOY/ hpovact.sar/deployer.war/ WEB-INF/web.xml	Contains the configuration for the deployer servlet used by Service Builder. See "Configuring Authentication or Authorization" in *HP Service Activator, Developing Plug-Ins and Compound Tasks* for a description of the configurable parameters in this file.
$JBOSS_DEPLOY/ hpovact-inventory-ds.xml	JBoss data source file containing the database configuration used by the UI.
$JBOSS_DEPLOY/ hpovact-ui-ds.xml	JBoss data source file containing the datasource configuration used by the UI.
dm.dtd (dm.xml)	Configuration file and associated DTD for the Deployment Manager.
deploy.dtd	DTD for deployment descriptor.
$JBOSS_DEPLOY/ mwfm-default-ds.xml	JBoss data source file configuring connections used by the default ("db") database module in the Workflow Manager.
$JBOSS_DEPLOY/ resmgr-default-ds.xml	JBoss data source file configuring connections used by the Resource Manager.
$JBOSS_HOME/ server/default/conf/ login-config.xml	The JAAS login configuration file. It is here that the information about user name and encrypted password for datasources are configured.
$JBOSS_DEPLOY/ hpovact.sar/ hpovact-EJBs.jar/ META-INF/jboss.xml	The MaximumSize setting configures the maximum number of concurrent invocations of the EJB that performs task activations. For additional information, see Chapter 4, "Configuring Activation Parameters," in *HP Service Activator, Developing Plug-Ins and Compound Tasks*.
*$JBOSS_DEPLOY/*hpovact.sar/ hpovact-EJBs.jar/ META-INF/ejb-jar.xml	Contains the activation time-out configuration parameter. See Chapter 4, "Configuring Activation Parameters," in *HP Service Activator, Developing Plug-Ins and Compound Tasks* for additional information.
$JBOSS_DEPLOY/ hpovact-ra-ds.xml	The max-pool-size setting configures the maximum number of instances of the resource adapter that can be used during task activations. For additional information, see Chapter 4, "Configuring Activation Parameters," in *HP Service Activator, Developing Plug-Ins and Compound Tasks*.
$JBOSS_DEPLOY/ hpovact-oracle-ds.xml	JBoss data source file configuring connections used to access the task repository.
menu.xml (menu.dtd)	Default menu definition file for Work Area menu and its DTD.
solutionmenu.dtd	DTD for solution specific modifications to menu definition.

Appendix C Java Message Service

This appendix gives you a quick introduction to those aspects of JMS, Java Message Service, which are relevant with respect to using this type of communication interface for an HP Service Activator solution.

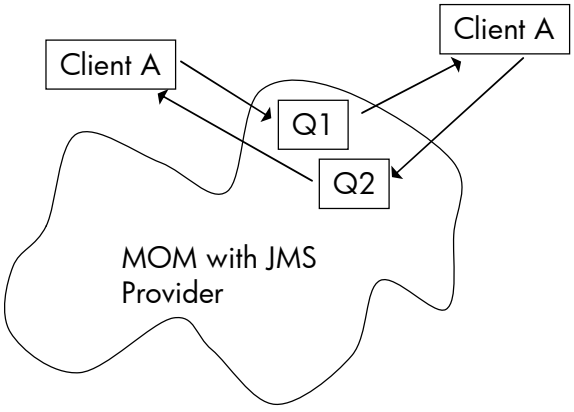
JMS is a standardized interface for access from a Java-based client to a MOM infrastructure (Message Oriented Middleware). A MOM product is used build the infrastructure for intercommunication between applications within an enterprise. Several MOM products come with JMS providers, i.e. a software layer that makes it possible to access the intercommunication service over a JMS interface. When such a MOM is available it can be used for intercommunication between HP Service Activator and other systems that it must be integrated with.

JMS comprises two data transfer modes, using two kinds of service access points known as *destinations*: queues and topics. An installation may support one or both modes. Queues are for point-to-point communication: one client, the sender, puts a message into the queue; the other client receives the message. Topics are for one-to-many communication, where the sender does not know the receivers. A sender client publishes a message on a topic; several clients can subscribe to receive messages from the topic.

When one client receives request messages from another client, the queue is the appropriate mode of communication. If the response is only of interest for the originator of the request, it is also appropriate to send it on a queue; two different queues will be needed for requests and responses. If the response is actually a general status update which could be of interest to multiple parties the appropriate mode will be topic. The same is the case when a client wishes to communicate a state change which is not the result of a specific request; this situation could occur, for example, when the client is an Element Manager detecting changes that happen on network elements, or even new network elements that are deployed in the network.

Figure C-1 shows two JMS clients exchanging messages through two queues, one for each direction of traffic.

Figure C-1 MOM, JMS Provider and JMS Clients



Queues and topics are generally created and managed by administration of the MOM and JMS provider software, not by client-server interactions. Clients connect to destinations which already exist through administrative configuration of the infrastructure by making requests to a JMS connection factory to create connection objects, which will be different for queues and topics. The client will find the connection factory and the destination by looking them up using a JNDI naming service.

JMS Listener and Sender Modules

From V5.1 HP Service Activator includes JMS listener and sender workflow manager modules which make it possible for an HP Service Activator-based system to act as a JMS client, i.e. to receive and/or send messages over a JMS interface.

The communicating parties on HP Service Activator will be workflow jobs. A running job can put a message to the JMS service by executing the SendMessage node with a parameter specifying the sender module. The listener module can start a workflow job to process each received message.

Some administrative facts about a JMS provider must be known to its clients. These facts must be known to the JMS listener and sender modules as configuration parameters.

Before a client can access a destination, it must connect to it. To do that, it must look up the destination through a JNDI naming server. The basic installation specific administrative facts that any communicating client must know are: the host name/address and port number where the JNDI server provides the lookup service, the name of the initial context class that must be instantiated to use the JNDI server, the name of the connection factory of the JMS server, and the name(s) of the destination(s) it will use. Finally, depending on its configuration, the JMS provider may demand that the client authenticate the request to connect to a destination by supplying username and password.

Additional parameters of the JMS listener and sender modules control the local behaviour of the modules, internally and vis-a-vis communicating workflow jobs. These parameters are not related to JMS as such. You will find them described in *HP Service Activator, Workflows and the Workflow Manager*, in the sections about these modules.

What is sent as a message from a workflow using the JMS sender module is conveyed as the body of a JMS message. There are currently no means to control and use JMS message header fields (JMSDeliveryMode, JMSMessageID, JMSTimestamp, JMSCorrelationID, JMSReplyTo, JMSRedelivered, JMSType, JMSExpiration, JMSPriority, or message properties). Likewise it is only the body of a received JMS message which is passed to the workflow job that will process the message.

Durable Topics

Queues are always store-and-forward. The MOM is expected to retain messages until they are consumed. The receiving client is not requried to listen for messages at all times. If it is inactive for a period and then reconnects, it will receive messages that have been sent in the meantime.

With topics, in general, listeners only receive the messages that are published while the listener is connected, but it is possible to create a durable topic subscription, thereby requesting of the JMS provider that when the listening client temporarily disconnects, published messages shall be retained and delivered when it reconnects. A durable topic subscription must be identified by a unique identifier supplied by the client. A durable subscription will exist until it is explicitly unsubscribed by the client.

Using JBossMQ as MOM

If you need to integrate HP Service Activator with another application that also supports JMS, and the capabilities of JMS are appropriate for the integration, but there is no suitable MOM in the customer's environment, then it is possible to use JBossMQ which is available as a part of the JBoss application server that HP Service Activator is running on.

To accomplish this, you will need to configure JBossMQ to support the necessary destinations with the appropriate security (roles, user names and password). The topic managing JBossMQ is beyond the scope of this manual. You must find and read the applicable JBoss documentation. As a hint, look in the directory *$JBOSS_DEPLOY*/jms, where the file jbossmq-destinations-service.xml is used to configure the MBeans which implement destinations and the file oracle-jdbc-state-service.xml is used to configure usernames, passwords and roles for clients.