HP Best Practices

Software Version: 3.00

Service Modeling

Document Release Date: January 2015

Legal Notices

Warranty

The only warranties for HP products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. HP shall not be liable for technical or editorial errors or omissions contained herein.

The information contained herein is subject to change without notice.

Restricted Rights Legend

Confidential computer software. Valid license from HP required for possession, use or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Copyright Notice

Trademark Notices

Adobe® is a trademark of Adobe Systems Incorporated.

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

UNIX® is a registered trademark of The Open Group.

Documentation Updates

The title page of this document contains the following identifying information:

- Software Version number, which indicates the software version.
- Document Release Date, which changes each time the document is updated.
- Software Release Date, which indicates the release date of this version of the software.

To check for recent updates or to verify that you are using the most recent edition of a document, go to: **http://h20230.www2.hp.com/selfsolve/manuals**

This site requires that you register for an HP Passport and sign in. To register for an HP Passport ID, go to: **http://h20229.www2.hp.com/passport-registration.html**

Or click the **New users - please register** link on the HP Passport login page.

You will also receive updated or new editions if you subscribe to the appropriate product support service. Contact your HP sales representative for details.

Support

Visit the HP Software Support Online web site at: **http://www.hp.com/go/hpsoftwaresupport**

This web site provides contact information and details about the products, services, and support that HP Software offers.

HP Software online support provides customer self-solve capabilities. It provides a fast and efficient way to access interactive technical support tools needed to manage your business. As a valued support customer, you can benefit by using the support web site to:

- Search for knowledge documents of interest
- Submit and track support cases and enhancement requests
- Download software patches
- Manage support contracts
- Look up HP support contacts
- Review information about available services
- Enter into discussions with other software customers
- Research and register for software training

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract. To register for an HP Passport ID, go to:

http://h20229.www2.hp.com/passport-registration.html

To find more information about access levels, go to:

http://h20230.www2.hp.com/new_access_levels.jsp

HP Software Solutions Now accesses the HPSW Solution and Integration Portal Web site. This site enables you to explore HP Product Solutions to meet your business needs, includes a full list of Integrations between HP Products, as well as a listing of ITIL Processes. The URL for this Web site is **http://h20230.www2.hp.com/sc/solutions/index.jsp**

Contents

Welcome to This Guide

Welcome to the HP Service Modeling Best Practices Guide. This guide provides the guidelines and recommendations for IT service modeling design, process, and practice. Most modeling practices are product-independent.

This chapter includes:

How This Guide is Organized

This guide is structured in three parts—from general to specific.

Part I "Service Modeling Concepts"

Part I describes why service models are created, explains use cases, and describes how models are consumed.

Part II "Service Model Development Cycle"

Part II explains in detail how to design and implement consumer-oriented service models that are manageable—describing how to transform use cases and conceptual models into logical, then realized models.

Quick Reference Guide: If you are already experienced with service modeling concepts and HP Universal CMDB (UCMDB), Part II contains a Quick Reference Guide, tool kit, and quick steps for creating service models. This section can be used as a cookbook to accelerate model development.

Part III "Service Models in UCMDB"

Part III is based on HP Universal CMDB (UCMDB) version 10, and takes Advantage, Inc., an example banking company, through the service modeling process. This section explains how to transform a logical model, use case, and requirements into a functional service model using an industry-leading tool set.

Prerequisites

Readers may find it helpful to familiarize themselves with:

- Consumer/Owner/Provider (COP) model in *CMS Strategy Guide* in the CMS Best Practices Library
- *Provider Onboarding Guide* and *Consumer Onboarding Guide* in the CMS Best Practices Library
- Universal Data Model (UDM) and Discovery Best Practices from the *CMS BP-Data Modeling* and *CMS BP-Discovery Planning* guides in the CMS Best Practices Library
- HP Business Service Management (BSM) modeling best practices document *RTSM Best Practices* in the HP Cross-Portfolio Software Best Practices Library
- *HP Universal CMDB Modeling Guide* in the product documentation

Who Should Read This Guide

This guide is intended for anyone who needs to understand, build, or use service models in an IT environment. People in the following roles commonly work with service models:

- Configuration Managers
- Change Managers
- Asset Managers
- Service Desk Managers
- Business owners
- Application owners
- Business Service Architects
- Technical Project Managers
- Technical Consultants
- Content developers

The information in this guide may duplicate information available in other Best Practices documentation, but is provided here for convenience.

Resources

- HP Live Network: *https://hpln.hp.com/home*
- HP Software Cross-Portfolio Best Practices Library: *https://hpln.hp.com/page/all-best-practices*
- CMS Best Practices Library: *https://hpln.hp.com/node/1630/attachment*
- BSM Best Practices: *http://support.openview.hp.com/selfsolve/document/KM00412297/binary/BSM_922_EffectiveModeling_BP.pdf*

Additional Online Resources

Troubleshooting & Knowledge Base accesses the Troubleshooting page on the HP Software Support Web site where you can search the Self-solve knowledge base. Choose **Help > Troubleshooting & Knowledge Base**. The URL for this Web site is http://h20230.www2.hp.com/troubleshooting.jsp.

HP Software Support accesses the HP Software Support Web site. This site enables you to browse the Self-solve knowledge base. You can also post to and search user discussion forums, submit support requests, download patches and updated documentation, and more. Choose **Help > HP Software Support**. The URL for this Web site is www.hp.com/go/hpsoftwaresupport.

Most of the support areas require that you register as an HP Passport user and sign in. Many also require a support contract.

To find more information about access levels, go to:
http://h20230.www2.hp.com/new_access_levels.jsp

To register for an HP Passport user ID, go to:
http://h20229.www2.hp.com/passport-registration.html.

HP Software Web site accesses the HP Software Web site. This site provides you with the most up-to-date information on HP Software products. This includes new software releases, seminars and trade shows, customer support, and more. Choose **Help > HP Software Web site**. The URL for this Web site is www.hp.com/go/software.

HP Software Solutions Now accesses the HPSW Solution and Integration Portal Web site. This site enables you to explore HP Product Solutions to meet your business needs, includes a full list of Integrations between HP Products, as well as a listing of ITIL Processes. The URL for this Web site is http://support.openview.hp.com/sc/solutions/index.jsp.

Glossary

API	Application Programming Interface
BSM	HP Business Service Management
CCRM	HP Change Control and Release Management
CI	Configuration Item
CLIP	HP Closed Loop Incident Process
CMDB	Configuration Management Database
CMS	Configuration Management System
COP	Consumer/Owner/Provider
CSV	Comma Separated Values
DNS	Domain Name Server
EA	Enterprise Architecture
ERP	Enterprise Resource Planning
FSC	Forward Schedule of Change
FTP	File Transfer Protocol
IDE	Integrated Development Environment
IP	Internet Protocol
IT	Information Technology
ITSM	IT Service Management
JDBC	Java Database Connectivity
MTBF	Mean Time Between Failures
MTTR	Mean Time to Recovery
ROI	Return on Investment
SACM	HP Service Asset and Configuration Management
SAN	Storage Area Network
SM	HP Service Manager

SME	Subject Matter Expert
SQL	Structured Query Language
TCP	Transmission Control Protocol
TQL	Topology Query Language
UCMDB	HP Universal CMDB
UD	HP Universal Discovery
UDM	Universal Data Model

Part I: Service Modeling Concepts

Chapter 1: Service Modeling

This chapter includes:

What is a Model?

A model is a representation of objects/entities that collectively combine to serve a functional purpose to the IT environment and either directly or indirectly support the business. Models referred to here are non-physical, existing as data, and managed by systems engaged in configuration and IT service management.

What is a Service?

ITIL V3 defines a service as a means of delivering value to customers by facilitating outcomes customers want to achieve without the ownership of specific costs and risks.

A simple way of saying this is: A service is an abstraction of a way of doing something for someone or something. But there is more to a service than the service itself. The new style of IT seeks to understand the service in action. When the service is used, we also want to understand what is happening with the consumer, the consumer experience, and the value generated as a result of the consumptive act.

By including these aspects of the service in our notion of a model, we can give IT a better way of accounting for cost, and managing the services as services rather than a collection of parts. In this document, the definition of a service includes the operation of the service, as well as its interactions with consumers.

An IT service is a foundational component for business services. Business services may be composed of multiple IT services. IT services themselves may be composed of other, more granular, IT services. Fundamentally, a service can also be thought of as a generic name for any functional component, all the way down to small, discrete services, such as an IP address, a network interface, or a database table.

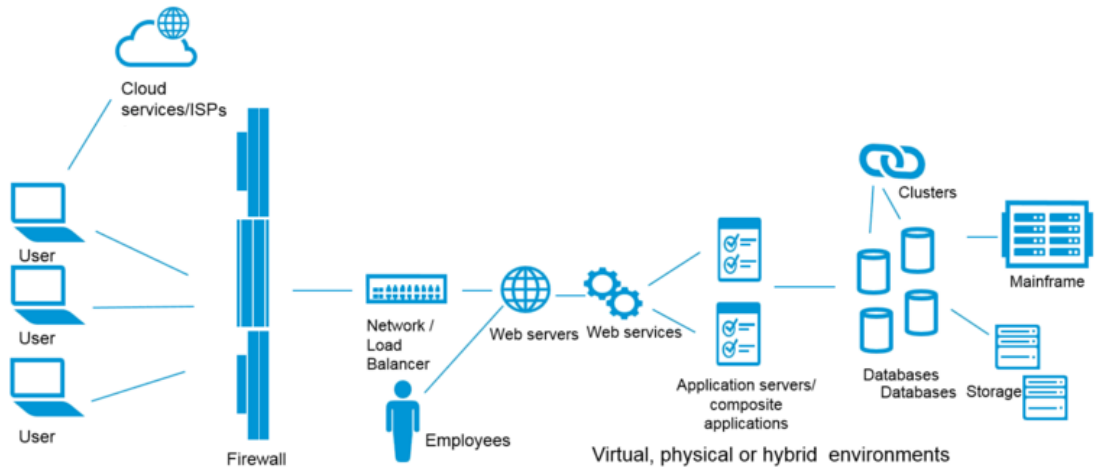
What is a Service Model?

This section includes:

Overview

A service model is composed of configuration items (CIs) and relationships. CIs represent the actual components of the real service. CIs and relationships are contained in the Configuration Management System (CMS). The CMS is a specific collection of data providers and consumers built around a Configuration Management Database (CMDB) such as HP Universal CMDB (UCMDB).

The following is a conceptual diagram of a service model showing typical objects and their relationships:



The service model's purposes are:

- To establish an ideal standard, a **model** service, or to allow rapid construction of an uncertain ideal in order to make an early evaluation of the ideal—for example, to answer the question “Is this really what we wanted?”
- To present the same view of a service to all involved parties—to facilitate communication between them and answer the question “Do we agree on exactly what it is?”
- To simulate reality to improve real return on investment (ROI)—a minimal investment in time to simulate an operation in the model and observe the results—with the possibility of changing the

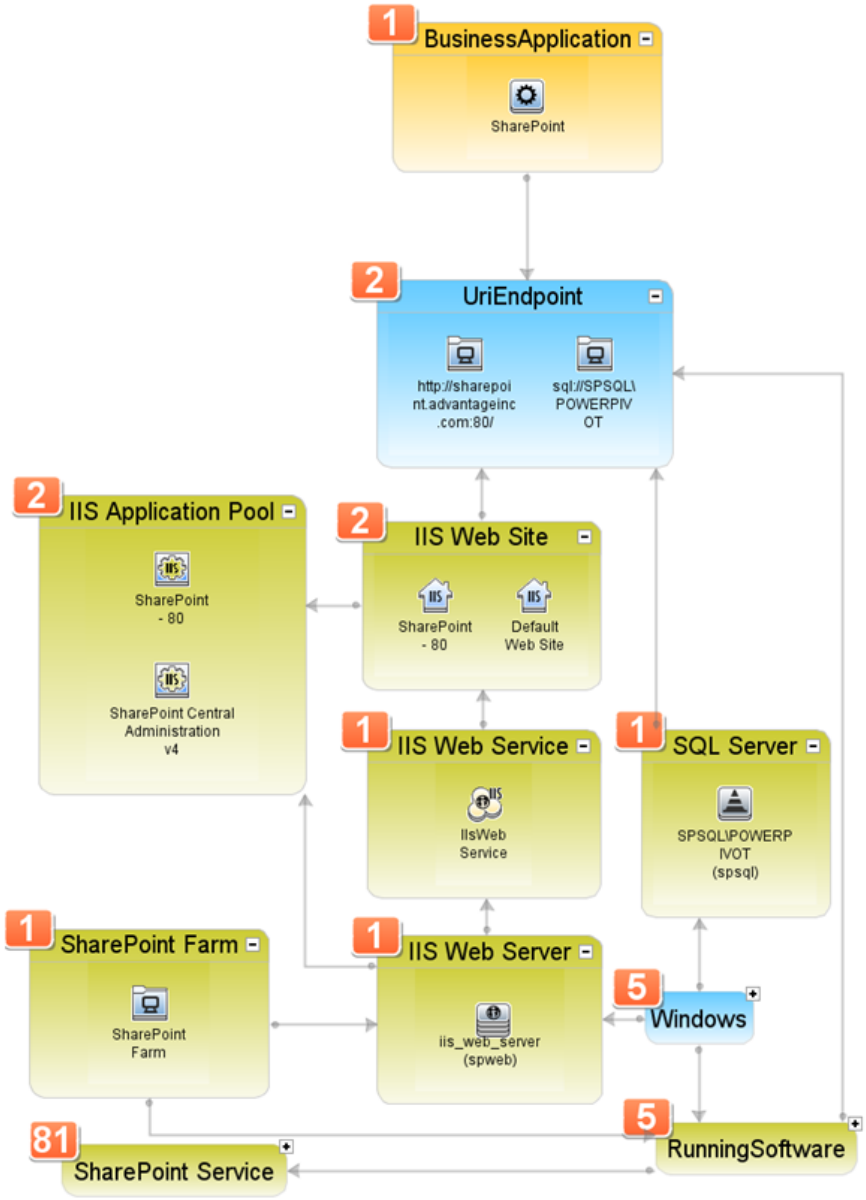
intended actions prior to actually incurring any risk or cost. To answer what-if questions based on real-world situations to discover and prevent potential problems before they happen.

- To store information about the reality represented by the model—a **working** or **applied** model such as is the case with HP Universal CMDB today.
- To answer any working questions about CIs, relationships, and the models constructed from them; such as using the UCMDB Browser as a common tool to consume CI and topology information from them. For example, an IT application administrator needs information about a specific application server in order to plan an upcoming change.
- To help owners and administrators manage the service's components through their life cycles, and to ensure the service provides the expected value to the organization. The service model is consumed by various people, processes, and tools in the management of IT environments.

A service model calls out, makes conscious, and enables visualization of the discrete components and, critically, the relationships between them. To this end, the service model is a tool to decompose a service into its constituent parts in order to manage and depict the service in useful ways. For example, a business service called **Online Banking** may use several IT services, such as Bill Pay, Account Management, Money Transfer, Stock Quotes, and Live Chat. Each of these IT services is, in turn, composed of software programs that run on hardware and communicate with each other, store data, and so on.

Service Model Example

Everything just described works in combination to create value—to provide a service. These objects and relationships can be expressed as a topology. The following diagram is a high-level example of a service model named **Sharepoint**, a common business application that is simple enough to be readily understood, and complex enough to demonstrate useful service modeling techniques. The high-level service model for a working Sharepoint system is depicted below. The diagram is visualized topologically and is grouped by CI type.



Note how the business service **Sharepoint** is composed of business and infrastructure layers consisting of multiple CI types and relationships. This service will be the focus of our running example with HP's example company, Advantage, Inc.

Understanding Service versus Application

Even though the **Sharepoint** CI type is **BusinessApplication**, Sharepoint can still be called a service. Either hierarchy works. Models used for different purposes may have different starting points, such as **BusinessService**, **BusinessApplication**, both—or something else. Part of the best practices is not to prescribe a rigid structure, but to learn how to find what structure works for a particular IT environment, based on size, needs, and limitations. **Service** is generally considered a higher-level entity than **Application**, but in isolation, the definitions of the two terms may blur. **Service** is generally a business-facing entity, seeking to establish the presence and identity of the service, whereas **Application** is a technology-facing entity, seeking to describe the service components.

In this example, the Sharepoint application can be understood in some detail at a glance:

- Sharepoint is a business application and is linked to the rest of the infrastructure by a URI endpoint. Since this is in fact the same way a consumer would access the service, this CI is always guaranteed to exist. This is why a communication endpoint is the best practice to link the upper and lower layers of a service model.
- Sharepoint consists of several application and infrastructure layers. The presentation, application, and database layers may easily be seen. The view of this model is grouped by CI type, allowing the relationships to be more restrained and letting the focus rest on the types of CIs that form this application.

This service model is useful for someone who needs to understand the service at a high level. More details may be needed about the model in order to answer more detailed questions. The service model may be extended with additional CIs and relationships depending on the consumer and use case requirements.

HP Universal Discovery does not discover **BusinessApplication** or **BusinessService** CI types.

Class Model

This section includes:

Overview

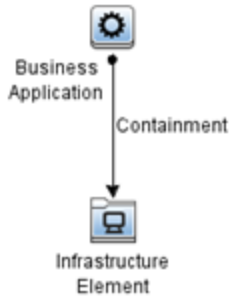
The class model is the meta-language of the service model and is a fundamental part of the CMS. This class model is the data dictionary of all the types of objects, their attributes, and relationships to other objects that are considered valid in the CMS. Each instance of an object in the CMS must be of a valid CI or relationship type. Furthermore, relationships must make sense with regard to the two CIs on either end. A relationship cannot go between any two CIs. The link must be a valid link as defined by the class model. As a best practice, research and understand the class model, including relationships and valid links, before getting far into service modeling.

Populating the CMS

All the object types in service models are based on the class model. By instantiating CIs and relationships, the CMS is populated. Population may occur in many ways, and is beyond the scope of this document, other than to say the need for additional discovery may be revealed only after the modeling process has started. As a best practice, discovery should be a continuous process. If possible, do not rely on a single discovery to create service models since there may be no easy way to add missing data manually. From a service modeling perspective, any population strategy must account for all the objects necessary to fulfill the use case.

Queries

Queries extract subsets of data from the CMS. Queries are the basis for service models. Queries can retrieve only the objects belonging to a single service. In UCMDB, queries are implemented in a visual modeling paradigm called topology query language (TQL). **TQL** and **query** can and are used interchangeably for the remainder of this document. TQLs consist of a set of CI types, relationships, and other configuration data which can dynamically and selectively retrieve the actual CIs and relationships—for example, a TQL to model the upper layers of the Sharepoint service.



TQLs are powerful and can be used in many ways. It is possible to control the CI types, relationships, conditions for matching, and complex expressions to include or exclude exactly what results. For more information on TQLs, refer to the UCMDB product documentation on the HP Software Product Manuals Web site. A UCMDB user interface online search for **TQL** will provide direct links to the relevant document sections.

Configuration Item Types

Configuration Item (CI) types and relationships commonly used in service models include:

- **Node.** represents device types, such as a computer or network device
- **Computer.** a sub-type of **Node** that is not a network device

  Sub-types such as **Windows**, **UNIX**, **Router**, and **Printer** each carry device-specific attributes relevant to that platform or hardware.
- **IpAddress.** an IP address
- **RunningSoftware.** represents the specific application instances running the service—for example, a specific instance of a **Webserver**

  Sub-types of **RunningSoftware** include **ApplicationServer**, **Database**, **WebServer**, and **MessageQueueServer**.
- **CommunicationEndpoint.** TCP connection from one instance of running software to another and linked by a dependency relationship

Relationships

Relationships, or **links**, exist between exactly two CIs. A relationship must always have a **start** and **end** CI. Below are some of the common relationships used in service models:

- **Dependency.** a relationship representing one entity's reliance on another for operation

  Dependency relationships are used by impact analysis to show critical reliance between CIs, but to indicate that the CIs are separate entities—for example, one server may depend on another server, but the two are physically separate.

- **Containment.** a relationship representing a logical entity contained within or as an aspect of another CI—for example, the network card which has been assigned the IP

  Containment is used to indicate more than dependency, but encapsulation—for example, a network interface is contained in a server.

To illustrate the differences, if a server is destroyed in fire, so are its interfaces. Therefore, the interfaces are **contained**. This does not necessarily hold true for two dependent servers. One server may be in a different location, and if it is destroyed, it does not necessarily follow that the other server is also destroyed. Therefore, the relationship type is **dependency**.

These examples and more of the class model will be used throughout this document. For more information, see the UCMDB product documentation and the Data Modeling Guide in the CMS Best Practices Library (https://hpln.hp.com/node/25/otherfiles).

Valid Links

The Class Model contains rules for relationships called **Valid Links**. Valid Links define whether a given relationship type may be used between two CI types.

In other words, you may not use relationship types arbitrarily. They must make sense. For example, it makes sense for a **Computer** CI type to have a relationship with a **disk** CI type. The computer contains the disk. **Contains** is a Valid Link from computer to disk.

An example of a link that would not make sense in a hypothetical model of reality is a computer containing a data center. This does not make sense. The relationship is the other way around. In addition, the data center would not fit in the computer. So **contains** would not be a valid link between those two CI types in that model.

Computer and **Datacenter** are an obvious example of why there is not a valid link between those two CI types. However, relationships in IT models are not always as intuitive. Therefore, it is necessary to have a set of rules that enforce the valid links of the reality of the data center model in UCMDB.

Using the Class Model

Modeling and using the class model can be intimidating at first. Questions such as "What relationship do I use?", "How do I know what is there to start with?", "Nothing seems to work, my calculator stays at zero objects, what do I do?" are common. A few best practices may help:

- Most modeling is done with a small amount of the class model. You will quickly learn which parts are relevant to you. Refer to "Modeling Quick Reference" on page 42 to see commonly used service model fragments. **Node**, **Dependency**, **Containment**, **RunningSoftware**, and **CICollection** are some of the most common components used to create service models.
- Reuse out-of-the-box examples
- Keep the size of models small. Smaller models are usually:
  - easier to understand
  - easier to maintain and debug as they change and grow
  - easier to keep focused as the model changes
- Do not alter the class model itself without help or prior experience.
- Use the **Get Related** CIs tool to navigate around in the CMS without having to build queries or search in the IT Universe. This will quickly show you what is populated (as well as what is missing) in the CMS and what you can use in service models.
- The first few successful models can be used as templates. By reusing the parts of the class model common to your environment and needs, service models can be built much faster.

Simplicity, the first best practice

Good service models allow simple articulation and interpretation of the contents. Despite the initial apparent complexity, it quickly becomes easy to visually understand which CIs and relationships are present. It is important for good service models to tell one story, solve one problem, or answer one question.

Do not overbuild service models. It is easy to overbuild and put things into a model. It is difficult to take things out.

Chapter 2: Modeling Use Cases

This chapter includes:

Overview

Both the configuration management system (CMS) and service modeling address IT problems of cost, quality, transparency, operational efficiency, and risk management. Efficient service modeling is a key to realizing the most return on investment (ROI) when implementing IT Service Management.

Service modeling is defined here as the activities associated with creating a CMS-based representation of a business service for the purpose of consumption within and for IT and business processes.

If created and used properly, service models consumed in a CMS can unlock business value. For instance:

- Make better decisions with greater visibility into your IT environment
- Enable cloud aware, service-driven IT, including virtual and clustered environments
- Drive information sharing and automation

Core service models are best recognized as tools for:

- Topology visualization of services, applications, and infrastructure
- Topology mapping, as a reference for service owners and administrators
- Impact analysis before, during, and after disruptive operational events

More refined and overarching use cases are constructed from these basic capabilities, such as:

- Closed Loop Incident Process (CLIP) solution that connects event management to incident management and breaks the silos between these two processes. For more information, see the *HP CLIP Solution Guides v9.30*.
- Change Control and Release Management (CCRM) solution that compares and analyzes service models. For more information, see the *HP CCRM Solution Guides v9.30*.
- Service Asset and Configuration Management (SACM) solution that implements the control function for the service asset life cycle, and builds on the CLIP and CCRM solutions For more information, refer to the *HP SACM Solution Guides v9.30*.
- Business Service Monitoring (BSM) for service components and endpoints
- Solution Architecture use cases providing continuity throughout the service life cycle
- Auditing use cases, both internal and external
- Security and Risk Mitigation use cases, using the CMS for IT security

CMS value is realized as configuration data and is consumed and used to make decisions. A key to efficient and value-generating consumption is the proper use of service models. Service models deliver the right information to the right consumers at the right time. Service models are driven by and constructed in the context of consumers and their use cases.

Use cases and process support build on the fundamental CMS capabilities. Models are used for many purposes. A service model serves as an abstracted representation of an actual service in order to learn, validate, and forecast useful information about the actual service.

Common Goals: Key Driver of Service Modeling

Despite the potential benefits, implementing service models can be challenging. A populated CMS must first exist, and this requires some degree of organizational maturity and executional ability. It may be argued that the organizations least capable of creating service models may be those most in need of them. To hold any expectation of success, concensus and collaboration across the IT departments/silos is critical. All the department heads and technical influencers in IT should be in agreement that a common view of services is needed and possible.

The key to getting agreement on common organizational objectives is to agree on a common language and business context that describes the structure and dependencies between service assets and services shared by IT silos. This common language, the class model, will not only help maintain the

level of service to which an organization may be obligated, but also improve the existing customer experience and attract further business. A good strategy is to include common goals in the project charter—such as reducing risk, forecasting capacity, improving support productivity, improving regulatory compliance, and so on—that are enabled incidentally as services are modeled for the first time.

Data Quality

Data quality is imperative to successful service modeling. Models created lacking the proper data or with poor-quality data that is impossible to test are unlikely to deliver value. There are multiple approaches to data quality, but an approach must be adapted and used in any case. Traditionally, HP's prescriptive model is the Consumer/Owner/Provider paradigm, which calls for processes to ensure scrutiny of on-boarded data and ensure against provider conflict, data bloat, undirected discovery, and many other problems associated with CMS population. For more information, refer to the *CMS Strategy Guide* in the CMS Best Practices Library.

The keys to data quality are:

- Completeness of relationships between service assets and between various services
- Organization's understanding for the importance of a service to the business
- Processes in place to filter out bad data, preferably by preventing it from getting into service models, but in any case, preventing it from getting to the consumer

In the example from the previous section, the Sharepoint service may rely on several IT services—such as authentication services, email services, storage services—all working in harmony to provide customers with a working service. These services may utilize various service assets—such as standalone applications, computers and network devices—which house critical information and run essential processes.

The services themselves are ultimately the end benefactors of well-defined service models. Having a common view of services could improve many outcomes.

For example:

- Planning a maintenance period for a poorly performing network device that affects many revenue generating business services
- Coordinating changes to common infrastructure among multiple groups
- Creating more accurate charge-back models based on resources utilized by various constituencies
- Identifying services which are at higher risk of change due to poorly designed or implemented architectures, such as single points of failure or major security vulnerabilities
- Reducing the unplanned down time of the service

Non-Discoverable Data in Service Models

First consider the source of CIs and relationships in the CMS:

• Autodiscovery

• Integrations and Federation

• Manual entry

Next consider a service model as layers—an upper and a lower layer. The upper layer is the **business** section, and the lower layer is the **infrastructure** layer. The upper layer CIs are typically **BusinessService** or **BusinessApplication**. An IT organization will usually have a list of application and/or service names somewhere, even if only on a spreadsheet on the CTO's laptop. The data may or may not be accessible via discovery, and may need to be entered manually, using a tool such as the HP Universal CMDB (UCMDB) Browser, HP Service Manager, or an Enterprise Architecture tool.

In the lower layer, some relationships may be difficult to discover, yet may be a key part of a service model. The relationship may be created manually, or advanced discovery techniques may be used to discover relationships in specific ways—for example, a custom script may be developed to read a configuration file that is used by an application to create connections.

Most importantly, the knowledge required to connect the upper and lower layers may often be obscure or tribal, unstructured, or non-electronic (for example, paper) in nature and, as a result, can create miscommunication between silos when dealing with entities which cross silo boundaries. The best practices in this document provide a consistent, reliable way to do this in all cases, regardless of the application role, platform, complexity, or consumer type. Modeling becomes easier by looking at the service from the consumer's perspective.

UCMDB offers various methods to easily find this information and rapidly develop this connection based on discovered data or integrating existing repositories of authorized data.

Modeling Use Cases Hierarchy

Models are fundamentally used for two primary purposes:

- topology visualization
- impact analysis

It is easy to understand how service modeling enables these capabilities. All other use cases are based on these two—adding context, user requirements, and granularity as necessary.

Building more refined use cases can be thought of as being built on top of more general use cases:

CMS Capability	ITSM Processes Supported	Major Use Cases
Topology Visualization	Configuration Management	Dynamic Visual Configuration Reference Architecture
Topology Visualization	Configuration Management	Service Asset and Configuration Management (SACM) Control Function
Topology Visualization	Configuration Management, Security Management, Continual Service Improvement, Service Level Management	Audit and Compliance, Infrastructure Quality
Topology Visualization	Event Management, Operations Monitoring	Business Service Management (BSM)
Impact Analysis	Change Planning	Change Control and Release Management (CCRM)
Impact Analysis	Change Planning	CCRM
Impact Analysis	Risk Assessment	CCRM
Impact Analysis	Problem Isolation and Notification	Closed Loop Incident Process (CLIP)

Topology Visualization

Topology visualization as a reference diagram was one of the earliest uses for service modeling. Think of service topology visualization as a dynamic version of a static picture of the service. For example, Microsoft Visio® has traditionally been used to build pictures of services as an architectural and operational reference. In a CMS, the same service is constructed from discovered and other authoritative components, and updated as those components and the relationships between them change.

- Updated information: What does the service look like right now?
- Visualization reference of the service: Is the service correct?
- Validation of the authenticity and authoritativeness of the components that comprise the service
- Ad hoc information about the components in many daily working IT scenarios (UCMDB Browser)

Configuration Management embodies this old use case in a fundamental sense—providing the right information to the right user at the right time, for everyone, all the time.

Impact Analysis: Analytical Modeling

Visual topography adds value for human consumption because most people think visually. Topology is also useful for tracing paths from one component to another systematically.

For example:

- Consider the potentially complex compound paths between a problem, the server encountering the problem, the application running on that server, and the business service that depends on that application. Easily visualizing these paths is useful for problem isolation, notification, prioritization, and creating an accurate incident.
- Prior to creating a request for change (RFC), an IT administrator may need to know who may potentially be affected if something goes wrong with a change. Part of the change process notifies all the owners of the potentially impacted services. This provides the context for discussion in change control meetings.

Change Management

Change Management asks the following questions:

- How do I understand the risk a specific change or set of changes pose to my most critical IT services?
- How do I coordinate the right discussions to ensure the changes are implemented correctly?
- How do I coordinate changes between groups?

Service modeling impact analysis, as well as topology visualization, are used in Change Management processes. By tying in the Service Desk and the CMS, you enable the Change Control and Release Management (CCRM) and Closed Loop Incident Process (CLIP) use cases (discussed in "Major Service Modeling Use Cases" on page 29).

Configuration Management

Like the CMS, the Configuration Management process underlies many other ITSM and SACM processes:

- How do I ensure that I control and track all of the configurations that make up our IT environments?
- How do I provide a common view of services to IT?
- How do the change and release management processes coordinate complex change activity?

Configuration Management is an underpinning process for service modeling because it establishes the CMS. The CMS is the source and foundation for creating service models.

Configuration Management may also be responsible for service modeling. Therefore, it is also naturally involved in Continuous Service Improvement and Service Level Management processes, being well-positioned in IT to provide service modeling guidance and assistance, driving IT towards a common view of services, and the positive outcomes resulting from the giant maturity step forward that is service modeling.

Auditing and Compliance

Auditing and compliance are broad consumers of CMS data, including service models. Auditors may request topology for audit. Administrators may review models for internal policy and compliance. For example, critical business services should not rely on a single point of failure or expose major security vulnerabilities. An understanding of IT topologies is necessary in order to find non-compliant topology such as single points of failure.

Data Center Transformation

In recent years, data center transformations have become consumers of CMS data. Project managers are asking:

- How do I develop a verifiable, accurate, and comprehensive view of what I have?
- How does it relate in order to effectively execute my data center consolidation?
- What are my move groups? What must move together?
- What will be impacted if I separate two components of a service?
- How can I comply with the massive reporting effort?
- How can I ensure all services and components are accounted for in the transformation plan?
- What is the business impact of an identified threat and how can I lower the risk to the business?

Cloud-building, virtualization, application rationalization, and similar use cases all involve some type of transformation. Service models easily represent move groups. If the move groups are created along application and service boundaries, the move groups become models of the transformed services in the target environment—a potentially large ROI of time, cost, and risk to build a separate CMS after the transformation. There is a collection of UCMDB models and views for data center transformations in the Data Center Transformations Accelerator.

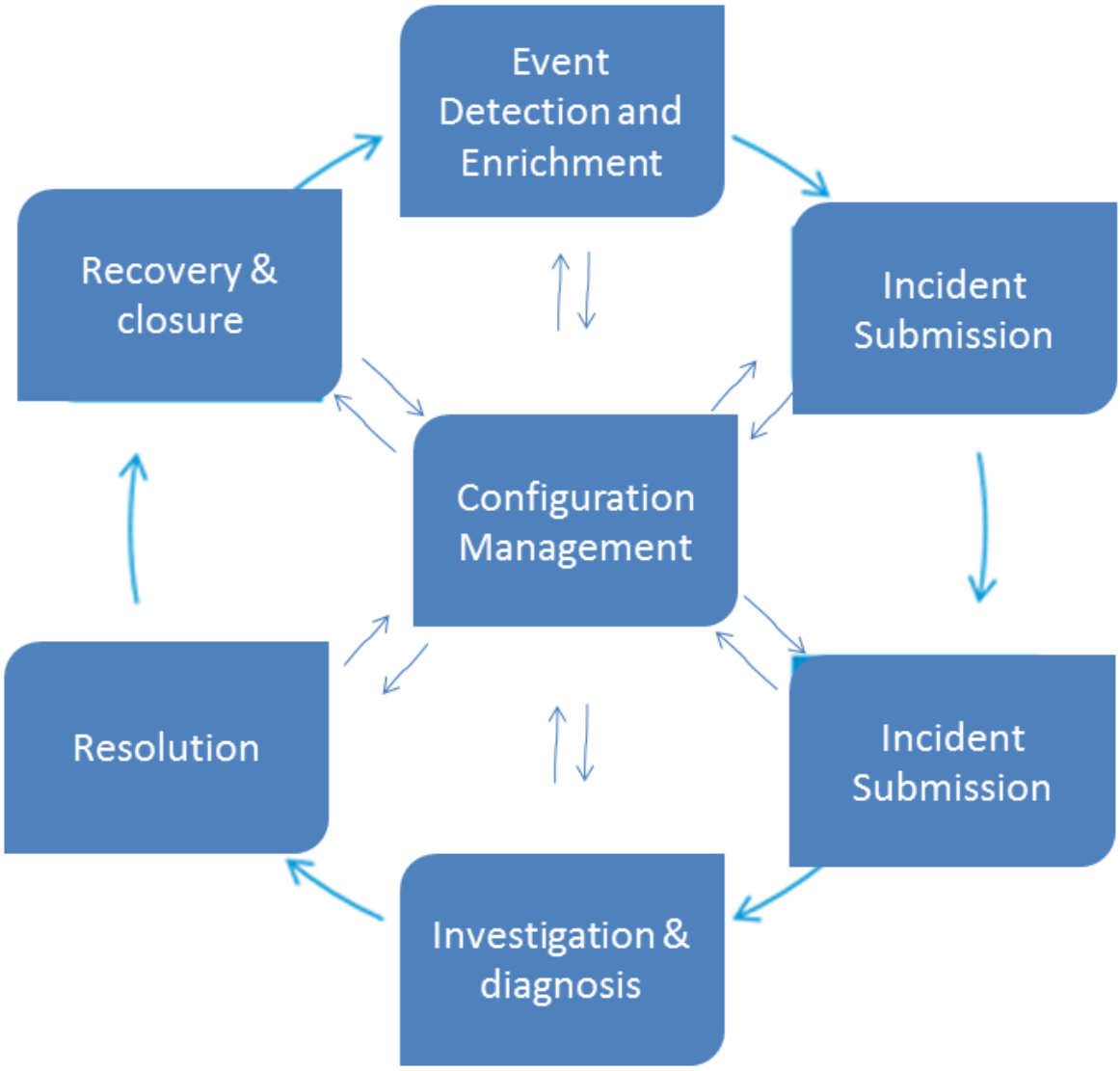
Major Service Modeling Use Cases

Now that the individual uses for services models are understood, they can be assembled into some general use cases that apply broadly to most IT organizations.

This section includes:

Closed Loop Incident Process (CLIP)

CLIP is the solution that connects event management to incident management and breaks the silos between these two processes. Service modeling increases automation and efficiency, reduces Mean Time to Recovery (MTTR) and increases Mean Time Between Failures (MTBF). CLIP creates the common language of CIs and relationships so that the consumer and providers of the service see the same things.



Customers may implement CLIP from any starting point, provided the technology provides the proper integrations. A CMS properly enables all the components to exchange CIs and service models and work together.

CLIP is implemented by exposing service models to all components in the loop. To understand how to create good service models for CLIP, it is helpful to understand the process flow. For a detailed explanation of CLIP flows, see the *HP CLIP Solution Configuration Guides v9.30*.

For event detection and notification, service models can be used for impact analysis and service impact notification.

For Incident management, subject matter experts (SME) may use service models for topology visualization and configuration reference (obtaining detailed information about components involved in the incident). Service Models may also be used to determine root cause and isolate the source of other related problems that may be contributing to the primary incident.

This whole notion of business impact reporting and impact analysis is a strong driver for the CLIP flow.

Usually we notify the SME that investigates the incident, but that is not necessarily the only persona. There are others that may need to learn about this incident that may consume CIs related to any affected services at this point.

Best Practice: Do not build a service model for a consumer that does not need one. If a user only needs a list, or access to a few attributes of a few CI types, it may be more efficient for them to use the UCMDB Browser rather than a service model.

For downtime management, service models can be used in two ways, depending on the timing and how critical the required changes are.

- For changes that can or must take some time, starting from an approved change request, a window is assigned to implement the change.

  For example, an integration with BSM communicates the downtime window so BSM will not create alerts during the downtime. A list of CIs is usually communicated, but depending on monitoring setup and technology, service models may be used as well.

- Administrators typically handle low-level emergency, immediate, or other short-term changes.

  For example, an administrator learns there will be some downtime, but the details are not yet known. The administrator sets up a downtime event for an entire service model for an entire service. All the CIs connected to that service will be shown to be in a **down** state, so BSM or OMi will not send alerts. The exact mechanism of either turning off the monitor or not creating or sending the event is in the domain of those monitoring products. The service desk technicians will then know the status of all the CIs and can communicate this to the business users.

  Service models provide the ability to generate a downtime event for an entire service model's CIs, including the **BusinessService** CI and everything underneath. All the components use the same service models—for example:

  - HP Service Manager (SM) defines a blackout window.
  - Integration to BSM disables event generation during the blackout window.
  - UCMDB Configuration Manager identifies impacted CIs and services and reports any missed CIs not identified as impacted during the blackout window.

Change Control and Release Management (CCRM)

CCRM is another main use case using service models. Like CLIP, CCRM implements a closed-loop feedback/management process.

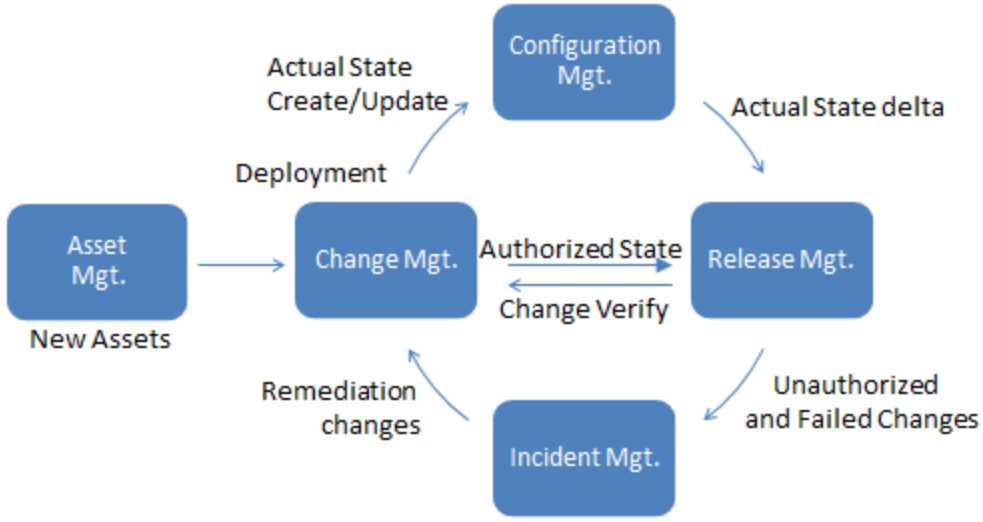
Whereas CLIP is post-incident, CCRM spans the entire change management process. The goal of CCRM is to ensure only authorized changes are made to services, and to detect and prevent unauthorized changes.

Using data from both configuration management and change management, changes can be verified; that is, correlated to an approved Request For Change (RFC). CCRM provides a common view of a service to the CAB, the change implementors, and the change requestors, so that a precise dialog can be conducted about changes:

- Excessively risky or problematic changes can be assessed as such and disapproved before any change is made. A history of changes is collected and analyzed to assess elements of risk.
- Any discovered changes immediately become part of the change process, so there is little delay between the change and the verification of the change. If any change is unauthorized, remediation can take place as soon as possible. In the case of a change that does more damage or incurs cost the longer it is in place, early detection is essential to remediation and for improved MTTR.
- Consistent impact analysis of services related to changed CIs is enabled.

The schedule of downtime Forward Schedule of Change (FSC) can be communicated quickly enough to reschedule changes without last-minute disruptions to change schedules.

A basic diagram of the CCRM cycle is shown here:



The reporting and correlation capabilities are based on the service model. For more information, see the *CCRM 9.30 Solution Configuration Guides* in the UCMDB product documentation library.

Business Service Management (BSM)

BSM is a separate domain, with its own product line. Service models are consumed regularly in these environments. BSM is usually a consumer of CMS data and service models. Modeling for BSM has a separate best practice document, *Effective Modeling for BSM: Best Practices*, which describes specific requirements for modeling for BSM. Generally, BSM models include precisely what is to be monitored—no more, no less.

During and after disruptive events, models are consumed for impact analysis and problem isolation (root cause analysis). As problem and incident management processes execute, models keep operations and the Service Desk synchronized.

For example:

- When I see a server having problems, how do I know what business services are being impacted?
- Who should be notified?
- Who should be assigned to resolve the problem?

All of these processes may use service model impact analysis.

Operationally, BSM and UCMDB must work together and exchange service models. If BSM reports new CIs from its monitors, these new CIs may represent new parts of a service.

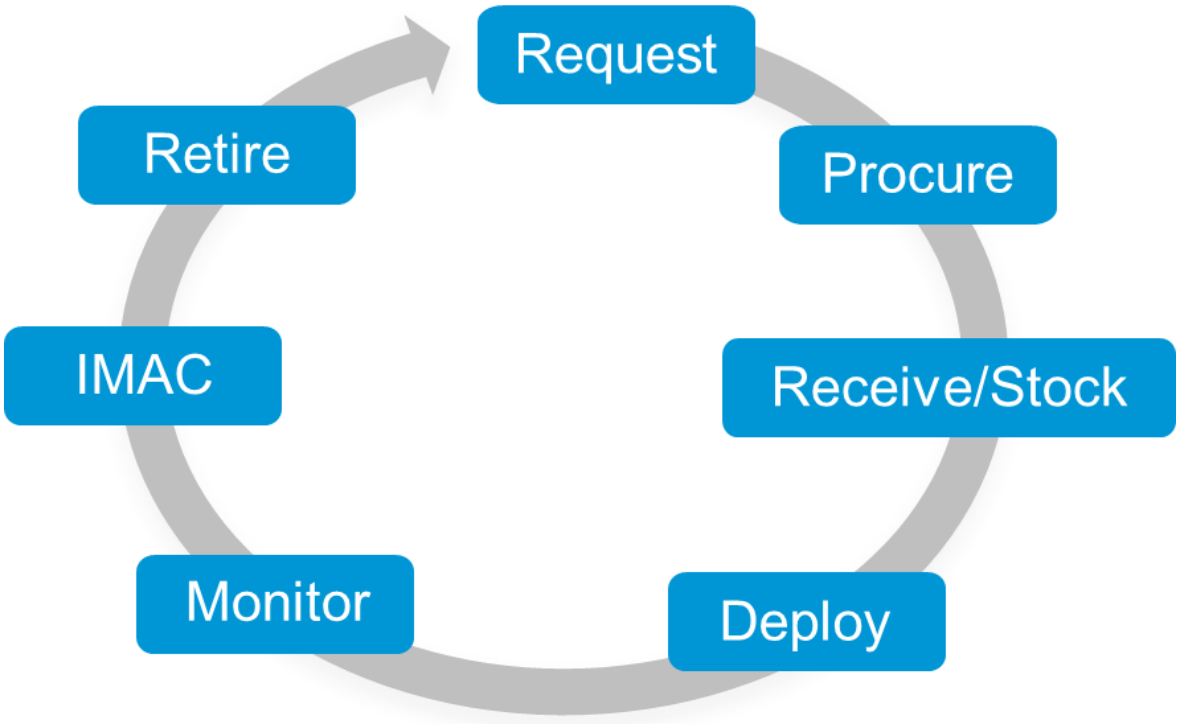
The Configuration Manager should work with the Operations Manager to ensure that CIs are exchanged properly between BSM and UCMDB:

- Aged CIs should not become **zombies**; that is, untouched by anything besides the integration.
- If a service model should be updated, newly reported CIs should be under Change Management.

Service Asset and Configuration Management (SACM)

SACM is the ITIL process that establishes the service life cycle. ITIL defines SACM as a process that manages the service assets in support of the other Service Management processes. According to ITIL, SACM's objective is to define and control the components of services and infrastructure and maintain accurate configuration information on the historical, planned, and current state of the services and infrastructure.

SACM enables IT to connect activities and investments of IT Asset Management and Configuration Management, accurately record and track the various costs associated with delivering IT services, and maintain an accurate picture of the organization's IT infrastructure used throughout the IT Service Management processes.



The cycle shown here begins with the asset management processes that acquire and deploy service components prior to those components being discovered. CIs are provided to Configuration Management by the Asset Management function. This is to enable IT processes to work on the deployment and provisioning of a new service while using the CLIP and CCRM processes. For example, an RFC must still be created to install software on a server, even though that server is not yet an auto-discovered CI.

Once the service components are deployed and the service begins operating, SACM prescribes and implements the control functions for monitoring, changing, and administering services throughout the life of the service.

Service models are updated by new CIs in the CMS from different sources. For example, if Asset Manager is used to create a new server asset, the CMS should see this and all the consumers should act accordingly. For example, BSM might not deploy monitors because the server is not yet online. However, Service Manager would accept the new server CI in order for subsequent RFCs to be created to continue provisioning and deployment of the new server.

For more information, refer to the *HP SACM 9.30 Concept Guide*.

Chapter 3: Service Modeling Approaches

Once the goals, consumers, and use cases are understood, a model can be constructed. There are two fundamental ways to think about modeling—bottom-up or top-down. Bottom-up is the older approach, and is problematic. Top-down modeling is the modern approach, which is more reliable and makes it easier to build models.

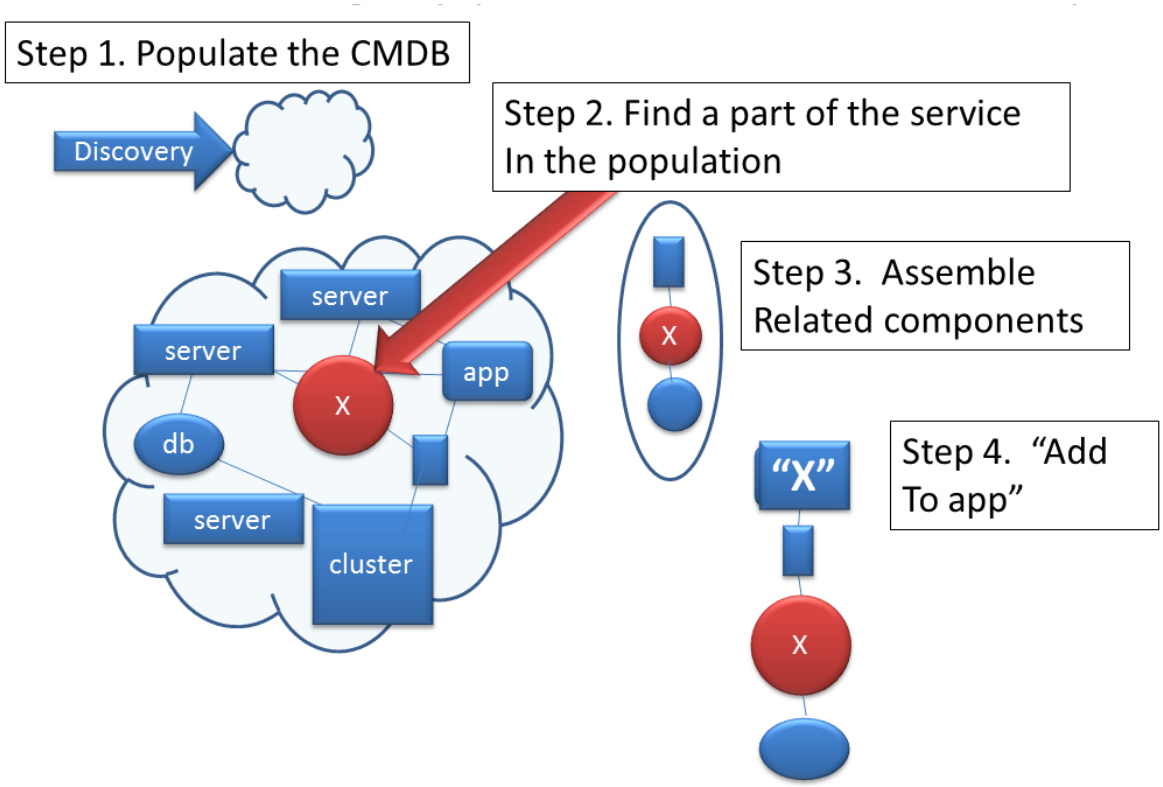
This chapter includes:

Bottom-Up Modeling

From an IT perspective, the service is a collection of infrastructure working together. IT consumers tend to think about services in this way, since they usually support some subset of this infrastructure. Service modeling by IT, for IT, with infrastructure-focused contents, and most importantly, defining a service by its infrastructure, is called bottom-up modeling.

The general approach to bottom-up modeling is to first discover the infrastructure, then identify applications and architecturally unique characteristics of the application as it exists in the infrastructure—for example, the name of a database. A database name is an architecturally unique characteristic of an application-specific component. Names of components such as databases are a proven method of application mapping and service modeling.

A basic illustration of the bottom-up approach is shown here:



This is not the best approach. Because bottom-up service modeling is based on naming conventions, it is subject to the quality and accuracy of the environment's past architectural naming decisions as well as all the operation since then.

Architecture can be defined as planning that which will be difficult to change, and an organization's naming conventions can be categorized as architecture. While bottom-up service modeling can potentially work well, it may also be impossible to use if the organization's naming conventions cannot be used to reliably or consistently identify service model components.

There are many sources of difficulties that make bottom-up modeling problematic, including:

- non-existent or inconsistent naming conventions
- virtualization
- security restrictions

In any given IT environment, it is likely that at least one application will be problematic to model using a bottom-up approach.

Since traditionally over time things that are static become more dynamic, the bottom-up approach has become less useful. A more modern way of thinking about service is needed.

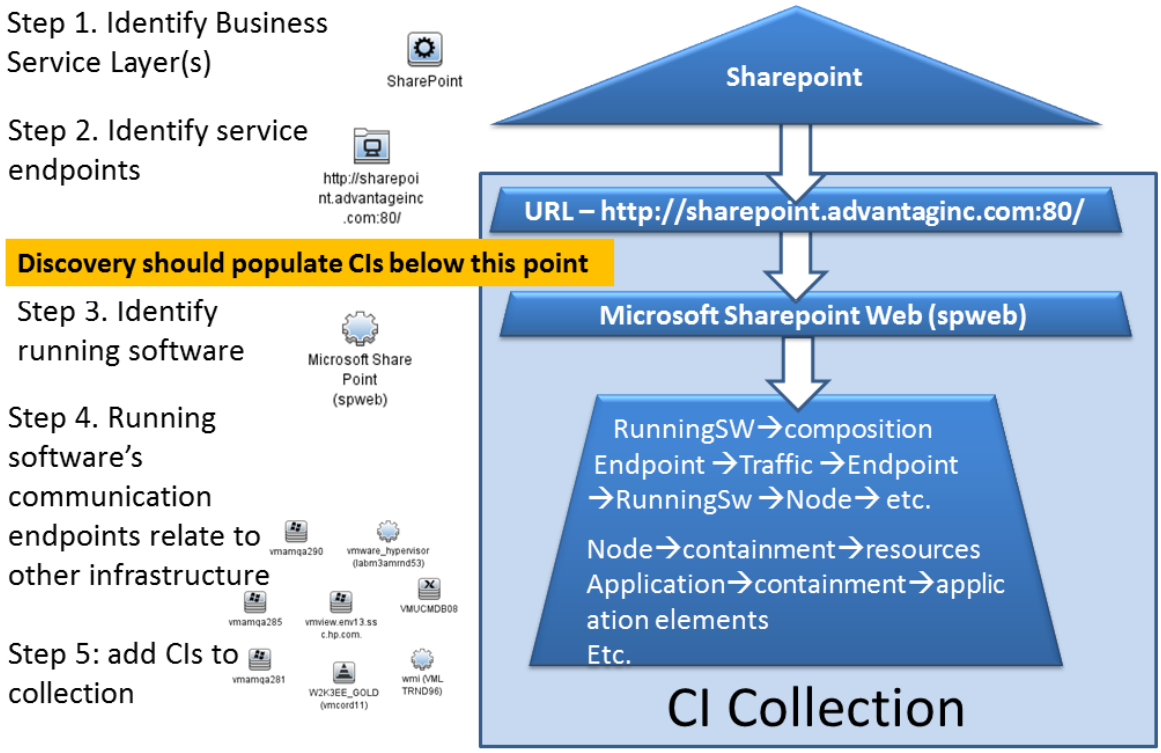
Top-Down Modeling

From an end-user perspective, the service is a consumption end-point. The consumer is insulated from the internal processes and resources required to provide the service. All these capabilities and resources are Service Assets. A service as modeled and viewed starting from the consumption point is called top-down modeling.

In modern IT terms, a service is composed of more services, each more specific in nature. Services consume and provide from each other in order to provide the overall service. Modularizing services in this way allows for greater control of change over the lifetime of a service. Top-down modeling ties the actual operating service more closely back to everything that comes before production operation, such as:

- Business case requiring the service
- Design and development
- Deployment
- Parameters for how the service operates

A basic illustration of the top-down approach is shown here:



Both the top-down and bottom-up approaches rely on discovered relationships to model a service. The starting point is the major difference. The top-down approach uses service entry points and the running software that creates them—something that is always known—whereas the bottom-up approach uses infrastructure component names traditionally associated by ownership or other human knowledge. This type of information is often difficult to find and obtain, or may not exist at all. Service entry points will always exist (otherwise the service could not be consumed).

Top-down modeling focuses on the consumer and the services they consume. The terms, visualization, and questions asked during planning—everything—becomes more closely tied together throughout the life cycles of a service.

Since top-down modeling is the best practice, the rest of this document assumes a top-down modeling strategy.

Top-down modeling does not call for a specific type of topology discovery or inclusion of infrastructure. The exact contents of the CMS will vary depending on what is able to be discovered and what is required for the use cases. Impact analysis and topology visualization can sometimes omit lower layers of infrastructure intentionally.

The top-down approach makes it simpler to include more CIs The best practice is still not to include something in the model unless it will be used. Resist the temptation to create 100% perfect service models using all possible infrastructure CIs, unless the use cases absolutely require it.

One example is performing impact analysis to determine potential impact to services based on a change to one server. Anything that makes the server unavailable could reasonably have the same level of impact. Therefore, a service model used for server impact analysis would not benefit from having the network interfaces and all the host resources included in the model. The server would suffice to propagate impact.

Consider using CI collections, or models to contain all the CIs in a service. These collections are useful for ad hoc reporting and essentially compose the service catalog. UCMDB can produce the service catalog by a TQL containing a single CI—**BusinessService**—or a list of business applications using the same technique.

Part II: Service Model Development Cycle

Chapter 4: Service Model Development Cycle

This chapter includes:

Overview

The service assets used to operate a service can cover a wide range of entities. In the digital world, these entities may refer to items such as a network router, database software, or an organizational directory service. Infrastructure items such as these typically work in mixed groups to support various applications, simple or complex, such as a simple shared FTP service or a complex enterprise email system running in multiple clustered virtual environments. By combining various technical service assets for different purposes, IT organizations are able to reduce their costs and also create new and valuable IT services which the business can package and sell as a business service. Therefore, a model of a service primarily defines a logical boundary around a collection of service assets and relationships which help operate the service.

Service modeling can be described as an exercise in defining the logical boundaries around a collection of technical service assets and explains how those service assets support services that are used by the service consumers. This **Application Boundary** problem is a fundamental aspect of service modeling. Shared infrastructure, for example, must be shared; that is, overlap, among service models.

The top-down approach generally states that you define the upper layers of the model first. These upper layers are the business or abstract or non-discoverable layers.

The upper layers are used to describe the service itself, and to show any other sub-services that compose this service.

There are usually one to three layers in the upper tier of a service model.

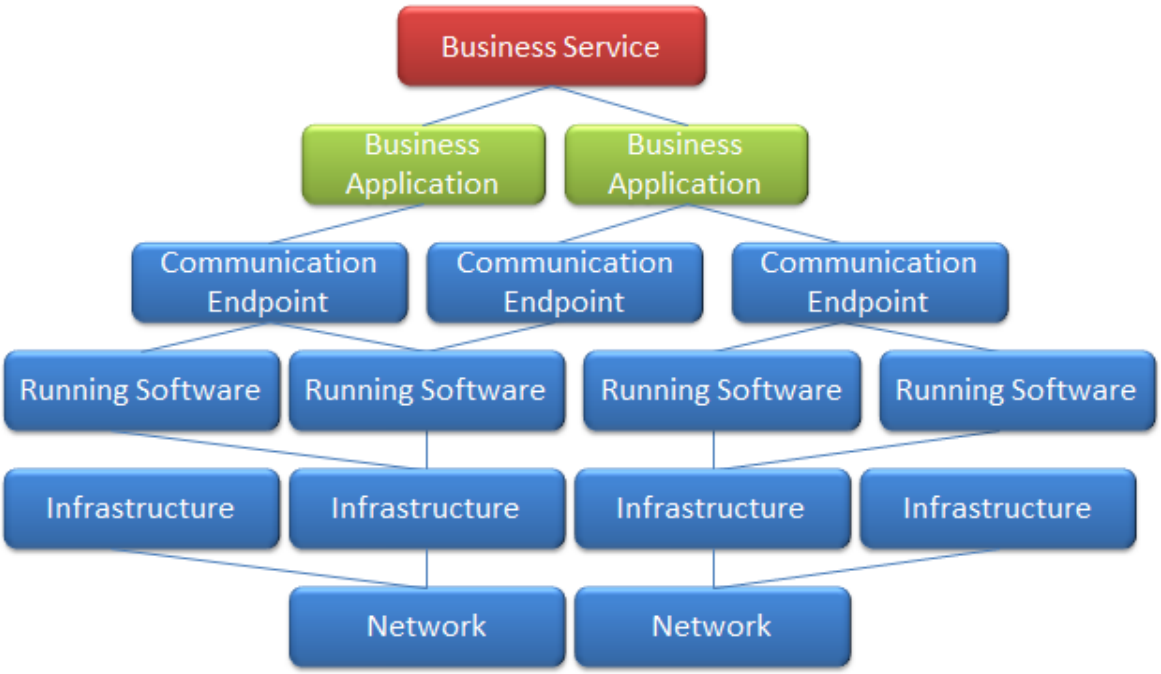
The upper-most layer is a single CI. The CI type is a business element such as **BusinessService** or **BusinessApplication**. At a minimum, this CI is named to reflect the name of the service to be modeled. As a best practice, add other information, such as the type and owner of the service.

Layers 2-n of the upper layers are used to show services that are composed of other services. If a service was composed of no other services and only consisted of specific infrastructure, there should be only one upper layer. Layers 2-n CIs are usually of the **BusinessApplication** type or similar CI types.

The upper layers are complete when everything about the service, except its infrastructure, is described. Primarily, this is the upper-most CI, and includes all of the CIs representing other services that compose this service.

The lower layers are understood as the discoverable, or actual, or infrastructure layers. The service endpoints will identify the relationships from the upper layers to the lower layers.

For example, using the standard set of layers of a service model, the following diagram depicts an example of an abstract service model:

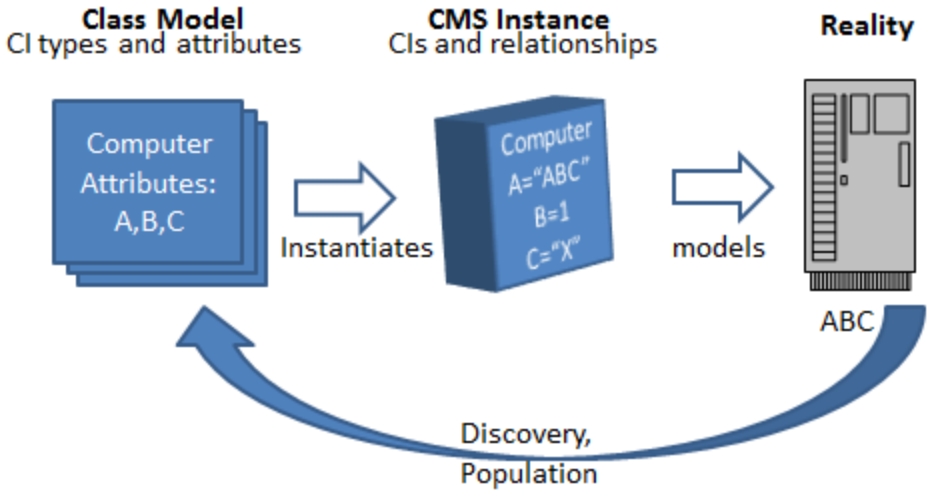


The diagram displays a generic hierarchy for a single business service. Applying the Sharepoint example to the figure above, Sharepoint could be the business application that relies on various IT services such as SPWeb and SPSearch (Business Application boxes). All applications have service endpoints which are related to running software. The service endpoints are traced to all infrastructure required to make the service endpoint available and functional. The communication infrastructure, such as routers, switches, and interfaces, are shown as the Network layer.

It is important to note that the business service in the diagram above could also be viewed as a service asset for a higher level business service. For example, the Sharepoint service may be viewed as one of many internal applications that make up Advantage, Inc. Services may be and are, in fact, commonly composed of other services.

At this point we should understand conceptually the layered approach for service modeling and the purposes of the layers. How and where should these layers be assembled? What is added first? What are the critical paths?

It is important at this point to understand the difference between a service model and the class model used as part of the CMDB and CMS. Any service modeling must be done in the context of a class model—a data dictionary that defines what CIs and relationships can be used in the models.



The class model contains all the definitions of the CIs and relationships that can be created. The CMDB contains this model, as well as the actual CIs and relationships instantiated from this model. The CIs and relationships are populated by discovery and other methods to form a CMS. The CMS is consumed by querying data through service models. The service models represent the reality in the actual data center, so when the consumers use configuration data in the service models, they are always working with some aspect of maintaining, monitoring, or changing that service.

Modeling Quick Reference

This section includes:

Overview

This section is a collection of modeling tools and templates which can be used to quickly create Service Models. This section can be thought of as a cookbook; a collection of recipes that may or may not work for a given situation.

Examples are shown using UCMDB queries and views. The examples are presented in roughly sequential order, but may be used in any order as needed.

CMS Content

One problem with service modeling is the variability of the CMS contents available to model. CMS contents differ based on many variables:

- Use cases on which the CMS is founded—what is expected versus what is available
- ITSM strategies and product portfolio—technical capabilities of IT
- Budget and organizational executional capabilities of the business and IT
- Time and expertise availability/constraints
- Organization's sector—policies may restrict credentials, discovery, or access to SMEs
- Ability to continue discovery—one-time discoveries tend to stagnate
- Ability to discover additional relationships and CIs as needed

For these reasons, it is impossible to provide a single service model template which works in all cases for all organizations. Service modeling strategies may need to be modified based on the available data and resources.

Service Models should also reflect the organization's way of thinking about its services. For example, if a business thinks of its services hierarchically or flat, the upper layers of that business's service models should reflect that. If an organization thinks of its service catalog as a flat list of atomic services, the upper layers of its service models should be flat.

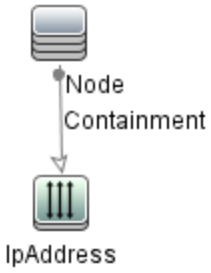
The same holds in the infrastructure layers of a service model. For example, if the relationship: **UriEndpoint** < **composition** < **Node** does not exist, the same use cases may be modeled using an indirection relationship, such as: **UriEndpoint** < **Composition** < **RunningSoftware** < **Node**.

Service Model Building Blocks

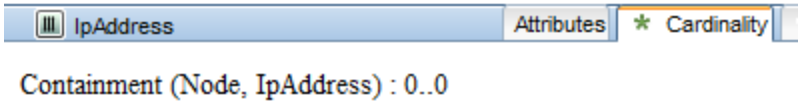
This section contains query fragments that can be used to build service models and perform other related tasks.

Discovery Verification

In the modeling process, it is common to find missing data which requires additional discovery or population. Queries can be constructed to systematically or randomly verify the presence of the required data. Here is an example to check the progress of discovery—for example, to verify that the credentials supplied for host discovery were correct and that the required network connectivity was in place.



This query consists of only four parts—a Node containing an IP Address as shown above, and a cardinal condition for the IP Address as shown below.



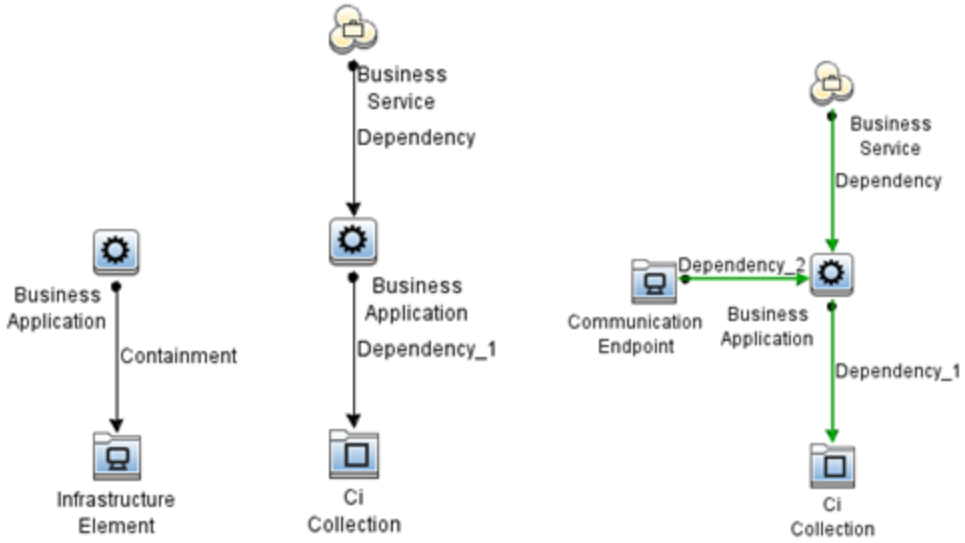
This TQL shows all IPs connected to exactly zero hosts.

Note the use of cardinality zero on the IP address CI type. If an IP's host is expected to already be discovered—that is, following host-level discovery—then this TQL will show IPs that remain unconnected.

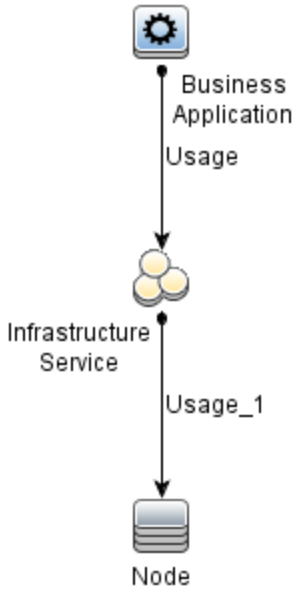
If these unconnected IPs belong to any devices which may be needed in a service model, then the service model will remain incomplete until the host is discovered.

Service Model Upper Layers

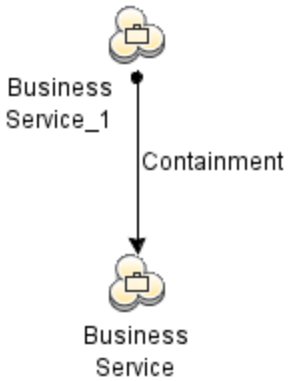
Multiple methods of organizing a service model's upper layers have been used successfully. The keys are to have a single CI at the top and a CI to hold links to infrastructure. Shown here are several TQL queries of service models which have been used in the past.



These are all acceptable ways to organize your hierarchy of services and applications in the upper layers of the service models. The Infrastructure element is used to represent any other CIs that may be related. A CI collection is shown as a container for all of a service's CIs. The communication endpoints will be related to other CIs in the CI collection. The **CommunicationEndpoint** is shown directly related to the **BusinessApplication** (or **BusinessService**, if desired) because the service's point of consumption must be known to use the service, therefore must be known to model the service.



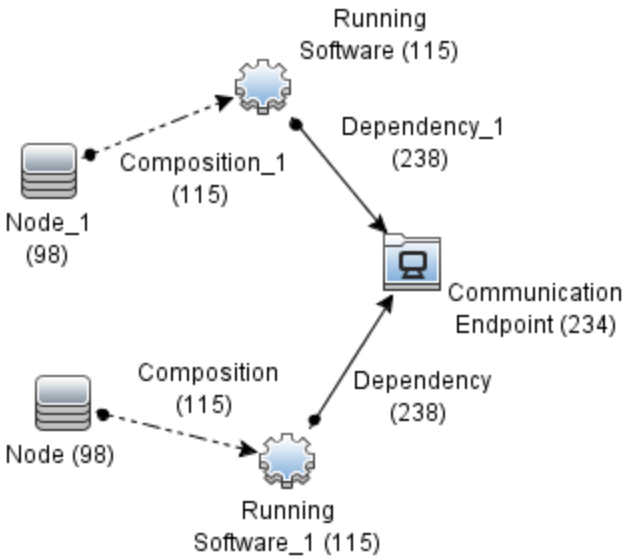
This example shows a method of modeling **InfrastructureService** CIs under **BusinessApplication** CIs. Note the relationships. The service is connected to the nodes (physical or virtual machines) where the application uses it. The application used by infrastructure uses a **usage** relationship. The **InfrastructureService** component is used for nodes that have the same function for a **BusinessApplication** grouping. The infrastructure service makes, in turn, a **usage** relationship to one or more nodes. The life cycle of these nodes can be determined by the service that delivers these servers.



Self-containment of business services is allowed in the class model without two nodes in a TQL, but it is shown here to emphasize the point.

Find Generic n-tier Applications

Service models should be constructed using the least granularity possible. However, this does not mean service models must be constructed CI by CI. Reusable components save time and effort. Simple service models may consist of only a few CI types in their queries. Many complex models can be composed of rather standard tiers—for example, the traditional 3-tier application consisting of database, application, and presentation tiers. Queries for standard tiers can be used as building blocks similar to the upper layer templates in the previous section. Below is a TQL to model a single tier of an n-tier application.

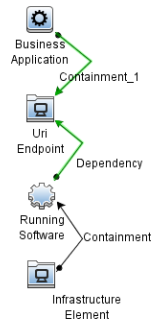


By using **RunningSoftware/CommunicationEndpoint** chains, application structures can be revealed generically without knowing whether the applications exist or the application's structure. Continue the chain by connecting additional running software chains to one of the nodes, and link as shown above.

HP Universal Discovery will discover the above CI chain using out-of-the-box discovery jobs. However, differences in standard infrastructure may require modifications to discovery and the corresponding sections of service models to accommodate the differences. For example, old or new versions, security hardening, in-house or custom-developed applications could all require adjustments to both discovery and service models.

Link Upper and Lower Layers of a Service Model

This TQL query fragment shows how the upper layers of a service model are linked to the lower layers using a **CommunicationEndpoint** CI—in this case, **UriEndpoint**, a sub-type of **CommunicationEndpoint**.



The **UriEndpoint** CI is identified (and discovered at some point) as the primary entry point of the service. The running software that instantiates the listening port of the entry point is also discovered and related.

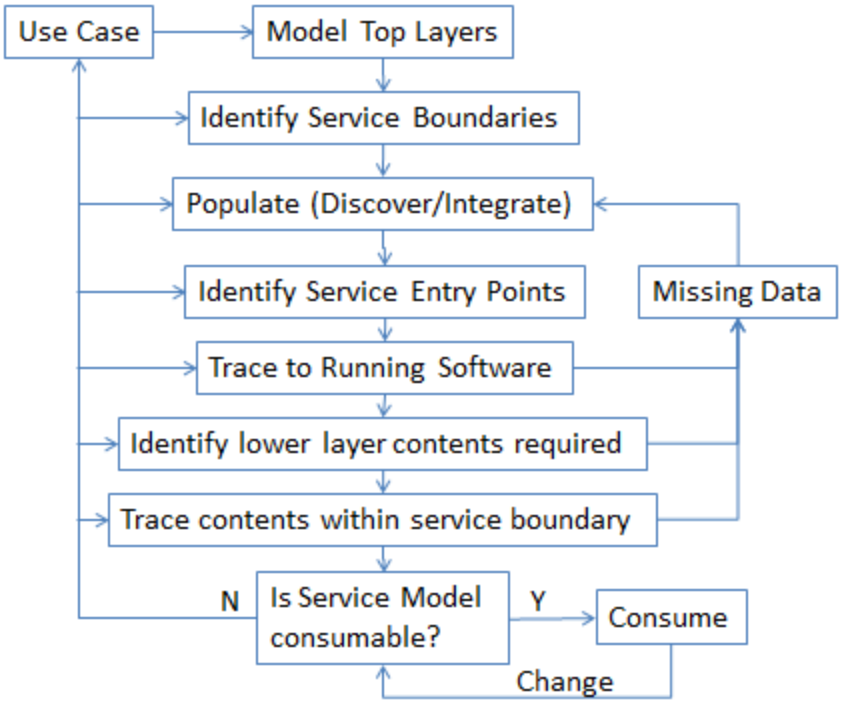
The URI endpoint is known and manually related to the lowest-level CI of the upper layers of the service. The running software is followed through its relationships to other infrastructure to map out and model the rest of the service's infrastructure.

Modeling Process

This section includes:

Overview

The preferred model development flow is shown below. Start with the Use Case at the upper left, and follow the arrows.



This process is iterative and is composed of repeatable processes to create service models. Missing data will be revealed by validation and may require additional population. The goal is to do precisely what makes the service model consumable—no more, no less. Once the service model is consumable, time passes, change occurs, and at some point a change to the service model will be necessary for it to remain consumable.

As a best practice, the top layers should be the first layers to be created. The processes that design, develop, and implement services also create service models—as diagrams, white-board drawings, and so on. This design information should be captured and used to create the upper layers of the service model, the business service, and IT service layers.

The connection between the known list of business services and the discovered IT infrastructure falls within the modeling layers. Even if the entire stack is [1]discoverable, the process of collecting the information is not enough. The model boundaries are use-case dependent. Therefore, they must be

[1]**Discoverable** really means the data is in electronic (and usually but not always, structured) form. A few shops have matured to this point where there is a discoverable connection between infrastructure, applications, and business services. Change and Release Management process maturity can enable meta-data to be placed and maintained on production servers that represent relationships between that server and the services of which it is a service asset.

defined. The ultimate choices of what to include and exclude should be clearly enumerated by the consumer on-boarding process. For more information, refer to the *Consumer Onboarding Guide* in the CMS Best Practices Library.

Most organizations must collect at least part of the top of the stack manually. Preferably, an application should be used to collect the business CIs and information which relates to the service endpoints or, if necessary, at least a part of the infrastructure.

The following steps describe the flow in more detail.

Step 1: Use Cases and Research

Part I of this document is dedicated to understanding how service modeling will benefit an organization. Building service models is part of an overall service life cycle with many other associated activities and processes. If done properly, use cases will be commonly known before developing service models. Plan the service modeling process by linking to higher level objectives which influence planning factors such as:

- What is the granularity the company wants to define for different business services? What is everyone used to? Is there a need to change the norm?
- Which services and applications have priority for modeling? Who are the most important consumers?
- What services will bring the most value for the least effort? What is at stake for the use cases?
- Who are the stakeholders of the service models? Who sponsors the consumers?
- What processes do you hope to improve by modeling services? What are the KPIs for the consumer processes?

Modeling services which support specific processes such as Change Management may also influence implementation factors such as depth or breadth of service views or the rate of refresh for discovery sources. The consumers drive the providers. The overarching goal in any service modeling process is to model once and consume anywhere.

Step 2: Model Top Layers of the Service

Once the use case is defined and understood, modeling begins by assigning a name to the service. The use case generally supplies the name and is meaningful to its consumers.

You may or may not use an Enterprise Architecture (EA) product to manage the top layers of the business services. Large organizations with many business services usually need a solution to manage them all. Applications like Troux®, ARIS®, or Casewise® can be used to define the top level business service layers, and integrate with HP Universal CMDB (UCMDB) to supply these to the CMS, ready to use in service models by relating them to the lower levels of running software and other infrastructure.

For more information, see Troux (http://www.Troux.com). Software AG (http://www.Aris.com), and Casewise (http://www.Casewise.com).

If you do not use an EA application to define the top level business layers, there are several options to create them:

- HP UCMDB user interface

  The UCMDB Modeling Studio can automatically create a **BusinessService** or **BusinessApplication** CI and relate it to everything in a query. This arrangement is called a model and can be used as a service model.

- HP UCMDB Browser
- HP Service Manager (SM)
- An integration to an external list such as a spreadsheet, text, or database

In order to properly model services, it is important to capture certain critical information about the services. Relevant questions may include:

- Who are the SMEs that can help validate models or be given the task of owning the models?

  It is important not only to know whom to contact but get mutual agreement on the business priority of the service modeling task. Interview SMEs to identify what service assets make up the service. In cases where the service asset is an application, architectural diagrams and installation guides are a big help if there is no existing knowledge on the application's structure.

- Who are the service owners and does their role need to be redefined?

  It is important to understand the existing service assets and their structure and also who are the business and technical owners of these assets going forward.

- Is there a definitive repository for business service names/definitions/sub components such as an organization-wide service catalog or enterprise architecture tool?

  These are the names of the logical entities that you will eventually tie to the underlying resources in the CMDB. Answering this question may result in additional work during discovery to synchronize this data with the CMDB.

Step 3: Verify Required CIs and Relationships in the CMS

In the past, all modeled data would reside in a single database—the CMDB. Today, not all modeled data is necessarily inside the CMDB. The entire CMS, all of the integrated systems, with a CMDB at its core, is thought of as the repository, with many providers exposing data in place, retrieved at query time through federation, merging with a set of core CIs in the CMDB acting as an anchor.

The use case will call for a set of CI types and relationships. The configuration manager must work with the consumers, providers, and owners of the data to decide what is to be left in place, what will be discovered, and how configuration data will be provided to consumers. From this study, a discovery plan will emerge.

Does the discovery plan cover all the technology and environments necessary to model the prioritized services? SMEs can help define the scope or depth of discovery required to accurately recognize

service assets utilized by the business service. Any gaps in coverage may require customizations to discovery such as enhancing discovery packages or customizing features such as the Application Signatures feature of UCMDB.

Thorough research at the inception stage of a service modeling initiative will significantly improve the chances of success and will most likely improve related processes such as Discovery Planning.

Step 4: Identify Service Entry Points

After the top-layer CIs, the next layer is the entry points of the service. Because there are relatively few networking protocols, most entry points are based around common addressing schemes. In a TCP/IP world, the entry points can be expressed as:

- URLs for Web clients
- URIs for Web Service clients
- Known domain name server (DNS) or domain or server name (load balancers or redirects connect the entry point)
- Java Database Connectivity (JDBC) connection point (if the service is a database service)

Other connection points, such as mainframe technology, will use equivalent but different technologies and formats of entry points, but these can still be used in the service modeling process provided the technology can be populated into the CMS. Most protocols—such as SNA, IPX, Vines—have TCP/IP bridging technology allowing at least basic discovery without much additional time and effort.

Examples of service entry points:

- https://sharepoint.advantageinc.com (domain name)
- http://www.advantageinc.com:80/sharepoint (virtual directory)
- https://google.com (Internet application)
- https://www.ImaWebService.myco.wslb1:8085/ws/SAPws.uriLB1.DCAEP1.ws.li?wsdl (internal web service)
- “file:c:\config\SAP1\config_data_containing_entry_point_here.cfg” (configuration file as the entry point)
- sql://SPSQL\POWERPIVOT (entry point for a database service)

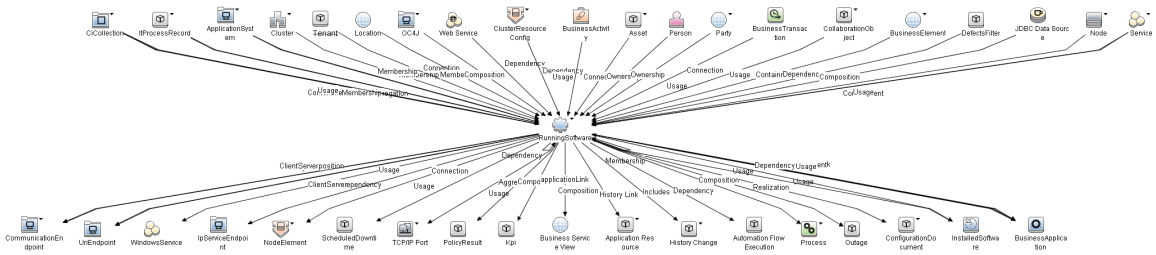
More entry points and the CIs that represent them will be documented in "Service Models in UCMDB" on page 63. Entry points are represented as **CommunicationEndpoint** CIs.

Step 5: Identify Running Software

RunningSoftware is a key CI type. It allows granular tracing of entry points to other infrastructure. It acts as a bridge between the upper and lower layers of the service model. UCMDB Universal Discovery (UD) can discover running software in several ways and is fairly comprehensive, so it is a reliable way of connecting entry points and infrastructure.

Software running on production servers will always be communicating with other entities such as clients, databases, and other servers. Mapping these communications provides a fairly accurate picture of a real operating service that can be traced through abstraction layers such as load-balancing, clustering, virtualization and storage networks.

Running software is always discovered in context with its relationships to other CIs. **RunningSoftware** and its many sub-types can be related to many other CI types:



Many CI types (and their sub-types) may be related to **RunningSoftware**, including these commonly used in-service models:

- **CommunicationEndpoint** (**IpServiceEndpoint** and **UriEndpoint**)
- **BusinessApplication**
- **BusinessElement**
- **Service**
- **Node**
- **JDBCDatasource**
- **Process**

Step 6: Link Software to Communication Endpoints

Running software and communication endpoints are actually parts of the same thing. **RunningSoftware** has many sub-types which account for all the application tiers, including:

- **WebServer**
- **ApplicationServer**
- **Database**
- **MessageQueueResource**
- **Cluster**
- **Virtual Host Resource**

For example, the **RunningSoftware** CI sub-type may be an Apache Web server, an SQL database server, or a Websphere application server. Running software creates and listens on ports that are configured as entry points; in other words, URLs, URIs, and other connection protocols. A Web server accepts a URL, which is an underlying communication endpoint (and in fact this is how the UCMDB data model represents this). All that is required is that the discovery jobs must be executed to discover both the running software and the communication endpoints. The relationships between these will be created by the discovery jobs.

For example, **BusinessService** > **BusinessApplication** is created by hand. Part of the process is to obtain the entry point of the service.

The users and owners of a service are the best places to obtain service entry points. Since entry points must always be provided for access, they are generally well-known and easy to obtain.

The entry point is used to find the service's **CommunicationEndpoint** CI. The UCMDB Modeling Studio is used to easily relate the **BusinessService** to the **CommunicationEndpoint** CI and any other related CIs. For more details, see Part III: "Service Models in UCMDB" on page 63.

Now we have **BusinessService** > **BusinessApplication** > **UriEndpoint**.

Follow the **UriEndpoint's** relationships to **RunningSoftware** CIs and add these to the model.

Then we have **BusinessService** > **BusinessApplication** > **UriEndpoint** > **RunningSoftware**.

For example, **Collect Money** (business service name) > **Online Retail** (service) > **http URL** (endpoint) > **Apache** (running software).

The **RunningSoftware** CIs now enable the top layers of the model to be related to the bottom layers.

Step 7: Model and View Bottom Layers

This section includes:

Overview

The next step is to evaluate what is left and finish the model. The use cases will drive this. The **RunningSoftware** CI has relationships to the infrastructure on which it is running.

Some best practices to keep in mind:

- **Model splitting.** If possible, allow multiple consumers to use common models, but avoid non-overlapping CIs. If there is a small non-overlap, this is usually acceptable, but be wary of scope expansion. If necessary, split the model into multiple models. This is a normal part of IT evolution, so do not be alarmed if some service models, in great demand, are asked to be expanded on a regular basis. You must determine a cutoff point that is acceptable for your needs on how much and what kind of non-overlap warrants a model split.
- **No extra CIs.** Do not populate the CMS today with CIs that only solve tomorrow's problems. Do not overpopulate the CMS or your service models **just in case**. Service models are an excellent example of the axiom **less is more**. For example, if **RunningSoftware** is used to create a service model, but this is not called for in the delivered model, then it should be hidden from the delivered results—no just-in-case CIs.
- **Use case priority efficiency.** Some use cases call for comprehensive discovery of a given CI type. Use cases such as CCRM and CLIP have the scope of the entire data center. Large organizations may have hundreds or thousands of service models. Time and cost efficiency is important when developing large numbers of service models. As a best practice, carefully prioritize the order in which you develop a large number of service models. Start with the models which will serve the most important consumers.

  Add the remaining required CIs and relationships to the model—trimming and adjusting as needed—until the service model delivers precisely what is required by all of its consumers. Do not be concerned at this stage with presentation and visualization. Do not be concerned with how to present the models. Just assemble its contents. Other tools will be used to further subdivide model contents into specific data sets for specific consumers.

Creating the Presentation Layers of the Model

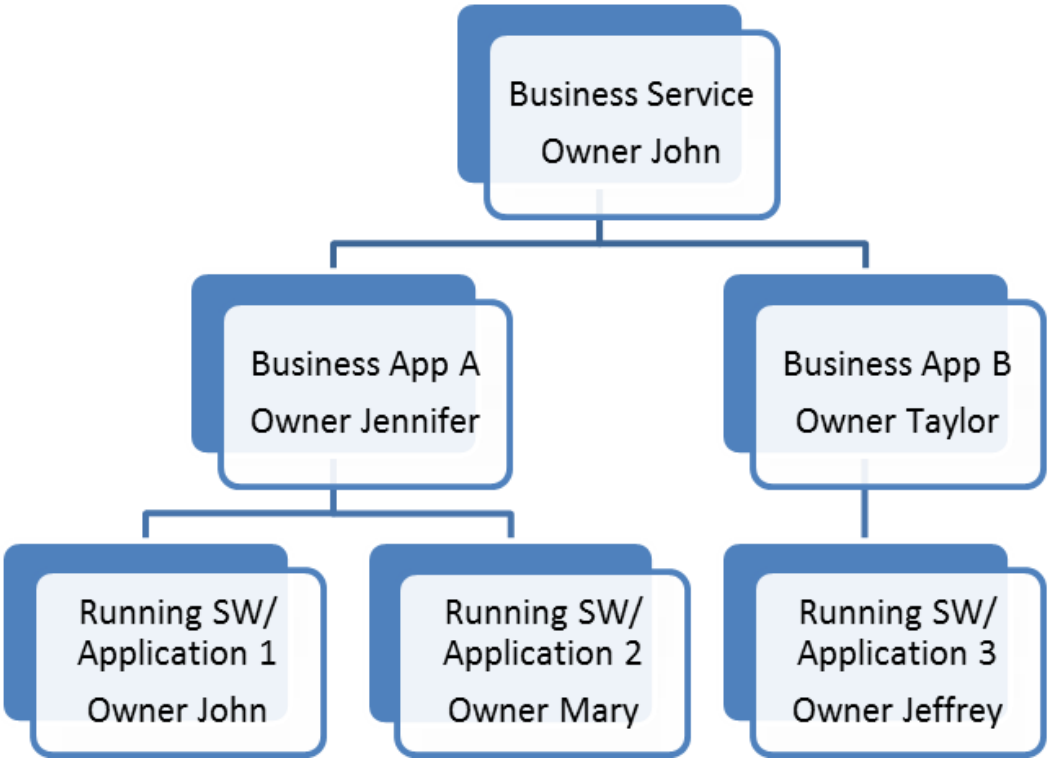
Validate the contents with the consumers or owners of the use case, and iterate the modeling process until the contents and model are correct. As this process occurs, use UCMDB Modeling Studio. The correct details of consumption will be forthcoming.

You may or may not stay with a specific type of model delivery as you test the consumption of the model. If a change can make things simpler or save risk or cost, then do it as a normal part of IT.

During this period, create and modify queries, views, models, templates, and perspectives. Do this in a manner that lets you deliver the final contents easily and without having to rebuild the production content afterward. For more information about specific parts of the Modeling Studio, see Part III: "Developing Service Models in UCMDB" on page 64.

Creating and Using Templates

Once service assets can be uniquely identified, use a modular approach to build service asset models. Creating complex service models through smaller, shared, reusable building blocks can help models scale to meet organizational complexity while adding minimum maintenance. The following graphic shows the top layers of business service building blocks. Each of the blocks in the diagram could be re-used in other service models.



The business service model is owned by John and the underlying business applications/IT services are owned by various other members of the organization. By distributing the maintenance of individual business service asset models to individuals or groups, John does not need to worry about the

contents within each service asset component. For example, if Mary changes the servers owned by Application 2, the Business Application model that Jennifer maintains, which depends on both Application 1 and 2, will automatically reflect the change in underlying infrastructure to Application 2, as will the overarching business service model John maintains.

It also helps to create a template of the core architecture that can be reused as a foundation for building models, as well as rendering stakeholder-specific views. For example, to create a template for a generic three-tier J2EE application used by a business service, it may be necessary that each application server utilize a specific database instance and that the application contains a specific domain name in its URL. By applying the unique fingerprints for an application on the generic three-tier J2EE application template, such as a URL which ends with a unique suffix, you can find a specific instance of an application from the production environment. By using a building block approach such as templates, any change to an underlying service asset will automatically be reflected in higher level services which utilize the service assets.

Ownership versus Dependencies

After specific service assets have been modeled, they can be tied together to form a higher level business service. In some cases, it is important to make a distinction between which service assets are owned by a service versus which service assets a service depends on. For example, a self-service portal may own a set of applications that create the portal. but depend on a shared service such as a central single sign-on application. Although the relationships between business services and its underlying service assets can be defined in a CMDB, the list of available business services which need to be tied to underlying infrastructure may stem from an alternative authoritative source. In either case, the CMDB provides features to accommodate multiple paths to service modeling.

Step 8: Reuse, Maintain and Refine

It is essential to continue validating the accuracy of service models to reflect reality. Take steps to periodically review models which have changed or create mechanisms to alert model owners of changes to the structure. Many customers do quarterly, semi-annual, or annual reviews of their service models.

It is a best practice to incorporate service modeling into standard change control practices. Ensure that proper controls are used to manage and refine service models.

The key points are:

- Identify unique service assets
- Build service models in a modular fashion
- Service models should support rendering of stakeholder-specific views

Change control is important because service assets in a model can be owned or depended on by a service. Consequently, the impact of change may not be apparent to any one person involved in maintaining the service.

Good documentation is necessary to aid those who may work with the model in the future. Notes may be added to the models themselves—in the notes and descriptions provided. Maintenance is much

simpler when the inner workings of the model are briefly described—for example, why you are including or excluding things, why you must have a many-to-many relationship, and so on. For specific techniques described in more detail, see Part III: "Developing Service Models in UCMDB" on page 64.

Organizing Models

Any medium or larger-size organization with more than a few dozen services will need a method of naming and storing models and model parts. Enterprise-sized organizations with upwards of 1,000 services and applications must carefully choose naming conventions and hierarchy of services. Without this, it is sometimes necessary to stop building further models, perhaps in the mid-hundreds, and reorganize all the models—a potentially costly and time-consuming detour.

The following is a list of guidelines for avoiding the most common problems:

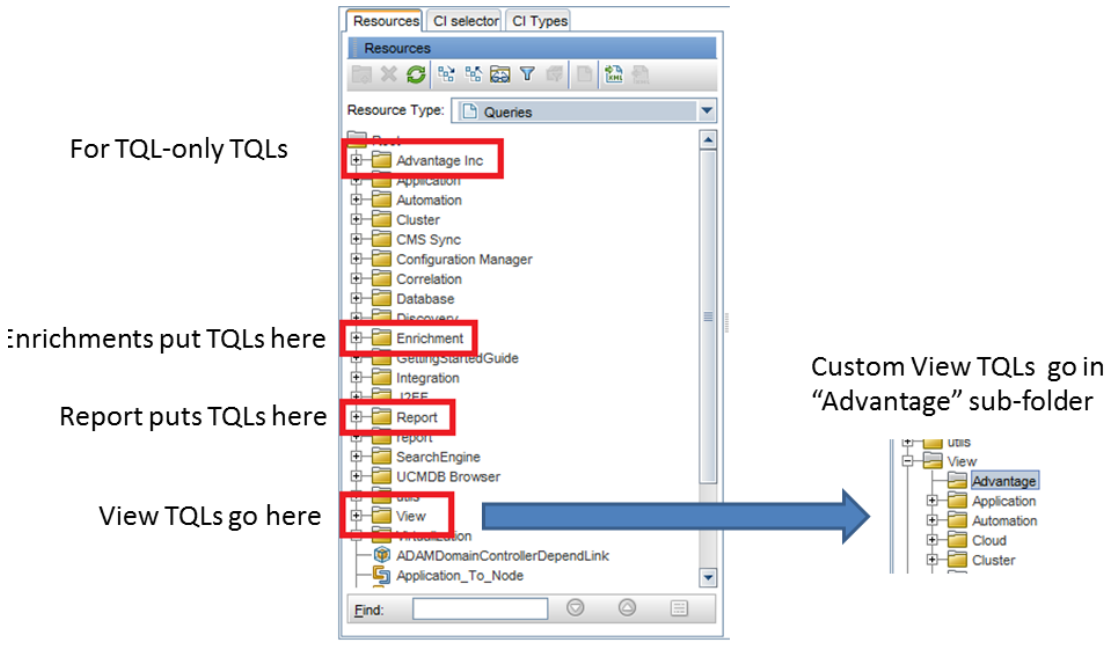
- Having a naming convention for services. One possible format is:

  ServiceName_Customer_Owner_Status

  For example, **Sharepoint_Infrastructure_Advinc_Prod**

- Name models according to the services, such as:
  - Online Banking
  - Online Customer Sharepoint
  - Quote to Cash
  - Shipping
- Names should account for:
  - Name of the service
  - Owner or owning organization
  - Status or production environment (Test/Production/and so on)
- Name all Queries, Models, Templates, Perspectives, Enrichments and Views consistently.
- Create folders for all of the objects for step 4. For queries without views, create a folder—for example, **Advantage Inc** under the query folders. View TQLs will be placed under the **View** subfolder under queries. Also create a user folder here and place view queries in it.

- When creating views, UCMDB will store queries in the View subfolder but not any further subfolder.



This figure shows all of the folders where custom folders should be created for model content.

Service Modeling Challenges

This section includes:

New versus Existing Services

Modeling new services is not as common as modeling existing services. However, lower TCO services can result from tracking modeling requirements from the start of the service life cycle. This means soon after the executive decision is made to create a new service, the service's uppermost CI should be created somewhere, and remain for the life of the service. As the service is developed, used, maintained, and eventually retired, the rest of the service's model comes and goes to reflect reality.

Existing services are generally already hardened and have complex relationships with many entities internal and external to IT. Discovery may be more problematic. Collecting information may be problematic if the original developers or implementers are not available. Some relationships may be unknown, and must be found empirically. The use cases for existing services are often laced with long-term or deeply foundational problems that service modeling may or may not be able to resolve depending on the value realized from the chain of consumers. However, this is by far the most common type of service environment you will be called on to model.

Here are some best practices that may be helpful:

- **Go beyond discovery.** Do as much research and see as many people as you can. Fill in discovery gaps by finding and discussing issues with application, business, and IT service people.
- **Dependencies are the most important, yet hardest relationships to discover completely.** You will never discover 100% of them. Use your time wisely and discover the dependencies as early in the process as possible. You will need more time later to discover missing dependencies. It is not against best practices to manually create hard-to-discover, yet known dependencies to facilitate proper service modeling. Refer to the CMS Best Practices library for advanced dependency discovery techniques.
- **Freeze Scope to accelerate progress.** Scope Creep can be devastating to a small project where every resource unit counts. Once a detail is validated with a customer, freeze that detail until delivered unless it is clear that it must be adjusted for technical reasons. In the later stages of model development, make it clear to the consumer that only big problems warrant changes in requirements. At some point, you must freeze the model entirely until it is delivered, and accumulate any later changes for a later version. It is almost always the best practice to deliver the originally requested model on time, rather than deliver a greatly modified model late.

- **Small, more frequent iterations get the job done sooner.** Make the consumer understand that there may be multiple validation meetings. Do not be trapped by a **one-chance** situation where it is politically painful or costly to meet with the user. There must be an initial, rough-draft validation; a refined, heading-in-the-right-direction validation; and a final-inspection validation. Do not compress all of these into a single meeting, if possible.

Value Realization may be Protracted

It may take a while from the time the configuration management system (CMS) is purchased and built to return on investment (ROI) realized. Sponsorship is key to successfully driving the IT organization to provide the correct data, discovery credentials, and so on. The service modeling quality will be directly proportional to this.

Expect Ambiguity

Designing and developing service models is closely tied to the development and design of the service itself. This means, ideally, the same people are involved in creating the service model as those who developed the service. The Configuration Manager (or designee) is responsible for interacting with people in IT to identify and validate that the service model is developed correctly. However, these people are not always available and their knowledge may be limited. Therefore, developing service models can be as much art as skill. In any case, it takes some time to learn. Many service modeling practices are not intuitive at first.

Service modeling is a balance between prescriptive best practices and customization to accommodate unique characteristics of a specific IT environment. Service modeling is neither completely generic nor completely tailor-made.

Expect Discovery Problems

Service models ultimately depend on the available CIs and relationships in the CMS, which in turn depends on the discovery technology. Even the best modeling tools and techniques are no match for missing or inaccurate data. Some technology is different and challenging to discover and reconcile—for example, the many types of clustering and virtualization technology. Depending on the specific version and vendor, it may be impossible to collect information on the present locations of a collection of virtual machines without specific code in a specific discovery job. This means additional code must be developed to complete the discovery. The situation arises where the service modeling project is waiting for the answer to an enhancement request to the vendor, which could take weeks, months—or may never happen. To keep the service modeling project on track, work closely with the person responsible for discovery.

Part III: Service Models in UCMDB

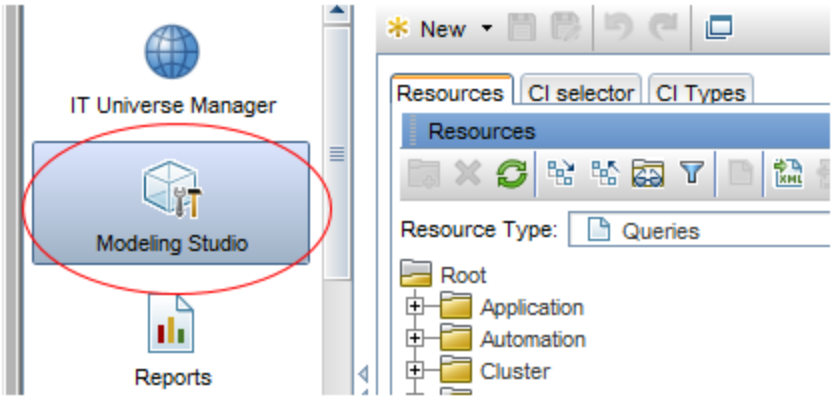
Chapter 5: Developing Service Models in UCMDB

HP Universal CMDB (UCMDB) offers several unique features which can help your organization build accurate and scalable service models. This chapter describes features of the View Manager and the Modeling Studio in HP UCMDB 10.x. Earlier chapters refer to this section for details on creating various types of service models. Advantage, Inc.'s Sharepoint service is used as the example.

This chapter includes:

Modeling Studio Overview

HP Universal CMDB's Modeling Studio is a suite of tools used to develop and use service models.



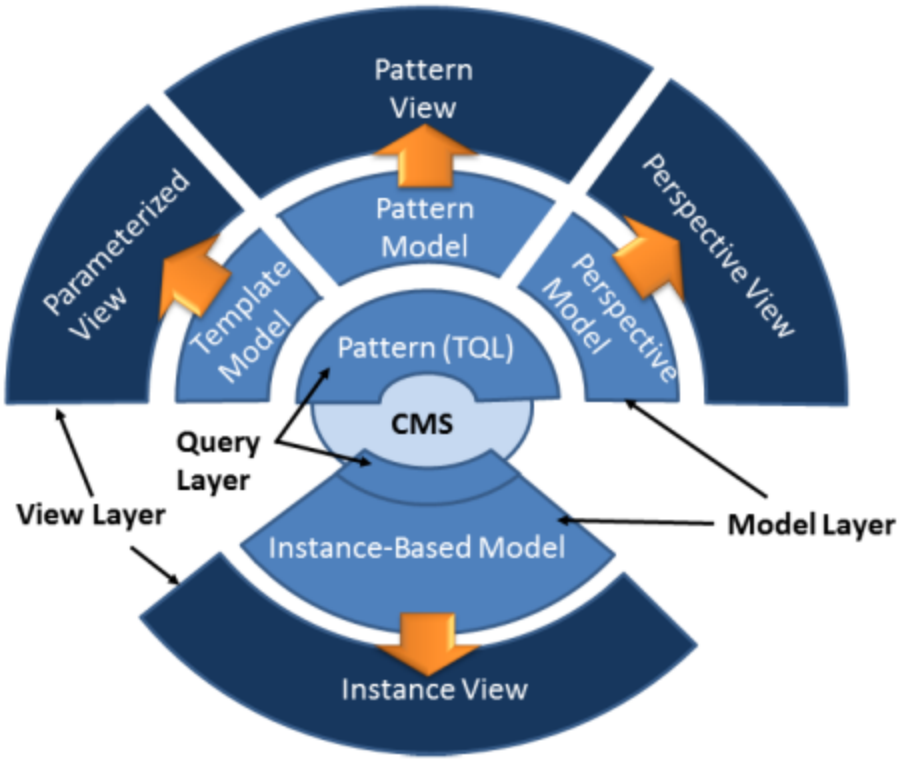
The Modeling Studio's main value comes from its ease of use, rapid time to value, and modular approach. Newer users can quickly browse the CMDB to locate specific resources with a simplified but powerful query engine which can save and reuse multi-step searches.

Newer users also appreciate an intuitive drag-and-drop interface that enables users to build service assets in blocks which can be combined to create models of complex business services.

Subject matter experts who own models find the Modeling Studio ideal for creating and maintaining their models, as well as quickly providing stakeholder-specific views using perspectives.

Finally, the Modeling Studio has the ability to define service ownership and dependencies that are crucial for accurate accountability and impact analysis.

The UCMDB product documentation explains how to use the parts of the Modeling Studio, but does not show how the Modeling Studio as a whole is used in the overall context of service modeling. The following diagram shows all of the types of models that can be created:



The Modeling Studio can be thought of as a set of layers that work together to provide both simple and complex CMS data sets.

Queries (TQLs) retrieve a precise set of CIs and relationships from the CMS. Models contain queries to filter and aggregate sets of data. Views present the data for direct human consumption. Instance-based Models also use a query, but the query is only a place to store a static list of instances.

Choosing the Right Model Type

The following table shows suggested model types based on the type of consumer:

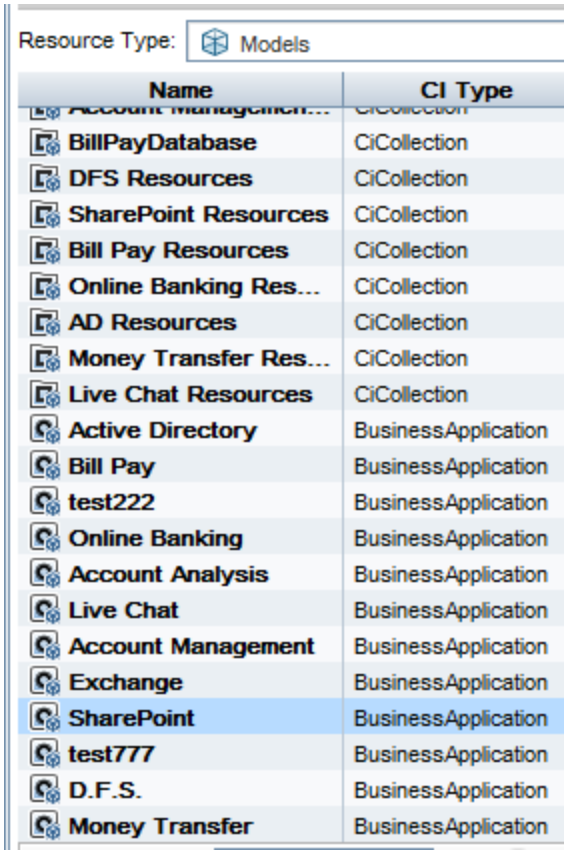
I want to ...	Consuming frequency	Use this type of service model	Why	Typical Consumer
Just see a list of specific CIs	One time	Instance-based model	Least cost, least resource usage, least time to deliver one-time result	Customer Support Business User
Model a business service	Ad hoc	Pattern-based model	Automates business service CI handling	All major IT functions
See attributes of a specific CI	Ad hoc	UCMDB Browser (no model)	Browser specifically suited for ad hoc content and periodicity	IT and Applications Administrator
Look at a different set of CIs based on information I supply	Ad hoc	Template-based view	Parameters allow users to supply controlled information to affect query results	IT and Applications Administrator
See a high-level view of my application	Periodically	Pattern-based model	Allows immediate development and visualization of the business service layers	Business Application Owner
See all the databases in my applications	Periodically	Perspectives and Perspective-based views	Perspectives efficiently filter results, and same query can be reused for many users	Applications and Operations Support
See application parts in different ways	Ad hoc	Perspective-based view	Perspectives allow the most reusable consumption of the base application model	Operations and IT Administrator
Consume a list of CIs in my application	Periodically	Pattern-based report (no model, reuses application query)	Simple, non-technical way to consume service model data. reuses model query to produce the report.	Business User

Example End State

When all the service models are created, the models will collectively represent the configuration of the entire service catalog.

All the business services can be understood in relation to each other in terms of how the services form the business itself. Since all the services are also understood vertically, in terms of infrastructure, another level of ITIL maturity is reached. The goal is no longer merely to control changes and manage configurations, but to improve business agility and lower total cost of service ownership. With everything in one place—reporting, costing, planning—all become more efficient because the configuration management system provides a common, granular view of business services.

In UCMDB, this list of models can be seen via the Models' **Resource Type** list:



This list shows the top layer of service models representing the business and infrastructure services of Advantage, Inc.

Modeling Studio

This section includes:

Overview

Modeling Studio will find most of its strength as a workspace for service modeling in the following two conditions:

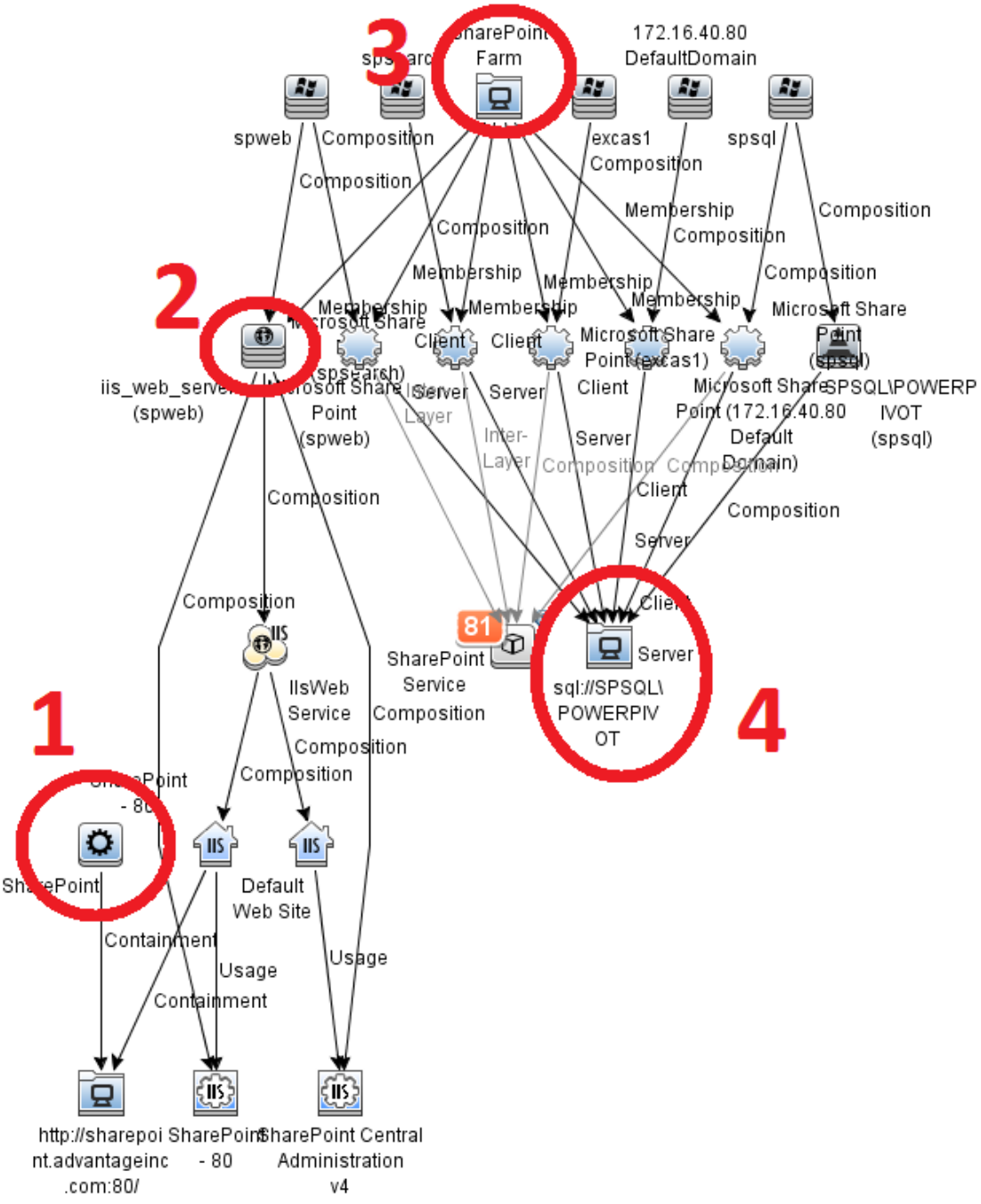
- When the service modeling activity is distributed in nature and various SMEs will own the models which they create; that is, individuals or groups are tasked with identifying the structure of a service model and maintaining its accuracy through the Modeling Studio
- When the CMS contains all the required CIs and relationships and missing CIs can be discovered or added manually

The drag-and-drop nature of the Modeling Studio makes it easy for organizations to distribute the effort required to create and maintain service models. Individuals or groups responsible for a model can find data to include in service models from a variety of sources within a single interface by browsing contents of utility views, invoking simple or complex searches directly against the UCMDB, or through the use of the **Reveal** function against specific instances of a CI in the model.

"Get Related" CIs Tool

Using the **Get Related** CIs tool is an easy way to quickly and on demand surf the HP Universal CMDB (UCMDB) to find what CIs and relationships are close to other CIs.

For example, the Sharepoint business application is partially revealed in the following diagram using the **Get Related** CIs tool:



Advantage, Inc's Sharepoint business application

The annotations are:

1. **BusinessApplication** CI
2. Web Tier
3. Application Tier
4. Database Tier

Get Related CIs works both inside and outside the Modeling Studio. **Get Related** CIs can quickly retrieve all the related CIs in an application when you are not sure what or if certain relationships are present.

For example, if the use case required the file systems to be present in the model, but no file systems were found using the **Get Related** CIs, this would indicate a need to go back and perform additional discovery, or implement an additional integration adapter to populate the CMDB with file systems and their proper relationships.

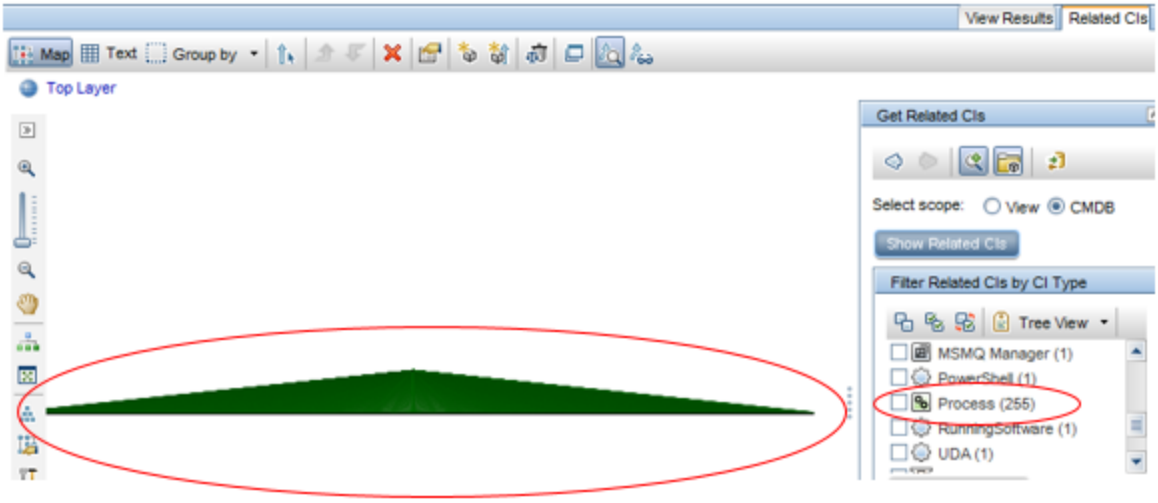
The **Get Related** CIs tools may be used anywhere in the UCMDB user interface where a CI is displayed graphically or tabularly. For example, the IT Universe displays a list of CIs, as well as a topology map. A right-click on a CI in either of these places reveals the **Get Related** menu item. **Get Related** can also be invoked from any CI list, at any layer in a model.

For small applications and infrastructure subsets, **Get Related** is a good tool to use. However, there are limitations with the **Get Related** tool that makes it difficult to use to find the topology of large or complex services. The **Reveal Path** tool is a better for this purpose.

Reveal Path: Increase Modeling Efficiency

To understand the value of **Reveal Path**, consider the **Get Related** tool in the previous section. Most UCMDB users know and use this tool, and it is a common way to quickly look at topology closely related to a CI. However, **Get Related** is not the best tool to find all the topology and relationships of a service, in preparation for modeling.

For example, one limitation of **Get Related** is CIs can only be revealed by CI Type, not by individual CI. If a server contains 100 process CIs, two of which are interesting, **Get Related** quickly becomes cumbersome to use because 98 process CIs and their containment relationships on the screen take up most of the space in the topology pane and render the text too small to read. Colloquially, this is known as a **big ugly pyramid** and is the problem with the **Get Related** tool.



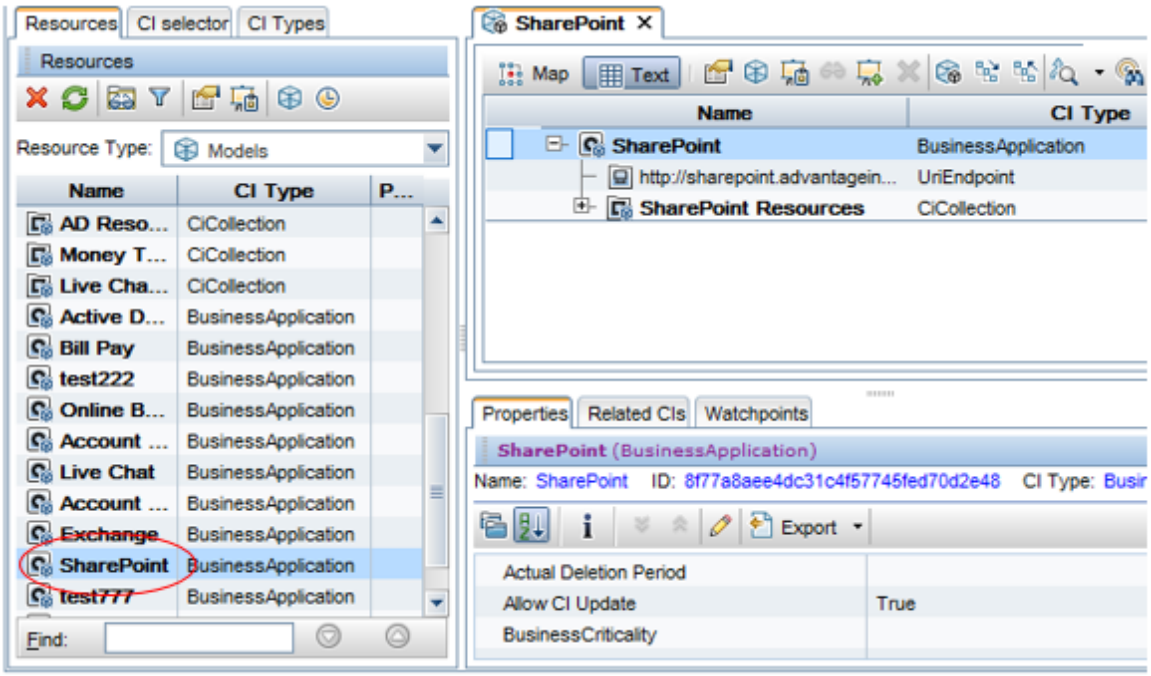
In finished views, models are folded by CI type to hide these types of structures. But in **Get Related**, there is no easy way to reveal only specific processes and hide the rest. **Reveal Path** solves this problem.

Reveal Path finds the related CIs and allows filtering by CI instance, not merely CI type. In this way, a host containing many CIs of the same type can quickly be analyzed to find the one CI needed for a service model. **Reveal Path** manages multiple links from a CI, as well as a compound path (multiple hops) from one CI type to related CI types. The **Reveal Path** user interface is iterative and allows a path similar to a **bread crumb trail** to be maintained and saved to use in modeling.

Saved paths are reused to support quicker browsing and easier access to common queries. The purpose of revealing information is to traverse a path of relationships in order to find the existence of configuration item instances with minimal knowledge of the configuration item relationships.

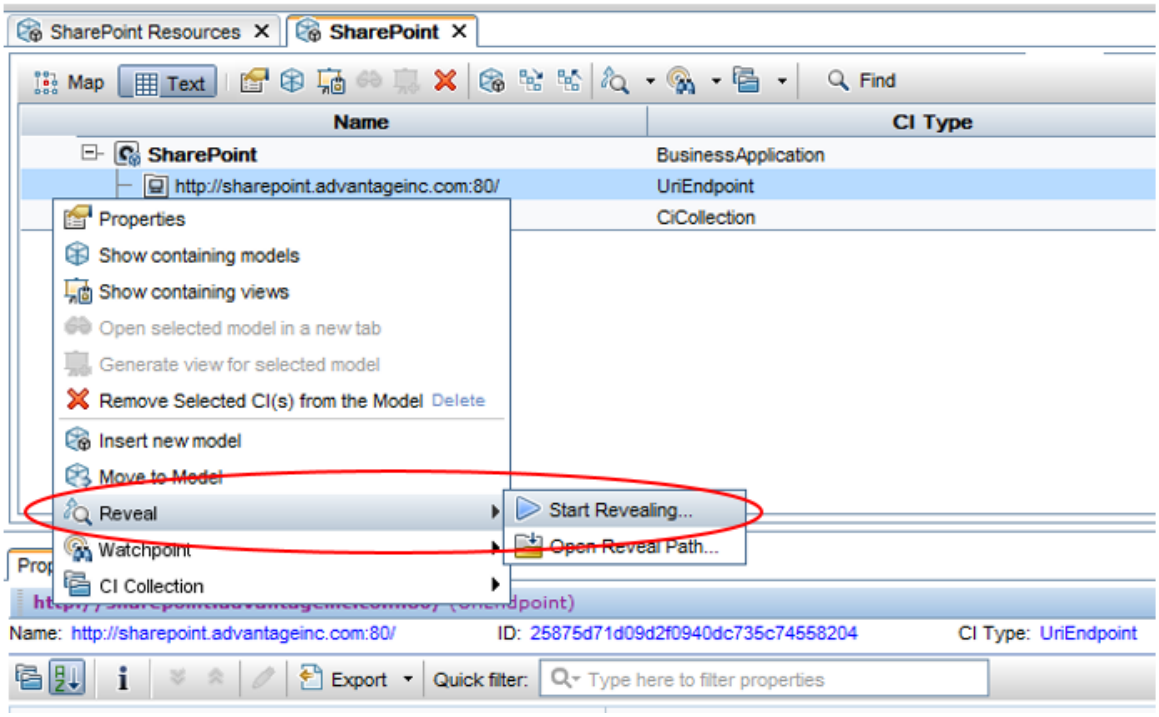
As discussed earlier, the access point of the Sharepoint application will be known. **Reveal Path** can be used to find the rest of the application's infrastructure. Beginning with the entry point, a **UriEndPoint** CI, **Reveal Path** will find the unique Sharepoint infrastructure, including clusters, servers, and network devices. From there, a user could find each server which uses the same unique cluster ID and add each of these servers to the Sharepoint service model. This simple **Reveal Path** can be saved and reused to help quickly find servers which use the unique cluster ID starting with any specific instance of a server.

Begin by creating any type of model CI, such as a **BusinessApplication**, **BusinessService** or **CiCollection** CI, and relating the service's entry point(s) to it via containment or dependency. These will show up in the Models pane of the Modeling Studio.



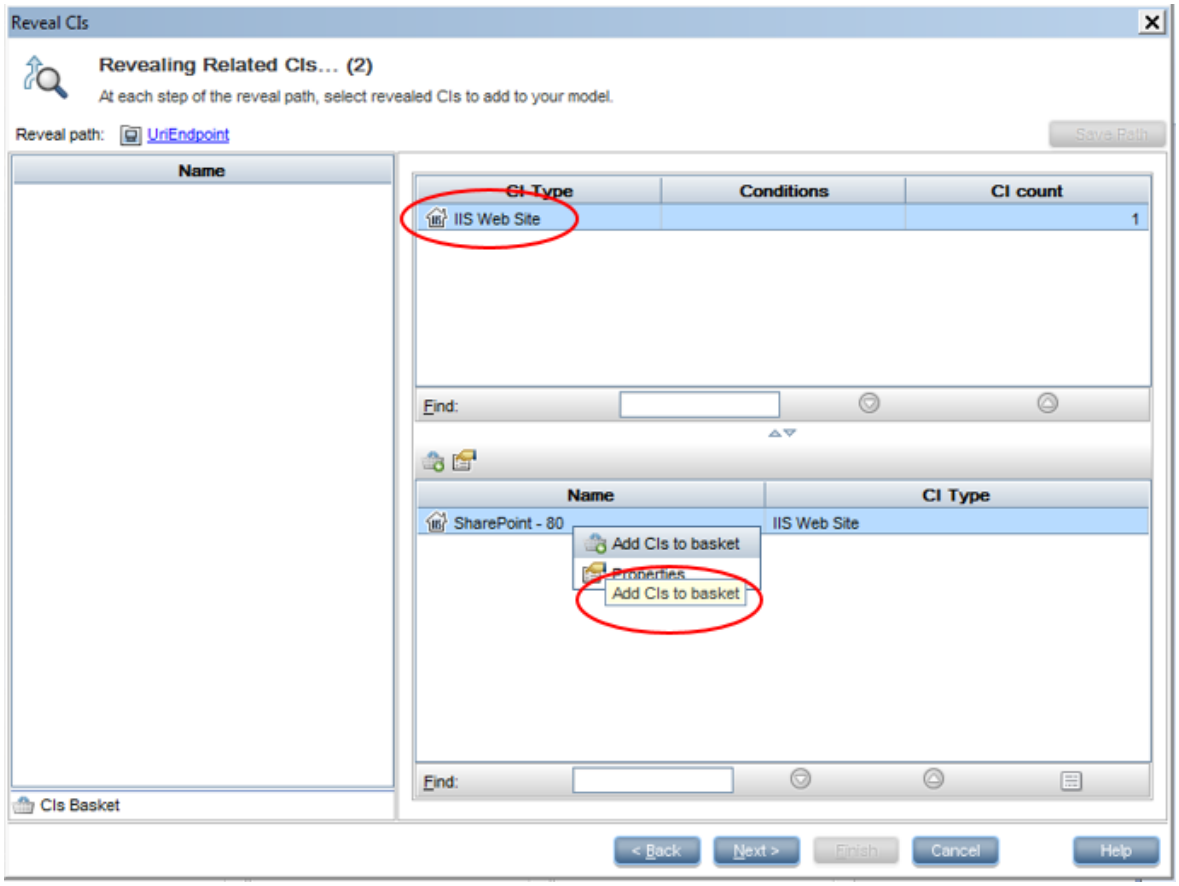
At this step, the model contains only a single CI—the **UriEndpoint** added in the first step.

From a CI instance in a model, right-click **Reveal** > **Start Revealing**. Then drag the application server into the **Reveal Path** window and click **Next**.

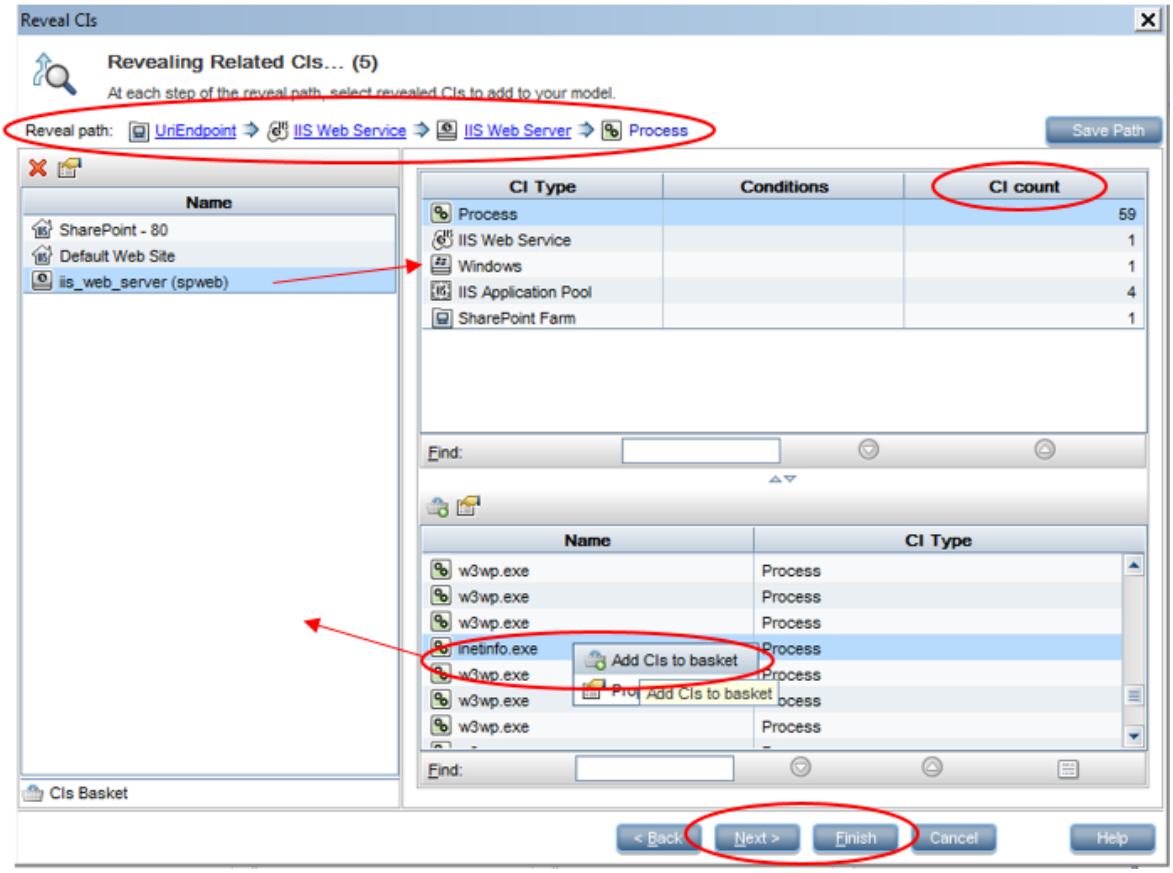


The new list of CI types are revealed to select the desired path. Add specific CIs to the basket from the list of **Reveal CIs** type selected in the upper pane. Click the **Next** button to add the selected CI to the path and place the next set of related CIs in the upper pane. Repeat until the desired path is complete, then click **Finish**, which adds the basket CIs to the model.

The following image shows a path revealed from the starting CI, **UriEndpoint**, to the next related CI, an IIS Web Site.



Reveal Path is used to easily find related CIs. For example, after seeing the **UriEndpoint** related only to a Web Service, the goal is still to find the running software. Several additional steps of **Reveal Path** are necessary to find the **RunningSoftware** CIs. **Process** CIs are shown here related to the **IIS Web Server**.



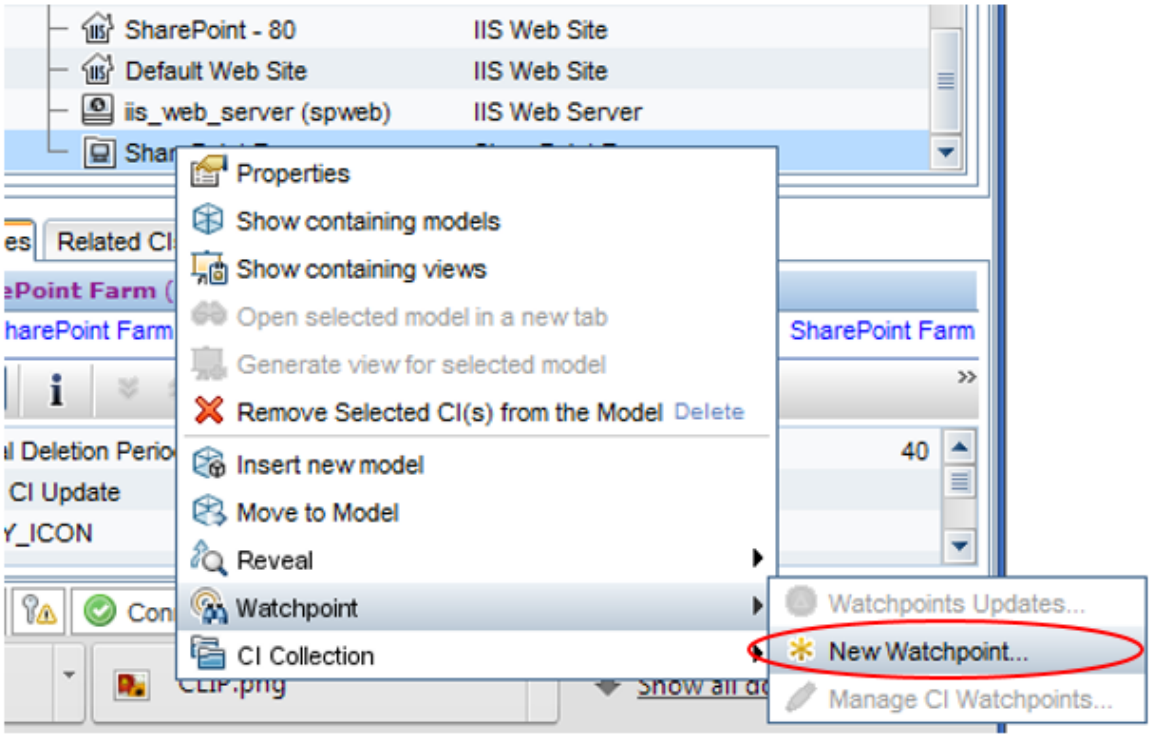
In the above example, note the count on the **Process** CI type (59). With the **Get Related** tool, it would be tedious to find the correct process. With **Reveal Path**, the one interesting CI can be easily identified, and only its relationships revealed.

Dynamic Models

The saved **Reveal Path** could also be used as a Watchpoint in order to dynamically update a model. By applying a Watchpoint on a specific node, using the simple **Reveal Path** saved from above that finds all servers using the same unique cluster ID as that server, any new servers which were discovered as using the unique cluster ID will automatically be added to the Sharepoint service model once this information enters the UCMDB. This avoids the manual step for the model owner to manually update a service model each time the cluster dynamically increases or decreases in scale due to reasons such as capacity or performance.

Each of the models created and maintained by an individual or group can be embedded in higher level models which own or depend on the **building block** service models. Although service modeling in Modeling Studio has some static qualities, embedded models reflect changes dynamically within higher level models to match the distributed ownership and maintenance of service modeling in Modeling Studio.

Once core service models are built, use perspectives to generate stakeholder-specific views of the model. Perspectives are another way to **build once, reuse many times**. This helps further reduce maintenance of stakeholder-specific views by allowing service model changes to be automatically reflected in views generated with perspectives on the model. Adding a New Watchpoint is depicted below:



Queries (TQL)

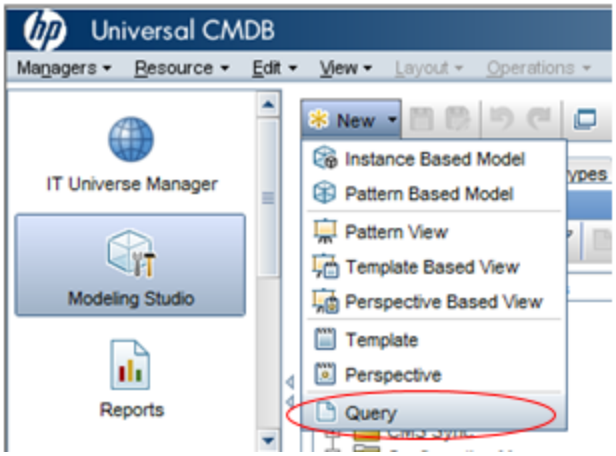
Topology Query Language (TQL) is the visual environment used to create queries in UCMDB. Think of TQL as a visual SQL—able to easily express and allow manipulation of topology and graphing concepts.

Many models begin by creating a query. Queries can be created as standalone, used by something else later, or used in an API with no human consumer.

Most functions in UCMDB that use CIs or relationships use TQLs—enrichments, Discovery jobs, integrations, as well as user-facing entities like views and perspectives being discussed here.

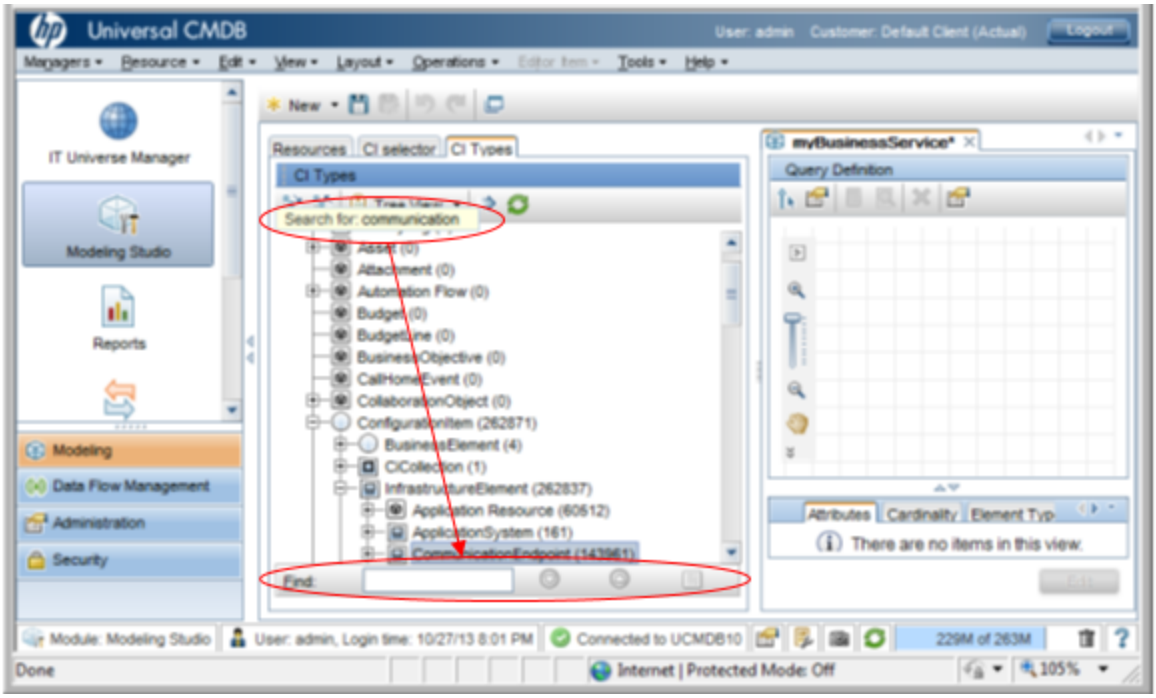
From the Modeling Studio, a new query may be created. Here are the basic steps to create a TQL for a simple service model showing the URI endpoint, running software, the host, a web server, and a database server.

In the Modeling Studio, click **New** > **Query** to start creating a new query.



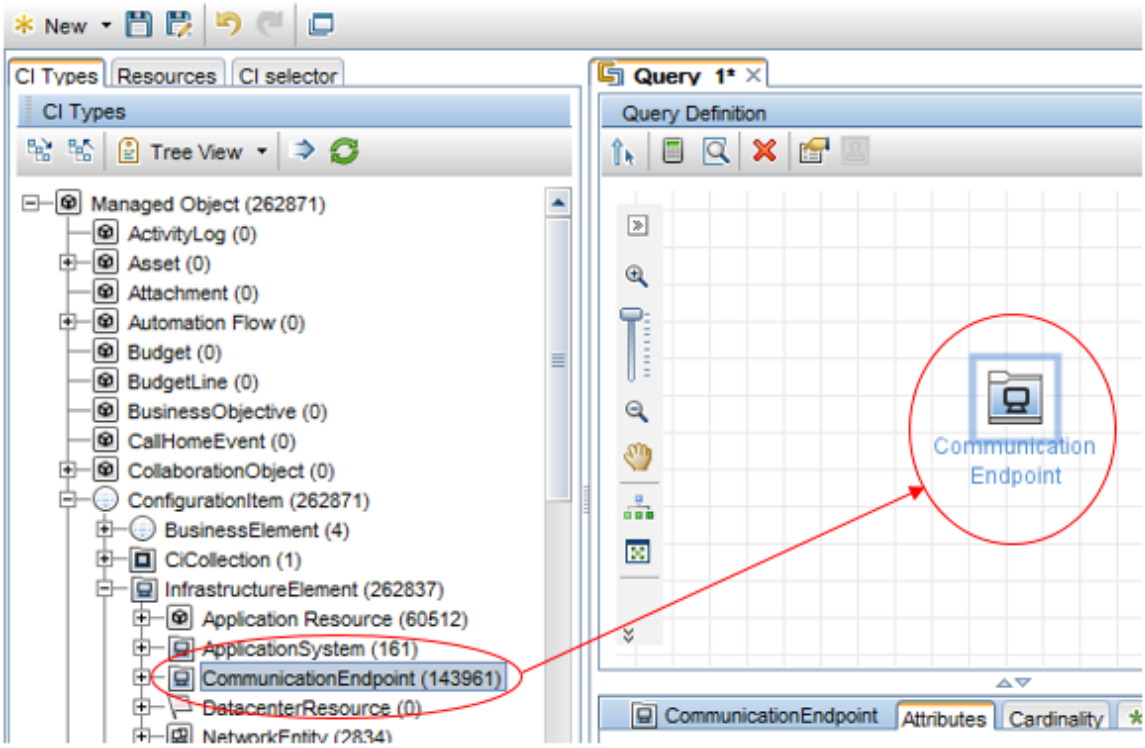
The modeling pane is shown below. Modeling is done in the pane on the right. The class model is in the list to the left. Drag and drop CI types into the modeling pane to put them in the TQL.

You can quickly find CI type names in the class list by simply typing in the start of the CI type name. To find a CI type that begins with **comm**, like **CommunicationEndpoint**, start typing anywhere in the class model, as shown here:

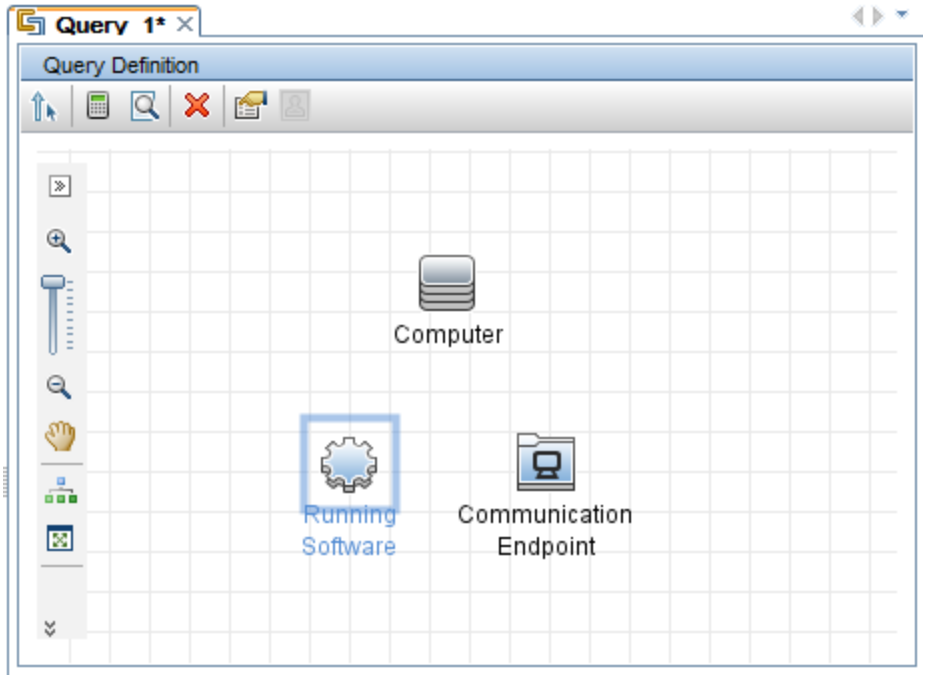


By starting to type **communication** while focused on the class model tree pane, the cursor jumps to the next CI type matching what is typed as shown above.

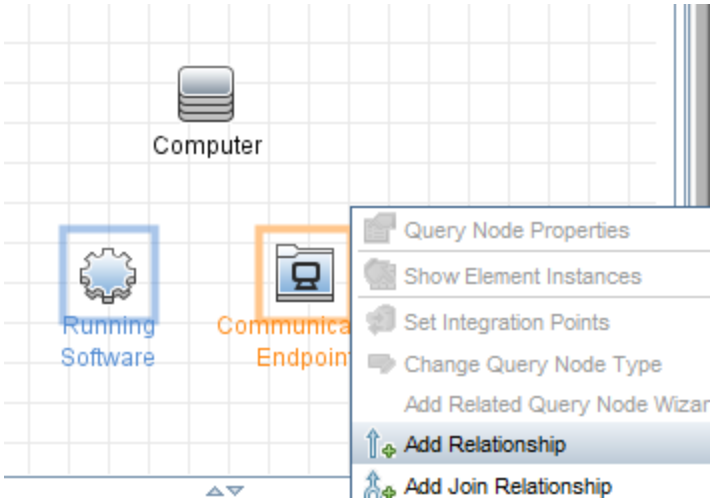
Next drag the **CommunicationEndpoint** into the query definition pane to add it to the query:



Now do the same for two more CI types—**RunningSoftware** and **Computer**. Find the CI type in the CI Types pane and drag it into the query definition pane.Then calculate the counts. You get something similar to the following:

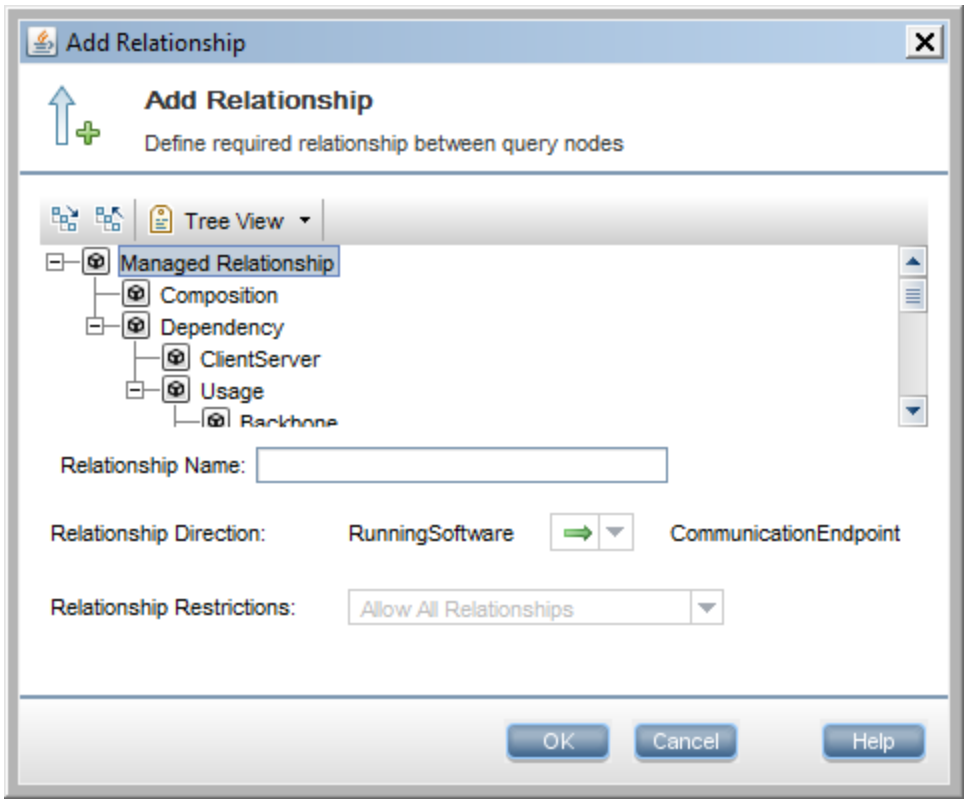


To relate CI types, select exactly two of the CI types. Right-click, and then select **Add Relationship**.

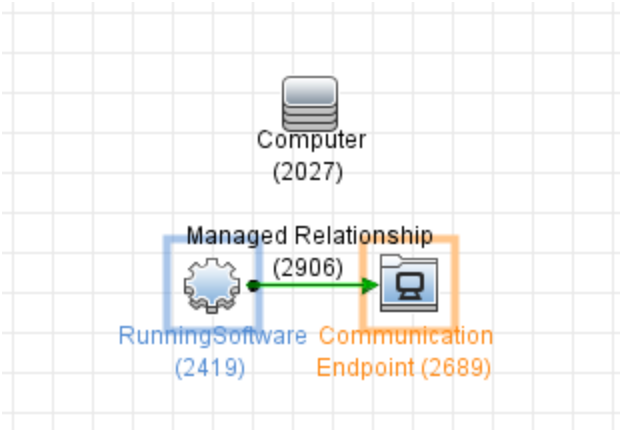


Note the two selected CI types, **RunningSoftware** and **CommunicationEndpoint**, and the right-click menu showing the **Add Relationship** item selected. If your screen looks similar to the screen capture above, you are in the right place.

Next select **Managed Relationship** to add the relationship to the query as shown here:



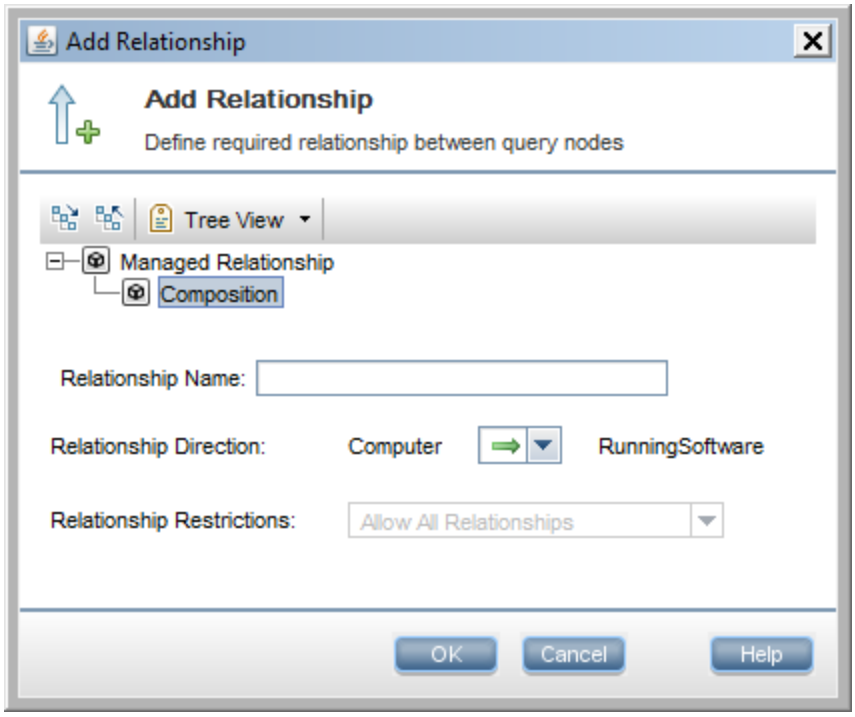
Do not change anything on the Add Relationship screen for now. Just click **OK**.



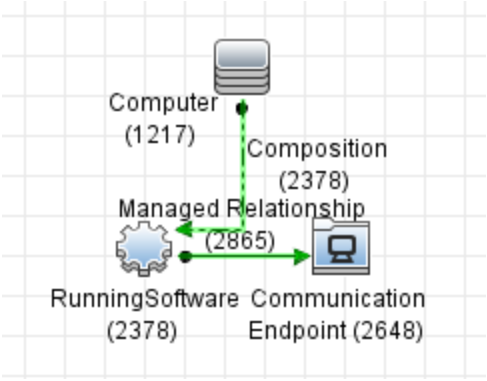
The relationship is added.

Click the calculator button (the  above the query builder area) and verify the counts are non-zero. If so, you are doing it correctly.

Now relate the **RunningSoftware** to the **Computer** in the same way. If **Computer** is selected first, the Add Relationship dialog box should show up offering a Composition relationship, like this:

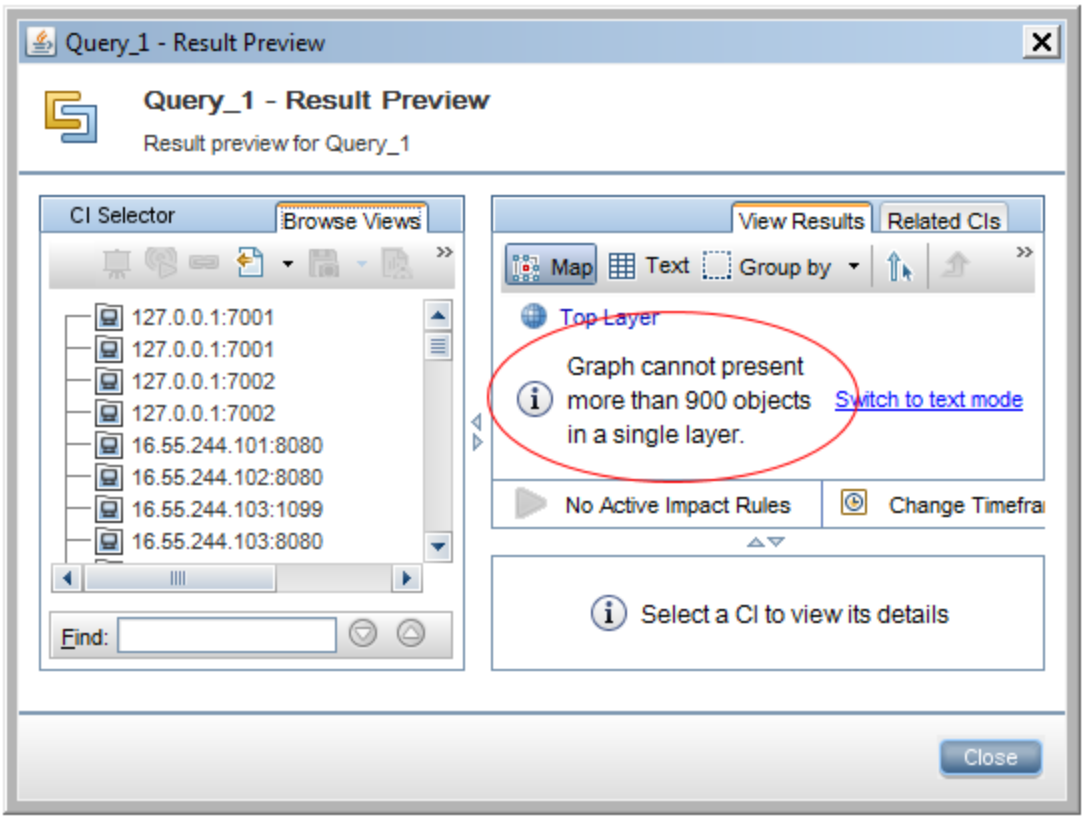


If it looks like the previous Add Relationship dialog box, showing the dependency tree, you have the CIs selected backwards. To correct this, either change the direction of the arrow (between **Computer** and **RunningSoftware** as shown here), or cancel and re-select the CIs in the reverse order. When you are finished, your query should look similar to the following:



This query shows CI Types, relationships, and non-zero counts.

If you previewed the query now, you may get an error message such as:



It is possible but not recommended to increase this 900-count limit to everything in the CMS that looks like **Computer** > **RunningSoftware** > **CommunicationEndpoint**. No applications or services have been distinguished or even identified. This view would be looking literally at every instance in the CMS of those related CIs.

The solution is to add specificity to the query. This can be done by:

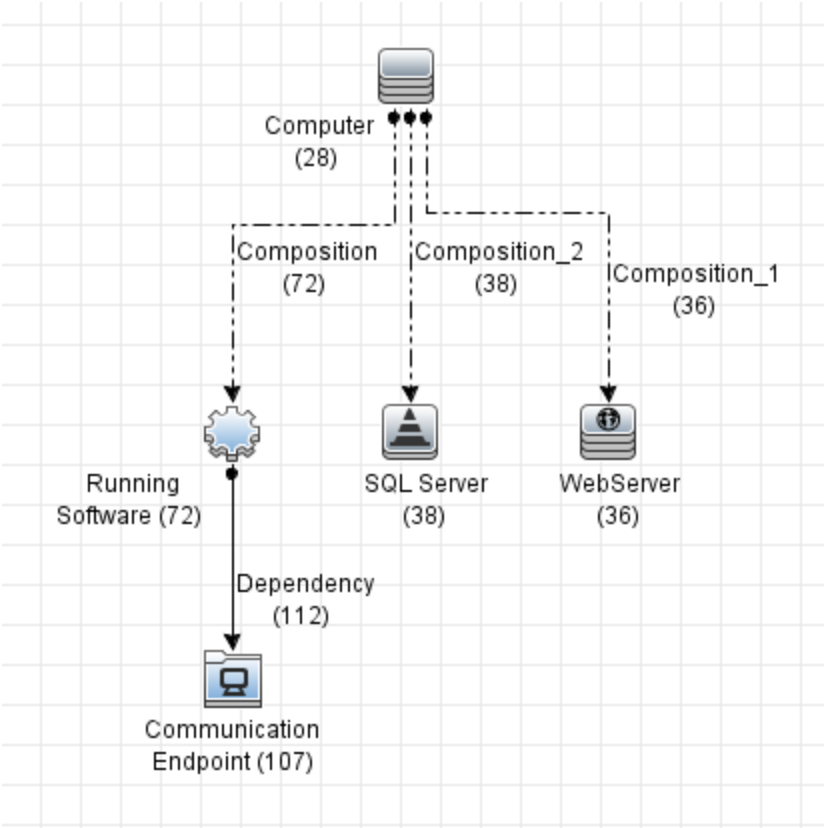
- **Do this anyway.** Add the rest of the CI types and relationships (natural specificity).
- **Top-Down method.** Use only a specific CI known to belong to a specific service, such as the communication endpoint.
- **Bottom-Up method.** Find something else in the infrastructure architecturally unique to the service.

We will follow the first two bullets, but not the third, according to the best practices in this document.

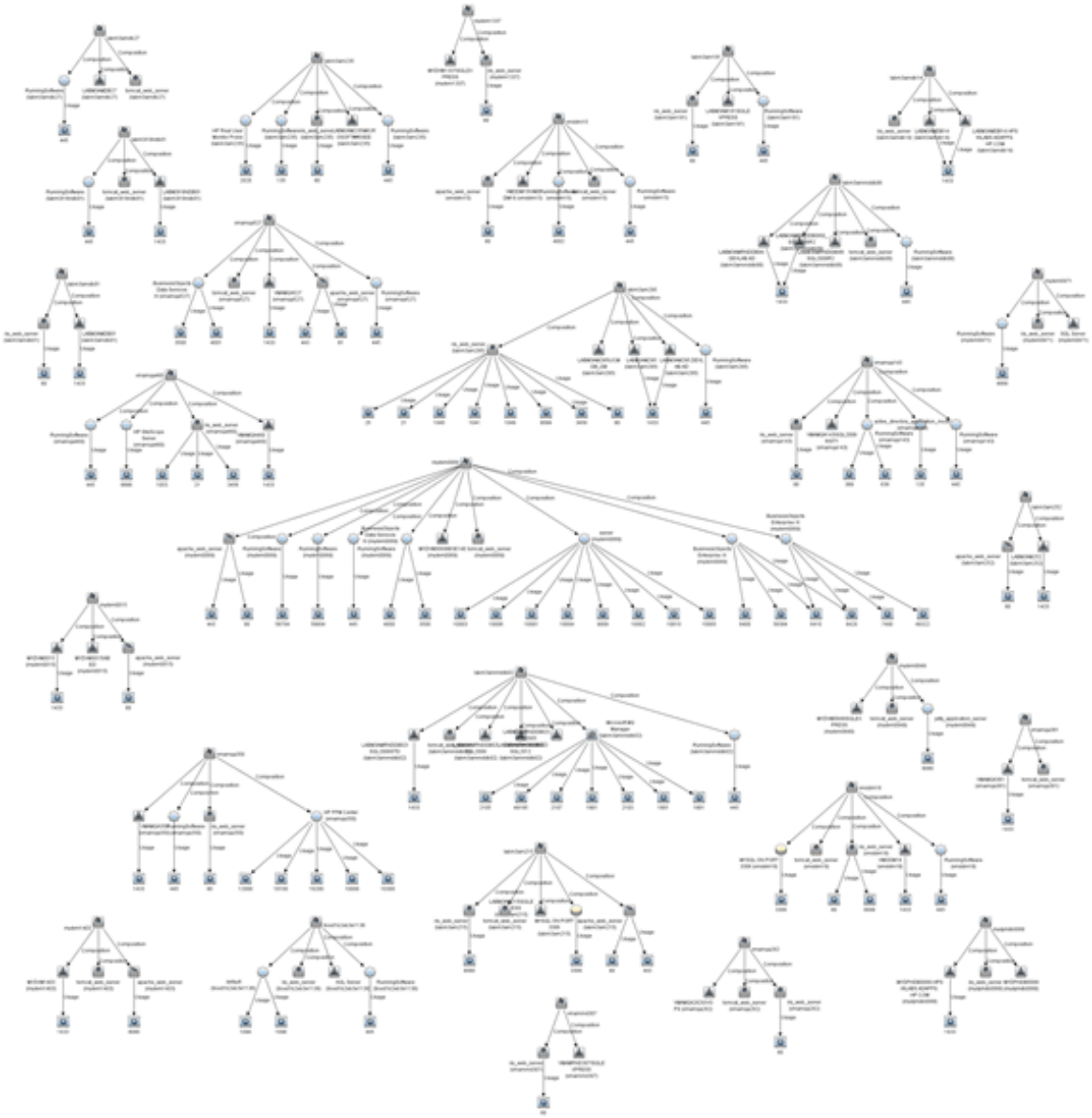
For example, the use case calls for application, database, and network components. The process is the same as above:

1. Find the CI Type name by typing over the class model.
2. Drag and drop a CI type into the query pane.
3. Calculate the count.
4. Select two CI types, right-click, add a relationship.
5. Calculate the count.
   - If the result is zeros, back up one step and re-evaluate.
   - If the result is a large count, continue the process.
   - If the result is a small count, preview the query using the **Search**  button and evaluate the results.
6. Repeat until the query results are consumable.

Here is a sample intermediate result:



A preview of the query at this point reveals something similar to the following:



Intermediate query results show proto-models matching the generic query.

When building service models, it is not uncommon to build these **utility** views to expose discovered data to model owners and SMEs in order to validate discovery results, as well as facilitate service resource identification.

It is not important at this point to see all of the labels and names of the CI types, although zoom works here. The query is correctly returning subsets of the topology matching the query.

Each of these groups could be part of an application or business service. The **bottom-up** technique could be used at this point to arbitrarily add some uniqueness to the CI types by adding a TQL node condition, but there is a better way. The application or business service can be associated with all the CIs in a group by creating a model. Alternatively, create an application or **BusinessService** CI

manually and associate it to any CI in the group. This will anchor the CIs and create a service model which will allow the **Reveal Path** tool to be used.

More modeling techniques using queries are shown in the following sections of this guide.

Some infrastructure elements are more challenging to model due to complexities both in the discovery as well as the representation of the model—for example:

- Storage Area Network (SAN) infrastructure
- Virtualization
- Clustering technology

These generally have more complex chains of CIs and relationships. Therefore, utility views are valuable for developers constructing service models.

Enabling view builders to add nodes to a view's TQL based on discovered data, as opposed to building view TQLs based on expected data, helps to quickly point out deficiencies in discovery as well as rapidly construct views without prior knowledge of the Configuration Item Type schema.

Best Practice: Using the TQL Node Wizard:

1. Add a few nodes to a view to begin.
2. Right-click on a node on the view's TQL to select and add related CIs by type that have discovered instances in the UCMDB.
3. Repeat from Step 2 until the ending node is found.

What was avoided here is the necessity to guess which node should be added to the TQL in order to get to the next step in the relationship path because the TQL Node Wizard reveals only CI types related to the selected node that have discovered instances. Additionally, because the TQL Node Wizard reveals only CI types that have a relationship to the selected node, it is not necessary for the user building the view to repeatedly switch between the View Manager and the CIT Manager to investigate what possible CI types can be added and related to any particular node in the view. Finally, by using the TQL Node Wizard, users building views can quickly find dead-ends to the discovered data to alert the discovery side of potential problems with the discovery scope or access.

Without prior knowledge of the CI types and relationships involved in the database to storage array path, one would need to build a view's TQL node-by-node.

For more information on TQL performance and tuning, refer to the best practices document *Maintaining a High Performing CMDB* in the HP Best Practices Library.

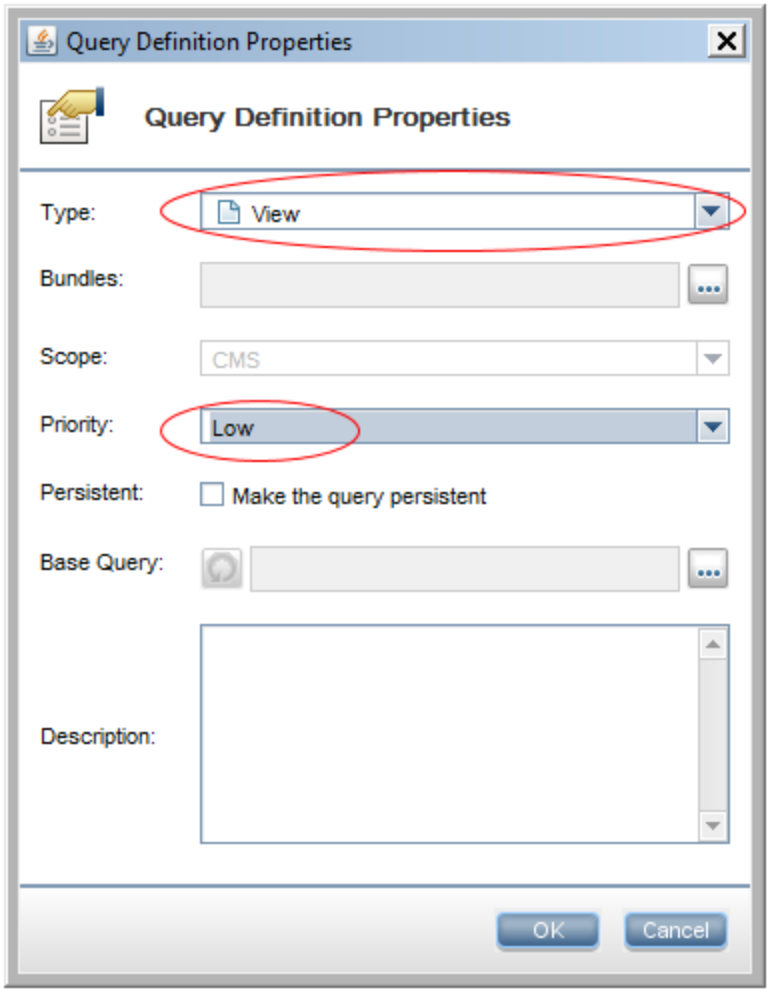
Pattern-based Views

In the context of service modeling through the Modeling Studio, a view of a model has a special meaning. In this case, a view is a combination of a model and a perspective. People use views to consume topology. Applications usually consume TQL/queries directly.

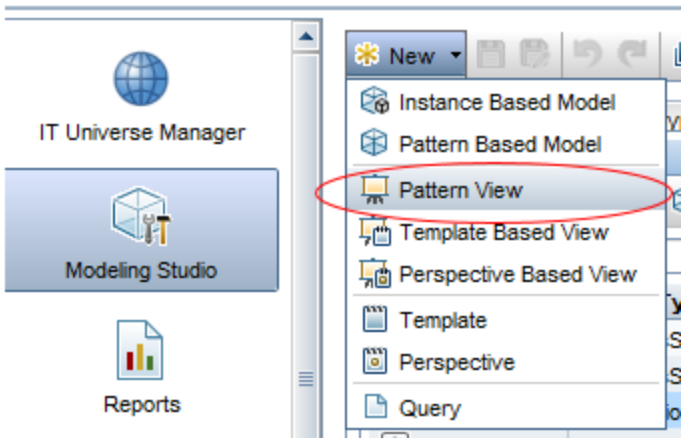
A view is a good choice when people are directly consuming the content. Views allow TQL/query results to be displayed in a drill-down fashion—collapsing and expanding topology as needed. Views also allow folding and grouping based on CI type or attribute values.

As a best practice, building the TQL inside the view builder saves time and avoids accidentally creating the wrong TQL type.

To conserve resources, ensure the priority is low. Once you have a view TQL, you can create a view:

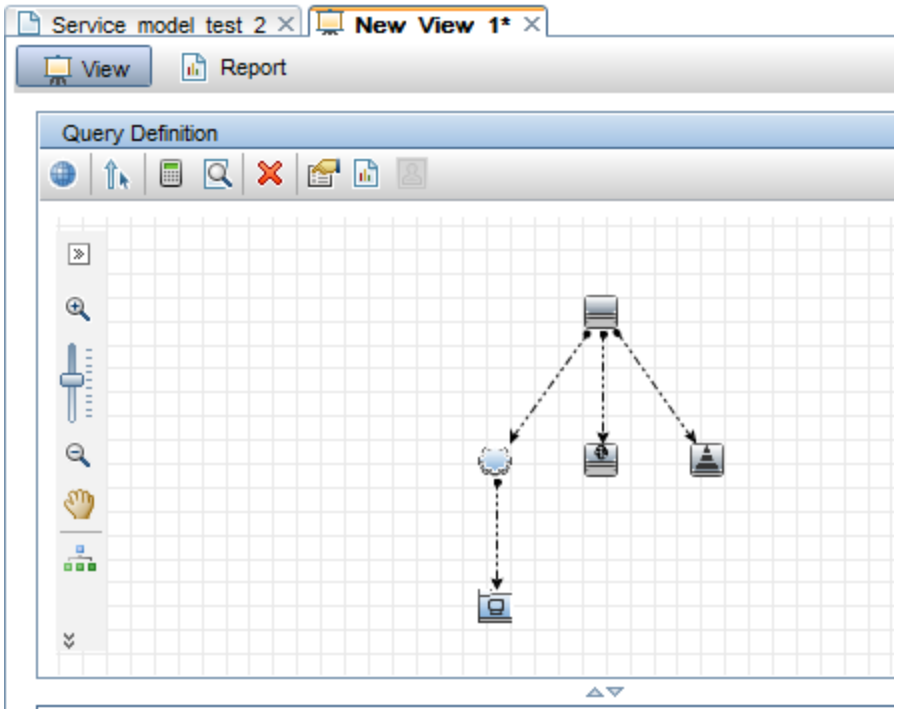


The query is retrieved and displayed in the View Builder.

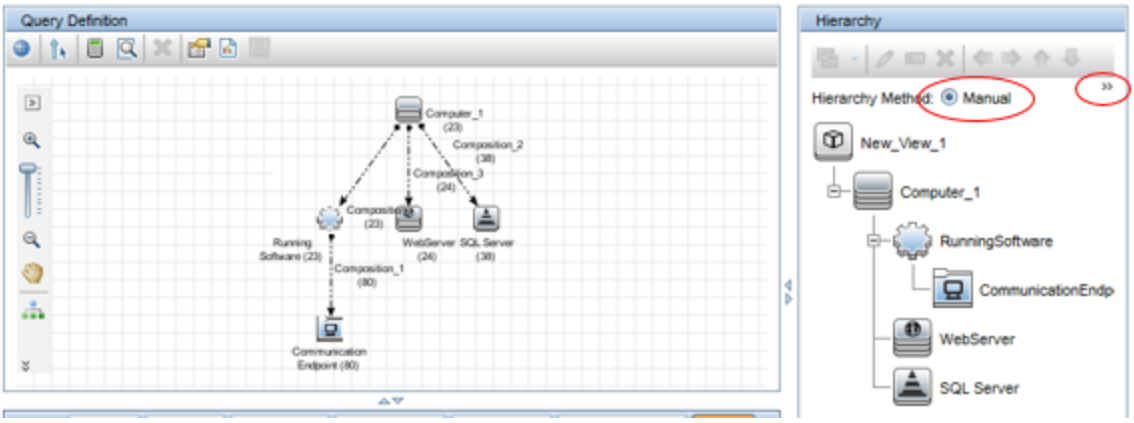


A view is created by modifying view-level characteristics of each node in the query.

Be sure to save the base query as type **View**.



A view is a convenient way to store how to present the topology. Grouping by CI type and Hierarchical layout is the default. Try to stay with a common presentation layout, but some exceptions are normal. Sometimes a circular layout or symmetrical layout is better at displaying groups and clusters of CIs.



Layering allows the presented view to allow drop-down, drill-down navigation. The layering layout is controlled in the right pane. Automatically arrange the Hierarchy or drag the CIs into the Hierarchy where desired.

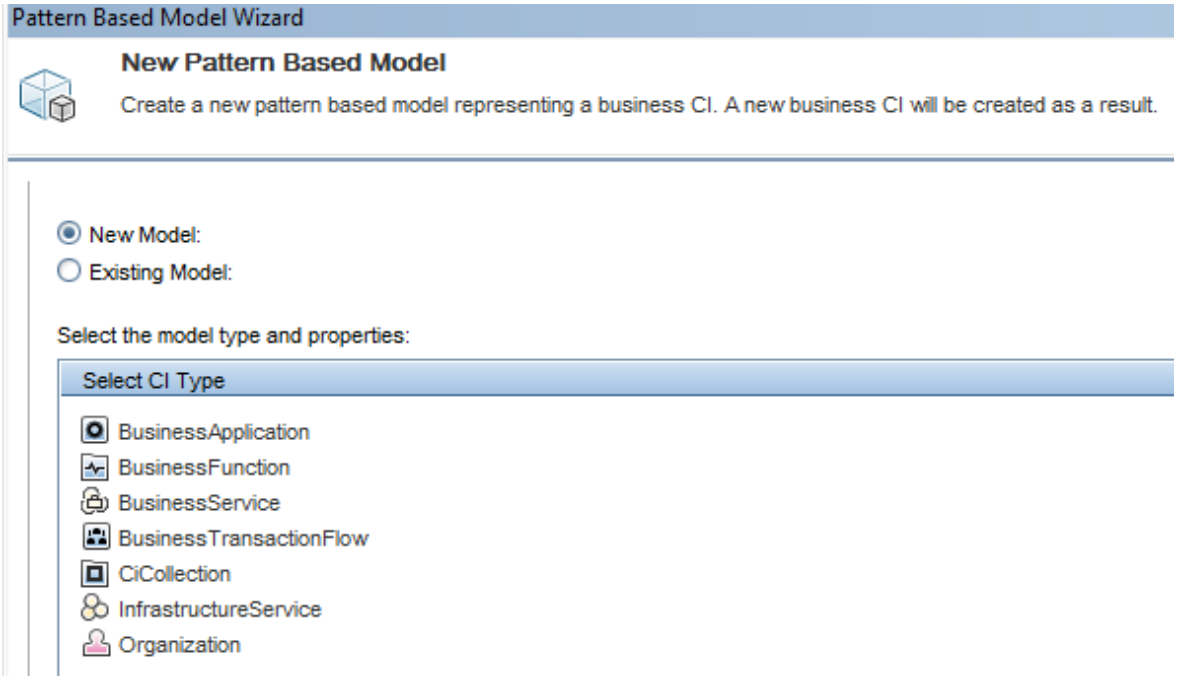
The view may now be tested, consumed, and so on. For more information, refer to the UCMDB administration reference documentation in the HP Software Product Manuals Web site.

Modeling Studio “Models”

Before proceeding, it is important to disambiguate the term **model**. The sections on models here refer to the entities in the Modeling Studio of UCMDB. Where this usage conflicts or is ambiguous with the meaning of service model (as described in the first part of this document), it will be mentioned specifically as **service models**.

Modeling Studio models are reusable collections of CIs which describe a logical entity and the resources it owns or depends on. Typically, an individual or group (of SMEs) would be responsible for creating and maintaining the model of this logical entity.

Models automatically create and relate a business CI to the rest of the CIs in the model:



This is a good way to begin creating service models.

Model Contents

A model's contents can be populated from a list, by a query, and from many places in the UCMDB user interface via the right-click menu **Add CI to Model** item.

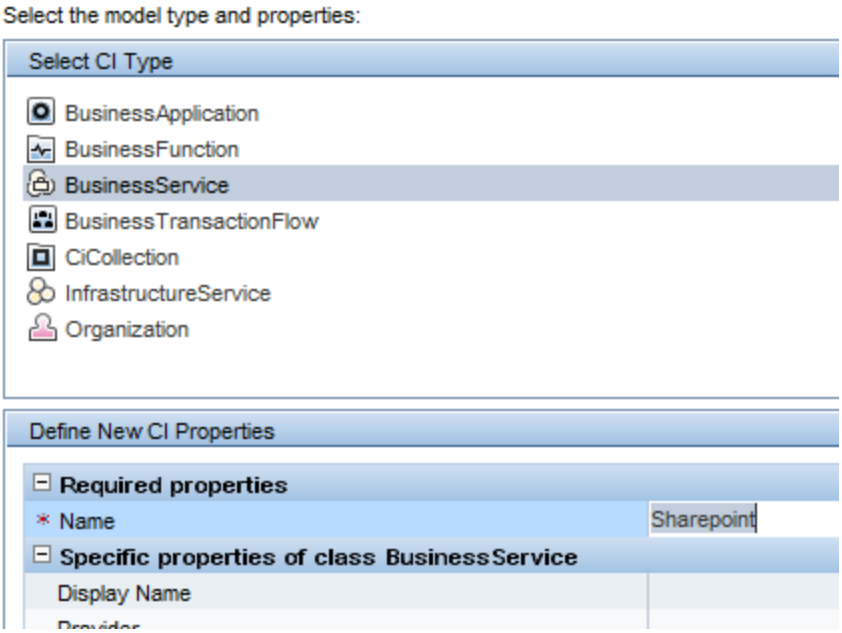
To simplify this concept further, think of the model as a box of pebbles where the box represents the logical entity and the pebbles inside this box are the configuration items which it owns or depends on. The owner of this box allows anyone to look at the contents of the box and the owner is the only one able to add or remove pebbles from the box. One important point to add is that a model can contain other models. In this case, the owner of any box can add the box of another owner. However, the owner adding another box into their box does not have authority to change the contents of the embedded box—only the ability to add or remove an entire box.

A model can also be consumed from a Perspective.

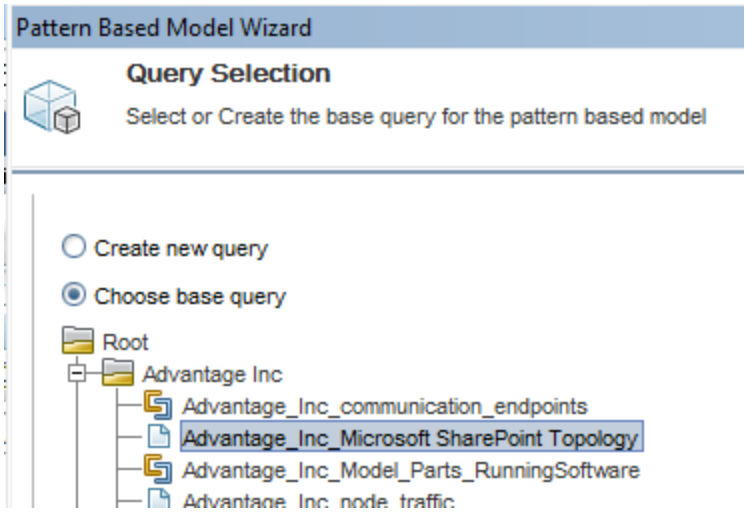
Pattern-based Models

Pattern-based models are useful for creating service models. The top layers of the service are automatically created and related to all the CIs in the model. Both **BusinessService** and **BusinessApplication** CIs are good starting points for most service models. Continuing with the example above, a pattern-based model will be created using a **BusinessService** CI.

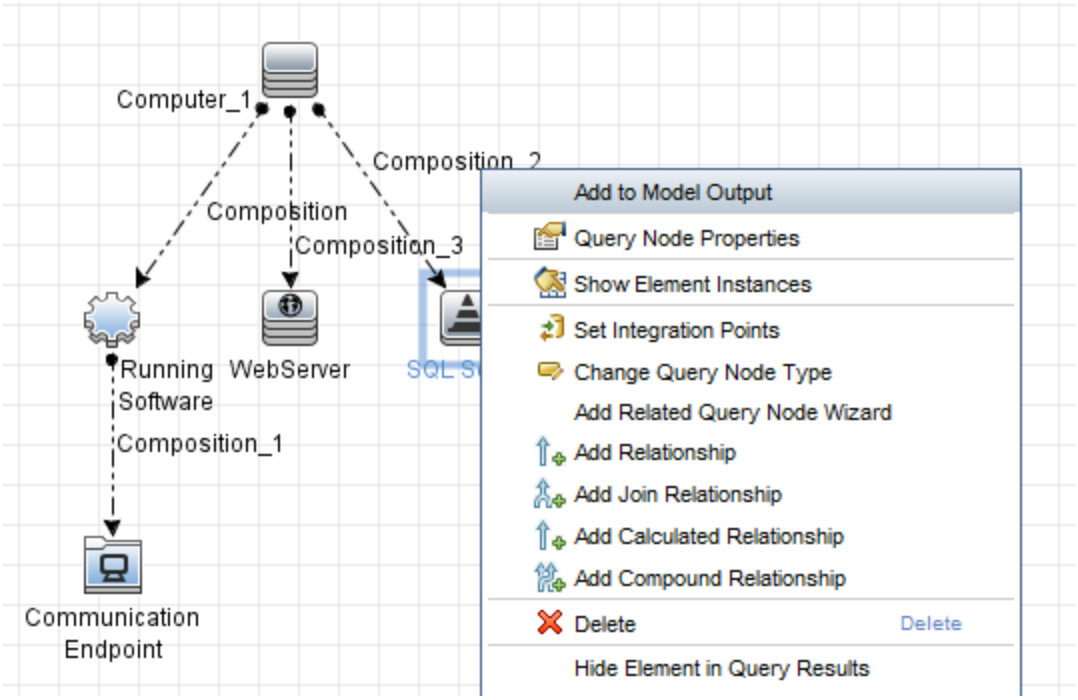
Create the **BusinessService** CI to associate with the model.



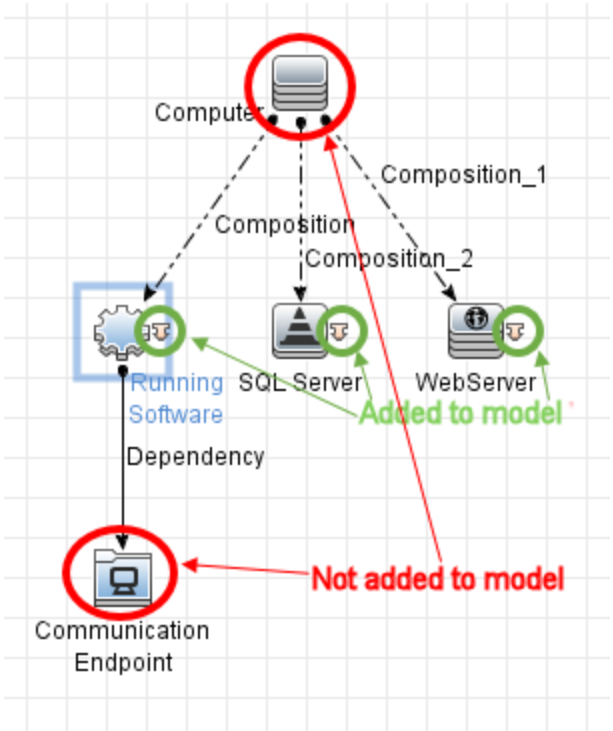
Select the query, or create it if you have not created it yet.



To add CIs in the query to the model output, right-click the desired node and select **Add to Model Output**. Repeat and save. CI types that are present only to link other CI types and are not intended for consumption are not be added to the model output.



The **CommunicationEndpoint** and **Computer** CI types are omitted from the model output.



When the model is saved, all CIs added to the model output are related to the **BusinessService** CI created at the beginning of the modeling process (see "Step 2: Model Top Layers of the Service" on page 51).

Static Models

Models may exist as a manually-selected fixed list of CIs. A model built this way has some advantages to pattern-based models:

- Quick to create—from start to finish in minutes or hours
- Easy to create—does not require advanced UCMDB TQL knowledge
- The service itself is also fixed and will not change
- Static model may be used even if there are no relationships present

Easy ways to organize CIs into a group that can be used as a service model:

- CI collections
- Instance-based models

CI Collections

CI collections use the **CiCollection** CI type to relate CIs to an arbitrary collection. Create a new **CiCollection** CI and name it something meaningful to the collection, such as the business application to which all the CIs in the collection belong.

There are two relationship types to link CIs to a CI collection:

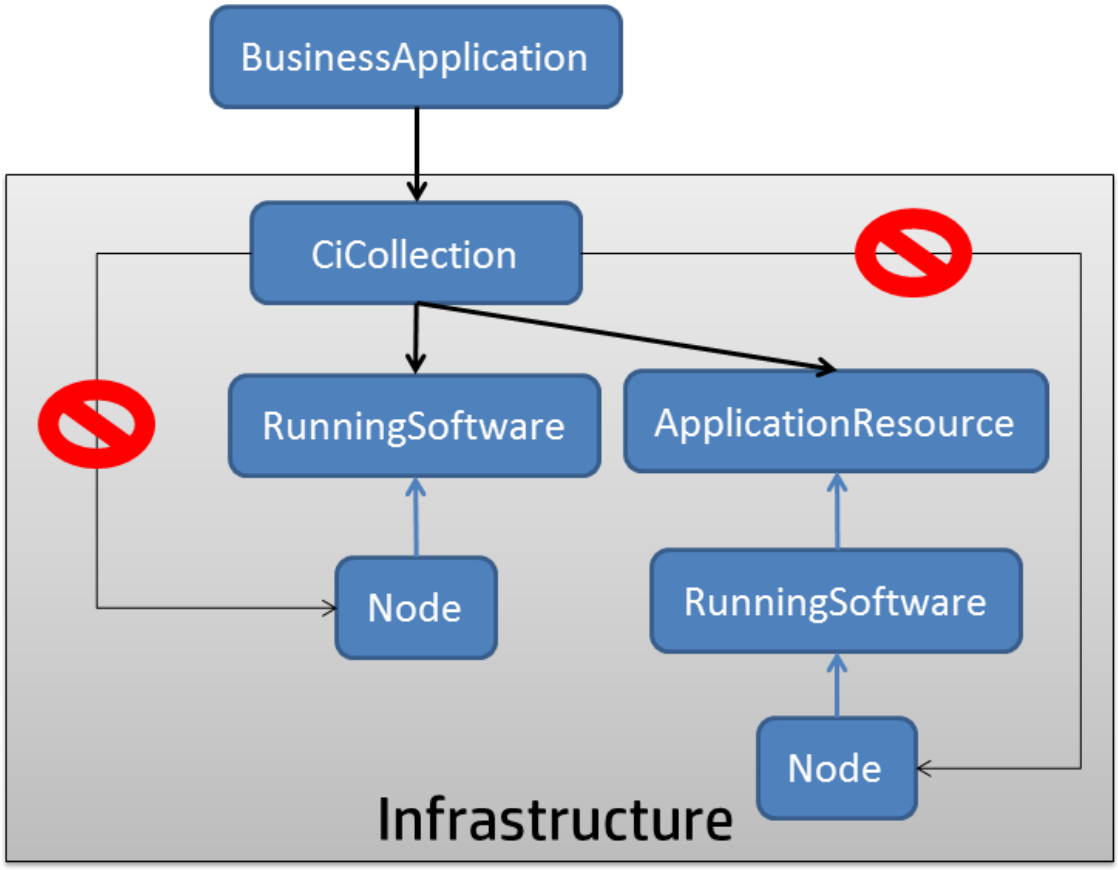
- **Usage**

  Usage means one component uses and therefore depends on another component. This relationship means that a change in a component carries impact consequences for **usage** relationship dependencies, but not direct life cycle impact.

- **Containment**

  Containment is a component in which a self-contained component is contained in a container component. This container component has a Containment relationship to the contained component. The container component determines the life cycle of the contained component and is, therefore, the owner of the relationship and the CI referred. The container component is the sole supplier of the contained component. For example, a UNIX CI may have a Containment relationship with Memory, CPU, and Interface CIs representing that host's resources.

Which CIs should be related to the **CiCollection**? Only CIs directly used by the containing business application. For example, do not relate both a node and a database running on the node. Do not relate both a web service and the web server on which it runs. Only relate the web service.

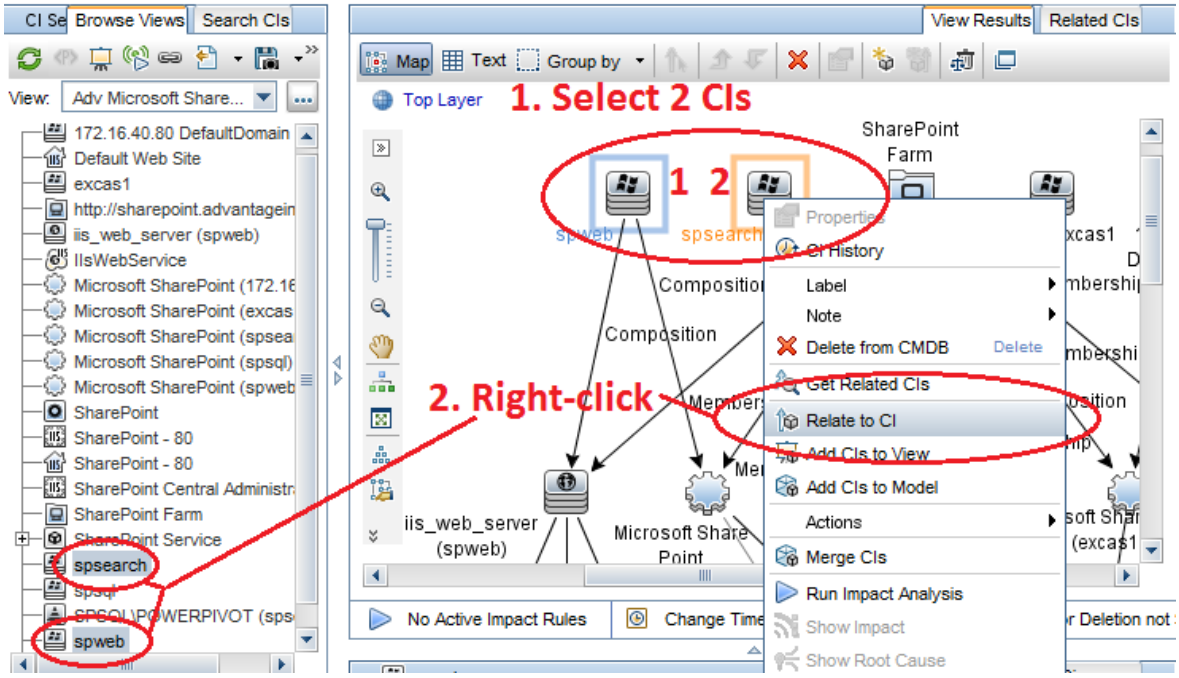


As shown in the diagram above, do not create additional links to lower layers of the infrastructure already connected. Creating unnecessary links can complicate or invalidate impact analysis.

To add CIs to the collection, use the **Relate to CI** tool in the UCMDB user interface. This powerful tool is often overlooked as a useful way to quickly relate many CIs to a collection. Using the **Search** function in CI mode lets you pick and choose CIs quickly and precisely. Using the **View** mode allows a TQL via a View to present the list of CIs to add. This may potentially be valuable, but since this is a static list, by definition, a TQL may not be the best way to produce a static list, unless the list is only part of what will be in the CI collection. Enrichments may also be used to create the relationships, based on matching data, but with the same caveats as using the View search.

The **Relate to CI** tool is accessible in the Modeling Studio and from other common tools, such as the **Get Related** tool.

An **IT Universe** view is shown below highlighting the **Relate to CI** tool.

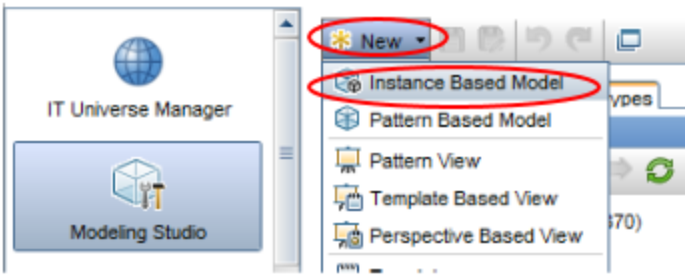


Use the **Relate to CI** search function to quickly locate CIs to add to a CI collection.

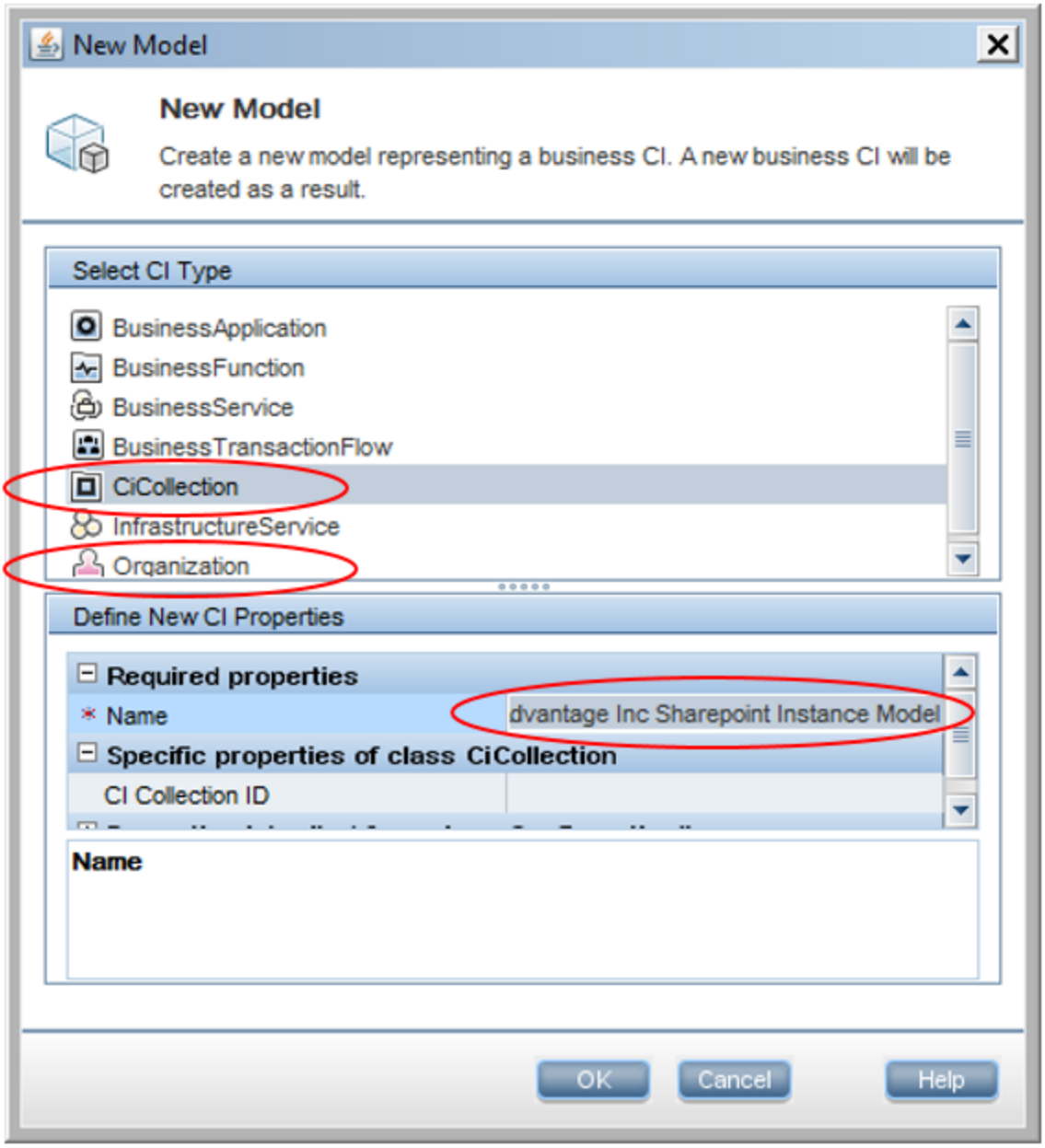
Instance-Based Models

Instance-based models can be populated from many places in the UCMDB—wherever a CI is displayed. Its right-click menu contains **Add CI to model**. This tutorial will go through creating an instance-based model in the Modeling Studio.

Click **New** > **Instance Based Model**.

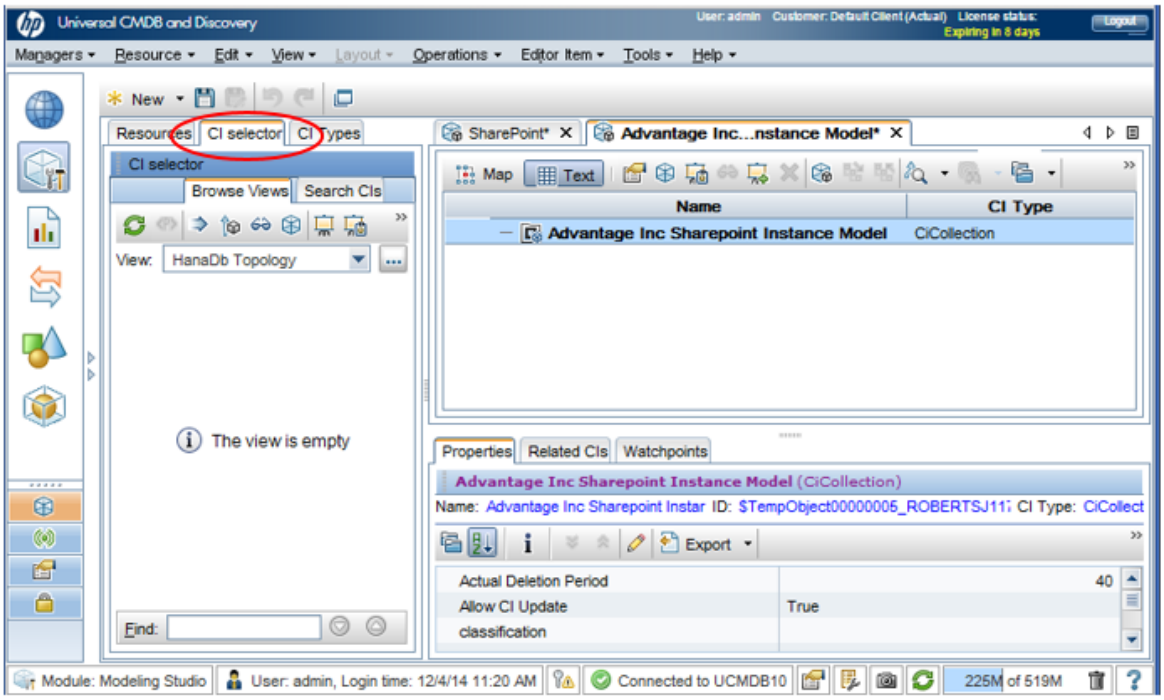


Select the type of top-level CI for the model, and give it a name. The CI types shown in the image below may all be used as models.

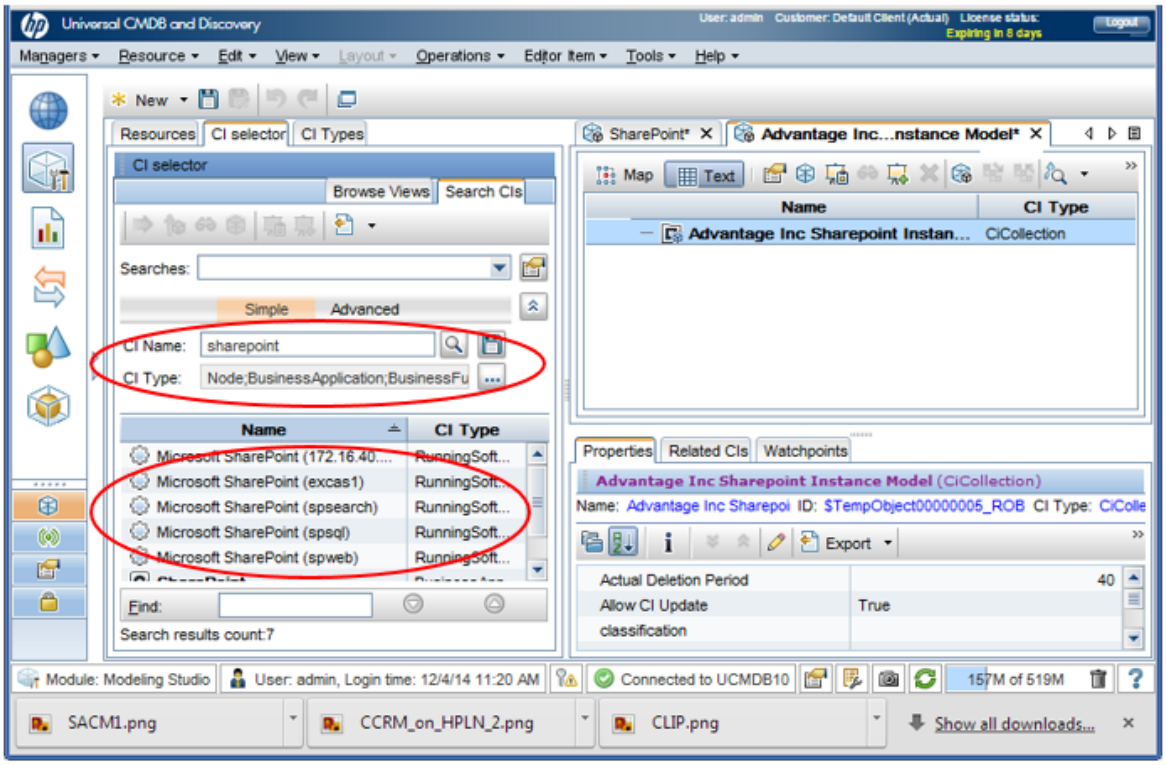


In the example above, a new model named **Advantage Inc Sharepoint Instance Model** is created using a **CiCollection**.

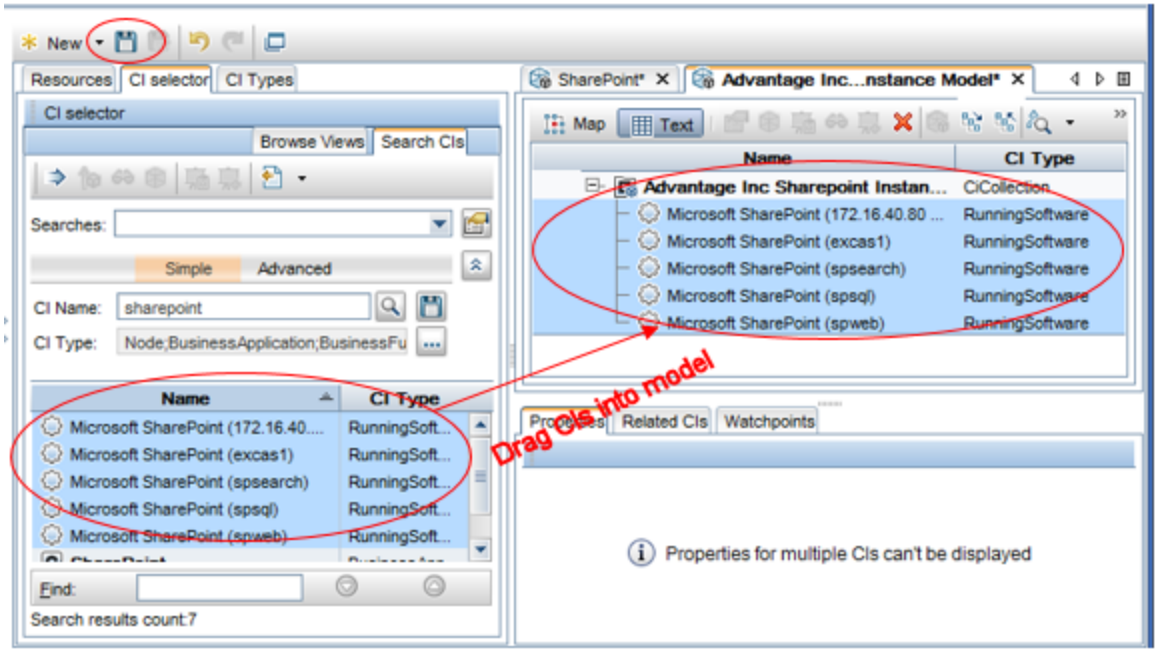
Click **OK** and then click the **CI selector** tab.



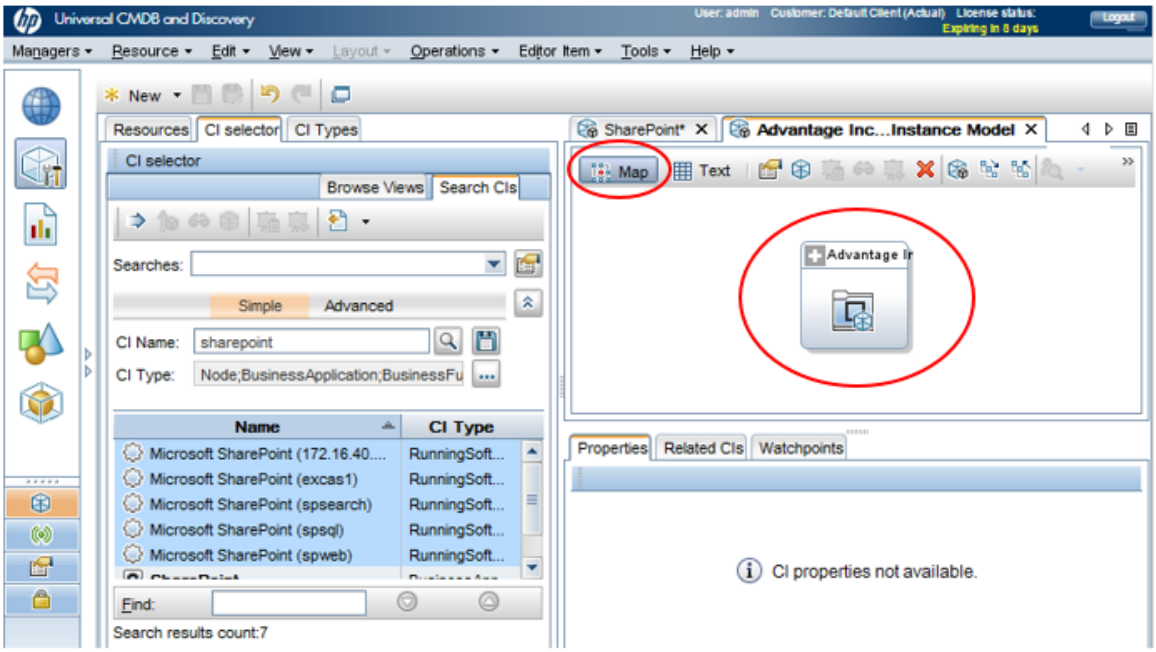
Click the **Search CIs** tab and search for CIs to be included in the model.



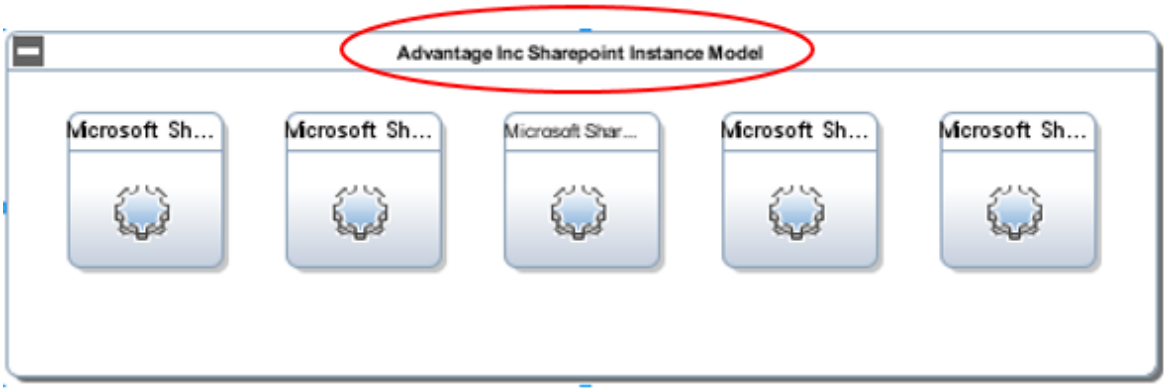
Drag the CIs into the list pane, then click the **Save** button.



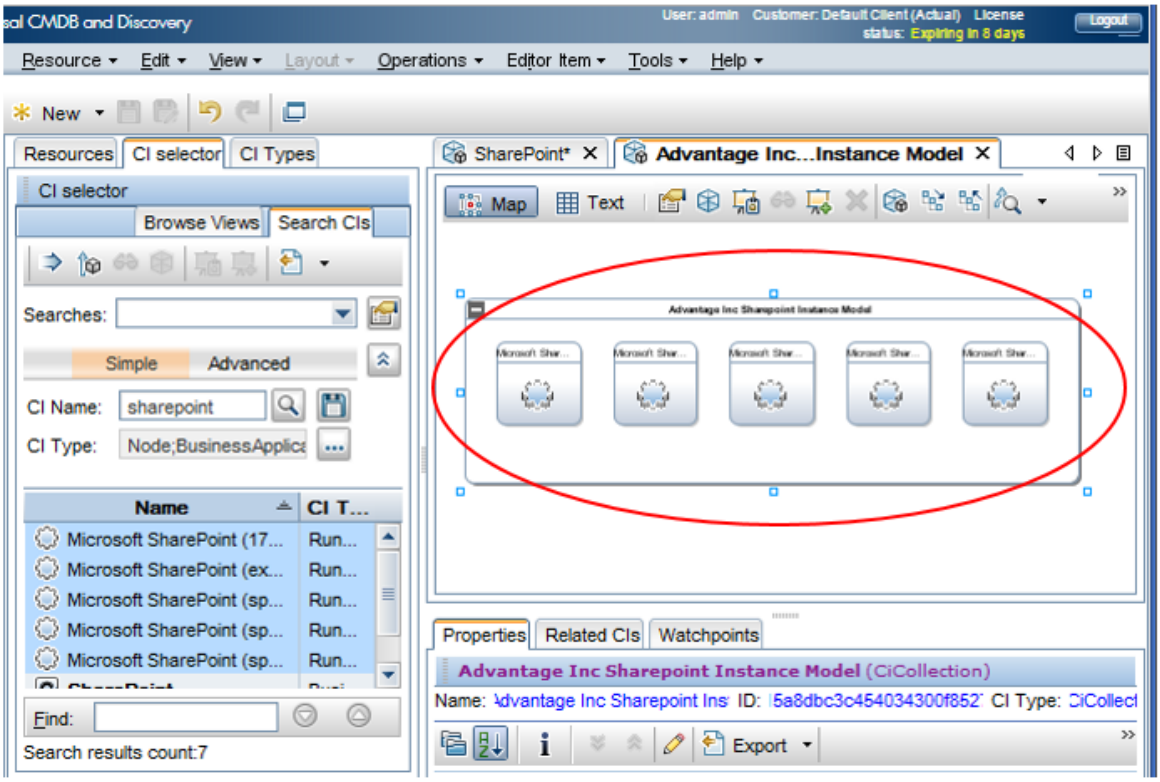
The **Map** button now shows the list.



Expanding the List shows the CIs contained in the list.



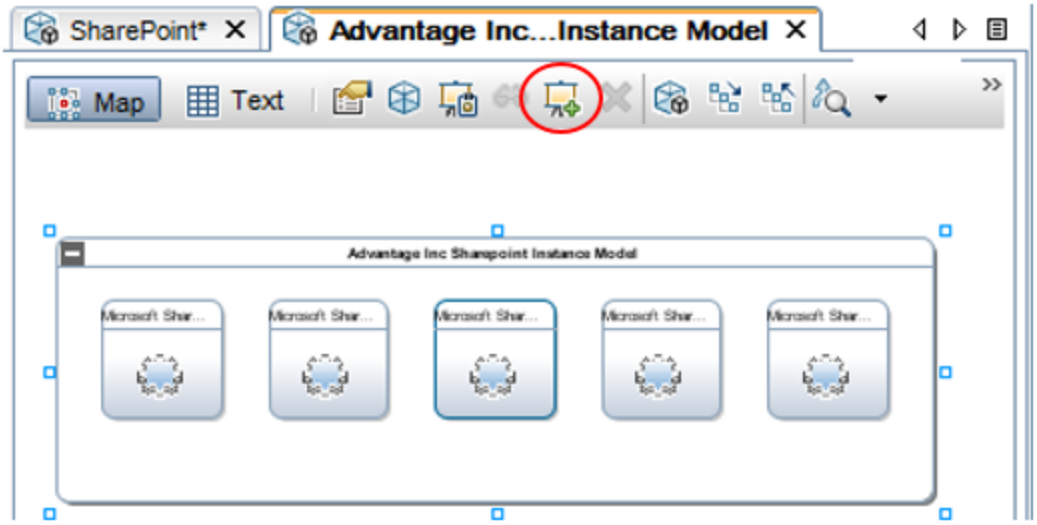
Now the model is complete and can be consumed.



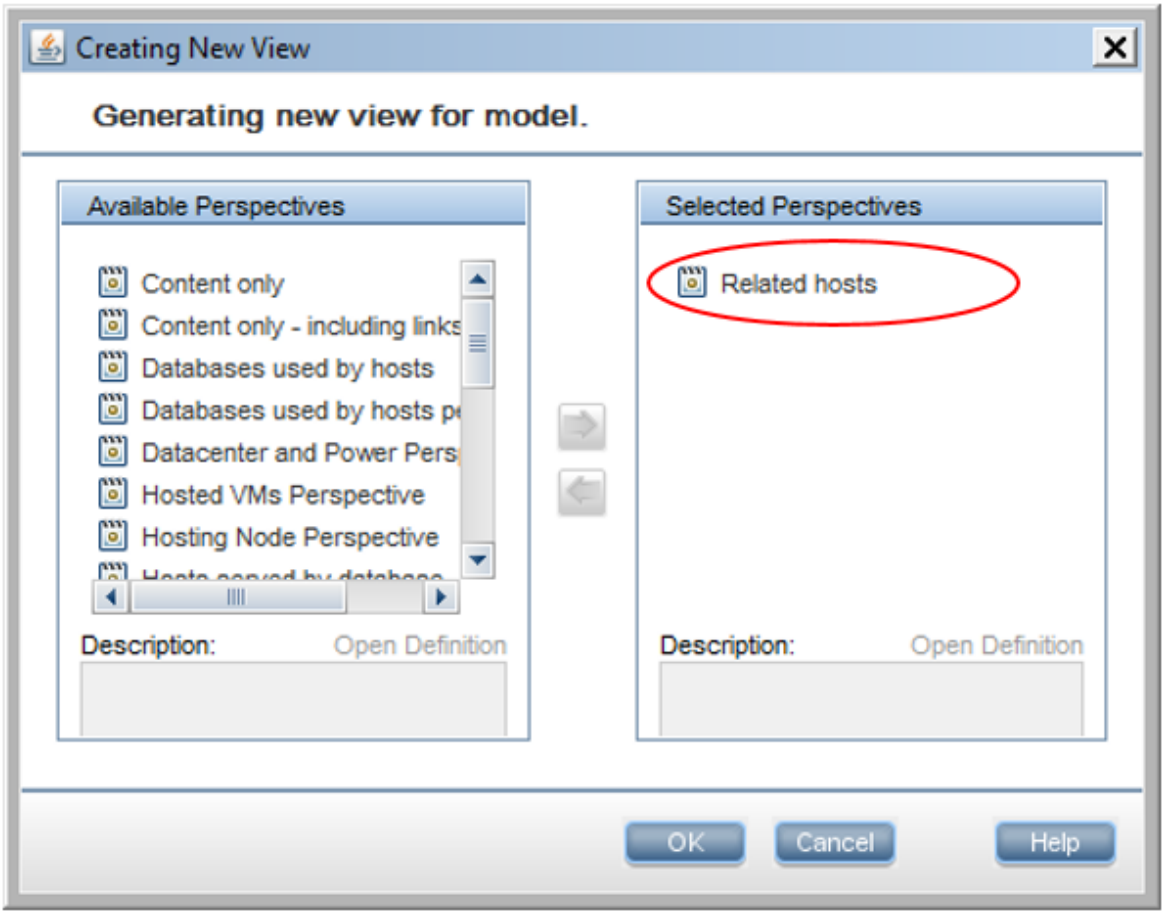
Perspective-based Views

Building on the previous example, we can generate a Perspective-based view for the list using the **View Builder** button.

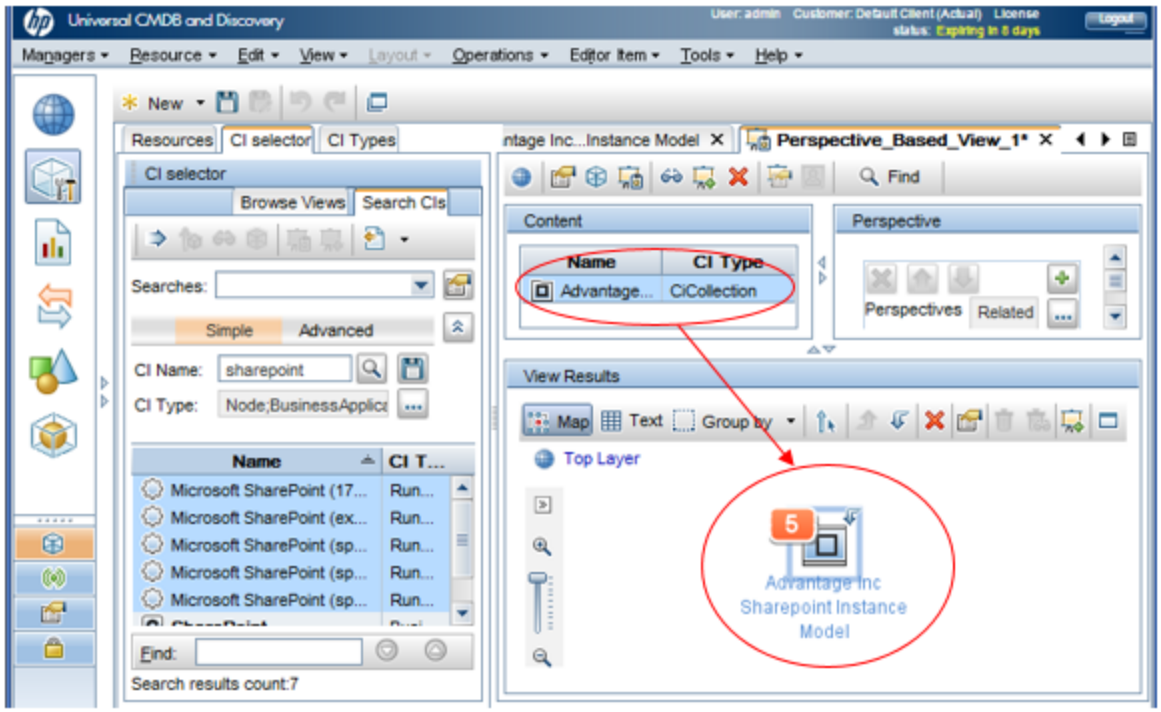
A list of perspectives is shown. These are out-of-the-box perspectives and do not need to be created.



Select the **Related hosts** perspective and click **OK**.

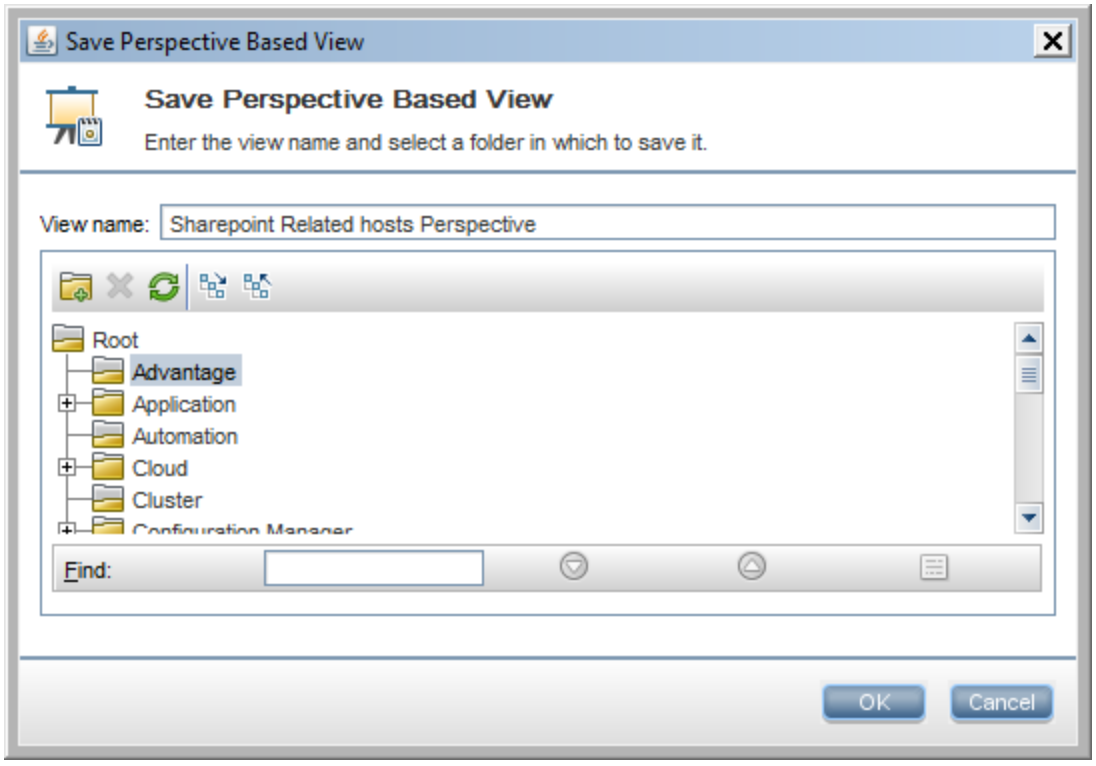


The **Related hosts** perspective now shows the view's related CIs by count in a list.



Double-click the container to reveal the **Related hosts** perspective. You can now save this model or use it as a service model or for any other useful purpose.

Click **OK** to save the Perspective-based View.



Perspectives

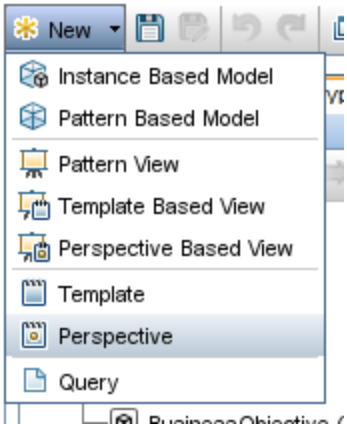
What is a Perspective?

A perspective is a simple and reusable query that exposes additional entities and relationships in order to render stakeholder-specific views for a given model.

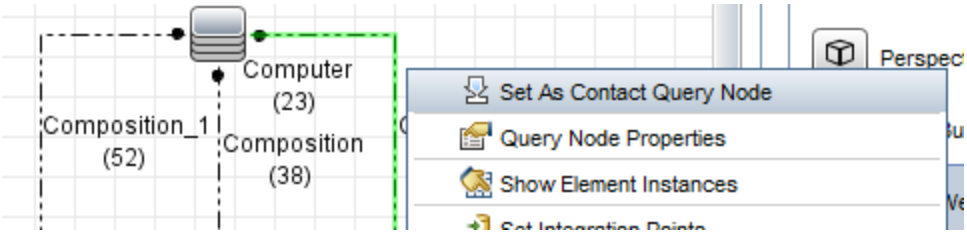
The following are examples of perspectives:

- Network perspective that exposes the underlying network infrastructure
- Storage Fabric perspective that exposes the underlying SAN storage fabric
- J2EE perspective that exposes the J2EE components related to the CIs in the model
- Many more out-of-the-box perspectives around hosts, resources, applications, and services

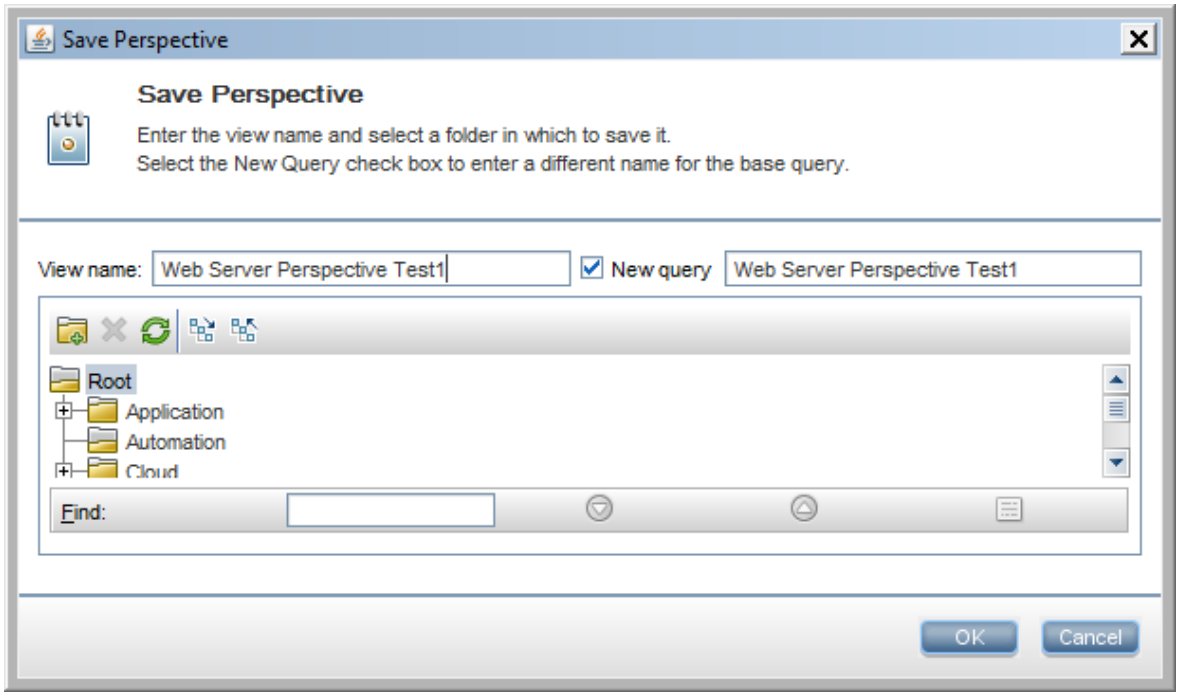
You must create or select a query first.



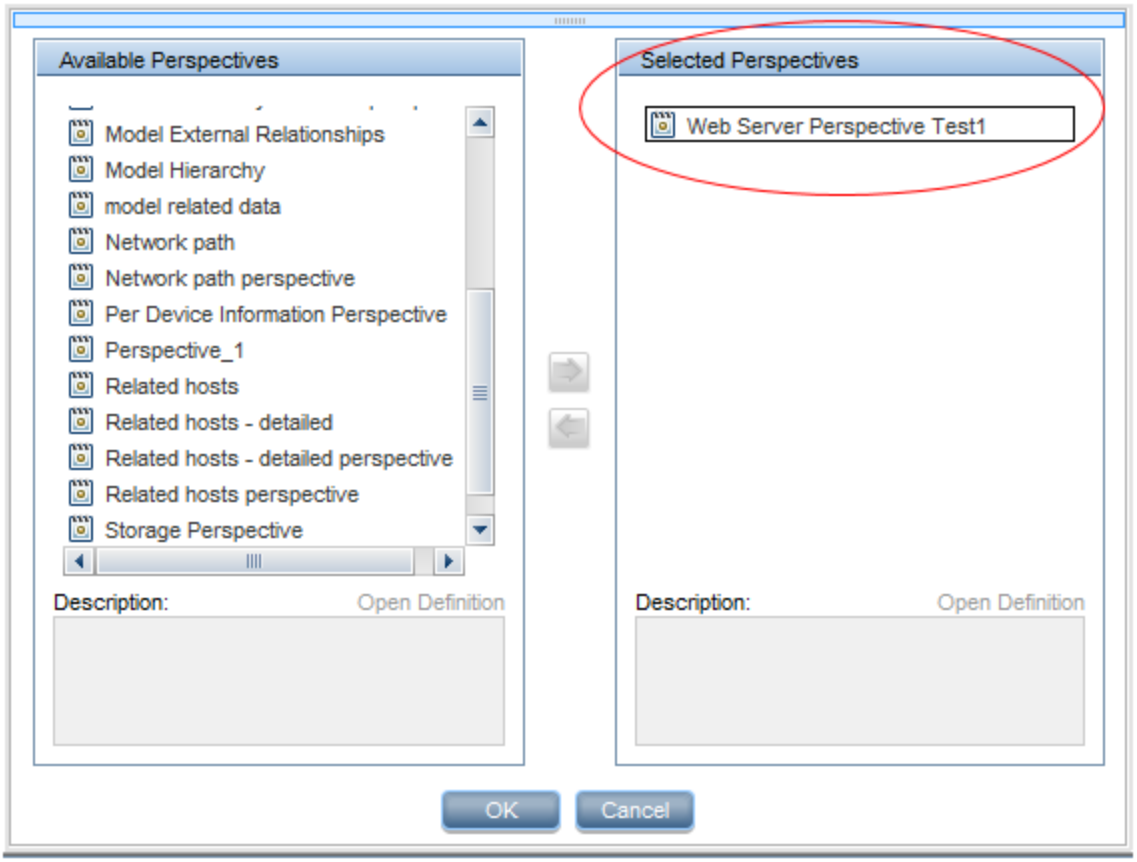
Then set one or more nodes as **Contact Query Nodes**.



Save the perspective with a meaningful name and click **OK**. The perspective now appears in all perspective lists in the Modeling Studio and can be used to create models.



The perspective selector in the model editor now shows the newly created perspective.



Perspectives can be used to filter model results for more specific content. A perspective can be applied with almost any scheme, allowing more reuse of models and queries.

Using Out-of-the-Box Perspectives

UCMDB provides many built-in perspectives for you to use or modify. The examples of perspectives previously listed are a partial list. Generally, the out-of-the-box perspectives are named to describe the type of content view applied in the perspective. For example, **Storage Perspective** focuses the view to SAN, disk, file systems, and related hardware. Likewise, **Network Perspective** produces a Layer-2-focused view.

You are encouraged to review and evaluate the default perspectives for use with your service models. Reusing default perspectives can save time and effort in delivering content that is precisely targeted to a specific consumer type.

Templates

This section includes:

Overview

Although utility views can provide a comprehensive listing of IT assets, they typically do not answer specific questions such as:

- Is server xyz running Solaris 9 or Solaris 10?

  or

- Which storage arrays support Oracle 9 databases?

  or

- Which databases use the storage array at 10.1.1.34?

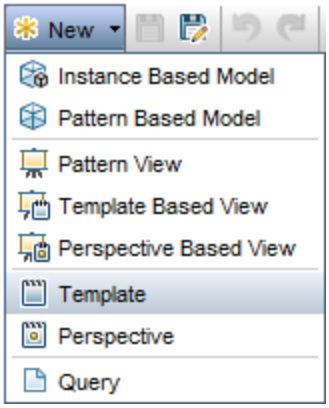
These questions may come up as service modelers and SMEs use the utility views to validate discovery results or build models in the Modeling Studio. Rather than modifying the utility view's underlying view TQL to answer each question, a utility view using parameterized view TQLs allow users of the view the ability to modify attribute conditions on the view to answer specific questions through the IT Universe Manager. This not only makes modification of the utility views much simpler by enabling limited modification of the view directly from the IT Universe Manager, but it also:

- Supports a security model which prevents users from accessing the utility's view TQL while still allowing controlled exposure to the view TQL conditions to facilitate consumer-specific viewing,
- Reduces maintenance by creating a **one size fits all** view that can be used by many consumers to answer different questions without creating one view to answer one question every time.

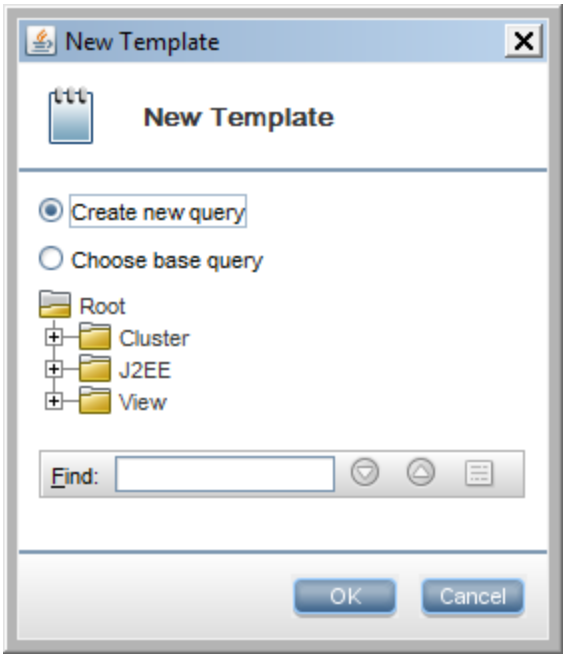
Templates, or parameterized views, are simple to enable and only require a few steps to utilize. When building the parameterized view TQL, add a node condition to any node whose instances vary based on an attribute value.

For example, to answer the question **Which hosts use Oracle 9, Oracle 10, or Oracle 11?**, a user would need to define a two-node view TQL with a host containing a database. Then the user would add a condition to the database node where the **Application Version** attribute has a parameterized value type. When viewing this parameterized view TQL from the IT Universe page, a small pencil icon along the top of the left-hand navigation bar exposes the modifiable parameters for the current view. Changing these values forces the parameterized view TQL to recalculate using a new value for the condition, resulting in data which helps answer a specific question such as **Which hosts run Oracle 10g?**

First, create a new template with **New** > **Template**.

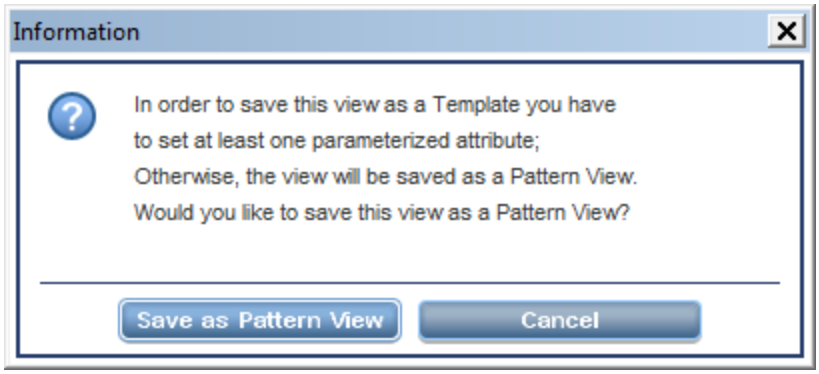


The first step in creating a template is to select or create a query. In this example, we will create a new query with **New Template** > **Create new query**.

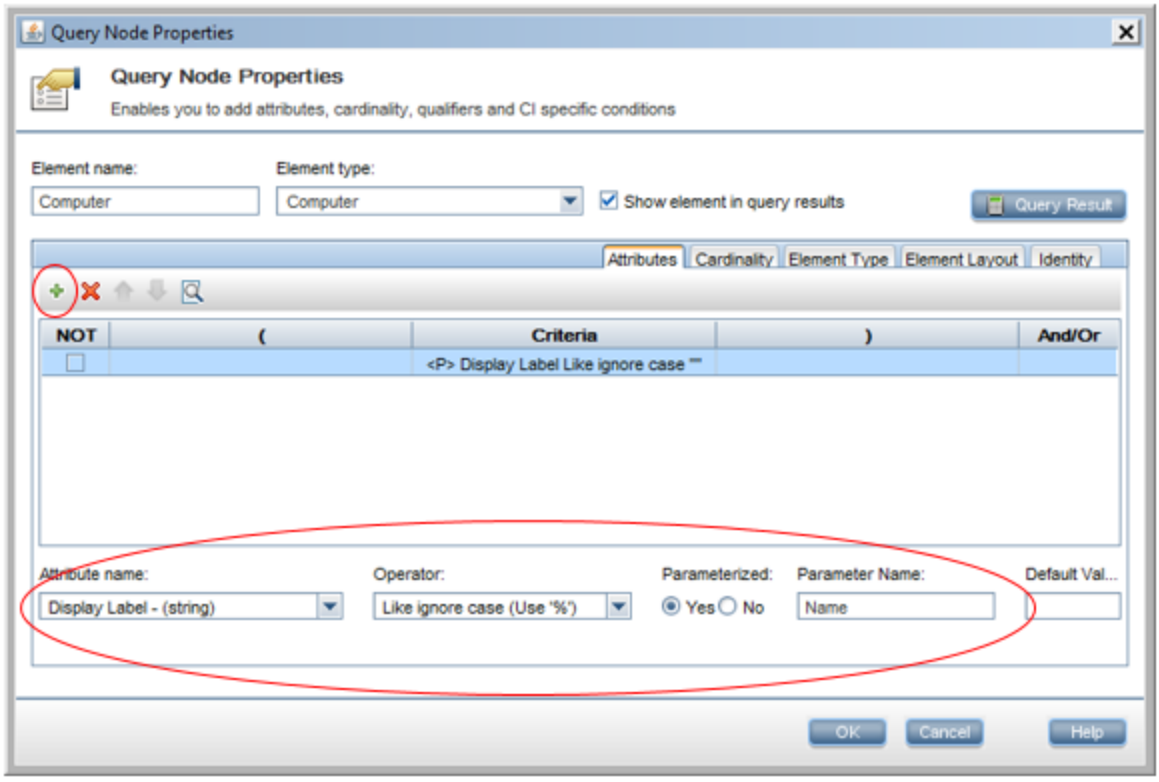


The query can represent a use case. For example, an IT administrator needs information about selected servers that support the Sharepoint service. Using a template, the administrator can supply a parameter and dynamically alter a template to render only those servers that match a given name. The parameter is used in the query to filter the **Name** of the **Computer** CI types.

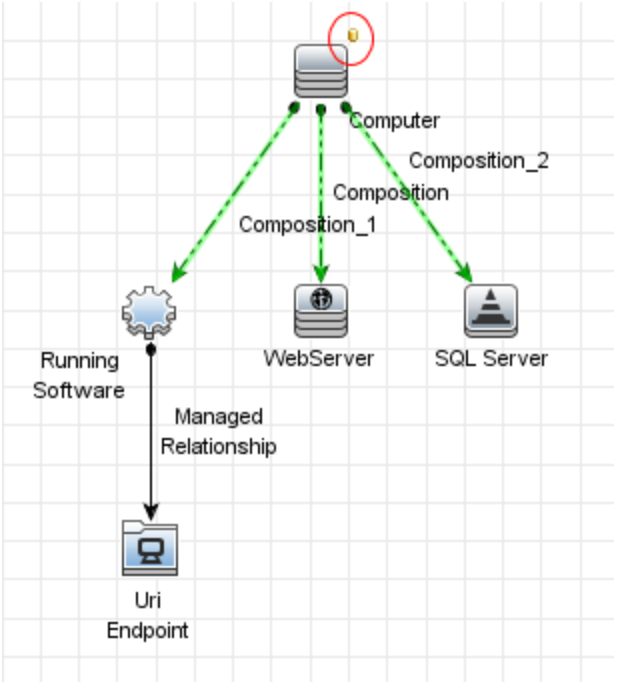
If you do not save any parameterized attributes, you can still save the template as a normal pattern view.



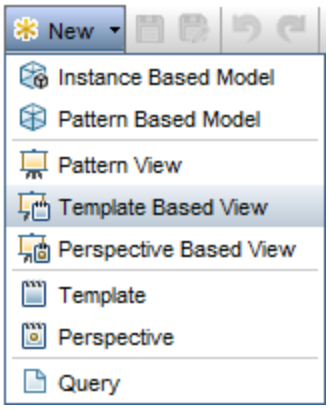
Create one or more parameterized attributes using the Query Node Properties dialog box.



A template query looks like a normal query with conditions. Save normally.



Now a new Template-based View can be created.

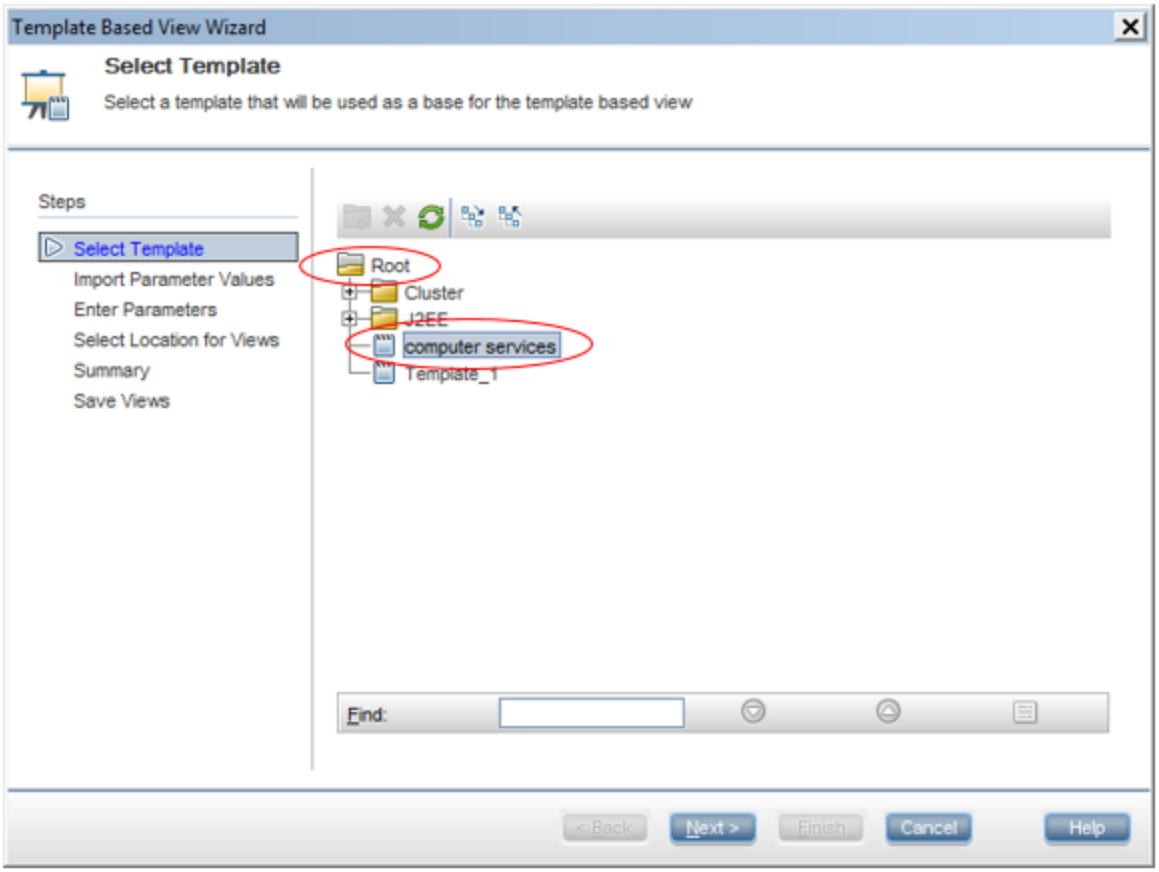


Template-based Views

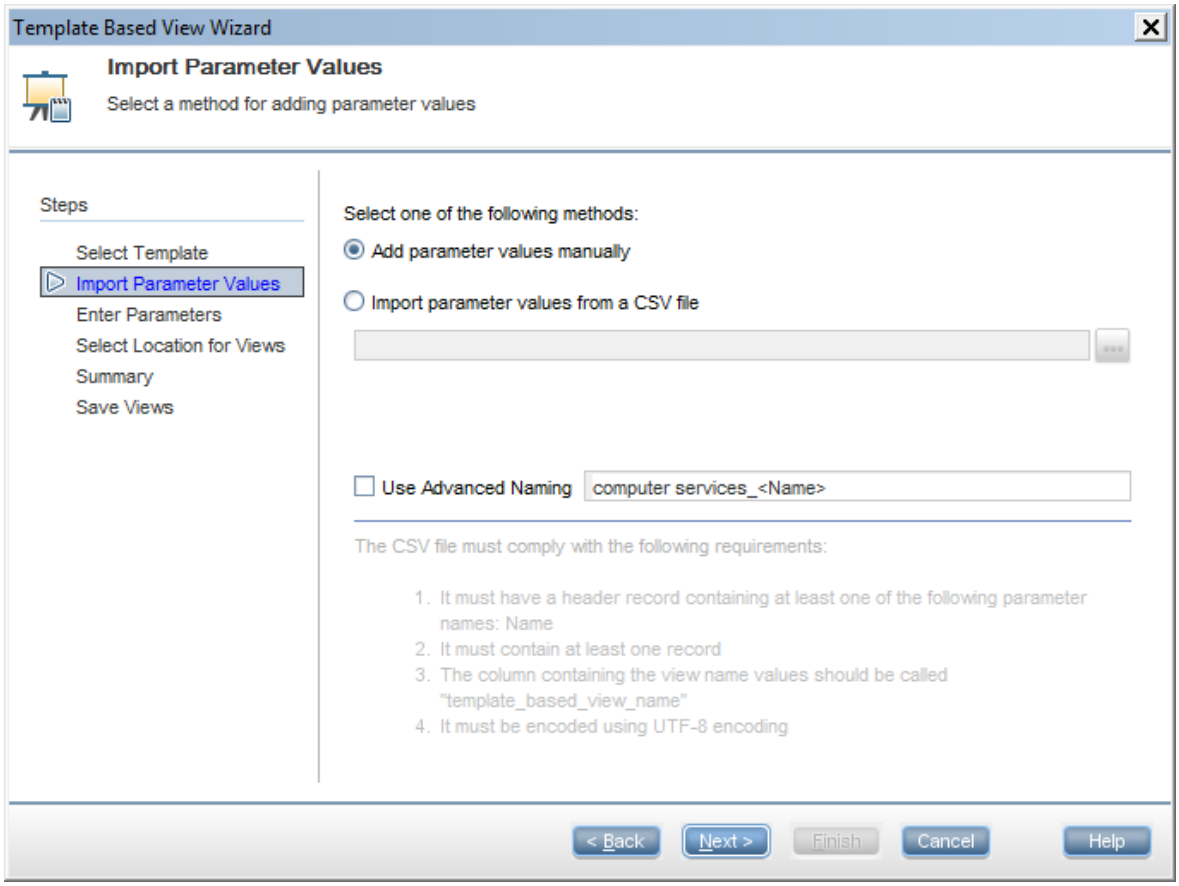
Like other views, a template-based view provides visualization configuration items such as folders, layers, and grouping. Template-based views also provide facilities for a user-friendly name and supply the parameter values that form the **template** part of template-based views. Parameter values are fed into TQL conditions which directly control the view's contents.

To continue the example started earlier, a parameterized query has been created and saved. Now we create a view to reference this query and render the view itself.

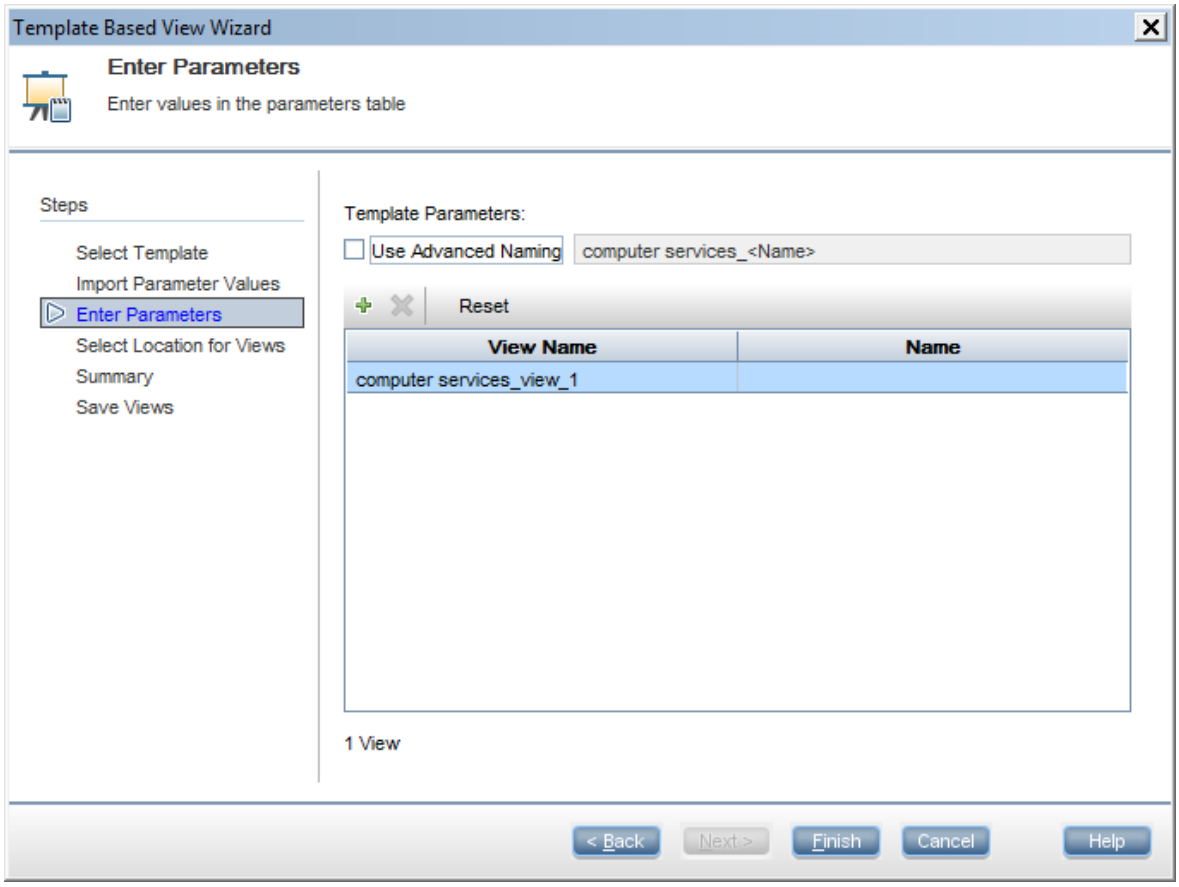
Select the template query created earlier to create the view.



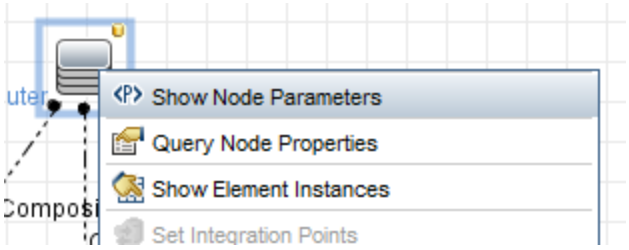
Parameters can be supplied in many ways via a CSV file, or using **Advanced Naming**.



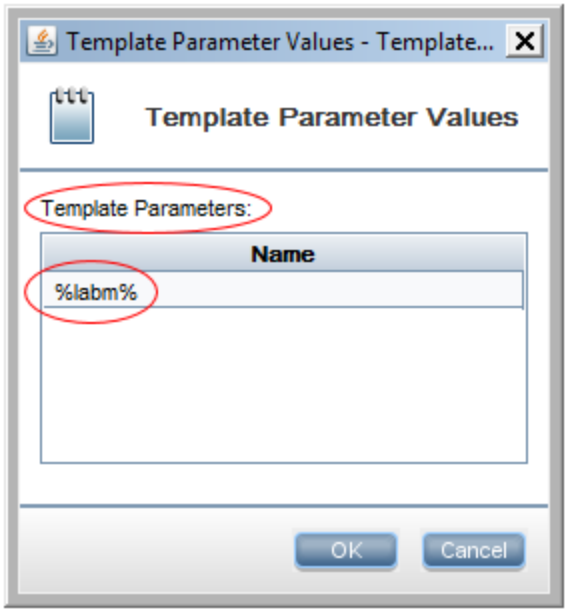
Enter parameter names to be used in the template.



Enter parameters to select the parameter value:



Apply the value **%labm%** to the **Name** attribute of the node in the query.



The view now displays only those models with names containing **labm** in the Web servers.