Oracle® Communications Service Catalog and Design
Concepts

Release 8.1

F96231-02

October 2024

ORACLE®

Contents

1   About Service Catalog and Design

2   About Solution Designer

3   About Design Studio

4   Working with Projects, Data Schemas, and Data Elements

5   Working with Design Patterns and Guided Assistance

ORACLE

6   Working with Conceptual Models

7   Design Studio Packaging and Integrated Cartridge Deployment

8   Extending Reference Implementations

A   Solution Development Methodology

Glossary

Preface

This guide provides a conceptual introduction to Oracle Communications Service Catalog and Design and it includes concepts related to the modeling and configuration of Oracle Communications products, and to Service Catalog and Design as an integrated design environment.

For detailed steps on how to perform specific tasks in Solution Designer, see *Solution Designer User's Guide*.

For detailed steps on how to perform specific tasks in Design Studio, see the Design Studio Help and *Design Studio Developer's Guide*.

Audience

This guide is intended for business analysts, architects, development managers, developers, and designers who are responsible for system integration or solution development involving the Oracle Communications operational support systems applications.

Ideally, you should be knowledgeable about your company's business processes, the resources you need to model, and any products or services your company offers.

Documentation Accessibility

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at http://www.oracle.com/pls/topic/lookup?ctx=acc&id=docacc.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=info or visit http://www.oracle.com/pls/topic/lookup?ctx=acc&id=trs if you are hearing impaired.

Diversity and Inclusion

Oracle is fully committed to diversity and inclusion. Oracle respects and values having a diverse workforce that increases thought leadership and innovation. As part of our initiative to build a more inclusive culture that positively impacts our employees, customers, and partners, we are working to remove insensitive terms from our products and documentation. We are also mindful of the necessity to maintain compatibility with our customers' existing technologies and the need to ensure continuity of service as Oracle's offerings and industry standards evolve. Because of these technical constraints, our effort to remove insensitive terms is ongoing and will take time and external cooperation.

1
About Service Catalog and Design

This chapter provides an overview of Oracle Communications Service Catalog and Design that comprises Solution Designer and Design Studio.

Introducing Service Catalog and Design

Oracle Communications Service Catalog and Design is an application agnostic TeleManagement Forum Open Digital Architecture (TMF ODA) aligned service and resource catalog driving end-to-end solutions. The primary focus is to have a no code and low code design environment for configuration of the orchestration, inventory and assurance solutions and individual applications. It requires no application expertise, so you can rapidly define services and technical building blocks without extensive application training. It simplifies the solution evolution by enabling you to update the solution design and to release updates that meet the evolving service requirements.

Service Catalog and Design comprises the following components:

- Solution Designer: Solution Designer runs as a cloud native application in a containerized and orchestrated deployment architecture. Solution Designer provides a unified environment for designing, configuring, and deploying a service, where all components, specifications, and configurations are consolidated. Service Catalog and Design provides journey and persona driven user experience for defining service and resource catalog for business solutions. For more information on Solution Designer and its capabilities, see *Solution Designer User's Guide*.

- Design Studio: Design Studio is an integrated GUI tool based on the Eclipse IDE. This enables designers and developers to use the fully-featured Java IDE capabilities to further enhance, extend or integrate the solution business logic. This design-time environment enables you to build and configure Oracle service fulfillment and network and resource management solutions. For more information on Design Studio and its capabilities, see "About Design Studio ".

Functional Architecture of Service Catalog and Design

Figure 1-1 shows the functional architecture of Service Catalog and Design. Service Catalog and Design provides all the functionality in the architecture using the applications such as Workspaces, Specifications, Initiatives, PSR Models, Data Elements, and Domains.

Figure 1-1   Functional Architecture

Figure 1-1   Functional Architecture

2
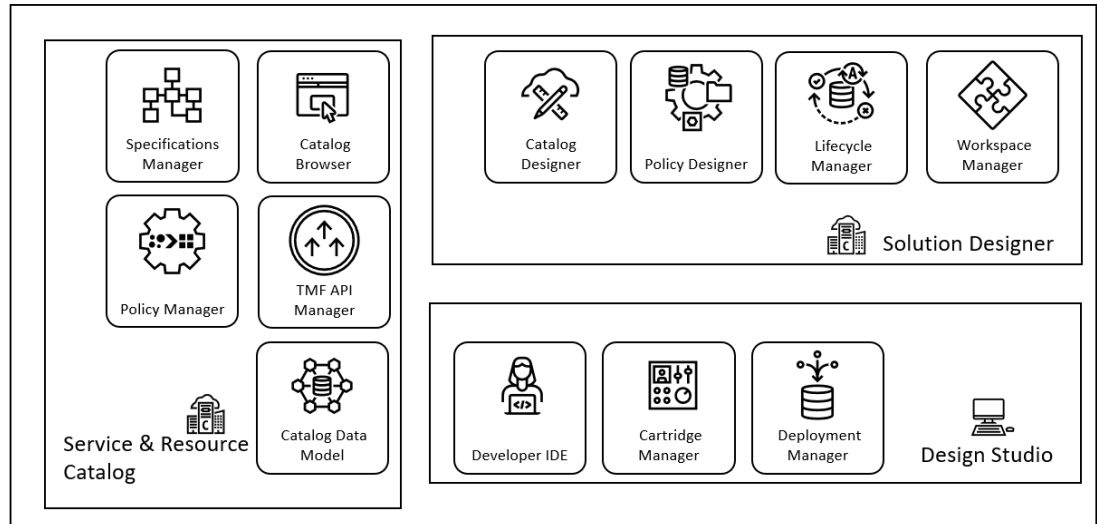
About Solution Designer

Solution Designer is a design tool for business users and domain experts from communications service providers. It unifies and accelerates the creation and delivery of services across Oracle Communications and minimizes the cost of ownership for operators and systems integrators. It simplifies the management and maintenance of services and networks by centralizing service, resource, and network specifications and configurations.

Introducing Solution Designer

Solution Designer enables you to model services and resources, and the interfaces between them, that make up a communications service and network solution. It enables you to create and use solutions quickly by providing a consistent design experience. It provides user journey and persona based design time user experience. It brings state-of-the-art, consumer grade user experiences across devices to the sophisticated enterprise scenarios that Oracle enables.

Solution Designer runs as a cloud native application in a containerized and orchestrated deployment architecture. It provides a unified design platform designed for business users and domain experts from communications service providers. It unifies and accelerates the creation and delivery of services across Oracle Communications and minimizes the cost of ownership for operators and systems integrators.

You can also use Solution Designer to maintain solutions and to change them over time. For example, you can quickly change your solution based on ongoing responses from customers, changes in technology, and market analysis. You use it to configure solutions at all levels of solution maturity, and over the lifetime of a solution. As requirements change, and as your communications services evolve, Solution Designer enables you to evolve your solutions.

An example solution design user journey is as follows:

1. Service catalog administrator creates the initiative and the domains.

2. Inventory specialist creates the data elements and the resource specifications.

3. Network specialist creates the technology model.

4. Service specialist creates the service model.

5. Service catalog administrator publishes the initiative. After all the design is completed, the service catalog administrator moves it through various transitions to release it to production. As the initiative moved to testing stage it interacts with DevOps engine to build UIM cartridges for deployment. DevOps engine also generates Design Studio workspaces that developers can use to build solution cartridges for OSM and ASAP.

6. After entire solution is tested and approved, the service catalog administrator releases the initiative to production for deployment.

For more information on Solution Designer and how to work with the user interfaces, see *Solution Designer User's Guide*.

About Solution Designer Applications

Solution Designer is a micro-services based cloud native application. You can manage the following applications in Solution Designer:

- **Initiatives**: Solution development process that contain a set of capabilities that needs to be delivered in a given phase of OSS transformation.

- **Domains**: Logical grouping or category that represents a specific area or type of service within the telecommunications industry.

- **Data Elements**: Define the attributes and properties of services and resources.

- **PSR Models**: Describe how customer services and technology specific RFSs are designed and implemented.

- **Specifications**: Define your services including CFSs, RFSs, resources, and locations

- **Workspaces**: Publish your initiatives for testing the design and launching the initiative to production.

3
About Design Studio

Oracle Communications Service Catalog and Design - Design Studio is an integrated design environment (IDE). This design-time environment enables you to build and configure Oracle service fulfillment and network and resource management solutions. Design Studio enables you to create and use solutions quickly by providing a consistent design experience for both technical and non-technical users.

Introducing Design Studio

Design Studio is an integrated design environment (IDE). This design-time environment enables you to build and configure Oracle service fulfillment and network and resource management solutions. Design Studio enables you to create and use solutions quickly by providing a consistent design experience for both technical and non-technical users.

About Design Studio Solutions

You use Design Studio to design the run-time behavior of Oracle Communications applications. For example, if you need to provision broadband service orders, you can use Oracle Communications Order and Service Management (OSM) to manage the service orders in real-time (you also need a CRM system to capture the order, an inventory system to create the service, an activation system to activate the service in the network, and so forth). Before you can do any real-time order management, however, you first use Design Studio to design the OSM run-time process, to customize the content of the orders, to customize the tasks, to define the data required to fulfill the order, and so forth.

In Design Studio, you package this collection of OSM entities and configurations in an OSM Cartridge project. You create additional Cartridge projects to define other parts of your business processes (for example, Inventory Cartridge projects that customize your design and assign run-time processes, Activation Cartridge projects that contain configuration to activate the service in the network, and so forth). You create other types of projects to augment the data defined in your Cartridge projects, such as Model projects that contain data that you define once and use across multiple applications, and Environment projects that contain information about your run-time servers.

This collection of projects is called a solution.

You package each Cartridge project into an archive file (the archive file is called a cartridge), which you deploy from Design Studio to core applications that are integrated into a run-time environment. When deployed to a run-time environment, cartridges are interpreted by the application server, which then activates the solution. You can undeploy cartridges and deactivate them by removing them from the run-time environment, and you can replace cartridges that exist in the run-time environment to change a solution. In OSM and ASAP, you can undeploy the cartridges or replace them to change a solution. In UIM, you can only replace the cartridges to change a solution. Together, the core applications and the deployed cartridges comprise the solution operations environment.

When creating new solutions in Design Studio, you can use productized cartridges (cartridges purchased from Oracle) or reference implementations as starting points to your solution. Also, you can develop your own solutions.

While solutions can include only a single application, typically they leverage multiple, integrated applications. When you design solutions, you can model and configure entities that cross the boundaries of any one specific application. The Design Studio environment integrates the applications used in the solution for a consistent and efficient design experience. During solution design, you use Design Studio to iteratively design, test, and debug solutions.

About Design Studio Roles

Team members may play many roles during solution development. Table 3-1 lists the roles and the tasks each role typically performs in Design Studio.

Table 3-1   Design Studio Roles and Tasks

Role	Tasks
Data Modeler	Designs the data types and structures necessary to support a cartridge or solution.
Cartridge Designer	Designs deployable components spanning a single product domain.
Solution Designer	Assembles collections of cartridges to deliver a solution that includes multiple Oracle Communications applications. Additionally, this role may design additional cartridges to achieve the desired solution functions.
Developer	Builds the code to support the metadata-driven components of a cartridge. This role may be an expert in development languages such as Java, XPath, XQuery, or SQL.
Release Engineer	Manages the solution development and test environment. This role configures and automates the solution build and the solution testing.
Cartridge and Solution Tester	Deploys cartridge archives to a test environment to certify that the cartridge or solution is working as intended.
System Administrator	Manages the design environment software. This role acquires, configures, and distributes Design Studio to cartridge designers.
Design Studio Report Designer	Develops custom reports. This role is a highly-skilled developer familiar with the technologies needed to produce report designs. These technologies include XML, XPath, BIRT, JavaScript, and familiarity with Design Studio cartridge development.

Working with Design Studio for Oracle Communications Applications

In addition to configuring common design-time models that apply across an entire solution, you use Design Studio to configure specific design artifacts for one or more Oracle Communications applications:

- Design Studio for OSM and Design Studio for OSM Orchestration, which you use to define solutions for OSS service order management and for BSS central order management, respectively.

- Design Studio for Inventory, which you use to define service and resource definitions, rules and domain-specific metadata.

- Design Studio for Network Integrity, which you use to configure network discovery, assimilation, and reconciliation behavior.

- Design Studio for ASAP, which you use to define service actions, network actions and scripts for service activation.

About Design Studio for OSM

You use Design Studio for OSM to configure and deploy OSM solutions for order orchestration and for order provisioning. Design Studio for OSM enables you to design process flows to fulfill requests for orders, such as customer, service, and technical orders.

Order provisioning functionality enables you to configure and deploy service provisioning cartridges that include integrations with Oracle Communications Unified Inventory Management (UIM), Oracle Communications ASAP, and Oracle Communications IPSA, as well as with third-party applications. Solution integration with UIM and ASAP is facilitated by Design Studio through design-time integration capabilities.

About Design Studio for Inventory

You use Design Studio for Inventory to define the configuration of services and to assign resources to them. Service configuration includes specifications, characteristics, rules, equipment models, capacity models, and component packaging. You define the metadata needed to configure services and map the services to logical and physical resources.

About Design Studio for ASAP

You use Design Studio for ASAP to model services such as voice services (including wireless, voice over IP), data services (including digital subscriber line, IPTV), and other services that require controlled and coordinated activation in the network.

As part of the service fulfilment process, Design Studio for ASAP provides a set of technical actions that you can implement against network and IT resources. You assemble the technical actions into order handling process flows (using OSM), and map the attributes and data from orders (at the service level) to specific resources.

The Design Studio Activation task supports IP Service Activator. For IP Service Activator, OSM data is transformed to a web service order that is sent to IP Service Activator to activate the specified services. You can create workflows that integrate IP Service Activator data and activation actions with other tools and systems.

About Design Studio for Network Integrity

You use Design Studio for Network Integrity to maintain the data integrity of telecommunications data sources. Using Design Studio for Network Integrity, you can connect to devices in the network (such as EMS, NMS, and other systems) to retrieve data; make changes to that discovered data; import inventory data back into the model; create rules to compare data and identify discrepancies; and integrate with external systems to update data and resolve any discrepancies.

About the Design Studio Role in Business Solutions

Design Studio is an integral component in the life cycle of the following business solutions:

- **Rapid Offer Design and Order Delivery (RODOD):** In this solution, you use Design Studio to configure the fulfillment flow definitions in OSM to support the central order management function. You synchronize sales catalogs in Siebel CRM with OSM by importing sales catalog elements into Design Studio and mapping them to fulfillment flow

functions. Additionally, you can configure the metadata and policies required to assist the functions of dynamic decomposition and orchestration plan generation.

- **Service and Network Orchestration (SNO)**: In this solution, you use Design Studio to define an overall solution model representing a high-level abstraction of service fulfillment behavior. This model can then be used to automatically realize a coordinated set of application-specific configurations. This allows a consistent set of configurations and interfaces to be generated for OSM, UIM, and ASAP. For more information about SNO, see the SNO Solution page from Oracle Learning Library:

  www.oracle.com/goto/orchestration

- **Network and Resource Management (NRM)**: In this solution, you use Design Studio to create, manage, and extend cartridges for each domain, or to modify and extend prebuilt definitions in UIM to define specifications, characteristics, and rulesets. Additionally, you can extend or modify the definitions to support the logic and protocol that enables Oracle Communications Network Integrity to discover, assimilate, and reconcile network configuration data in inventory systems.

About Design Studio Product Architecture

Design Studio builds on an extensible Eclipse platform to facilitate the creation of service fulfillment and network and resource management solutions. Eclipse provides a vendor-neutral open development platform comprised of extensible frameworks and tools for building and deploying software. See the Eclipse website for more information about the Eclipse platform.

Working with the Design Studio User Interface

The Design Studio interface includes a number of components to assist you with configuration. Interface components include workspaces, a workbench, perspectives, views, editors, menus, and toolbars.

About Workspaces

In Design Studio, the workspace contains a collection of your projects whereas in Solution Designer, a workspace is an interface that interacts with the DevOps engine to generate the cartridges. A Design Studio workspace appears as a directory on your local machine and it is where Design Studio saves your resources. Your projects exist independent of a workspace, but can only be used in a workspace. The workspace directory root is created on your local machine when you create a Design Studio workspace. The root exists as long as the workspace exists. You can create more than one workspace, but the only one workspace can be open at a time. Using multiple workspaces can help you organize complex projects. For example, you can configure the perspectives in your workspaces to display a specific set of views for specific projects. Or, you can use one workspace to test projects.

When you start Design Studio, you identify a workspace in which you want to work. You can switch to a different workspace, when necessary. Design Studio will automatically close down and restart using the new workspace.

About the Workbench

The Design Studio workbench is a set of tools you can use to navigate within and manipulate the workspace, access functionality, and edit content and properties of resources.

About Perspectives

Perspectives are collections of views, menus, and toolbars that appear in a specific layout, and they determine how information appears in the workbench. The Design Studio provides two predefined perspectives that work together with Eclipse and third-party perspectives that are used for implementation, debugging, builds, and version control.

Perspectives are task oriented. For example, you use the Design perspective to model the entities in your cartridges. You use the Environment perspective to create and manage the attributes associated with your environment, and to deploy and undeploy cartridges to run-time environments.

Each perspective contains a default set of views and editors, which you can customize. The views automatically included in the Design perspective assist you with cartridge modeling. You can also create your own perspective.

About Views

A view provides access to a specific set of functions, available through the view's toolbars and context menus. For example, the Problem view displays errors that exist in the model entities, so you use the Problem view to locate and resolve entity errors. You use the Dictionary view to model and review data in your workspace. The Dictionary view and Problem view each provide access to a different set of Design Studio functions.

A view can appear by itself, or it can be stacked with other views. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the workbench window.

Figure 3-1   Design Studio Views
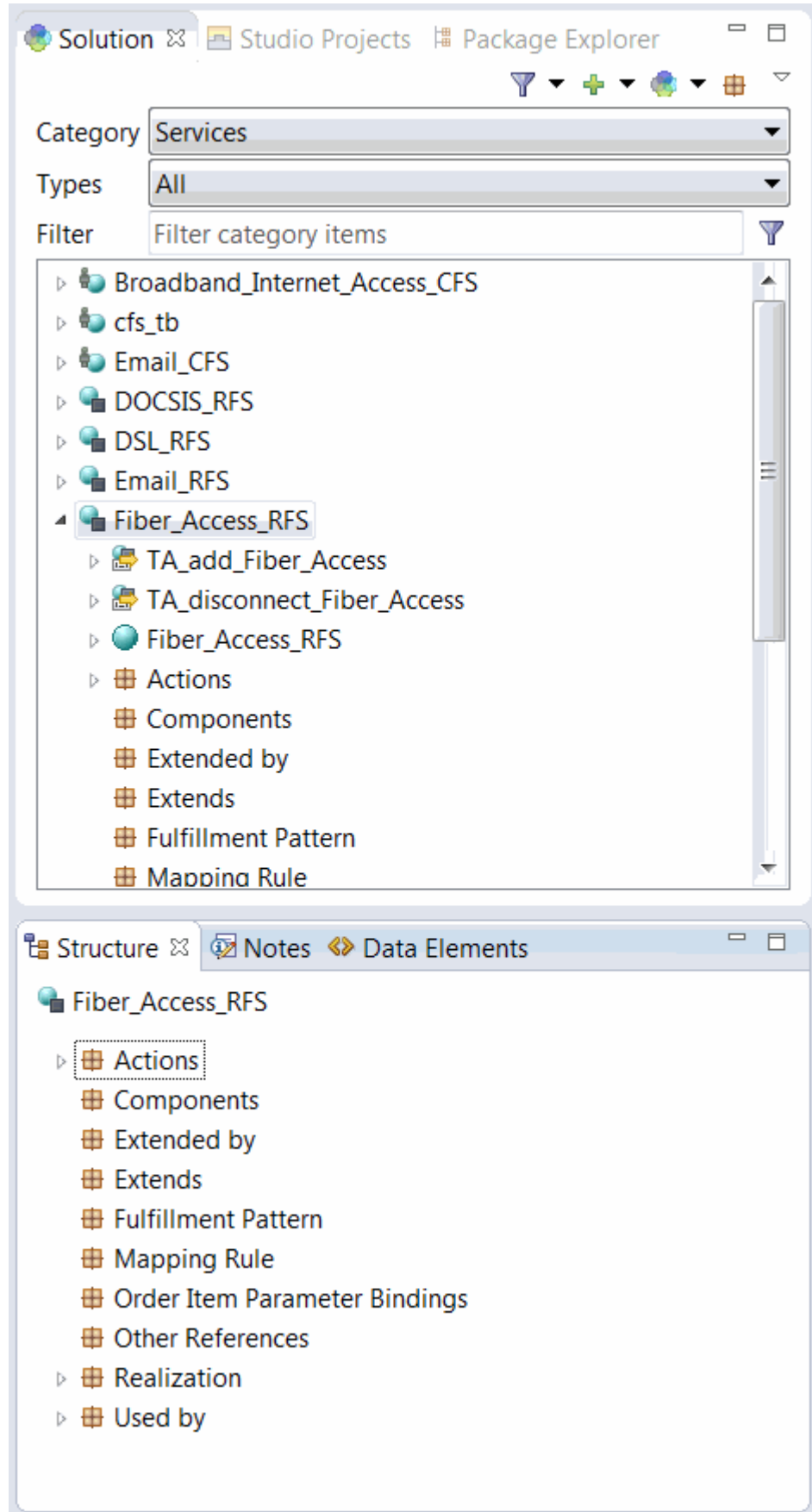
About Editors

An editor is a special type of view that enables you to edit data, define parameters, and configure settings. Editors contain menus and toolbars specific to that editor and can remain open across different perspectives.

Design Studio editors are associated with entities. Many Design Studio views enable you to double-click entities to open the entity in the associated editor. For example, you can double-click an entity in the Solution, Studio Projects, Data Elements, and Problems views (among others) to open the associated editor. Additionally, you can double-click table entries that reference entities to open the entity in the associated editor.

For example, if you double-click a Process entity in the Studio Projects view, the Process editor opens. You can create diagrams in the Process editor to illuminate patterns and identify inefficiencies in tasks and processes. Process editor shapes, colors, and presentation can communicate information about the flows and processes.

If you double-click a Specification entity in the Studio Projects view, the Specification editor opens. You can use the Specification editor to define properties of business entities. The Process editor and the Specification editor are just two of the editors available in Design Studio.

Design Studio supports drag-and-drop functionality, enabling you to drag files or entities from the Studio Projects view to editors. Additionally, you can open an editor associated with an entity by dragging the entity from a view into the editor area. An asterisk in the editor title bar indicates that the changes you made in an editor are unsaved.

Navigating Across Solutions Using the Design Perspective

The Design perspective includes a set of linked views that you can use as an entry point to your domain solutions and to view the relationships among the entities and elements in a solution.

Team members of varying skills and at varying points in the solution design can leverage the functionality of the views in the Design perspective. For example:

- Business users and architects who define the underlying concepts of a solution can use these views to navigate across conceptual models without needing to drill down to application-specific details.

- Developers can use these views to begin designing application-specific configurations, using the high-level conceptual model as a starting point.

- Business users can use these views (after a solution is designed and implemented) to reassemble existing capabilities and map them to new product offers for customers. From the Solution view, technical users can import product specifications from a customer catalog and map the product specifications to existing customer facing service specifications.

About the Design Perspective Default Layout

The default layout of the Design perspective enables you to navigate across different views while keeping critical information visible. The default layout includes the Solution, Structure, Data Elements, Notes, and Outline views, among others. These views are linked together, such that when you make a selection in one view or editor, Design Studio updates the content in related views based on the active selection.

For example, the Solution view enables you to organize and view your solutions through products, services, resources, actions, orders and other categories of entities that implement a solution. When you filter this view for a specific category of entity (for example, for all orders defined in the workspace), the view displays all of those types of entities in the workspace and includes child folders that represent all relationships to the entities. When you make a selection in the Solution view, Design Studio updates the content in the Structure, Notes, and Data Elements views to reflect the relationships and content of the selected entity.

Figure 3-1 illustrates how making a selection in the Solution view updates the content in the views linked to the Solution view (in this example, the Structure view).

About Context Menu Options

You can right-click in Design Studio views to open the context menu, which enables you to perform operations in views and to implement data model changes across a solution. You can right-click in the Solution view to add new entities, delete existing entities, or refactor a selected entity. You can maintain focus on a specific entity in the Solution view and perform many of the same operations in the linked views.

For example, you can highlight an Order entity in the Solution view and rename, change the location of, copy, or move the data elements included in the Order entity in the Data Element view. See the Design Studio Help for more information about the Design Studio context menu and about refactoring entities and data elements.

Example: Navigating Across a Solution Design

The following procedure is an example of how you can navigate across a solution using the Solution view and the related, linked views.

1. From the Studio menu select **Show Design Perspective**.

2. Click the **Solution** tab.

   The Solution view appears.

3. In the **Category** field, select a type of entity for which to filter the view.

   For example, Select **Orders** if you want to view all orders in the workspace and navigate among the entities related to orders, such as tasks, processes, permissions, and so forth.

   When you select a category, the Solution view displays all entities of associated with the category type.

4. In the Solution view tool bar, ensure that the **Show/Hide Folders** button is enabled.

5. Select an entity, and expand the root folder.

   The view displays relationships among entities by displaying child folders that represent relationships to the entity. For some relationships, these folders will appear even if the relationships are not yet established (the folder will be empty).

6. Navigate through the hierarchy by expanding the nested folders until you reach the content you need.

7. Add new entities, delete existing entities, or refactor entities in the Solution view.

   By right-clicking and opening the context menu, Design Studio enables you to perform operations in views and to propagate data model changes across a solution without sacrificing model integrity. See the Design Studio Help for more information about the context menu and about refactoring entities and data elements.

8. Click the **Notes** tab.

Use the Notes view to provide documentation for the entity or data element selected in the Solution view. You can annotate entities and data elements when you want to communicate to other team members information about the solution.

9. Click the **Structure** tab.

   Use the Structure view to view relationships for a selected entity or data element and perform operations on the relationship contents. The Structure view enables you to navigate through and to perform operations on relationship folder contents while maintaining focus on the active entity in the Solution view.

10. Click the **Data Elements** tab.

    Use the Data Element view to view the simple and structured data elements for the entity selected in the Solution view and to perform operations on those data elements.

    The Data Element view enables you to navigate through and to perform operations on relationship folder contents while maintaining focus on the active entity.

11. In the Solution view, double-click the selected entity to open it in an editor.

    You can edit the data associated with the entity, define parameters, and configure settings for the entity in the entity editor.

12. Click the **Outline** tab.

    Use the Outline view to view the relationships and to perform operations applicable to the entity associated with the active editor (this view is linked to the active editor).

    For example, you can view data elements associated with the entity and perform applicable refactoring operations in the Outline view while keeping the entity open in the editor.

    See "Navigating Across Solutions Using the Design Perspective" for more information.

About Design Studio Reporting

Design Studio enables you to generate reports that include information about an implemented solution. For example, a report can summarize the structure of the solution by listing projects and dependencies, or a report can summarize the composition of a service. Reports can capture the names, types, descriptions, and relationships of projects, entities, and data elements.

Design Studio reports can be used by team members who have not installed Design Studio or who require Design Studio configuration data in document form. Design Studio reporting facilitates information sharing, solution development, and model review tasks.

Design Studio includes reference reports that provide a foundational set of capabilities. You can use these reports as is or as a starting point for customizing your own reports. For example, you can customize the report designs for content, layout, or branding.

You can also develop your own report designs using the Eclipse Business Intelligence and Reporting Tools (BIRT) feature. See *Design Studio Developer's Guide* for more information about customizing reports.

System administrators can integrate report generation into an automated build system to automatically generate reports that you can reference when developing solutions. See *Design Studio System Administrator's Guide* for more information.

4

Working with Projects, Data Schemas, and Data Elements

This chapter provides information about Design Studio projects, data schemas, and data elements. Also, it provides high-level information about modeling data elements.

Working with Projects

Design Studio projects contain artifacts (entities, data, rules, code, and so forth) that you use to model and deploy Design Studio cartridges. Your solution uses various types of projects. For example, you use projects to build cartridges that can be deployed to a server, for version management, for sharing, and for resource organization.

You can create various types of projects and you can extend cartridges that you purchase with your own projects. Oracle Communications supports a library of extensible cartridges that are fully compatible with Design Studio and provide a basis from which to assemble solutions.

The most common types of projects you use in Design Studio are:

- Model projects, which contain data common to multiple cartridge projects.

- Cartridge projects, which contain collections of entities and supporting artifacts that represent a cartridge deployed to a run-time environment.

- Environment projects, which you use to manage attributes associated with your run-time environments.

In each project, you can model the data necessary to achieve your solution (and share that data across all projects in the workspace) build and package the projects, and test them in run-time environments.

Application integration and cross-product modeling and data sharing reduce the effort and time to deploy solutions. Design Studio supports the design-time configuration for integrated (or standalone) service fulfillment solutions and for network resource management solutions.

Importing Projects

One way to start working in Design Studio is by importing domain-specific and vendor cartridges into Design Studio and using these cartridges as the foundation for your new solutions. For example, if you have obtained cartridges from Oracle, you can import them into Design Studio and reuse their components to create your own cartridge projects.

There are multiple methods for importing data into Design Studio. The methods depend on your own preferences and on the applications with which you are working. For example, Oracle Communications ASAP enables you to import data directly from an existing ASAP run-time environment, and Oracle Communications Order and Service Management (OSM) enables you to import existing XML data models. When importing data for Design Studio application plug-ins, it may be necessary to refer to the Help specific to the application for more information.

> **Note:**
>
> Importing from ASAP run-time environments and from OSM XML models is intended for migration purposes only.
>
> When importing Design Studio for Inventory cartridge projects that include changes to tool tips or characteristic display names, you must redeploy the Oracle Communications Unified Inventory Management (UIM) application server when you deploy the cartridge that contains the changes.

When you import a cartridge, it becomes a project in the current workspace. Some cartridges are sealed, meaning that they are read-only. Sealed cartridges cannot be modified without first being unsealed.

Some projects reference entities in other projects, and these references create dependencies among the projects. If you import a cartridge project that has dependencies on other cartridge projects that are not in the current workspace, Design Studio displays an error. Import all dependent cartridges, then clean all projects to remove the errors. See "Working with Project Dependencies" for more information.

Upgrading Projects

When working in a new version of Design Studio, you must upgrade projects from the previous Design Studio version to the latest Design Studio version. Additionally, you must obtain and import the latest versions of all sealed productized versions if you want to use the cartridge in the updated version. During the upgrade process, Design Studio automatically detects old project versions and completes all necessary project upgrades in the workspace.

See *Design Studio Installation Guide* for more information about upgrading projects.

Controlling Project Visibility in a Workspace

Design Studio solutions can contain large numbers of productized, sealed, and application-specific projects that are not directly related to your work. You control which projects appear in your workspace by creating and applying a filter, called a working set.

The projects that a working set filters out exist in the workspace and remain open but do not appear in the Studio Projects view. Also, you can define a separate working set for the Solution view to control visibility of project entities at the root level.

For example, you can facilitate design modeling and workspace navigation by creating a working set that displays only those projects related to your present design work. The working set can filter projects based on the project type and based on a tag that you associate with a project. Working set filters are limited to the workspace in which they are defined.

One working set is delivered with Design Studio and is applied to the Studio Projects view when you install Design Studio. This working set is named **Exclude Base Projects**, and it excludes from display all Design Studio base projects, any projects associated with the **Base Project** tag, and all non-Design Studio projects (such as Eclipse projects and Java projects). The **Exclude Base Projects** working set is not editable, but you can deactivate this working set filter. See the Design Studio Help for information about deselecting a working set and for more information about controlling which projects appear in your workspace.

Working with Model Projects

Model projects are collections of data elements intended to be referenced by other projects in a workspace to fulfill a solution. Model projects include simple data elements and structured data elements that are not specific to any one Oracle Communications application and that enable you to share that data across a solution.

Entities in Design Studio application projects can reference data in a model project. You can model these data elements once and then configure different applications to reference these data elements, as appropriate. This modeling strategy enables you to identify which entities in a solution share the data elements in the solution.

For example, an Activation project entity, an OSM project entity, and an Inventory project entity can all share the same data elements (for example, the data elements Caller ID, Call Waiting, and Call Forwarding) created and saved in a model project.

Working with Cartridge Projects

Cartridge projects contain collections of entities that you can deploy to a run-time environment to support your business processes (for example, cartridges deployed to Oracle Communications Order and Service Management run-time environments provision services requested on incoming sales orders). When modeling application-specific entities in Design Studio, you configure all entities in a cartridge project.

Cartridge projects can have design-time dependencies on other cartridge and model projects. During project builds, Design Studio adds the necessary design artifacts from any dependent projects to your cartridge project.

Cartridge projects can be built and packaged into deployable cartridges. A cartridge is a collection of entities packaged into an archive file, which you can deploy to a run-time environment.

You can create the following cartridge projects:

- Activation IPSA project
- Activation project
- Activation SRT project
- Integrity project
- Inventory project
- Order and Service Management Composite project
- Order and Service Management project

See the Design Studio Help for more information about these cartridge projects.

Working with Environment Projects

Environment projects enable you to manage the attributes associated with your run-time environments, including connection attributes, projects ready to be deployed, projects previously deployed, and associated project attributes such as the version and build numbers.

You must create at least one environment project. Environment projects can be shared among all team members who use the same test environment. You work with environment projects and their entities in the Environment perspective.

Working with Project Dependencies

Projects have dependencies on other projects when entities in one project reference entities in another project. These dependencies enable you to share data elements and entities across applications. For example, an OSM project entity might reference the Caller ID, Call Waiting, and Call Forwarding data elements in a model project.

Defining dependencies among projects helps eliminate circular dependencies and duplicate data elements from your data solution, and enables you to better understand how projects are related. Defining project dependencies ensures that all of the data a cartridge project requires is available when you deploy the cartridge project to a run-time environment.

When defining project dependencies, you can specify whether the dependency is required in the Design Studio workspace or whether the dependency is required in the Design Studio workspace and in the target run-time environment.

When you build a project, Design Studio ensures that all entities that are referenced in but defined outside of the project are declared. Design Studio saves the project dependency information and uses this information to validate dependencies at deployment.

> **Note:**
>
> Oracle recommends that you plan relationships between projects and configure project dependencies early in your development cycle.

Project Dependency and Data Modeling

Project dependencies control the data that is available when you model cartridge projects. Design Studio restricts the data that is available (in data selection dialog boxes and in views) to data defined in the project and in dependent projects only. Filtering the data that is available helps prevent circular dependencies and unintended model reuse.

Project Dependency Warnings and Errors

If you configure a cartridge project to reference content in other cartridge projects without declaring project dependencies, Design Studio creates an error or a warning. You control this level of severity, based on how you define the diagnostic level in the Project editor. See the Design Studio Help for more information.

Project Dependency and Cartridge Deployment

The dependencies defined for a project impact the order in which you deploy cartridge projects. You must deploy all dependent cartridges first. For example, if project A depends on project B and project C, and project B depends on project C, when you deploy project A, Design Studio determines that you must deploy projects C and B first (and in that order).

Working with Data Schemas

A data schema is a formal description of a data model, expressed in terms of constraints and data types governing the content of elements and attributes.

You use data schemas when defining products, services, and resources, including the associated actions and the information necessary to perform the processes and tasks for those actions, as well as the interface definitions for integrating between applications.

All data elements are created and saved in data schemas, which can be accessible across all projects in a workspace. Design Studio automatically creates a project-specific data schema when you create a cartridge project (for example, an OSM, Activation, Inventory, or Network Integrity project). You can use this default schema to contain the data you require to model the project, you can create multiple schemas in the same project, or you can create schemas in common projects. You can model your cartridge project using data from any combination of these data schemas.

Model project data schemas include data elements that you want to use across a fulfillment solution. They are product-agnostic; that is, the data elements stored in a model schema are independent of any application project.

```xml
<?xml version="1.0" encoding="utf-8"?>
<xs:schema elementFormDefault="qualified"
    targetNamespace="http://www.oracle.com/ServiceActivation/MobileGSMActivation"
    xmlns="http://www.metasolv.com/ServiceActivation/MobileGSMActivation"
    xmlns:mca-serviceentities="http://www.oracle.com/MCA/ServiceEntities"
    xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:import namespace="http://www.metasolv.com/MCA/ServiceEntities"/>

<xs:element name="C_ACME-GSM_1.0.0_ADD_GSM_SUBSCRIBER">
    <xs:complexType>
        <xs:sequence>
            <xs:element name="TelephoneNumber_TN" type="TelephoneNumber_TNType"/>
          <xs:element name="TelephoneNumber_TNType"
                type="TelephoneNumber_TNTypeType"/>
          <xs:element name="callDisplay" type="callDisplayType"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:simpleType name="callDisplayType">
    <xs:restriction base="xs:string">
        <xs:maxLength value="255"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TelephoneNumber_TNType">
    <xs:restriction base="xs:string">
        <xs:maxLength value="255"/>
    </xs:restriction>
</xs:simpleType>
<xs:simpleType name="TelephoneNumber_TNTypeType">
    <xs:restriction base="xs:string">
        <xs:maxLength value="255"/>
    </xs:restriction>
</xs:simpleType>
</xs:schema>
```

Working with Data Elements

There are two types of data elements, simple data elements and structured data elements. Simple data elements are reusable data types that contain no child dependencies. A simple data element has no structure, and is associated (directly or indirectly) to a primitive data type (Integer, Boolean, String, and so forth). See "About Primitive Data Types" for more information.

Structured data elements are reusable data types that include embedded data types and are containers of simple data elements and other structured data elements. For example, you might create a structured data element called building that contains the floor, room, aisle, rack, and shelf child data elements.

About Primitive Data Types

Design Studio supports primitive types defined by the XML Schema specification.

Table 4-1 lists the full set of supported types and their formats. These types represent the foundation from which all data elements in Design Studio are derived.

Table 4-1   Primitive Data Types Supported in Design Studio

Type	Format
Structure	Hierarchical construct of structures and simple data elements.
Boolean	**true** or **false**
Date	*yyyy-mm-dd* For example: **1971-05-21**
Date Time	*yyyy-mm-dd***T***hh:mm:ss.szzzzzz* where *zzzzzz* represents a time zone, *.s* represents fractional seconds. Fractional seconds and time zone are optional. Use **Z** for Coordinated Universal Time (UTC), or UTC relative form (+ | -) hh':'mm For example: **1971-05-21T14:22:35.2Z, 1971-05-21T14:22:35+05:00**
Time	*hh:mm:ss* For example: **14:22:35**
Integer	Infinite set {..., -2, -1, 0, 1, 2, ...} using decimal digits
Long	Between **-9223372036854775808** and **9223372036854775807**, inclusive
Float **Double** **Heximal**	Lexical representation consisting of a significand and an optional exponent. For example: **-1E4** **1267.43233E12** **12.78e-2** **12**
Hex Binary	*[0-9a-fA-F]* tuples Must have an even number of hexadecimal digits. Hex binary allows length restriction. For example: **A1, 80FF, 18A2C797**
String	Series of UTF-8 characters. String allows length restriction (default **0** to **40**).

About Data Element Icons

Design Studio communicates properties of data elements with associated icons. These icons include problem marker severity, data element multiplicity, and some state details. Data elements that are read-only appear in gray.

> **Note:**
>
> The information conveyed through these icons is also available on the **Details** tab. The problem marker information is available in the Problems view. See Design Studio Help for more information about the **Details** tab fields and Problems view.

Figure 4-1 displays a list of icons and the manner in which they appear with data elements.

Figure 4-1   Design Studio Data Element Icons

Modeling Data Elements

This section describes the features and functionality you use in Design Studio to model solutions for and share data across service fulfillment and network and resource management business solutions.

About the Data Modeling Tabs

Some Design Studio editors include tabs in which you can model information about data elements. These tabs appear in multiple Oracle Communications applications and enable you to configure entities by modeling a data tree to hierarchically represent all associated data elements. These tabs facilitate reusing data elements within a modeling solution and provide tools for locating and using existing data elements.

The tabs consist of a data tree and subtabs. You select a data element in the data tree to review and model details of the selected data element. You open a context menu from the data tree to perform various types of refactoring operations on the data elements.

About the Details and Attributes Tabs

You use the **Details** tab and the **Attributes** tab to define attributes of a data element (such as the name and primitive type), as well as specific constraint values for a data element.

Figure 4-2   Attributes Tab

About the Enumerations Tab

You use the **Enumerations** tab to define sets of valid values for data elements.

Enumerations define values for data elements that are available for selection in a run-time environment. For example, you can define a set of values that appear as lists in the run-time environment.

Figure 4-3   Enumerations Tab
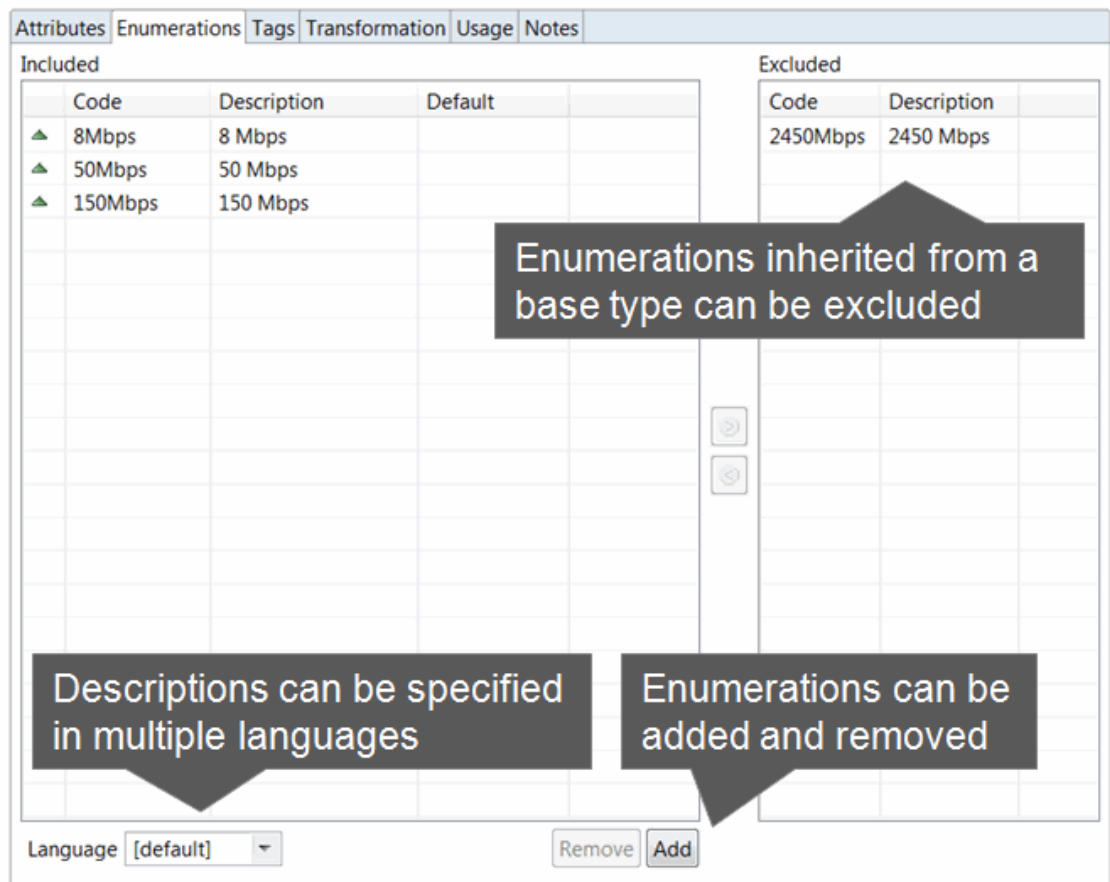
About the Tags Tab

You use the **Tags** tab to characterize data elements by associating the data elements with predefined keywords called tags. Tags help you filter and search for data elements that are associated with specific Oracle Communications applications.

The tags delivered with Design Studio cannot be inherited.

Figure 4-4 Tags Tab
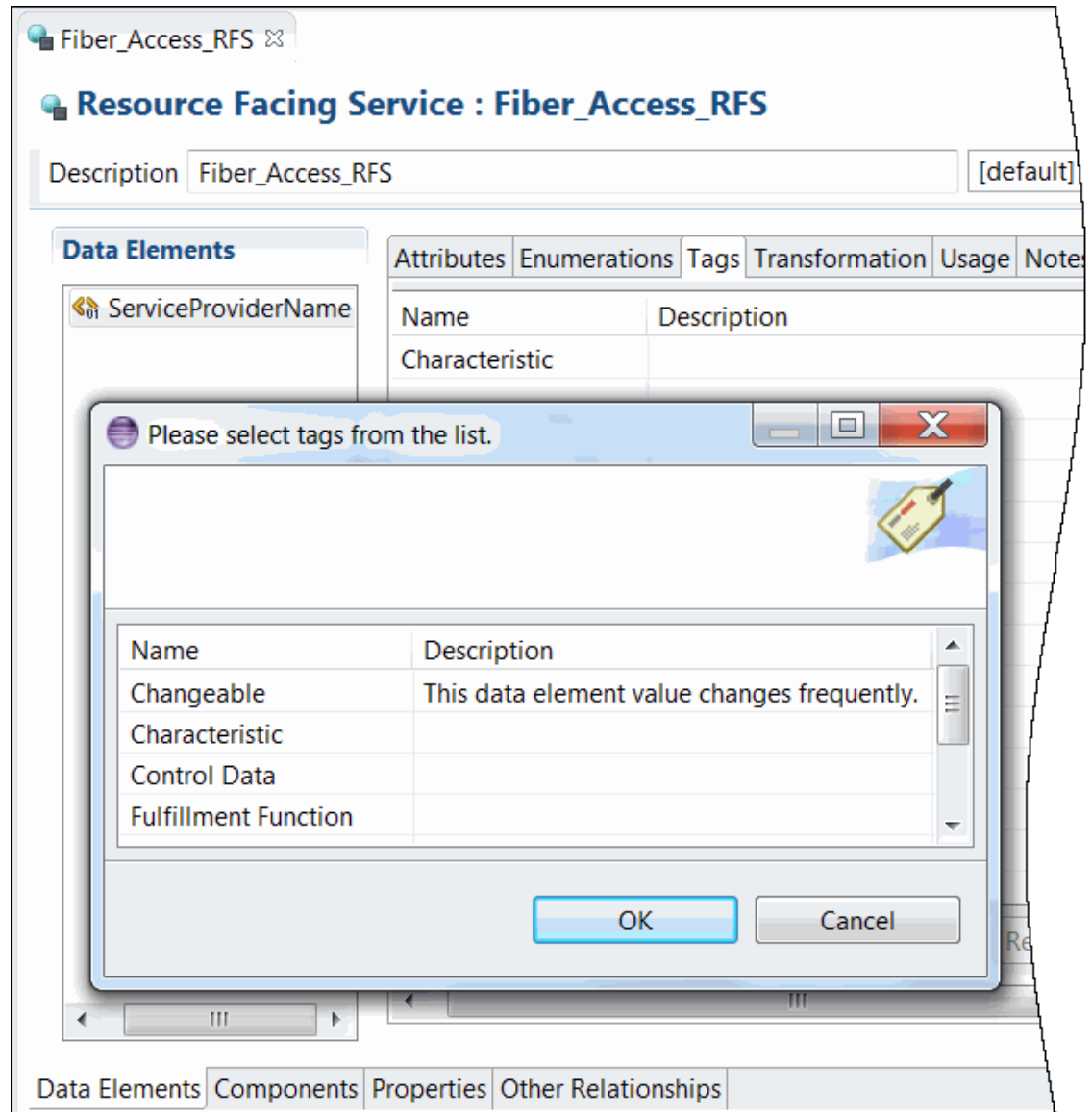
About the Usage Tab

You use the **Usage** tab to review the projects and entities in which a data element is used and to review all references to a specified data element.

About the Notes Tab

Use the **Notes** tab to annotate data elements with descriptions or other applicable information to support the data element.

Design Studio supports multiple languages for this tab. The field at the top of this tab displays your list of languages. If your preferences are set up to work in one language only, the system displays only **[default]**.

Data Element Application Details

You can define for data elements application-specific properties, which you configure using the application-specific data element tabs. For example, you can define information for data elements that are tagged as Inventory characteristics, or you can define behaviors for data elements that will appear in an OSM run-time environment.

Figure 4-5   Application-Specific Tabs
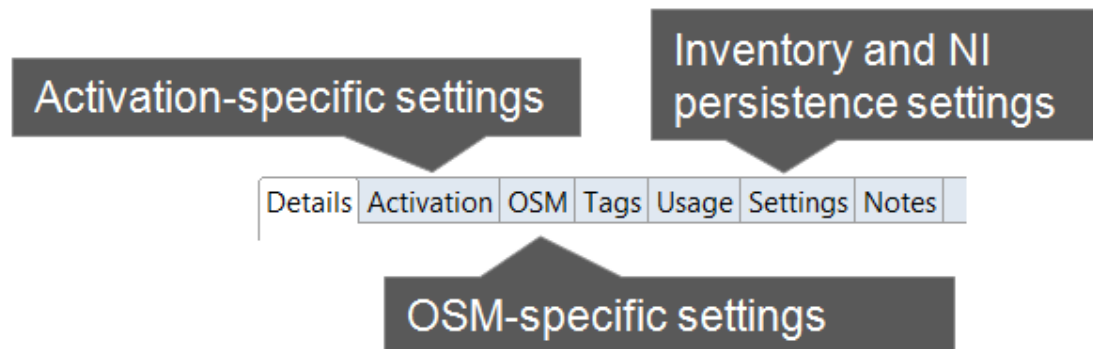
About Data Modeling Strategies and Techniques

Design Studio enables you to:

- Create new data elements that obtain attributes from other data elements. See "Leveraging Information from Existing Data Elements " for more information.

- Use components and features to search for and organize data elements. See "Organizing and Searching for Data Elements" for more information.

- Use the refactoring tools to normalize data models. See "Refactoring Data Models" for more information.

Leveraging Information from Existing Data Elements

To increase modeling efficiency, Design Studio enables you to create new data elements that obtain attributes from other data elements. In Design Studio, you can leverage information using two methods:

- Deriving from a base type element (where the new element automatically obtains the information in the base element

- Extending entities

Deriving from Base Type Elements

Leveraging information already defined for base types enables you to define attributes once, share the common attributes among multiple entities, and edit those entities in a single location. Changes that you make to a base type are automatically saved to all entities that derive from that base type.

For example, if a structured data element **person** contains the **first_name**, **last_name**, and **social_security_number** child elements, you can leverage the information defined for **person**

by using it as a base type for a new structured data element called **employee**. You can add to **employee** the **employee_number**, **hire_date**, and **department** child elements. If you make changes to **person**, Design Studio automatically updates **employee** with the same changes.

You can override some of the information derived from the base type element. When you override these fields, Design Studio does not automatically update that value when the base type value is changed. Rather, Design Studio retains the override value that you defined for the subtype field.

Attributes that you can override include the minimum and maximum number of times the element can appear as a field in a run-time environment, the default value defined for the element, the length defined for string and HexBinary data elements, internal documentation for the element, and so forth.

Figure 4-6   Derived Data Elements
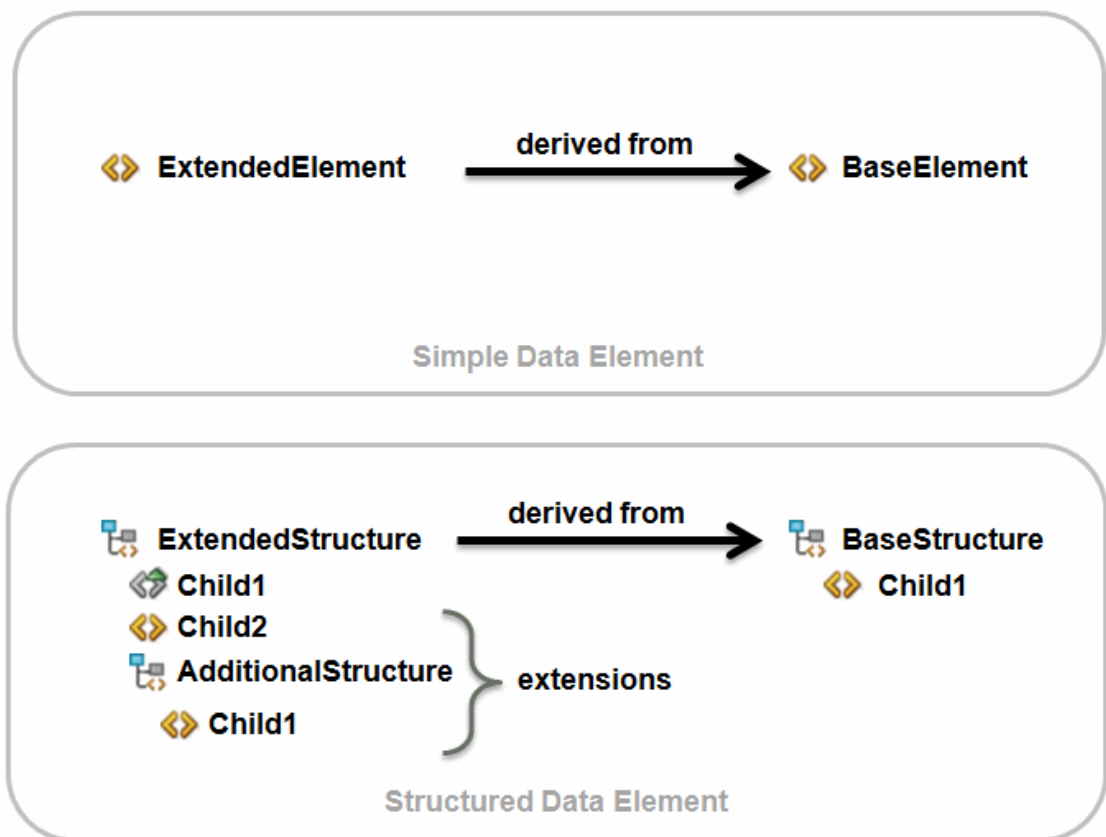
Aliased Data Elements

Typically, derived data elements inherit the name of their base type. If you want to give a derived data element a unique name, you can define an alias for the data element. You can use an alias to give context to the specific use of a data element or to distinguish multiple data elements derived from the same base type.

Figure 4-7   Aliased Data Elements
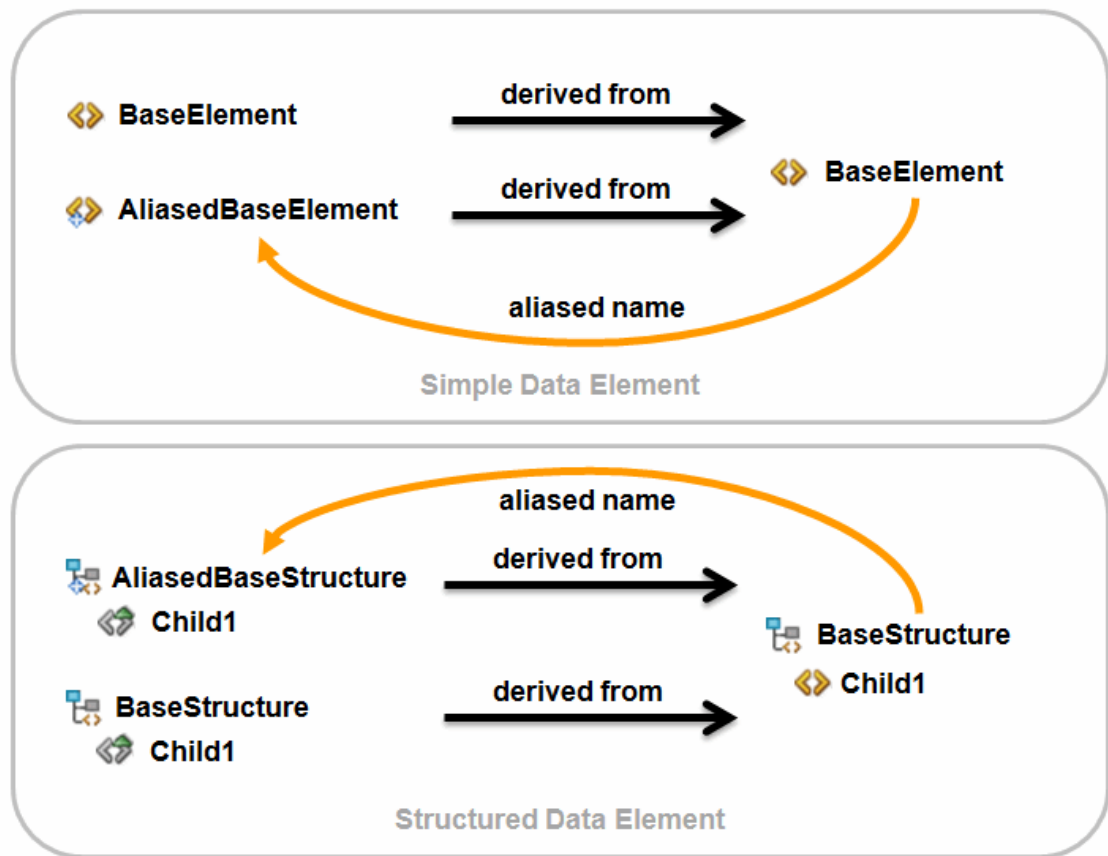
Data Element Recursion

Data element recursion refers to a pattern where a data element repeats in the ancestry of derived data elements. Recursion provides useful patterns for data modeling, and Figure 4-8 demonstrates some of these patterns. When recursion is present, the point of recursion starts an infinite nesting of inherited data elements. Most variants of structured data element recursion are valid in Design Studio.

Simple data element recursion, however, is not valid in Design Studio models because a data element can never resolve to a primitive type. Design Studio validates base type selection to prevent you from defining simple data elements as recursive. If an invalid recursion occurs, Design Studio generates an error marker during the build process.

Figure 4-8   Recursion Patterns
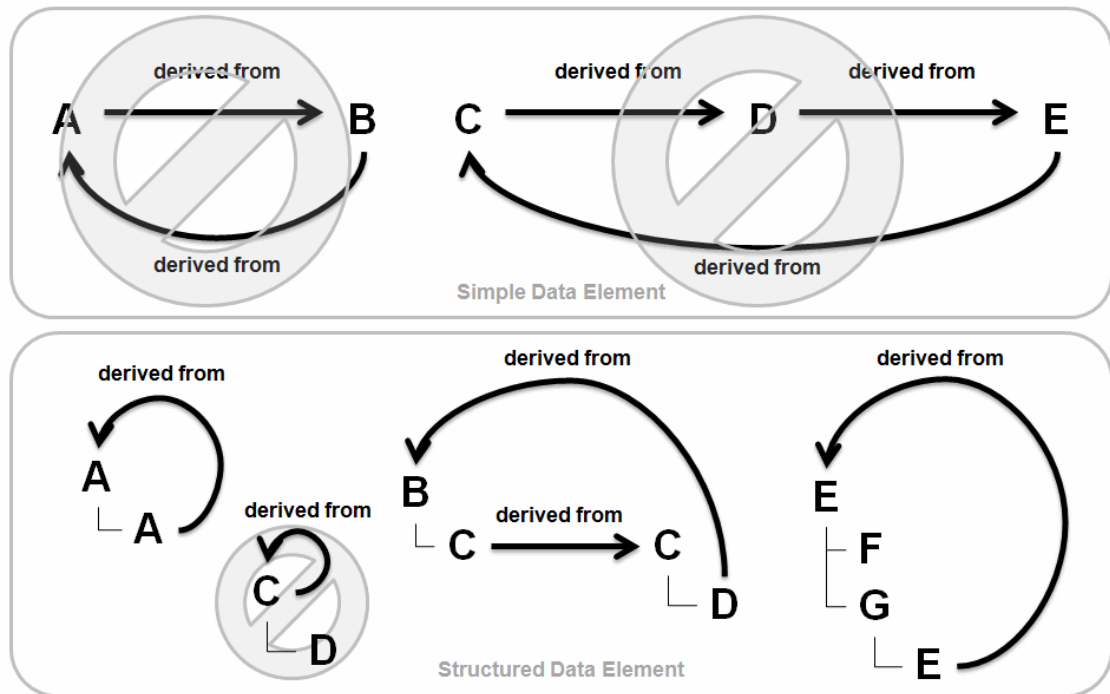
Structured data elements with recursion inherit the data elements of the base type. The Design Studio user interface expands to display a finite number of recursion levels (the number of levels is configurable in the workspace preferences). All structured data elements, including derived structured data elements, can be extended to include additional child data elements.

A recursive reference can be indirect when two structures have data elements that derive from each other. Indirect relationships can be realized through multiple levels of derivation and are not immediately apparent. Figure 4-9 illustrates two different data models using recursion.

Figure 4-9   Data Models Illustrating Recursion
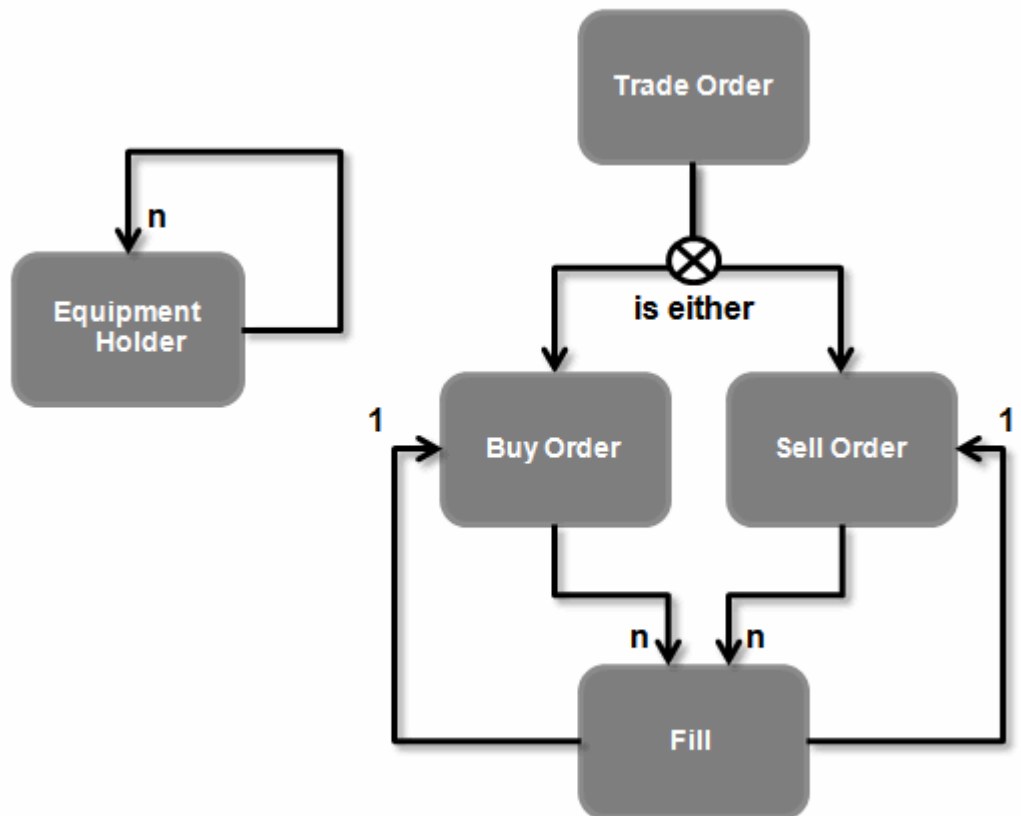
Entity Extensions

You can increase your modeling efficiency by extending entities. Data extensibility enables you to leverage data when building new, similar entities. For example, you can extend orders and tasks. When you extend one entity from another, the target entity inherits all of the data elements defined for the extended entity.

If you extend an entity that includes structured data elements, you can add any number of additional simple and structured child elements (the inherited data elements are read-only).

Figure 4-10   Extending Entities
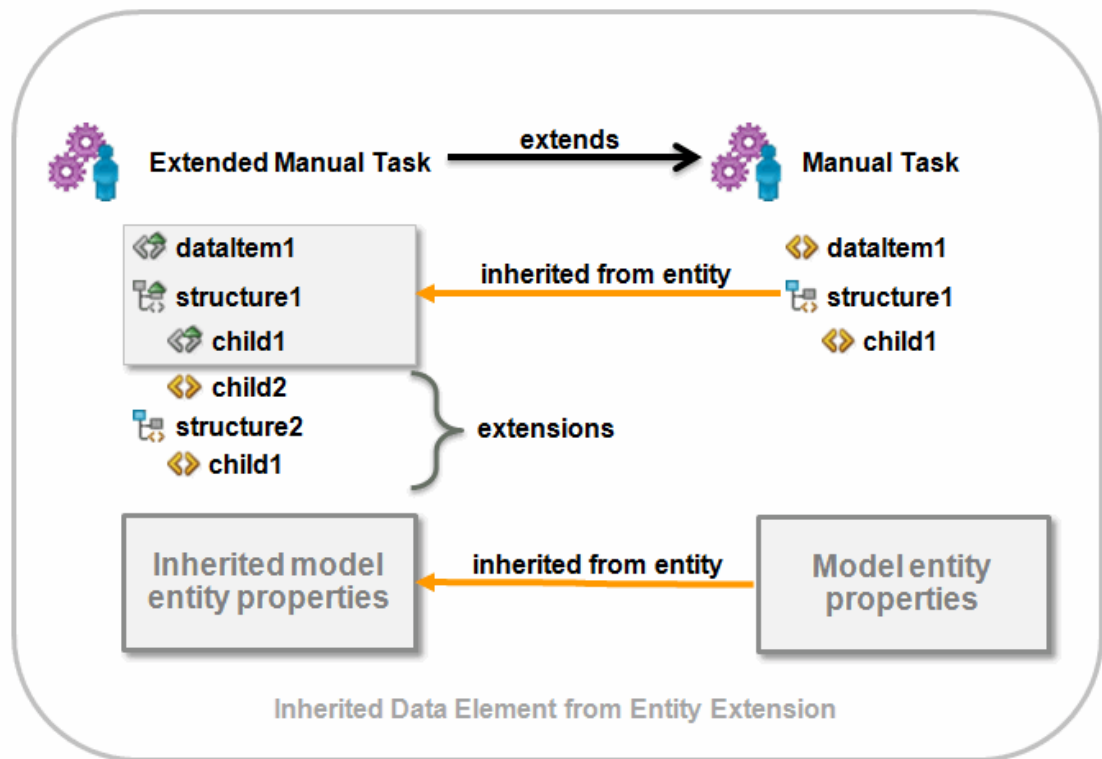
Inheritance

When you extend one entity from another, the target entity inherits all of the data elements defined for the extended entity. These inherited data elements are read-only. Unlike data elements inherited from a base type, the inherited data elements can be extended at any level of the inherited data element.

For example, an OSM manual task can extend another manual task. The data elements modeled on one manual task are extended to another manual task. The target entity can include data elements in addition to those data elements the target entity inherits from the extended entity.

Organizing and Searching for Data Elements

Design Studio helps you keep your data models organized and locate specific data elements during model configuration.

Data Model Hierarchies

Figure 4-11 defines the categories of projects in which you can model data schema entities. Understanding to which category a project belongs will help keep your model organized and efficient. When determining the location for data elements, consider how specific that data element is to a product or to a domain.

For example, common product projects are often sealed models provided by the product teams for solution development, whereas domain-specific model projects are ideal for domain-specific content.

Oracle recommends that you always model in the most generic category possible.

Figure 4-11   Modeling Hierarchies

Category	Domain Specific	Product Specific		
Model: Common	No	No	Least Specific	Used for multiple domains across products
Product: Common	No	Yes		Product specific supporting multiple domains
Model: Domain	Yes	No		Domain specific supporting multiple products
Product: Domain	Yes	Yes	Most Specific	Apply to one domain and one product

Dictionary View

You can use the Dictionary view to facilitate data modeling. The Dictionary view presents a view of the Data Dictionary, including all reusable data elements contributed by entities in the workspace.

The Dictionary view includes an option to filter data elements visible only to the active editor, which enables you to hide those elements in the Data Dictionary that are not intended to be used by specific entities. Additionally, you can filter this view to display only those entities flagged with a specific tag or defined by a specific entity type.

Search fields provide an additional mechanism to locate data elements.

Navigation and Modeling Tips

Design Studio includes shortcuts for navigation. When switching between views and editors, these shortcuts help minimize modeling time and effort.

For example, you can double-click any editor or view tab to maximize the editor or view area. Double-click the tab again to return it to the original size. From the Dictionary view, double-clicking a data element opens the related entity. Double-clicking a data element in an editor opens the base type data element. Clicking the **Type** field link in the Data Schema editor **Details** tab opens the base type of an element.

Also, you can drag data elements from the Dictionary view to an editor data tree when modeling your solution.

Refactoring Data Models

In Design Studio, refactoring is the process of changing data elements without modifying the external functional behavior of a solution.

Refactoring in Design Studio enables you to propagate data model changes across the entire solution without sacrificing model integrity. You can rename, change the location of, copy, and move data elements. Additionally, refactoring enables you to copy data elements to create similar data entities, and to create modular and reusable data structures. See the Design Studio Help for more information.

The following sections include additional information about some of the refactoring actions, all of which are available from the **Refactoring** option in content menus.

Renaming Entities and Data Elements

The rename refactoring option ensures that all references are properly updated when you rename an entity or a data element, and that all aliased names are maintained.

Design Studio displays a detailed list of all the changes to be performed against the workspace that you can review.

In Figure 4-12, a data element named Caller ID is renamed to Call Display. The Rename refactoring functionality ensures that the entities that reference the data element (in this example, an atomic action called Add GSM Subscriber, an OSM order called Mobile GSM Delivery, and an Inventory supplementary service called Supplementary Service) are updated with the new data element name, Call Display.

Figure 4-12   Refactoring Solutions Using Renaming
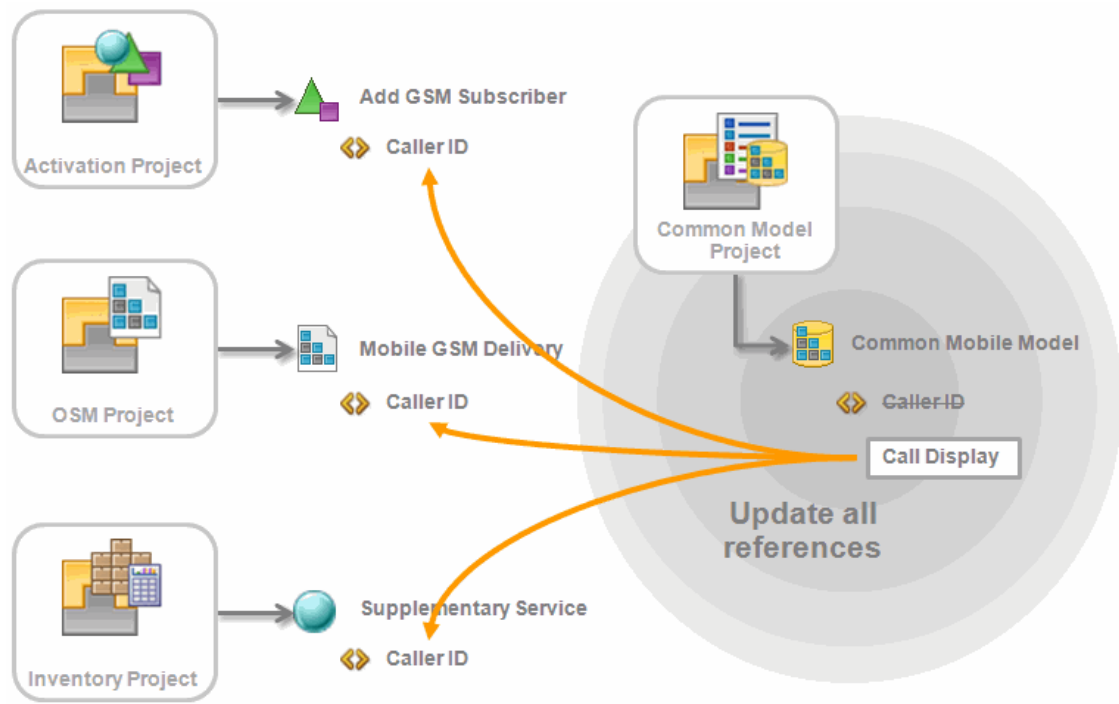
Removing Entities and Data Elements

When you remove entities or data elements from the workspace, Design Studio locates and removes all referenced instances. When you remove entities or data elements, ensure that other constructs in the solution are updated appropriately.

In Figure 4-13, a data element named Anonymous Call Reject is deleted. The Design Studio refactoring functionality ensures that the data element is also deleted from all entities that reference the data element (in this example, an atomic action, an OSM order, and an Inventory supplementary service).

Figure 4-13   Refactoring Solutions Using Remove
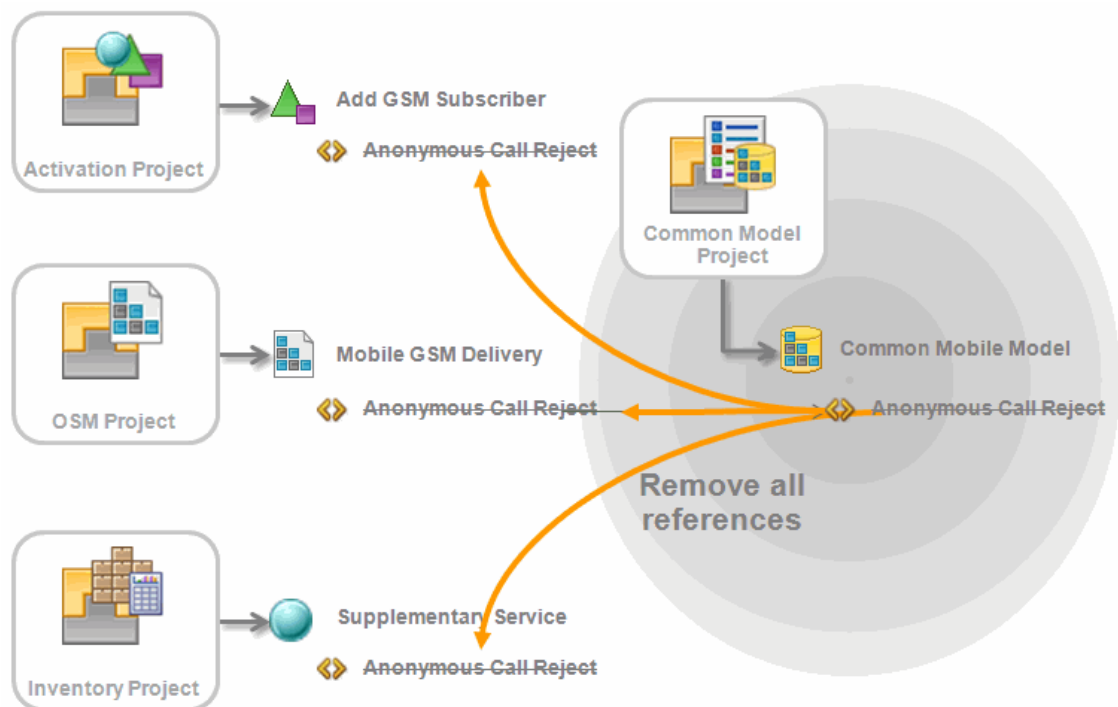
Making Data Elements Reusable

When a structure nested inside a structure is identified as a data element suitable for reuse, you can convert that structure into a reusable root level structure.

Often, the action to make an element reusable is combined with an action to move the data element to a generic location, such as a common data schema entity. In Figure 4-14, the Subscriber structure will be converted into a reusable data element. The original structure is a new, derived structure of the same hierarchy.

Figure 4-14   Refactoring using Make Reusable
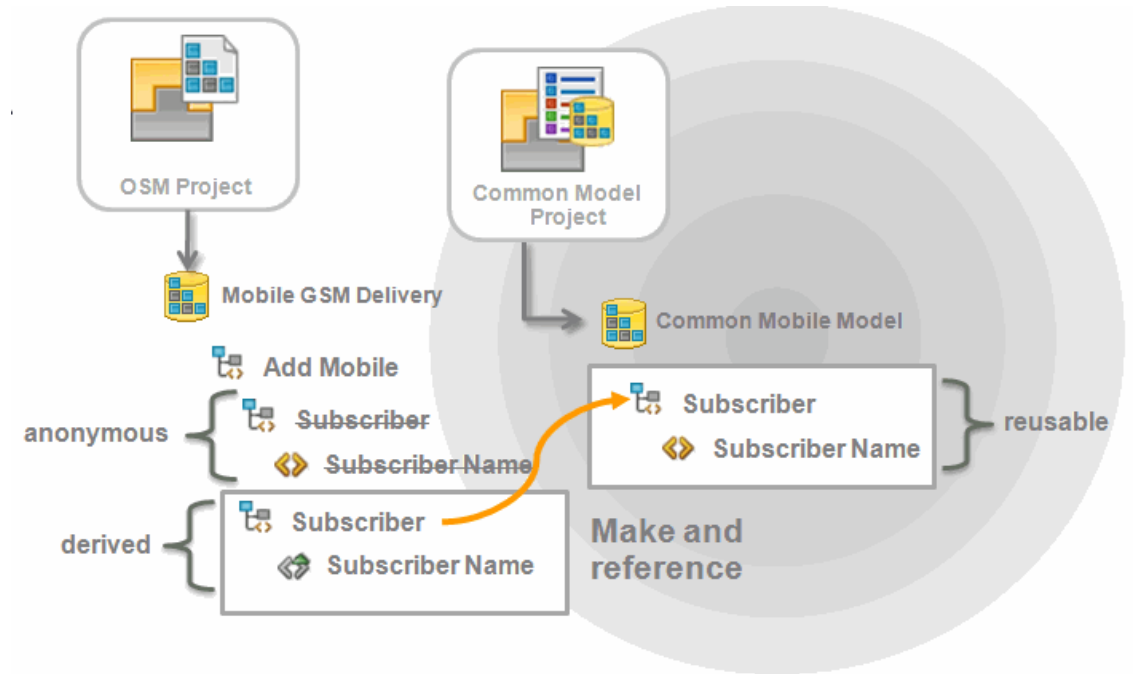
Replacing Data Elements

Figure 4-15 demonstrates how a data element can be replaced with another compatible data element to normalize the solution data model, or to migrate to a newer representation. Depending on the purpose of the replacement, the original data element may be deprecated or removed after the replacement is complete.

Figure 4-15   Refactoring Solutions Using Replace
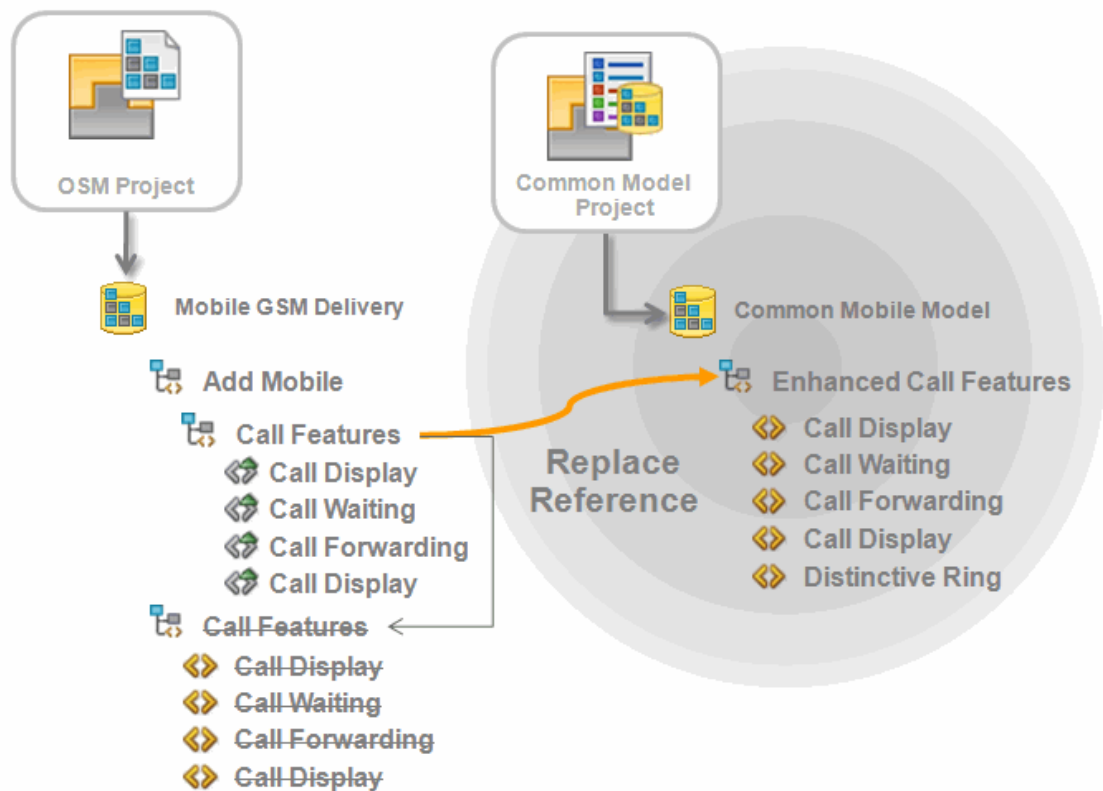
When you refactor solutions using the replace and resolve actions, Design Studio ensures that suitable substitutions are made and that the overall model integrity is maintained. For example, data element replacements must not break any data element references or introduce primitive type inconsistencies. Replacements must meet the following criteria:

*   Primitive types must match

*   Structure is a match or superset

*   Children of structures must be compatible

*   Proposed data elements are reusable

*   Replacement data element does not create an invalid recursion

*   Replacement data element does not reference the data element being replaced

*   Resolve action proposals are less specific

Working with Predefined Data Models

Predefined data models that are provided by Oracle Communications applications include product definitions to support your solution development. You can use predefined data models as the foundation for your new solutions, and build your own data models to extend and augment the predefined models.

There are two types of predefined data models: productized models and common models.

Productized models are predefined cartridges that you can use as the foundation for your new solutions. For example, Oracle Communications supports a library of extensible service

activation cartridges that provide a basis from which to assemble services. You can purchase any of these cartridges, import them into Design Studio, and reuse them to create your own cartridge projects. Productized cartridges include configuration that supports product definitions.

The following common models might also be required by your Design Studio solution:

- The Activation Routing model includes parameters required to support Design Studio for ASAP routing capabilities in a central schema. This model enables you to modify the routing support for each atomic action without requiring you to define the routing parameters. Design Studio adds this model to the workspace when you create an Activation project.

- The Order Management Control Data model contains predefined simple data elements and structured data elements necessary for OSM order templates. For example, the **ControlData** structure is a reserved data area managed by decomposition and orchestration functions. When you create a new OSM project, Design Studio prompts you to add the model if it is not already present in the workspace.

- The Activation Task Response model contains the Activation response parameters as sent from activation work order requests. You use the data in this model to map activation responses to the OSM task data. When you create a new Activation task in OSM, Design Studio prompts you to add this model if it is not already present in the workspace.

You are required to obtain and import the following common models if you are working with Design Studio for Inventory or Design Studio for Network Integrity:

- The **ora_uim_model** common model is a read-only project that represents the UIM model. It supports the ability to define specifications and characteristics and is also used to validate which entity types can be assigned or referenced by configuration items.

- The **ora_uim_mds** common model is a read-only project that represents the fields available displayed for entities in UIM. This project enables you to define the layout of fields in entities.

- The **NetworkIntegritySDK** common model contains software components and libraries required for creating and extending Design Studio for Network Integrity cartridge projects.

Sharing Data Across Application Projects

When you design your fulfillment solution, you need to develop a model comprised of data for use in multiple applications. For example, you may want to create order templates, atomic actions, and service specifications, and share the data defined for those entities across your OSM, ASAP, Network Integrity, and UIM applications. Design Studio enables you to design integrated solutions that reduce occurrences of design errors when sharing data across applications.

About The Data Dictionary

The Data Dictionary is a logical collection of all data elements and types in a workspace. The Data Dictionary enables you to leverage common definitions across an entire Oracle Communications solution.

Design Studio projects are containers of entities, one of which is a data schema. You save all of your solution data in data schemas (all Design Studio features include data schemas). All data schemas in a workspace at any given time contribute to the Data Dictionary. Some Oracle Communications features enable you to create entities and data elements in specific editors, which are also included in the logical collection. You can review the Data Dictionary logical collection in the Dictionary view.

The Data Dictionary enables you to:

- Integrate and correlate the data models for multiple applications.
- Reduce the size and complexity of a solution model.
- Simplify the application integration by eliminating data translation among applications.
- Validate data model integrity.

About Data Leveraging

When modeling entities for your solution, you can use configuration from any other entity. Data modeling is not limited to using data elements within data schema entities. Any data configuration that contributes to the Data Dictionary is available for use.

The following sections provide examples of solution modeling using the Data Dictionary.

Example: Activation Leveraging Inventory Data

Figure 4-16 illustrates how an atomic action (Add GSM Subscriber), created in an Activation project, can be defined using the Caller ID, Call Waiting, and Call Forwarding data elements from a service specification created in an Inventory project.

Figure 4-16   Example: Activation Leveraging Inventory Data
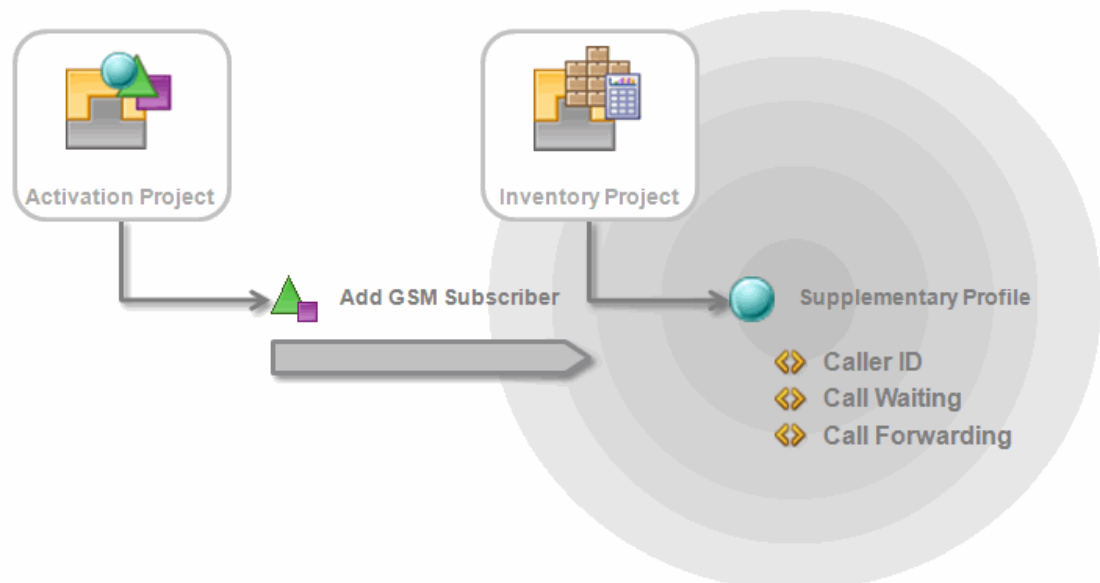
Example: OSM Leveraging Activation Data

Figure 4-17 illustrates how an order (Mobile GSM Delivery) created in an OSM project can use the TN (telephone number) and TN Type (telephone number type) data elements from a service action modeled in an Activation project.

Figure 4-17   Example: OSM Leveraging Activation Data
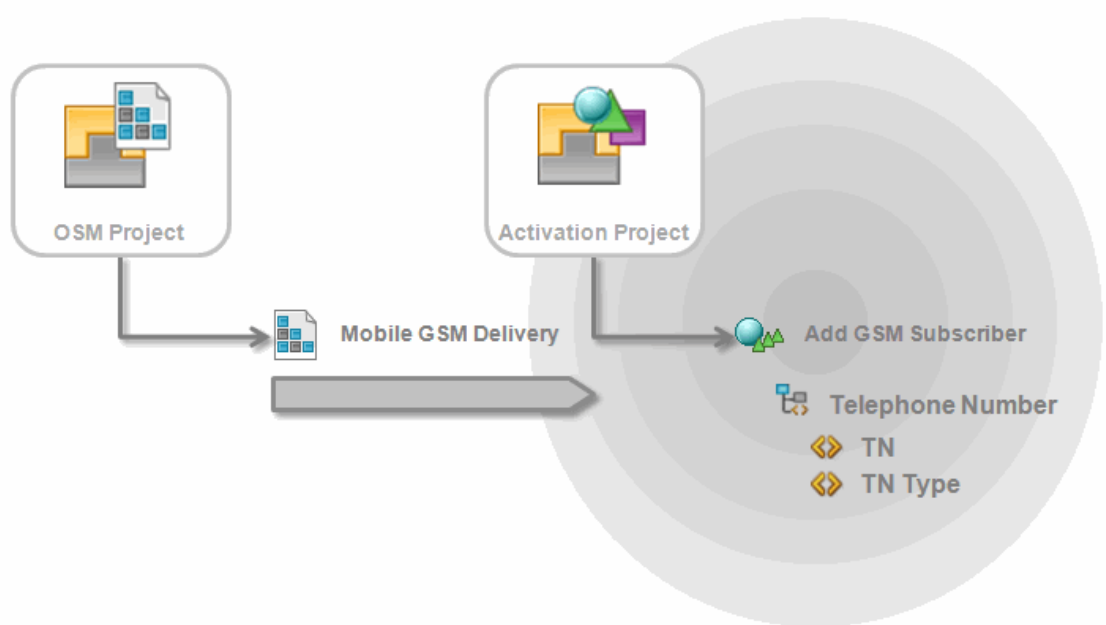
Example: OSM Leveraging Inventory Data

Figure 4-18 illustrates how an order (Mobile GSM Delivery) created in an OSM project can use the Caller ID, Call Waiting, and Call Forwarding data elements from a service specification modeled in an Inventory project.

Figure 4-18   Example: OSM Leveraging Inventory Data

5
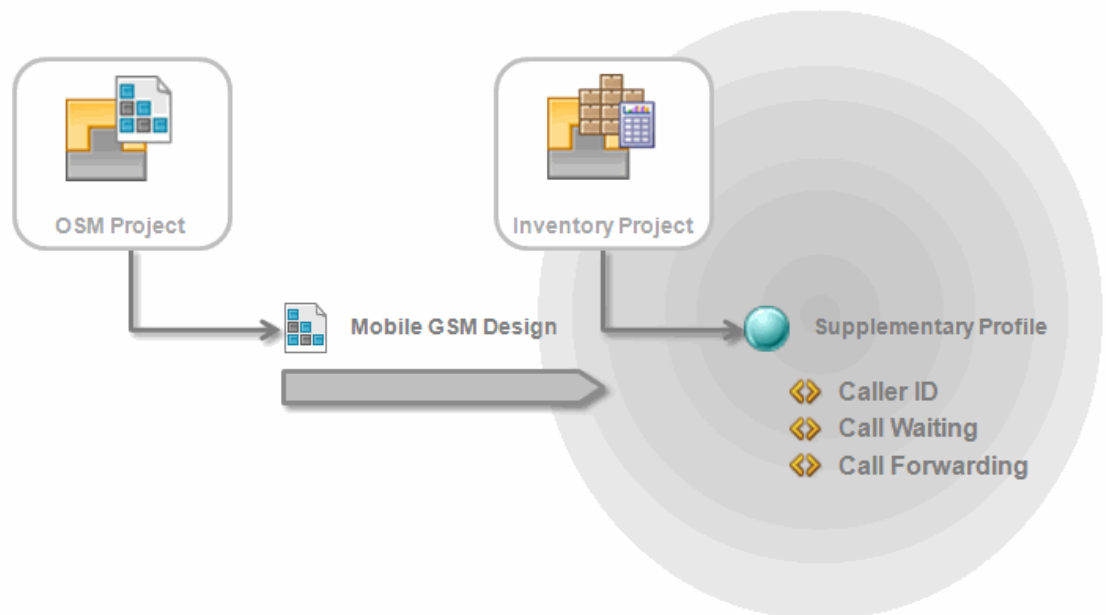
Working with Design Patterns and Guided Assistance

This chapter provides information about design patterns and guided assistance.

About Design Patterns

In Design Studio, design patterns are wizards that can be run to apply sets of resources to a Design Studio workspace and that can be used to deliver complex sets of preconfigured artifacts that serve some domain-specific function. They enable you to formalize modeling patterns into reusable components that can be applied to various solutions, and reduce the time and effort required during implementation.

During solution design, designers often create and configure complex sets of related entities and the relationships among them. These sets of tasks can be documented as best practices and can be used as templates, or design patterns, for reproducing the configuration.

Design patterns enable automation of complex, repeatable tasks, and enable team members with varying levels of skill to complete these tasks. For example, a solution implementation team can develop design patterns to enable network engineers to manage VPN-related tasks, such as creating and deleting VPNs, adding sites and removing sites, and so forth.

Typically, designers create design patterns using groups of design artifacts identified from existing implementations. Designers package the design patterns and distribute them to solution design teams. Solution design teams can install a design pattern as a Design Studio feature and, using a wizard, apply the pattern to their workspace. These wizards ensure compliance with the best practices and reduce the need for coding and complex configuration. Your teams can use design patterns to reduce errors, simplify modeling, and increase productivity.

See *Design Studio Developer's Guide* for more information about creating custom design patterns. See Design Studio Help for more information about applying design patterns.

About Guided Assistance

Design Studio guided assistance is a range of context-sensitive learning aides mapped to specific editors and views in the user interface. For example, when working in editors, you can open the Guided Assistance dialog box for Help topics, cheat sheets, and recorded presentations that are applicable to that editor.

When working with guided assistance, you can review the learning aids delivered with Design Studio, and you can create your own and map them to projects and entities by using design patterns or by defining values for attributes directly in the guided assistance extension point.

See *Design Studio Developer's Guide* for more information about guided assistance.

About Cheat Sheets

Design Studio supports cheat sheets, which refers to the integration of documented procedures with wizards in the application. Cheat Sheets are XML documents that can be

interpreted by the Eclipse Cheat Sheet framework, and developers can map cheat sheets to specific points in the Design Studio user interface (for example, in editors and views). Users can access the cheat sheets that are relevant to current tasks, and complete those tasks using the included instructions. You can configure cheat sheets to provide links to relevant Help topics and facilitate the learning of those procedures.

For example, you can use cheat sheets with design patterns to describe the resources added to a workspace, and to assist users with any manual steps required after a design pattern is applied. Cheat sheets are not mandatory for design patterns, but recommended.

You can develop and edit cheat sheets using the Eclipse Cheat Sheet editor.

See *Design Studio Developer's Guide* for more information about cheat sheets.

6

Working with Conceptual Models

This chapter describes how you use Design Studio to build a conceptual model of a service domain.

About Conceptual Models

Conceptual models define the relationships between your commercial products, the services that they represent, and the resources that are required to implement the services. They define how commercial products and technical services are related, and they enable you to associate the products that you sell with the technical services and resources that are required to fulfill orders.

Conceptual models comprise entities that represent components of a service but that contain no detailed application-specific information. The information that you define for conceptual model entities determines how service capabilities can be commercialized. For example, the conceptual model defines your products and services and the actions that must be performed in a run-time environment to provision a service order request.

When designing conceptual models, you define the associations among the conceptual model entities. You also define the patterns that are used to fulfill service orders and technical orders. These patterns define how requests to design reusable service components are fulfilled and how delivery systems activate and manage the work to be performed on resources in the network.

Figure 6-1 is a simple example that illustrates how a product (called Broadband) that is sold to a customer is fulfilled by a service (called Broadband Internet Access).

Figure 6-1   Conceptual Model: Products and Services
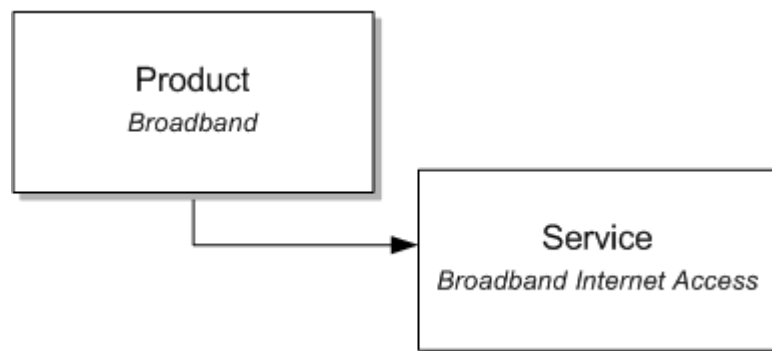
Figure 6-2 expands on the example and illustrates how you can associate multiple products to the same service. The Broadband and Broadband Bandwidth products are now both associated with a single customer facing service, the Broadband Internet Access service. Also, you can associate the service to a specific location.

Figure 6-2   Conceptual Model: Products, Services, and Locations
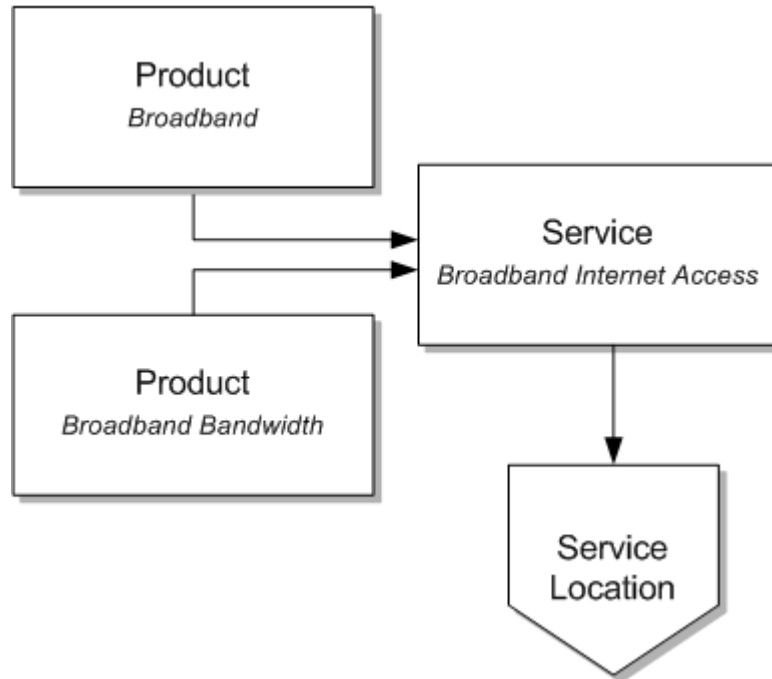
Conceptual models also include definitions of the actions that are required to provision a service. For example, you can define the actions necessary to add a DSL service for a new customer or disconnect an existing DSL service for an existing customer.

Figure 6-3 expands the example further. The example shows a small subset of the technical components required to provision the Broadband Internet Access service; for example, a resource facing service (DSL) and two resources (customer premise equipment and a DSL interface, named DSL CPE and ADSL Interface, respectively).

Figure 6-3   Conceptual Model: Products, Services, CFS, RFS, and Resources, and Actions
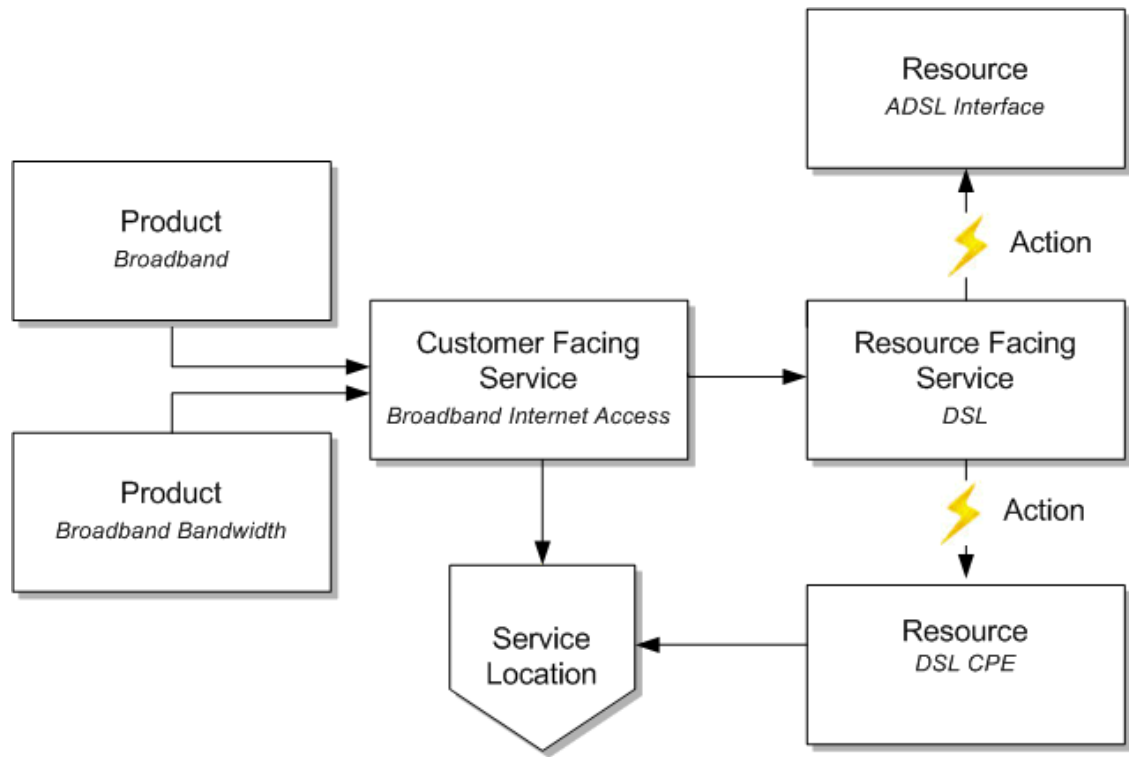
is a simple conceptual model for a broadband domain.

Figure 6-4   Conceptual Model Example
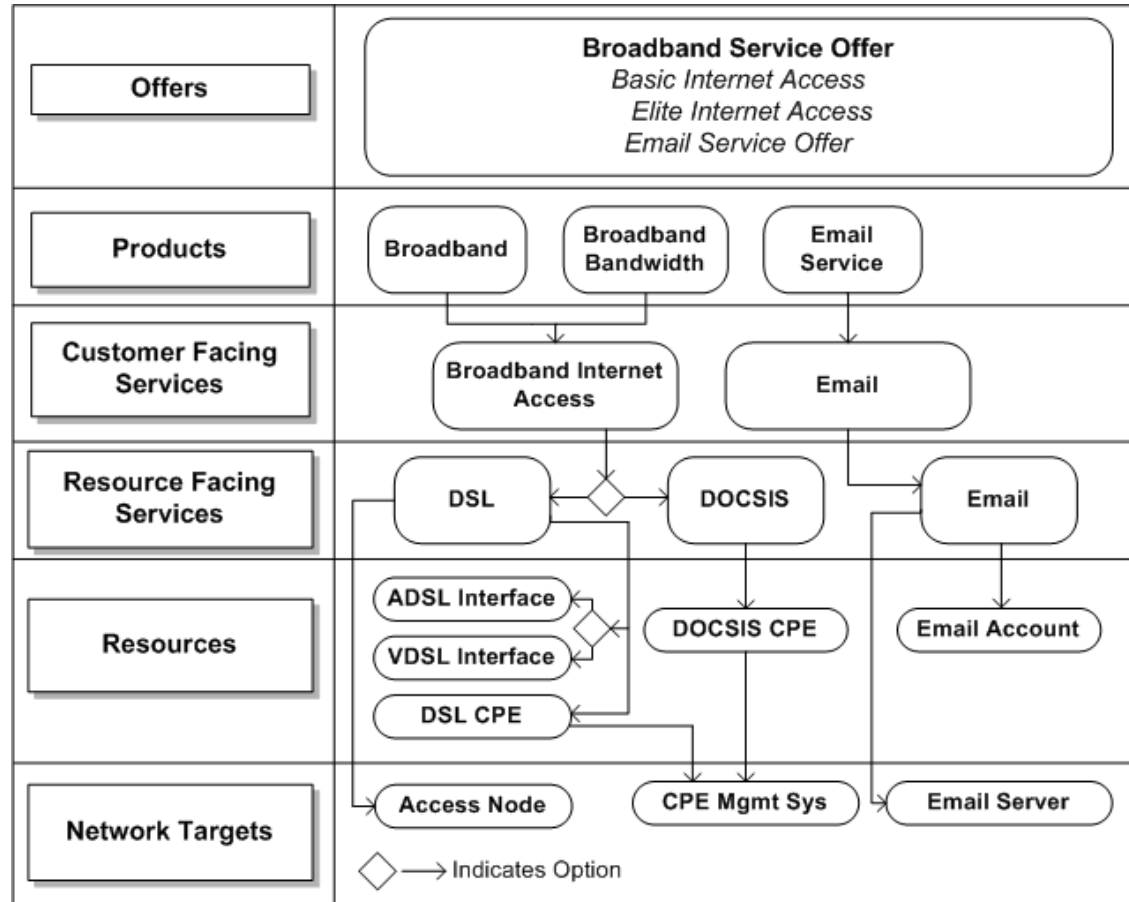
You define conceptual models in Design Studio Model projects, which are not deployed to run-time environments. You convert conceptual model entities into application entities (you save application entities in application cartridge projects). This conversion is called realization. You can enrich the application-specific entities with additional configuration and deploy the application cartridge projects to run-time environments.

Conceptual model entities can also serve as placeholders for products and services that are not realized by Oracle Communications applications. For example, you can use conceptual models to enable Oracle Communications Order and Service Management (OSM) to work with external systems if you intend to associate conceptual model entities to applications that are not part of the Oracle Communications suite of applications.

About Conceptual Model Entities

A conceptual model includes the following entities:

- Customer facing services (CFS), which represent services from a customer perspective. See "About Customer Facing Services" for more information.

- Resource facing services (RFS), which represent a technical view of a service. See "About Resource Facing Services" for more information.

- Resources, which represent the entities that are required to configure the service. See "About Resources" for more information.

- Products, which represent commercial products. See "About Products" for more information.

- Locations, which represent physical locations. See "About Locations" for more information.

- Actions, which describe how conceptual model entities change, cause change, or retrieve information. Actions are associated with customer facing services, resource facing services, and resources. See "About Actions" for more information.

- Action codes, which represent the specific types of actions permitted for each type of action. For example, an action can include a number of action codes to represent create, disconnect, and remove.

- Action parameter bindings, which enable you to identify the conceptual model entities (such as a resource or a resource facing service) that contribute to the creation of technical actions, and to map the source data (a conceptual model entity, attributes defined for a conceptual model entity, and components defined for a conceptual model entity) to target data (parameters defined for technical actions). See "About Action Parameter Bindings" for more information.

- Domains, which are groups of entities and actions that you can use to organize and filter solutions. See "About Domains" for more information.

The following conceptual model entities help to define the model infrastructure and rarely require updates. You can extend these entities to meet the requirements of a specific deployment:

- Application roles, which represent types of downstream delivery systems that are responsible for specific types of delivery, such as activation, supply chain management, work force management, and so forth. See "About Application Roles" for more information.

- Functional areas, which are the logical layers of an installation and can be commercial, service, and technical layers. These layers are supported by an OSM order type or by an external order management system. See "About Functional Areas" for more information.

- Provider functions, which are processing components that perform a defined set of tasks based on their role in a solution. Provider functions comprise standard sets of unique capability, defined by the input the function accepts (an order or standard request), the output it generates (an order or standard request), and the data that drives the provider function description and purpose. Design Studio includes the configuration for some provider functions, such as Calculate Service Order, Design and Assign, Calculate Technical Order, and Activation. See "About Provider Functions" for more information.

- Fulfillment patterns, which describe the high-level functions that are required to process an action or a conceptual model entity. See "About Fulfillment Patterns" for more information.

- Fulfillment functions, which represent the work to be performed against an action. Fulfillment functions can be augmented with conditions to determine whether the function is to be performed against an action. See the Design Studio Help for more information.

About Customer Facing Services

Customer facing services represent the commercial view of the services that you provide to your customer (a service represents the way that a product is realized and delivered to a customer). You can use the same CFS to fulfill different but similar product offers. For example, the same **Broadband_Internet_Access** CFS can be used to fulfill an order for a **Broadband** product and a **Broadband_Bandwidth** product.

You relate the data elements defined for customer facing services to data elements defined on products. Additionally, you associate customer facing services with resource facing services

(as components). For example, you can associate with a CFS the resource facing services available to fulfill the service, such as DSL or DOCSIS.

Service actions change a customer facing service. Service actions represent requests to change a customer facing service in the technical inventory. These actions are used during product-to-service mapping and during run-time Calculate Service Order and Design and Assign provider function activities.

Customer facing services are represented in Design Studio for Inventory as Service specifications and as Service Configuration specifications.

Customer facing services are associated with products through primary and auxiliary relationships. See "About Conceptual Model Entity Relationships" for more information.

Use the following set of guidelines when creating customer facing services:

- Define customer facing services to support multiple products. A CFS can support multiple products if it is not defined for a specific technology.

- Define customer facing services to include attributes of the service that are important to a customer, and hide technology details that are not relevant to a customer.

- Define customer facing services to represent a single domain. For example, the **Broadband_Internet_Access** CFS represents the Broadband domain.

- Define customer facing services with no explicit dependencies to other customer facing services. An explicit dependency is a direct relationship. Oracle does not recommend, for example, that you create a relationship between the **Broadband_Internet_Access** CFS and an **Email** CFS.

About Resource Facing Services

A resource facing service (RFS) describes how customer facing services are configured. For example, you can fulfill a customer facing service named **Broadband_Internet_Access** using multiple modes of delivery, each represented by a resource facing service, such as DSL, DOCSIS, or Fiber. You determine the resource facing service used to provide the commercial-level services during service design.

Resource facing services are technology-specific but not vendor-specific. Resource facing services have associations with resources or with other, finer-granulated resource facing services. For example, the **DSL** RFS has an association with the **CPE** (customer premise equipment) resource to represent a cable modem.

Resource facing services can be defined as targets for technical actions, which represent operations that make changes in the network. Resource facing services are represented in Design Studio for Inventory as Service specifications and as Service Configuration specifications.

About Resources

Resources are entities that are required to configure a service. A resource is a specific object in the network and in the inventory that can be consumed, referenced, or shared by a service when provisioning a resource facing service. Resources can be physical, such as a port, or logical, such as bandwidth. Examples of resources include IP addresses, VoIP phones, and DSLAM ports.

Resources have associations with other resources and can be defined as targets for technical actions.

Technical actions represent changes to the network. For example, a technical action to add customer premise equipment (CPE) is defined with the data required by the network to recognize the connection made by the new CPE.

Resources can be converted to Design Studio for Inventory entities. See "About Realizing Resources in Design Studio for Inventory" for more information.

Additionally, you can define resources that you intend to realize in external systems and realize the entities manually. You can also define resources as **Other** if no Design Studio for Inventory resources meet your business requirements, and realize the entities manually. See "About Conceptual Model Realization" for more information.

About Products

A product is an entity that your business sells. A product defines a set of product characteristics, validation rules, and relationships. For example, you might create products for **Broadband**, **Broadband_Bandwidth**, and **Email** products.

You can create products in Design Studio, or you can import products into Design Studio. For example, when new products are added to the product catalog, the corresponding product must be imported into Design Studio. After you create or import products, you can create or review the associated attributes in the Product editor.

Products are used by orchestration processes to map order lines to fulfillment actions and to map order lines to Service specifications.

> **Note:**
>
> Conceptual model products are not realized as application entities.

For products, you define primary and auxiliary relationships to customer facing services. See "About Conceptual Model Entity Relationships" for more information.

Use the following guidelines when creating products:

- Ensure that your products represent functionality meaningful to a customer.

- Define products to facilitate reuse in multiple bundled offers. Minimize overlap among product definitions to ensure that a simple assembly of product offers can be maintained. Duplication among product definitions can complicate customer relationship management processes and increase operations cost.

- Define products so that they do not expose unnecessary details. For example, a **Broadband** product includes only data elements that represent the properties of the service being ordered, such as upload speed, download speed, service address, customer ID, and so forth. Products do not include data elements that represent technical elements of the service, such as the MAC address or IP address of the home location register (HLR) server.

About Locations

A location represents a geographical place that is relevant to services or resources. Locations are realized as Design Studio for Inventory Place specifications defined as type **Site**. For example, in a **Broadband** service, the **DSL** RFS for a DSL service can be associated with a location to represent the address of the customer.

When realizing Location entities, Design Studio also creates a Place Configuration specification.

About Domains

A domain is a group of entities and actions that you can use to organize and filter solutions (you do not realize domain entities as application entities). For example, you can create a domain called **Alcatel_DSLAMs** that contains the resources (the Alcatel devices) that can be used as a DSLAM. You can include conceptual model entities in multiple domains, and you can create a hierarchy of domains that include subdomains. Subdomains can decompose into smaller groupings (for example, into broadband products and broadband services). Domains can be used as subdomains, and subdomains can be shared across multiple domains.

You can filter the Solution view to display domains and view and navigate among only those entities that are associated with domains.

At run time, the Order and Service Management application uses the domain and the provider function to decide which transformation sequence to use. For example, to ensure that the TS-1 transformation sequence is always used for the Mobile domain and that the TS-2 transformation sequence is always used for the Carrier Ethernet domain, you would model the data in Design Studio such that the Mobile domain and CSO provider function combination uses TS-1 and the Carrier Ethernet and CSO provider function uses TS-2.

About Application Roles

An application role represents a type of downstream delivery system that is responsible for a specific type of delivery, such as activation, supply chain management, work force management, and so forth. Design Studio includes a set of predefined application roles. You can create your own application roles using the Application Role editor.

When you create a new action, Design Studio prompts you to select an action type of service or technical. If you select an action type of technical, Design Studio requires that you also select an application role. Design Studio populates the Action editor **Action Codes** tab with the specialized action code names defined for the application role. A specialized action code is an action code that you rename to align a technical action with a downstream fulfilment system.

For example, you may need to rename the **Create** action code to **Activate** to better align with code defined in a downstream activation system. For a supply chain management system, you may need to rename the **Create** action code to **Ship**. You may need to rename the **Remove** action code to **Uninstall** to better align with a downstream workforce management system.

Additionally, the Application Role editor enables you to define multiple specialized action code names for each default action code. For example, a downstream fulfillment system may require multiple versions of the **Modify** action code. You can differentiate between these versions by defining two unique specialized action code names, such as **Change** and **Revise**.

About Provider Functions

Provider functions are processing components that perform a defined set of tasks based on the provider function role in a solution. Design Studio includes the configuration for some provider functions, such as Calculate Service Order, Design and Assign, Calculate Technical Order, and Activation.

Provider functions accept conceptual model entities and the actions that are associated with those entities as input and generate conceptual model entities or their actions as output. The output that provider functions generate is used as input by other, downstream provider functions. Calculate Service Order accepts products as input and generates the service actions

associated with customer facing services as output. This output is required as input by the Design and Assign provider function, which generates output that is required by the Calculate Technical Order provider function.

For example, during the fulfillment of a new wireless voice service, the Design and Assign provider function creates instances of Service specifications and Service Configuration specifications that are required to deliver the service (the telephone number, the bandwidth, and so forth). Information about these specifications and the actions that are associated with them is used by Calculate Technical Order to calculate the difference between the current and the requested state of a configuration and to identify the technical actions that are required to achieve the requested changes.

Provider function definitions determine the types of components that are available to model for a conceptual model entity. A conceptual model entity is associated with a provider function when the conceptual model entity is defined as an input type. The components that you can define for a conceptual model entity are limited to the output types defined on all provider functions with which the conceptual model entity is associated.

For example, consider that you are defining components for a Customer Facing Service entity, and that there exists in the workspace only one provider function, named Design and Assign. And, consider that this provider function defines the Customer Facing Service entity as input and defines the Resource Facing Service entity and the Location entity as output.

In this example, when configuring components for a Customer Facing Service entity, you can define as components only resource facing services and locations. If you need to add the resources as components of a customer facing service (such as when modeling a Carrier Ethernet) you can edit the Design and Assign provider function definition to add the Resource entity as an output type.

Provider functions are associated with OSM transformation sequences and orchestration processes.

About Functional Areas

Functional areas describe the logical layers of an installation, and can be commercial, service, and technical layers. These layers are supported by an OSM order type or by an external order management system.

When configuring functional areas, you specify whether the functional area supports actions. If it does, you indicate which conceptual model entities support the type of actions defined by the functional area and whether these actions are multi-instance. For example, the **Service** functional area supports actions, but only on customer facing services, resource facing services, and resources. The **Technical** functional area also supports actions, but those actions can be associated only to resource facing services and resources.

You can define default data elements for service actions associated with a functional area. When a service action is created, the default data elements defined in the Functional Area editor appear on the Service Action editor **Data Elements** tab. The data elements that you add to a functional area are automatically associated with an implicit parameter tag to make the data elements read-only in the Service Action editor.

Functional areas are associated with a list of action codes. Each action code represents an action that can be performed on a conceptual model entity. When a new action of the type defined by the functional area is created, all of the action codes defined in the functional area are added to the new action. You can define the applicability of each default data element to the service action codes associated with the functional area.

About Fulfillment Patterns

Fulfillment patterns define a set of high-level functions that can be performed to process an action or conceptual model entity. For example, a fulfillment pattern can define the functions that are required to process conceptual model entities and actions for provisioning, billing, or installation. In OSM, you associate fulfillment patterns with the processes that run the order items.

You can associate any conceptual model entity or action to a fulfillment pattern. For example, you can associate multiple products to one fulfillment pattern, which enables you to introduce new products with minimal configuration in Design Studio. Fulfillment patterns realize as OSM fulfillment patterns.

About Fulfillment Functions

Fulfillment functions are operations that can be performed to process an order item. In the context of the conceptual model, the line item is an action or a product. After you create fulfillment functions, you associate them with fulfillment patterns. You also associate actions with fulfillment patterns, and the fulfillment pattern associated with any action determines which fulfillment functions can be associated with the action.

Fulfillment functions can be realized as OSM Order Component specifications. In OSM, line items are sent to a fulfillment pattern, which includes a set of order components that can be used to process the line item. The order components represent work that needs to be done against the line item. The fulfillment pattern determines which order components are to be used based on the conditions and dependencies defined in OSM. Each order component can have dependencies to other components (for example, one component may require that another be completed first).

About Action Parameter Bindings

Action parameter bindings represent the aggregate of an entity and its parameters in the context of an application role. They enable you to identify the conceptual model entities (such as a resource or a resource facing service) that contribute to the creation of technical actions, and to bind the source data (a conceptual model entity, attributes defined for a conceptual model entity, and components defined for a conceptual model entity) to target data (parameters defined for technical actions).

A technical action represents a unit of work that is performed to realize a resource or a resource facing service in a network. Technical actions are performed by a fulfillment system. For example, an activation system performs technical actions to configure a network; a shipping system performs technical actions to pick, pack, and ship physical goods; and a work force management system performs technical actions to dispatch work to a technician.

During service order fulfillment, a design and assign process produces a service configuration that defines the technical actions that must be executed to fulfill the requested service action. The subject and the target of a technical action each bind to a resource or to an RFS that is assigned to or referenced by a CFS or by an RFS service configuration. A technical action also defines parameters that describe the work to be done, and these parameters also bind to properties of a resource or of an RFS assigned to or referenced by the service configuration. In Design Studio, you model these bindings in Action Parameter Binding entities.

At run-time, after a requested service is designed, a calculate technical actions process compares the current configuration against the requested configuration. To determine if there are differences between the current and requested configurations, the process analyzes the

action parameter bindings and determines whether any of the parameters have differences. A difference in any parameter means that the technical action must be performed (but only under specific conditions; for example, a technical action may be required only when a subject resource is newly allocated). Using this method, the calculate technical actions process identifies all of required technical actions that must be performed to affect the necessary changes in the network (and to activate the services).

About Action Parameter Bindings and CTA Metadata

The process that compares a current configuration against a requested configuration is called the Calculate Technical Actions (CTA) provider function. The metadata that is necessary for the CTA provider function to calculate the delta is contained in an XML file for each Service specification in the Inventory system. Action parameter bindings help you create the metadata in the XML files by graphically illustrating how the attributes in the conceptual model contribute to the parameters in the technical actions.

The CTA metadata in the XML files is not, however, generated automatically in Design Studio. Rather, Design Studio enables you to complete the modeling necessary to facilitate the CTA processes. To automate the creation of CTA metadata, you must develop a CTA metadata generator. This generator explores the conceptual model configuration and generates the necessary metadata.

You can leverage the Design Studio Exchange Format and create your own CTA metadata generator, or you can use the example that is included in the Oracle Communications RSDOD Reference Solution, which is available on the Oracle Technology Network. See *Design Studio Developer's Guide* for more information about the Design Studio Exchange Format. See the *Oracle Communications RSDOD Reference Solution Developers Guide* for more information about generating CTA metadata.

About Conceptual Model Entity Relationships

The conceptual model entities and the relationships among them form a model of your service domain. These associations are called named relationships, and you add named relationships to entities by defining components.

A component is a container that represents all of the viable configurations that can be defined for the relationship. For example, the **Broadband_Internet_Access** customer facing service (CFS) can include the **Access** component. The **Access** component represents all of the resource facing services that can be used to deliver the **Broadband_Internet_Access** service.

> **Note:**
>
> The types of components that are available to model for conceptual model Customer Facing Service, Resource Facing Service, and Resource entities are determined by provider function definitions. See "About Provider Functions" for more information.

When you first begin modeling, you may not have information about a component except for the name and component type. For example, early in conceptual model design, you may not yet have specific details about the **Access** component except to know that the **Broadband_Internet_Access** CFS requires specific technologies to support the service. The **Access** component name describes the role of the relationship (Access) and the component type describes the specification (a resource facing service).

To complete the model, you define the component options. For example, the **Access** component may include **DSL**, **Fiber**, and **DOCSIS** options. Figure 6-5 illustrates how these options are instances of resource facing services, and they all represent viable resource facing services that can support access for the **Broadband_Internet_Access** CFS. The type of access that you provision may depend on the geographic location, where one location requires you to use DSL, another requires you to use DOCSIS, and a third requires Fiber.

Figure 6-5   Defining Conceptual Model Components
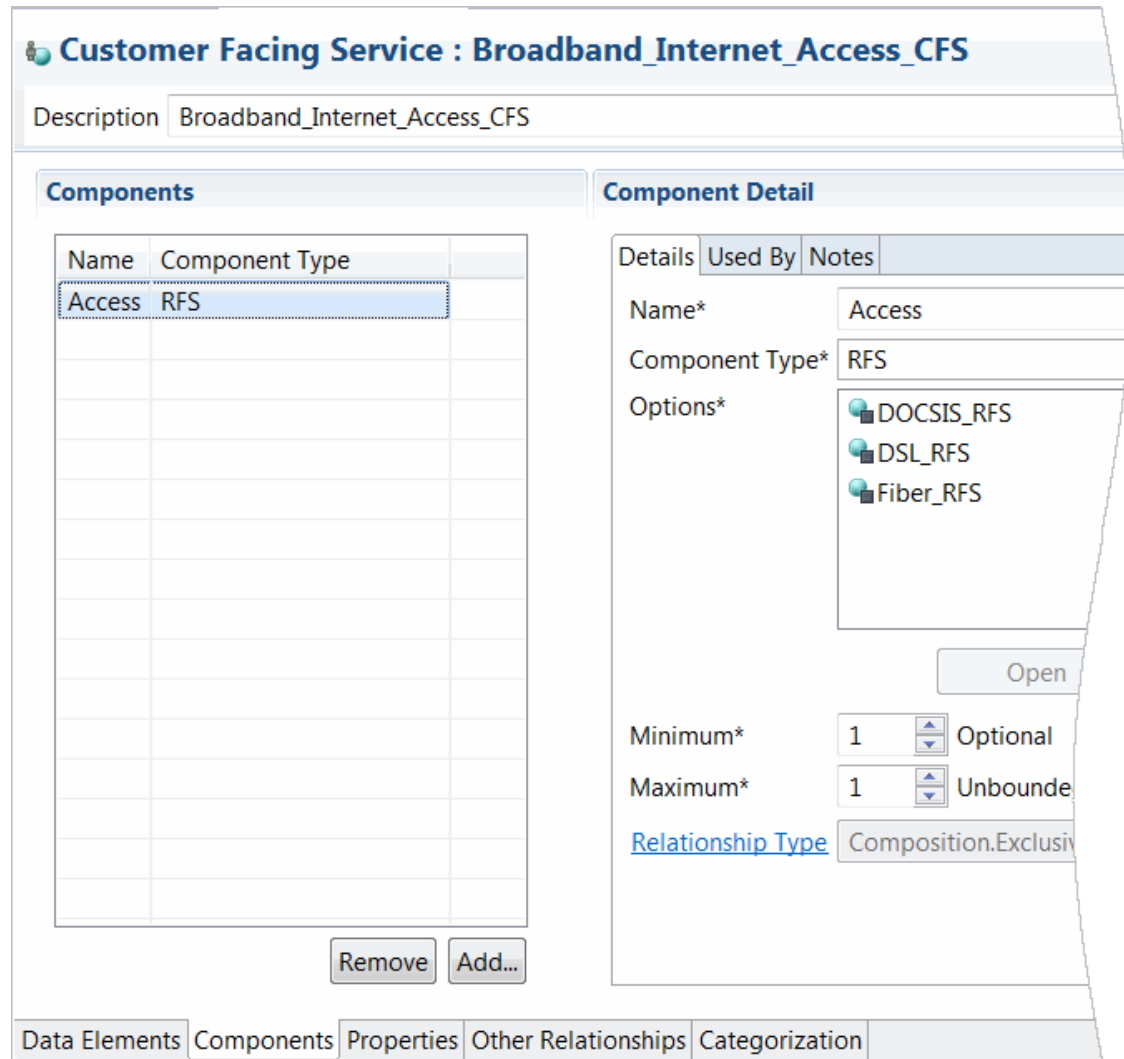
Figure 6-6 illustrates how the **DSL** RFS also has named relationships defined as components. For example, the **DSL** RFS can have an **Account** component, a **CPE** (customer premise equipment) component, and a **DSL_Interface** component. Each of these components defines the viable resources that can be used to provision a service in the network. For example, the **DSL_Interface** component can be defined with two options: an **ADSL_Interface** resource and a **VDSL_Interface** resource.

Figure 6-6   Defining Components at the RFS Level
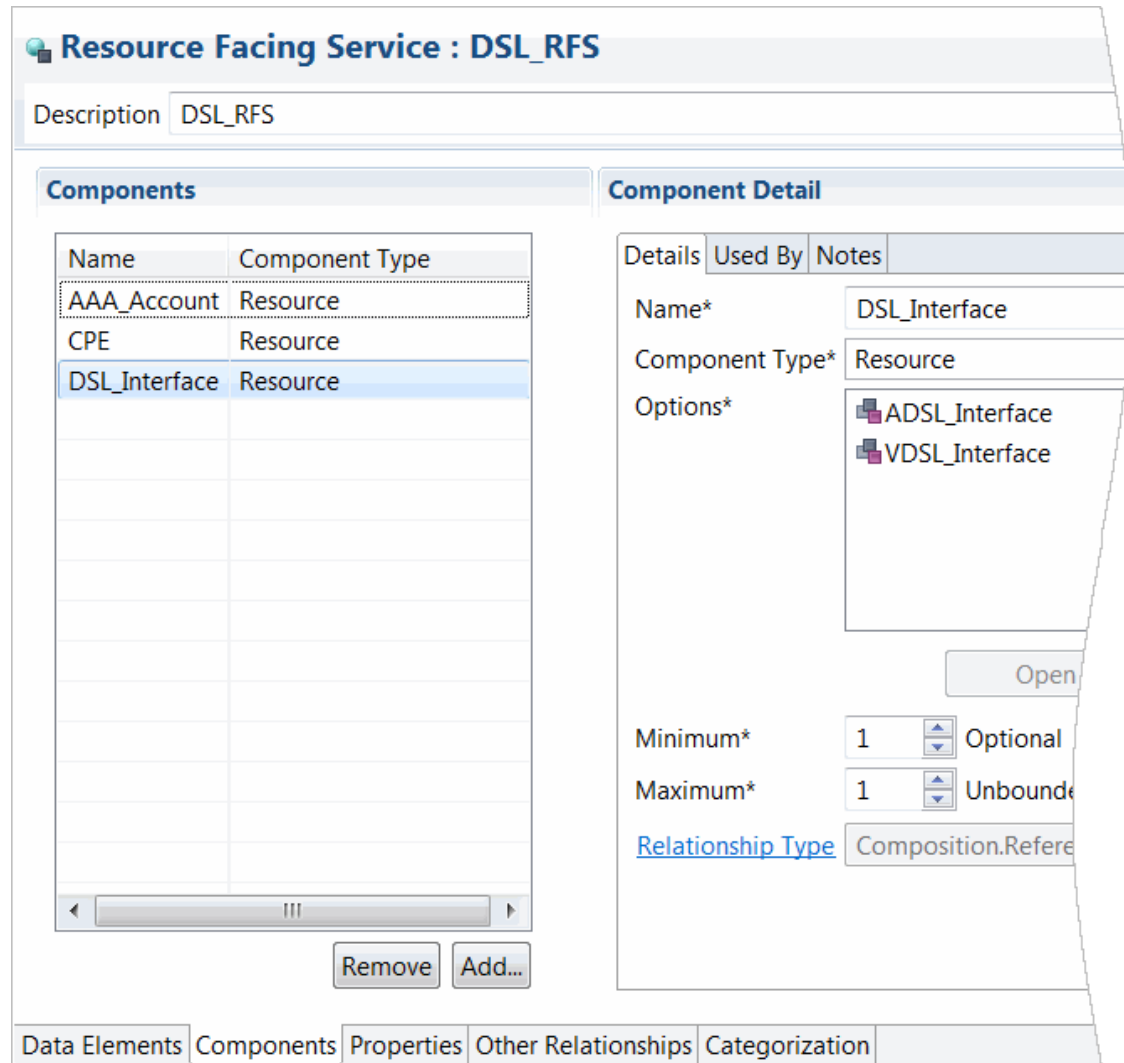
About Relationship Types

Conceptual model entities have associations with other entities through named relationships, which are defined as components. Each component associated with a conceptual model entity is defined with a specific relationship type.

The following relationship types exist in Design Studio:

- Primary relationships are used to associate products to services. In a run-time application, this type of relationship is used to instantiate a new service order line. The Primary type defines relationships between a product and a customer facing service (CFS). For example, when a customer orders the **Mobile** product, the **Mobile** CFS fulfills the order; the CFS connects the ordered product with the technical aspects required to fulfill the product services.

  For each product, you define a Primary relationship to a single CFS. Every CFS must have a Primary relationship defined with at least one product.

  In some scenarios, the Primary type can define a relationship between a product and a resource. For example, if a customer orders a mobile phone case, there is no service

required to provision the case. In this scenario, you associate the **Case** resource with the **Mobile** product.

- Auxiliary relationships are used to associate products to services, but this type of relationship is used by run-time applications to enrich existing service order lines (it does not instantiate a new order line). The Auxiliary type defines relationships between a product and a CFS (and in some rare cases, between a product and a resource).

  A CFS can define any number of Auxiliary relationships with products, but the Auxiliary relationships cannot exist independent of a Primary relationship. For example, the **Mobile** product may include a number of features, such as call waiting, caller ID, and so forth. Each of these features may be defined as a product. The **Mobile** product defines a primary relationship to the **Mobile** CFS. The **CallerID** product and the **Call_Waiting** product both define auxiliary relationships to the **Mobile** CFS.

- Exclusive relationships are used to model run-time constraints that ensure that entities are not shared with other service instances. For example, telephone numbers cannot be used by multiple instances of a mobile service. In this example, there is an exclusive relationship between the **Mobile** resource facing service (RFS) and the **TelephoneNumber** resource. The Exclusive type defines relationships among CFSs, RFSs, and resources.

- Shared relationships are used to model run-time constraints to ensure that source entities can be shared with other service instances. For example, because an HLR can maintain several user accounts simultaneously (and is not exclusive to any one service), there is a shared relationship between the **Mobile** RFS and the **HomeLocationRegister** resource.The Shared type defines relationships among customer facing services, resource facing services, and resources.

- Reference relationships are used to model run-time constraints to ensure that a target entity references a source entity. For example, you define a Reference relationship between a **Broadband** CFS and a **London** location (representing the geographical address). When defining Reference relationships, there is no source entity capacity requirement. The Reference type defines relationships among customer facing services, resource facing services, and resources.

- ConfigHierarchy relationships are used to model a specific run-time constraint in Oracle Communications Unified Inventory Management (UIM) in which no real entity is referenced. Instead, an intermediate hierarchical structure is referenced at run-time. You can use this new relationship to characterize the relationship between a resource facing service and a resource component. The ConfigHierarchy relationship indicates that a UIM realization of a resource component should result in a hierarchy of configuration items and should not generate a UIM entity.

About Actions

Design Studio includes the definitions for two action families, service and technical:

- Service Actions are the signatures of design operations that apply to customer facing services, resource facing services, or to resources. Actions are executed at run-time to initiate design and assign activities. Service actions are grouped into families that represent the range of operations that can be invoked on the entity, and each action consists of a specific set of parameters.

  These actions are used during product-to-service mapping and during run-time Calculate Service Order and Design and Assign provider function tasks. Service actions include a group of action codes, each of which can be performed against the associated specification. For example, a service action can affect change to a customer facing service because it includes the action codes **Add**, **Move**, and **Delete**.

- Technical actions are requests to downstream delivery systems to make changes in the network (the downstream delivery systems are represented by an application role). Technical actions are associated with resource facing services or with resources. Technical Action entities do not inherit the data elements defined on the associated specification because technical actions can include data elements defined on any entity in the conceptual model. Technical Action entities are realized as Design Studio for ASAP service actions (CSDLs) or as Design Studio for Network Integrity scan actions.

Conceptual model entities are the subjects of actions. You associate actions with entities to indicate that the action or group of actions can be performed against the associated entity. You can create your own actions, or you can configure Design Studio to create actions automatically when you create new entities.

All customer facing services are associated with one service action only. By default, when you create new customer facing services, Design Studio automatically creates a new Service Action entity and associates it with your new CFS. The new Service Action entity inherits all of the data elements defined on the CFS. This default inheritance enables you to keep the data synchronized between customer facing services and the service actions that are performed on the customer facing services.

Also, the service action includes a set of default data elements that are inherited from the associated functional area. These default data elements are associated with the **Implicit Parameter** tag (in the Functional Area editor) and they are not editable.

A service or technical action must have access to a specific set of data required to perform the action against an entity, and you identify which of the inherited data elements are applicable to the action. On the Action editor **Data Map** tab, you can specify whether a data element is required by an action by defining applicability to specific action codes (for service actions) or by defining applicability to specialized aliases defined for the action codes (for technical actions). For example, you can specify whether a data element value must be supplied to or returned by each action in an associated action family. You can overwrite the applicability settings for inherited data elements, and you can change the information defined for the data elements inherited from the functional area.

You can associate resource facing services and resources with one service action and with multiple technical actions. For example, a resource that represents customer premise equipment can be the subject an action that ships the equipment to a customer, and can also be the subject of an action that facilitates the return of the equipment.

Conceptual model service actions are realized as Design Studio for Inventory rulesets. Conceptual model technical actions are realized as Design Studio for ASAP service actions (CSDLs) or as Design Studio for Network Integrity scan actions.

About Conceptual Model Realization

In Design Studio, realization is the conversion of a conceptual model entity into an application entity. Conceptual model entities represent abstractions of services, so you cannot deploy conceptual model entities to run-time environments. Rather, you convert conceptual model entities into application entities, and then deploy the application projects that contain the realized entities to run-time environments.

You run design patterns to realize conceptual model entities. When creating a conceptual model entity, you identify which application entity it realizes and which design pattern performs the realization.

You realize the following conceptual model entities:

- Customer facing services and resource facing services, which are realized in Design Studio for Inventory projects as Service specifications and as Service Configuration specifications. See "About Realizing Services in Design Studio for Inventory" for more information.

- Resources, which are realized in Design Studio for Inventory projects as Inventory entities. For example, you can realize resources as Logical Device specifications, Logical Device Account specifications, Telephone Number specifications, and so forth. When realizing Resource entities, Design Studio also creates a configuration specification. See "About Realizing Resources in Design Studio for Inventory" for more information.

- Service actions, which are realized in Design Studio for Inventory projects as rulesets and Java code structures. However, design patterns do not automatically generate rulesets, so you must create the rulesets in an Inventory project. The Design Patterns and code generators for realizing Service Actions are packaged with the SNO Reference Implementation.

- Technical actions, which are realized in Design Studio for ASAP projects as service actions or in Design Studio for Network Integrity projects as scan actions. Realizations of signatures for other downstream delivery systems can be supported by crafting Design Patterns that support these delivery systems.

- Action parameter bindings, which are realized for Inventory projects as metadata that controls the behavior of Calculate Technical Order Functionality. The metadata generator for realizing Action Parameter Bindings is packaged with the SNO Reference Implementation.

- Locations, which are realized in Design Studio for Inventory projects as Place specifications defined as type **Site**. Design Studio also generates a corresponding Place Configuration specification. See "About Realizing Locations in Design Studio for Inventory" for more information.

- Fulfillment patterns, which are realized in Design Studio for OSM projects as fulfillment patterns.

About Design Patterns That Realize Conceptual Models

Design Studio design patterns are wizards that automate complex, repeatable tasks that team members can perform. Design Studio includes default design patterns that you can use to automate the realization of conceptual model entities into application entities. You run these design patterns using the Design Pattern wizard and you can set up conceptual model entities to run the design patterns automatically. You can also create your own design patterns. See "Working with Design Patterns and Guided Assistance" for more information about design patterns.

After you run a conceptual model entity's design pattern once, Design Studio automatically reruns design pattern when you make changes to the entity.

When a design pattern creates a realized application entity, the pattern uses the name of the conceptual model entity as a prefix for naming the realized application entity. For example, you create a customer facing service (CFS) named **Broadband** and then run the **Default Customer Facing Service Realization** design pattern for the **Broadband** CFS entity. When configuring the design pattern, you provide values for the **Major**, **Minor**, and **Fix** configuration version fields. For example, you may enter **1**, **0**, and **0**, respectively. When the pattern is run, Design Studio creates a Service Specification entity named **Broadband** and a Service Configuration Specification entity named **Broadband_Configuration_v1-0-0**.

> **Note:**
>
> See the Design Studio Help for more information about renaming conceptual model entities subsequent to application entity realization.

You can keep conceptual model entities and the realized application entities synchronized so that any changes you make to the conceptual model entities are automatically updated in the application entities. However, changes that you make to application entities are not automatically updated in conceptual model entities.

Design Studio conceptual model design patterns are associated by hierarchical relationships so that when you synchronize the conceptual model entity with the application entity, the design pattern automatically runs all relevant child design patterns to realize the entire conceptual entity tree.

For example, you define a CFS with two associated resource facing services (RFS), and each RFS includes three resources. The first time that you run the **Default Customer Facing Service Realization** design pattern, the Design Pattern wizard prompts you to input the information it requires to run the pattern. When the design pattern completes, Design Studio automatically runs the **Default Resource Facing Service Realization** design pattern twice (once for each RFS) and the **Default Resource Realization** design pattern six times (once for each resource in each RFS).

About Realizing Services in Design Studio for Inventory

When you realize customer facing services and resource facing services using the **Default Customer Facing Service** and the **Default Resource Facing Service** design patterns, the design patterns automatically create Design Studio for Inventory Service specifications and Design Studio for Inventory Service Configuration specifications. The design patterns save these specifications in the Inventory project that you specify when running the design pattern.

Only those data elements tagged as a **Characteristic** (at the Data Dictionary level) and as **Persisted** (on the conceptual model entity) are added to the Service specification during realization. When you add data elements to Resource Facing Service, Resource, and Location entities, Design Studio automatically applies the **Persisted** tag to data elements. When adding data elements to Customer Facing Service entities, you must apply the **Persisted** tag, if applicable.

You can also tag data elements as **Changeable** if you expect these elements to change frequently or if you need to track the life cycle of a service. When realizing customer facing services and resource facing services, the default design patterns save data elements that are tagged as **Changeable** to the Service Configuration specifications. Also, the design patterns save structured data elements to the corresponding Service Configuration specification. Data elements that are tagged as **Changeable** and **Persisted**, and structured data elements that are tagged as **Persisted** are added to the Service Configuration specification (as configuration items) during realization.

See the Design Studio Help for more information about using tags.

> **Note:**
>
> When realizing conceptual model entities for Inventory, ensure that all data elements defined on the conceptual model entity refer to a base element.
>
> For example, if on the conceptual model entity you define a structured data element with child data elements, ensure that the child data elements all reference a base element. During realization, the structured data element is added to the Configuration specification as a configuration item. Child structured data elements are added to the Configuration specification as child configuration items. Child simple data elements are added to the Configuration specification as characteristics.

About Realizing Service Components

When customer facing services and resource facing services define components, the **Customer Facing Service Realization** design pattern and the **Resource Facing Service Realization** design pattern create a configuration item for each component that resolves to a conceptual model entity that can be realized.

The default design patterns add the configuration item to the Service Configuration specification. The design patterns also do the following:

- Tag the configuration items as **Realization Item**, which identifies the element as data that is realized from a conceptual model specification to an Inventory configuration item.

- Add specification options to the configuration items. See *Modeling Inventory* in the Design Studio Help for more information about specification options.

- Assign the specification option **Item Option Type** value based on the relationship type defined between the conceptual model entity and the component. See *Modeling Inventory* in the Design Studio Help for more information about specification options.

- Remove orphaned data elements and add new data elements, configuration items, and specification options during synchronization.

For example, a **DSL_RFS** resource facing service can define simple data elements, as illustrated in Figure 6-7. The **UploadSpeed** simple data element is tagged as a **Characteristic**, as **Changeable**, and as **Persisted**:

Figure 6-7   DSL_RFS Simple Data Elements
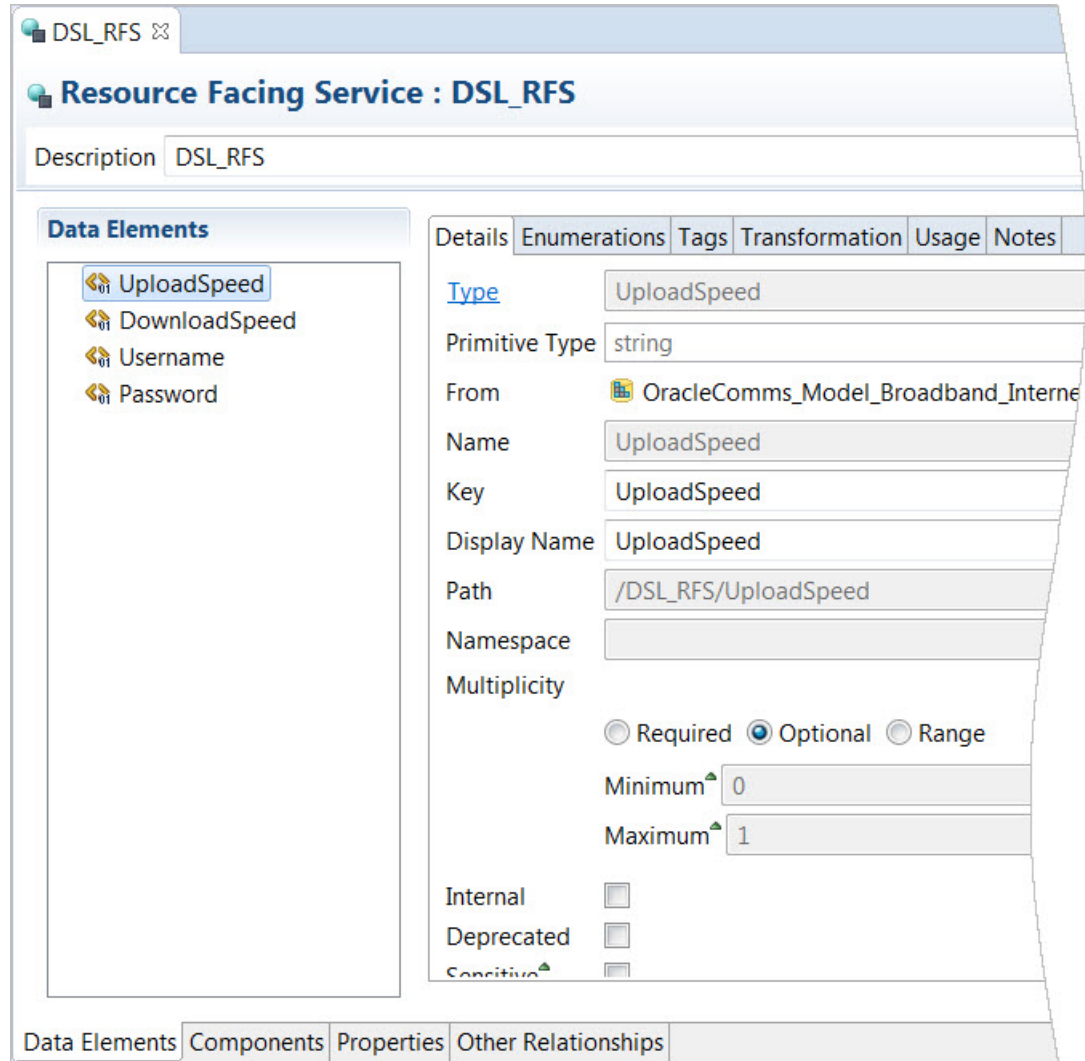
In this example, the **DSL_RFS** resource facing service also defines three resource components: **AAA_Account**, **CPE**, and **DSL_Interface**, as illustrated in Figure 6-8:

Figure 6-8   DSL_RFS Components
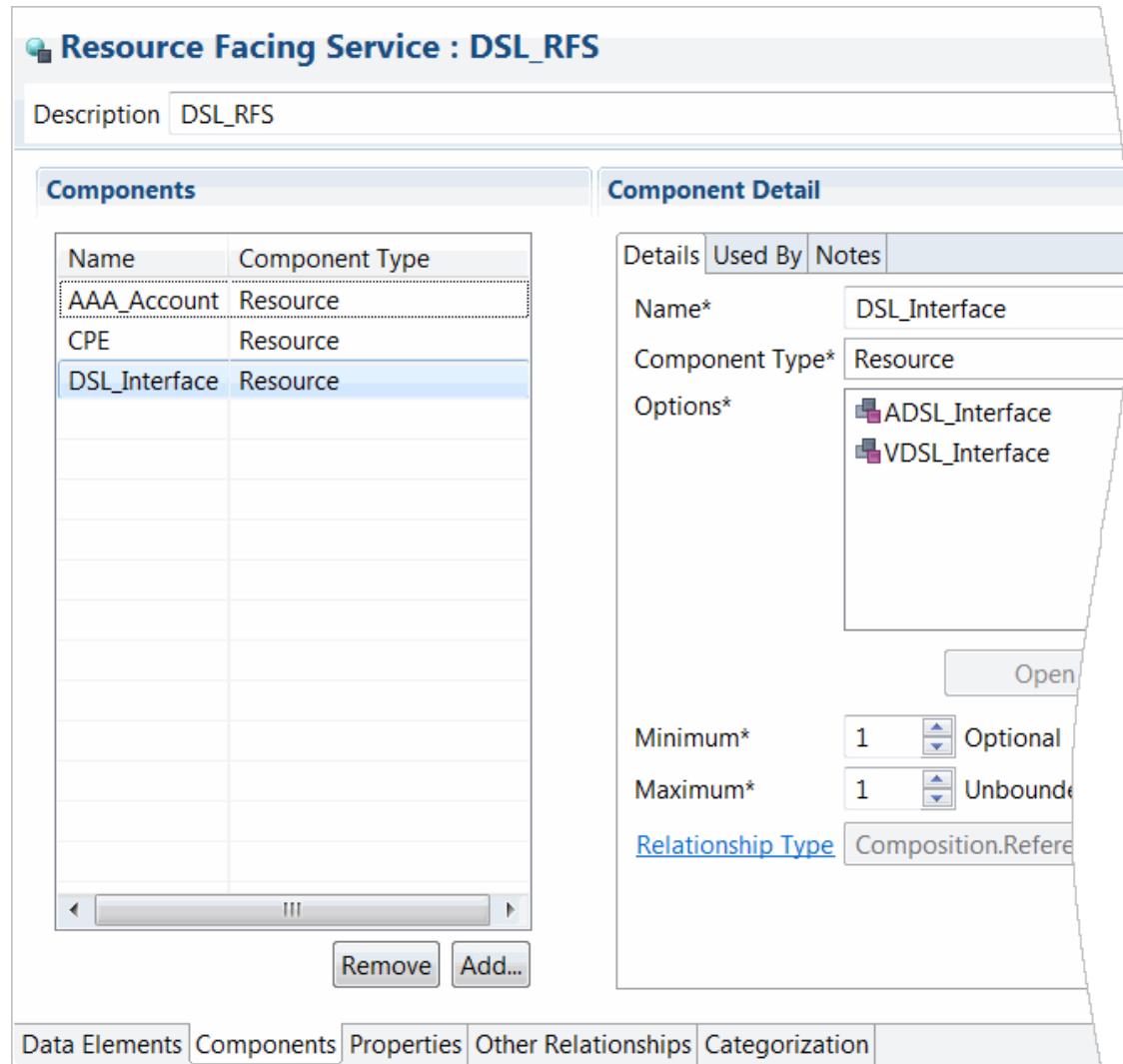
Figure 6-8 also shows that the **DSL_Interface** component is defined with two options, the **ADSL_Interface** resource and the **VDSL_Interface** resource. When you define multiple options for a component, you also define rules to determine which option is used at run time. You define these rules on the **Rules** tab of the Service Configuration specification.

When you realize the **DSL_RFS** service using the **Resource Facing Service Realization** design pattern, the pattern creates a Service specification and a Service Configuration specification. Data elements defined on the **DSL_RFS** service are saved to the Service Configuration specification if they are defined as **Changeable**. Additionally, the design pattern saves all structured data elements to the Service Configuration specification.

Resource components are added as configuration items on the Service Configuration specification. In this example, the resource components **AAA_Account**, **CPE**, and **DSL_Interface** are added as configuration items on the Service Configuration specification, as illustrated in Figure 6-9:

Figure 6-9   Configuration Items Defined on the Service Configuration Specification

When you run the **Resource Facing Service Realization** design pattern, you can select the **Create Nested Configuration Items** option to include a more detailed data tree on the service configuration. When you select this option, the components defined for resources are added to the service configuration as child configuration items. This configuration item hierarchy is the realization of all resources that are modeled as components on a resource facing service.

For example, the **DSL_RFS** resource facing service defines the **CPE** component. The **CPE** component defines a single option, the **DSL_CPE** component, as illustrated in Figure 6-10:

Figure 6-10   The CPE Component and the DSL_CPE Option
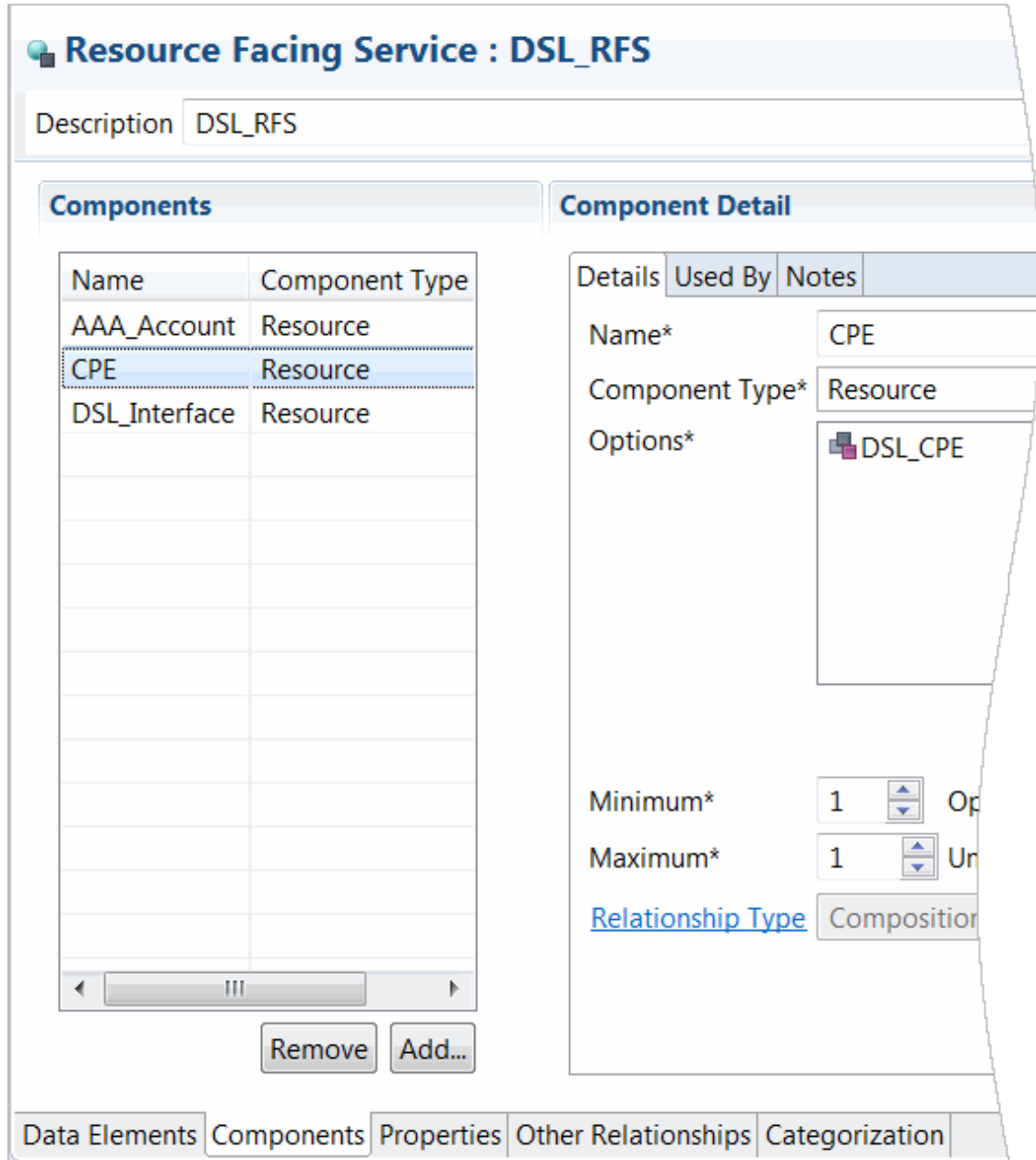
The **DSL_CPE** component defines a single option, the **VoD_Access** component, as illustrated in Figure 6-11:

Figure 6-11   The DSL_CPE Resource and the VoD_Access Component
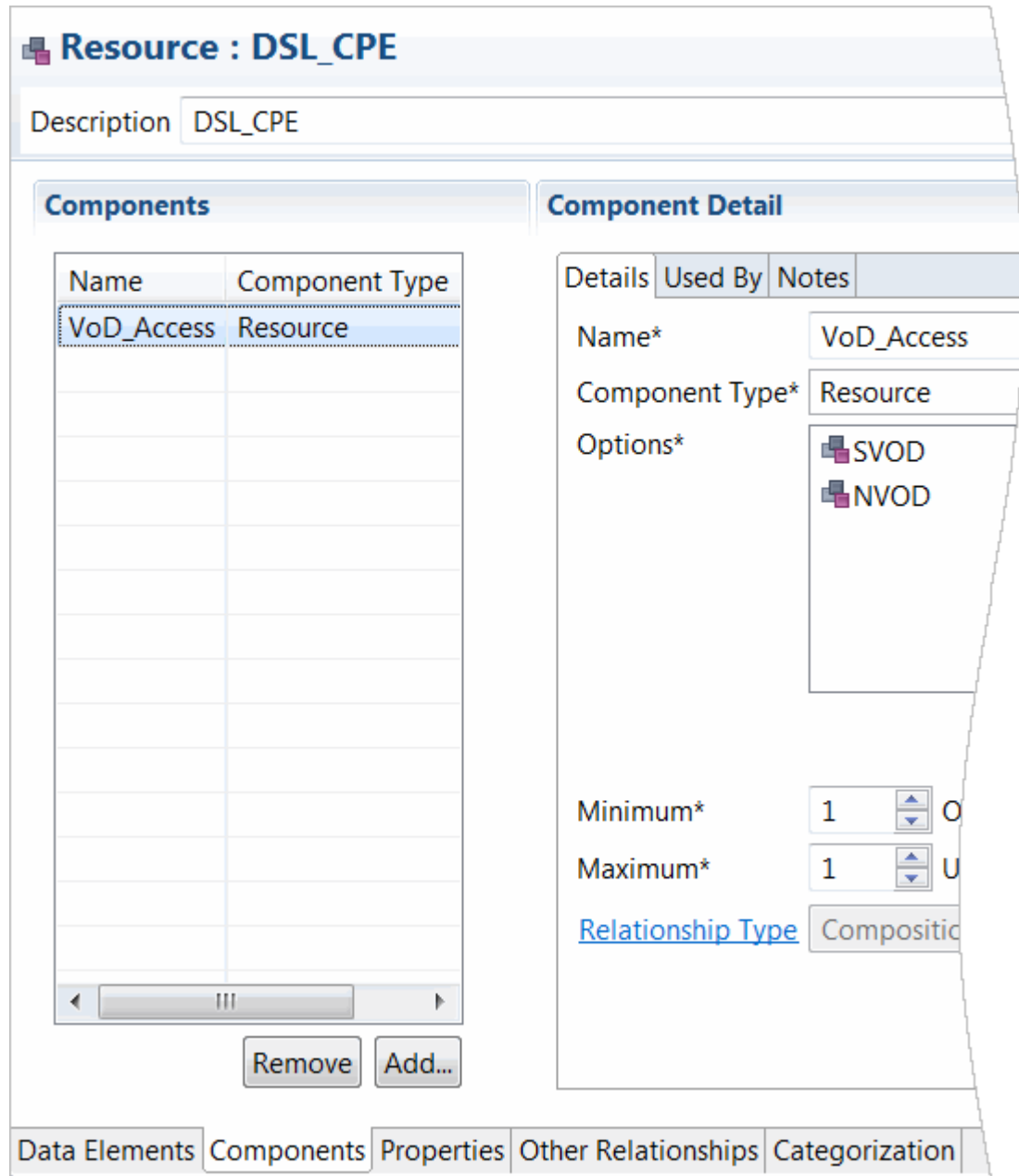
In this example, after the **Resource Facing Service Realization** design pattern is run with the **Create Nested Configuration Items** option selected, the design pattern adds to the **DSL_RFS** Service Configuration specification a child configuration item for the video on demand component, **VoD_Access**, as illustrated in Figure 6-12:

Figure 6-12   Resource Attributes and Components Added as Child Configuration Items

Figure 6-13 shows how the **DSL_CPE** resource realization is configured (in the **Realization** area) and shows the realized application entity, an Inventory Logical Device specification named **DSL_CPE** (in the **Realized By** table):

Figure 6-13   DSL_CPE Properties
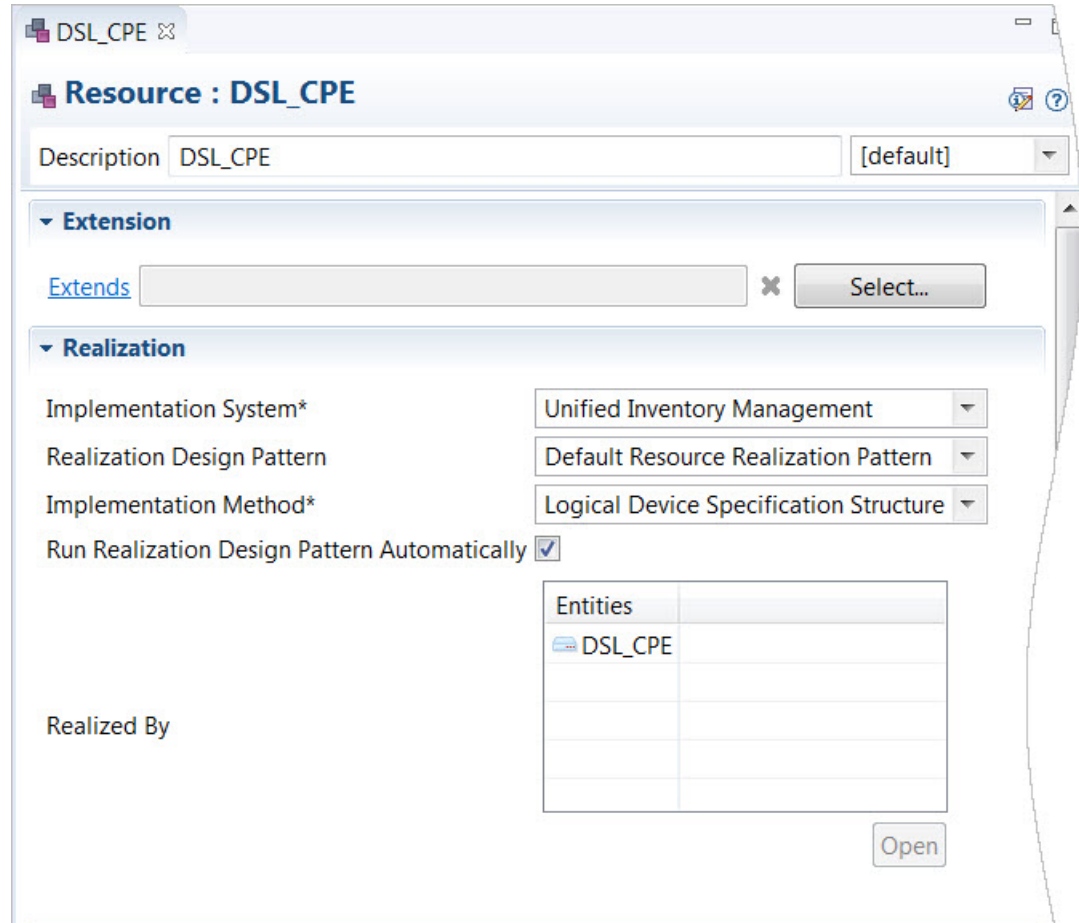
The **DSL_CPE** Logical Device specification appears as a specification option for the **CPE** configuration item on the Service Configuration specification, as illustrated in Figure 6-14:

Figure 6-14   Specification Options Defined for Configuration Items
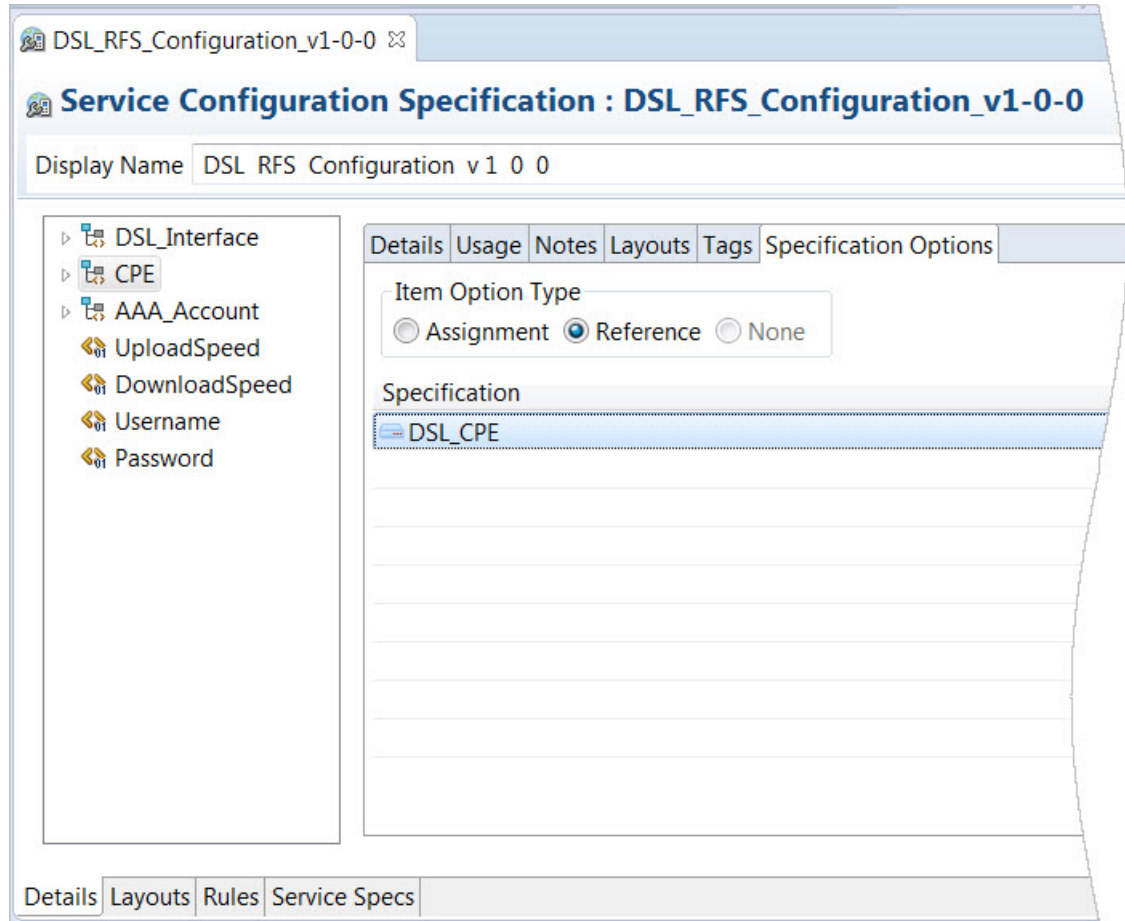
About Realizing Resources in Design Studio for Inventory

When you create a conceptual model Resource entity, you define how the resource is realized in Inventory by selecting an Inventory resource as the implementation method. When you realize resources using the **Default Resource Realization** design pattern, the design pattern creates an Inventory entity specification and creates a corresponding Configuration specification.

When you realize resources with one of the following implementation methods, the design pattern creates the Inventory entity and a corresponding Configuration specification:

• Logical Device Account

• Logical Device

• Flow Interface

• Network

• Pipe

When you realize resources with one of the following implementation methods, the design patterns creates an Inventory entity, a Configuration specification of type Logical Device, and a Logical Device specification container that includes the name of the resource:

• Connectivity

- Customer Network Address

- Custom Object

- Device Interface

- Flow Identifier

- IPv4Address

- IPv6Address

- Media Stream

- Property Location

- Telephone Number

The default design pattern saves data elements tagged as **Characteristic** (at the Data Dictionary level) and as **Persisted** (on the conceptual model entity) to the specification, and saves data elements that are tagged as **Characteristic**, **Persisted**, and **Changeable** to the Configuration specification. Also, the design pattern saves structured data elements tagged as **Persisted** to the Configuration specification (as configuration items).

You can define the implementation method as a resource that exists outside of Inventory if the entity must realize as a resource that is different from the available realization options. When you select this option, there is no design pattern that generates a realization entity in Design Studio for Inventory (you can, however, write your own design pattern). See *Design Studio Developer's Guide* for more information about developing design patterns.If you elect to create nested configuration items when running the **Default Resource Facing Service Realization** design pattern, resource components and their data elements are saved to the Service Configuration specification as child configuration items. Design Studio adds structured data elements to the Service Configuration specification and adds simple data elements to the Service Configuration specification if they are defined with the **Changeable** tag.

About Realizing Locations in Design Studio for Inventory

A location represents a geographical place that is relevant to services or resources. The **Default Location Realization** design pattern generates a Place specification defined as type **Site** and a generates a corresponding Place Configuration specification. The **Location Realization (deprecated)** design pattern generates a Place specification of type **Address**. To generate a Place specification defined with a different type, you must navigate to the realized entity and change the entity type, or create your own design pattern.

The **Default Location Realization** design pattern saves data elements tagged as **Characteristic** (at the Data Dictionary level) and as **Persisted** (on the conceptual model entity) to the Place specification, and saves data elements that are tagged as **Characteristic**, **Persisted**, and **Changeable** to the Place Configuration specification. Also, the design pattern saves structured data elements tagged as **Persisted** to the Place Configuration specification.

About Realizing Technical Actions in Design Studio for ASAP

Service Action entities in Design Studio for ASAP are realized from conceptual model Technical Action entities.

Technical action families include a set of action codes that are mapped to a corresponding set of specialized action names that are specific to an application role. The action codes that initially appear are those defined by the action type (or the **Technical** functional area).

For example, Figure 6-15 shows a technical action named **TA_ACT_DSL**. This technical action includes the action codes **Create**, **Modify**, and **Remove**, which are mapped to the specialized

action names **Activate**, **Alter**, and **Deactivate**. The specialized action names are specific to the **Activation** application role.

> **Note:**
>
> Technical Action Generation design pattern may not work in end-to-end scenario because of the naming and the action code discrepancies.

Figure 6-15   Technical Action Editor Action Codes Tab
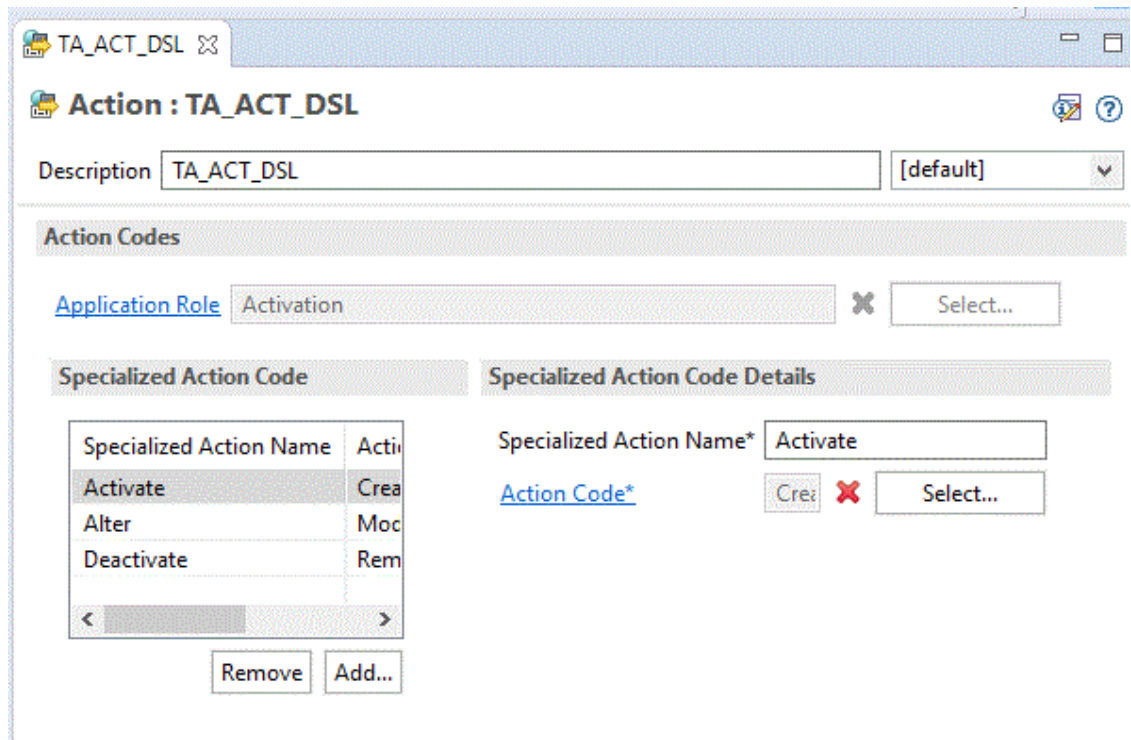
After you run the **Activation Technical Action Realization** design pattern, keep the conceptual model Technical Action entities and the ASAP Service Action entities synchronized by running the **Activation Service Action Synchronization** design pattern. Figure 6-16 illustrates how to set up a technical action that is realized by an ASAP service action.

Figure 6-16   Technical Action Editor Properties Tab
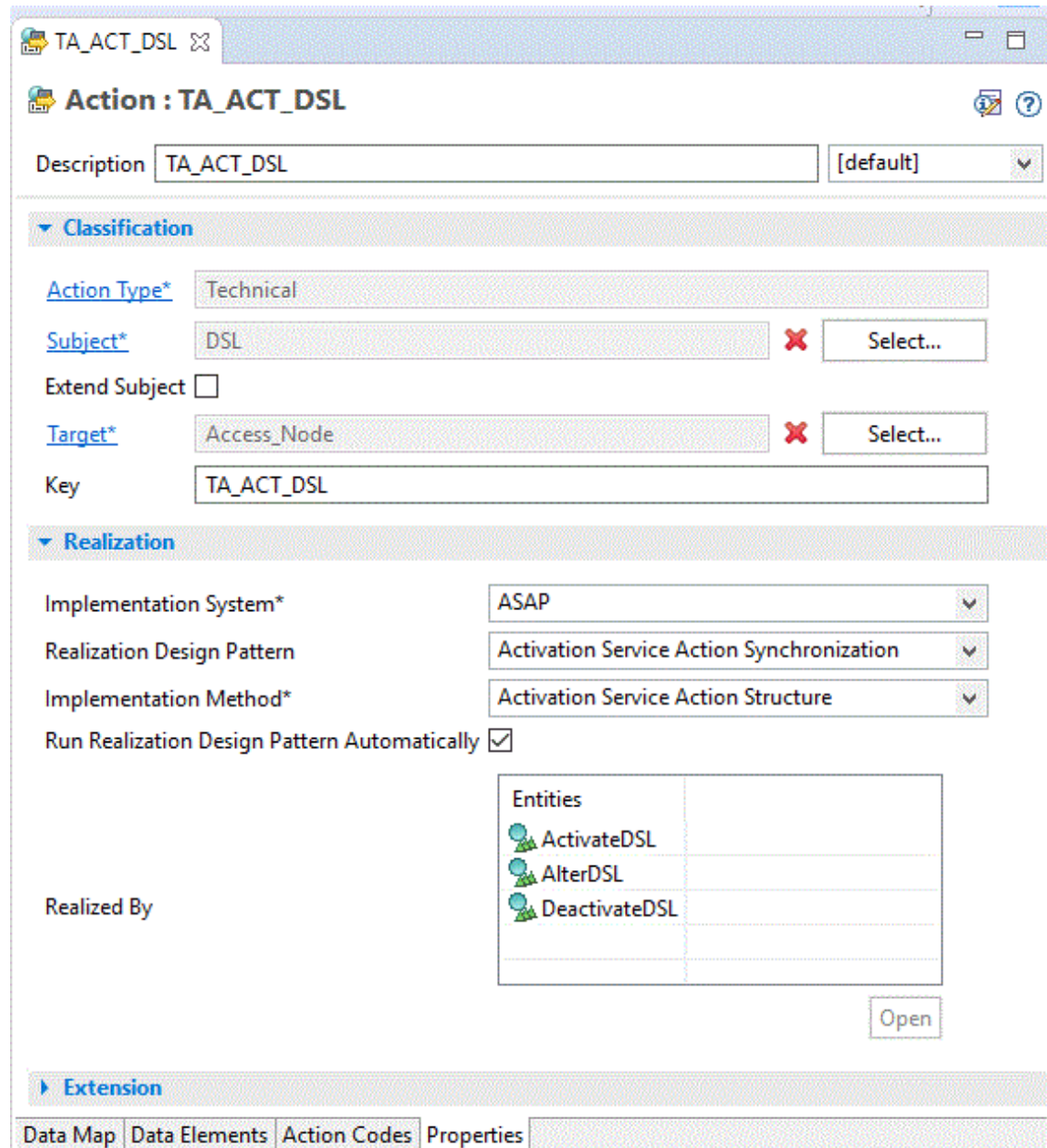
After you complete the realization, the Solution view displays the entities and the relationships among them.

About Conceptual Model Synchronization

You synchronize conceptual model entities with application model entities to ensure that the configuration of the application entities and the related conceptual model entities remain aligned.

After you realize conceptual model entities into application-specific entities, you can enrich the application entities with data required by the application. During this process, you may be required to change the entities in the application project. For example, you may need to add

new characteristics to a Service specification that is realized from an RFS. If the data that you add is required by multiple entities across applications, Oracle recommends that you make the change to the associated conceptual model specification and then synchronize the projects.

When you synchronize conceptual model and application projects, Design Studio does not modify or delete the data elements that you defined on the application entities. During synchronization, Design Studio recreates in the application entities only the data elements and configuration items inherited from the conceptual model entities.

You synchronize conceptual model entities with application model entities by re-running the realization design patterns. You can configure these design patterns to run automatically (when an entity is saved), or you can run the design patterns manually by selecting a pattern from a context menu. Also, you can perform a bulk update by running a design pattern for an entity and instructing Design Studio to run all required design patterns for all child entities in the hierarchy.

Before you can synchronize a conceptual model entity and an application entity, you must realize the conceptual model entity in an application project. You provide data in the Design Pattern wizard that Design Studio uses to generate the application-specific configuration. To keep the entities synchronized, Design Studio re-runs the design pattern in the background using the data that you previously specified.

If you do not want to synchronize automatically, you can synchronize manually. For example, if you need to track all changes to conceptual model entities and application model entities, you can disable automatic synchronization until you have finished the analysis and determined the impact of the changes. When finished, you can select **Synchronize All** from the context menu to perform a bulk update on a hierarchy of conceptual model entities. See the Design Studio Help for more information.

You can synchronize the following conceptual model entities with their associated application entities:

- Customer Facing Service

- Resource Facing Service

- Resource

- Location

If you rename a conceptual model entity, and you subsequently use the Design Studio synchronize feature to ensure that the configuration of the application entities and the related conceptual model entities remains aligned, Design Studio looks for a realized application entity with the same name as the conceptual model entity. If no application entity with the same name exists, Design Studio creates a new application entity. In this scenario, Design Studio creates two realized entities, one created initially with original name, and one created after synchronization with the new name.

To avoid creating multiple realized entities after renaming a conceptual model entity, you must manually update the realized application entity names before using the Synchronize feature. See the Design Studio Help for more information about renaming conceptual model entities.

About Synchronization Records

When you run design patterns, Design Studio saves the values that you supply in a synchronization record. Design Studio uses these values to re-run the design pattern automatically (if you have defined the conceptual model entity to run automatically) and to keep the conceptual model entities synchronized with the realized application entities.

Design Studio saves synchronization records in the **synchronizationRecords** folder, which is viewable only from the Package Explorer view. You can open any **.SyncRecord** file in the Synchronization Record editor if you need to correct invalid data in the record.

If a valid synchronization record does not exist for a conceptual model specification, the **Synchronize** and **Synchronize All** context menu options are not available when you right-click the specification in the Solution view.

If a valid synchronization record exists for a conceptual model entity and you re-run the design pattern for that entity, Design Studio uses the information in the synchronization record to pre-populate the values in the Design Pattern wizard.

About Importing Conceptual Model from External Catalogs

You can import conceptual models from external catalogs using Exchange Format XML. The Exchange Format XML file is a consistent representation of Design Studio entities and external catalog system entities. You can import multiple entities across multiple projects with a single input XML file.

The import operations do not update sealed projects, read-only projects, or read-only entities.

You can import the following conceptual model entities, along with supporting data elements, data schemas, and model projects, into Design Studio:

- Product
- Customer facing service
- Resource facing service
- Resource
- Service action
- Technical action

You can perform a partial or complete import of the XML file.

> **Note:**
>
> **Partial** import is the default option.

A partial import:

- Creates new projects, entities, and elements if they are not present in the workspace
- Appends new information and updates the existing information for entities or elements
- Leaves existing information on entities and elements, if information is not included in the import file
- Renames the existing entities or elements if their names are changed and the IDs are not changed, in the input file
- Renames the manually created entities or elements if new names are provided in the input file and the IDs in input file are same as the IDs in **.studioModel** exchange format export file

> **Note:**
>
> Partial import does not support the removal of information from existing entities or elements.

A complete import:

- Creates new entities or elements if they are not present in the workspace

- Replaces existing information on entities and elements with information that is provided in the import file

- Removes information from existing entities or elements if that information is not included for these entities or elements in the input file

- Renames the existing entities or elements if their names are changed and the IDs are not changed, in the input file

- Renames the manually created entities or elements if new names are provided in the input file and the IDs in input file are same as the IDs in **.studioModel** exchange format export file

- Deletes the existing elements that are created manually or through import, if their information is missing in the input file

- Deletes the existing entity references (such as components to CFS/RFS/Resource, other reference relations to CFS/RFS/Resource) that are created manually or through import, if their information is missing in the input file

- Handles enumerations as follows:

  – Removes existing information of entities or elements if they are missing from the input file

  – Updates existing entities or elements with information that is modified in the input file

- Handles non-list or elements (for example, Copyright information, project name) as follows:

  – Defaults existing entities or elements with information that is missing from the input file

  – Updates existing entities or elements with information modified in the input file

> **Note:**
>
> Unique external identifiers must be provided for new instances of projects, entities, and elements that are imported.
>
> Identifiers for existing or imported projects, entities, and elements cannot be modified by import operations.

About the Common Model Base Data Project

To build a representation of a service domain in Design Studio, you must first generate the Common Model Base Data project using the Common Model Base Data design pattern. The Common Model Base Data project contains predefined rules and data for processing the entities and actions in your conceptual model, such as action codes, relationship rules, and

entities that support conceptual modeling. This data is foundational to the conceptual model design required for service fulfillment solutions.

See the Design Studio Help for information about generating the Common Model Base Data project.

About Conceptual Models and Service Order Fulfillment

This section describes how a Design Studio conceptual model contributes to each layer of service fulfillment run-time order orchestration.

A typical service order includes a service request by a customer (for example, for Broadband service with 50 Mbps download speed and 25 Mbps upload speed). When the order is received, the CRM system creates a service request and sends the request to an order and service management orchestration system. The order and service management orchestration system identifies and configures the service components that are required to fulfill the order (for example, configuring access from the customer premise equipment to the network, dispatching technicians to perform physical work on the network, executing actions in the network to deliver the service, and so forth). The service is subsequently delivered to the customer, and the customer sends payment for the service.

The orchestration of a service order spans the following run-time business processes:

- Central Order Management, which includes the processes required to capture the order data and send the data to an order and service management system. See "Conceptual Models and Central Order Management" for more information.

- Service Order Management, which includes the processes required to manage and provision the service order. See "Conceptual Models and Service Order Management" for more information.

- Technical Order Management, which includes processes required to run the technical actions in the network. See "Conceptual Models and Technical Order Management" for more information.

Conceptual Models and Central Order Management

At the Central Order Management layer, order data is captured and sent to OSM. This layer includes processes that transform the customer order into a service order. During this transformation, the actions requested against the products are transformed into actions to be performed against services. This layer also includes processes that submit requests to the service order management layer to process the service order and processes that enable the service order management layer to communicate order updates to the order capture systems (for example, the order is complete, the order failed, and so forth).

To support the Central Order Management layer, Design Studio solution designers must define in the conceptual model:

- The mapping rules for all products in each domain. The mapping rules define how actions defined against products on a customer order are transformed into actions defined against services on service orders. Solution designers also map customer facing services to specific instances of Service Order Management (for example, to mobile services or to fixed services).

- The order item parameter bindings for each product supported in the conceptual model. The order item parameter binding specifies where to find on an incoming order line the parameter data that matches the conceptual definition. At run time, the order item parameter bindings validate the sales order line input against the product definition.

Conceptual Models and Service Order Management

The Service Order Management layer includes processes that:

- Identify the technical components required to deliver service

- Determine the technical actions to be performed against the technical components

- Submit requests to the Technical Order Management layer to process those actions

- Send order updates to the Central Order Management layer and receive updates from the Technical Order Management layer

To support the Service Order Management layer, Design Studio solution designers must first build a representation of the service in the conceptual model to enable the run-time processes to determine the technical components required to provision a service. Solution designers create the customer facing services, resource facing services, and resources; define the service actions on the customer facing services; convert the conceptual model into application projects; and enrich the conceptual model data with application-specific data.

> **Note:**
>
> You program service actions in Design Studio for Inventory using Java and rulesets.

Also, to determine the technical actions required to provision a service, a solution designer must define which resources are the targets of the actions and define the data required to perform the actions.

Finally, solution designers create order item parameter bindings for each service action in the conceptual model. The order item parameter binding specifies where to find on an incoming order line the parameter data that matches the conceptual definition. At run time, the order item parameter bindings validate the provisioning order line input against the service action definition.

Conceptual Models and Technical Order Management

The Technical Order Management layer is a collection of processes required to process the technical actions in the network. This layer includes the processes that ship resources to customers, request services from partners, dispatch technicians to perform manual work in the network, and run commands on network devices to activate the services.

To determine the commands required for the network devices, solution designers define the technical actions for RFS and resources in Design Studio conceptual models.

Additionally, solution designers create order item parameter bindings for each technical action in the conceptual model. The order item parameter binding specifies where to find on an incoming order line the parameter data that matches the conceptual definition. At run time, the order item parameter bindings validate the technical order line input against the service action definition.

7

Design Studio Packaging and Integrated Cartridge Deployment

This chapter provides Design Studio information about building, packaging, and deploying cartridge projects to environments. Additionally, this chapter describes tools and processes that you can use to prepare your solutions for a production environment.

About Packaging and Cartridge Deployment

When building, packaging, and deploying cartridge projects:

- Oracle recommends that you review *Design Studio Developer's Guide* and the developer's guide for each Oracle Communications application in your solution. These guides provide information about packaging and cartridge development.

- You can automate the process of building and packaging applications. See *Design Studio System Administrator's Guide* for more information about automating build processes.

- Package projects to facilitate the import of solution components into the Design Studio workspace and the deployment of an OSS solution.

- Create a model project to contain all simple data elements and structured data elements that you use in multiple applications. Package these model projects separately from your application-specific cartridge projects (for example, Inventory, OSM, ASAP, and Network Integrity projects) and define project dependencies to use these model definitions.

- Group all cartridge projects by application for ease of maintenance.

- Create separate cartridge projects for content that is not specific to a solution domain. This enables reuse.

- When defining entities, ensure that you organize them such that they do not create any cyclic dependencies when defining project dependencies.

- When organizing application-specific components, consider that the requirements may differ among applications. For example:

  – For Oracle Communications Unified Inventory Management (UIM) solution components, you can package Service, Resource, and Infrastructure specifications in different cartridge projects and define project dependencies accordingly. You can package multiple UIM cartridges together and deploy them collectively. See *UIM System Administrator's Guide* for more information about grouping and deploying multiple cartridges.

  – For OSM solution components, you can package cartridge projects based on the function they perform. For example, you can separate cartridge projects containing service orders from those that contain technical orders. You can divide the cartridge projects for each function into smaller component cartridge projects. You can assemble component cartridge projects serving a particular function in a composite cartridge project. Composite cartridge projects simplify the deployment of OSM cartridges, because when you deploy a composite cartridge, Design Studio automatically deploys all included component cartridges, as well.

Collaborating in Teams

Because solution development workflow among project team members is complex, Oracle recommends the following practices to facilitate the editing and sharing of solution components.

Using Software Configuration Management Systems

Use a software configuration management (SCM) system to coordinate concurrent revisions and establish baselines of software. A baseline is a snapshot of the state (a particular revision) of all artifacts contributing to the overall solution at a particular milestone in the project schedule, as the team works iteratively and incrementally toward completion.

The Eclipse platform provides support for SCM systems. Two examples of SCM systems are Subversion and Git. You can install the plug-ins for Subversion or Git (Subclipse and Egit, respectively) in Design Studio for either one of these SCM systems, or use a different SCM system that is supported by Eclipse. See the Eclipse Help for more information about using Subversion or Git with Eclipse.

> **Note:**
>
> Do not check into an SCM repository any artifacts in the **bin**, **cartridgeBuild**, and **out** directories of any cartridge project. These directories contain artifacts that are generated temporarily during builds and do not need to be stored.

See *Design Studio Developer's Guide* for more information about working with source control.

Using Continuous Integration

Design Studio enables you to implement continuous integration software development. Continuous integration employs processes that allow you to continually verify software quality.

In a continuous integration development environment, developers check-in code, and that code is picked up by automated builds. Metrics on code quality are gathered (based on industry standard rules or custom defined rules) and the metrics are made available to a management team through a version of management reporting.

Continuous integration offers many advantages for a project team, where each member is contributing components that must function in close collaboration with components developed by others. Continuous integration processes help discover when these components stop functioning together.

Building Projects

Individual developers can perform a build locally (in their own Design Studio workspace) and check-in the resulting cartridge binary to the SCM repository to share the new build with the team. However, Oracle recommends enabling a continuous integration server to perform automated builds using source artifacts retrieved directly from the SCM repository. You can run Design Studio using Apache Ant to facilitate automated builds.

> **Note:**
>
> Oracle recommends that solution developers disable the **Build Automatically** setting in the **Project** menu in Design Studio. Initiate builds explicitly, when you need them, by cleaning the project. The **Clean** build option enables you to run a build only when a set of coordinated changes are ready to deploy and test. See Design Studio Help for more information about these options.

See *Design Studio System Administrator's Guide* for more information about automating builds. See the Design Studio Help for more information about building and packaging projects.

Setting Up Integration Test Environments

Oracle recommends setting up an integration test environment, to continuously deploy solution components and test them together in integrated scenarios. Integration test environments facilitate discovery of incompatibilities, breakage, and errors related to component interaction and collaboration. See "Testing Design Studio Solutions" for more information.

Using Continuous Integration Systems

You can use a continuous integration system like Hudson to automate builds and test continuously. Hudson makes it easier for developers to integrate changes to a project and provides a way for various teams to frequently obtain fresh builds. Hudson supports software like Apache Subversion and Git, and can generate a list of changes made into the build from the VCS system. Hudson also executes Apache Ant and Apache Maven based projects, as well as arbitrary shell scripts and Windows batch commands.

Communicating Changes

Because changes to solution components occur during solution development, it is essential to understand the impact of any changes on other solution components and communicate them to other team members. When making changes, consider the following:

- Changes made to a common data model, such as removing simple or structured data elements or updating their definitions, may impact application-specific cartridge projects that have a project dependency on the model project.

- Changes made to a data model defined in an application-specific cartridge project may impact other cartridge projects that share those definitions.

- Changes that affect the content of a request or response of web service operations used for integration between applications (such as capture interaction, process interaction, createOrderByValue, and so forth) can impact other solution components.

Working with Design Studio Builds

Builds are processes that update existing resources and create new resources. You run builds against projects to create or modify workspace resources. The type of project determines the type of build. For example, when you run a build for a Java project, the build converts each Java source file (**.java** files) into one or more executable class files **(.class** files).

You run build processes against projects to create or modify workspace resources. There are two kinds of builds:

- Incremental builds, which affect only the resources that have changed since the previous build was computed.
- Clean builds, which affect all resources.

There are two ways to run builds: automatically and manually.

- Automatic builds are always incremental and always affect all projects in the workspace.
- Manual builds can be incremental or clean, for specific projects or for the entire workspace.

Build processes detect errors in your projects. You must resolve all errors in a cartridge project before you can deploy it to a run-time environment.

About Incremental Builds

By default, incremental builds are performed automatically when you save resources. To ensure that builds are performed automatically when you save resources, you must confirm that the Build Automatically option is selected (you can access this option from the **Project** menu).

You can disable automatic building and manually invoke incremental builds, if, for example, you want to finish implementing cartridge changes before building the project.

To manually run incremental builds on projects, disable **Build Automatically**. When you disable **Build Automatically**, the incremental build options become available. You can:

- Select **Build All** to build all projects in the workspace.
- Select **Build Project** to clean a specific project.

These options affect only the resources that have changed since the last build. You can also manually run clean builds against specific projects or against all projects in the workspace.

About Clean Builds

You run clean builds by selecting **Clean** from the **Project** menu. In the Clean dialog box, you can clean all projects in the workspace or limit the clean and build to a specific project or group of projects. Additionally, you can start a build immediately if you want to clean and build the selected projects in a single step.

About the Design Studio Builder Process

The Design Studio Builder process generates several artifacts automatically every time you create a cartridge project or make a change to a cartridge project. You can access the following from the Package Explorer view of the Java perspective or from the Navigator view of the Resource perspective:

- The cartridge archive file that Design Studio sends to the run-time server when deploying a cartridge. The file is located in the **cartridgeBin** directory.
- A pre-compressed version of the cartridge archive file, which contains all folders, subfolders, and files in the archive. This directory is contained in the **cartridgeBuild** directory. When you make a change to a cartridge, the Builder process makes changes in the **cartridgeBuild** directory, then it builds the file.

> **Note:**
>
> The Builder process is automated; consequently, you should not make any changes in the **cartridgeBuild** or **cartridgeBin** directories (any changes you make will be overwritten) or check these directories into source control.

Working with Integrated Cartridge Deployment

After building and packaging Inventory, OSM, Activation, and Network Integrity cartridge projects, you can deploy them from Design Studio to test environments. Design Studio-integrated cartridge deployment enables you to manage cartridges in the test environment consistently, manage common test environment connection parameters across the design team, and compare cartridge version and build numbers in the development environment with those of the cartridges deployed in the test environment.

You manage the cartridge project variables and the system parameters for run-time application instances in environment projects. You can save these environment projects in source control and share them among solution designers.

About Cartridge Deployment

The following tasks are done once before deploying cartridge projects from Design Studio. The information can then be shared among team members:

1. Create an Environment project.
2. Create a new Environment entity to contain the connection parameters necessary to connect to the test environment.
3. Query the test environment to determine what's already on the test environment server.
4. Define any environment-specific variables for the test environment.

The final two steps in the deployment process are to deploy and test your cartridges. You perform these steps iteratively, in combination with making incremental design improvements or updates and building the cartridge projects as necessary.

About the Environment Perspective

The Environment perspective is a collection of views that enable you to create and manage the attributes associated with your environment. You use the Environment perspective to deploy and undeploy cartridge projects to one or more test environments.

Two important components of the Environment perspective are the Studio Environment Editor and the Cartridge Management view. The Cartridge Management view displays the selected environment information obtained from the most recent queried state. See "About the Cartridge Management View" and "About the Studio Environment Editor" for more information.

Design Studio requires a WebLogic user name and password from any person attempting to deploy to an environment (to protect against unauthorized access to environment servers). Design Studio collects authentication details for connection when required, and securely disposes this information when the application closes.

See *Design Studio System Administrator's Guide* for information about setting up users for Design Studio deployment.

Figure 7-1   Environment Perspective
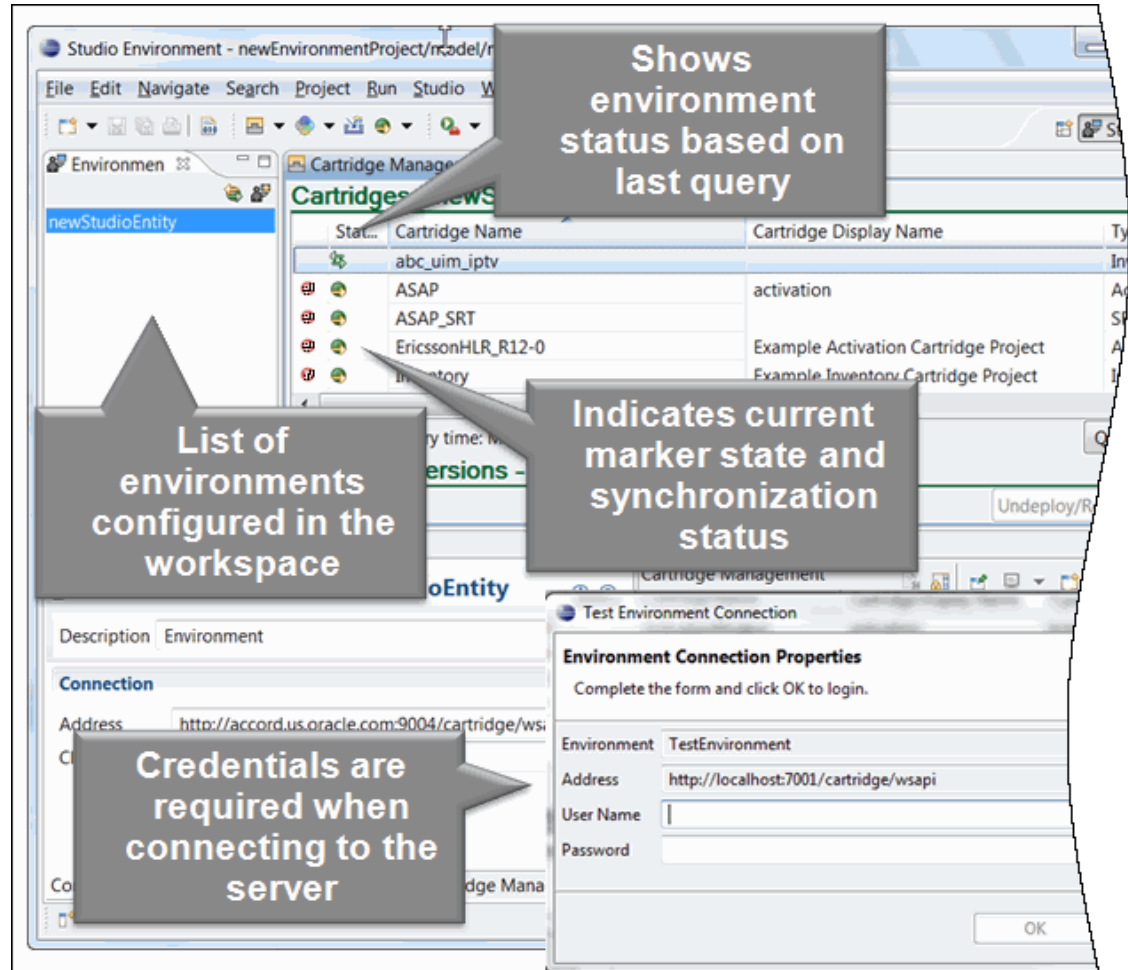
About the Cartridge Management View

After you query the test environment, you can use the Cartridge Management view to review and manage your cartridge project deployments. The Cartridge Management view is a dashboard. It facilitates the deployment process and enables collaboration among team members by displaying all of the cartridge projects in the workspace and all of the cartridges deployed to the server. For example, if you're uncertain whether to deploy your piece of the solution, you can query an environment to see what versions of a cartridge other team members have deployed.

The Cartridge Management view includes a status column to indicate which cartridges have been deployed and, if so, whether they are synchronized with the target environment. You use the Cartridge Management view to deploy cartridges in the Design Studio workspace and undeploy them from run-time environments.

The Deployed Versions table lists which cartridge version and build combination is currently deployed in the target environment (for the selected cartridge). The last refresh time appears at the bottom of the table. Design Studio refreshes the table after cartridge queries, imports, deploys, and undeploys.

Figure 7-2   Cartridge Management View
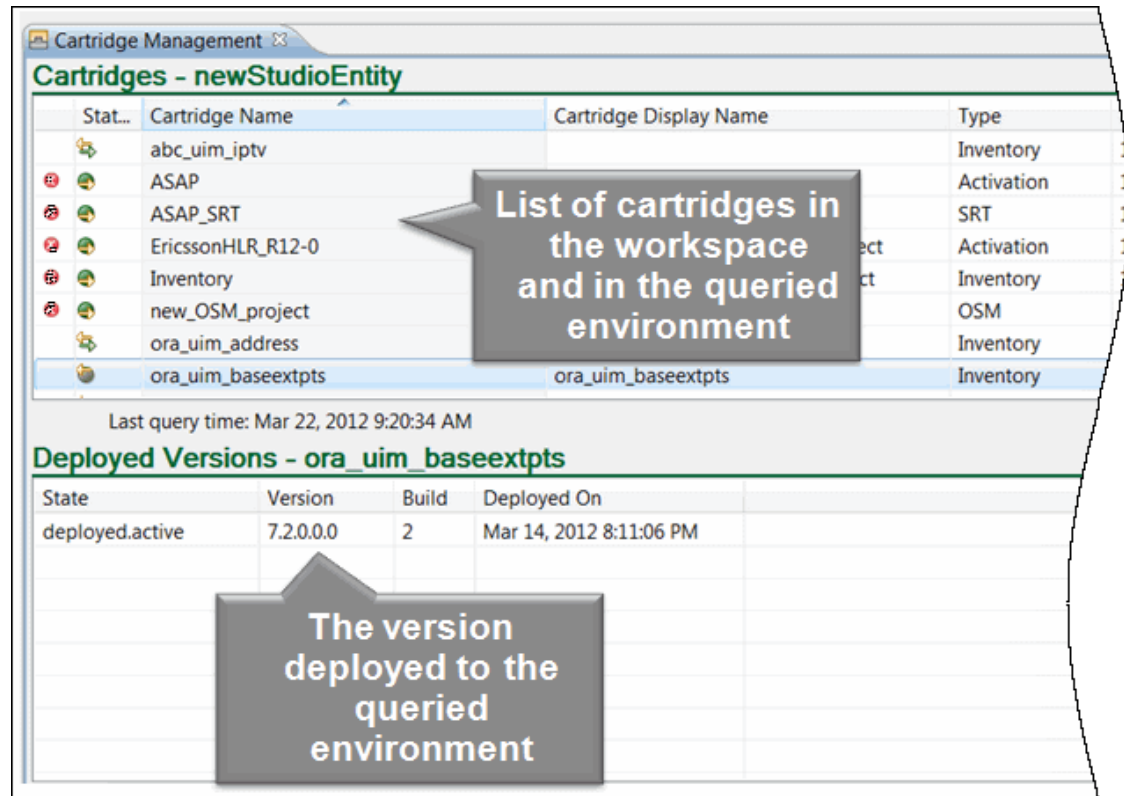
Deployment Synchronization States

The Cartridge Management view includes reconciliation details of environments and the present workspace. When you query an environment, Design Studio displays the cartridges in the workspace, those on the server, and whether the cartridges are synchronized.

The synchronization states are:

• **In-synch**, meaning that the cartridge exists in the workspace and in the run-time environment, and the versions are identical.

• **Out-of-synch**, meaning that the cartridge exists in the workspace and in the run-time environment, but the versions are not synchronized.

• **On server, not in workspace**, meaning that the cartridge exists in the run-time environment but it does not exist in the workspace.

• **In workspace, not on server**, meaning that the cartridge exists in the workspace but it does not exist in the run-time environment.

State information is updated each time a cartridge management action occurs or when you explicitly query the environment.

Figure 7-3   Cartridge Deployment Synchronization
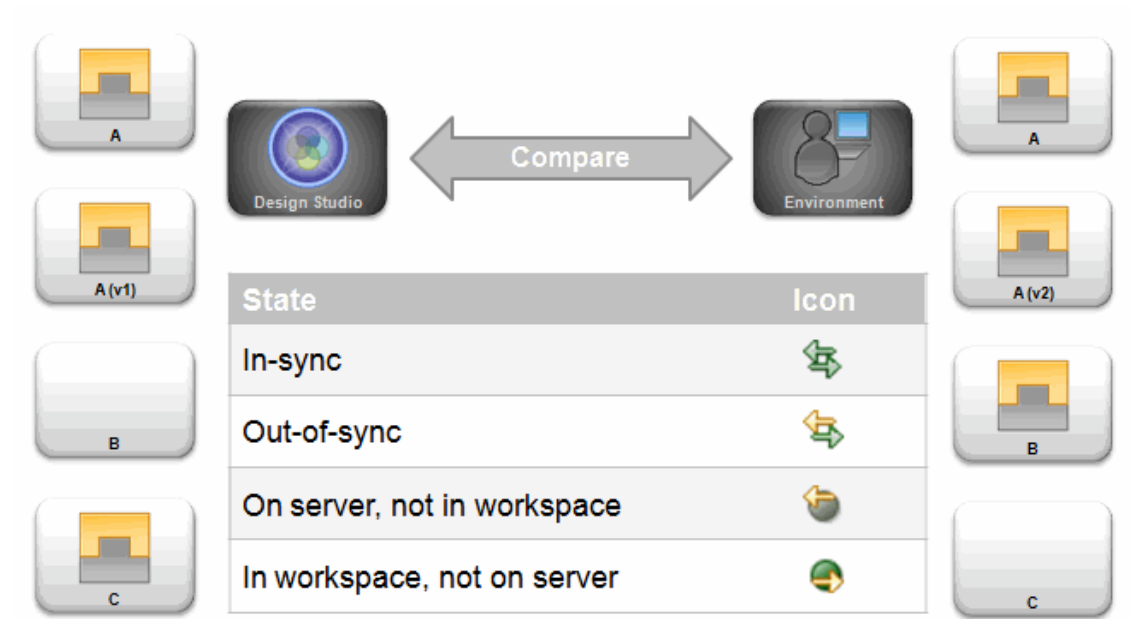
About the Studio Environment Editor

The Studio Environment editor enables you to define the run-time environment connection information, define the Secure Sockets Layer (SSL) keystore file location, review and edit the cartridge and model variables defined for cartridge projects, and to define application-specific connection information.

Figure 7-4   Studio Environment Editor
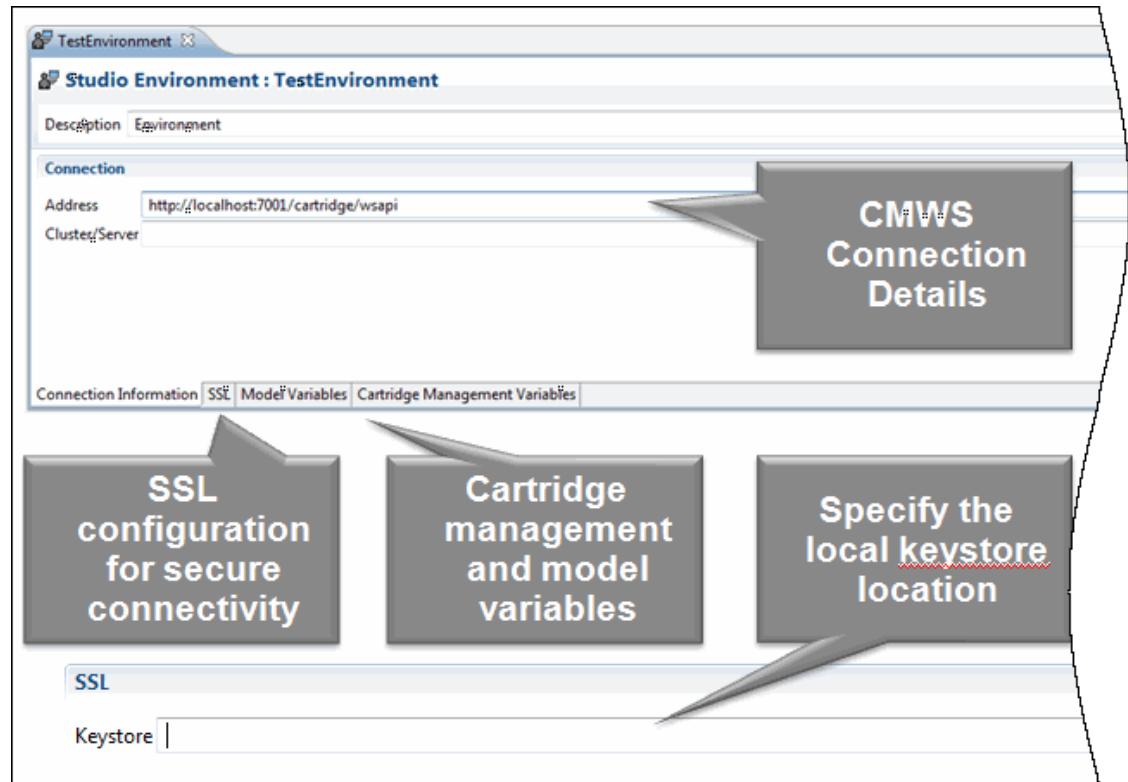
About Model Variables

When you create cartridge projects, some of the information you provide may depend on a specific environment. If you have environment-specific values for variables that you will need at run time, you can create tokens for the variables and later define values for each environment in which you will use the variable. These tokens, also known as model variables, are placeholders for environment-specific values that can be defined at the time of deployment.

For example, consider that you must define the credentials used for running automated tasks in two different environments (your testing environment and your production environment) and that the value required by the testing environment is different than that required by the production environment. Rather than editing the variable value in the source code each time you deploy to one of these environments, you can create a model variable, then define environment-specific values for that variable.

Model variables enable you to realize run-time-specific values at deployment time. During model design, you use a placeholder variable to represent the specific environment value.

Select the **Sensitive** option on the **Model Variables** tab to secure model variable values from unwanted disclosure. Variables marked as sensitive are protected using encryption.

About Cartridge Management Variables

Cartridge management variables specify deployment directives that affect the behavior of a cartridge project deployment. For example, variables can indicate whether a component should restart after deployment or whether run-time data should be purged as part of deployment.

Design Studio variables can be defined at a project level or as overriding values specific to an environment.

Select the **Sensitive** option on the **Cartridge Management Variables** tab to secure cartridge management variable values from unwanted disclosure. This option should be used in conjunction with SSL communication to fully secure variable values. Variables marked as sensitive are protected using encryption.

Figure 7-5   Cartridge Management Variables

Preparing Solutions for Production Environments

Deployment to production environments should be done through a controlled and scripted process. This process should be verified in a staging environment prior to execution against a production environment.

When preparing your solutions for a production environment, you can use the Design Studio testing tools to ensure that solutions are free from errors, and use the Cartridge Management Tool and automated build processes to automate the production process.

Testing Design Studio Solutions

Oracle recommends setting up an integration test environment to continuously deploy solution components and test them together in integrated scenarios. Integration test environments facilitate early discovery of incompatibilities, breakage, and mismatches in component interaction and collaboration.

An integration test environment is composed of the following:

- A server run-time environment

  The environment includes the database server, application servers, OSS applications, and service provisioning solution components. Oracle recommends that you develop scripts that automate the deployment of solution components that are produced by a successful automated build.

- An automated test driver

  The automated test driver initiates the execution of automated tests and collects the test results for reporting quality metrics.

- Set-up and tear-down scripts

  These scripts load seed data, purge test data, and reset the state of the system to ensure that test execution is reliably repeated.

Testing Activities

Table 7-1 lists some solution testing activities, tools, and recommendations.

Table 7-1   Recommended Test Activities

Activity	Tool	Recommendations
Unit and integration testing for ASAP solution components	Design Studio for ASAP (manual)soapUI (automated)	Configure ASAP to run in development/test and loopback mode when testing network elements.Use the createOrderByValue request to submit an order for execution.Represent each test case by the CSDLs that are captured on the order and submitted to ASAP for execution.Use HermesJMS to listen for events from the JMS Topic to validate whether an order executed successfully.
Unit and integration testing for OSM solution components	Design Studio for OSM (manual)soapUI (automated)	Configure store-and-forward queues in OSM to forward to emulators (mock implementations that reply with prerecorded results) for UIM and ASAP. Mocking is optional. Oracle recommends that you call UIM and OSM in an integration test environment, if mocking is unavailable.Represent each test case by the product actions captured on the AIAProvisioningOrderEBM and submitted to OSM for execution.
Unit and integration testing for UIM solution components	soapUI	Create a test case with test steps that call the web service operations (captureInteraction, processInteraction, updateInteraction) in sequence. The captureInteraction request carries the inputs for autodesign into the business interaction. The processInteraction response returns the result of autodesign. Use XPath expressions to validate the autodesign logic.

Table 7-1   (Cont.) Recommended Test Activities

Activity	Tool	Recommendations
Manual integration and system testing for end-to-end scenarios	Design Studio for OSM	In a fully integrated test environment, submit customer orders as AIA ProvisioningOrder EBMs that have coverage of the end-to-end scenarios.
Automated integration and system testing for end-to-end scenarios	soapUI	In a fully integrated test environment, submit customer orders as AIA ProvisioningOrder EBMs that have coverage of the end-to-end scenarios. Oracle recommends that you use automated testing.

Automating Builds

You can configure automated builds to script build processes of cartridge projects. For example, multiple team members can check projects into a source control system, which itself is connected to a secure server maintained by a system administrator. The administrator can schedule automated builds so that official builds can be made available to organizations, such as testing or operations. When using automated builds, no user interface interaction is necessary to build your cartridge projects.

To automate Design Studio builds, create a process that builds a cartridge project, and schedule that process to run using a build automation system.

Running frequent automated builds to verify and test code integrations and to check run-time product archives helps to detect integration errors.

See *Design Studio System Administrator's Guide* for more information about automating builds.

About the Cartridge Management Tool

The Oracle Communications Cartridge Management Tool (CMT) is available on the Oracle software delivery website. The CMT is a command line tool that enables you to deploy cartridges to run-time environments. Oracle recommends that you use the CMT to deploy to production run-time environments.

See *Design Studio Developer's Guide* for more information about the CMT.

8

Extending Reference Implementations

Oracle Communications reference implementations provide starting points for you and facilitate system integration and solution development. Reference implementations do not implement complete sets of features and actions for a domain, and they are not comprehensive solutions.

When developing and extending OSS solutions, begin with the following:

1. Gain an understanding of the service domains.

2. Identify the types of customer-facing services.

3. Identify the actions that a customer can request for services on an order.

4. Identify the input parameters of each action by determining information elements.

5. Describe the steps in the business process for how each action is realized in the network.

   For example, describe the work that is performed and network elements that are configured.

6. Identify the types of resources and resource facing services that drive the business process and realize the actions in the network.

7. Describe the administrative policies and behaviors that determine how each resource is managed.

   For example, administrative policies might include capacity management, and behaviors might include the lifecycle, searching, and selecting an instance of a resource for use by a service.

8. Describe how the resources can be organized in the inventory with respect to readiness for provisioning.

9. Identify the interfaces and protocols for integration for steps in the business process that involve application integration not supported by the reference implementation.

A

Solution Development Methodology

This appendix describes solution development methodologies.

Working with Project Phases and Tasks

This section describes project phases and associated tasks that you can use during operations support systems (OSS) solution development. These example phases and tasks focus on service fulfillment scenarios.

This list is not intended to be prescriptive; it is provided as one part of a best practice approach to solution methodology.

1. Inception and requirements analysis

    See "Inception and Requirements Analysis Phase" for more information.

2. Functional design

    See "Functional Design Phase" for more information.

3. Construction

    See "Construction Phase" for more information.

4. System test

    See "System Test Phase" for more information.

5. Deployment and maintenance

    See "Deployment and Maintenance Phase" for more information.

Inception and Requirements Analysis Phase

During the inception phase, you define the goals of the project.

During requirements analysis phase, you:

- Identify the types of service to be fulfilled and the actions that can be ordered for those services.

- Identify the types of resources (for example, devices) in the network (and for suppliers and partners) on which services will be provisioned.

- Describe the business processes for realizing those service actions in the network.

- Describe the business policies (life cycle, capacity, allocation) for managing services and the resources that support those services.

- Identify any application integrations (beyond what is supported by the service provisioning reference implementation).

- Define the scope of the project with respect to the requirements identified.

- Develop a work breakdown structure and task dependencies for the chosen scope.

- Estimate the effort required to perform the work.

- Develop a schedule based on the work breakdown, effort, and development resource assignments.

Functional Design Phase

During the functional design phase, you:

- Model services and actions.

- Describe the behavior of realizing service actions in the network.

- Model service configurations and resources to support the realization of service actions in the network.

- Describe auto-design behavior for service actions to design the necessary service configurations and assign the necessary resources.

- Describe interfaces and protocols for application integration (beyond what is supported by the service provisioning reference implementation).

> **Note:**
>
> Proving at-risk aspects of the solution architecture can require that you perform some tasks in the functional design phase and construction phase concurrently, enabling the team to clarify requirements and evaluate design alternatives.

Construction Phase

During the construction phase, you:

- Test the technical design, code, and integration. Additionally, perform unit tests and test all components that implement the solution.

- Develop documentation such as user guides, administration guides, and online help.

- Develop data migration plan and scripts.

System Test Phase

During the system test phase, you:

- Develop automated tests and manual test procedures with coverage of the functional requirements (end-to-end scenarios).

- Develop automated tests for non-functional requirements (performance, scalability, availability, and maintainability).

- Set up the test environment with hardware, platform components, and applications.

- Deploy solution components to test environment.

- Run tests, report bugs, and fix bugs.

- Test data migration.

> **Note:**
>
> Complete test development in conjunction with the construction phase. Test development includes the development of tools or scripts for generating data, including installed base of subscribers, services, and resources, for example.

Deployment and Maintenance Phase

During deployment and maintenance phase, you:

- Evaluate hardware sizing and procure hardware and software platform components.
- Set up the production environment with hardware, platform components, and applications.
- Deploy solution components to the production environment.
- Migrate data from the legacy environment.
- Train users and administrators.

Working with Document Artifacts

Table A-1 lists an example set of document artifacts that your team can create and share among stakeholders during solution development. This list is not intended to be prescriptive; it is provided as one part of a best practice approach to solution methodology.

Table A-1   Example Document Artifacts

Document	Use
Requirements	- Defines the functional or business requirements the OSS system should meet. - Documents the requirements analysis phase conclusions.
Functional Specifications	- Documents the services and actions to be supported. - Documents the use case for each service action and the flow through each application component; for example, through Oracle Communications Order and Service Management (OSM), Oracle Communications Unified Inventory Management (UIM), Oracle Communications Network Integrity, and Oracle Communications ASAP. - Documents the function that each application component should perform internally for each service action. - Documents any application extensions and business logic to be implemented.
Integration Specifications	- Documents the integration architecture. - Documents each interface (for example, UIM to OSM and OSM to ASAP), the integration technology used, and the interface protocols.

Table A-1   (Cont.) Example Document Artifacts

Document	Use
Technical Design	• Documents the list of cartridges to be developed or extended, and all the cartridge dependencies. • Identifies all major Design Studio entities, including UIM rulesets, OSM XQueries, ASAP CSDLs and ASDLs, and so forth. • Identifies all major Java classes and methods to be implemented to extend the solution. **Note**: You can create one technical design document for each application component.
Test Strategy	• Describes the testing methods to be used during solution development, such as unit testing, integration testing, and performance testing. • Describes the methods for testing environments, tools, and hardware. • Documents test metrics and measures to be used.
Test Cases	• Describes the test cases for each test type (for example, for unit testing and integration testing).
Technical Architecture	• Documents the technical architecture of the system implemented. • Documents the implemented integration architecture, including JMS resources. • Documents the environment specifications, such as hardware, operating system, and database specifications. • Lists all application components that require backup, such as WebLogic domains, database schemas, and application home directories.
User Guides, Administration Guides, and Online Help	• Describe the documentation to support the solution.

Glossary

action

A specific operation to be performed by a system during solution processing. An action is an interface to be implemented by some system.

action family

A group of actions, designated by action codes, that can be performed against an entity. There are two types of action families, service action families and technical action families.

Activate Technical Order (ATO)

A provider function that generates activation commands for and manages dialogs with specific network devices, based on activation-relevant lines in a technical order.

application role

A type of system in IT architecture. In a multi-layered fulfillment solution, application roles correspond to a fulfillment system type. Examples of application roles include service order management, service and resource management, and activation.

artifact

A general term for the things you can define in Design Studio, such as entities and data elements.

ASAP

Oracle Communications ASAP equips telecommunications service providers with a single platform for automated service activation. ASAP receives service requests from any source and transmits the required service activation information to any destination network device.

ASAP's core architecture isolates business semantics (rules and behavior) from technology semantics (interface implementations and protocols). This architecture allows ASAP to handle multiple, heterogeneous network technologies and supports various interfaces.

base type element

A data element from which other data elements obtain attributes.

To increase modeling efficiency when modeling simple and structured data elements in Design Studio, you can create new data elements that derive from existing base types. Rather than

referencing one of the primitive types (String, Boolean, Integer, and so forth), you reference another data element as their data type. In Design Studio, this is called deriving from a base type element, where the new element automatically obtains the information in the base element.

See data element for more information.

BIRT

(Business Intelligence and Reporting Tools) An Eclipse-based, open-source reporting system for web applications, especially those based on Java and Java EE. BIRT has two main components: a report designer based on Eclipse, and a run-time component that you can add to your application server.

build

A process that updates existing resources and creates new resources. You run builds against projects to create or modify workspace resources. The type of project determines the type of build. For example, when you run a build for a Java project, the build converts each Java source file (**.java** files) into one or more executable class files (**.class** files).

Calculate Service Order (CSO)

A provider function that takes as input order items on a customer order and generates as output order items in a service order. To convert customer order lines into service order lines, CSO uses relationships and mappings defined between products and customer facing services.

Calculate Technical Order (CTO)

A provider function that calculates the difference between a current and requested state of a configuration, identifies the technical actions that are required to achieve the requested changes, and creates a technical order (based on the required technical actions and the dependencies between technical actions) to effect the change in the network.

cartridge

A collection of entities and data defined in Design Studio and packaged in an archive file for deployment to a run-time server. In Design Studio, you build cartridges in cartridge projects. You can create your own custom cartridges to extend Oracle Communications applications. Additionally, you can obtain from Oracle customized cartridges that support integration with other common applications, and cartridge packs that bundle cartridges containing data for particular technology domains.

cartridge designer

A person tasked with design of a deployable component spanning a single product domain. This person is considered an expert for a product in Oracle Communications and focuses on

design in a single product domain. Some cartridge designers may be competent in this role for more than one product domain.

Cartridge Management Web Service

A web service that enables life cycle management of cartridge project (for example, deploy, undeploy, redeploy, and so forth).

cartridge project

A Design Studio project that contains a collection of application-specific entities and data. The collection of entities is packaged into an archive file, which you can deploy to a run-time environment.

Cartridge projects are the only Design Studio projects that are deployable to run-time environments.

CFS

See customer facing service.

clean build

A build that resolves any dependencies or similar errors from all previous build results. Clean builds update all resources within the scope of the build.

CMWS

See Cartridge Management Web Service.

composite design pattern

A design pattern that leverages the logic of existing design patterns and combines that logic with its own configuration. The ability to share logic among design patterns enables you to define common logic in a single design pattern and leverage that logic, as required. When users run a composite design pattern, the design pattern presents all of the fields, pages, and custom logic defined in all of the leveraged design patterns. See **design pattern**.

CSO

See Calculate Service Order (CSO).

CTO

See Calculate Technical Order (CTO).

customer facing service

A technology-agnostic, vendor-agnostic object representing a service.

customer order

A type of order processed by OSM, where the subjects of the order line actions are products.

central order management (COM)

An application role that accepts customer orders from CRM systems and orchestrates the orders among multiple BRM (billing and revenue management), SOM (service order management), WFM (workforce management), and SCM (supply chain management) system instances.

Data Dictionary

A logical collection of data elements and data types in a workspace, enabling you to leverage common definitions across an entire Oracle Communications solution. For example, the Data Dictionary enables you to create order templates, atomic actions, and service specifications, and share the data defined for those entities across your OSM, ASAP, and Inventory applications.

Entities in a workspace contribute data types to the Data Dictionary, and data schemas in a workspace (which are accessible across all projects) contribute data elements to the Data Dictionary. The Data Dictionary enables you to integrate and correlate data models for multiple applications, reduce the size and complexity of a solution model, simplify the application integration by eliminating data translation among applications, and validate data model integrity. See also data schema.

data element

A structured or simple type data definition. When modeling data for a project, you create data elements that you can reuse throughout your model. There are two types of data elements: simple data elements and structured data elements.

See simple data element and structured data element for more information.

data model

The data configuration required by your solution design. A data model includes simple and structured data elements that are defined in data schemas. Design Studio refers to the logical collection of all data schemas and data types in the workspace as the Data Dictionary. See simple data element, structured data element, and Data Dictionary for more information.

data modeler

A person responsible for designing the data types and structures necessary to support a cartridge or solution.

data schema

An XML schema that provides a formal description of a data model, expressed in terms of constraints and data types governing the content of elements and attributes.

All data elements are created and saved in data schemas, which can be accessible across all projects in a workspace. Design Studio automatically creates a project-specific data schema when you create a cartridge project. You can use this default schema to contain the data you require to model the project, you can create multiple schemas in the same project, or you can create schemas in common projects. You can model your cartridge project using data from any combination of these data schemas.

Design and Assign Service Order (DASO)

A provider function that assembles a future-state design configuration for a customer facing service. DASO takes as input the existing configuration. This provider function interprets the constraints of the request, and generates as output a service design configuration containing all data required for delivery.

design pattern

A template containing a self-describing set of entities that can be applied to a Design Studio workspace. Solution designers use design patterns to deliver to end-users sets of pre-configured entities (and their relationships) that serve some domain-specific function. Design patterns enable you to create complex modeling patterns using a wizard. This approach reduces implementation time and effort.

Design Studio

An integrated design environment for the development of solutions based on the Oracle Communications OSS Applications. Design Studio enables solution designers to configure application-specific and multi-application solutions by leveraging application-specific concepts. Design Studio is built on an open architecture based on the Eclipse framework, and it uses a wide variety of innovative technologies.

editor

An editor is a type of view that enables you to edit data, define parameters, and configure settings. Editors contain menus and toolbars specific to that editor and can remain open across different perspectives. You can open entities in editors at any time to modify existing projects and elements.

entity

A functional unit created and edited in Design Studio; for example, tasks, processes, physical and logical resources, and projects. Entities are collected in projects and deployed to run-time environments to support your business processes.

ORACLE®

enumerations

Values defined for data elements that are available for selection in a run-time environment. For example, you can define a set of values for data elements that appear as lists in run-time environments.

environment project

A project that enables you to manage the attributes associated with your run-time environments, including connection attributes, projects ready to be deployed, projects previously deployed, and associated project attributes such as the version and build numbers.

Exchange Format

An XML document based on the data model defined for Design Studio projects. The XML document is generated by a project build. The Exchange Format represents the output of Design Studio configuration in a published XML format, facilitates the exchange of solution modeling information between Design Studio and other systems or applications, and enables you to extend Design Studio functionality.

extended entity

An entity with defined attributes that you leverage when creating new, similar entities. When you extend one entity from another, the target entity inherits all of the data elements defined for the extended entity. In Design Studio, you can extend orders and tasks.

Inherited data elements are read-only. If you extend an entity that includes structured data elements, you can add any number of additional simple and structured child elements.

feature

A package of plug-ins that create a single, installable, and updatable unit. Features are delivered as JAR files, and each plug-in included in a feature is included as a separate JAR file.

Design Studio is a collection of features and plug-ins that you install with a single executable archive file.

fulfillment

Operations that fulfill a customer's order, such as providing, modifying, resuming, or canceling a customer's requested products and services.

guided assistance

A range of context-sensitive learning aides mapped to specific editors and views in the user interface. For example, when working in editors, you can open the Guided Assistance dialog box for Help topics, cheat sheets, and recorded presentations that are applicable to that editor.

incremental build

A build performed automatically in Design Studio when you save resources. You can disable incremental building and manually run builds if, for example, you want to finish implementing cartridge changes before building the project.

manifest

A file that can contain information about the files packaged in a JAR file. By editing information that the manifest contains, you enable the JAR file to serve a variety of purposes.

All Oracle Communications features have manifest file.

metadata

The data definitions you model for entities, specifications, actions, and all other Design Studio artifacts.

model project

A collection of data elements that can be referenced by other projects in a workspace. Model projects include business entities and schema entities that are not specific to any one Oracle Communications application and that enable you to leverage common definitions and share that data across a solution.

model variable

A variable that you create as a placeholder for environment-specific values that you will need at run time.

When you create cartridge projects, some of the information you provide may depend on a specific environment. If you have environment-specific values for variables that you will need at run time, you can create tokens for the variables and later define values for each environment in which you will use the variable. See token for more information.

namespace

A method for uniquely naming elements and attributes in an XML document. Design Studio supports entity and cartridge namespaces. You pair the entity or cartridge name with a namespace name to create a fully qualified namespace. For example, you can pair entity names with a namespace name to enable different groups of Design Studio users to create different entities without concern for naming conflicts. Services can be implemented independently by different teams and then deployed into a single run-time environment.

Network Integrity

Oracle Communications Network Integrity enables you to keep two data sources (such as an inventory system and a live network) synchronized. This improves data accuracy, which increases your service provisioning success rate. It enables better business planning, based on having an accurate view of your inventory, and supports scheduled or ad-hoc audits to

ensure alignment of inventory with your network. Network Integrity can also be used as a convenient way to load initial network data into your inventory system.

operator

A person responsible for managing a product run-time system, performing functions such as installing cartridges to production systems.

optimize deploy

Optimize deploy is a method of deployment that, when enabled, attempts to deploy only the changes you have made in your Design Studio cartridge project. For example, you can use optimize deploy when testing or debugging changes to your cartridge data.

Oracle Communications Design Studio for ASAP

A feature included in the Design Studio preconfigured installation that you use to define service actions, network actions, and scripts for service activation.

See feature for more information.

Oracle Communications Design Studio for Inventory

A feature included in the Design Studio preconfigured installation that you use to define service and resource definitions, rules, and domain-specific metadata.

See feature for more information.

Oracle Communications Design Studio for Network Integrity

A feature included in the Design Studio preconfigured installation that you use to configure network discovery, assimilation, and reconciliation behavior.

See feature for more information.

Oracle Communications Design Studio for Order and Service Management

A feature included in the Design Studio preconfigured installation that you use to define solutions for OSS service order management and for BSS central order management, respectively.

See feature for more information.

Oracle Communications Design Studio for Order and Service Management Orchestration

A feature included in the Design Studio preconfigured installation that you use to define solutions for BSS central order management.

See feature for more information.

Oracle WebLogic Server

Oracle's application server for building and deploying enterprise Java EE applications. The WebLogic server hosts the Design Studio application servers.

orchestration

The process used to manage the fulfillment of a complex order. Order fulfillment often requires interaction with many fulfillment systems. Various dependencies may require that these interactions be run in a specific order to ensure that order items are sent to the proper systems, and that the required steps, in the proper sequence, are run.

Orchestrate Customer Order (OCO)

A provider function that decomposes and distributes lines of customer orders. OCO takes simple offers, bundled offers, and products as input and creates child orders for use in other fulfillment functions and systems.

Orchestrate Service Order (OSO)

A provider function that decomposes and distributes lines of a service order. OSO creates requests to SRM systems to build service designs and delivery plans, then creates a child order to execute the delivery plan.

Orchestrate Technical Order

A provider function that coordinates the delivery of changes to a network based on a set of technical actions defined in a technical order. Orchestrate Technical Order decomposes a technical order into order components that are used to make changes or perform operations in activation, workforce management, supply chain management, and automated test systems. For example, this provider function determines which resources to ship to customers, whether technicians must be dispatched to complete physical work in the network, and what commands to execute on devices in the network.

Order and Service Management (OSM)

Oracle Communications Order and Service Management (OSM) coordinates the order fulfillment functions required to complete a customer order created in a customer relationship management (CRM) system, or other order-source system. As an order management system, OSM receives and recognizes customer orders and coordinates the actions to fulfill the order across provisioning, shipping, inventory, billing, and other fulfillment systems. OSM occupies a central place in order management solutions.

order subject type

A classification of orders based on the type of entity acted upon. Order subjects can be of customer, service, or technical types. Order lines are requests against items of one order subject type. For example, a product offer line item is a request against a customer order; a

customer facing service line item is a request against a service order; and a resource facing service or resource line item is a request against a technical order.

panel

A portion of a user interface that you can collapse to hide or expand to display.

persisted data element

A data element defined on a conceptual model entity. Only persisted data elements are added to Inventory specifications when you realize the conceptual model entity. Persisted data elements are required throughout the lifecycle of a service.

perspective

A defined set and layout of views and editors in the workbench window.

Perspectives determine how information appears in the workbench, in menus, and in toolbars. Each perspective contains a default set of views and editors, which you can customize. The Design Studio perspectives work together with other perspectives that are used for implementation, debugging, builds, and version control.

plug-in

Modular, extendable, and sharable units of code that enable integration of tools within Eclipse. Each plug-in specifies its own dependencies and specifies the set of Java packages it provides. Additionally, plug-ins integrate with other plug-ins.

Plug-ins can be exported as directories or JAR files, shared among different applications, and grouped into features.

POMS

Persistent Object Modeling Service. POMS provides the modeling language, tooling, and framework for modeling entities, the relationship between entities and capabilities that entities have. POMS then generates the plain old Java objects (POJOs) that represent the entities and the relationships between the entities and persists them in a relational database schema.

POMS generates a Java interface as well as a Java implementation with the annotations required by JPA, specifically the EclipseLink JPA implementation, which is required for managing the persistence of the objects. The service also provides framework that provides methods for performing CRUD operations on the entities within a transaction.

The metadata that represent the modeling language is called Entity Relationship Model Language (ERML).

product

A conceptual model entity that represents something that your business sells. Because Design Studio is primarily used for service fulfillment rather than sales, products are often identifiers associated with information from other systems.

Project

An entity that contains artifacts (entities, data, rules, code, and so forth) that you use to model and deploy Design Studio cartridges. Your solution uses various types of projects. For example, you use projects for version management, for single sourcing data, for resource organization, and to build cartridges that can be deployed to a server.

You can create various types of projects and you can extend cartridges that you purchase with your own projects. Oracle Communications supports a library of extensible cartridges that are fully compatible with Design Studio and provide a basis from which to assemble solutions.

project dependency

A state in which entities in one project reference entities in another project, creating a dependent relationship between the projects. For example, an application project might reference data elements defined in a common model project.

provider function

A standard set of unique capabilities, defined by the input the provider function accepts (an order or standard request), the output it generates (an order or standard request), the data that guides the provider function behavior, and a description the provider function purpose. Examples of provider functions include Orchestrate Service Order, Calculate Technical Order, and Orchestrate Technical Order.

provisioning

A set of processes that provide the data necessary for enabling a service, with the actual enabling done by activation.

Rapid Service Design

The configuration of the OSS system to reflect evolving business requirements. A service provider's technical community uses application tools and a standard methodology to complete this design, which includes:

- A technical catalog that contains an information model (with PSR entities) and a functional model that contains fulfillment patterns for service order management, technical order management, and design and assign patterns.

- A technical Inventory that includes assignable resources and network targets.

- A service design methodology that uses an information model as an anchor.

- A dynamic provider function pattern that contains anchor entities, request types, fulfillment patterns, and transformation sequences.

- An order to activation provider function blueprint that maps provider functions to service order management, technical order management, service and resource management, and activation applications. The blueprint also contains order contracts and provider functions.

refactor

The process of changing data elements without modifying the external functional behavior of a solution.

Refactoring in Design Studio enables you to propagate data model changes across the entire solution without sacrificing model integrity. You can rename, change the location of, copy, and move data elements. Additionally, refactoring enables you to copy data elements to create similar data entities, and to create modular and reusable data structures.

Report Development Kit (RDK)

A set of tools delivered with Design Studio that facilitate the creation of custom report designs. The RDK includes a library of key BIRT objects required to retrieve data from Design Studio models, reference report designs, a published Design Studio exchange format, and sample XML files that you can use to test report design features.

See *Design Studio Developer's Guide* for more information about developing your own reports.

resource

A specific object in the network and in the inventory that can be allocated for use by a service.

resource facing service

A technology-specific, vendor-agnostic object representing a service.

RFS

See resource facing service.

root data element

A data element found at the root of a schema entity. A root has no parent data element.

schema entity

An independent resource containing a set of data elements.

service

An entity that represents the way that a product is realized and delivered to a customer. For example, if you sell DSL Gold as a product, it is delivered as a DSL Gold service, enabled by appropriate resources.

service design methodology

A framework, the terminology, and a set of processes that describe the workflow necessary to manage products and services in B/OSS environments.

service domain

A group of services provided by a service provider that includes all of the technical services and resources to support a set of customer facing services. A service domain is also referred to as a service family, a line of service, or a play. Examples of a service domain include mobile, broadband, and IPTV.

service fulfillment

A business process in which a customer order is accepted and a new service is provisioned to meet it.

service order

A type of order processed by OSM, where the subjects of the order line actions are customer facing service instances.

service order management (SOM)

An application role that accepts service orders and orchestrates them among multiple service relationship management (SRM) system instances.

simple data element

Reusable data types that contain no child dependencies. A simple data element has no structure, and is associated (directly or indirectly) to a primitive type (integer, Boolean, string, and so forth).

specification

A blueprint that defines the composition of an entity, including the attributes and relationships between an entity and other objects. There are different types of specifications for different types of entities, such as telephone numbers, networks, customer facing services, and resources. Specifications are defined in Design Studio and deployed into run-time environments, where entities can be created based on them.

solution

In Design Studio, a solution is a model that meets the requirements of use cases that solve a market problem. You use Oracle Communications applications to design solutions that deliver services to customers for multiple service domains.Design Studio documentation generally focuses on solutions for service fulfillment (or Concept-to-Activate, which aligns closely with Product Lifecycle Management, Operations & Fulfillment processes in the TMF Business Process Framework). Solutions can be single-product but typically refer to a suite of products

that provide a set of end-to-end use-cases for both manual and automated processes in the design and run time.

solution designer

A person responsible for pulling a collection of cartridges together to deliver a multi-product solution; an activity that may involve the design of additional cartridges to perform the desired solution functions. The solution designer focuses on cross-product interactions, rather than on the details of a single product.

solution tester

A person who validates that a cartridge or solution is functioning correctly. The solution tester deploys cartridge archives, produced by Design Studio, to a test environment to certify that the cartridge or solution is functioning as intended.

structured data element

Reusable data types that include embedded data types and are containers of simple data elements and other structured data elements.

technical catalog

A catalog of services, resources, fulfillment patterns, and other artifact definitions and metadata organized to support service fulfillment, assurance, and other operational support system (OSS) processes.

technical order

A type of order processed by OSM, where the subjects of the order line actions are resource facing service instances or resource instances, or a combination of resource facing service instances or resource instances.

technical order management

An application role that accepts technical orders and orchestrates them among multiple activation, WFM (workforce management), and SCM (supply chain management), and PGW (packet data network gateway) system instances.

token

A placeholder for environment-specific values that can be defined at the time of deployment.

Unified Inventory Management (UIM)

Oracle Communications Unified Inventory Management gives service providers a single, comprehensive, accurate view of customer services and maps these services to logical and

physical resources, ensuring that trusted, actionable, real-time information is available to any business process for both current and next-generation services and technologies.

view

A presentation of information in the workbench. Views enable you to customize the manner in which information is presented, and provide access to a specific set of functions, available through the view's toolbars and context menus.

For example, the Problem view displays errors that exist in the model entities, so you use the Problem view to locate and resolve entity errors. You use the Data Element view to model and review data in your workspace. The Data Element view and Problem view each provide access to a different set of Design Studio functions.

A view can appear by itself, or it can be stacked with other views. You can change the layout of a perspective by opening and closing views and by docking them in different positions in the workbench window.

workbench

A set of tools you can use to navigate within and manipulate the workspace, access functionality, and edit content and properties of resources.

working set

A configuration that describes a subset of files, classes, folders, and projects. Working sets help you categorize resources across projects into a contextually relevant representation. Working sets facilitate efficient modeling in large, multi-project solutions by enabling you to limit visibility to relevant projects only.

workspace

A representation of your data. Workspaces are directories on your local machine that contain resources, including projects at the top of a tree structure and folders and files underneath. A workspace root directory is created internally when you create a Design Studio workspace. You can create more than one workspace, but you can have only one workspace open at a time.