Business Application discovery and mapping with TADDM 7.2.1

Document version 1.0

September 9, 2013

Morten Moeller (moellerm@us.ibm.com)

Preface

In IBM® Tivoli® Application Dependency Discovery Manager (TADDM) v7.2.1 you can use the Grouping Composer feature to define Business Services, Business Applications and Collections. Business Applications and Collections may even be defined dynamically, so that they are populated in a dynamic way based on the information that is available in the TADDM database. When new resources are discovered, or dormant resources are removed, the Business Application or Collections are automatically updated.

The purpose of this paper is to provide the reader with the basic understanding of how to discover and define Business Applications. In addition the paper includes a thorough discussion of dynamic application definition using MQL Rules, and recommendations on how to automate application mapping using application patterns.

The first half, starting at *3 Discovering the business application components* on page 15, introduces discovery, static application definition, and exploring application topologies. In addition, basic concepts such as dependencies and relationships are discussed.

The second half of the paper, starting at *6 Business application definition using relationships* on page 53, is dedicated to dynamic application based on rules. When reading this section you are expected to understand the basics of application definitions, and have a good understanding of the Common Data Model as well. In this section you will find sample MQL queries that can be used to associate different types of resources with your business application.

As you will see, MQL rules are highly reusable, so in *7 Automating application mapping* on page 97 it is discussed how you can refine and manipulate the information in the TADDM database to help automate the application definition process.

This paper is a revised reprint of a hands-on exercise used at the Pulse 2013 conference. For that reason most of the information is worded as instructions on how to use and manipulate TADDM to produce certain results in a controlled environment. It is the hope of the author that you can use these instructions to replicate the scenarios in your own environment. The first two sections of the paper introduce the exercise environment, and describes the application environment that is discovered and mapped in the exercises.

To familiarize yourself with the discovered environment, please read sections 2 through 2.6 so you understand what you would have experienced had you attended the Pulse conference.

It is the hope of the author that you will find the information in this paper useful - even though you do not have access to the exercise environment, and therefore cannot reproduce the exact results as they are described in the this paper. If you have any comments or questions, please do not hesitate to contact the author.

Table of Contents

1 Before you begin

In these exercises, you will be introduced to the art of defining business applications in TADDM. The Common Data Model, which is the logical model that is use throughout most of the Tivoli portfolio to store IT infrastructure configuration and relationship information, defines an application this way:

An application represents a group of primarily software components and application servers that accomplish one or more tasks.

To successfully define your logical groupings, you have to ensure that the infrastructure components that should be included in the business application are discovered. When your components are registered in the TADDM database, you can define the business in one of three ways, each of which being more sophisticated and requires more in-depth TADDM knowledge than its predecessor,:

- Statically, using the TADDM Data Management Portal
- Dynamically, based on custom queries
- Relationship-based, based on manually created MQL queries

The following exercises have been designed in a modular fashion to allow you to focus on the particular topics of your interest. However before you jump to the section of your choice(s) you must start an application in the exercise environment that generates traffic, so the sample applications in the environment are active during discovery. Then you must perform a discovery to capture the configuration and relationship information about the components in the environment. All of these steps are included in *2 Introduction* on page 2.

Based on your experience you now have the following options:

- If you are familiar with the basic TADDM functionality for static and rule-based business application definition, you can jump directly to *6 Business application definition using relationships* on page 53.
- If your need an update on how to use rules in the TADDM Grouping Composer, you should consider starting at *5 Dynamically defined business applications* on page 41.
- If you have no experience in defining business applications in TADDM, start at *4 Static business application definition* on page 22.

2 Introduction

Welcome to the IBM® Tivoli® Application Dependency Discovery Manager (TADDM) 7.2.1 Discovering and mapping Business Applications exercises. In these exercises, you work with TADDM 7.2.1, using both the Discovery Management Console and the Data Management Portal, to discover and map a business application.

The main value that is provided by TADDM is that the configuration and relationship information is stored in the TADDM database, so you can visualize application topologies, and analyze configuration details. This information is critical for many related tasks such as IT Service Management, event management, change planning and more. Since the current configuration and relationships are stored in the TADDM database, TADDM can also identify changes to the infrastructure components. You can configure TADDDM to emit events to IBM Tivoli Monitoring or Netcool OMNIbus when changes are discovered. TADDM maintains the entire change history for the discovered resources and this is extremely helpful when managing incidents and problems in order to find the root cause.

When you deploy TADDM, you will very quickly suffer from information overload, because of all of the details TADDM discovers. Even in environments with less than 100 physical or logical servers, you will literally work with thousands of application server resources, so you need a way to group these together in groups of logically related components that represent your key business applications. Using these groupings, you can focus on the components that are related to specific tasks or business processes when leveraging the configuration information for troubleshooting and debugging incidents and problems. You can also use this information to identify the business systems that will be affected when applying changes to specific resources.

Unfortunately, TADDM has no way of applying your knowledge of your business to the discovered infrastructure configuration. Only you can determine that for example the *ABC.ear* software module is the key component of your payroll system, or that the WebSphere Cluster named *DDB22WS* represents the online banking staging environment. However, after you have identified the identifying resources – also known as *marker modules* - , you can use the information in the TADDM database to identify all the components that support this resource either directly or through discovered relationships.

The objective for these exercises is for you to understand how you can query the TADDM database to find related components, and use this information to create business application definitions that include all related resources. Applying this technique, you can map an entire business application system by designating any component as the marker module, and then use the discovered relationships to identify the related resources.

The flow of these exercises is such that you first discover all the individual components in the exercise environment. Once you know what resources are available, you use the TADDM explorer to reveal related components. This step is necessary for you to understand and appreciate how you can use these relationships to identify components that participate in the provisioning of a business application. In the final step you create the MQL rules necessary to map the various components to a business application.

Throughout these exercises screen captures are provided to help you verify that you achieve the expected results. Due to differences in the environments used to develop this documentation and the systems you are using, you might notice slight discrepancies between the results you see and the provided screen captures.

2.1 Benefits of using business application definitions

To gain the full benefit of Tivoli Application Dependency Manager, (TADDM) most organizations take advantage of TADDMs business application definition capabilities. mapping. By associating specific resources in your IT infrastructure with specific business applications, you enable the organization to easily determine which resources partake in the provisioning of specific solutions that are critical to run your business.

The application mapping capabilities are particularly interesting when you realize, that the information stored in the TADDM CMDB is leveraged throughout your management infrastructure to provide contextual information for most of your management processes, including, but not limited to, your event-, incident-, configuration-, and change management processes.

By associating technical resources to business systems, you augment the technical information with business context. Not only does this allow for easier communication between technical and non-technical teams, but it also facilitates reporting of technical issues and information in a business context, and enables integration of business priorities in your IT management processes. For example, using business application mapping, you can:

- Assess the impact to the business when planning change to the IT infrastructure.
- Automatically enrich events with business context in order to reduce mean time to repair (MTTR).
- Notify business application owners when changes to the application infrastructure have been discovered.
- Report inventories for business applications in order to provide the basis for accounting, charge-back or license management.
- Enhance cross-team collaboration by providing a common view and understanding of the application topology, configuration, and change history.

2.2 TADDM basics

The TADDM infrastructures for production environments include at least on Storage Server and one or more Discovery Servers. The Discovery Servers are responsible for executing discoveries. To traverse firewalls, discoveries can be supported by (distributed) anchor hosts. The purpose of the Storage Servers is to store the data that is discovered by the discovery servers. One Storage Server is named the Primary Storage Server, and this assumes special responsibilities to perform authentication, orchestrate storage operations, build the topologies and relationships based on the information in the TADDM database.

TADDM discovers the configuration and relationship information through snmp, ssh or wmi sessions between the TADDM server (or an anchor host) and the target systems. There is no need for you to install agents on the systems you plan to discover. In case you want to discover Windows-based systems, a TADDM Windows Gateway must also be present to act a proxy to convert discovery requests form the ssh protocol to wmi.

Depending on the level of details you want to discover, the following terminology is used:

Level 1 Discovery	Discovers active IP addresses so you know which systems are active in the environment
Level 2 Discovery	Access the operating system layer on the active systems and gathers information about the configuration and the active processes (servers).
Level 3 Discovery	Access the sub-systems and collect detailed configuration and relationship information.

To control TADDM discovery you must provide information about scope sets, discovery profiles and access control lists:

Scope sets	Specifies the IP addresses (or ranges) to be discovered.
Discovery Profiles	Define which sensors (small programs that are developed to discover configuration information for specific technologies) run during discovery. Discovery Profiles can also contain overwrites to default parameters used during discovery.
Access Control Lists	Provide credentials to be used to access the target operating systems and application server components.

TADDM provides two user interfaces. The Discovery Management Console is used by the TADDM administrators and operators to configure and execute discoveries. The Data Management Portal is primarily used to view the discovered information by all authorized users, but also contains a few administrative features such as user and group management.

TADDM also provides APIs that allow external components to update the information in the TADDM database. One of these is the REST interface which can be used by monitoring agents to update the information in the TADDM database based on the latest information available to the agent. This is particularly relevant in dynamic infrastructures, such as cloud environments, where it is critical to keep the configuration information up-to-date without having to perform a discovery.

2.3 TADDM relationship basics

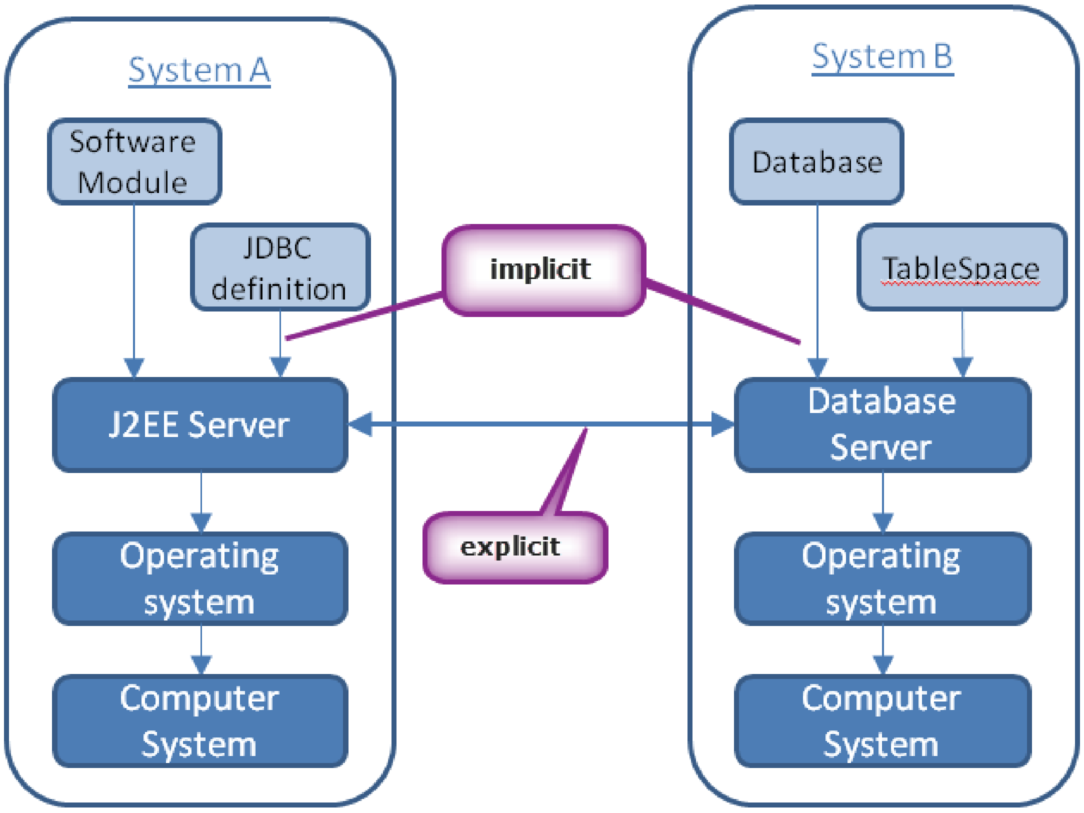
The relationships TADDM discovers between components in your infrastructure are key to mapping components to business applications.

TADDM discovers the configurations and relationships of computer systems, operating systems (OS), running applications, as well as many common networking and storage devices. When the configuration information about these resources is stored, implicit relationships are built between resources that rely on one another. The reliance is determined by the Common Data Model, which provide a logical structure by which your IT infrastructure components are stored in a database. For example, the implicit relationships defined in the Common Data Model dictate that an application server (for example a database) must be running on an operating system, and the operating system must be running on a computer system. If the operating system does not exist, how can you install and run the application server executable code?

Implicit relationships are enforced and created at the time resources are created in the database. Notice that these implicit relationships (for the most part) are vertical relationships that are used to ensure that all the logical representation of the components within a single system is logically sound, and that all the prerequisite parent resources for any component exist.

During discovery, TADDM also identifies the communications between otherwise unrelated components. If, for example, a J2EE server uses a JDBC connection to a database server at the time TADDM discovers the servers, the information is recorded, and saved in the TADDM database as *explicit* relationships. Contrary to implicit relationships, explicit relationships are optional. Just because you have installed a web server on system A, and a J2EE server on system B there is nothing that requires or guarantees that the two server components communicate with one another. The two application server components can easily be part of different business applications so it is more than likely that they should never communicate.

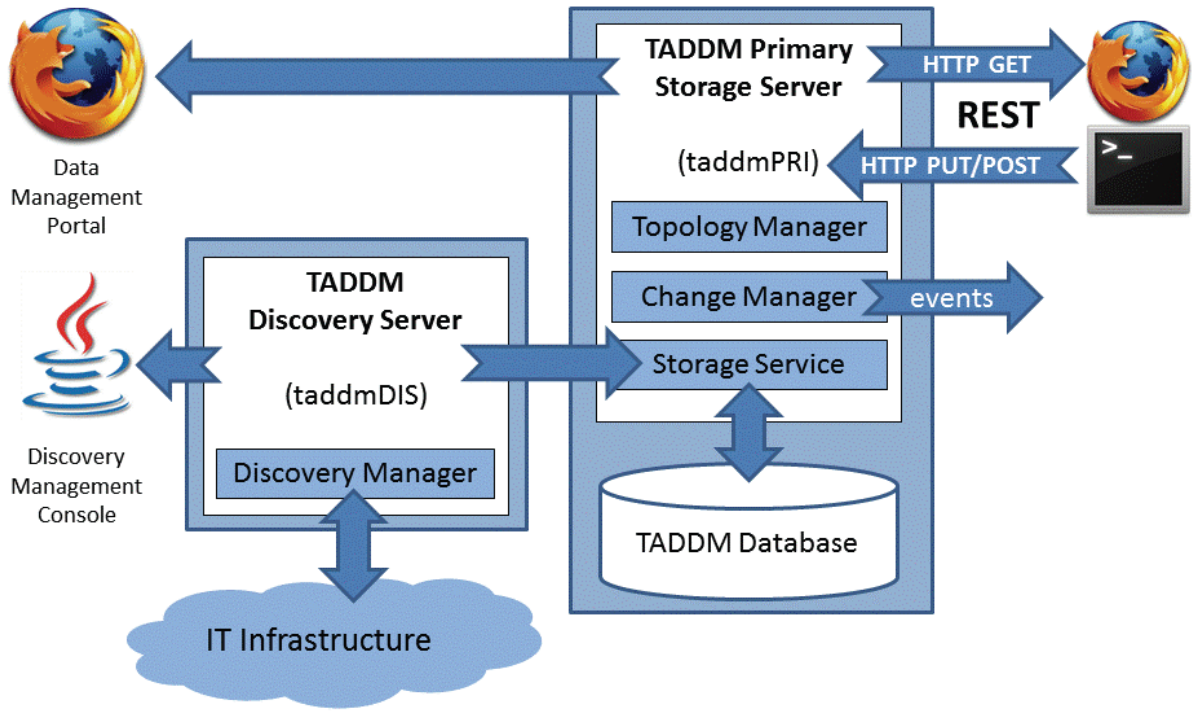
The following diagram illustrates the use of implicit and explicit relationships. Note that relationships are always uni-directional.



The way resources are stored, you can almost always find parent-child information in the resource attributes for resources that are implicitly related to one another. This is not the case for explicit relationships. Due to the optional nature of explicit relationships are stored in a separate table that contains the source, the target, and the nature of the relationship. However, the relationships can easily be referenced from queries in order to identify related components. In essence, business application mapping is all about building queries that identifies related resources using the explicit relationship information.

2.4 The exercise environment

To provide an environment in which you can try out the facilities provided by TADDM, you have access to two systems that make up a tiny TADDM environment.



You have access to a set of two virtual machines (VMs) deployed in the same environment:

taddmPRI	A TADDM 7.2.1 primary storage server hosted on Linux®. This system hosts the Data Management Portal you use to see the discovered information.
taddmDIS	A TADDM v7.2.1 discovery server on a Linux system configured with various applications for discovery. In addition, this system hosts the DB2 instance used by TADDM. This system hosts the Discovery Management Console you use to define and launch discoveries.

Because of the networking configuration for this environment, these two virtual machines are the only systems that are visible by TADDM. However, depending on the configuration of the system hosting the virtual environment, TADDM might also be able to access resources on the hosting platform.

The TADDM environment that you use, represents a pristine setup, as you would experience after initial installation in your own environment. Only the following modifications have been applied by manipulation of configuration parameters in the collation.properties file on the primary storage server:

- To accommodate for excessive load on the Discovery Server during discovery, the value for the com.collation.discover.DefaultAgentTimeout property has been raised from 600000 (10 minutes) to 1200000 (20 minutes).
- The frequency with which the background agents, including the topology manager and the change manager tasks, execute has been changed from the default value of every 4 hours to every 6 minutes by setting the value of the com.collation.topobuilder.agent.BackgroundAgentGroup property to Background@0.1

- The frequency with which the dependency agents execute has been changed from the default value of every 30 minutes to every 3 minutes by setting the value of the com.collation.topobuilder.agent.DependencyAgentGroup property to Dependency@0.05
- To show all the pre-build topologies, the value for the com.collation.gui.doNotShowGraphs property has been set to blank.
- The change event forwarding has been enabled by setting the value of the com.ibm.cdb.omp.changeevent.enabled parameter to true.

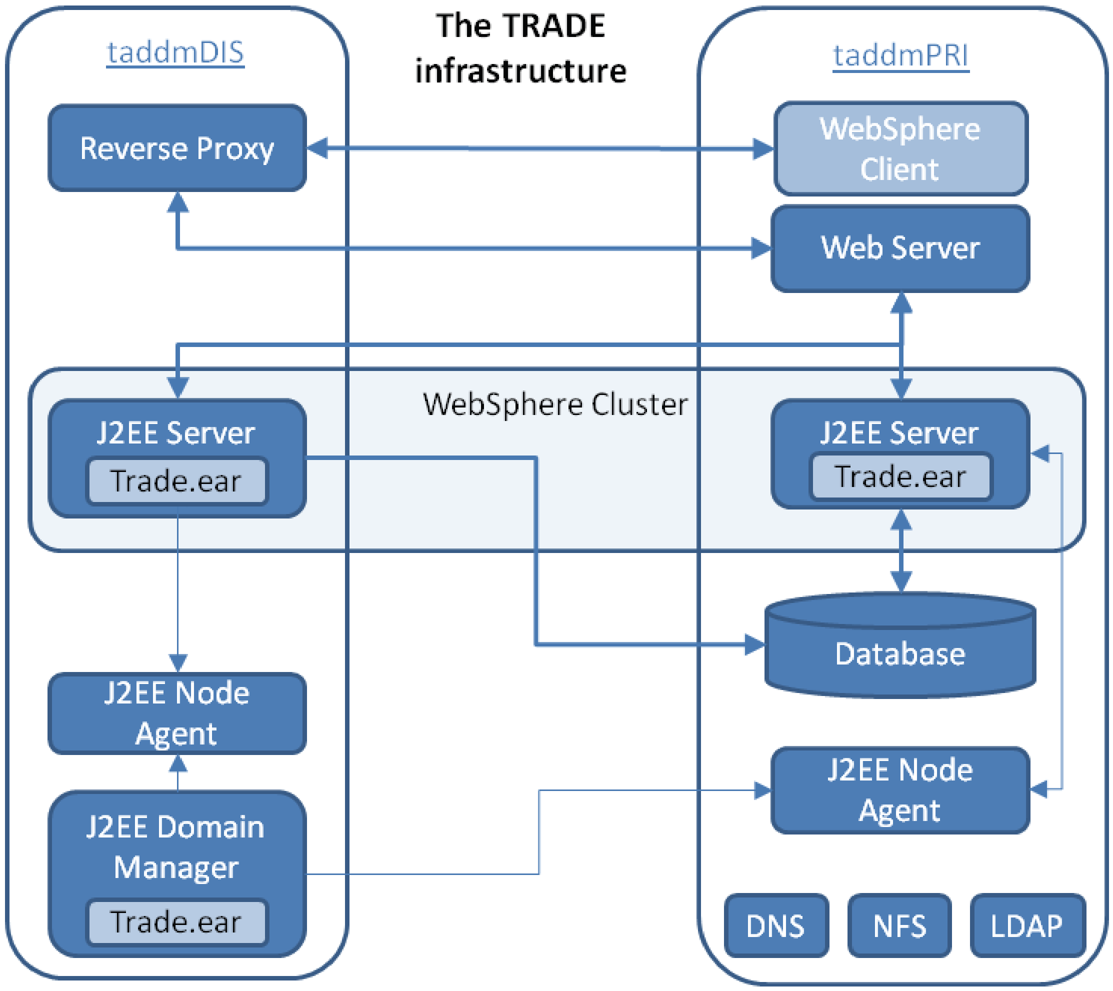
In the following exercises, you create the necessary definitions to perform a discovery of the network, computer systems, and application server configuration and relationships of the virtual infrastructure. When you have discovered the systems, you define Business Applications, augment the information with data from external sources, and experience how you can report on the information in the TADDM database.

2.5 The Trade application environment

In order to provide a business application to discover and map, a three-tiered application infrastructure has been installed on the two systems in the exercise environment. This application environment is made up of a front-end, reverse proxy web server, an application web server, an application cluster made up of two J2EE application servers and a management instance, and a single database instance. In this environment, a number of applications are available, including the sample applications and Trade, and PlantsByWebSphere.

The Trade application has a client application that can be used to generate application requests, which are processed by all the server components in the environment. When the Trade client application execute, the server components will be communicating with one another, and these connections will be discovered by TADDM. If no traffic is flowing between the server components when TADDM is discovering, the relationships between the server components cannot be discovered (for obvious reasons), and you will miss important information about your application topology.

The following diagram illustrates how the application server components communicate.



Besides the application server components, a DNS server is also running on the taddmPRI system. When you start the Trade client application configure it to send requests to a web address that is resolved by the DNS. Notice that the taddmPRI system also hosts an LDAP server. This is used to authenticate users of the J2EE servers in the application infrastructure.

2.6 Additional application server and services components

In addition to the TADDM and Trade application infrastructure, the exercise environment contains a number of standalone J2EE and database servers. Since these servers are not being actively used during discovery, you will not see if they are connected to one another or not.

The exercise environment also contains a number of active IT Services, such as LDAP, DNS, and NFS. During discovery these services will be identified and registered in the TADDM database. The relationships between the components that use these services and the services themselves are naturally recorded as explicit relationships in the TADDM database.

Exercise 1: Verifying the environment

In order to complete these exercises you must ensure that all the TADDM components as well as the Trade application infrastructure are operational. The two exercise systems have been configured to start all the necessary components automatically, so all you need to do is to verify that they all started successfully.

2.7 The TADDM environment

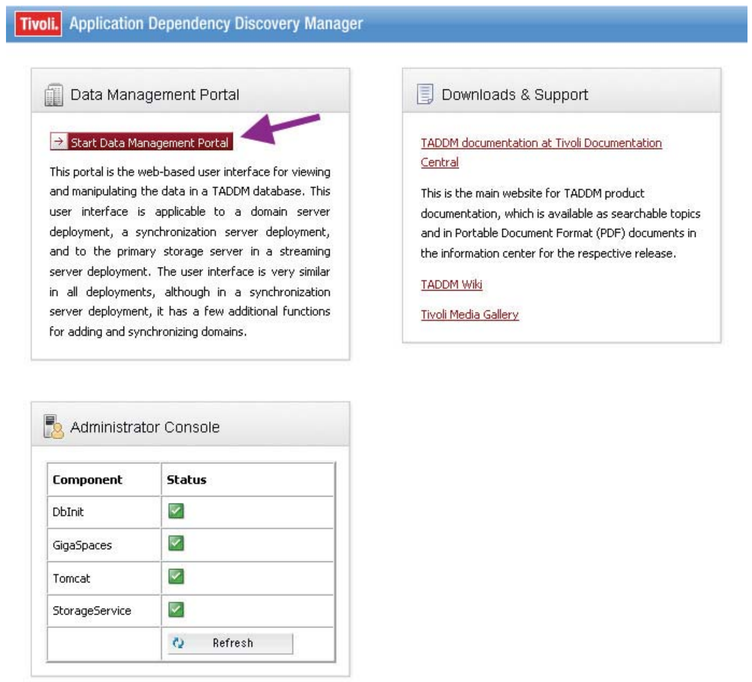
To verify that the TADDM components are operational, you launch the Data Management Portal hosted on the TADDM Storage Server system, and from there, you verify that the TADDM Discovery Server is started. Once you have ensured that the TADDM components are running, you launch the Discovery Management Console so you are ready to start discovery of your business application.

To do all of this, complete these steps:

1. Launch the Data Management Portal.
   a. From the desktop of any of the exercise systems use the Firefox link ( ) to launch a browser. When the browser has started, open the following web address:

      ```
      http://taddmPri.tivoli.edu:9430
      ```

   b. When the TADDM Portal for the TADDM Storage Server system has loaded, verify that the operational status of all four sub-tasks is started - indicated by a green check mark (✓).



Tivoli. Application Dependency Discovery Manager

Data Management Portal

Start Data Management Portal

This portal is the web-based user interface for viewing and manipulating the data in a TADDM database. This user interface is applicable to a domain server deployment, a synchronization server deployment, and to the primary storage server in a streaming server deployment. The user interface is very similar in all deployments, although in a synchronization server deployment, it has a few additional functions for adding and synchronizing domains.

Downloads & Support

TADDM documentation at Tivoli Documentation Central

This is the main website for TADDM product documentation, which is available as searchable topics and in Portable Document Format (PDF) documents in the information center for the respective release.

TADDM Wiki

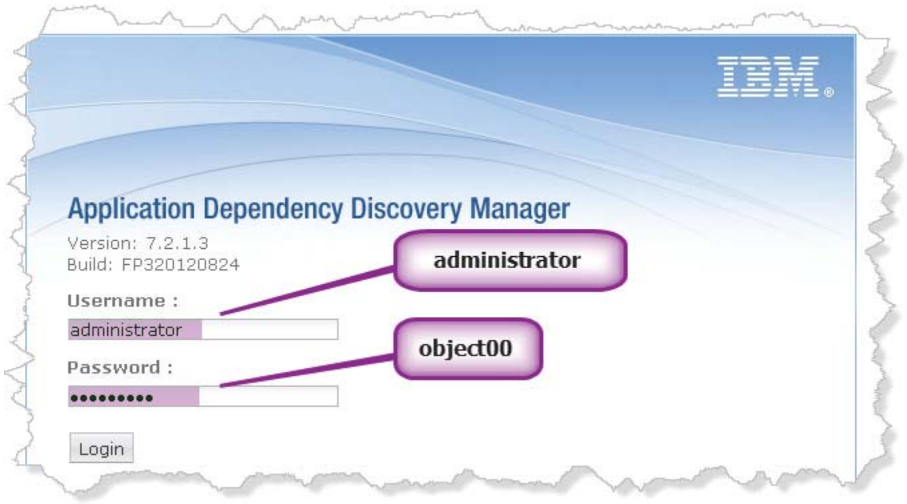
Tivoli Media Gallery

Administrator Console

Component	Status
DbInit	✓
GigaSpaces	✓
Tomcat	✓
StorageService	✓
	Refresh

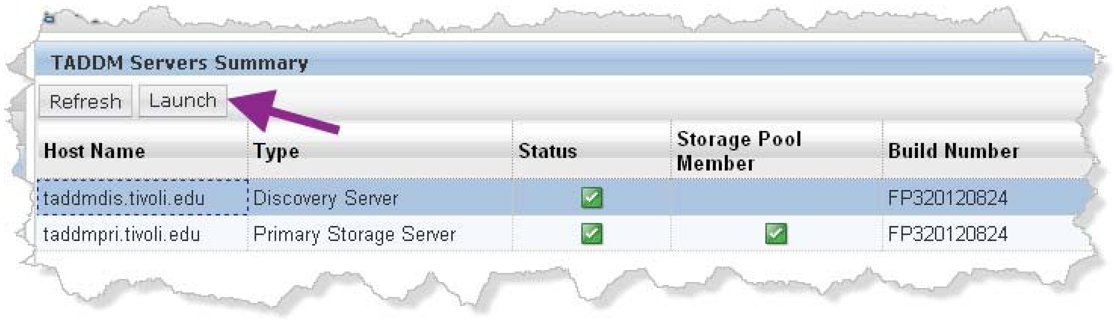
c. To launch the Data Management Portal, click Start Data Management Portal. When you are prompted to provide login credentials use these values:

Username:	`administrator`
Password:	`collation`



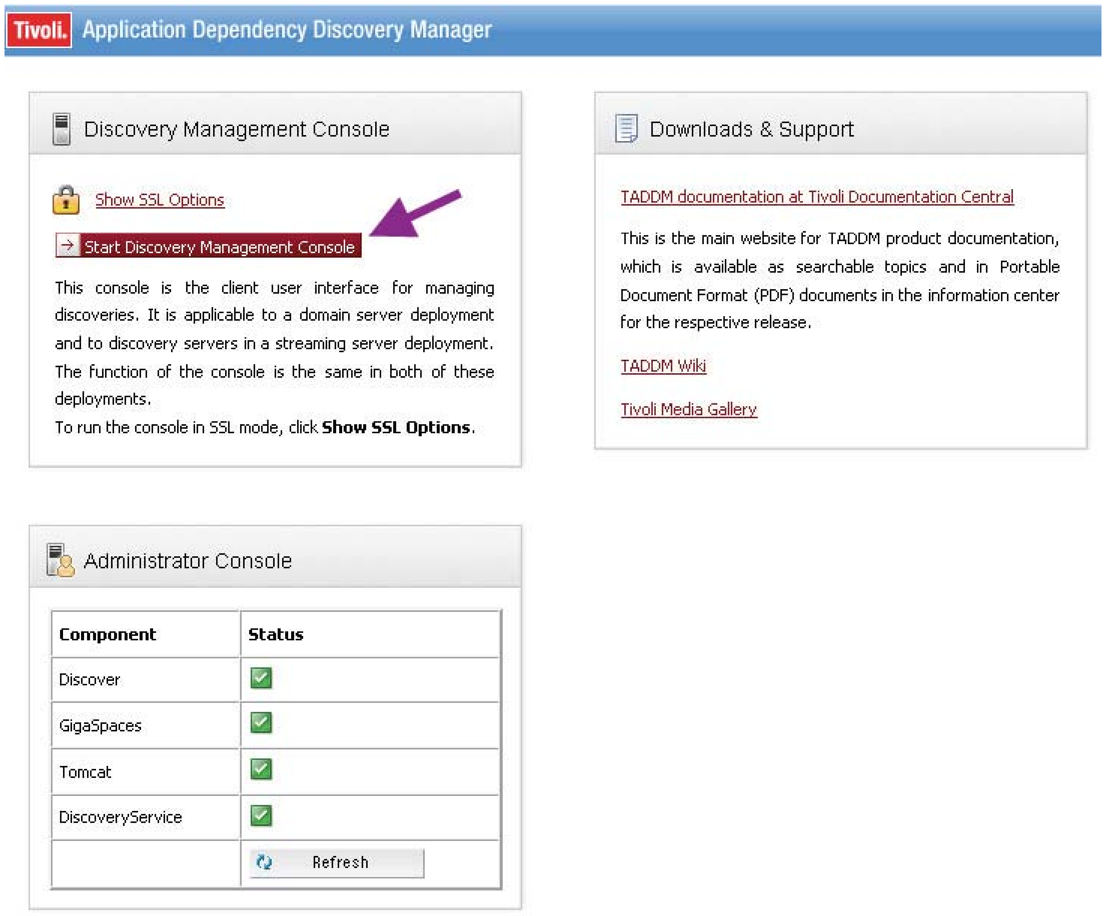
2. Verify the operational state of the TADDM Discovery Server.

   From the TADDM Data Management Portal, navigate to **Administration** > **TADDM Servers Summary**, and verify that the status of the Discovery Server hosted on the taddmDIS.tivoli.edu system is started.



Host Name	Type	Status	Storage Pool Member	Build Number
taddmdis.tivoli.edu	Discovery Server	✓		FP320120824
taddmpri.tivoli.edu	Primary Storage Server	✓	✓	FP320120824

3. Launch the Discovery Management Console.
   a. From the Data Management Portal, select the Discovery Server, and click Launch to open the TADDM Portal for the discovery server a new browser window.
   b. When the TADDM Portal has loaded click Start Data Management Console.

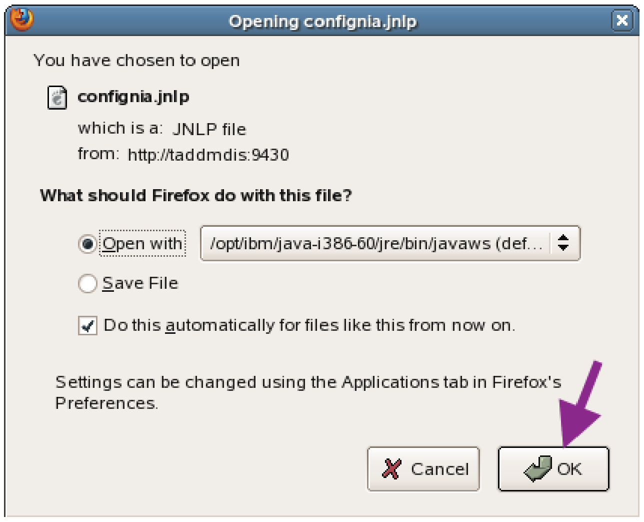

Tivoli. Application Dependency Discovery Manager
Discovery Management Console
Show SSL Options
Start Discovery Management Console
Downloads & Support
TADDM documentation at Tivoli Documentation Central
TADDM Wiki
Tivoli Media Gallery
Administrator Console
Refresh

This console is the client user interface for managing discoveries. It is applicable to a domain server deployment and to discovery servers in a streaming server deployment. The function of the console is the same in both of these deployments.
To run the console in SSL mode, click **Show SSL Options**.

This is the main website for TADDM product documentation, which is available as searchable topics and in Portable Document Format (PDF) documents in the information center for the respective release.

Component	Status
Discover	✓
GigaSpaces	✓
Tomcat	✓
DiscoveryService	✓

   c. When prompted to select an action for opening the confignia.jnlp file, accept the default **Open with** javaws and click **OK**.



You have chosen to open

confignia.jnlp

which is a: JNLP file
from: http://taddmdis:9430

What should Firefox do with this file?

[✓ Open with] /opt/ibm/java-i386-60/jre/bin/javaws (def...

Save File

[✓] Do this automatically for files like this from now on.

Settings can be changed using the Applications tab in Firefox's Preferences.

d. When you are prompted to provide login credentials use these values:

Username:	`administrator`
Password:	`collation`

When your credentials have been accepted, the Discovery Management Console will be launched.

4. Keep both the Data Management Portal and the Discovery Management Console open, you will use them in a short while.

You have now verified that the components that facilitate the TADDM application infrastructure are operational. Next you must verify that the sample application environment is also running.

2.8 The Trade application infrastructure

As already described, the application infrastructure that is implemented on the two systems in the exercise environment host several web server, and application server components along with a single database. In addition the server components have been configured to use LDAP, DNS, and NFS services which are common to the entire environment.

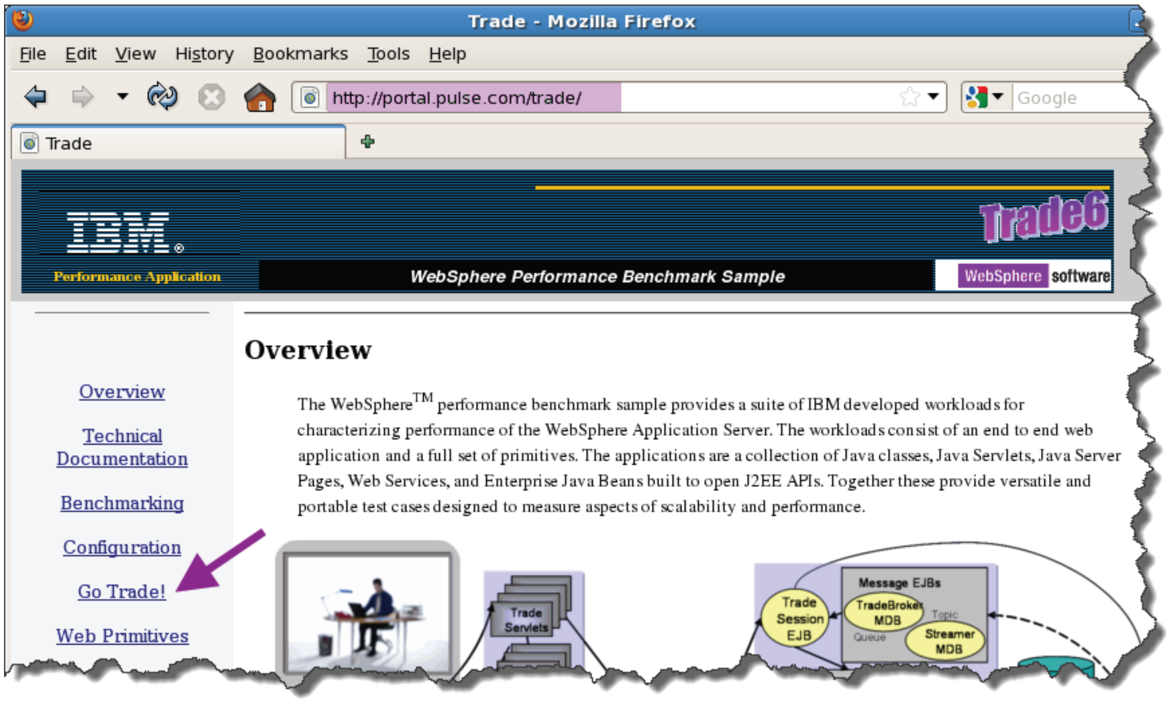
To verify that all the components are operational, all you need to do is to submit a transaction request against the web address of the reverse proxy web server. The transaction will flow through all the components, and if you receive a (meaningful) answer, you can assume that all the required components are running.

To verify that the application environment is operational, complete these steps:

1. From any browser in the exercise environment, open a new tab, and enter the following web address in the address field:

```
http://portal.pulse.com/trade
```

Press **Enter** to access the web address from your browser.

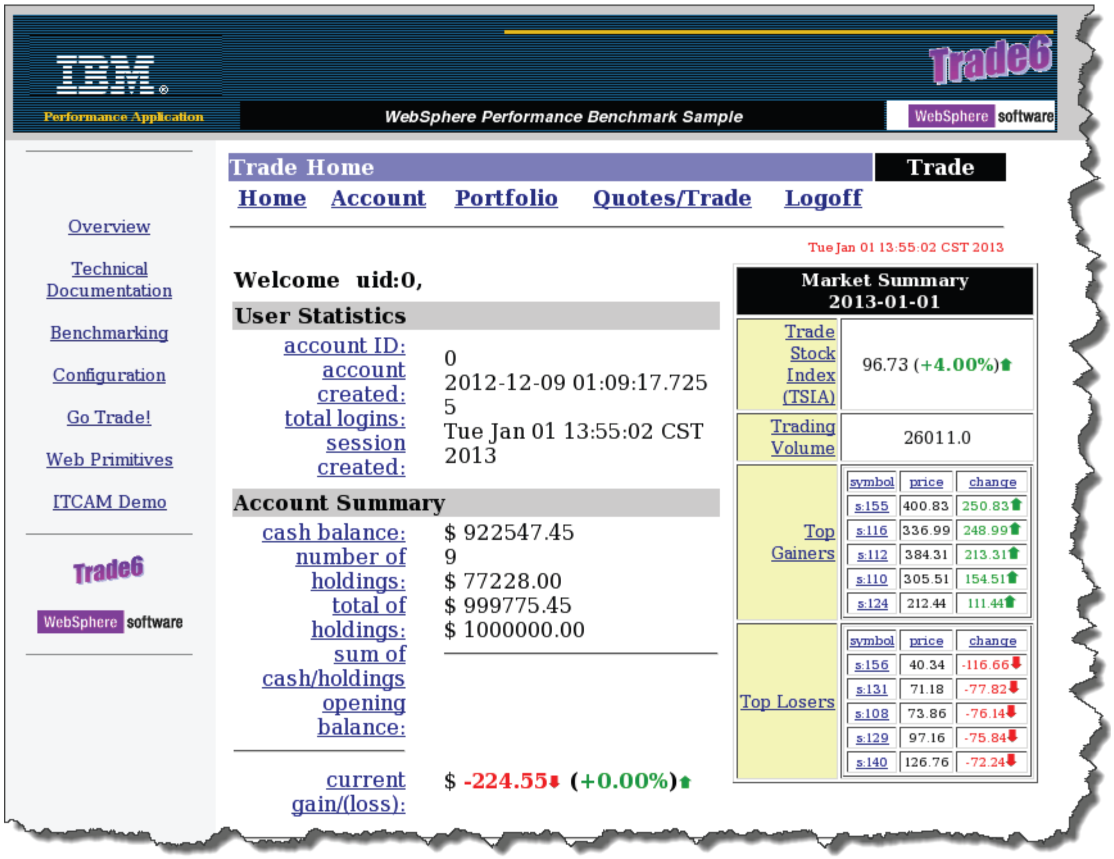


Overview

The WebSphere™ performance benchmark sample provides a suite of IBM developed workloads for characterizing performance of the WebSphere Application Server. The workloads consist of an end to end web application and a full set of primitives. The applications are a collection of Java classes, Java Servlets, Java Server Pages, Web Services, and Enterprise Java Beans built to open J2EE APIs. Together these provide versatile and portable test cases designed to measure aspects of scalability and performance.

2. When the Trade application Overview page is shown, click **Go Trade**.

   On the Trade Login page, accept the default information and click **Login**. The Trade home page for the user you used is shown:



Tue Jan 01 13:55:02 CST 2013

Welcome uid:0,

User Statistics

account ID:	0
account created:	2012-12-09 01:09:17.725
total logins:	5
session created:	Tue Jan 01 13:55:02 CST 2013

Account Summary

cash balance:	$ 922547.45
number of holdings:	9
total of holdings:	$ 77228.00
sum of cash/holdings	$ 999775.45
opening balance:	$ 1000000.00
current gain/(loss):	$ -224.55 (+0.00%)

Market Summary 2013-01-01

Trade Stock Index (TSIA)	96.73 (+4.00%)
Trading Volume	26011.0

Top Gainers

symbol	price	change
s:155	400.83	250.83
s:116	336.99	248.99
s:112	384.31	213.31
s:110	305.51	154.51
s:124	212.44	111.44

Top Losers

symbol	price	change
s:156	40.34	-116.66
s:131	71.18	-77.82
s:108	73.86	-76.14
s:129	97.16	-75.84
s:140	126.76	-72.24

3. When the portfolio overview for the current user is shown, the application has fetched the details from the back-end database. At this point you have touched all the components that are used t facilitate the Trade application.

   You will not be using this browser interface to the application anymore, so if you want you can close the browser tab.

You have now verified that the application infrastructure is working as expected. Next you must start a Trade Client program that will automatically generate transactions in the application environment.

Exercise 2: Generating trade transactions

In order to be able to discover explicit relationships between the application components traffic must be flowing when the discovery takes place. To generate these transactions automatically, you use a TradeClient program which does not do anything except performing automated buying and selling transactions.

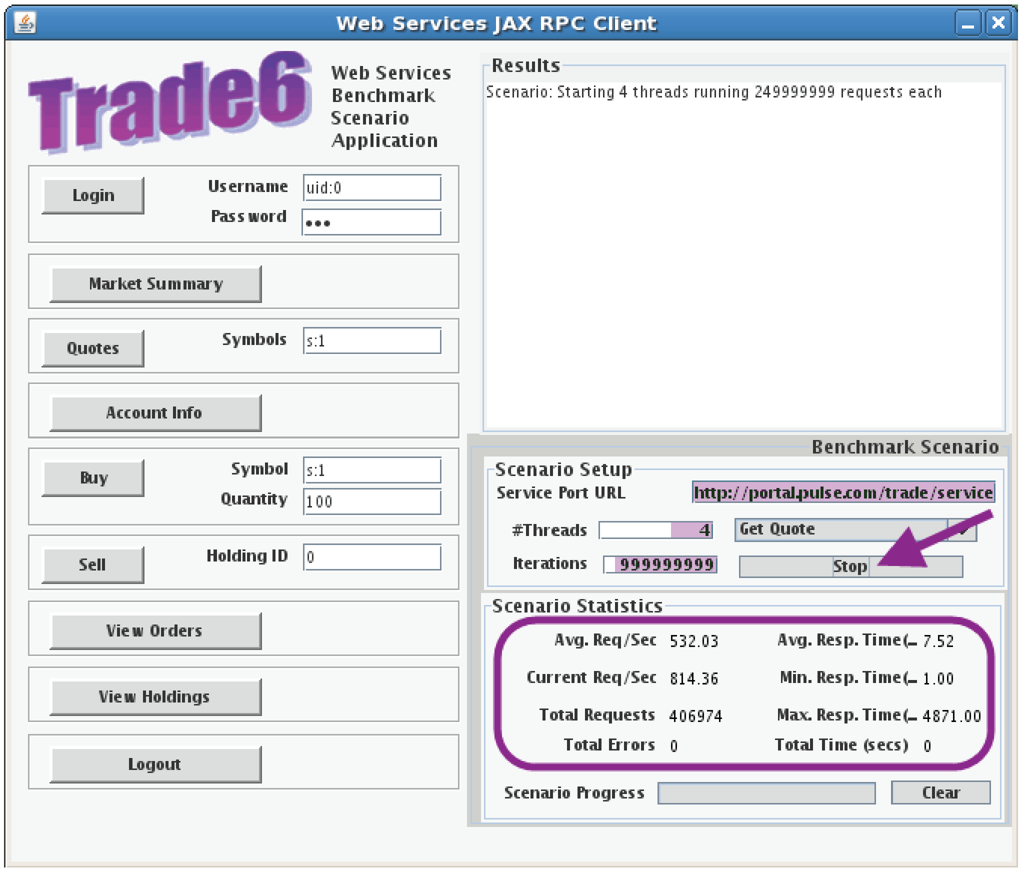
To start the trade client application, and force it to generate transaction requests that will be processed by the application servers, complete these steps:

1. From the desktop of the taddmPRI system, click TradeClient icon to the open the trade client application. After a short while the *WebServices JAX RPC Client* application appears.

2. To start the generation of transactions, you must specify the Service port URL, the number of threads, and the number of iterations. Provide the following information:

Service Port URL:	`http://portal.pulse.com/trade/services/TradeWSServices?wsdl`
#Threads:	`4`
#Iterations:	`999999999 (nine times the digit 9)`

When you are ready to start the application click **Start Scenario**.



Results

Scenario: Starting 4 threads running 249999999 requests each

Username	uid:0
Password	•••
Symbols	s:1
Symbol	s:1
Quantity	100
Holding ID	0

Benchmark Scenario

Scenario Setup

Service Port URL	http://portal.pulse.com/trade/service
#Threads	4
Iterations	999999999

Scenario Statistics

Avg. Req/Sec	532.03	Avg. Resp. Time(…	7.52
Current Req/Sec	814.36	Min. Resp. Time(…	1.00
Total Requests	406974	Max. Resp. Time(…	4871.00
Total Errors	0	Total Time (secs)	0

Scenario Progress

Notice that shortly after you have started the scenario, the values in the Scenario Statistics section will be populated. This indicates that the transactions are being processed, and thus proves that your application server components are communicating with one another.

Now that you have something meaningful to discover, you can start looking at TADDM.

3 Discovering the business application components

To be able to map discrete components to business applications the components must be discovered, so they are represented in the TADDM database. This implies, that you need to plan your discovery to ensure that the resources, you intend to use as *marker modules*, are discovered. Marker modules are those specific resources that you use as the key identifying resources to determine which resources are related to a specific business application.

As you know, discovery of the various component types is performed by TADDM sensors, some of which can be configured to gather more or less information. In addition, it is likely that you want to use resources that are not discovered by default to identify your application components. Such resources typically manifest themselves physically as files, or specific content within a file. However, often these files represent resource types known to TADDM as J2EEModules, J2EEApplications, databases and such. For example, if you want to use a specific link embedded in a website to identify an application you need to discover the html file that implements the link. Once the file has been discovered, you can reference the owning web server by searching for files that contain the link, and use the implicit relationships to identify the web server that owns the file.

In order to discover as many resources as possible, you would enable all the sensors in a copy of default the Level 3 Discovery default discovery profile, and configure the configurable sensors, such as the WebSphereCellSensor, to discover the maximum level of details. In addition, you should take a critical look at your key identifier components and ensure that you create and extend custom server templates to discover the resources you expect to find in the TADDM database. Furthermore, you must provide the credentials necessary access your resources, and optionally update platform properties to support your environment.

Exercise 3: Loading discovery definitions

In order to save you some time, all the definitions you need to discover the application components in the exercise environment have been defined in advance, and can be imported into your TADDM environment using two simple commands. Complete these steps to import the needed definitions:

1. All the definitions you are about the import must be loaded into the TADDM Discovery Server (taddmDIS). To prepare for issuing the import commands, open a terminal window on the taddmDIS system, and navigate to the /opt/IBM/taddm/dist/bin directory. Enter the following commands:

```
su - taddmusr

cd $COLLATION_HOME/bin
```

2. To load the access control list entries that are required to access the systems in the exercise environment as well as the application server components, issue the following command:

```
./authconfig.sh -u administrator -p collation -m -f
/labfiles/mapping/acl --f /labfiles/mapping/TADDMSec.properties

(all on a single line)
```

When the command completes, you will have ACL entries necessary to access the Computer Systems, DB2 databases, LDAP, and SNMP information n the exercise environment. The WebSphere environments in the exercise environment have not been configured to enforce administrative security, so special WebSphere related credentials are not necessary.

3. To load the scopes, discovery profiles, and custom server templates to be used for the discovery, issue this command:

```
./datamover.sh -u administrator -p collation -a import -t all
-f /labfiles/discoveryDefinitions

(all on a single line)
```

By now, you should have at least three scopes, three custom discovery profiles, and a set of custom server templates that has been tailored to discover the exercise environment.

If you want to check the details, use the Discovery Management Console to view the definitions. The main modifications that have been applied, compared to standard settings, are:

Custom Server Templates:

WebSphereCommunityEditionServer	Added to discover java processes listening on port 1050. Discovers files that end with `.class` as software modules.
Derby Server	Added to discover java processes that used the `derbynet.jar` class listening to ports greater than 40000.
CollationProcesses	Enabled.
HTTP Server	Collects configuration files with content from `/var/www/htdocs/*htm*.`

Scopes:

taddmDIS	Includes the taddmDIS.tivoli.edu system.
taddmPRI	Includes the taddmPRI.tivoli.edu system.

Discovery profiles:

Pulse Level 3	Relevant sensors enabled. WebSphereJDBCSensor enabled. WebSphereCellSensor configured to perform deep discovery**.**

For the exercise environment these modifications were necessary in order to be able to use a link in an html file, a Web Service, an ear file, as the marker modules. As stated earlier, the marker modules must be discovered in order to use them as anchor points for business application mapping.

Exercise 4: Launch a discovery

Now that all the necessary definitions are in place you can launch a discovery. To discover all the low-level details you are looking for in order to provide the information needed for business application mapping, you must use the Pulse Level 3 discovery profile and execute it against both the taddmDIS and taddmPRI scopes.

You can launch the discovery from the Discovery Management Console, or from the terminal window you opened on the taddmDIS system. To start the discovery from the terminal window, issue the following commands:

```
cd /opt/IBM/taddm/dist/support/bin

./api.sh -u administrator -p collation discover start --profile
"Pulse Level 3" taddmDIS taddmPRI
```

Once the discovery is launched, you can follow the progress in the Discovery Management Console. The deep discovery of the WebSphere resources in the exercise environment will take 10-15 minutes to complete. While you wait for the discovery to complete, continue working through the next (optional) exercises which provide information about how data are organized in the TADDM database, and how you can use the MQL query language to access it. Understanding these two topics is critical when if you want to create dynamic business applications definitions.

Note: If you look at the progress or status of the discovery in the Discovery Management Console, you may notice that warnings have been issued for a number of WebSphere related sensor instances. This is perfectly normal, and should not cause any concern.

If you are already familiar with the CDM and the MQL language, you should jump to the section of your choice and read the introduction. Based on your experience consider the following options:

- If you are familiar with the basic TADDM functionality for static and rule-based business application definition, you can jump directly to *6 Business application definition using relationships* on page 53.
- If your need an update on how to use rules in the TADDM Grouping Composer, you should consider starting at *5 Dynamically defined business applications* on page 41.
- If you have no experience in defining business applications in TADDM, start at *4 Static business application definition* on page 22.

No matter which path you decide for, you should complete *Exercise 6: Analyzing your discovery results* on page 18 when the discovery has completed.

Exercise 5: (optional) Understanding how data is organized

If you are not intimately familiar with the way configuration information is organized in accordance with the Common Data Model you should read the information provided in *9.7 Understanding how configuration information is organized* on page 107. This section provides a brief introduction to the Common Data Model and will help you better understand now you can use attributes and implicit relationships to locate dependencies between resources.

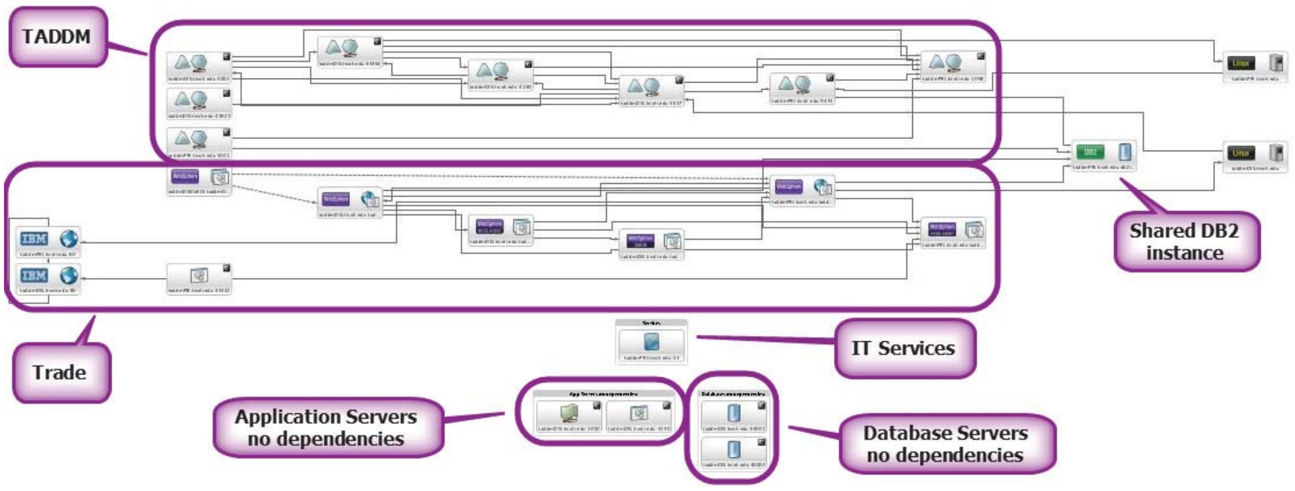
The main information in the "Understanding how configuration information is organized" section can be summarized to this:

All resource types defined by the Common Data Model are characterized by:

- A number of attributes that contains the configuration information for the resource. Attribute values can be:
- A specific value
    - A pointer to another configuration item to which the resource has an implicit relationship
    - An array of pointers
    - Relationships that specify how the resource can be related to other resources. Implicit relationships are enforced by the CDM, and explicit relationships are used to associate resources that otherwise have no logical relationship that can be modeled.
- Naming rules, combinations of attributes that are used to uniquely identify individual resources. In many instances, pointer attributes are used in naming rules.

Exercise 6: Analyzing your discovery results

To see which resources were discovered by TADDM, you should take a look at the Application Infrastructure for the entire exercise environment. To launch the application infrastructure topology activate the Data Management Portal browser window, and navigate to **Topology** > **Application Infrastructure**. The topology should be similar to this:


TADDM
Shared DB2 instance
Trade
IT Services
Application Servers no dependencies
Database Servers no dependencies

Note:	Normally the Application Infrastructure feature is disabled. In most cases the shear number of resources makes the topology almost impossible to read. However, in the tiny exercise environment, it provides a nice overview.

If your topology does not look similar to the one shown above, there is a chance that the background topologyBuilder process has not yet been executed. You can force the execution of all the background tasks by issuing the following command from a terminal window on the taddmPRI system:

```
/opt/IBM/taddm/dist/support/bin/runtopobuild.sh -b -w
```

When the command terminates, the topology information in the TADDM database has been rebuilt, and when you refresh the topology shown in the Data Management Portal, you see the updated topology,

The Application Infrastructure reveals that the majority of the discovered application server components are somehow related to other application server components. These connections are the result of live sessions between the two components that participate in a communication. If you look closely, you can see that the application servers in the exercise environment can be grouped into five major groups:

TADDM:	A number of interconnected CollationProcesses application servers. These were discovered by the CollationProcesses custom server template, and represent the processes related to the live TADDM environment.
Trade:	A group of interconnected web, application, and database servers. Surprisingly, the topology has a close resemblance to the diagram outlining the Trade application architecture.
IT Services:	Shared IT services such as LDAP, and DNS. Connections to these services are not shown. Nearly all the components in the infrastructure leverage these services, so visualizing all the connections would render the topology unreadable.
Application Servers - no Dependencies :	At the bottom of the topology you see all the discovered application servers for which no transactional, or explicit, dependencies were discovered. In this group you see the WebSphereCommunityEditionServer and any additional servers for which no explicit relationships exist.
Database Servers - no Dependencies :	At the bottom of the topology you see all the discovered database servers for which no transactional, or explicit, dependencies were discovered. In this group you see the two Derby servers.

The topology hopefully reveals the importance of performing discovery while the systems are performing production work. If no business transactions had been active during discovery all you would have seen would be a number of resources similar to the ones in the No Dependencies group.

3.1 Looking at low-level resources

The topology viewer in TADDM only displays top-level resources. Top-level resources are defined in the Common Data Model as *resources that are generally displayed independently on a user interface*. For example, an operating system is normally only shown in the context of a computer system, and therefore is not considered to be top-level, whereas the computer system itself is generally considered to be a top-level object. Application servers, clusters, virtualization platforms, storage subsystems, and network devices are examples of other top-level resource types.

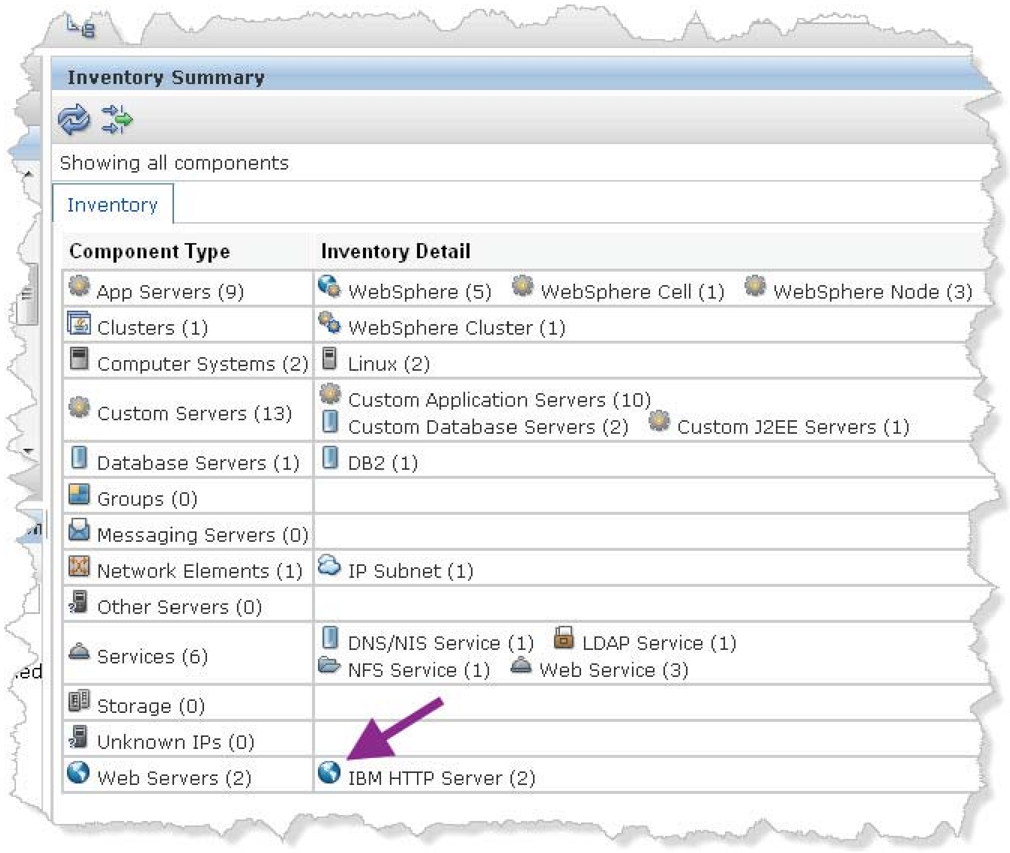
This implies that in the topology you cannot directly see the low-level resources that are owned by the top-level resources. However, the Details information for any resource contains information about the low-level resources that are owned by the top-level resource.

HTTP Server html files

You may recall that the HTTP Server custom server template in the exercise environment has been extended to discover all the *.htm* files in the /var/www/htdocs. Both the directory name and the name of the files are hardcoded in the custom server template, but can easily be made dynamic by augmenting the template with Jython extension that reads the value for the DocumentRoot property, and locates the files at runtime.

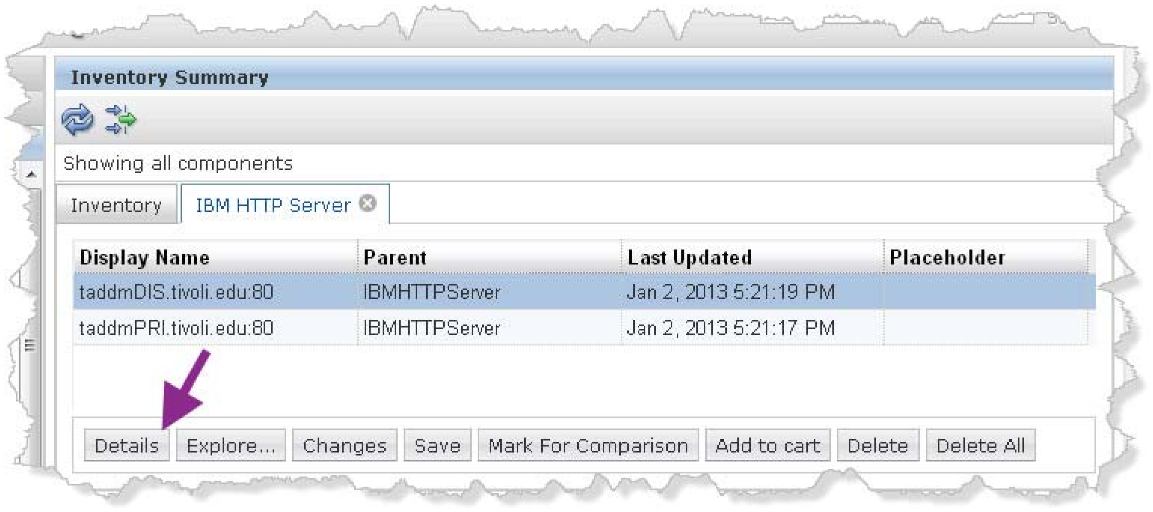
To see which configuration files were discovered for one of the HTTP Server instances in the environment, complete these steps:

1. From the Data Management Portal, open the Inventory Summary by navigating to **Analytics** > **Inventory Summary**.
2. At the bottom of the list you see that the Web Servers section contains 2 (two) IBM HTTP Server instances. To see the two instances, click IBM HTTP Server.



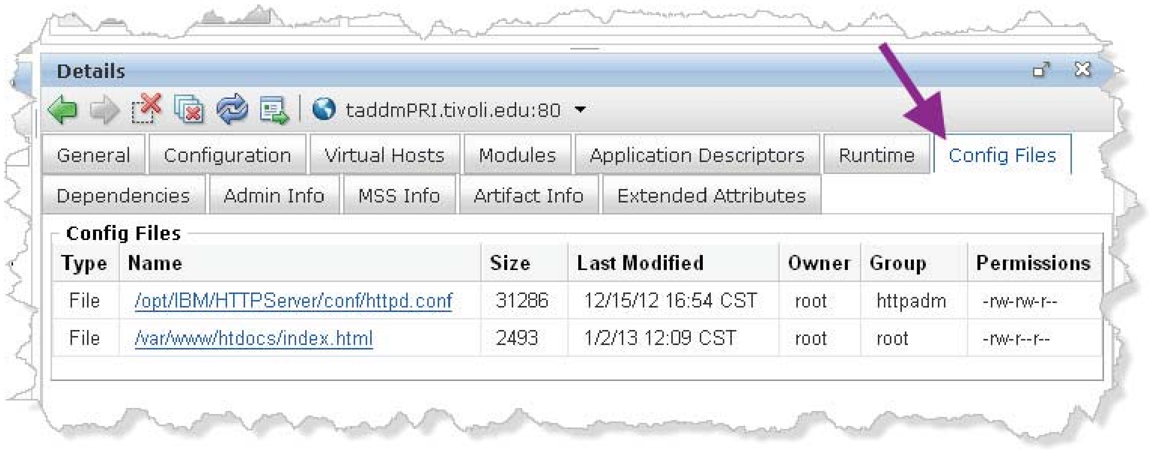
Component Type	Inventory Detail
App Servers (9)	WebSphere (5) WebSphere Cell (1) WebSphere Node (3)
Clusters (1)	WebSphere Cluster (1)
Computer Systems (2)	Linux (2)
Custom Servers (13)	Custom Application Servers (10) Custom Database Servers (2) Custom J2EE Servers (1)
Database Servers (1)	DB2 (1)
Groups (0)	
Messaging Servers (0)	
Network Elements (1)	IP Subnet (1)
Other Servers (0)	
Services (6)	DNS/NIS Service (1) LDAP Service (1) NFS Service (1) Web Service (3)
Storage (0)	
Unknown IPs (0)	
Web Servers (2)	IBM HTTP Server (2)

3. Select one of the two servers, and click Details to load the information about the IBM HTTP Server instance and its related low-level resources into the Details pane.



Display Name	Parent	Last Updated	Placeholder
taddmDIS.tivoli.edu:80	IBMHTTPServer	Jan 2, 2013 5:21:19 PM	
taddmPRI.tivoli.edu:80	IBMHTTPServer	Jan 2, 2013 5:21:17 PM	

4. In the Details pane, you can inspect the information in the various sections. However, the information about the discovered files is located in the Config Files tab. Open the tab, and verify that you find the `http.conf` file, as well as a single html file named `index.html`. If you want to you can see the content of any one of the files by clicking the appropriate link.



Type	Name	Size	Last Modified	Owner	Group	Permissions
File	/opt/IBM/HTTPServer/conf/httpd.conf	31286	12/15/12 16:54 CST	root	httpadm	-rw-rw-r--
File	/var/www/htdocs/index.html	2493	1/2/13 12:09 CST	root	root	-rw-r--r--

The reason why it does not matter which IBM HTTP Server you look at is, that on the taddmDIS system, the `/var/www/htdocs` directory is NFS-mounted from the taddmPRI system, so the two instances always use the same file structure.

Notice that besides the configuration files that are stored in the database because it is embedded in the IBM HTTP Server custom server sensor, you can see the configuration of the servers, the virtual host definitions, which modules they are using, and of course all the administrative information related to Dependencies, MSS, and Admin Info.

4 Static business application definition

The purpose of defining business application is to provide a logical grouping of your infrastructure components. This grouping represents only the components that contribute to the provisioning of the IT systems that support a specific business task. Business applications can be grouped into business services that represent the resources supporting an entire business process.

By defining these groups you provide information that is helpful to support personnel such as incident, and problem managers. When working on an issue related to for example application slowdown, they can immediately see which components participate in the provisioning of the application. In addition, if the components are discovered on a regular basis, the support personnel can immediately see if any components supporting the application with degraded performance have been recently changed. Under the covers, discovered changes are automatically propagated to the business application. This allows you to focus on the changes to business applications, and when they occur, you can use the change history function in TADDM to identify the root cause of the change.

The business application and business service information is also extremely helpful for your Configuration and Change Management processes. When planning changes to existing components, or the deployment of new ones, you can immediately see which business processes and tasks will be affected of the planned changes, and this helps you to schedule the implementation for a time where the impact to the business is minimal.

Within a business application, components are always organized in functional groups. Functional groups represent groups of resources that perform the same task. This extra layer in the model is necessary in order to be able to assign the correct operational status to a clustered resource. Depending on the status of the members of a functional group, the status of the functional group itself (the cluster) may be affected. If you decide to add for example multiple J2EE servers to an application they may - or may not - be members of the same functional group. Logically they should only be members of the same functional group if they perform the exact same tasks (have the same applications installed and enabled) but you often see that the functional groups, incorrectly are used to group together resources of the same type.

For reporting purposes, many customers prefer to add computer system to the business applications, even though TADDM itself visualize the physical and OS layers when you view the physical topology. In this situation you often see that the same type of servers, based on OS, are assigned to the same functional group even though they perform very different functions. This is just mentioned here as a precaution, so you take the impacts of your decisions into consideration when you start populating your own business application and their functional groups.

When defining business applications you can associate resources with the business application statically, dynamically, or as a combination of the two. No matter which method you use, Static association implies that you, based on your knowledge about the IT infrastructure, assign one or more specific components to an application. Once these components are assigned, they will (in most cases) remain members of the business application until you, manually, remove them, or delete them from the TADDM database.

You would use this option to associate resources that are not likely to change to your business applications. LDAP, DHCP and DNS servers are typical examples of resources that may be assigned this way.

Static association can also be used in situations where you have not been able to discover transactional (explicit) relationships between resources, or you want to map otherwise unrelated resources to the same business application.

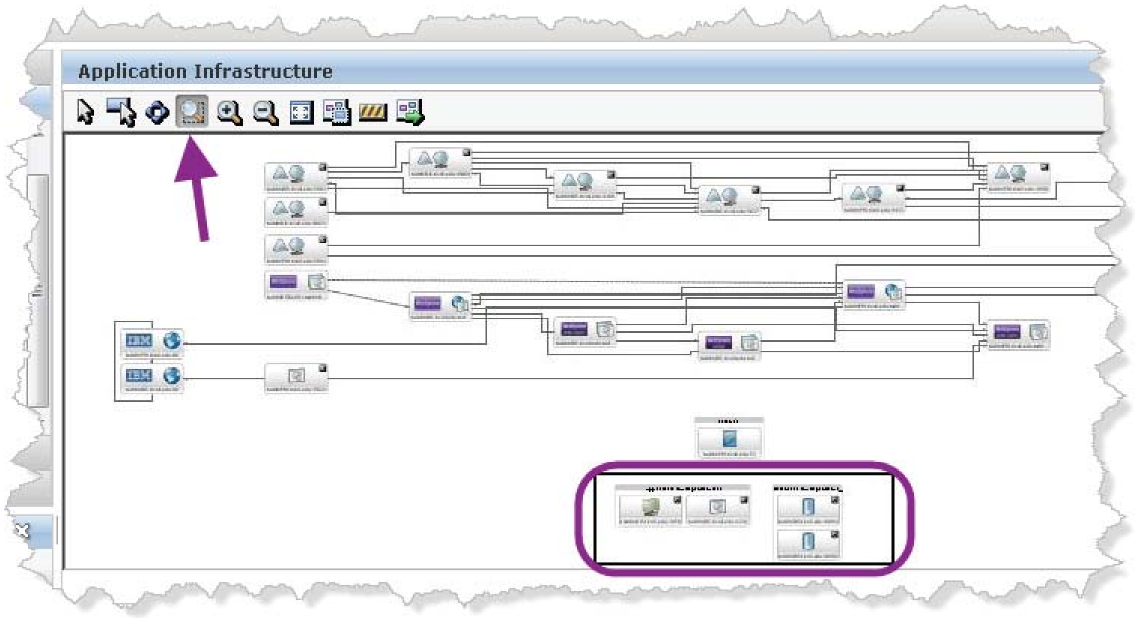
In this exercise, you use the static resource association to create a business application definition for the DayTrader application that is hosted on the WebSphereCommunityEditionServer. During discovery there were no traffic to or from this server, so you can only rely on your knowledge about the application infrastructure, or consult the solution architect. As a matter of fact, the DayTrader application uses both of the Derby servers hosted on the same system as the WebSphereCommunityEdition server. Since no dependencies were discovered, you will add dependencies manually in order to record the fact that the DayTrader application actually relies on the two Derby servers.

Exercise 7: Creating a simple, static business applications

In TADDM v7.2.1 you can create business application definitions in multiple ways. Normally you launch the Grouping Composer tool from the toolbar, but in this exercise you use the method of selecting a number of components from the topology view and create the business application based on your selection.

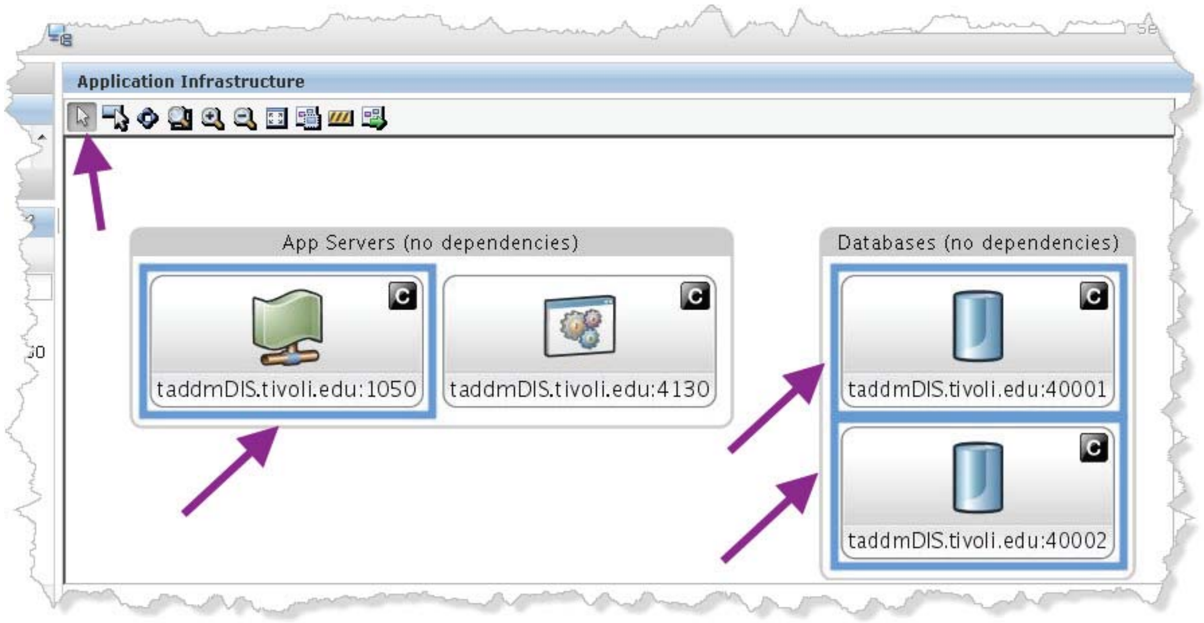
To create your first, static, business application, complete these steps:

1. Ensure that you are looking at the Application Infrastructure. If you have navigated away from it, you can open it again by selecting Topology > Application Infrastructure from the toolbar.
2. To select the three resources you want to add to the new business application, use the Zoom in Rectangle tool (🔍) to enlarge the group of applications with no dependencies.



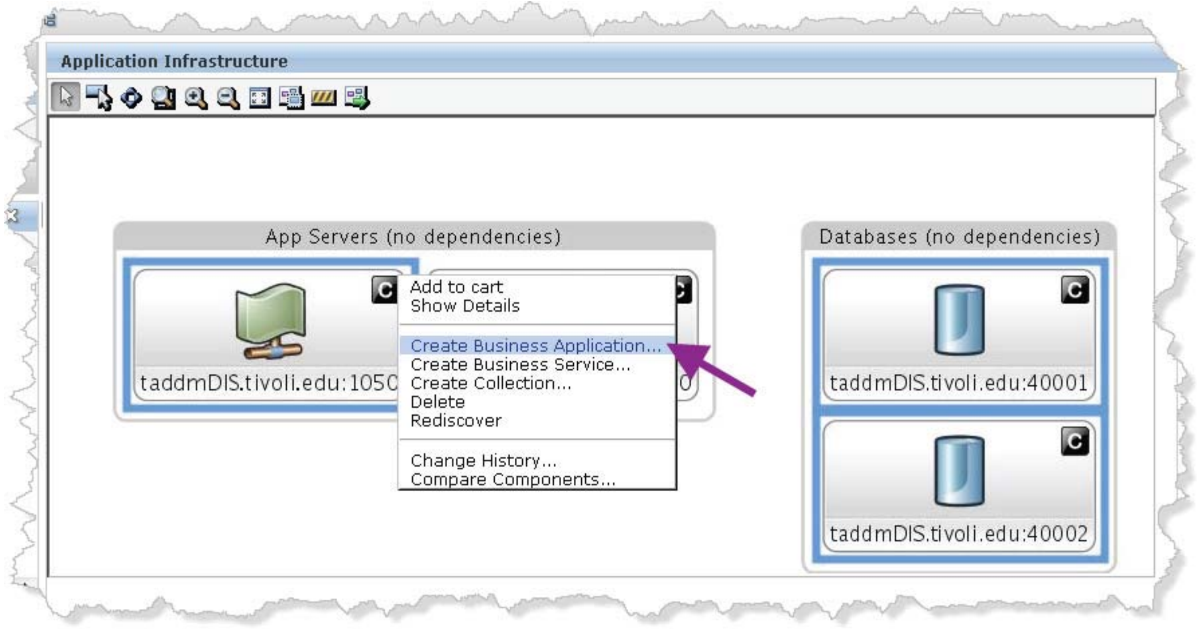
When you see the subset of the topology you selected, you can actually read the label of each component. Notice how the components are named. The naming rules for application servers state that an application server always is named by the PrimarySAP attribute, which contains the concatenation of the hostname of the system hosting the application server, and the lowest port number the server listens to. In this example all three application servers are hosted on the same system, but they listen to different ports.

3. When you see only the resources you want, choose the select tool (🖱) and select the WebSphereCommunityEditionServer (listening on port 1050), and the two Derby Servers (listening on ports 40001, and 40002) by clicking each one of them.


Application Infrastructure
App Servers (no dependencies)
taddmDIS.tivoli.edu:1050
taddmDIS.tivoli.edu:4130
Databases (no dependencies)
taddmDIS.tivoli.edu:40001
taddmDIS.tivoli.edu:40002

When you have selected any resource, it will be highlighted with a surrounding blue frame.

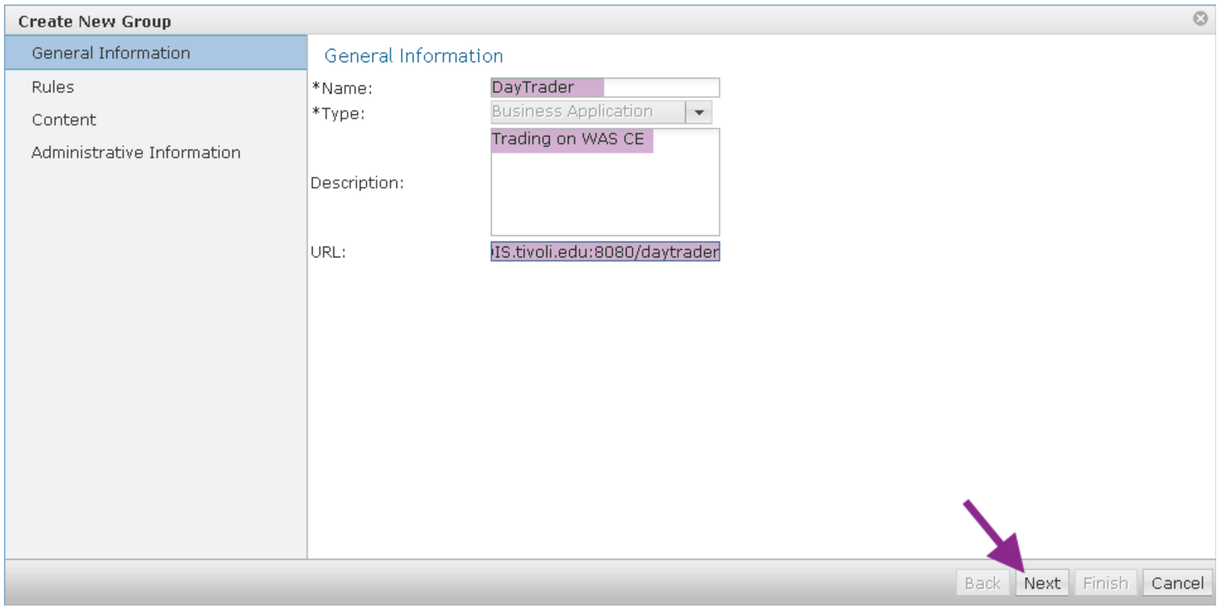
4. To perform an action on all three selected components, for example creating a business application, right-click on any one of them and select **Create Business Application...**.


Application Infrastructure
App Servers (no dependencies)
taddmDIS.tivoli.edu:1050
Add to cart
Show Details
Create Business Application...
Create Business Service...
Create Collection...
Delete
Rediscover
Change History...
Compare Components...
Databases (no dependencies)
taddmDIS.tivoli.edu:40001
taddmDIS.tivoli.edu:40002

5. The Grouping Composer tool is now automatically started and you must provide basic identification information for the new business application. Supply the following properties:

Name:	`DayTrader`
Description:	`Trading on WAS CE`
URL:	`http://taddmDIS.tivoli.edu:8080/daytrader`

When you are ready click **Next** to progress the definition of the business application.



Create New Group

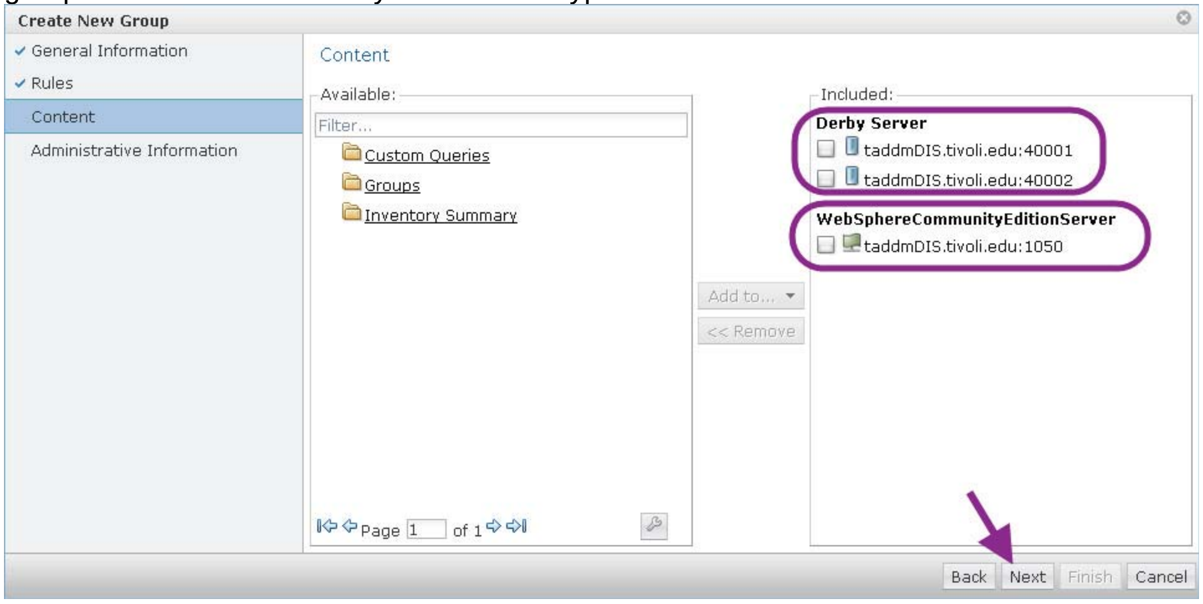
General Information
Rules
Content
Administrative Information

General Information

*Name: DayTrader
*Type: Business Application
Description: Trading on WAS CE
URL: IS.tivoli.edu:8080/daytrader

6. On the next page, you can supply rules that are used to dynamically populate the functional groups of the business application. In this example you are only dealing with static associations, so click **Next** to bypass the rule definitions.

7. In the Content page you see that your components have already been associated with the application - as you expected. Notice that your components are organized into two functional groups that are named after your resource types.



Create New Group

✓ General Information
✓ Rules
Content
Administrative Information

Content

Available:
Filter...
Custom Queries
Groups
Inventory Summary
Page 1 of 1

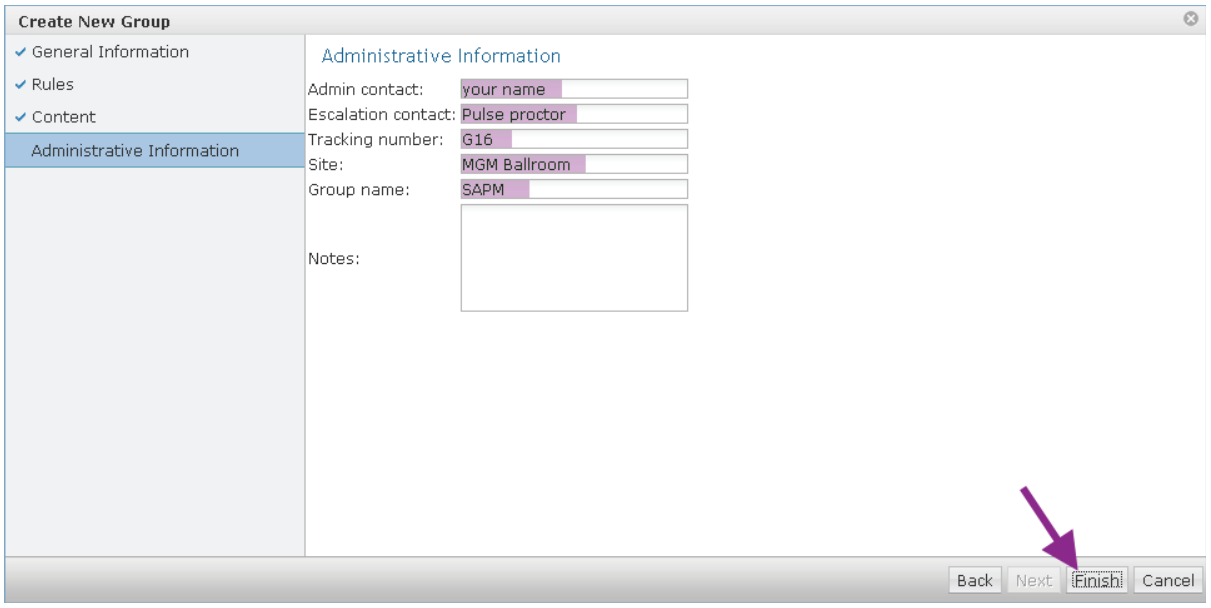
Included:
Derby Server
taddmDIS.tivoli.edu:40001
taddmDIS.tivoli.edu:40002
WebSphereCommunityEditionServer
taddmDIS.tivoli.edu:1050

This is exactly what you expected, so click **Next** to continue.

Notice that if you wanted to add additional components, you can use the Resource Navigator in the middle of the window to identify the components, and use the Add button to associate the selected resources to a functional group.

8. If the final Administrative Information page you can provide information about who is administering the application, and add additional administrative information. Supply the following properties:

Admin contact:	`<your name>`
Escalation contact:	`Pulse proctor`
Tracking number:	`G16`
Site:	`MGM Ballroom`
Group name:	`SAPM`



Administrative Information

Admin contact:	your name
Escalation contact:	Pulse proctor
Tracking number:	G16
Site:	MGM Ballroom
Group name:	SAPM
Notes:	

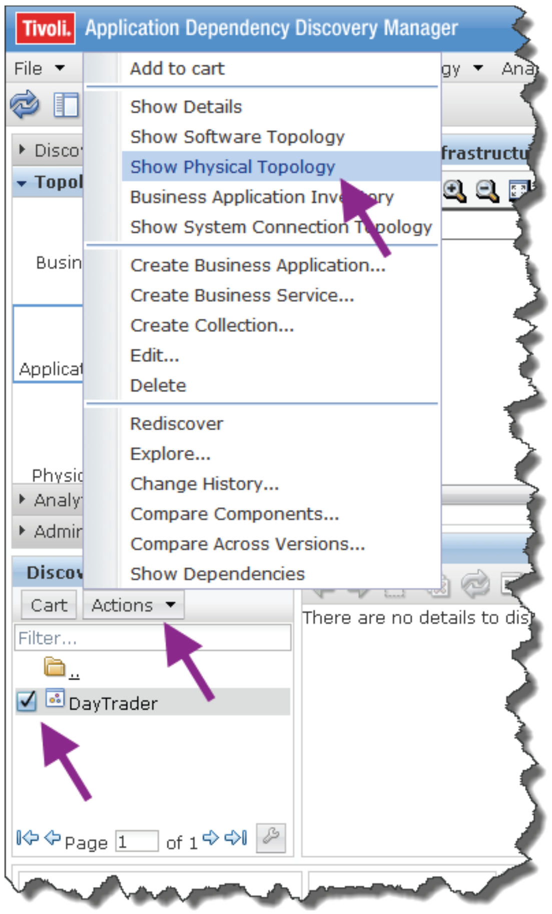
When you are ready click **Finish** to complete the definition of the business application.

Give yourself a pad on the back, you have just created your first business application.

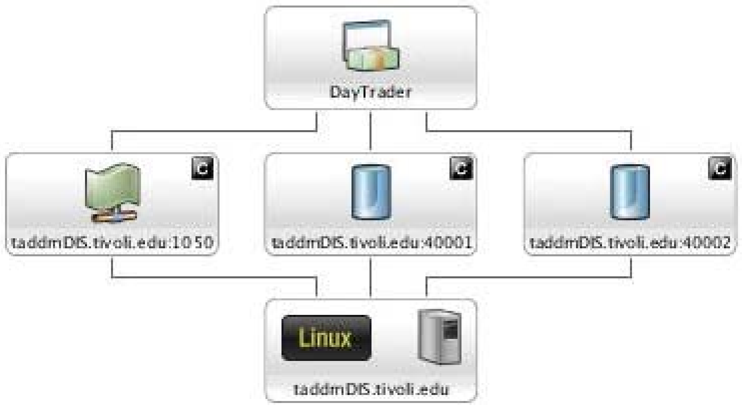
Exercise 8: Viewing the business application topologies

TADDM provides two topology views related to business applications, a physical view and a software view. To see how your business application topologies are represented in these views, complete these steps:

1. From the Resource Navigator in the Discovered Components pane at the lower left of the Data Management Portal, navigate to **Groups** > **Business Applications**. When the Business Application branch has been opened, you see only a single application: DayTrader.
2. To view the physical topology, check the DayTrader application, and select Action > Show Physical Topology.

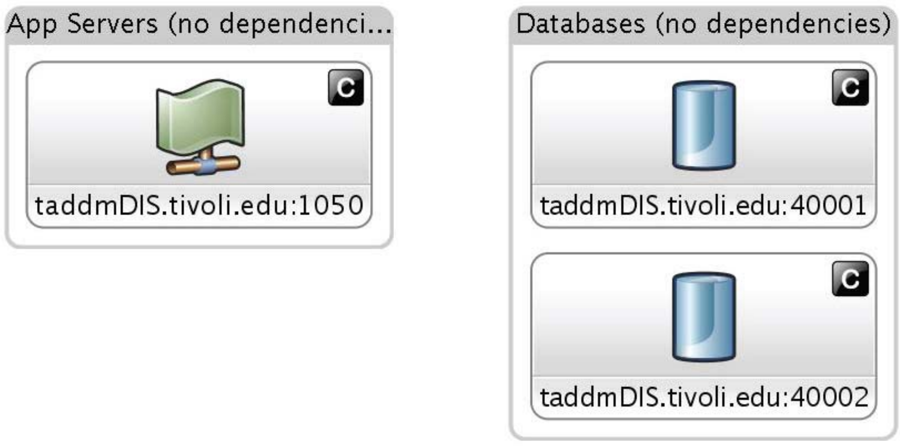


3. In the physical topology you see that all three application server components are hosted on the same Linux system.



The physical topology is built by looking at the members of all of the functional groups that are associated with the DayTrader application, and use the parent attribute for each application server component to find the operating system that hosts the application server.

4. To see the software topology, focus on the Discovered Components pane again, and select **Action** > **Show Software Topology**. This will reveal a topology similar to this:


App Servers (no dependenci...
taddmDIS.tivoli.edu:1050
Databases (no dependencies)
taddmDIS.tivoli.edu:40001
taddmDIS.tivoli.edu:40002

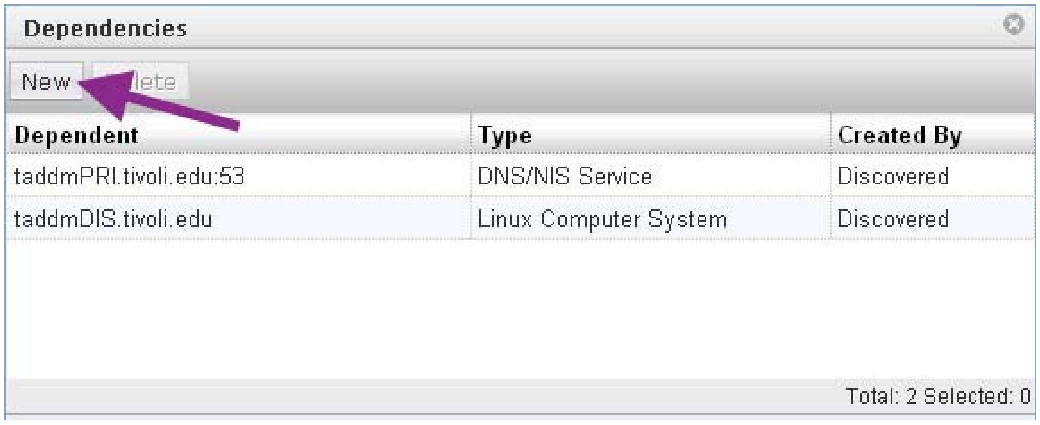
As you expected, you see a topology similar to the subset you selected from the Application Infrastructure, and there are still no dependencies between the components.

Exercise 9: Adding relationships manually

As mentioned a number of times, dependencies between components are created when TADDM discovers communication between the components. In this case, if a session from the process listening on port 1050 (the WebSphereCommunityEdition server) to a server listening to port 40001 or 40002 was discovered, the dependency between the two components would be created.

Assume for a while, that you know that the WebSphereCommunityEditionServer always connect to both Derby databases. To represent this knowledge in the topology, you can manually add relationships to the topology. To create the dependencies between the WebSphereCommunityEditionServer and both of the Derby servers, follow these steps:

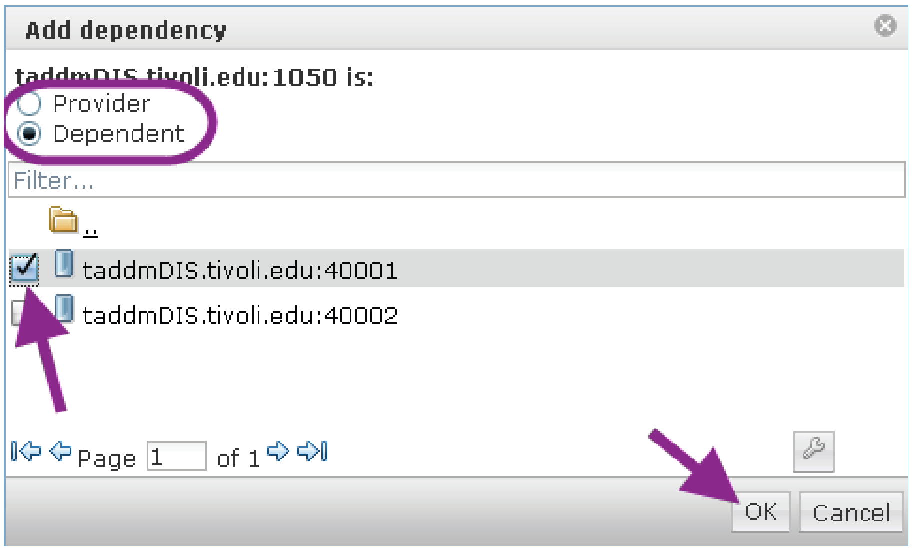
1. Right-click on the WebSphereCommunityEdition server (the one listening on port 1050), and choose **Show Dependencies** from the context menu.
2. When the dependencies window opens, you see that TADDM has discovered that the WebSphereCommunityEditionServer depends on the taddmDIS system. What a surprise.



Dependent	Type	Created By
taddmPRI.tivoli.edu:53	DNS/NIS Service	Discovered
taddmDIS.tivoli.edu	Linux Computer System	Discovered

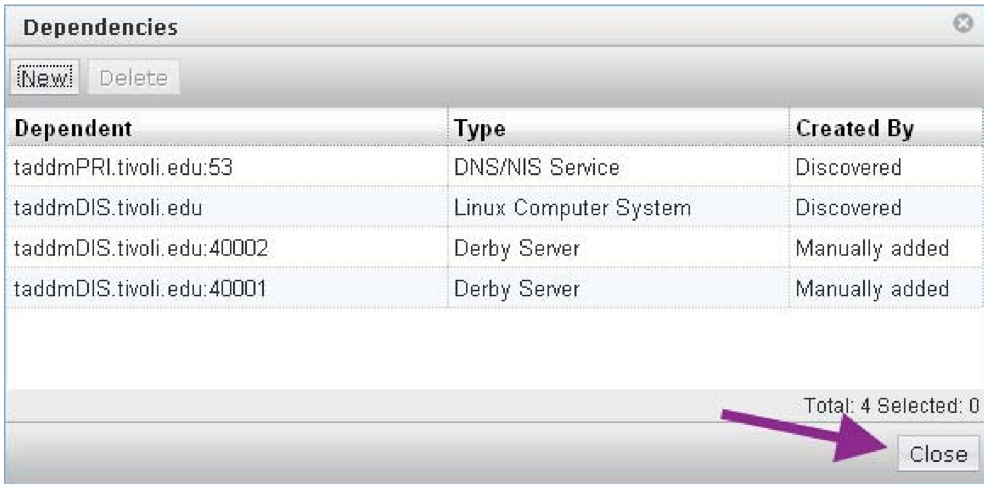
Total: 2 Selected: 0

3. To manually add a dependency, use the following procedure:
   a. Click **New**.
   b. In the Add Dependency window that appears, navigate to **Custom Servers** > **Derby Server**, and check one of the instances you see.



[✓] taddmDIS.tivoli.edu:40001
[ ] taddmDIS.tivoli.edu:40002

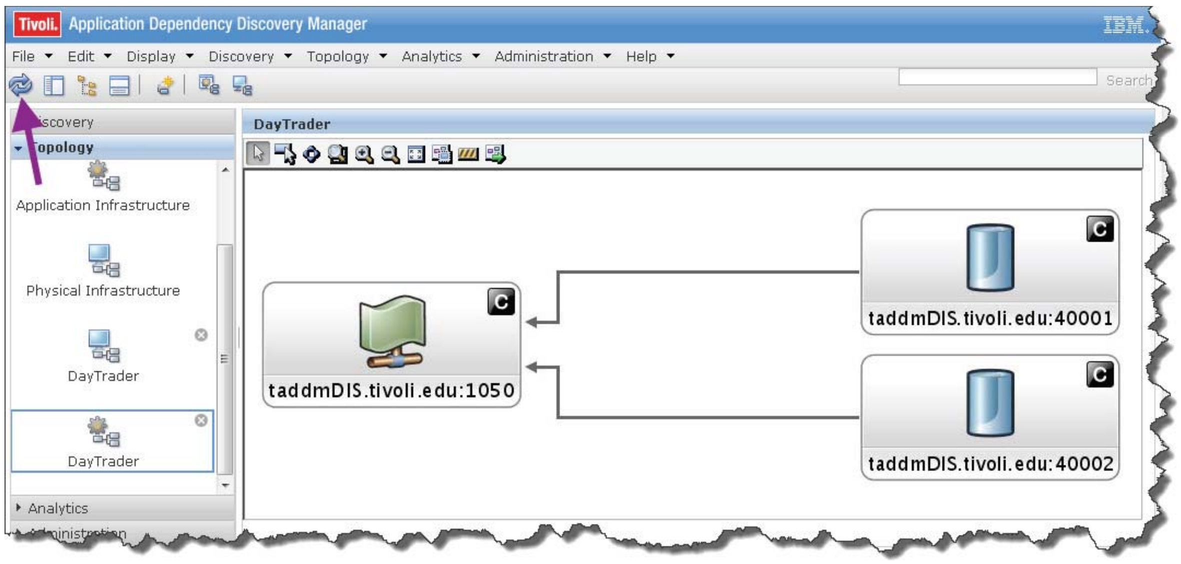
   c. Ensure that the WebSphereCommunityEditionServer (taddmDIS.tivoli.edu:1050) is marked as Dependent of the selected component.
   d. Click **OK** to save the dependency in the TADDM database.
4. Repeat the previous step to mark the WebSphereCommunityEdition server as dependent of the other Derby Server in the exercise environment.
5. When you have created both dependencies, the list of dependencies for the WebSphereCommunityEdtionServer should look like this:



Dependent	Type	Created By
taddmPRI.tivoli.edu:53	DNS/NIS Service	Discovered
taddmDIS.tivoli.edu	Linux Computer System	Discovered
taddmDIS.tivoli.edu:40002	Derby Server	Manually added
taddmDIS.tivoli.edu:40001	Derby Server	Manually added

To dismiss the Dependencies window, click **Close**.

6. Immediately, you do not see any changes to the software topology for the DayTrader application. However, if you refresh the topology view, by clicking the Refresh icon (🔄) to the far left in the Data Management Portal action bar, you will see that the new relationships have affected the way the topology is visualized.


Tivoli. Application Dependency Discovery Manager
IBM.
File Edit Display Discovery Topology Analytics Administration Help
Search
Discovery
Topology
Application Infrastructure
Physical Infrastructure
DayTrader
DayTrader
Analytics
Administration
DayTrader
taddmDIS.tivoli.edu:1050
taddmDIS.tivoli.edu:40001
taddmDIS.tivoli.edu:40002
C

Notice how the arrowheads indicate that the two Derby servers provide services to the WebSphereCommunityEdition server. If the opposite was also the case you would see another line pointing from the WebSphereCommunityEdition server to the Derby servers. Explicit relationships are by nature uni-directional, and all of them are visualized in the topology.

You have completed the creation of your first business application. This application was defined using static association between components and the functional groups in the application, so the topology will not change based on new information that may be discovered. In addition, you added explicit relationships to record dependencies that you know exist, but were not discovered due to lack of transactions during discovery.

Exercise 10: (optional) Understanding your discovery options

If you are not intimately familiar with the way TADDM discovers resources through custom server templates and custom server template extensions, you should read the information provided in *9.1 Discovering detailed information* on page 103. This section provides a brief introduction to how you can extend the discovery to include marker modules that apply to your environment. Hopefully this will help you better understand your options for discovering resources that are not included in the standard discovery process.

The main information in the *Discovering detailed information* section can be summarized to this:

- TADDM provides many built-in discovery sensors that discover very detailed information about standard technologies such as physical and virtual hardware, networking devices, operating systems, application server platforms such as WebSphere, WebLogic, Siebel, SAP, and the supporting database-, web-, and messing servers.

- You can implement discovery of your specific resources through custom server sensors. In a custom server sensor you can instruct TADDM to discover certain files on the target system and register these files as either configuration files or software modules.
- Custom server templates can be extended by providing control files that populate the values of specific attributes with the data returned by scripts that are executed on the target system.
- As part of the extensions, custom scripts can be developed to perform extensive analysis on the target system and create multiple resources in the TADDM database. For example, if you want to register individual databases hosted on a Derby server, your custom script can create the databases and relationships to the Derby server as part of the discovery of the Derby server itself.
- You can augment the information in the TADDM database by importing information that has been extracted from other solutions or custom data by a discovery library adapter. This provides information that TADDM cannot discover, and helps paint the complete picture of your IT infrastructure.

Exercise 11: Identifying discovered relationships

As you understand, the relationships that can be discovered by TADDM is a key piece of information that allows you to relate components across operating platforms. In the Data Management Portal these relationships are shown in the business application topology views, but these are not available until you have created the business application.

In the previous exercises you saw the relationships in the Application Infrastructure map, but, as mentioned, this function is normally disabled because it will contain too much information to be useful. So how do you visualize the relationships before defining business applications?

The Data Management Portal has an explore function with which you can interactively build a topology map of your application by traversing the relationships between top-level components. The explore function uses both explicit (transactional) relationships an implicit (CDM enforced) relationships. These relationships that are stored in the TADDM database. This function is very helpful when you want to investigate how your components relate to one another. The explore function can be used to interactively create a map of your business application without having to define the business application itself.

In this exercise you use the explorer function to reveal the topology of the TADDM infrastructure, so you can identify the components that it relies on. However, before you start, you may spend a few minutes figuring out exactly what the marker modules for the TADDM solution are.

4.1 TADDM marker modules

When you looked at the Application Infrastructure earlier, you may have noticed that TADDM has discovered multiple CollationProcesses servers. These are all related to the TADDM solution itself, so it would be great if they can be grouped into a single group that represents all the resources that support the TADDM application. TADDM allows you to view business application topologies, similar to the Application Infrastructure. However, since a business application only contains a subset of the infrastructure these business application topologies are readable, and a very useful source of information when planning changes or debugging problems.

Application systems like TADDM provide a special challenge. Under the covers, the TADDM servers launch several processes that are responsible for performing specific functions. Each of these functions may have specific operational characteristics, and may interact with other processes that are not directly related to TADDM. For example, the TADDM storage server is made up of four distinct processes (as you see when you open the TADDM Portal). One of these processes is responsible for accessing the database, one handles the API and user traffic, one manages cached information, and one coordinates all the work, and manages authorizations and access control. For the discovery server, you can make a similar mapping between distinct functionality and processes. However, all the TADDM-related processes share one common characteristic: The arguments parsed to the processes contain the string *COLLATION*.

In addition, TADDM also uses a database. Naturally, this database and the database management process through which it is accessed, should also be part of the TADDM business application. In a production environment, it is not very likely that the name of the database changes over time, so you can use the name of the database as it is used in the TADDM configuration file `collation.properties` which is used to control the configuration of the TADDM processes.

As you probably have already guessed, processes for which the string *COLLATION* exists in the arguments that are parsed to the process is the marker module for the application server components to include in a business application that represents the TADDM solution. Luckily, a custom server template that identifies all these processes is provided in TADDM. The name of that custom server template is CollationProcesses, and in the exercise environment it is enabled. That is why you have already discovered a number of CollationProcesses.

Note: If you do not know which attributes to use to define custom server templates that can be used to identify special process types, you can use the Unknown Servers function to see which server processes are running on a particular computer system, and use that information to determine which custom server templates you need.

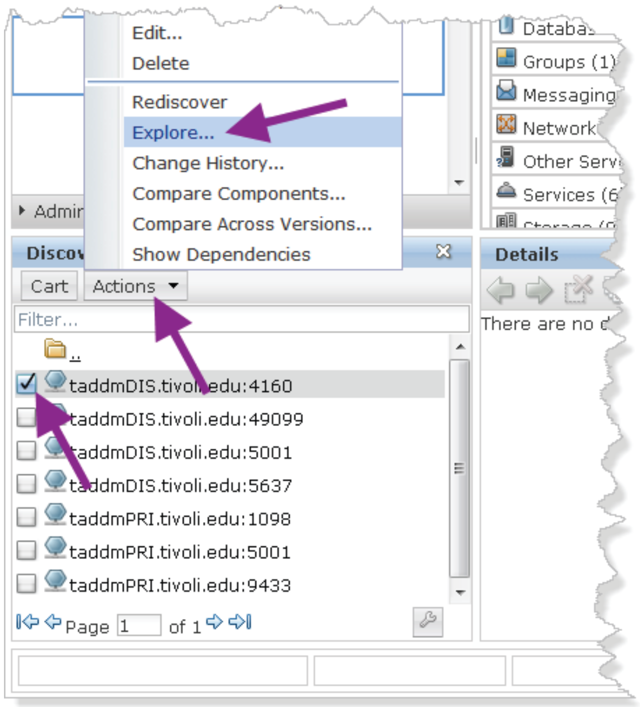
Exercise 12: Exploring the TADDM infrastructure

The explore function in TADDM is used to identify the components at the other end of any relationships associated the selected component. To interactively create a topology map, you start by selecting a component, and identify its related components based on the relationship types. Once the initial topology map is created, you can select components from the map and use the *extend* function to add additional components to the topology. The process is much like peeling an onion, revealing more and more details as you progress.

Now, to figure out which components are used by the TADDM solution as it is implemented in the exercise environment, complete the following steps in the Data Management Portal:

1. From the Discovered Components pane (at the lower left) navigate to **Inventory Summary** > **App Servers** > **Custom Application Servers** > **CollationProcesses** and verify that TADDM has discovered seven instances of CollationProcesses servers.

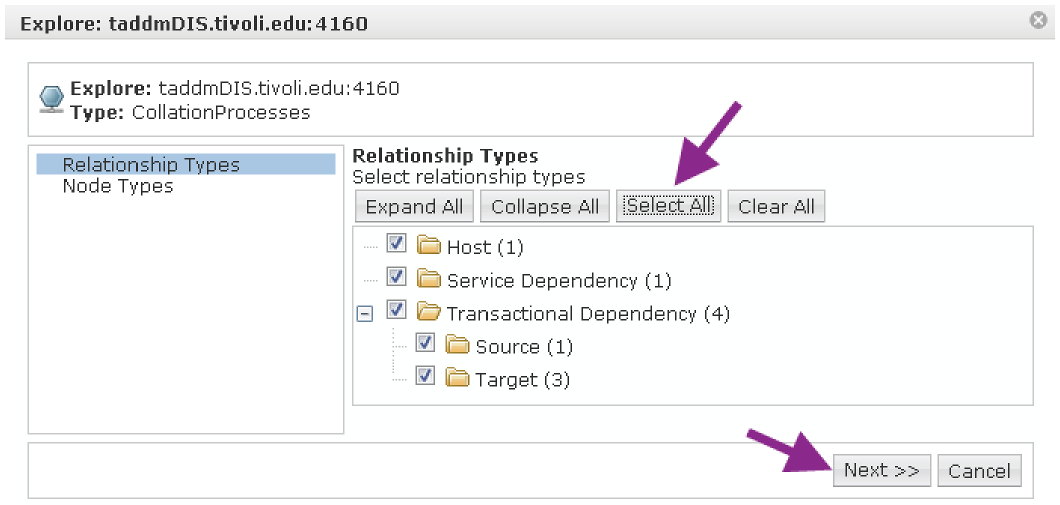
2. Check any CollationProcesses instance, and select **Actions** > **Explore...**.



After a short while, the Explore wizard is displayed.

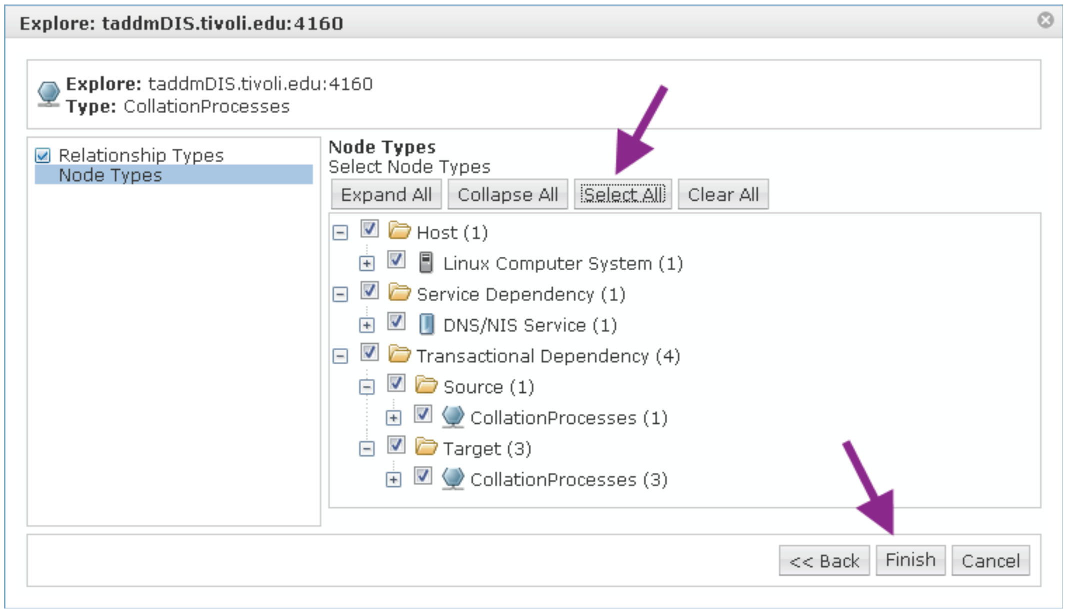
3. To select the components to include in the topology map, complete these steps:

   a. On the Relationship Types page of the explore wizard you can the types of relationships for which you want to include the related components. For this exercise click select All to include all relationship types that apply for the selected component.



Notice that there are multiple types of relationships. The discovered relationships are typically listed as Service or Transactional Dependencies. Click **Next** to continue.

b. The Node Types page of the Explore wizard shows all the components that are related to the component you are exploring, through the types of relationships you selected in the Relationship Types page. In this case, since you selected all relationship types, you see all the related components.



Explore: taddmDIS.tivoli.edu:4160

Explore: taddmDIS.tivoli.edu:4160
Type: CollationProcesses

- ✓ Relationship Types
- Node Types

Node Types
Select Node Types

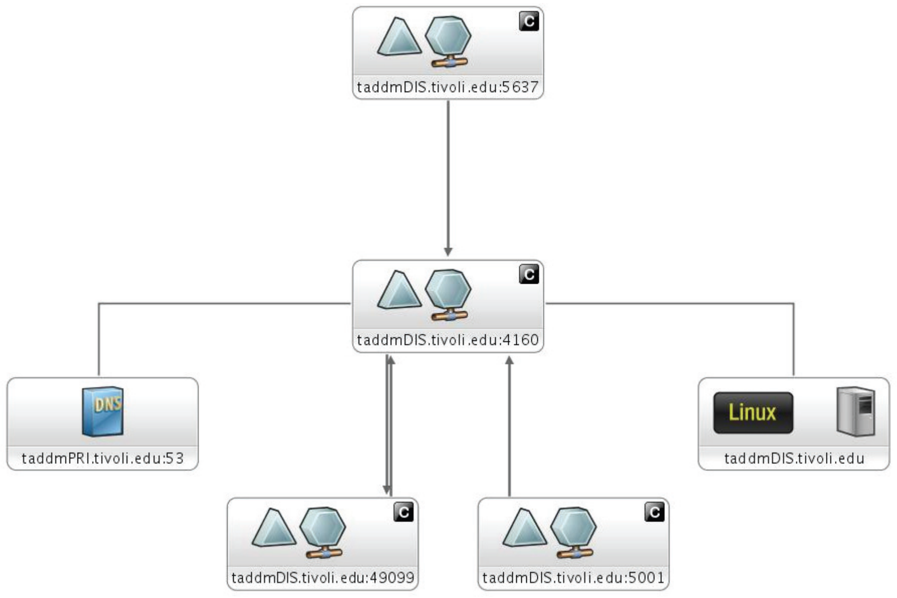
Expand All | Collapse All | Select All | Clear All

- ✓ Host (1)
  - ✓ Linux Computer System (1)
- ✓ Service Dependency (1)
  - ✓ DNS/NIS Service (1)
- ✓ Transactional Dependency (4)
  - ✓ Source (1)
    - ✓ CollationProcesses (1)
  - ✓ Target (3)
    - ✓ CollationProcesses (3)

<< Back | Finish | Cancel

To include all the related components in the topology map, click select All, and then **Finish** to see the topology.

c. The topology is not displayed. Depending on which component you selected the content of the topology may vary. In this example, you see that the explored CollationProcesses instance (the one listening to port 4160), is related to a host system, and a DNS, as well as a number of other CollationProcesses.



Notice that the connection between the explored instance and the host system as well as the DNS does not have any arrowhead. This implies that the nature of the relationship is implicit (enforced by the Common Data Model). The relationships between the CollationProcesses instances are explicit (discovered, uni-directional, transactional relationships) so there you see the arrowheads indicating whether the component is a provider (no arrowhead) or a consumer. (incoming arrowhead). In this example, the explored component consumes services from all the other CollationProcesses instances, and provides services to the instance listening on port 49099.

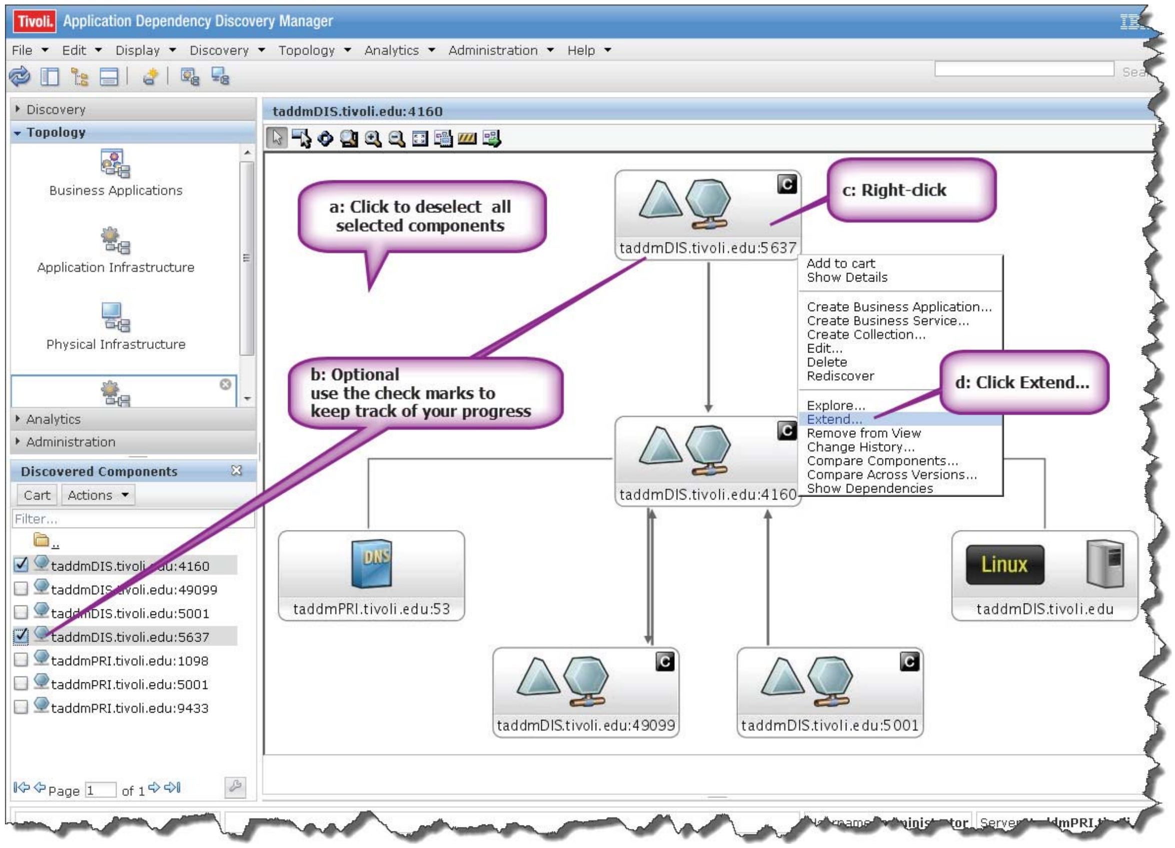
4. Now to add additional resources that are related to the new CollationProcesses instances that are related to the originally explored component, follow these steps:

   a. Click the background of the topology to cancel any active selections.

   b. Determine the component to extend. Consider using the related check mark in the Discovered Components pane to keep track of which components you have extended and which ones you still need to process.

   c. Right-click the component you want to extend.

   d. Select **Extend...** from the context menu.

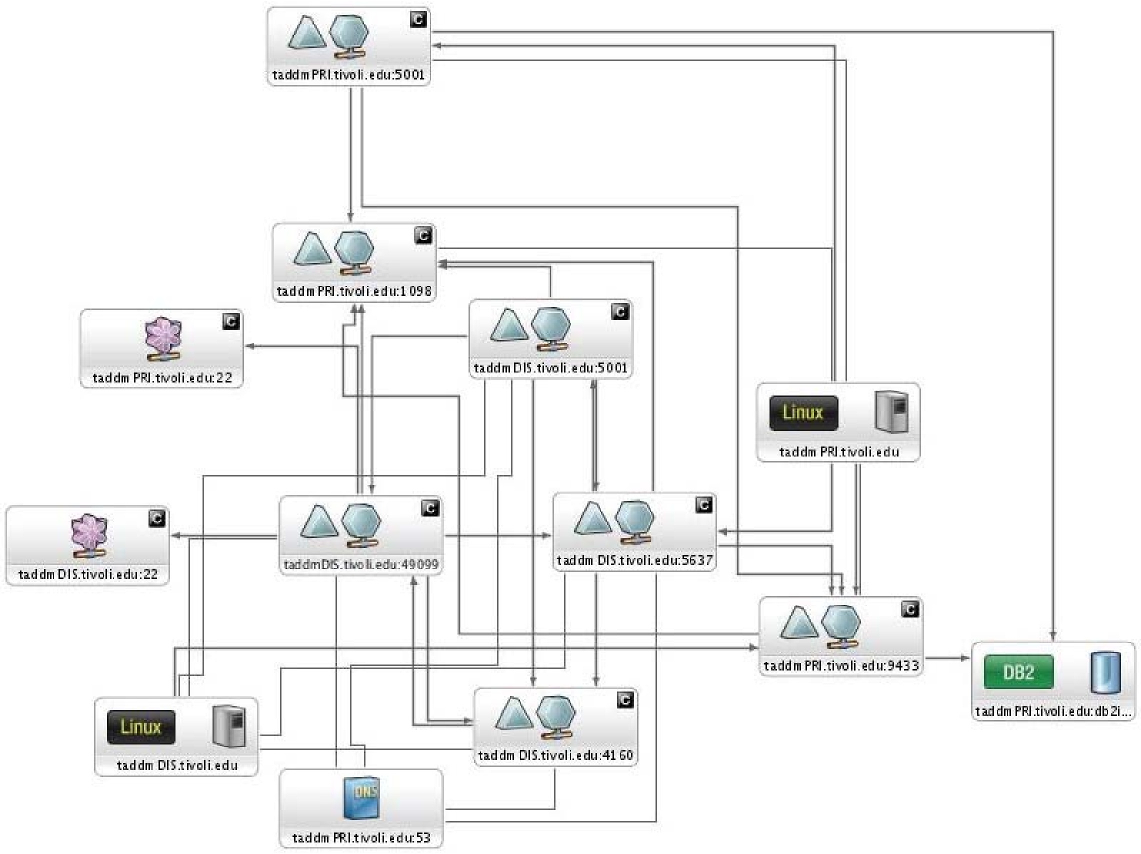


   e. To select the new set of related components select all relationship types and all node types from the Explore wizard, similar to the actions you performed in step 3 on page 33.

   Notice how the topology is augmented with additional components.

5. Repeat the previous step until you have extended all the CollationProcesses instances in the topology.

6. When you have extended the topology with all components that are related to all seven CollationProcesses instances your topology should look similar to this:



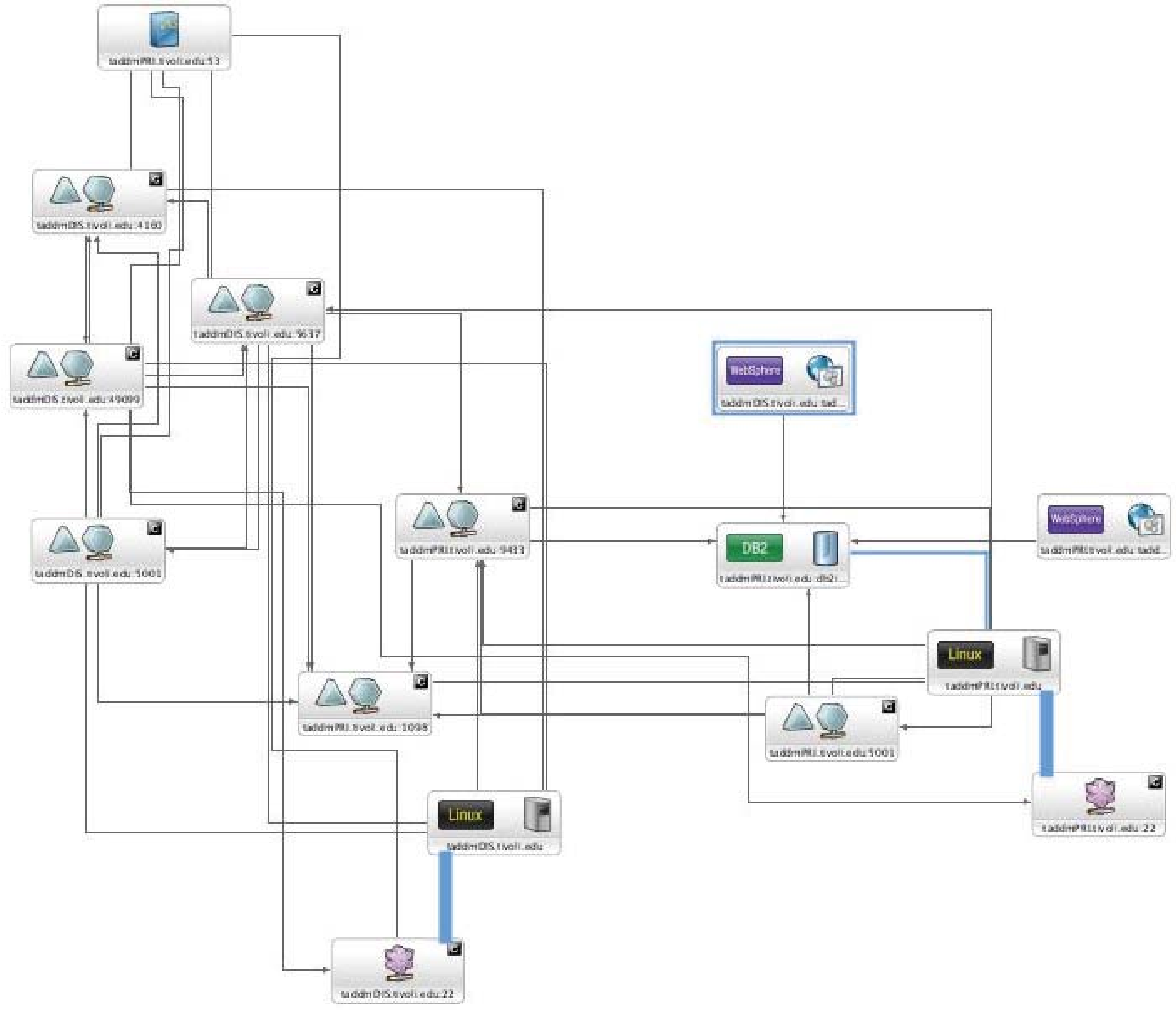
Notice that in addition to the CollationProcesses and the systems that host them, the topology also contains a DB2 Instance, two SSH servers, and a DNS. As you know, TADDM uses a database to store the discovered information, DB2 in this case, so it is hopefully no surprise that the relationship between the CollationProcesses instances that access the database is discovered.

During discovery, TADDM uses ssh to access the systems that are discovered. This is why you see the relationships between the CollationProcesses instance that is manages discovery and the two SSH Servers.

The DNS service is used by the components to locate one another. Depending on the caching status at the time of discovery, TADDM may have discovered more of fewer dependencies between the components and the DNS server.

7. To complete the topology, you should extend it with the components related to the DB2 Instance and the two SSH servers.Repeat the extend actions for those three components.

   When you have completed the topology, it looks like this:

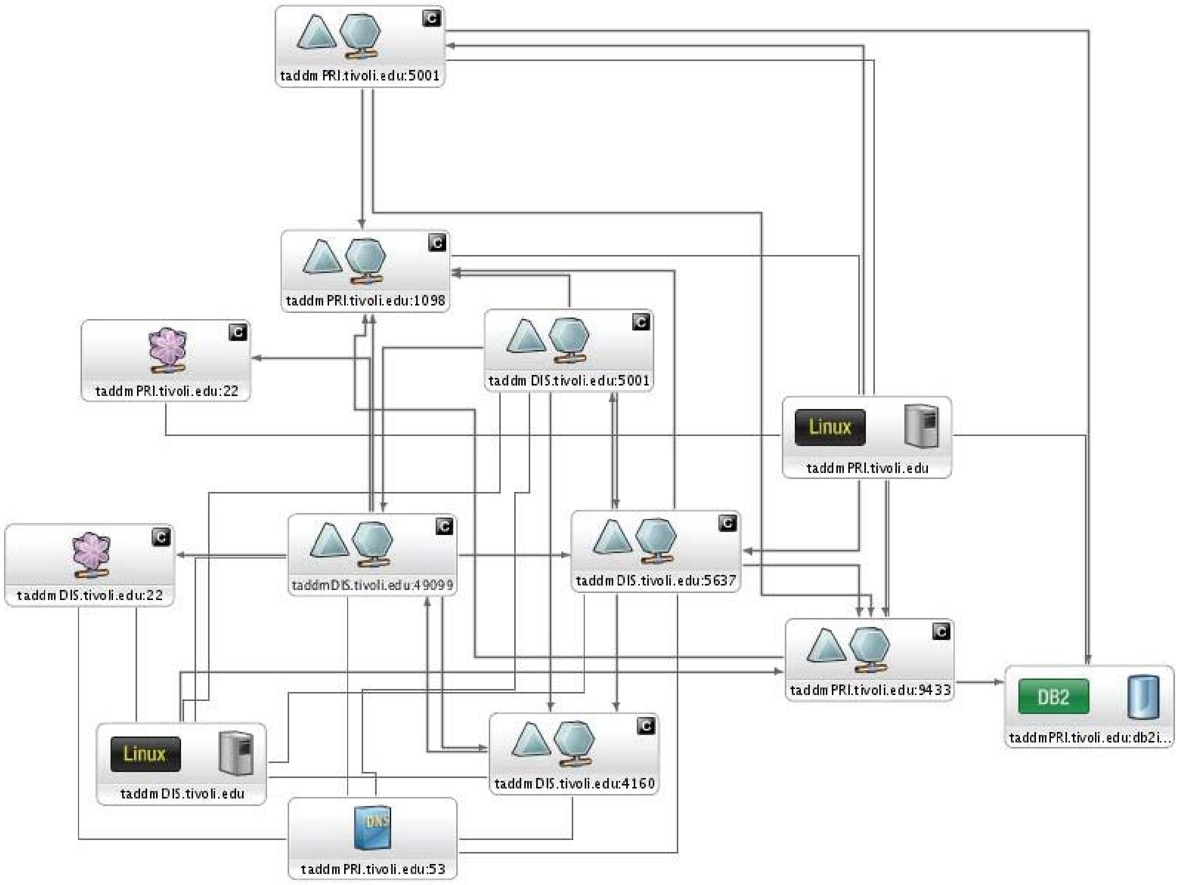


   Notice that as a result of extending the DB2 Instance, two WebSphere Server components were added. These are part of the Trade application but uses a database that is hosted on the same DB2 Instance as the database used by TADDM.

   Besides these two WebSphere Servers not additional components were added when you extended the map with relationships and nodes related to the last three components. However, new relationships were added to the topology. If you look closely, you see that now both SSH Servers and the DB2 Instance are related to a host system, so you get a better idea of the physical implementation.

   Remember, in this example you included all relationship types. If you choose only to include Host relationships you can build a topology similar to the Physical Topology for business applications, and if you exclude Host and IP Dependency relationship types, you get a topology similar to the business application Software Topology.

8. To clean up the topology, right-click each of the two WebSphere Server components and select Remove From View. When you are done, the final topology should look like this:


taddm PRI.tivoli.edu:5001
taddm PRI.tivoli.edu:1098
taddm PRI.tivoli.edu:22
taddm DIS.tivoli.edu:5001
Linux
taddm PRI.tivoli.edu
taddm DIS.tivoli.edu:22
taddmDIS.tivoli.edu:49099
taddm DIS.tivoli.edu:5637
taddm PRI.tivoli.edu:9433
DB2
taddmPRI.tivoli.edu:db2i...
Linux
taddm DIS.tivoli.edu
taddm DIS.tivoli.edu:4160
DNS
taddm PRI.tivoli.edu:53

9. Now that you have your final topology, you can export it to an image (png, svg, or jpg format). To save the topology in a reusable format you need to select all the components and create a new business application. This will create a business application with statically assigned resources. This guarantees that you capture the exact content as it exists at this point in. If additional CollationProcesses resources are added to your topology, for example if you add an additional Discovery Server, this will not be included in the business application. However, by saving the current topology as a business application, you create a baseline that later on can be compared to other business application definitions.

   As an alternative, you can also create a collection that includes all the components in the topology. This will create a group of components for which you can see the physical, and the relationship topology. In addition, the collection can be used to control access to resources or as an inventory baseline.

   To create a baseline business application, complete these steps:

   a. Use the select Rectangle tool () to mark all the components in the topology.
   b. Click the select tool (), right-click on one of the selected components, and select **Create Business Application...** from the context menu.
   c. When the Create New Group window opens, provide a name of TADDM Baseline in the General Information page and click **Next**.
   d. In the Rules window, click **Next**.
   e. In the Content window, notice how the components are associated with the functional groups. Then, click **Next**.
   f. In the Administrative Information page, supply a value of `Baseline` in the Group field, and click **Finish**.

10. To create a collection that includes all the components in the topology, right-click any of the selected components, and select Create Collection... form the context menu. In the Create New Group wizard, provide a name of TADDM Baseline Inventory, and click **Next**, **Next**, **Finish** to complete the creation.

    You can access the newly created groups from the Discovered Components pane by navigating down the Groups path. If you want to, you can try looking that the various topology views that are available for the two different types of groups.

This completes the exploration exercise. You have seen how you can use the TDDAM explorer to identify components that are related to one another either based on relationships enforced by the Common Data Model (implicit) or relationships identified during discovery (explicit).

5 Dynamically defined business applications

To create the business application that contains all CollationProcesses instances as well as the database management instance that supports them, the SSH Server instances needed for discovery, and DNS Services in the exercise environment you must create a business application using the Grouping Composer.

In the previous exercises you have already been working with the Grouping Composer to create business applications and collections, but so far you have not been working with rules. Rules are used to assign components to functional groups based on queries against the TADDM. The queries are based on the Model Query Language (MQL) which is very similar to the Structured Query Language (SQL). To build your own queries you must understand how data are organized in the TADDM database. This is documented in the Common Data Model information. However, TADDM provides a Query wizard with which you can easily create basic queries to be used to identify top-level components. Custom queries can also be used to generate basic reports or lists of resources.

In this exercise you create business application that represents the TADDM infrastructure in the exercise environment by crating and populating the functional groups from queries against the TADDM database. To include all four component types (CollationProcesses, SSH Servers, DB2 Instances, and DNS Services) you define four rules, some of which are based on custom queries. When you are done, the new business application will resemble the TADDM Baseline you created in the previous exercise.

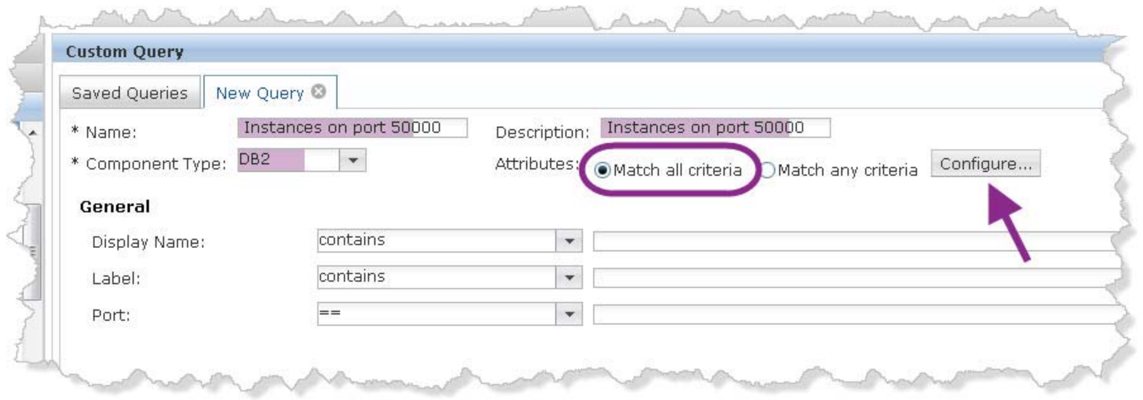
Exercise 13: Creating custom queries

Before you start creating the business application you should create a couple of custom queries that can be used as templates for your business application rules. In the following you create two queries: one to find all CollationProcesses and one to find all DB2 Instances that are listening on port 50000. In case you did not notice, the DB2 Instance that TADDM connects to listens on port 50000, so to avoid identifying DB2 instances listening to other ports this restriction is added to the query.

To create these two queries, complete these steps:

1. Open the Custom Query tool from the Data Management Portal by navigating to **Analytics** > **Custom Query**.
2. When the, empty, list of custom queries is shown, complete these steps to create your first query which is used to identify CollationProcesses instances.
   a. Click **New**.
   b. In the *New Query* tab, provide the following information:

Name:	`CollationProcesses`
Description:	`Find all TADDM processes`
Component Type:	`CollationProcesses`
Attributes:	`Match all criteria`



Custom Query

Saved Queries | New Query

* Name: Instances on port 50000

Description: Instances on port 50000

* Component Type: DB2

Attributes: [✓ Match all criteria] Match any criteria Configure...

General

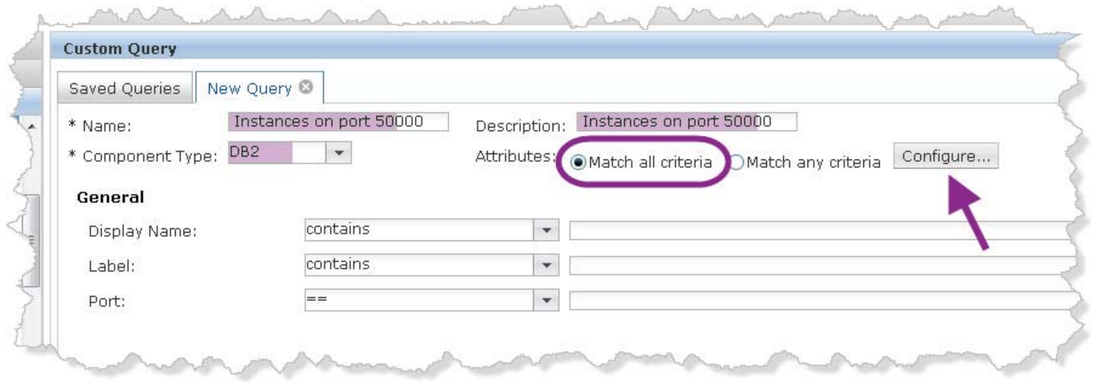
Display Name: contains

Label: contains

Port: ==

   c. If you want to verify that all seven CollationProcesses are identified, click **Run Query**, and inspect the results in the Results tab.
   d. Go back to the New Query tab, and click **Save**.
3. To create the query to locate the DB2 Instance, follow these steps:
   a. From the Saved Queries tab, click **New**.
   b. In the New Query tab, provide the following information:

Name:	`DB2 Instances on port 50000`
Description:	`DB2 Instances on port 50000`
Component Type:	`DB2`
Attributes:	`Match all criteria`



Custom Query

Saved Queries | New Query

* Name: Instances on port 50000

Description: Instances on port 50000

* Component Type: DB2

Attributes: [✓ Match all criteria] Match any criteria Configure...

General

Display Name: contains

Label: contains

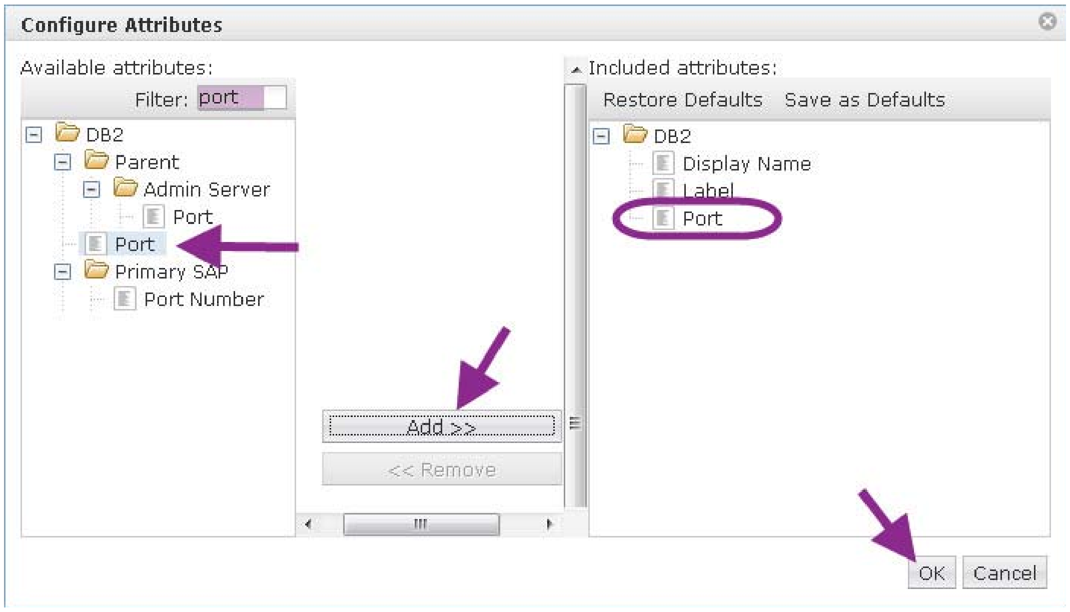
Port: ==

c. To define additional criteria in order select only DB2 Instances that listen to port 50000, click **Configure**.

When the Configure Attributes window opens, enter `port` in the Filter field in the Available attributes section. This will limit the list of available attributes to only those that contains the string your provided.

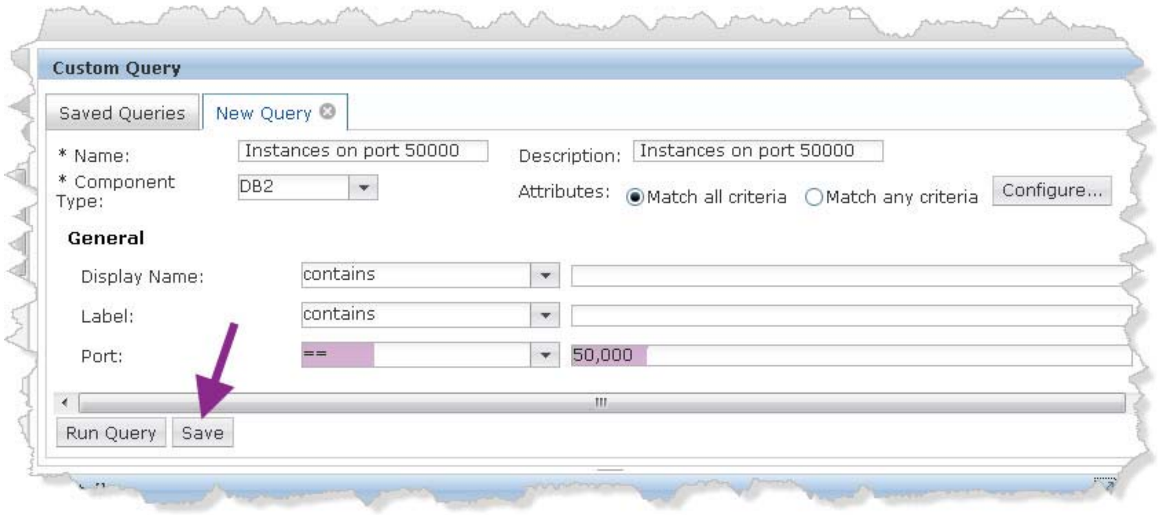
Now, select the Port attribute that is an immediate child of the DB2 resource type. Be careful not to select the port that is related to the Admin Server.

To include the selected attribute in the query, click Add, and verify that the Port attribute now appears in the Included attributes section.



Click **OK** to save the included attributes.

d. Upon your return to the Custom Query pane, select == as the value for the comparison field for the Ports attribute, and enter `50000` in the value field.



* Name: Instances on port 50000
Description: Instances on port 50000
* Component Type: DB2
Attributes: (●) Match all criteria ( ) Match any criteria

General

Display Name:	contains	
Label:	contains	
Port:	==	50,000

Click **Save** to store your query in the TADDM database.

e. If you want to verify that the DB2 Instances on port 50000 query produces the expected results, select it and click Run Query. If you run the query, expect to see only one DB2 instance.

4. Once you have defined you queries, you will be able to see them in the Discovered Components pane. Try opening the Custom Queries link at the root level in the Discovered Components navigation hierarchy, and verify that you see both of your queries. When you open a query, you will see the resources that you saw in the reports.

You have now created tow queries that can be used as templates for the rules that are used to populate the functional groups of the business application you are about to create.

Exercise 14: Defining a rule-based business application

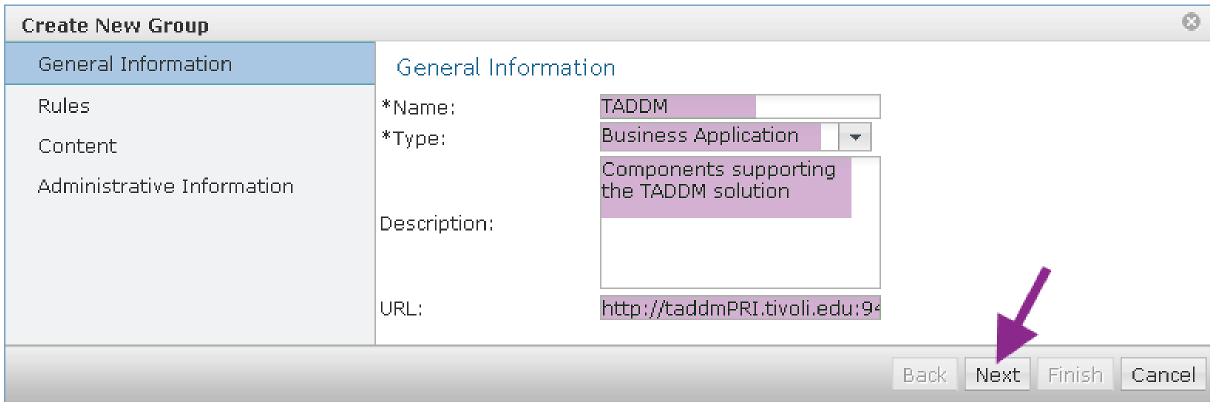
At last you are ready to define your first business application with which components will be dynamically associated. In this example you create a business application definition that is similar to the statically defined TADDM Baseline application. The new business application requires four different functional groups:

- CollationProcesses
- SSH Servers
- DB2 Instances
- DNS Service

To create the business application definition and the functional groups, complete the following steps:

1. In the Data Management Portal, navigate to **Discovery** > **Grouping Composer**.
2. Click **New...** to create a new group. This action will launch the Create New Group wizard.
3. Provide the following information in the General Information page of the Create New Group window:

Name:	`TADDM`
Type:	`Business Application`
Description:	`Components supporting the TADDM solution`
URL:	`http://taddmPRI.tivoli.edu:9430`



*Name: TADDM
*Type: Business Application
Description: Components supporting the TADDM solution
URL: http://taddmPRI.tivoli.edu:94

Click **Next** to continue.

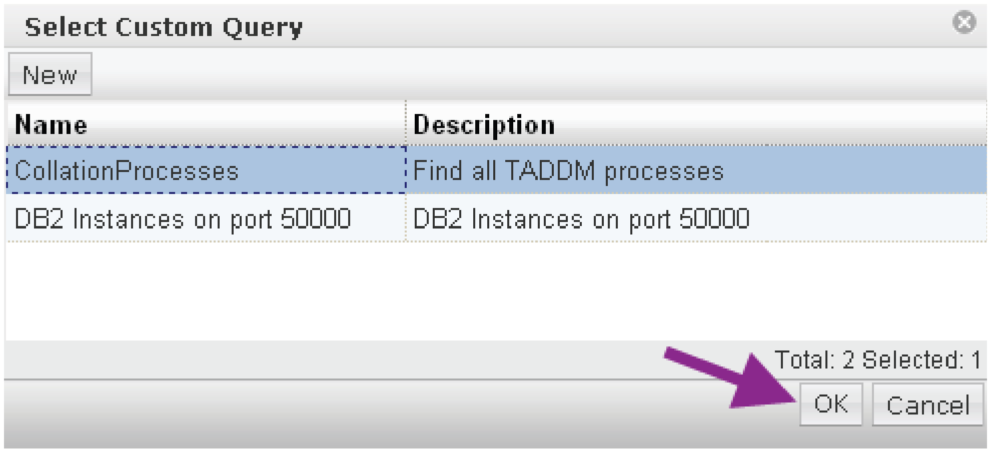
4. The Rules page is where the magic happens. Here you can specify multiple rules, each of which populate a dedicated functional group.

   To create the rule that is responsible for adding the CollationProcesses components complete these steps:

   a. Provide a value of CollationProcesses for both the Rule name, and the Functional group name fields.

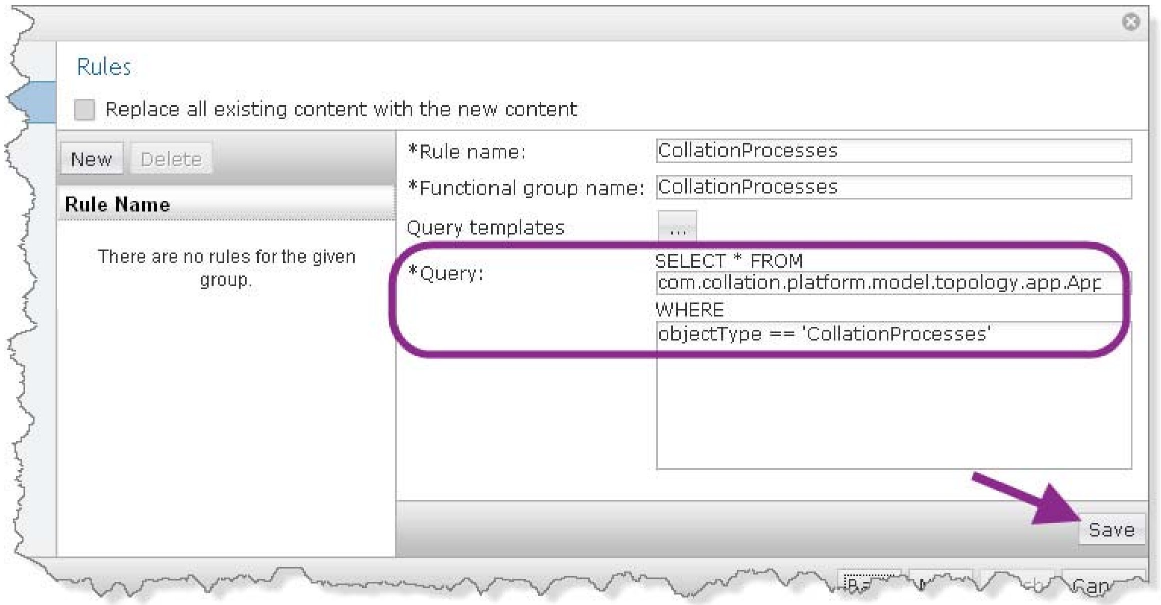
   b. Click the Query templates selection button (...) in order to clone one your custom queries into the Query field.

   c. From the select Custom Query window, select the query named CollationProcesses, and click **OK**.



Name	Description
CollationProcesses	Find all TADDM processes
DB2 Instances on port 50000	DB2 Instances on port 50000

   d. When you are returned to the Rules page in the Create New Group wizard, notice how the Query field has been populated


Rules
Replace all existing content with the new content
New
Delete
Rule Name
There are no rules for the given group.
Query templates
...
Save

*Rule name: CollationProcesses
*Functional group name: CollationProcesses
*Query: SELECT * FROM com.collation.platform.model.topology.app.App WHERE objectType == 'CollationProcesses'

The actual MQL Query that is used to populate the CollationProcesses functional group is constructed by combining select * FROM with the information in the upper field, and assign a WHERE clause as defined in the bottom field. For most component types, the long class name, *com.collation.platform.model.topology.app.AppServer* in this case, can be substituted with the short form *AppServer*. If you want to test this query from the TADDM SDK API command-line interface, you could rewrite it to this:

```
select * from AppServer where objectType equals
'CollationProcesses'
```

Also notice that rules only return the guids of the identified components. So the starting the manual query with `select guid` instead of `select *` would be more accurate.

Click Save to save the rule, and notice how the rule now appears in the list of rules in the middle of the Rules page.

5. To create a second rule, to populate a functional group named Db2Instances, click New, and repeat the previous step, with these modifications:
   - Use `DB2 Instance` as both the name of the rule and the name of the functional group.
   - Clone the custom query named *DB2 Instances on port 50000*.

   Remember to click **Save** when you are done.

6. Now, the next rule is intended to populate a functional group named SSHServer with all the SSH Server components in the environment. To create this rule, do the following:

   a. Click **New**, and use the value SSHServer for both the name of the rule and the name of the functional group.

   b. Click the Query templates selection button (...) in order to open the list of custom queries.

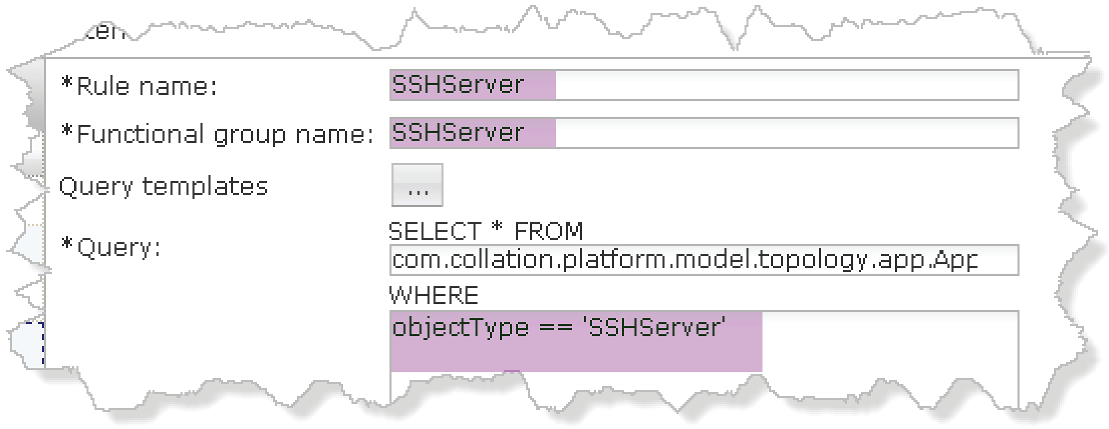
   c. Select the CollationProcesses query and click OK to copy the custom query to the query in the rule.

   d. Update the where clause by replacing:

   ```
   objectType == 'CollationProcesses'
   ```

   with

   ```
     objectType equals 'SSHServer'
   ```



*Rule name: SSHServer

*Functional group name: SSHServer

Query templates ...

*Query: SELECT * FROM com.collation.platform.model.topology.app.App WHERE objectType == 'SSHServer'

   e. Click **Save** to store the rule in the TADDM database.

7. You create the final functional group, for DNS Servers, manually. Complete these steps:

   a. Click **New**, and use the value `DNS/NIS Service` for both the name of the rule and the name of the functional group.

   b. Provide a value of `DNSService` in the select * FROM field, and leave the WHERE field empty.

Notice that the name of the specific component type name to use in the select * FROM field can be found in the Common Data Model documentation.

c. Click **Save**.

8. By now you have defined all the necessary rules. These will all be executed every time the topologyBuilder background task runs. By default this is ever 4 hours, but in the exercise environment it has been changed to every 6 minutes. If new resources that match any of the rules are found, they will be added to the functional groups. Notice that by default nothing is removed.

   However, if you check the *Replace all existing content with the new content* option, all functional groups in the business application will be cleared and repopulated with the results from the rules. This way, you will not only automatically include new components, but also remove the association components that may have been changed or deleted. For example, if the port number of the DB2 Instance is changed to 50001, it will no longer be part of the business application, provided the *Replace all existing content with the new content* option is enabled. Notice that any statically added components also will be removed at the next topologyBuilder run when the *Replace all existing content with the new content* option is enabled.

   To enable automatic removal of static and obsolete components, check the *Replace all existing content with the new content* option, and click **Save**.

9. Click **Next**, to navigate to the Content page.

   At this point, no components have been associated with any functional group. Since you are creating a new business application and you have not executed any of the rules, no resources have been assigned yet. You can use this page to statically associate components with the business application, but remember that they may be removed automatically.

   Click **Next** to continue.

10. The Administrative Information page contains the usual fields you normally find in an AdminInfo component. Enter your name as the Admin contact, and click Finish.

    At this point all the rule queries are validated, and if errors are found you will be notified. If you experience this, use the Back button to navigate back to the Rules page, correct your error(s), and use the **Next** > **Next** > **Finish** to try to save the business application definition again.

You are done. The group definition behind the TADDM business application has now been created, and because of the validation all the rules have been executed, and the business application itself is created.

If you take a look at physical and software topologies of the TADDM business application you just defined, you will see that it, as expected is remarkably identical to the TADDM Baseline application. That was exactly what you wanted - and hopefully what you expected.

The main difference between the two definitions is that the TADDM Baseline definition is static, and thus will not be updated if the configuration of the underlying components changes. The TADDM definition, however, is refreshed every time the topologyBuilder process executes, and the entire content of all functional groups is replaced with the current components that meet the criteria defined in the rules. If you, in the TADDM group definition, disable the *Replace all existing content with the new content* option, the new members will be added to the functional groups, but nothing will be removed. Naturally, all additions or removals will be marked as changes to the business application, and this information can also be used to submit notifications (events) to the application owners or the operational staff.

So, the use of rules, in combination with the *Replace all existing content with the new content* option allows you to define business applications that are automatically adjusted to conform with the discovered reality. This is extremely important in dynamic virtualized infrastructures, such as cloud environments. For cloud environments, remember that monitoring agents can submit changes to the infrastructure they manage to TADDM using the REST interface. This allows for updating the TADDM database without discovery, and thus your business application topologies are updated as near-to-real-time as possible.

Exercise 15: Comparing business applications

If case you forgot, you can access the physical and software topologies for your business application directly from Discovered Components pane by following the **Groups** > **Business Application** path. Notice that from the Application Summary, you can use the Groups action to see the members of the individual functional groups within an application.

Now to compare the TADDM and the TADDM Baseline applications complete these steps:

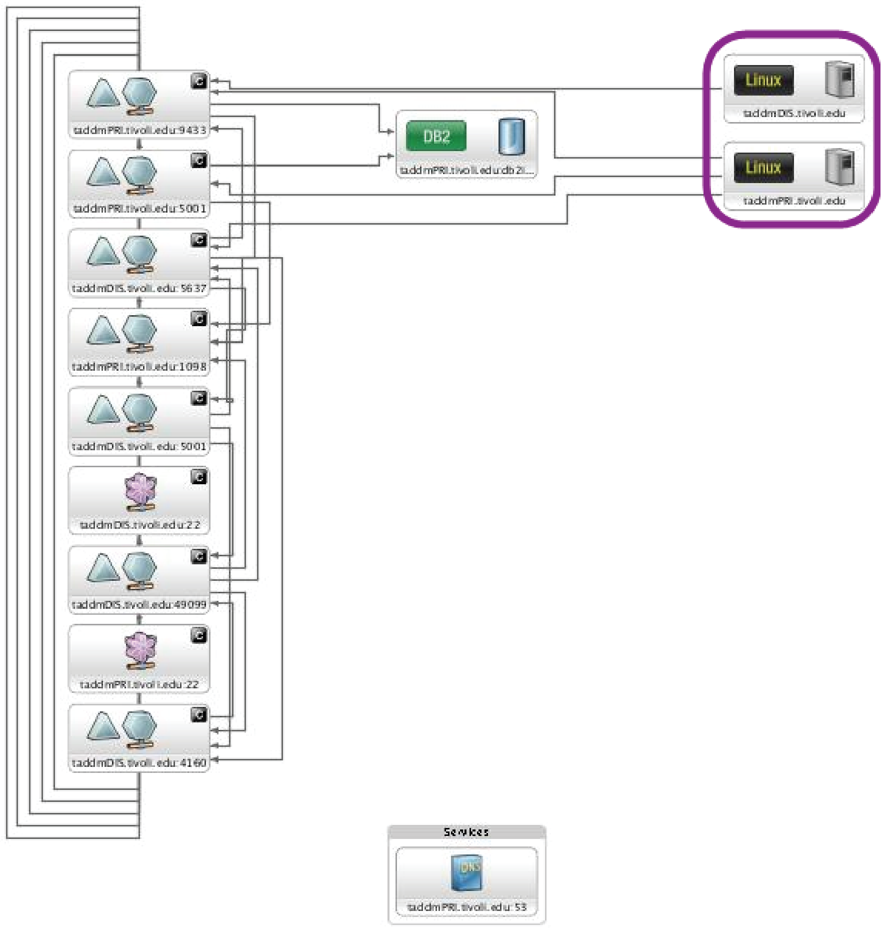
1. From the Discovered Components pane, navigate to **Groups** > **Business Application** so you see all three business applications in your environment.
2. To compare the physical topologies of the two TADDM applications complete these steps for each of the applications:
   a. Check the first of the TADDM applications and select **Action** > **Show Physical Topology**.
   b. Now clear the first application and check the second TADDM application in the Discovered Components pane, and repeat the action.

   Did you notice any significant differences between the two topology maps?

Note: Hopefully you have noticed that the topologies are cached in the Topology drawer in the Navigation pane. If you need to revisit a topology, it is easier to work with the cached copy than re-creating the topology from the Action in the Discovered Components pane.

3. Now try the comparison from for the software topology using the Show Software Topology action.

   You will find that the TADDM Baseline Software Topology contains two Linux systems that you do not see in the TADDM software topology.



   This has happened because the hosting system components were included in the exploration map you used as the basis for the TADDM Baseline application, and you did not create a rule for the hosting systems in the TADDM business application. However, the two physical topologies were identical because TADDM under the covers include the hosting platforms, and optional network components, in the physical topologies so you can easily see where your software components are hosted. The hosing platforms are identified by an implicit relationship between any application server component, and its related host.

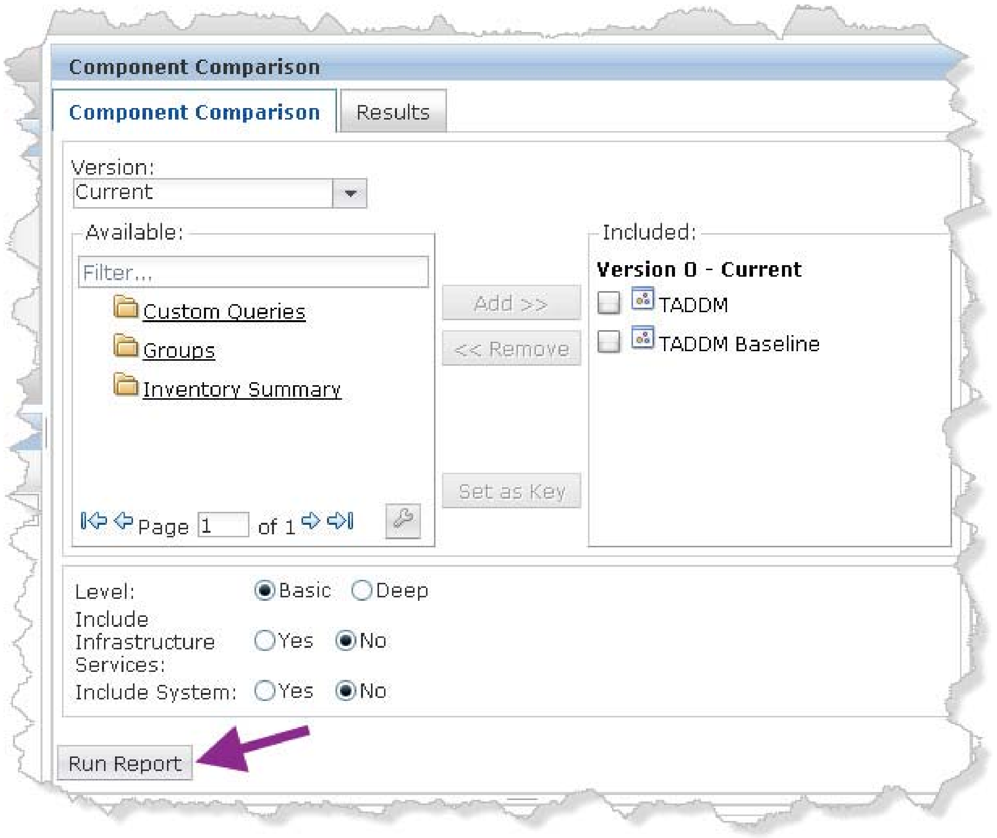
It is entirely your decision which component types to include in a business application. If you primary goal of creating the application definitions is to provide the logical groupings to support your Incident, Problem, Configuration, and Change Management, as well as your operational processes, there is not need to include the OSI layer 2 and 3 components. When the TADDM data are leveraged by the solutions that support your management and operational processes, the hardware-related information is automatically included.

However, if you create business applications in order to produce inventories and build reports to be used for accounting, enforcement of authorizations, and other administrative functions that are not based on the Common Data Model hierarchy and relationships, you may want to include all the relevant components types in the business application. Only the components are specifically associated with a functional group are included in the business application inventory.

Comparing the two business applications using the topology maps may have worked for the small topologies in this example, but your application infrastructure quickly becomes so complex that it is practically impossible to compare the topologies manually. TADDM provides a Component Comparison feature that you can use to let the system compare the two business applications, and provide a detailed report that highlights the differences.

To use the comparison feature to identify the differences between the two TADDM business applications, complete these steps:

1. From the **Groups** > **Business Applications** branch in the Discovered Components pane, check both TADDM business applications, and click **Actions** > **Compare Components...**.
2. When the Component Comparison pane opens, accept the default settings, and click **Run Report** to generate the basic comparison report.



Version: Current

Included:

Version 0 - Current

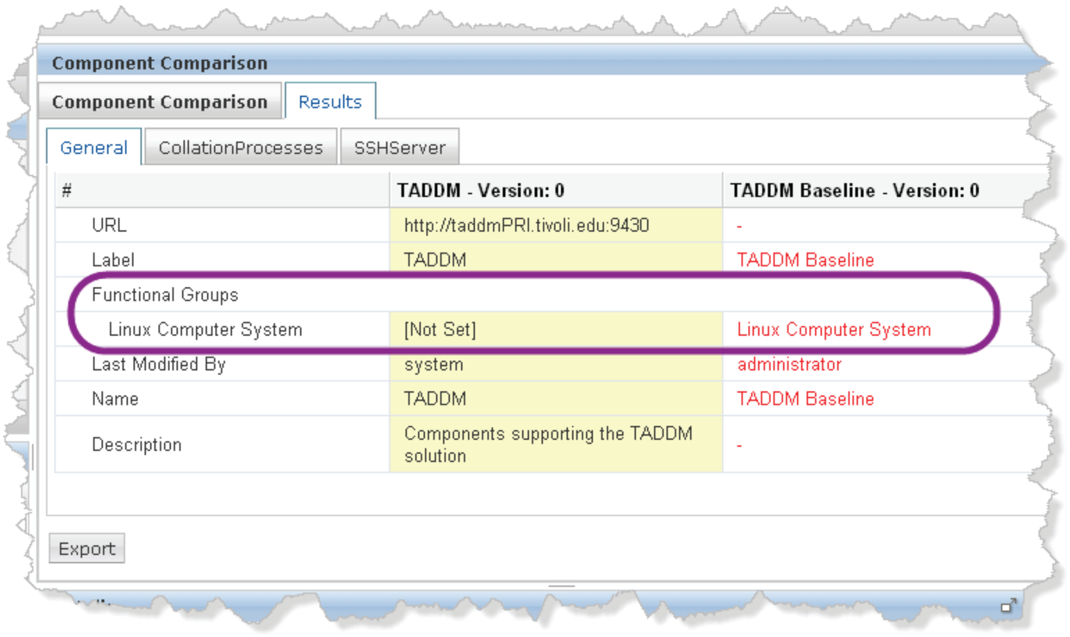
- TADDM
- TADDM Baseline

Level: [✓ Basic] Deep

Include Infrastructure Services: Yes [✓ No]

Include System: Yes [✓ No]

3. When the report is generated you see the basic business application definitions. Differences between the key component (if no key is set, the first component in the list is used as the key component) and the other components in the list are highlighted in red.



Component Comparison

Component Comparison | Results

General | CollationProcesses | SSHServer

#	TADDM - Version: 0	TADDM Baseline - Version: 0
URL	http://taddmPRI.tivoli.edu:9430	-
Label	TADDM	TADDM Baseline
Functional Groups		
Linux Computer System	[Not Set]	Linux Computer System
Last Modified By	system	administrator
Name	TADDM	TADDM Baseline
Description	Components supporting the TADDM solution	-

Pay particular attention to the Functional Groups row. You see that the functional group named Linux Computer System is not represented in the TADDM business application. You also notice that the remaining functional groups are not represented in the row. This is because they are identical, and therefore are excluded from the list.

However, in the tabs across the top of the report, you see the CollationProcesses and SSHServer tabs. if you open these tabs, you see the detailed comparison between the components that are associated with those particular functional groups. You would expect the find the DNS/NIS Service in the tabs as well, but since he content of the two functional groups are exactly the same, the DNS/NIS Service functional group has been omitted.

Had you known about this feature before comparing the topology maps, your life would have been so much easier.

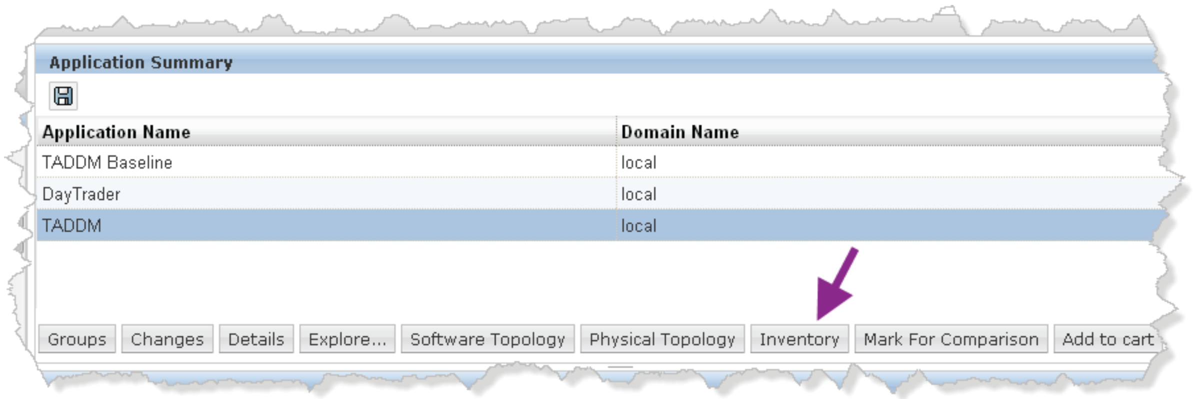
Exercise 16: Business application inventories

In the Discovered Components pane, one of the actions you can perform against a business application is Business Application Inventory. This will list the inventory of the business application you have selected. However, to access a list of all business applications with easy access to the topology, inventory, and functional group information, you can also use the Application Summary tool.

To see the inventory for the TADDM applications, complete these steps:

1. From the Data Management Portal, open the Application Summary tool by selecting **Analytics** > **Application Summary** from the toolbar.

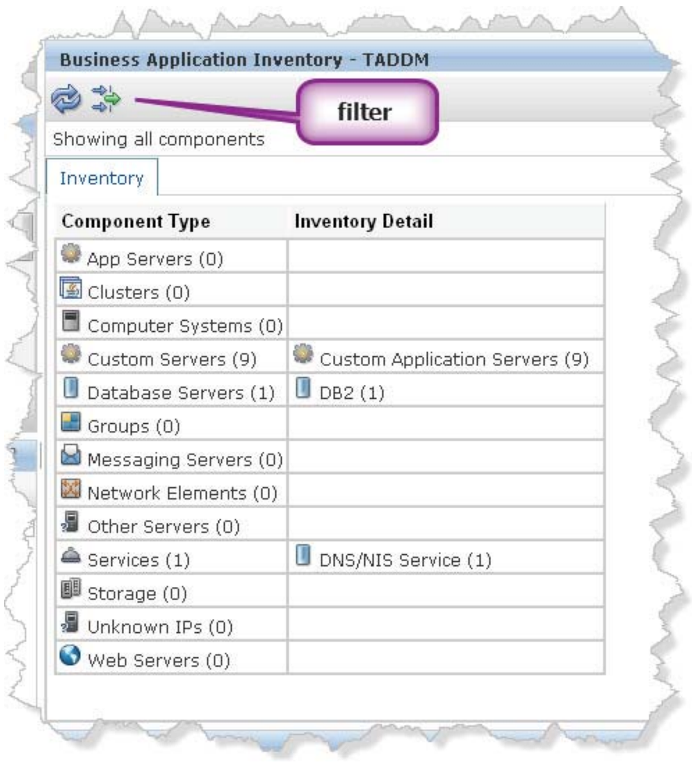
2. When the Application Summary pane opens, notice that you see all three applications, and that the available actions are listed at the bottom of the list. To see the inventory for the TADDM application, simply select the TADDM application and click Inventory.



Application Summary

Application Name	Domain Name
TADDM Baseline	local
DayTrader	local
TADDM	local

3. Notice that the business application Inventory Summary is similar to the Inventory Summary for the entire environment. in the inventory report you see the familiar component groupings and can naturally drill down into the details as you are used to from the Inventory Summary. In addition, you can use the filtering capability to filter the content of the inventory report based on dates, so you can see recently discovered components, or components that may have been retired.



Business Application Inventory - TADDM

Showing all components

Component Type	Inventory Detail
App Servers (0)	
Clusters (0)	
Computer Systems (0)	
Custom Servers (9)	Custom Application Servers (9)
Database Servers (1)	DB2 (1)
Groups (0)	
Messaging Servers (0)	
Network Elements (0)	
Other Servers (0)	
Services (1)	DNS/NIS Service (1)
Storage (0)	
Unknown IPs (0)	
Web Servers (0)	

This completes the exercises related to dynamic business application definition. You experienced how you can use queries to populate the functional groups in the business applications, and how to force the removal of changed or retired components from the business application definition.

6 Business application definition using relationships

So far you have seen how you can define business applications statically by manually associating specific, discovered components to the functional groups in the application. You have also seen how you can apply rules to a business application definition to dynamically populate the functional groups based on queries into the database. In addition, you learned that when using rule-based definitions, you can instruct TADDM to automatically remove any previously associated components from the business application if the configuration or relationships of the components have changed, so they no longer can be identified by a rule.

However, so far, the examples you have been working with only associate specific components, or component types for which certain criteria apply. The rule-based example you worked with in the previous exercises identified all CollationProcesses, or all DB2 Instances listening on a specific port without using any application-specific context. This means, that when the rules are used to query the TADDM database, all components that are identified by the rule, are associated with the business applications. In other words, the techniques demonstrated in the previous works for simple groupings, but cannot differentiate between different business application environments.

In the following, you will learn how you can create an application-specific context, and use that along with the explicit relationships discovered by TADDM to create rules that are context aware, and thus can be used to associate only a specific subset of resources to the functional groups in the business application.

In order to create these rules, you need to manually create MQL queries that identify components based on your criteria. This requires that you understand how the information in the TADDM database is organized, and known how to access the Common Data Model documentation which provides invaluable reference information. You must also master the MQL language and understand how use can use the unary operator to traverse implicit relationships.

However, the most important skill you must apply is your knowledge about TADDM discovery customization so you ensure that the components you use as marker modules are discovered, and also ensure that you discover at a point in time where transactions are flowing through your infrastructure so you discover the transactional (explicit) relationships.

6.1 Relationship-based business application definition methodology

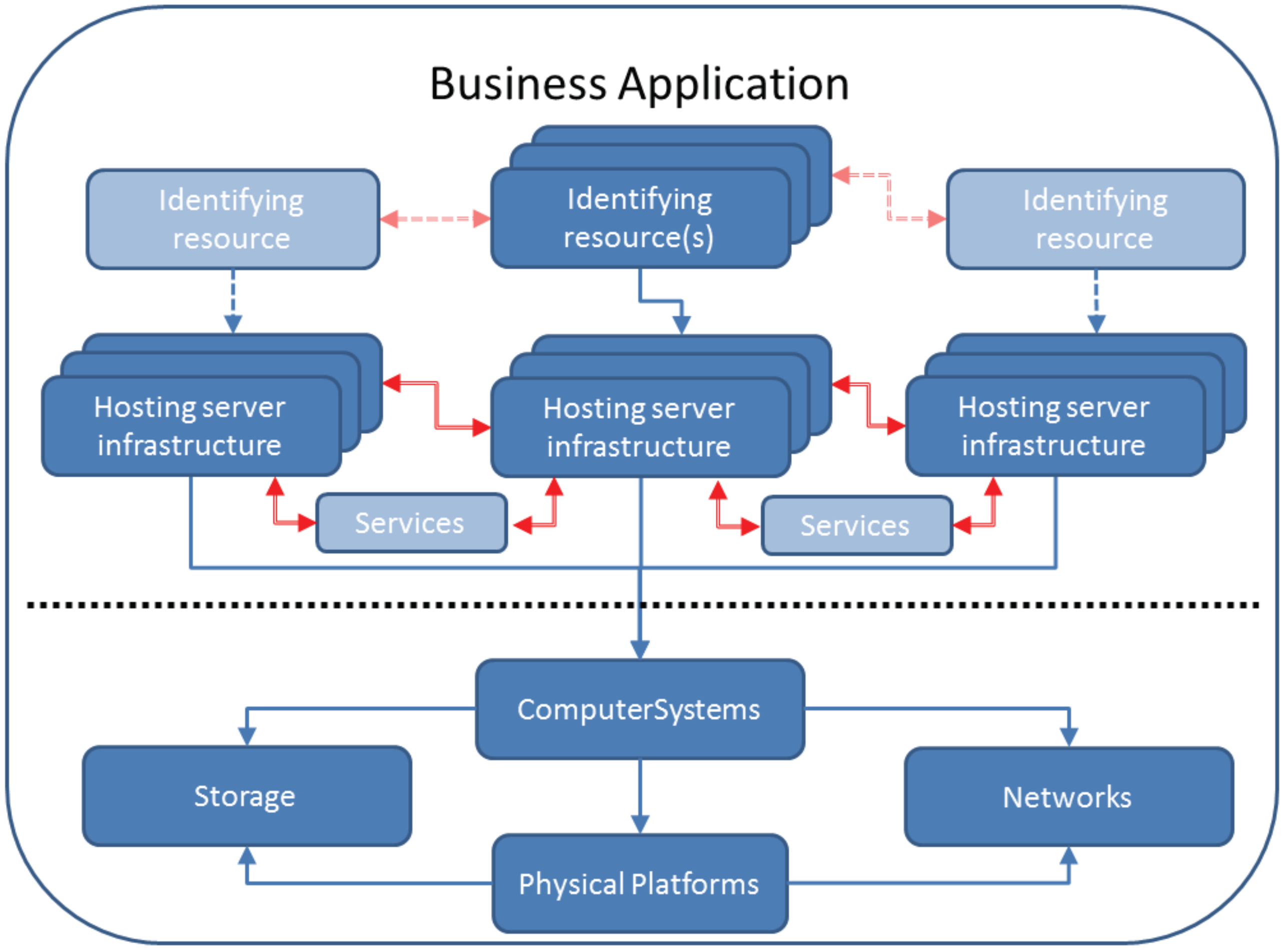
To define business applications based on context and relationships you need to identify the key components (marker modules) that can be used to uniquely identify your business applications. In the case of TADDM, processes to which an argument that contains the string *COLLATION* has been supplied could be one such component, and the name of the database used by TADDM could be another. The main point is that you must identify one or more components that can be used to provide the application context.

When the marker modules have been identified, you must ensure that it is discovered by TADDM. In many situations, the marker module is a low-level component such as a database, a software module, or a file with specific content. Often this is so specific to a particular environment that it is not discovered by the standard TADDM sensors (WebSphere, and WebLogic applications and modules, as well as databases are a few exceptions to this generalization).

Once you have identified and discovered the marker modules for your business application, you can use this in a rule to create a functional group that contains the marker modules. This functional group is the reference point to which you can refer when creating rules to populate additional functional groups that contain specific types of components that are related to the marker modules.

Next you create rules that identify the components related to the components that are related to the marker modules, and so on. You hopefully understand how the marker modules are used as the bootstrap, and from there the relationships that are maintained in the TADDM database are used to add more and more component types.

The following diagram depicts how you can organize the functional groups for the various types of components you want to include in the business application. In the top row you find the functional groups that contain the marker modules. For those groups, you must specify business application-specific criteria in the MQL queries that are used to populate the functional groups.



The functional groups in the second row represent the parent components of your marker modules. In most cases all the component types in this layer are application server or cluster component types. The MQL used to populate these functional groups only references the marker groups in the top row. If you adhere to a naming standard, these MQL queries are so general in nature that the only application-specific content is the name of the business application. The lines between these functional groups represent the explicit relationships that can be used to identify members of a group that are related to members of another group.

The functional groups at the bottom of the diagram represent the operational platforms that support your business applications. This layer of functional groups is not required because the TADDM database contains all the implicit relationships necessary to build all-encompassing physical topologies and leverage this information in related solutions such as IBM SmartCloud Control Desk or IBM Tivoli Business Services Manager. You would only define these functional groups if you want to include all the components used by the business application in order to create an all-encompassing business application inventory that can be used for accounting and other administrative purposes.

6.2 Determining your marker modules

To determine which components to use as the marker modules for a business application you must understand that the following two conditions must be met:

- You must be able to identify one or more core components that uniquely identify a particular instance of the business application.
  To differentiate between test, staging, and production instances consider using a substring of the fqdn of the parent, the contextIP or the CIRole attributes.
- The configuration of the key resources and their transactional relationships must be discoverable by TADDM.
  To discover application components that are not discovered by the TADDM sensors, consider defining and extending custom server sensors.

The identification of the key application components is usually performed by the business application owner. This is the person who should know exactly which critical components, J2EE applications, databases, message queues and so on, are the ones that used to compose the business application. If the application owner does not know, the discovered configuration and relationships may be very helpful to identify core components and reconstruct the business application from there.

Ensuring that the key resources, their parents, and their transactional relationships are represented in the TADDM database is a task for the TADDM administrator. TADDM provides capabilities to discover the most commonly used resource types, but it is very likely that some customization will be required to discover custom resources and resources for which TADDM does not provide built-in support. Often, existing tooling for monitoring and managing parts of the infrastructure is capturing the configuration information that needs to be represented in the TADDM database. This information can be loaded into the TADDM environment with a minimum of effort through discovery library adapters.

6.3 The methodology

Applying a methodology and standards to the definition of business applications will benefit you in the long run. The typical steps you need to complete to define a business application are:

1. Understand your application.
2. Identify the marker modules for the business application.
3. Identify the functional groups you need to represent the component types you want to include in the business application.
4. Configure discovery to register the functional components and relationships in the TADDM database.
5. Ensure that the marker modules have been discovered.
6. Verify if explicit relationships have been, or will be, represented in the TADDM database.
7. Create rules for the marker modules.
8. Create functional groups for the implicitly related component types.
9. Create functional groups for the explicitly related component types.
10. Test and verify.

As an example, to define the business application for the TADDM solution, each step of the methodology would involve the following tasks:

1. *Understand your application*

   At this point you should understand how TADDM works, and how the discovered information can be leveraged to understand how your application components are interconnected.

2. *Identify the marker modules for the business application.*

   All TADDM server processes are called with an argument that contains the string *COLLATION*. The name and location of the single database used by TADDM is specified in the collation.properties file, and is not likely to change over time.

3. *Identify the functional groups.*

   As you have seen in the previous exercises you need the following functional groups: CollationProcesses, Db2Instance, SSHServer, DNSService, and optionally ComputerSystems.

4. *Configure discovery to register the functional components and relationships in the TADDM database.*

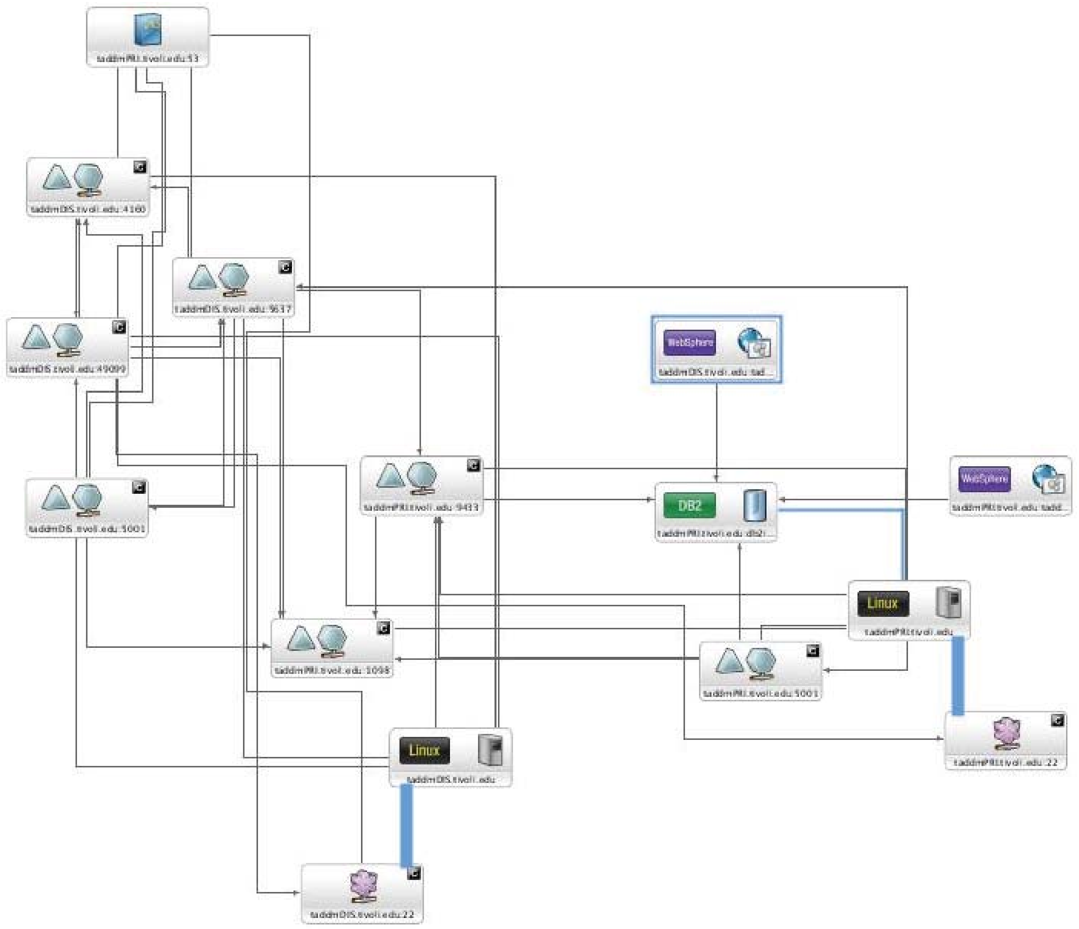
   In the discovery profile you imported at the start of these exercises the CollationProcesses custom server template is enabled to discover the TADDM application server components.

5. *Ensure that the marker modules have been discovered.*

   You performed this action in the previous exercises.

6. *Verify if explicit relationships have been, or will be, represented in the TADDM database.*

   You performed this action in the previous exercises. You may recall how you built a topology for the TADDM environment by exploring and extending the topology using both implicit and explicit relationships.



7. *Create rules for the marker modules.*

   You would create these functional groups:

CollationProcesses	A group that contains all the CollationProcesses application server instances.
Database	(optional) The group that contains the marker modules for the TADDM database. In this example, a DB2 database named TADDM.

8. *Create functional groups for the implicitly related component types.*

   You would create these functional groups:

ComputerSystems	The group of ComputerSystems that are the hosts of any member of the CollationProcesses, Db2Instance, SSHServer, and DNSService groups
Db2Instance	(a) The group of Db2Instances that are parents of all the members of the Database group. (b) As an alternative, if the database group is not created, this group can be specified as the Db2Instances that are the target of a relationship for which any member of the CollationProcesses group is the source.

9. *Create functional groups for the explicitly related component types.*

   You would create these functional groups:

SSHServer	The group of SSHServer application servers that are the target of a relationship for which any member of the *CollationProcesses* group is the source.
DNSService	The group of DNSService components that are the target of a relationship for which any member of the *CollationProcesses* group is the source.

10. Test and verify.

Does not apply in this context.

You see that the methodology is pretty basic, and that the two main steps are related to understanding the application complex you want to define as a business application, and ensuring that the marker modules and the explicit relationships are discovered.

6.4 Naming functional groups

As you have seen most of the rules you will be creating contain references to other functional groups in order to reference the members. This reference can be made by using the application name and functional group name in the rule. By applying the application name, you enable easy cloning of the rules, so you just have to update the application name if when working with a new copy of the query.

For this reason you are encouraged to apply a naming standard for your functional groups. The recommended standard is to name the functional groups similar to the singular form of the first component type that is referenced in the query.

6.5 Defining rules

As you can see from the descriptions outlined in steps 7,8,and 9 in the previous, the rules used to populate the functional groups basically take three different forms:

- Identifies one or more discrete components that are the marker modules. You must provide criteria to identify the components that should be included (*CollationProcesses* and *Database*).
- Identifies components that are implicitly related to a member of any other group using a component attribute associated with either the parent or the child (*Db2Instance*(a), *ComputerSystems*).
- Identifies components that are explicitly related to a member of any other group using the Relationships components in the TADDM database and specifying whether the group member is the source of the target of the relationship (*Db2Instance*(b), *SSHServer*, *DNSService*).

The main point is, that when you have identified the marker modules, you can use the implicit and explicit relationships to identify only the components that are directly related to the marker modules one or more levels away from the marker modules.

Exercise 17: (optional) Accessing data through MQL queries

You will quickly find, that the TADDM Custom Query wizard does not provide much help to define the queries that are embedded in the rules used to populate functional groups. The queries you need are too complex for the wizard to handle them. So, when defining rules you have to apply the queries manually.

To create the queries you need to understand both how the information in organized in the TADDM database, and the Model Query Language (MQL). Normally you create and test your queries using the command-line API which is available on the TADDM server, of if your install the TADDM SDK on your workstation.

For your reference, a brief introduction to the MQL language is included in *9.13 Querying the TADDM database* on page 113.

Exercise 18: Creating a relationship-based business application definition

In this exercise you map all the components that are used to facilitate the Trade application in the exercise environment using relationship-based rules. You may recall, that the application is implemented in a WebSphere Cluster which is front-ended by two layers of HTTP servers. For storage, the application uses a DB2 database.

Note: When you work your way through these exercises, you have to enter some rather long commands and queries. For your convenience, these commands and queries have been stored in the `/labfiles/mapping/relationship-based-queries`. It is recommended that you open this file using your favorite editor (for example `gedit`), so you can copy and paste the commands.

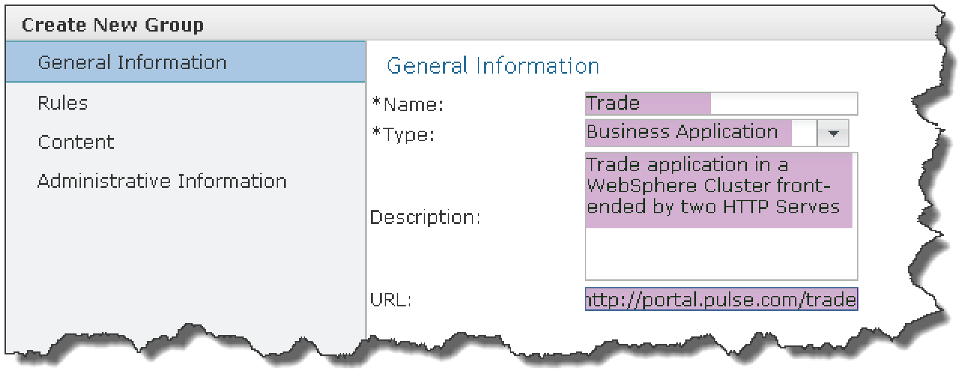
The discovery you performed earlier has discovered all the components, and relationships has been discovered if you remembered to start the TradeClient tool before you initiated the discovery.

Complete these steps to create the application definition named Trade, which will represent the components used to facilitate the Trade application you started at the beginning of these exercises,

1. From the TADDM Data Management Portal, navigate to **Discovery** > **Grouping Composer**.
2. Click **New**.

3. In the General Information page, provide the following information:

Name:	`Trade`
Type:	`Business Application`
Description:	`Trade application in a WebSphere Cluster front-ended by two HTTP Serves`
URL:	`http://portal.pulse.com/trade`



*Name: Trade
*Type: Business Application
Description: Trade application in a WebSphere Cluster front-ended by two HTTP Serves
URL: http://portal.pulse.com/trade

Click **Next** to continue.

4. To ensure that the contents of the functional groups in the business application are always refreshed, check the *Replace all existing content with the new content* option at the top of the Rules page.

At this point you have created the business application definition in TADDM, and you are ready to start associating components.

Exercise 19: Mapping your marker modules

In this example the marker module is a discovered software module. This is a common way of defining J2EE applications that, for the most part, are identified by specific ear, car, or war modules.

In this example, the key resource that identifies the business application is the trade,ear software module. Unfortunately, the Custom Query wizard only support querying top-level object types, so you have to rely on the command-line API to model your query.

If you want to practice your MQL skills, you can try issuing the following command from the `/opt/IBM/taddm/sdk/bin` directory on a TADDM Storage or Domain server, such as taddmPRI in the exercise environment:

```
./api.sh -u administrator -p collation find -d 1 "select * from
SoftwareModule where lower(fileName) ends-with 'trade.ear' and
SoftwareModule.parent.guid is-not-null"
```

Notice that the query selects all software modules for which the fileName ends with the string trade.ear no matter how it is upper or lower cased. In addition, in WebSphere Network Deployment environments the ear modules are located both on the DeploymentManager and the WebSphere Server nodes. to include only the WebSphere server nodes, the restriction to only select modules for which the parent attribute has been populated is added.

If you use this query in a WebSphere-only environment, you will see the components that are returned by the query all are of the WebSphereJ2EEApplication type, which is a subclass of the SoftwareModule type. This illustrates the fact that if at all possible, you should query super-classes to include all possible implementations of the key resource you are looking for. Naturally, if you only want to include components hosted on a specific technology platform, you should query the type that represents the specific implementation. The purpose of using super-classes is to make your application definition as transparent to the physical implementation as possible.

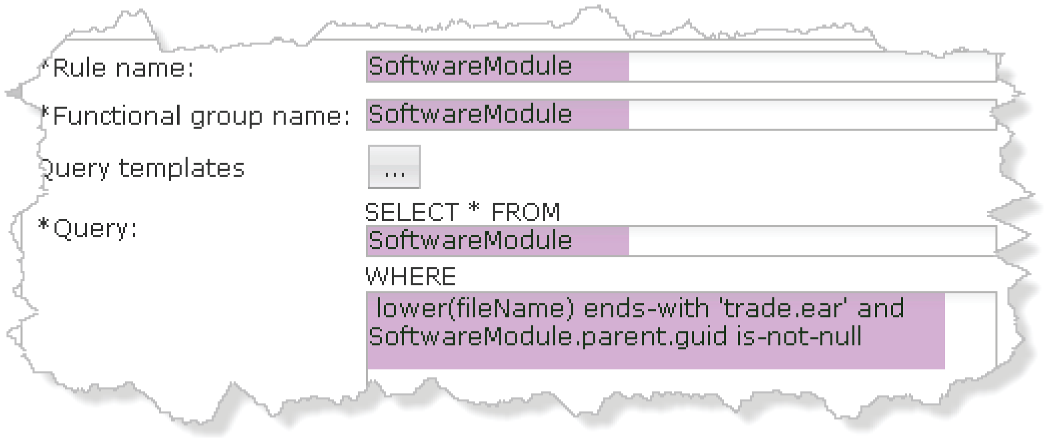
If you can successfully list the expected number of resources and are convinced that the resources returned by the query really identify your marker modules, you should copy the where clause *(lower(fileName) ends-with 'trade.ear' and SoftwareModule.parent.guid is-not-null*) to the clipboard, so you can paste it into the rule definition.

To create the functional group for the marker modules for your Trade business application, complete these steps:

1. Provide the following information for the parameters in your first rule:

Field	**Value**	**Remarks**
Rule Name	`SoftwareModule`	A label the help you identify the results returned by the query.
Functional group name	`SoftwareModule`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`SoftwareModule`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`lower(fileName) ends-with 'trade.ear' and SoftwareModule.parent.guid is-not-null`	

When you are done, your rule should look similar to this:



*Rule name:	SoftwareModule
*Functional group name:	SoftwareModule
Query templates	...
*Query:	SELECT * FROM SoftwareModule WHERE lower(fileName) ends-with 'trade.ear' and SoftwareModule.parent.guid is-not-null

2. Click **Save** when you are ready so the query is not lost, and then click **Next**, **Next**, and **Finish**, so your application definition is saved, and the rule is validated.

When the application definition is saved, TADDM validates the rules in the definition, builds the application inventory, and analyzes the relationships in order to generate the application topology. Normally, this only happens when the topolologyBuilder background task executes, but the editing of the application definition forces these activities to be performed.

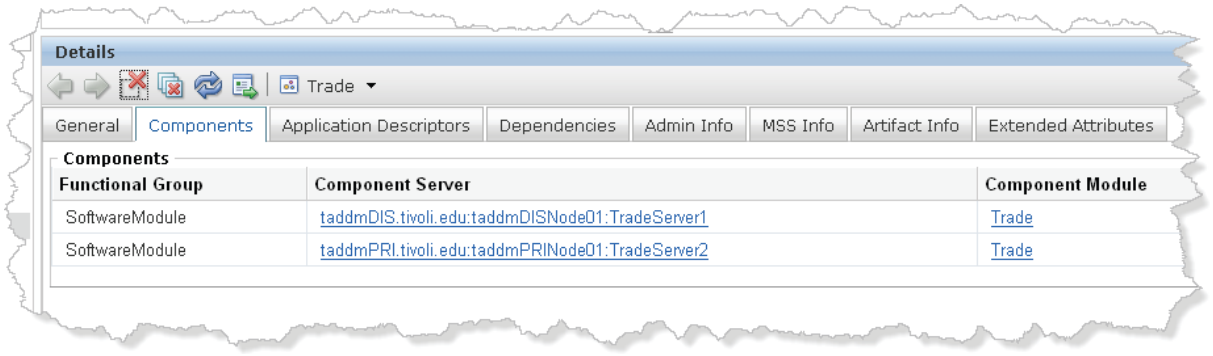
6.6 Viewing the application information

You have just added the marker modules to the Trade application. It might be interesting to see what information TADDM can show you at this point in time.

Looking at the business application details

To view the initial definition of the Trade application, complete these steps:

1. Focus on the Discovered Components pane at the lower left of the Data Management Portal, and navigate to **Groups** > **Business Applications**.
2. When you see the Trade application, select it and double-click to load the relevant information into the Details pane.
3. In the Details pane open the Components tab to see which resources are associated with the business application.



Components

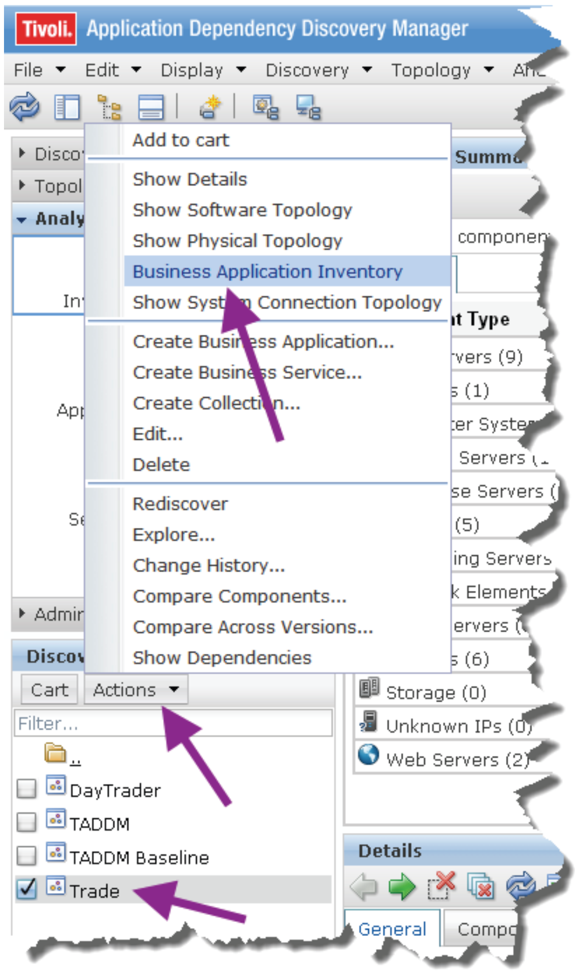
Functional Group	Component Server	Component Module
SoftwareModule	taddmDIS.tivoli.edu:taddmDISNode01:TradeServer1	Trade
SoftwareModule	taddmPRI.tivoli.edu:taddmPRINode01:TradeServer2	Trade

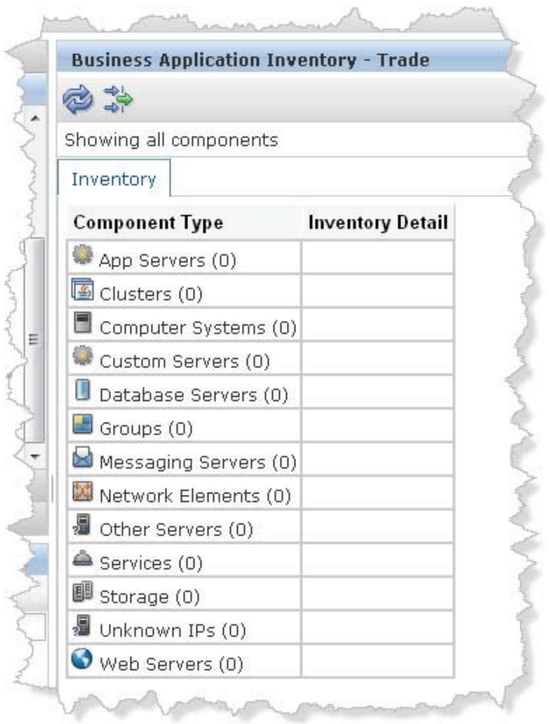
Notice how the application is composed only by the SoftwareModule functional group, and it contains two instances of the expected resources. Notice that for the software modules, the Component Server field is also populated, even though you did not add any specifications to include any WebSphere resources. This information has been picked up from the parent attribute of the Software Module.

So far it looks as if the business application has been specified correctly.

Viewing the Business Application Inventory

Instead of looking at the details, which lists all the resources that are associated with the business application, you can get an overview of the application by selecting the check mark in front of the application name in the Discovered Components pane, and choose **Actions** > **Business Application** Inventory.





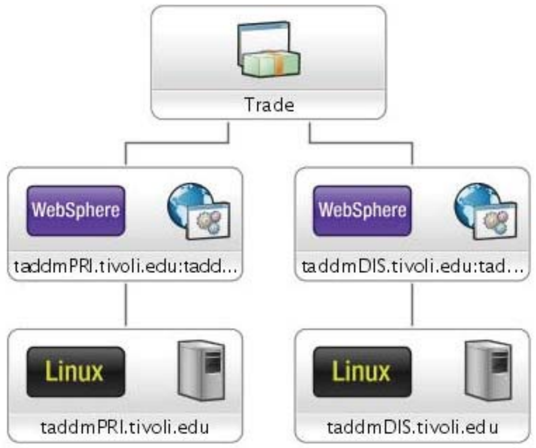
Component Type	Inventory Detail
App Servers (0)	
Clusters (0)	
Computer Systems (0)	
Custom Servers (0)	
Database Servers (0)	
Groups (0)	
Messaging Servers (0)	
Network Elements (0)	
Other Servers (0)	
Services (0)	
Storage (0)	
Unknown IPs (0)	
Web Servers (0)	

When the Business Application Inventory for the Trade application is shown, you should see the same basic information as you saw in the Components tab in the Details pane. However, since the business application does not contain any top-level components, you see that the inventory summary is completely empty.

The Physical topology

To see if TADDM can provide you with a physical topology turn your attention to the Discovered Components once again, ensure that the Trade business application is selected, and click **Actions** > **Show Physical Topology**.

The following topology view will be displayed:



You see that even though the inventory did not reveal any components in the application, a physical topology. Notice that the topology contains the WebSphere Application Server resources that host the SoftwareModules you included in the application.

The reason why the WebSphere Servers are included in the topology is, that they are identified as the nearest top-level parents (through the implicit relationship hierarchy) of the SoftwareModules.

Top-level objects are defined in the Common Data Model as:

> *Expresses whether instances of this object class form units that are generally displayed independently on a user interface.*
>
> *Instances of top-level object classes form units that are generally displayed independently on a user interface. For example, an operating system is normally only shown in the context of a computer system, and therefore is not considered to be top-level, whereas the computer system itself is generally considered to be a top-level object.*

This implies, that the all the software modules in your application, in this case one on each of the WebSphere Server instances, are being consolidated visually in the top-level objects that 'own' them.

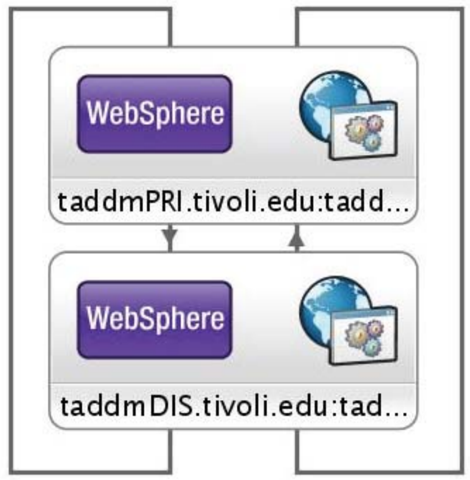
Note: This behavior is specific to the TADDM Topology Builder. In the TADDM database, the entire configuration topology and relationships including the low-level resources is stored. When this information is repurposed in related tools such as IBM SmartCloud Control Desk or Tivoli Business Services Manager, the entire application topology, including low-level resources and all their relationships, are considered.

For change processing, TADDM keeps track of all the minute details to all the tiny configuration items in your infrastructure, and propagates the discovered changes to the top-level resource. In this way, if the software module is replaced on one WebSphere Server, you will see the change reported on that specific WebSphere Server component (and even at the business application too) and you can drill into the change history to see the exact nature of the change.

Notice that for the physical topology, TADDM also uses the available relationship information to automatically include components that are not directly associated with the application. In this particular example, only computer system components are shown. TADDM did not find any information in its database related to neither the networking nor the storage infrastructures. The test environment is a locally hosted virtual environment, so that information is not discovered.

The Software Topology

If you, from the Discovered Components pane now choose **Actions** > **Show Software Topology** you will see a topology map of the application server components that are used in the application.



Note: Depending on the situation in the WebSphere environment at the time you discovered the environment, TADDM may not have been able to discover the relationships between the various WebSphere components.

If this is the case, you can either launch the Pulse Level 3 discovery again, or rediscover the WebSphere Cell.

As you see, the topology only shows the two top-level components that own the software modules. You also see that TADDM has discovered the two WebSphere Server instances are communication with one another. You do not see any relationships to any other components in the topology, because you have not yet added any other components to the business application.

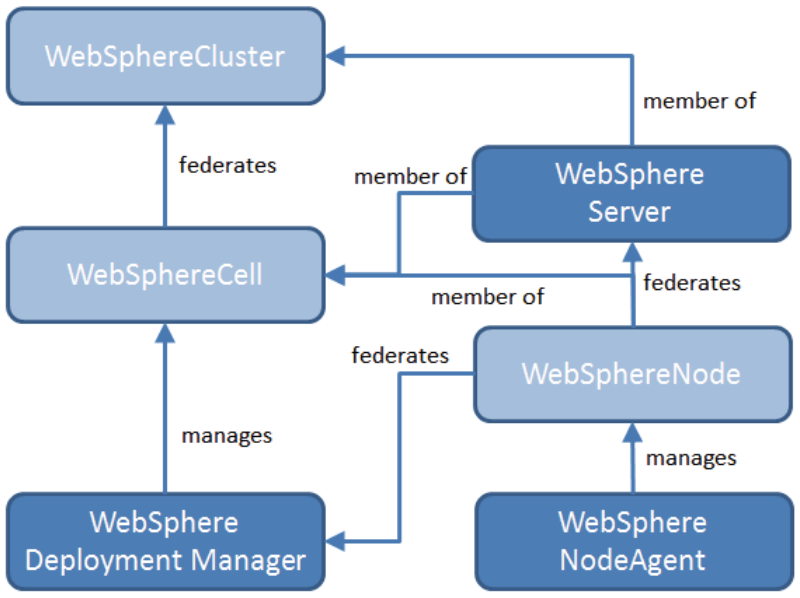
Even though the low-level resources are not visualized in the topologies, they serve an important role in the identification of the top-level resources to which they are related. By creating functional groups for the software modules you create a group of important components that later on can be referenced in order to identify the related resources. You can of course add the software module criteria to the rule that is used to add J2EE application servers, but that would lead to more complex MQL statements and introduce a risk of creating unintended results, and make maintenance of the application definition a nightmare.

Exercise 20: Identifying J2EE hosting infrastructure components

Even though there are many similarities between the vendor specific J2EE platforms on the market, the implementations differences are so significant, that the generic J2EE class hierarchy does not support them all. For that reason, the CDM provides vendor specific models for the most popular J2EE platforms such as WebSphere, WebLogic, Oracle, and JBoss.

Because of these differences it is necessary to depart from the stated intent to use generic definitions that are implementation technology transparent. This means, that you cannot clone the following example and expect them to work in your own environment – unless you use the same technology as in the example: WebSphere.

The CDM contains very nice UML diagrams that are helpful when trying to identify the key J2EE resources that are of interest from an operational or accounting perspective. These are the key WebSphere resource types that are relevant to your business application:



In this architecture the darker resources represent processes that must execute in the infrastructure in order to provide the wanted functionality. For that reason they should be included in the business application. The lighter resources are all logical definitions, and should generally not be included. However, if you include the WebSphereCluster definition in the business application, you will see in the topology maps that there a component that balances the load in the WebSphere servers. So for that simple reason, it is recommended to include the WebSphereCluster in the business application definition.

Adding implicitly related resources

Now, to create a rule that will add the WebSphereServer instances hosting the software modules to the business application inventory, you use the implicit relationships that are enforced by the Common Data Model. Only in our fantasy can resources like SoftwareModules exist on their own. Have you ever seen an ear-file float around on its own in outer space?

In the IT reality, SoftwareModules are always hosted on an application server. In this particular case, the application server technology used happens to be a WebSphere cluster that consists of a couple of WebSphere Servers. However, in the query that is a minor detail, since you can use the already identified SoftwareModules to locate the specific components that host them.

If you consult the CDM documentation, you will find that the SoftwareModule has an attribute named parent. The data type of this attribute is AppServer which means, that the content of the field is a reference to the application server to which the software module is deployed. The reference contains the component type and the guid (globally unique identifier) of the application server that hosts the software module. In the query you can use this implicit relationship to identify the applications servers that are part of your business application. In the CDM, AppServer is the parent class of the J2EEServer class, which is the parent of the WebSphereServer class - so you can use these three classes interchangeably. Since the purpose of this particular functional group is to contain WebSphere Server instances, you should use the WebSphereServer class instead of the AppServer class in the definition.

To identify the WebSphere Servers that host SoftwareModules you would include the following in the WHERE clause for the new query:

```
SoftwareModule.parent.guid equals WebSphereServer.guid
```

This query will select only the application servers that are defined as parents of any software module.

To reference the software modules that you, in the previous step, associated with the SoftwareModule functional group in the Trade application, you need to use this criteria in the WHERE clause:

```
Application.name == 'Trade' and FunctionalGroup.groupName ==
'SoftwareModule' and Application.guid == FunctionalGroup.app.guid and
exists(FunctionalGroup.members.guid == SoftwareModule.guid)
```

This fragment identifies the Trade application as well as the functional group named SoftwareModules, and combines the two because you may have functional groups with similar names in other applications. The functional group and application are joined using implicit relationships represented in the *app* attribute of the FunctionalGroup, similar to the one used to join software modules and application servers.

The select * FROM section defines which component types to select. In this case you need to specify WebSphereServer class since this the type of resources to want to populate into the new functional group.

So, the final query that you can test using the command-line API looks like this:

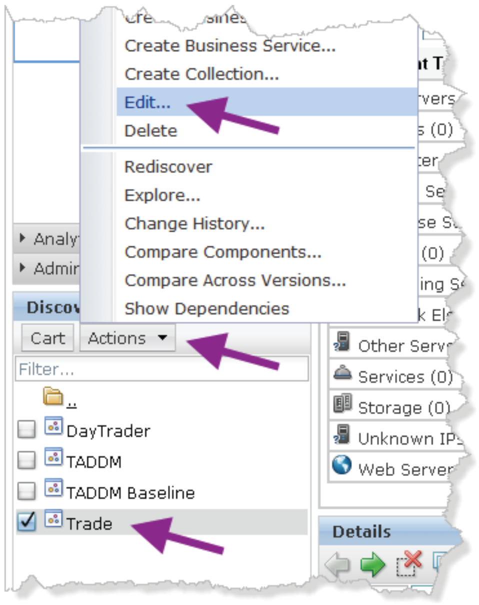
```
select * from WebSphereServer
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'SoftwareModule'
and Application.guid == FunctionalGroup.app.guid
and WebSphereServer.guid == SoftwareModule.parent.guid
and EXISTS(FunctionalGroup.members.guid == SoftwareModule.guid)
```

Notice that the EXISTS clause is placed at the end of the query. It has been observed that you may see MQL validation errors or unreliable results if the exists clause is not the last clause in where clause.

6.7 Adding WebSphere Server components

To create the rule that will populate a functional group named WebSphereServer with the J2EE servers that host your previously identified software modules complete these steps:

1. Launch the Trade business application definition in the Grouping Composer by selecting the Trade business application in the Discovered Components pane, and select **Action** > **Edit...**.

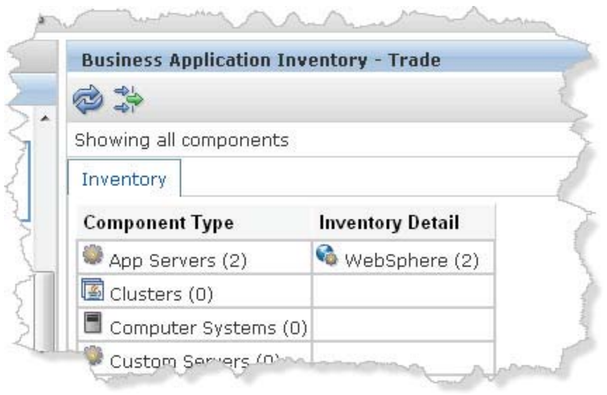


2. When the Grouping Composer opens, navigate to the Rules tab.
3. In the Rules page, click **New**, and provide the parameters shown in the following table:

Field	Value	Remarks
Rule Name	`WebSphereServer`	A label the help you identify the results returned by the query.
Functional group name	`WebSphereServer`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`WebSphereServer`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName == 'SoftwareModule' and Application.guid == FunctionalGroup.app.guid and WebSphereServer.guid == SoftwareModule.parent.guid and EXISTS(FunctionalGroup.members.guid == SoftwareModule.guid)`	

1. Click **Save** when you are ready so the rule is stored in the TADDM database.

If you save the application definition and view the Software Topology, you will not see any changes – because all you have done is to identify the J2EE Servers that were displayed because they host the software modules. However, if you look at the Business Application Inventory for the Trade application, you see that the two WebSphere Servers are now included in the inventory.


Business Application Inventory - Trade

Showing all components

Inventory

Component Type	Inventory Detail
App Servers (2)	WebSphere (2)
Clusters (0)	
Computer Systems (0)	
Custom Servers (0)	

Naturally, the WebSphere Server components will also show up in the Components tab of the Details pane.

Next, you will create a new functional group that contains the WebSphere Deployment Manager components related to the Trade application.

6.8 Adding the WebSphere Deployment Manager

To add the WebSphere Deployment Manager to a new functional group in the business application, you follow the standard method of referencing members of another functional group. This time you can use the implicit relationship chain *server > node > cell > deploymentManager* in order to identify the WebSphereDeploymentManager instance that manages the servers you already identified.

To follow the links of the chain, the WHERE clause must include this sequence:

```
WebSphereServer.node.guid == WebSphereNode.guid
and WebSphereNode.cell.guid == WebSphereCell.guid
and WebSphereCell.deploymentManager.guid ==
WebSphereDeploymentManager.guid
```

In addition, you must ensure that you select the WebSphereDeploymentManager component type in the query.

As usual you will be using the application and functional group components to identify the source resources.

The final query that can be tested in the command-line API, provided you have defined the Trade application and the WebSphereServer functional group, looks like this:

```
select * from WebSphereDeploymentManager
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'WebSphereServer'
and Application.guid == FunctionalGroup.app.guid
and WebSphereServer.node.guid == WebSphereNode.guid
and WebSphereNode.cell.guid == WebSphereCell.guid
and WebSphereCell.deploymentManager.guid ==
WebSphereDeploymentManager.guid
and EXISTS(FunctionalGroup.members.guid == WebSphereServer.guid)
```

When you have verified the query, you can create a new functional group named WebSphereDeploymentManager, by complete these steps:

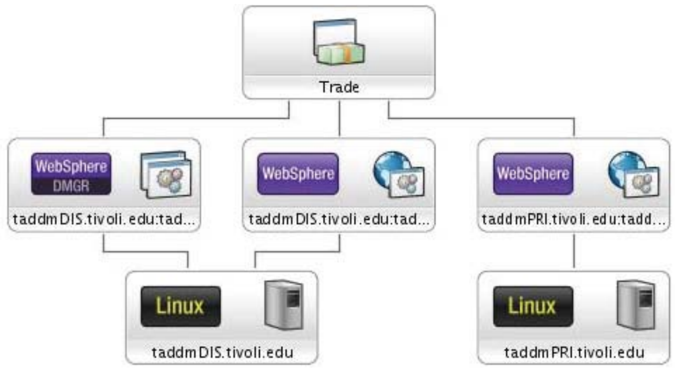
1. In the Rules tab of the grouping Composer, click **New**, and provide the parameters shown in the following table:

Field	Value	Remarks
Rule Name	`WebSphereDeploymentManager`	A label the help you identify the results returned by the query.
Functional group name	`WebSphereDeploymentManager`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`WebSphereDeploymentManager`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName == 'WebSphereServer' and WebSphereServer.node.guid == WebSphereNode.guid and WebSphereNode.cell.guid == WebSphereCell.guid and WebSphereCell.deploymentManager.guid == WebSphereDeploymentManager.guid and Application.guid == FunctionalGroup.app.guid and EXISTS(FunctionalGroup.members.guid == WebSphereServer.guid)`	

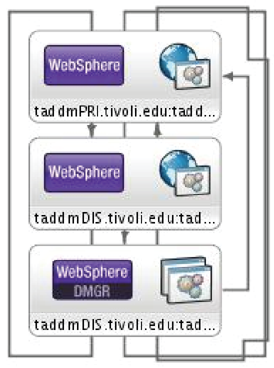
   After having created the new functional group, remember to click **Save**.

2. Click **OK** to save the application definition and close the Grouping Composer.

At this point you will notice that a WebSphere Deployment Manager has been added (as you would expect) as new component in the business application. The only significant changes to the topologies are that the WebSphereDeploymentManager and its relationships to the existing components are displayed.



If you open the Software Topology you also see the implicit (manages) relationships between the all three WebSphere Nodes and their related server components. This relationship is shown because the servers have become directly associated with the application, and not added as the result of top-level virtualization.


WebSphere
taddmPRI.tivoli.edu:tadd...
WebSphere
taddmDIS.tivoli.edu:tad...
WebSphere DMGR
taddmDIS.tivoli.edu:tad...

As before, you also see that the transactional connections between the servers have been mapped (the thicker lines with an arrowhead). This indicates that the components actually were communicating during the discovery.

6.9 Adding the WebSphere Node Agents

In WebSphere, the Node Agents represent processes that are used to facilitate communications between the Deployment Manager and the nodes local to the system hosting the Node Agent. A single Node Agent can support multiple nodes on the same system.

As such, Node Agents are critical resources that directly influence the operation of the WebSphere environment. Node Agents provide the means to remotely manage the various nodes hosted on a system, and facilitates collection of performance metrics used to manage the internal load balancing in WebSphere Clusters. For those reasons, Node Agents should be part of the application topology.

To add the WebSphere Node Agents to a new functional group in the business application, you basically repeat the previous example. You should use the functional groups named WebSphereServer and WebSphereDeploymentManager in an IN clause in the MQL query to identify the components that are related to the Node Agents. In addition, you could use the WebSphere Node as an intermediary to identify the WebSphere Node Agents that are associated with a WebSphereServer.

```
WebSphereServer.node.guid == WebSphereNode.guid and
WebSphereNode.nodeAgent.guid == WebSphereNodeAgent.guid
```

If you clone an existing query, you have to remember to modify the name of the functional group from which you read the members that are the source of the relationship.

Before creating the new functional group, try this query in the command-line API:

```
select * from  WebSphereNodeAgent
where Application.name == 'Trade'
and FunctionalGroup.groupName IN ('WebSphereServer',
'WebSphereDeploymentManager')
and WebSphereServer.node.guid == WebSphereNode.guid
and WebSphereNode.nodeAgent.guid == WebSphereNodeAgent.guid
and Application.guid == FunctionalGroup.app.guid
and EXISTS(FunctionalGroup.members.guid == WebSphereServer.guid)
```

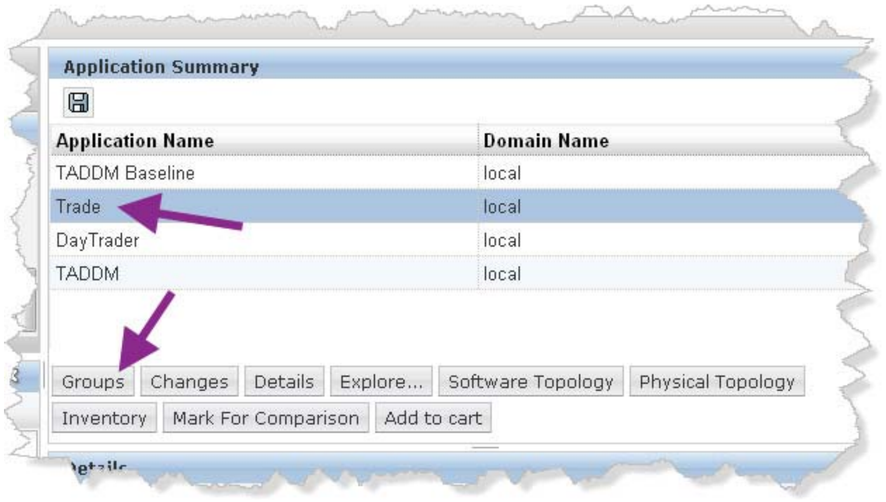
To create a new functional group named WebSphereNodeAgent, complete these steps:

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide these values:

Field	Value	Remarks
Rule Name	`WebSphereNodeAgent`	A label the help you identify the results returned by the query.
Functional group name	`WebSphereNodeAgent`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`WebSphereNodeAgent`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName IN ('WebSphereServer', 'WebSphereDeploymentManager') and WebSphereServer.node.guid == WebSphereNode.guid and WebSphereNode.nodeAgent.guid == WebSphereNodeAgent.guid and Application.guid == FunctionalGroup.app.guid and EXISTS(FunctionalGroup.members.guid == WebSphereServer.guid)`	

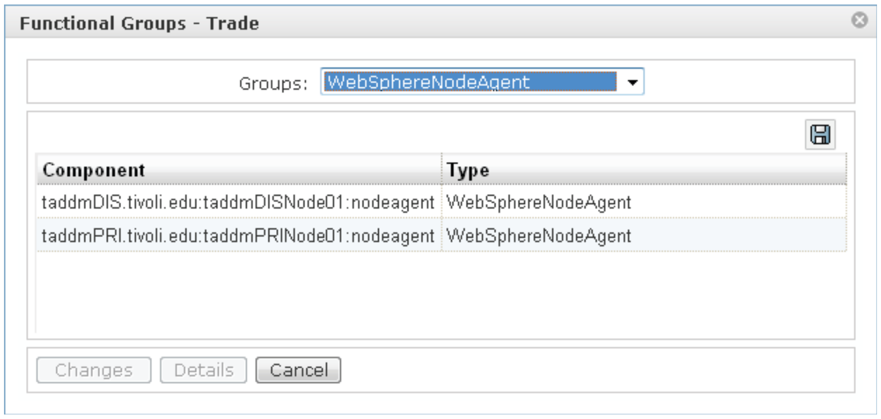
After having created the new functional group, remember to click Save.

4. To save the entire business application, force the validation of the rules, and populate the functional groups, click **OK**.
5. Now, to verify that the new functional group was populated, you can use the Application Summary. Navigate to **Analytics** > **Application Summary**.
6. When the Application Summary opens, select the Trade application, and click **Groups** in order to see which functional groups have been defined, and which resources are members.



Application Name	Domain Name
TADDM Baseline	local
Trade	local
DayTrader	local
TADDM	local

7. Select the group named WebSphereNodeAgent, and verify that you only see two agents.

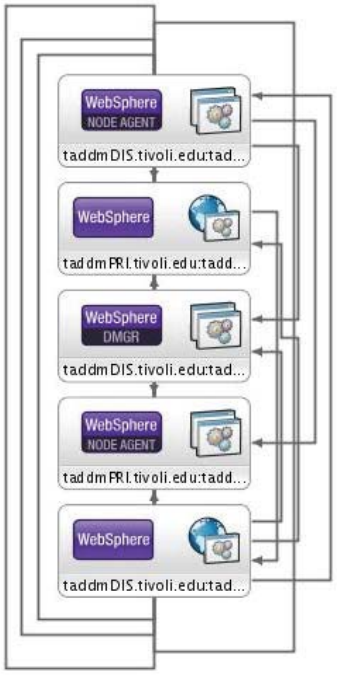


Component	Type
taddmDIS.tivoli.edu:taddmDISNode01:nodeagent	WebSphereNodeAgent
taddmPRI.tivoli.edu:taddmPRINode01:nodeagent	WebSphereNodeAgent

Even though you have three nodes (two WebSphere Servers, and one WebSphere Deployment Manager) you only see two WebSphere Node Agents. The Manager and one of the Servers are hosted on the same system, so they are supported by the same Agent.

Click **Cancel** to return to the Trade application summary,

8. Now, to see the software topology and click **Software Topology** to investigate if there have been any changes to the application server topology. The topology you see should be similar to this:


WebSphere NODE AGENT
taddmDIS.tivoli.edu:tad...
WebSphere
taddmPRI.tivoli.edu:tadd...
WebSphere DMGR
taddmDIS.tivoli.edu:tad...
WebSphere NODE AGENT
taddmPRI.tivoli.edu:tadd...
WebSphere
taddmDIS.tivoli.edu:tad...

You notice, that the two Node Agents have been added, and even if there are only five nodes in the topology, it had become harder to figure out whom is talking to whom.

You still have a few WebSphere components to add.

6.10 Adding WebSphere Clusters

WebSphere Clusters are managed by the WebSphere Deployment Manager through the WebSphere Cell so you can use the relationship from the members of the WebSphereDeploymentManager functional group to identify the cells they manage, and thereby the clusters that are part of this business application.

Just because the deployment manager manages a cluster it does not necessarily indicate that it is part of the business application, so you may have to add a clause to qualify the cluster names you want to include in the application thereby almost treading the cluster as a key identifying component.

The query to identify the clusters managed by the members of the WebSphereDeploymentManager group looks like this:

```
select * from WebSphereCluster where Application.name == 'Trade'
and FunctionalGroup.groupName  == 'WebSphereDeploymentManager'
and WebSphereCluster.parent.guid == WebSphereCell.guid
and WebSphereCell.deploymentManager.guid ==
WebSphereDeploymentManager.guid
and Application.guid == FunctionalGroup.app.guid
and EXISTS(FunctionalGroup.members.guid ==
WebSphereDeploymentManager.guid)
```

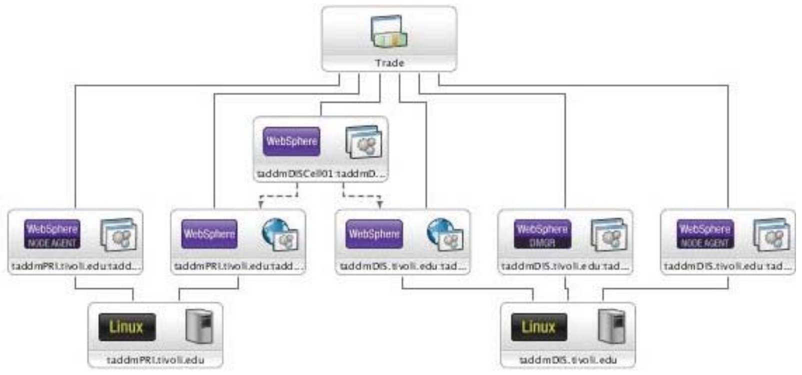
When you have verified the query, you can create a new functional group named WebSphereCluster, by completing these steps:

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide these values:

Field	Value	Remarks
Rule Name	`WebSphereCluster`	A label the help you identify the results returned by the query.
Functional group name	`WebSphereCluster`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`WebSphereCluster`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade'` `and FunctionalGroup.groupName == 'WebSphereDeploymentManager'` `and Application.guid == FunctionalGroup.app.guid` `and WebSphereCluster.parent.guid == WebSphereCell.guid` `and WebSphereCell.deploymentManager.guid == WebSphereDeploymentManager.guid` `and EXISTS(FunctionalGroup.members.guid == WebSphereDeploymentManager.guid)`	

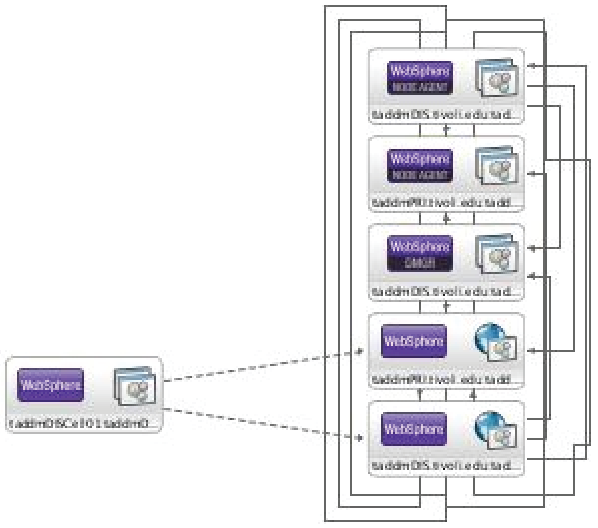
After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

At this point the application topology is taking shape. Take a quick look at the Physical Topology.



Hopefully you see, that an extra layer has been added between the application (at the top) and the application servers (second from the bottom). This layer represents the cluster layer, and as you can see, that the WebSphere Cluster has been added. By flowing the dotted lines from the cluster, you can even see which application servers are members of the cluster.

In the software topology, you can also see the implicit relationships between the cluster and the servers that are members of the cluster.



6.11 Adding additional WebSphere resources

The WebSphere platform contains many more logical resource types that can be added to the application. However the Cluster, Application Servers, Deployment Manager, and Node Agents represent the resources that are critical to the operation of the infrastructure. These components are also the main licensed software components, so by using this subset you should be covered from both a technical and an accounting perspective.

You should carefully consider whether to add additional components or not. Remember, TADDM is NOT an application decomposition tool so a significant amount of work will go into the development of queries, and it is doubtful which benefits you can gain. You should strive to keep the application topologies as simple as possible to enhance readability and allow the users of the topology information to easily interpret the information. Finally, remember, that all the technical details are leveraged in specialized tools for analyzing the impact of operational issues (Tivoli Business Services Manager) and change management (IBM SmartCloud Control Desk), so those specialized tools should be used for the purpose they are built for.

The primary mission of TADDM is to discover the configuration and relationships, and keep track of changes that have been applied to the infrastructure. The primary use of the application topology maps is to provide an overview, and highlight changed components, to assist root-cause identification, and problem determination.

Exercise 21: The database hosting infrastructure

Now that you have mapped the relevant J2EE resources into your application it is time to look at the backend database environment. This is made up of the top-level Database Management Systems (DBMS) and a variety of lower level components that represent buffer pools, table spaces, databases and so on.

6.12 Using explicit relationships to find the DBMS

The next rule you define will create a functional group named DatabaseServer. This functional group you add the DBMSs, known in the CDM as Database Servers, with which any of the J2EE Servers are communicating.

Remember that TADDM does not perform application decomposition, so it does not record relationships between specific EJB modules and specific databases. Instead the relationships that may be discovered are at the higher level - between J2EE Servers and Database Servers. However, the detailed information can be discovered by related monitoring tools such as ITCAM for Transaction Tracking, and the information this tool collects can be loaded into TADDM through a DLA. Without specific application tracking or decomposition software, TADDM can only see that the J2EE Server is in session with the DBM system, provided this session was active at the time of discovery.

Assuming that the session has been discovered, a Relationship component is created to reflect this explicit relationship. The relationship component has attributes that provide links to the source and the target of the relationship. In the MQL query used to identify the Database Servers that the J2EE Servers are related to, the source and target information is used. In this context, it is important to realize that the J2EE Server always is the source and the Database Server is the target.

The WHERE clause criteria needed to identify the DBMs that the J2EE servers are talking to can be defined like this:

```
J2EEServer,guid == Relationship.source.guid
and Relationship.target.guid == DatabaseServer.guid
```

This type of criteria naturally needs to be augmented with criteria to qualify the J2EEServer resources that are tested, and to achieve that you use the standard group membership to select the members of the WebSphereServer functional group.

The final, testable, query should look like this:

```
select * from DatabaseServer where  Application.name == 'Trade'
and FunctionalGroup.groupName == 'WebSphereServer'
and Application.guid == FunctionalGroup.app.guid
and J2EEServer.guid == Relationship.source.guid
and Relationship.target.guid == DatabaseServer.guid
and exists(FunctionalGroup.members.guid == J2EEServer.guid)
```

Note: Notice that the J2EEServer component type is used instead of WebSphereServer. The WebSphereServer is, like other component types representing different J2EE servers, a subclass of the J2EEServer class, so using the parent class will make this query applicable to most environments.

6.13 Adding Database Servers

To define the rule that includes the DatabaseServers in the application topology, complete these steps:

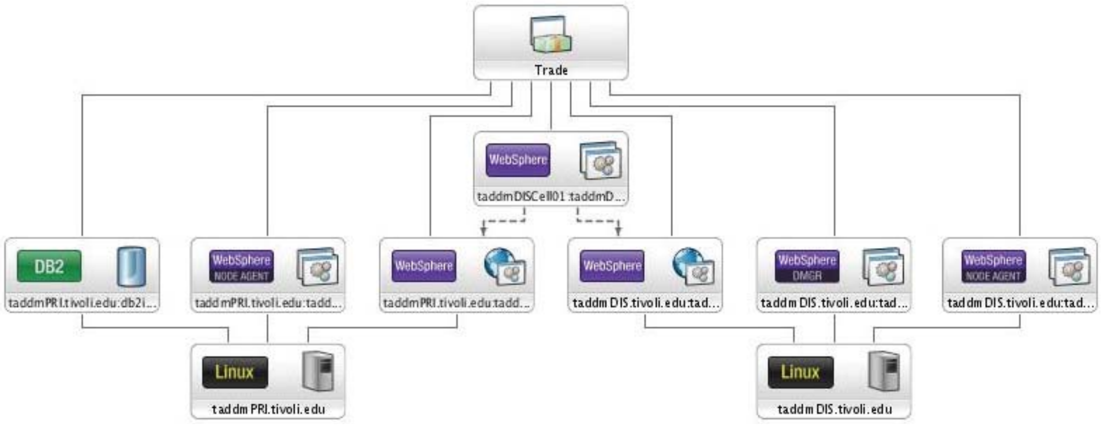
1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide these values:

Field	Value	Remarks
Rule Name	`DatabaseServer`	A label the help you identify the results returned by the query.
Functional group name	`DatabaseServer`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`DatabaseServer`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.

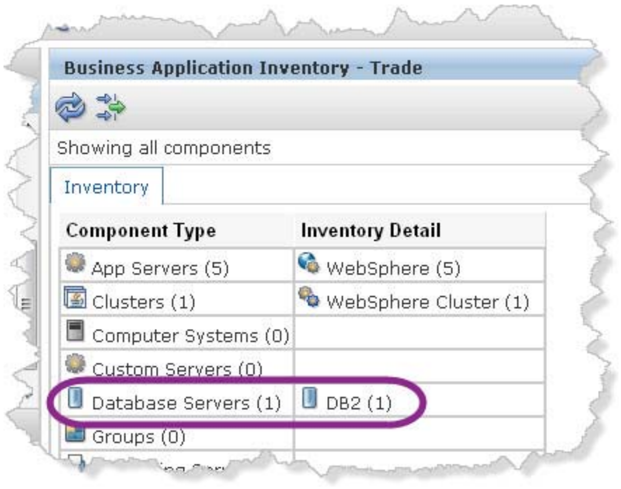
(WHERE)	`Application.name == 'Trade'`
	`and FunctionalGroup.groupName == 'WebSphereServer'`
	`and Application.guid == FunctionalGroup.app.guid`
	`and J2EEServer.guid == Relationship.source.guid`
	`and Relationship.target.guid == DatabaseServer.guid`
	`and exists(FunctionalGroup.members.guid ==`
	`J2EEServer.guid)`	

   After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

4. If you view the Physical Topology, you will see that one or more database management systems have been added.


Trade
WebSphere
taddmDISCell01 :taddmD...
DB2
taddmPRI.tivoli.edu:db2i...
WebSphere NODE AGENT
taddmPRI.tivoli.edu:tadd...
WebSphere
taddmPRI.tivoli.edu:tadd...
WebSphere
taddm DIS.tivoli.edu:tad...
WebSphere DMGR
taddm DIS.tivoli.edu:tad...
WebSphere NODE AGENT
taddm DIS.tivoli.edu:tad...
Linux
taddm PRI.tivoli.edu
Linux
taddm DIS.tivoli.edu

5. You can also take a look at the Business Application Inventory by selecting the Trade application, and choose **Action** > **Business Application Inventory** from the Discovered Components pane, which shows all the components that are associated with the application.



Business Application Inventory - Trade

Showing all components

Inventory

Component Type	Inventory Detail
App Servers (5)	WebSphere (5)
Clusters (1)	WebSphere Cluster (1)
Computer Systems (0)	
Custom Servers (0)	
Database Servers (1)	DB2 (1)
Groups (0)	

As you can see, the DB2 Instance has been added. However, you also see that the computer systems supporting the infrastructure are not automatically included in the inventory. Hopefully this does not come as a surprise because you have not specifically added computer systems to the business applications. The only reason you see them in the Physical Topology is because TADDM tries to help by automatically identifying the hosting system for the application server resources.

6.14 Adding databases

Individual databases are treated by TADDM in a similar fashion to SoftwareModules. Databases are logical components that are shielded from the outside world by a management system which provides services such as authorization, connectivity, registry, and storage to the database. When TADDM discovers a DatabaseServer, these services are consulted in order to maintain the configuration information in the TADDM databases. Processes that consume information in a database, communicate with the DBM, so for that reason, TADDM cannot discover the relationship between a process and a specific database within the DBM.

However, TADDM registers all the databases that are known to a DatabaseServer. This implies, that if you know the specific name of the database(s) your business application uses, you can either add an additional marker module group, in which you specify the exact name of the databases you are looking for.

To create a query that lists all the databases managed by the members of the DatabaseServer group you just identified, try this query:

```
select displayName from Db2Database
where Db2Database.parent.guid == DatabaseServer.guid
and Application.name == 'Trade'
and FunctionalGroup.groupName == 'DatabaseServer'
and Application.guid == FunctionalGroup.app.guid
and exists(FunctionalGroup.members.guid == DatabaseServer.guid)
```

This query will list ALL the databases hosted on the selected Database Serves. If you want to qualify the databases you include, simply add a criteria that filters in only the database names you want. For example, to include only the databases that start with the string trade (any case) you should add the following:

```
and lower(Db2Database.name) starts-with 'trade'
```

Remember that the EXISTS clause has to be the last one in the WHERE clause. You can submit the following MQL query from the command-line API to find the DB2Instance that hosts the TRADE database used by the WebSphere Servers that host the trade.ear SoftwareModule.

```
select name from Db2Database
where Db2Database.parent.guid == DatabaseServer.guid
and Application.name == 'Trade'
and FunctionalGroup.groupName == 'DatabaseServers'
and lower(Db2Database.name) starts-with 'trade'
and Application.guid == FunctionalGroup.app.guid
and exists(FunctionalGroup.members.guid == DatabaseServer.guid)
```

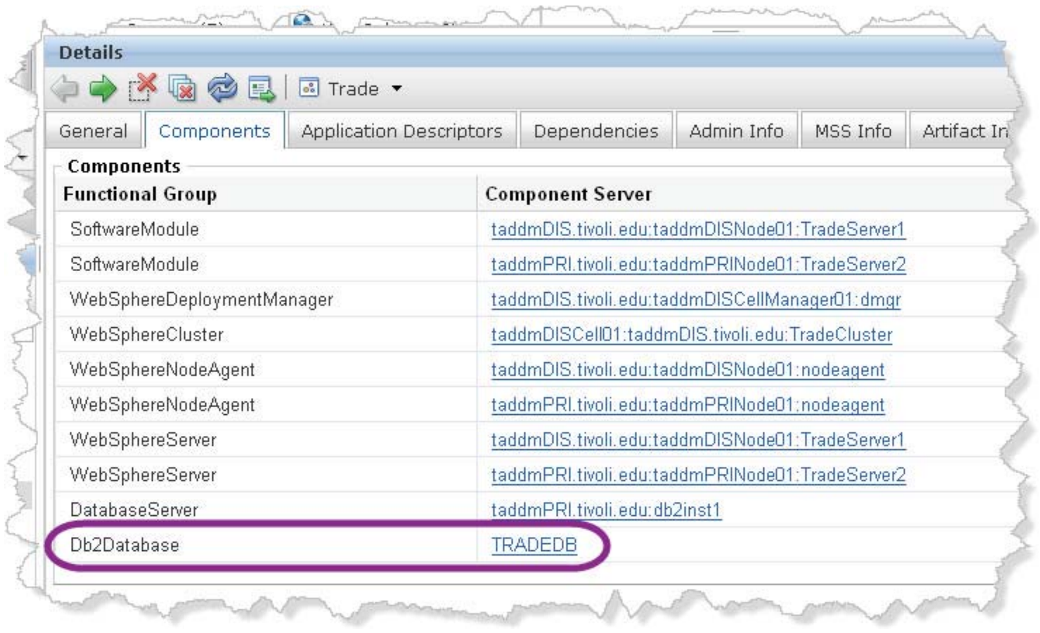
To define the rule that includes the database management systems in the application topology, complete these steps:

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide these specifications:

Field	Value	Remarks
Rule Name	`Db2Database`	A label the help you identify the results returned by the query.
Functional group name	`Db2Database`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`Db2Database`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Db2Database.parent.guid == DatabaseServer.guid and Application.name == 'Trade' and FunctionalGroup.groupName == 'DatabaseServer' and lower(Db2Database.name) starts-with 'trade' and Application.guid == FunctionalGroup.app.guid and exists(FunctionalGroup.members.guid == DatabaseServer.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

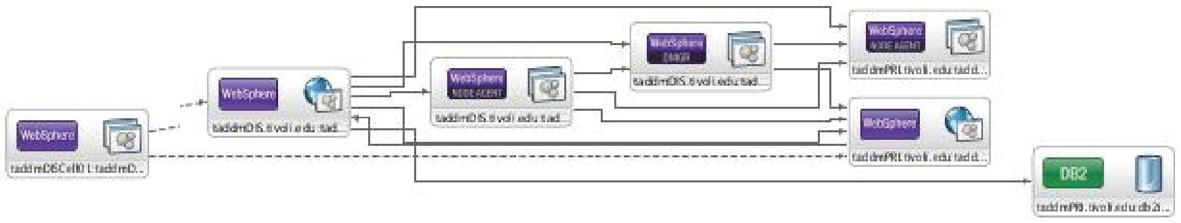
4. When you look at the Components tab of the Details pane for the Trade business application you will see that the TRADEDB database has been added.



Components

Functional Group	Component Server
SoftwareModule	taddmDIS.tivoli.edu:taddmDISNode01:TradeServer1
SoftwareModule	taddmPRI.tivoli.edu:taddmPRINode01:TradeServer2
WebSphereDeploymentManager	taddmDIS.tivoli.edu:taddmDISCellManager01:dmgr
WebSphereCluster	taddmDISCell01:taddmDIS.tivoli.edu:TradeCluster
WebSphereNodeAgent	taddmDIS.tivoli.edu:taddmDISNode01:nodeagent
WebSphereNodeAgent	taddmPRI.tivoli.edu:taddmPRINode01:nodeagent
WebSphereServer	taddmDIS.tivoli.edu:taddmDISNode01:TradeServer1
WebSphereServer	taddmPRI.tivoli.edu:taddmPRINode01:TradeServer2
DatabaseServer	taddmPRI.tivoli.edu:db2inst1
Db2Database	TRADEDB

5. If you look at the Software Topology for the Trade application, you will notice, that only the Database Server is visible:


WebSphere
taddmDISCell01:taddmD...
WebSphere
taddmDIS.tivoli.edu:tad...
WebSphere NODE AGENT
taddmDIS.tivoli.edu:tad...
WebSphere DMGR
taddmDIS.tivoli.edu:tad...
WebSphere NODE AGENT
taddmPRI.tivoli.edu:tad.d...
WebSphere
taddmPRI.tivoli.edu:tad.d...
DB2
taddmPRI.tivoli.edu:db2i...

As was the case for the software modules, databases are not top-level objects, and as such they are not shown in the topology, but are represented by their nearest top-level predecessor. In this case, the predecessor is the Db2 Instance, which was already added to the business application by the Database Servers rule.

Do not despair if you, as in the example above, only see a relationship between one of the WebSphereServer instances and the Db2Instance. This is a perfectly normal situation. Apparently traffic was only flowing between a single WebSphereServer and the DatabaseServer at the time of discovery.

This completes the exercise related to adding database resources to the business application. You noticed how the explicit relationships were used to identify the database server that is used by the WebSphere Servers.

Exercise 22: Web Infrastructure

You have now associated the back-end database and J2EE server tiers of the business application, so all that is left is the front-end web tier.

To associate Web resources you adopt the same approach as was the case for identifying the database servers to which the J2EE servers connect. You use the explicit relationships to identify the relationship between Web servers and the WebSphere servers and create a new functional group of Web servers. Once this group has been built, you can use the implicit relationships to locate any resources hosted on the Web serves you want to include in the business application.

Naturally, you can also apply your knowledge about the application and define specific virtual hosts as identifying resources and locate the Web servers that host those virtual hosts. This approach requires application insight, but will help you to find the exact Web servers that support your application in case multiple web servers (supporting multiple applications) connect to the WebSphere infrastructure.

Once you have identified the Web servers that front-end your J2EE application servers, you may want to locate the proxy servers that most likely are implemented to protect your environment.

6.15 Adding Web servers to the business application

To find the Web servers that front-end your J2EE applications, you need to rely on the discovered relationships, similarly to the way you found the database servers. In this case, since the Web servers route requests to the J2EE servers, the Web servers will be the source of the relationship and the J2EE servers are the targets.

The following query uses the discovered relationships to identify the Web servers that are routing requests to the J2EE servers that are members of a functional group named WebSphereServer, which is part of the Trade application:

```
select * from WebServer
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'WebSphereServer'
and Relationship.source.guid == J2EEServer.guid
and Relationship.target.guid == WebServer.guid
and Application.guid == FunctionalGroup.app.guid
and exists(FunctionalGroup.members.guid == J2EEServer.guid)
```

If you have not discovered any transactions between the Web servers and the J2EE serves, you may want to use specific virtual hosts as marker modules. To find the Web server that host a specific virtual host, you can clone this sample query:

```
select * from WebServer where
ApacheVirtualHost.serverName contains '<your search string>'
and  ApacheVirtualHost.parent.guid == WebContainer.guid
and WebContainer.parent.guid == WebServer.guid
```

This query can be used as an alternative to the one based on discovered relationships to populate a functional group with specifically named WebServer. This functional group can in turn be used to find the proxy serves in your environment.

Complete the following steps to define the rule that includes Web servers for which communications between the Web server and any member of the WebSphereServer functional group has been discovered:

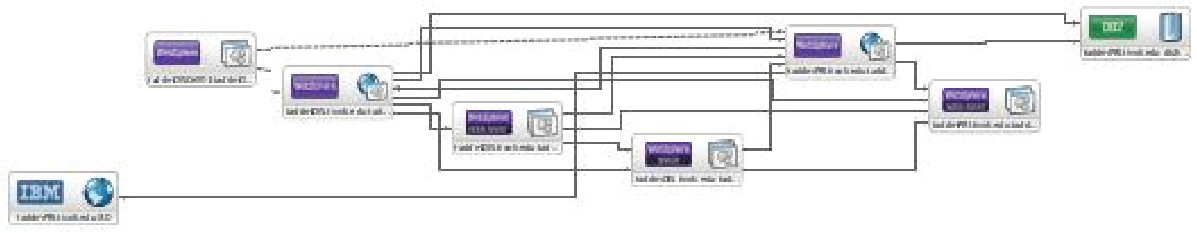
1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide these specifications:

Field	Value	Remarks
Rule Name	`WebServer`	A label the help you identify the results returned by the query.
Functional group name	`WebServer`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`WebServer`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.

(WHERE)	`Application.name == 'Trade'`
	`and FunctionalGroup.groupName == 'WebSphereServer'`
	`and  Relationship.source.guid == J2EEServer.guid`
	`and Relationship.target.guid == WebServer.guid`
	`and Application.guid == FunctionalGroup.app.guid`
	`and exists(FunctionalGroup.members.guid ==`
	`J2EEServer.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

4. If you look closely at the software topology you will see that a single Web Server has been added, and that it communicates with one or both WebSphere Server instances. The results may vary depending on the actual traffic flow during discovery.



You see that as you add layer upon layer the business application takes shape. If you recall the Trade application architecture diagram, all that is missing for the application itself is the reverse proxy server that front-ends the whole application.

6.16 Adding proxy servers to the business application

It is not uncommon to use reverse proxies in front of the application infrastructure. These reverse proxies can server many purposes including authentication, package routing, monitoring or various protection services.

To identify the proxy servers that route requests to any of the Web servers that support the application, you need to look for transactional relationships between Web servers. Since the proxy servers send requests to the application web server, and the application web server return results to the proxy server, you should have discovered two relationships: one with the proxy server as the source and one in which the application web server is the source. However, in some situations the results may be routed through a different path, so to play it safe you should base the identification of proxy servers on the relationship in which the proxy server is the target.

Note: Remember that unless you have imported specific application transaction relationships from other solutions, TADDM does not know the application level dependencies. For example, TADDM cannot see from which virtual host the relationship originates which virtual host receives the request.

Try this query to find the proxy servers that route requests to any of the web serves in the WebServer functional group of the Trade application.

```
select displayName from WebServer where
Application.name == 'Trade'
and FunctionalGroup.groupName == 'WebServer'
and Relationship.target.guid == AppServer.guid
and Relationship.source.guid == WebServer.guid
and Application.guid == FunctionalGroup.app.guid
and exists(FunctionalGroup.members.guid == AppServer.guid)
```

Notice how the AppServer resource type (it is the parent class of the WebServer class – among others) is used to qualify the members of the WebServer functional group. This is necessary in order to be able to differentiate the filtering resources and the result resources in a situation like this where the result and filtering resources are of the same type.

To define the rule which includes the proxy web servers that forwarding requests to the application web servers, all you need to do is to complete these steps:

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.

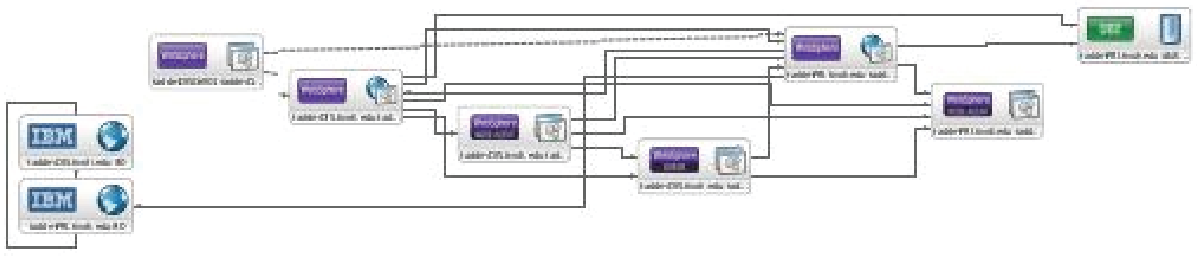
3. Provide these values:

Field	Value	Remarks
Rule Name	`ProxyWebServer`	A label the help you identify the results returned by the query.
Functional group name	`ProxyWebServer`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`WebServer`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName == 'WebServer' and Relationship.target.guid == AppServer.guid and Relationship.source.guid == WebServer.guid and Application.guid == FunctionalGroup.app.guid and exists(FunctionalGroup.members.guid == AppServer.guid)`	

Note that the query to identify the specific instances uses AppServer instead of WebServer for the group members because the query returns WebServer guids.

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

4. As usual, the Software Topology has been updated:



Notice that there is one relationship in each direction between the Web Servers.

This completes the exercise related to adding the three-tiered application components to a business application. You have experienced now you can use implicit relationships to identify parent-child relationships between logically connect components, and how you can use explicit relationships to identify specific components pairs that at some point in time (during discovery) have communicated with one another. In addition, you have seen several examples on how to structure your MQL queries to reference members of existing functional groups when identifying related components. Deliberately it has been attempted to make these examples as generic as possible, so you can use them in your own environment with a minimum of effort.

Exercise 23: Services

In addition to the application server resources you want to associate with a particular business application, you should also consider adding services to the business application. Many of your application server resources may leverage common IT services such as FTP, DNS or NFS services that are made available to them. By adding these services to the application, you highlight the fact that the business applications depend on the services. This highlight is only necessary to visualize the dependencies in the TADDM topology maps, and to add the service components to the business application inventory. When exporting TADDM configuration and relationship information to IBM SmartCloud Control Desk and IBM Tivoli Business Services Manager, the services dependencies are automatically included based on the discovered explicit relationships.

6.17 Adding DNS services to the business application

When TADDM discovers that an application server is in session with a DNS server the information is stored as an explicit relationship in the TADDM database. To include the DNS services that are used by the application servers in your business applications you can query the Relationship table similarly to the way you related database and web servers to the application. The nature of the relationship is recorded as a ServiceDependency and it should go without saying that the source of the relationship is the application server, and the target is the DNS server.

The query needed to test the population of a DNSServer functional group should be similar to this:

```
select * from DNSService
where Application.name == 'Trade'
and FunctionalGroup.groupName IN ('WebSphereServer',
'WebSphereDeploymentManager', 'WebSphereNodeAgent', 'DatabaseServer',
'WebServer','ProxyWebServer')
and Relationship.target.guid == DNSService.guid
and Relationship.source.guid == AppServer.guid
and Application.guid == FunctionalGroup.app.guid
and exists(FunctionalGroup.members.guid == AppServer.guid)
```

To define the rule that includes the database management systems in the application topology, complete these steps:

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide the following values:

Field	Value	Remarks
Rule Name	`DNSService`	A label the help you identify the results returned by the query.
Functional group name	`DNSService`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`DNSService`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName IN ('WebSphereServer', 'WebSphereDeploymentManager', 'WebSphereNodeAgent', 'DatabaseServer', 'WebServer','ProxyWebServer') and Relationship.target.guid == DNSService.guid and Relationship.source.guid == AppServer.guid and Application.guid == FunctionalGroup.app.guid and exists(FunctionalGroup.members.guid == AppServer.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

4. If you open the Software Topology for the Trade application, you will see that the DNS Service has been added at the bottom, without any dependencies. Naturally, it will also appear in the Business Application Inventory, and the Physical Topology.

The fact that you do not see the dependencies in the Software Topology is normal TADDM behavior. The algorithms behind the building of the topologies automatically hide connections to IT Services such as DNS, DHCP, LDAP, NSF, FTP, and SMB services. If all the dependencies to any of these services are shown, the sheer number of connections would (almost) render the topology unreadable.

6.18 Adding LDAP services to the business application

To add the services provided by the LDAP servers your business application components use, you should basically follow the standard pattern. First you identify the components (application servers, computer systems, or other component types) that are the sources of the explicit relationship, and then you specify the type (LDAPService) that is the target of the relationship. If any relationships exist, the LDAPService instances will be added to our business application.

The query needed to test the population of an LDAPService functional group should be similar to this:

```
select * from LDAPService where Application.name == 'Trade'  and
FunctionalGroup.groupName IN ('WebSphereServer',
'WebSphereDeploymentManager', 'WebSphereNodeAgent', 'DatabaseServer',
'WebServer','ProxyWebServer', 'ComputerSystem') and
Relationship.target.guid == LDAPService.guid and Application.guid ==
FunctionalGroup.app.guid and exists(FunctionalGroup.members.guid ==
Relationship.source.guid)
```

This should be used to include only resources from specific groups. If you want to include resources in any functional group associated with the application, you can simply remove the *FunctionalGroup.groupName IN (…)* clause. An all-inclusive query would look like this:

```
select * from LDAPService where Application.name == 'Trade'  and
Relationship.target.guid == LDAPService.guid and Application.guid ==
FunctionalGroup.app.guid and exists(FunctionalGroup.members.guid ==
Relationship.source.guid)
```

Note: Notice that because the source types vary, the guid of the members of the functional groups is used to define the source of the relationship. This way you can mix and match several different object types that are members of different functional groups.
Also notice that the ComputerSystem group is included in the IN clause. This functional group is created in a short while. The ComputerSystem group has been included to prepare the LDAPService rule to also include LDAP services used by ComputerSystems that are specifically added to the business application.

To define the rule that includes the LDAP services in the application topology, complete these steps:

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide the following values:

Field	Value	Remarks
Rule Name	`LDAPService`	A label the help you identify the results returned by the query.
Functional group name	`LDAPService`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`LDAPService`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.

(WHERE)	`Application.name == 'Trade'`
	`and FunctionalGroup.groupName IN ('WebSphereServer',`
	`'WebSphereDeploymentManager', 'WebSphereNodeAgent',`
	`'DatabaseServer', 'WebServer','ProxyWebServer',`
	`'ComputerSystem')`
	`and Relationship.target.guid == LDAPService.guid`
	`and Application.guid == FunctionalGroup.app.guid`
	`and exists(FunctionalGroup.members.guid ==`
	`Relationship.source.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

At this point, you should be familiar with the process to verify that the LDAPService was added to the business application. Notice that LDAP connections may not have been active when the exercise environment was discovered.

6.19 Adding more system services

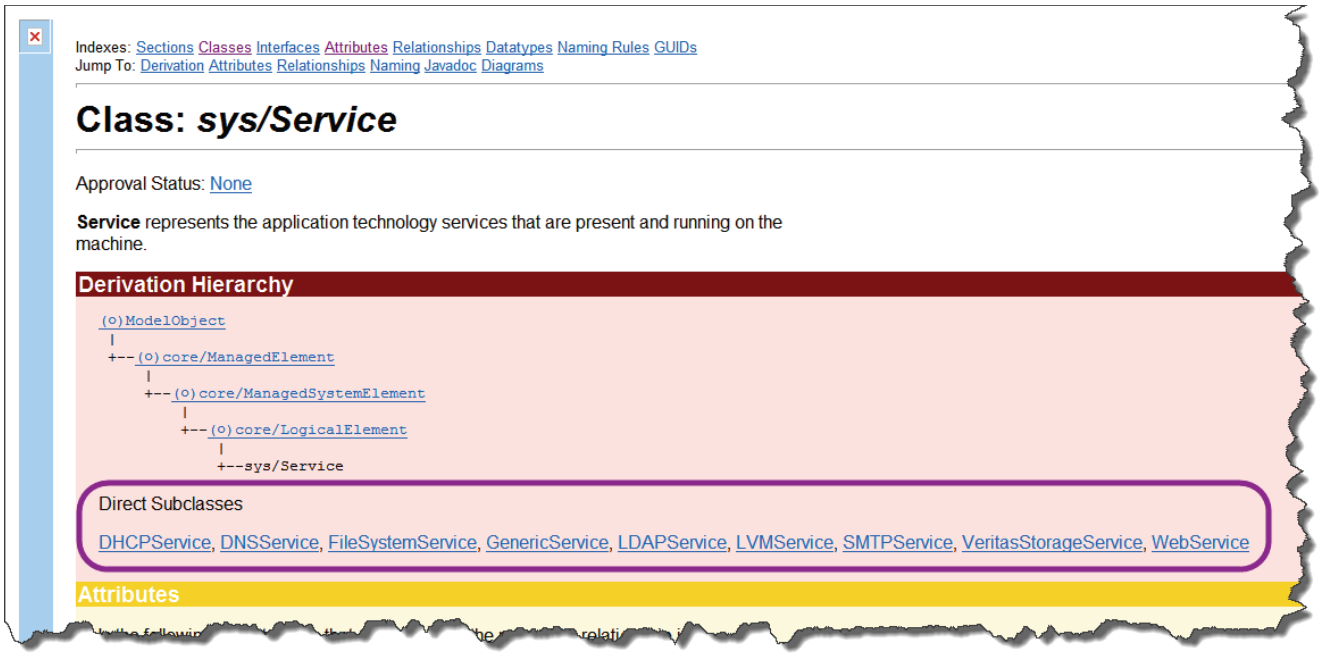
Using the DNS definition as an example, you can easily create additional functional groups that, for example, represent security, mail, file, web, or other services.

To see the specific resource names of the IT services that are defined in the Common Data Model, open the Service class definition, and take a look at its derivation hierarchy.

From a browser in the exercise environment, open the following web address:

```
http://taddmPRI.tivoli.edu:9430/cdm/datadictionary/cdm/classes/sys/Ser
vice.htm
```

Just beneath the derivation hierarchy you find the Direct Subclasses.



Indexes: Sections Classes Interfaces Attributes Relationships Datatypes Naming Rules GUIDs
Jump To: Derivation Attributes Relationships Naming Javadoc Diagrams

Class: *sys/Service*

Approval Status: None

Service represents the application technology services that are present and running on the machine.

Derivation Hierarchy

```
(o)ModelObject
 |
 +--(o)core/ManagedElement
     |
     +--(o)core/ManagedSystemElement
         |
         +--(o)core/LogicalElement
             |
             +--sys/Service
```

Direct Subclasses

DHCPService, DNSService, FileSystemService, GenericService, LDAPService, LVMService, SMTPService, VeritasStorageService, WebService

Attributes

Notice that some services, such as DNS, LDAP, and WebServices can be associated directly with the application server components, while other services (DHCP, file and storage) are related to the computer system hosting the application servers.

Exercise 24: Computer systems and system services

Typically, you will not need t add ComputerSystems to the application definition. Remember, all the application servers you identify have an implicit relationship to an OperatingSystem that resides on a ComputerSystem. TADDM uses these relationships to identify the ComputerSystems that support a given ApplicationServer. When TADDM information is propagated to other tools that consume TADDM data, for example Jazz for Service Management, IBM SmartCloud Control Desk, or Tivoli Business Services Manager, the implicit relationships are maintained, and therefore available to the functions provided in tools. However, for reporting and accounting purposes, you may wish to specifically include ComputerSystems so they appear in the Business Application summary

Computer systems can easily be identified by joining the ComputerSystem guid and AppServer.host.guid attributes. Because the AppServer component type is the parent of all of the server components defined in the functional groups, this component type can be used to identify all the application server systems. The same is true for the ComputerSystem class which is the parent of all computer system types defined in the Common Data Model.

To include multiple functional groups in the query use the IN clause to include all the functional groups that contains server resources.

The computer systems that are used by the business application can be identified by this query:

```
select guid from ComputerSystem
where Application.name == 'Trade'
and FunctionalGroup.groupName IN ('WebSphereServer',
'WebSphereDeploymentManager', 'DatabaseServer', 'WebServer',
'ProxyWebServer')
and ComputerSystem.guid == AppServer.host.guid
and exists(FunctionalGroup.members.guid == AppServer.guid)
```

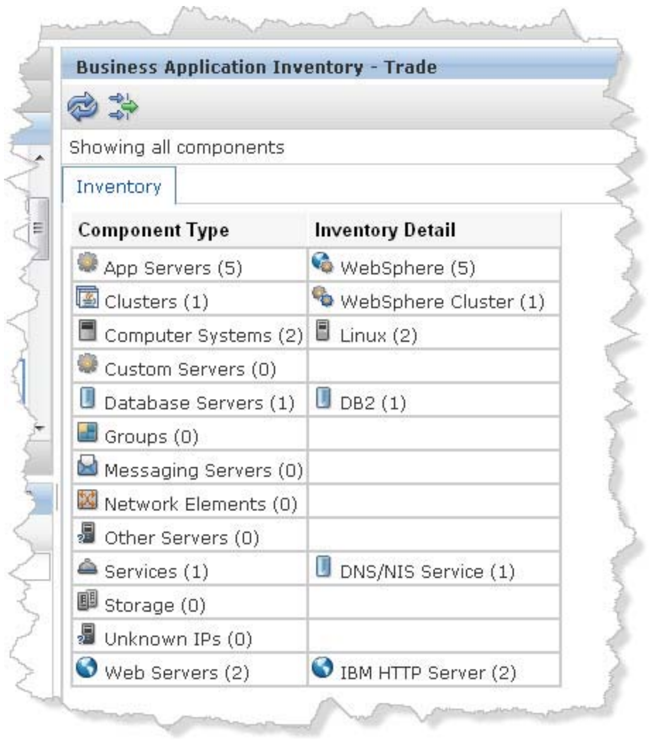
To define the rule that includes the computer systems in the application topology, complete these steps

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide the following specifications:

Field	Value	Remarks
Rule Name	`ComputerSystem`	A label the help you identify the results returned by the query.
Functional group name	`ComputerSystem`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`ComputerSystem`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName IN ('WebSphereServer', 'WebSphereDeploymentManager ', 'DatabaseServer', 'WebServer', 'ProxyWebServer') and ComputerSystem.guid == AppServer.host.guid and exists(FunctionalGroup.members.guid == AppServer.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

4. To verify that the ComputerSystem functional group has been populated, you can either use the Groups function of the Application Summary, or simply look directly in the Business Application Inventory. Since computer systems have now been added specifically to the application, they will appear in the inventory.



Showing all components

Component Type	Inventory Detail
App Servers (5)	WebSphere (5)
Clusters (1)	WebSphere Cluster (1)
Computer Systems (2)	Linux (2)
Custom Servers (0)	
Database Servers (1)	DB2 (1)
Groups (0)	
Messaging Servers (0)	
Network Elements (0)	
Other Servers (0)	
Services (1)	DNS/NIS Service (1)
Storage (0)	
Unknown IPs (0)	
Web Servers (2)	IBM HTTP Server (2)

As you can see, the computer systems, along with all the other top-level resources you have defined as members of the application, are shown in the Business Application Inventory.

6.20 Including virtualization platforms

When TADDM has discovered your computer systems – as well as the virtualization platforms – virtual systems can easily be identified through the virtual attribute. If the value of this attribute is *true*, the system is a virtual system, and the reference to the virtualization platform can be found in the *hostSystem* attribute.

To include information about the virtualization platforms that support your business application all you need to do is to create a new functional group that is populated by a rule similar to this:

```
select * from UnitaryComputerSystem
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'ComputerSystem'
and UnitaryComputerSystem.guid == ComputerSystem.hostSystem.guid
and ComputerSystem.virtual
and exists(FunctionalGroup.members.guid == ComputerSystem.guid)
```

Naturally, you could have augmented the rule to identify computer systems to include the virtualization platforms, but populating a specific functional group with virtualization platforms makes it easier for you identify the virtualization platforms.

The example above uses the functional group that includes ComputerSystems. As discussed, you may decide to not include ComputerSystems from the application definition. In that case you need to use the *host* attribute of the ApplicationServer resources as a pointer to the ComputerSystem – or similar references for other resource types. To create a rule that identifies resources related to a ComputerSystem, without including the ComputerSystems as members of the application, you can apply a rule similar to the following example that identifies virtualization platforms for Application Servers. Notice how *AppServer.host* is used as a pointer to the ComputerSystem resource, and *AppServer.host.virtual* is a reference to the *virtual* attribute of the ComputerSystem resource type. Also notice that the candidate application servers are selected from all members of all functional groups associated with the application.

```
select * from UnitaryComputerSystem where Application.name == 'Trade'
and Application.guid == FunctionalGroup.app.guid and
AppServer.host.virtual and UnitaryComputerSystem.guid ==
AppServer.host.hostSystem.guid and exists(FunctionalGroup.members.guid
== AppServer.guid)
```

Note: If you decide to add a specific functional group for virtualization platforms, remember to modify the `FunctionalGroup.groupName ==` or `FunctionalGroup.groupName IN(…)` criteria in the rules used to associate services, and other components, with the application.

As it has been mentioned a number of times, many resources are automatically related when the TADDM information is imported into supporting tools such as TBSM and IBM SmartCloud Control Desk. This also applies to virtualization platforms, so you would only add these to the business application to support reporting and administration.

Unfortunately the exercise environment does not include discoverable virtualization platforms that you can discover, so you can verify that the query works.

6.21 Adding NFS services to the business application

To add NFS Services to your business application, follow the same procedure as for adding DNS Services. You only have to replace the service type (DNSService) with NSFService, and reverse the relationship so that the NFSService is the source.

The query needed to test the population of a NFSServices functional group should be similar to this:

```
select * from NFSService
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'ComputerSystem'
and Relationship.source.guid == NFSService.guid
and Relationship.target.guid == ComputerSystem.guid
and Application.guid == FunctionalGroup.app.guid
and exists(FunctionalGroup.members.guid == ComputerSystem.guid)
```

To define the rule that includes the NFS services in the application topology, complete these steps:

1. Ensure that you have opened the Grouping Composer Rules tab for the Trade business application.
2. Click **New**, to create a new rule.
3. Provide the following specifications:

Field	Value	Remarks
Rule Name	`NFSService`	A label the help you identify the results returned by the query.
Functional group name	`NFSService`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`NFSService`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName == 'ComputerSystem' and Relationship.source.guid == NFSService.guid and Relationship.target.guid == ComputerSystem.guid and Application.guid == FunctionalGroup.app.guid and exists(FunctionalGroup.members.guid == ComputerSystem.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

6.22 Adding SMB services to the business application

To add SMB Services to your business application, follow the same procedure as for adding NFS Services. You only have to replace the service type (NFSService) with SMBService.

For your reference, the query needed to test the population of a SMBService functional group should be similar to this:

```
select * from SMBService
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'ComputerSystem'
and Relationship.source.guid == SMBService.guid
and Relationship.target.guid == ComputerSystem.guid
and Application.guid == FunctionalGroup.app.guid
and exists(FunctionalGroup.members.guid == ComputerSystem.guid)
```

The exercise environment does not contain components that provide SMB services, so no further actions are needed.

Exercise 25: (for reference) Adding network devices to the business application

Normally when working only with TADDM, it is not necessary for you to add network devices to the business application definition. In the physical topology view, TADDM will automatically use the known relationships to add any networking devices to which the computer systems hosting the application servers are connected.

If you intend to leverage the business application definitions in related solutions such as Tivoli Business Service Manager or IBM SmartCloud Control Desk, you do not have to worry about the addition of networking devices either. During the import of TADDM data into the related solutions, the inherent relationships are also used to identify the networking devices on which the business application relies.

The only case, in which you need to add network devices to a business application, is when you want to build a complete inventory of the business application to be used for providing access, or to support administrative tasks such as accounting, or reporting.

The following demonstrates how you can build MQL Queries to include bridges, switches, and routers to your application. Most modern routing devices contain both bridging and routing capabilities, so you will see that most of the devices you add will be available in both the functional groups for both.

6.23 Bridges

Bridges, or switches, are used to connect the L2Interface in a computer system to the network. In OSI terms Level 2 is the DataLink layer, so the connection between a computer system and a bridge/switch represents logical connection to the network infrastructure.

To add information about the bridges to which the L2Interfacs on the individual computer systems are connected, you need to focus on the Segment object type in the TADDM database. The Segment represents a network segment, which is a portion of a computer network wherein every device communicates using the same physical layer - that is a Layer 2 Network.

Because of the restriction in the MQL language that you can only use one exists clause, you must build a helper functional group that contains segments, and use this group to identify all the bridges that are members of the segments.

Try this query to identify all the segments that are connected to all the computer systems in the application:

```
select * from Segment,
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'ComputerSystem'
and Segment.guid == L2Interface.segment.guid
and L2Interface.parent.guid == ComputerSystem.guid
and ComputerSystem.type == 'ComputerSystem'
and exists(FunctionalGroup.members.guid == ComputerSystem.guid)
```

Once you have verified that this query produces a list of segments that includes the L2Interfaces in your application, you need to perform a second query that extracts the UnitaryComputerSystem instance that hosts the Bridge function:

```
select * from UnitaryComputerSystem
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'Segment'
and UnitaryComputerSystem.type == 'Bridge'
and L2Interface.parent.guid == UnitaryComputerSystem.guid
and L2Interface.segment.guid == Segment.guid
and exists(FunctionalGroup.members.guid == Segment.guid)
```

Notice that the second query extracts the L2Interfaces to inspect from the Segments and not the ComputerSystems functional group.

To define the rule that includes the segments in the application topology, all you need to do is to add a new rule with the following specifications:

Field	**Value**	**Remarks**
Rule Name	`Segment`	A label the help you identify the results returned by the query
Functional group name	`Segment`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application
(select * FROM)	`Segment`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName == 'ComputerSystems' and Segment.guid == L2Interface.segment.guid and L2Interface.parent.guid == ComputerSystem.guid and ComputerSystem.type == 'ComputerSystem' and exists(FunctionalGroup.members.guid == ComputerSystem.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

Creating this helper group as a functional group within the business application may not necessarily be the implementation you want. This approach creates a functional group in the application that contains segments, which have no significance what-so-ever for the high-level management or administration of your application. To avoid that the segments become members of the business application, you can define the L2Segments group a collection, instead of a functional group. If you do this, the collection will still be populated by the topologyBuilder, but the members will not be associated with the business application.

When the Segment group has been populated, you can define the rule that populates the Bridge functional group:

Field	Value	Remarks
Rule Name	`Bridge`	A label the help you identify the results returned by the query.
Functional group name	`Bridge`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`UnitaryComputerSystem`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.

(WHERE)	`Application.name == 'Trade'`
	`and FunctionalGroup.groupName == 'Segment'`
	`and UnitaryComputerSystem.type == 'Bridge'`
	`and L2Interface.parent.guid == UnitaryComputerSystem.guid`
	`and L2Interface.segment.guid == Segment.guid`
	`and exists(FunctionalGroup.members.guid == Segment.guid)`	

After having created the new functional group, remember to click **Save**, and **OK** to force TADDM to process your changes.

If you implemented the Segment group as a collection, substitute FunctionalGroup in the where clause with Collection, and replace the leading criteria:

```
Application.name == 'Trade'
and FunctionalGroup.groupName == 'Segment'
```

with:

```
Collection.name == 'Segment'
```

Finally, replace the trailing exists clause with this:

```
exists(Collection.members.guid == Segment.guid)
```

When you view the physical topology after having added network devices to the application, you will see that under the covers, TADDM will include the next layer of network devices that connect to the back-plane of your bridges and switches. This is normal behavior of the algorithms that are used to generate the physical topology.

6.24 Routers

Routers are devices that are used to in the network layer to provide inter-network routing of data packets. In order to be able to send data to a system on another network, the computer system must be configured to forward the packets to be sent to a router, along with information about the intended recipient. This means, that routers are critical to almost any business application because applications typically are expected to provide responses to requests submitted by users or other application components.

To identify the routers that are used by the components in a business application, you start by identifying the computer systems that are associated with the application, and use the implicit relationships to locate the router through the ComputerSystem > IpInterface > IpNetwork > IpRoute > Router path in the database.

Try this query using the command-line API:

```
select * from UnitaryComputerSystem
where Application.name == 'Trade'
and FunctionalGroup.groupName == 'ComputerSystem'
and UnitaryComputerSystem.guid == Router.parent.guid
and Router.guid == IpRoute.parent.guid
and IpRoute.destination.stringNotation == IpNetwork.subnetAddress
and IpNetwork.guid == IpInterface.ipNetwork.guid
and IpInterface.parent.guid == ComputerSystem.guid
and ComputerSystem.type == 'ComputerSystem'
and exists(FunctionalGroup.members.guid == ComputerSystem.guid)
```

To define the rule that includes the routers in the application topology, all you need to do is to add a new rule with the following specifications:

Field	**Value**	**Remarks**
Rule Name	`Router`	A label the help you identify the results returned by the query.
Functional group name	`Router`	Name of the group. This should be standardized so you can easily cut-and-paste your definitions to another application.
(select * FROM)	`UnitaryComputerSystem`	The object class, as known in the CDM of the resources that are returned. If you query multiple resource types, the one to return must appear first in the list.
(WHERE)	`Application.name == 'Trade' and FunctionalGroup.groupName == 'ComputerSystem' and UnitaryComputerSystem.guid == Router.parent.guid and Router.guid == IpRoute.parent.guid and IpRoute.destination.stringNotation == IpNetwork.subnetAddress and IpNetwork.guid == IpInterface.ipNetwork.guid and IpInterface.parent.guid == ComputerSystem.guid and ComputerSystem.type == 'ComputerSystem' and exists(FunctionalGroup.members.guid == ComputerSystem.guid)`	

As you saw after having added the bridges, the physical topology will include the networking devices to which the newly added routers are connected, so you will see an additional layer of networking infrastructure devices.

7 Automating application mapping

As you worked your way through the previous section you may have realized that applications more or less follow the same structure (or pattern). If you think about the environment in which you work, you may also realize that your organization only have a limited number of application patterns you need to support.

As described in the previous sections, the rules that support application mapping can be divided into three main categories:

Marker rules	Queries that identify the marker modules for which you want to inspect the relationships to identify related resources,
Implicit rules	Queries that uses the implicit relationships between the marker modules and related resources to identify resources to be included in the application.
Explicit rules	Queries that leverages the explicit relationships that are stored in the TADDM database to identify resources that are related to one another so that resources that are related to a marker module, or any other application component, can be identified.

For most applications a general set of rules can be applied to capture all the resources related to the marker modules. This means, that besides the internal references to application specific functional groups, the implicit and explicit rules are identical across most application definitions. This implies that if the names of the functional groups are standardized, the implicit and explicit rules can easily be cloned from one application to the next. All that needs to be changed in the process is the name of the application.

However, for marker rules it is not that easy. The marker rules are the bootstraps of the application mapping process so by nature the specifics of these rules vary.

7.1 Standardizing marker rules

Even though marker rules must be different in order to identify the identifying resources of different applications, they can be standardized, if you rely on relationships.

In essence, marker rules are used to specify the lowest-level resources that uniquely identify a business application. Typically these resources are web services, J2EE applications, modules, or resources, virtual hosts, web modules and databases. The marker modules are identified. based combinations of specific attributes, such as name, state, role, category, location and other meta data, user data, or extended attributes. To determine which marker modules support which applications you need to understand the business and the business applications – and that cannot be automated. This type of information can typically be obtained from delivery managers, application owners, developers, deployment scripts and logs, or application documentation.

Once the marker modules are identified, they must be discovered so that they are represented in the TADDM database. At this point you can query the database to investigate exactly which attributes are available for each type of marker module, and use this information to specify unique rules for each application.

Instead of specifying the marker module attributes directly in the application definition rules, consider registering the dependency between the business application and the marker modules in the TADDM database. If you do this, you can create generic marker rules that use your registration to identify the marker modules, and kick off the mapping process.

So, by manually adding relationships between an application and the specific marker modules it is identified by, your marker rules become standardized, and rely only on the name of the business application. A sample marker rule would look like this:

```
Select * from J2EEModule where Application.name ==
'MyBusinessApplication' and ((Relationship.source.guid ==
Application.guid and Relationship.target.guid == J2EEModule.guid and
Relationship.type ends-with 'Uses') or (Relationship.target.guid ==
Application.guid and Relationship.source.guid == J2EEModule.guid and
Relationship.type ends-with 'UsedBy'))
```

The rule has to return objects of a specific type, so you should use the top-most persistent resource types to specify the type. You can find the super classes for specific resource types in the Common Data Model. The J2EEModule used in the example above is the super-class of most J2EE related marker module types. However, J2EEModule is a subclass of SoftwareModule, so this class could also have been used, and would have been more generic.

You also see in the example that you can specify multiple relationships in the query. This allows you to identify the marker module in a single query independently of the direction of the relationship.

The example above demonstrates how you can use specific relationship types to identify marker modules. Based on your implementation you can choose to look for specific relationship types or not. Some of the relationship types that may be relevant are: assignedTo, basedOn, contains, dependsOn, implements, memberOf, owns, provides, relates, supports, usedBy, and uses. Depending on the flexibility you want to build into the marker rules, these relationship types can be used to specify the direction and nature of the relationship.

7.2 Registering marker module relationships

In order to register a relationship in the TADDM database, both the source and target of the relationship must have been discovered in advance.

Even though you can manually create relationships from the Data Management Portal, it cannot be used to add relationships that involve non top-level resources. You will need a small API-based utility to create the relationships you need.

Considering both of these restrictions, you should design the program to identify resources that are part of the relationship based on a query instead of using specific guids. This way you can schedule the program to run periodically, and automatically add relationships for marker modules as they are discovered. Naturally, you program can also be designed as a DLA that extracts information from your development environment. As a matter of fact, a DLA that extracts software definitions from the IBM Rational Asset Manager is available. You can use this information in combination with the discovered SoftwareModule information to define marker modules based on meta data related to the information in your definitive software library.

Another source of information to determine marker modules is your transaction or web services monitoring tool – for example ITCAM for Transactions or ITCAM for SOA. Typically these tools decompose transactions or SOA interactions, and can provide a wealth of transactional dependencies that might be needed to identify your marker modules. The standard TADDM discovery does not include information at that level of detail, so you have to rely on an external solution. However, based on the standard TADDM discovery, the TADDM database contains transactional dependencies between the application servers that host the marker modules, and this information is used in the explicit relationship rules that map the application.

Relationships between marker modules and applications

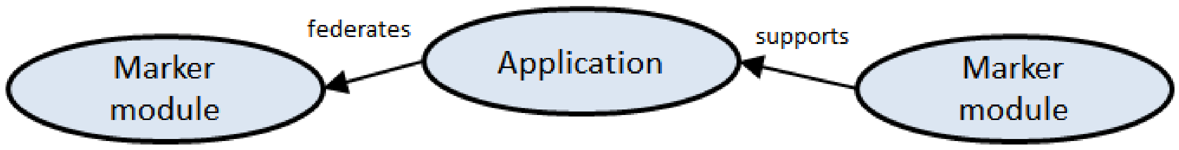
If you look into the relationship details described in the Common Data Model, you will find that TADDM support more than a 100 different types of relationships. This implies that you can convey a very specific meaning in the relationship which can be used to register and use different uni-directional relationships for different purposes. Under the covers TADDM applies different behavior to different relationships. In most cases target resources are removed when the source resource are deleted from the TADDM database – which does not seem unreasonable. However, the *federates* relationship does not demonstrate this behavior, so when associating an application with marker modules the application should federate the marker modules. The federates relationship is defined in the CDM as follows:

> *A relationship in which one entity (the source) logically contains one or more other entities (the targets) but if the source is deleted, the targets remain.*

If you want to use a relationship that points from the marker module to the application, you should use the *supports* relationship. The CDM defines the supports as follows:

> *A relationship where one entity (the source) does not directly provide, but is needed for the proper operation of or the availability of another entity (the target).*

So, to define marker modules relationships that can be used to relate for example specific ear files to an application, you can use federate relationships from the application to the marker module, or supports relationships from the marker module to the application – or both. Which type of relationships to use is entirely up to you, and depends on how you wish to use the relationships in your MQL rules.

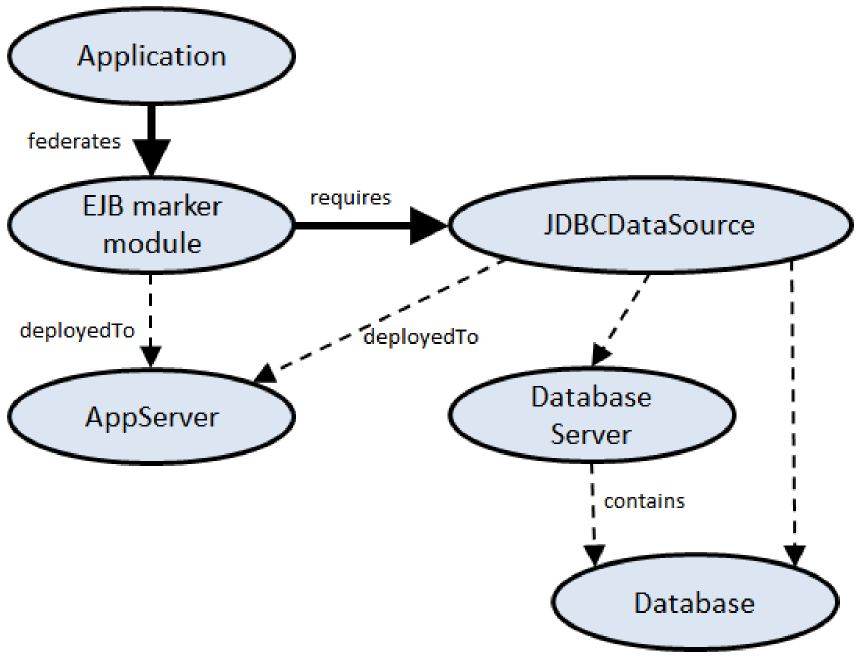


Marker to resource dependencies

If you want, you can also add relationships between maker modules and other resources. In some cases you want to include resources for which no relationships have been discovered in the application. In this case, you can manually define relationships between your marker modules and the related resources, and use these relationships in your rules.

Most often this is needed when dealing with components that share resources such as J2EE environments. TADDM does not discover which JDBCDataSource is used by an EJB, and is therefore not able to relate a specific database instance to the EJB.

If you manually add a *requires* relationship from the EJB to the JDBCDataSource definition, you can use this to identify the JDBCResourceProperties (not shown) that defines the specific database instance, and the database server SAP that is used by the application in which the EJB is a marker module.



It is important to understand, that the dependency between the EJB and the DataSource is defined in the EJB itself, and can only be discovered by a J2EE specific monitoring tool that can look deep into the J2EE runtime environment. Also understand that even though TADDM discovers and saves a Transactional Dependency between the AppServer and the Database Server, you cannot rely on this because it only relates the server resources. The Transactional Dependency can represent the connection from any EJB or servlet on the AppServer that connects to any database hosted on the Database Server. In this case, the J2EE pooling mechanisms handle the physical connections and obfuscate the logical dependencies. This is why you must crate the EJB > DataSource relationship manually – or rely on transaction information from your J2EE monitoring tool.

Business processes and applications

You may also consider applying a process layer of definitions between the business application and the resources that are discovered.

In reality, application components such as Web Services, servlets and EJBs implement certain business processes and business services. Business process and services can be modeled in TADDM to represent the specific activities that are included in a business application. This means that by defining business processes and services to TADDM, and relating them to specific application resources, you can create application models or patterns by using the process definitions rather than the specific resources that implement the processes. This technique is perhaps the most accurate way to model your business applications – but it requires that you define the business processes and services to TADDM, and relate them to application resources.

However, Business processes and service definitions can be loaded into TADDM from DLAs that are provided with the ITCAM for Transactions, and ITCAM for SOA tools. These tools are aware of the runtime topologies of your application infrastructure, and can therefore provide valuable information in relation to mapping applications in TADDM.

7.3 Cloning business applications

As you have seen, the resources that are part of an application are grouped in functional groups. Each functional group is populated by a rule. The rule defines a query into the TADDM database to identify the resources that becomes members of the functional group.

By now you should also understand that rules contain application specific information, and that most rules can be designed so generically that only the name of the application changes from one instance of a rule to the next. You have already seen, in *Exercise 20: Identifying J2EE hosting infrastructure components* on page 66, how you can save resources in a functional group, and reference the members in this group from another rule. If you apply techniques like this and a naming standard for the functional groups, your rules will be so generic that they can be used in any application that applies to a specific pattern.

For example, a rule that is used to identify databases or database servers that are used by EJBs in a functional group named EJBs is identical from one application to the next (except for the application name).

This means, that you will save yourself a lot of time and debugging (due to typos) if you could easily copy one application definition to the next – or even better, export an existing application to an external format so it can be used as the pattern for future application definitions. Unfortunately, the current versions of TADDM (7.2.1.5 and 7.2.2.0) do not provide facilities to copy or clone group definitions. This means that you have to copy, paste, and modify 10-20 rules every time you need to define a new business application.

You might consider creating a utility that can clone applications so you do not have to perform error prone manual processes. This utility should include options to cloning relationships, as well as admin info, user data and extended attributes related to a group.

8 Summary

We hope that you feel that the knowledge you have gained from completing these exercises will be beneficial for your future work. Remember, the definition of business applications in not only relevant for TADDM related tasks. The business application definitions you create will most likely be leveraged by specific tasks in related solutions such as

- Business Services management in TBSM
- Event enrichment in IBM NetCool Impact
- Incident and Problem Management in IBM SmartCloud Control Desk.
- Configuration, Change, and Release Management in IBM SmartCloud Control Desk.

You should also keep in mind, that TADDM discovery does not perform low level transaction discovery or decomposition. IBM Tivoli provides other tools that discover transaction and SOA Service dependencies, and the data collected by these tools can be loaded into the TADDM database through a Discovery Library Adapter.

Network topology information can also be imported from IBM Tivoli Network Manager (ITNM) through a DLA. If you have ITNM installed, you might consider relying on this tool, rather than having to maintain access to network equipment in TADDM.

9 Additional information

This section contains background information that you may find useful to better understand how TADDM discovery works, how the data are stored, and how you can use the MQL query language to access the information in the TADDM database.

9.1 Discovering detailed information

A prerequisite for using low-level details to identify application components and map them into Business Applications is that the resources are represented in the TADDM database. This implies that you must ensure that they are discovered.

For technology-specific low-level details such as DB2 databases and schemas and WebSphere war and ear modules these details are discovered by the Level 3 discovery. In the case of WebSphere, you may have to configure the sensor to perform deep discovery. What is discovered, and how to configure sensors is documented in the Sensors Guide which is available at: http://pic.dhe.ibm.com/infocenter/tivihelp/v46r1/index.jsp?topic=%2Fcom.ibm.taddm.doc_721fp3%2Fwelcome_page%2Fwelcome.html.

The built-in sensors in TADDM are responsible for discovering the low-level details for some types of resource. For other types of resources, typically represented by custom server templates, you must configure your custom server sensors to gather the specific information you are interested in. Custom server sensors are by default not configured to collect any related resources, so before you can use specific attributes and relationships to define Business Applications you must ensure that they are discovered and stored in the TADDM database.

There are several ways to augment the information that is gathered for custom servers, all of which are controlled from the custom server template definition. By default, only the process definition (process name, arguments, environment, and listening port), the implicit relationships to OS and ComputerSystem instances, and explicit relationships to resources with which the process is communicating are discovered. From the custom server template, you can add controls to gather these additional details:

- Configuration file information and its content
- Software Modules
- Application Descriptor Files

In addition, by placing specific files in the file system on the TADDM Discovery Server you can populate specific attributes with the value provided by a script, create your own configuration files at runtime, and even execute Jython or javascripts on the target system to gather additional information and populate attributes. From these scripts, you can also create new objects in the TADDM database and define relationships between the discovered resource and these new objects. This allows you to, for example, create low-level resources that uniquely define your application resource.

In the following it will be demonstrated how set up the necessary definitions to augment the Tomcat custom server template to:

- Discover ear and war software modules
- Discover the server.xml and web.xml configuration files and their content

9.2 Specifying customer servers

Custom servers are used to instruct TADDM to discover resources for which there are no built-in sensors. TADDM does this by identifying all server processes (active processes that are listening on one or more ports), and matching these processes to process signatures which are defined in a custom server template. During discovery, the GenericServerSensor is responsible for identifying the processes (using lsof, and netstat) and the CustomAppServerSensor tries to match each entry in the list of discovered server processes to a custom server template.

The custom server template defines a set of criteria that are used in the matching process. You can use these process attributes to identify instances of a particular custom server:

- Program name
- Process environment
- Process arguments
- Listening Port

In addition, the custom server template specifies the label and type of application server that will be registered when a match is found.

To create a custom server template that is used for identifying Tomcat servers that host the TADDM engine, you can, for example, define a custom server template with the following specifications:

Name	`TADDM_Tomcat`
Type	`J2EEServer`

Criteria	`Program name ends-with 'java'`
	`Argument contains 'catalina.home'`
	`Argument equals '.*-DCOLLATION.*'`
	`Environment variable $COLLATION_HOME is-not-null`
	`Port is greater than 9000`

When a process that matches these specifications is found, TADDM creates a new J2EEServer, and assigns a type of TADDM_Tomcat. During discovery, TADDM identifies the ComputerSystem, OperatingSystem, IpInterfaces, and IpAddresses related to the process, so all the necessary details required to honor the naming rules and implicit relationships are known.

9.3 Discovering configuration files and software modules

In addition to discovering the process that represents the J2EEServer, you can also discovery and capture configuration files, software modules, and application descriptor files that are related to the instance of the TADDM-Tomcat server.

The term *configuration file* is used to describe files that control the configuration and behavior of the resource. Often you want to keep track of these files, not only to keep their content in the CMDB, but also to identify changes to the files, and be able to use the information in the files to assign the resource they control to the proper Business Applications.

To include configuration files in the discovery of custom server, you add the file specifications (pathname/filename) in the custom server template. During discovery, TADDM will resolve any variables you may have used, and register the files with the custom server.

Software module is a generic term that applies to all types of application servers. In the Common Data Model, a software module is defined as:

SoftwareModule	Represents packaged components that are deployed in a Software Container. Software Module - is the thing that is "deployed" and "installed". The deployment is putting files and such on the correct file system or machine. The installation is doing the steps necessary to put the software in an executable state, like unpacking JAR files, creating registry entries, and so on. The reason modules are confused with products is because, in simple cases, there is a one-to-one correspondence between the product and the set of files that need to be deployed. Taking Microsoft Word as an example, there is probably only one set of files needed to install Word, so there is one "module". If it is needed to install three pieces of a product on three computers, then there is one product, but three modules.

Using this definition, almost anything can be software modules. For example, an application or module deployed to a J2EEServer or a J2EEDomain is a software module. As a matter of fact, in this case it is represented in the CDM as a subclass of SoftwareModule named J2EEDeployedObject.

To discover software modules with a custom server, you specify the software modules in a similar fashion to the way configuration files are specified.

In addition to configuration files and software modules, you can also include application definition files in the custom server template. Application definition files are xml files that contain information about how resources and software modules are related to business applications. Prior to TADDM version 7.2.1 these files provided the only way to automatically associate components with business application instances. However, since the application description files needed to be reside on the discovered systems, the use of application descriptor files for application mapping was hard to adopt. The new query-based facilities delivered with the grouping composer in TADDM version 7.2.1 provide much greater flexibility, and ease-of-use.

9.4 Extending custom server discovery

As described, the standard custom server discovery registers application servers, their basic attributes, and any related configuration files, software modules, or application descriptor files that have been specified in the custom server template.

You may want to apply attribute values for some of the attributes that are not automatically populated, or perhaps even overwrite some of the discovered attribute values. This can be achieved by providing a simple file that contains instructions on how to extend the discovery. In this file, you can easily instruct the discovery to:

- Apply a value to an attribute based on the output from a command which is executed on the discovered system
- Create a configuration file and populate it with the output form a command that is executed on the discovered system
- Execute a javascript or Jython script on the TADDM discovery server, and manipulate attributes and relationships. This includes running commands on the discovered system to gather information.

For example, to provide a value for the KeyName attribute, you would simply add a line to the file that controls the extension of the TADDM-Tomcat custom sensor similar to this:

```
CMD:keyName = echo “This is my name”
```

In a similar fashion, the following line can be used to create a new configuration file named servlets and populate it with the names of the servlets defined to the Tomcat server.

```
CMD:CONFCONTENT servlets = grep servlet-name
$COLL_PROGPATH$/../install/WEB_INF/web.xml
```

The last option, invoking a script, is in many ways the most interesting. It allows you to launch a custom script that runs in thread on the TADDM discovery server. This implies that the script has full access to the TADDM environment, and can access all the resources in the CMDB. In addition, from within the TADDM discovery server environment you can execute commands on the system that is being discovered, in order to retrieve information that can be used to populate attribute values and relationships. To invoke a script, you would add a line to the custom server template extension file similar to this:

```
SCRIPT:scrpt-name
```

If you prefer, you can also execute one or more operating system-specific (bash or cmd) scripts directly on the discovered system. Once the results from these scripts are parsed back to the TADDM discovery server, you can use a Jython script to analyze the results, and manipulate resources in the CMDB.

As you can tell, there are plenty of options for you to develop our own sensors, and invoke them through the CustomComputerSystemSensor or the CustomAppServerSensor.

9.5 Relationship discovery

During discovery of the target system, the GenericServerSensor discovers all the processes that are listening on a port. In addition, all the open sessions are also identified, and stored in the CMDB. When the background topology builder task executes, the processes that are listening are matched up with the sessions, it can now be determined which processes on a given system rely on processes on another system. In the TADDM user interface, these connections are known as dependencies. In the CMDB the information is stored as relationship objects.

You must understand, that TADDM identifies only the processes and sessions that are active when the discovery is performed. This means, that in order to discover connections between resources, the resources must be active, and communication when the discovery is performed.

When trying to assemble business applications from related and seemingly unrelated components, you must also realize that TADDM does not perform any kind of transaction decomposition. This means, that the built-in TADDM discovery sensors, while discovering minute details of for example a WebSphere Cluster and all its components, do not discover and analyze the internal runtime relationships in each WebSphere Server. Modern application server technology has become so advanced and dynamic, that the application server itself manages a lot of resources on behalf of the entire environment, for example sessions, and provides these resources as services to the components that implement the business functions. For that reason TADDM cannot discover relationships between, for example an ear-file and a database, or a message queue.

In addition, TADDM has absolutely no idea of the functionality of he discovered processes are. If for example, TADDM discovers a process for which the program name is klz, TADDM has no knowledge that helps it determine that this program as a matter of fact provides monitoring services for a Linux operating system. When the klz program is discovered, it should be registered as a TMEAgent resource with a *manages* relationship between itself and the operating system on which it runs.

However, other solutions such as ITCAM for Transactions and ITCAM for SOA are capable of discovering and decomposing transactions and web service interactions. The data these solutions gather are stored in proprietary databases, and contains a lot of valuable information about the interactions and topology of the domains they manage. The same is true for the IBM Tivoli Monitoring solution which maintains information about the monitoring agents that manage specific operating system, and application server instances.

TADDM leverages the Discovery Library Adapter (DLA) technology, which enables you extract all the relevant information from related solutions, including definition of resources, their attributes, and their relationships, and load it directly into TADDM. You can say that this is similar to a discovery, but instead of discovering the resources themselves, you discover the information know to a trusted source.

By importing the information that is maintained by these supporting solutions, TADDM does not only get access to resource and relationship information that TADDM cannot discover, but the information that is extracted is gathered over time. This means that the topology that is built from the CMDB is much more trustworthy, and provides a more complete picture of your IT infrastructure.

From an application mapping perspective, the augmented CMDB enables much more granular application definitions because you can use the component-level relationship information in the queries that are used to create the definition of our business applications.

9.6 Discovery summary

TADDM provides many built-in discovery sensors that discover very detailed information about standard technologies such as physical and virtual hardware, networking devices, operating systems, application server platforms such as WebSphere, WebLogic, Siebel, SAP, and the supporting database-, web-, and messing servers.

In addition, you can provide your custom scripts to discover details related to your own servers, and technologies for which TADDM does not provide built-in support.

Finally, you can augment the TADDM CMDB by importing information that has been extracted from other solutions or custom data. This provides information that TADDM cannot discover, and helps paint the complete picture of your IT infrastructure.

9.7 Understanding how configuration information is organized

Before you start defining Business Applications it might make sense for you to gain a better understanding of how configuration and relationship information is represented in TADDM.

TADDM stores data and maintains relationships between resources based on the Common Data Model (CDM). This model provides a logical representation of all you IT infrastructure resource types, their attributes and relationships. In addition the CDM specifies naming rules, which are combinations of resource attributes that are used to uniquely define resource instances. The naming rules, and the attributes they refer, are very important, because they are used to reconcile resources.

When working with the Common Data Model, you must understand, that the model represents IT resources down to (almost) an atomic level. Each resource, known in the data model as a Configuration Item (CI), represents a unique resource that has a manageable configuration. This implies that each minute detail, such as a JDBC definition, ear file, or database configuration parameter is considered individual CI. CIs are interconnected with unidirectional relationships, many of which are defined implicitly in the model. For each resource type, the model defines valid relationships to other resource types. For example, an ear file is represented in the model as a SoftwareModule CI, and the only valid relationships between a Software Module and a J2EEServer are:

SoftwareModule	deployedTo	J2EEServer
SoftwareModule	uses	J2EEServer
J2EEServer	uses	SoftwareModule

The *uses* relationship is a very generic one that is allowed between any combination of resource types, so for all practical purposes, the only relationship between SoftwareModules and J2EEServers is the *deployedTo* relationship.

The Common Data Model documentation is located in the `CDMWebsite.zip` file which is located on your TADDM server in the `$COLLATION_HOME/sdk/doc/model` directory. When you have unpacked this file, a good starting point to view the documentation is the `<unpack_location>/CDMWebsite/misc/CDM.htm` file. If you have access to a running TADDM v7.2.1 system, you can also access the Common Data Model documentation in `<taddm_server>:/9430/cdm/datadictionary/cdm/misc/CDM.htm`.

In this document the Tomcat server that TADDM uses to provide user interface services is used to demonstrate how you can create Business Application definitions, so let us take a look at the CDM definitions for the J2EEServer resource type. The Common Data model does not provide a specific object class for Tomcat servers, so when it is discovered, the Tomcat server is registered in the TADDM database as a J2EEServer.

9.8 J2EEServer derivation hierarchy

The Common Data Model is an object-oriented model of your IT resources. This implies that resource instances inherit characteristics from their parent classes.It also implies that resource instances can be referenced through the parent class definition.

The class derivation hierarchy for the J2EEServer is:

```
(○)ModelObject
 |
 +--(○)core/ManagedElement
      |
      +--(○)core/ManagedSystemElement
           |
           +--(○)core/LogicalElement
                |
                +--app/AppServer
                     |
                     +--app/j2ee/J2EEServer
```

From this derivation hierarchy, you can see that the J2EEServer is a sub-class of the general AppServer class. This means, that when defining applications, you can use the generic AppServer class to reference all application servers, including J2EEServers. The practical use of this is, that, if you want, you can define general rules that include any application server, for which certain conditions apply.

In a similar fashion, other resource types are direct subclasses of the J2EEServer class. The CDM defines these resource types as subclasses of the J2EEServer class:

- GenericJ2EEServer
- JBossServer
- MySAPJ2EEEngineInstance
- OracleAppJ2EEServer
- WebLogicServer
- WebSphereServer

This means that these types of resources can be referenced as AppServer and J2EEServer resources, as well as by their specific type.

9.9 J2EEServer attributes

The Common Data Model defines no less than 65 attributes for a J2EEServer. Of these 26 are *owned* by the parent class (AppServer). In the CDM, attributes represent one of three possible properties:

- A specific configuration value
- An array of related resources
- A reference to a related resource

The attributes are used both to uniquely identify the resource, and keep track of the detailed configuration of the resource.

On the next page, you see a subset of the attributes defined for the J2EEServer resource type.

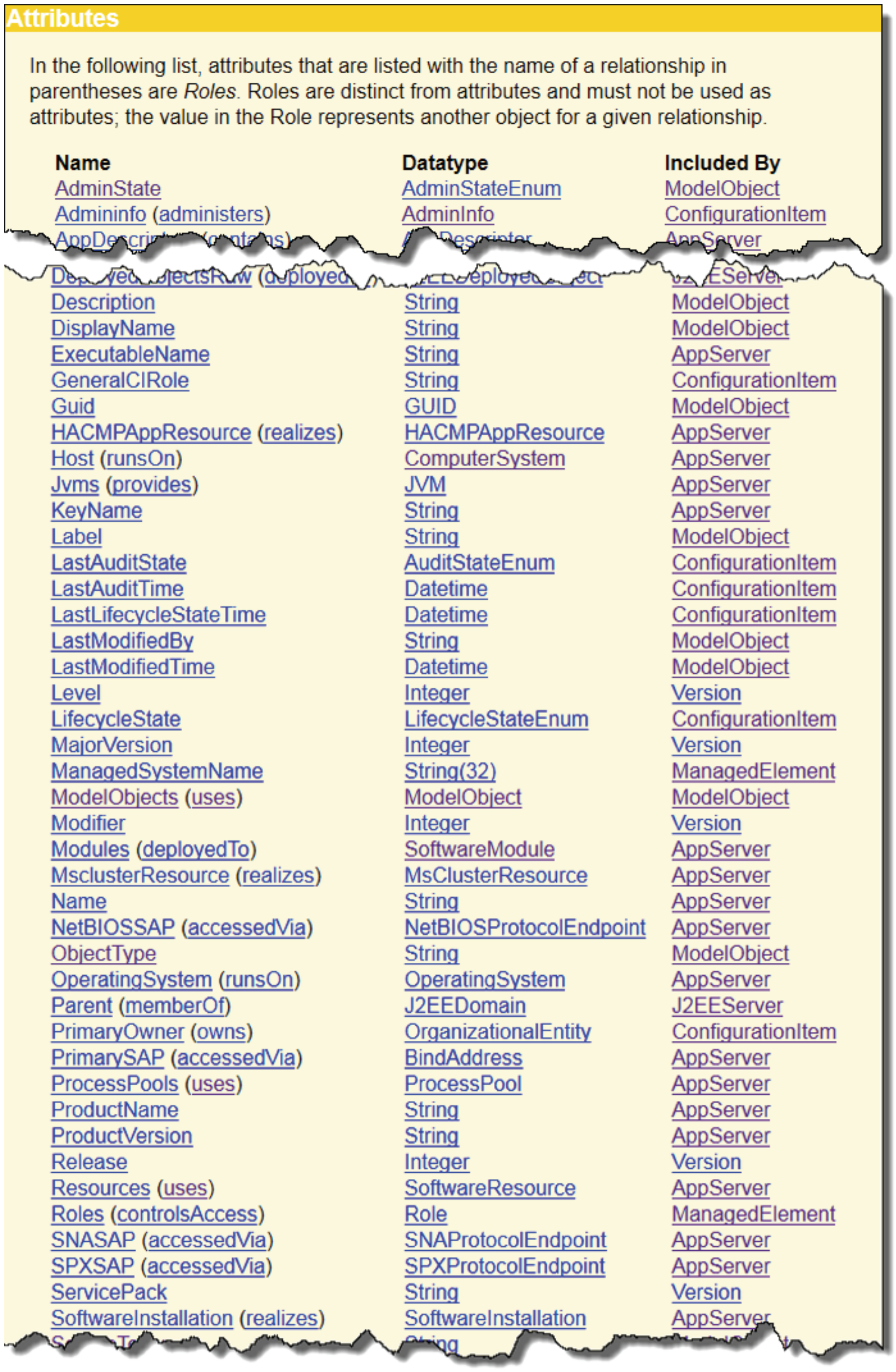
The J2EEServer class specifies configuration value attributes for Name, KeyName, ProductName, BuildLevel, and many more. It is the responsibility of the sensor that discovers the J2EE Server to provide values for these attributes. Many attributes have been inherited from the parent class, AppServer in this case, and even from upper level classes such as the ManagedSystemElement, and ModelObject classes. When you populate attributes, it is important to understand that the attributes owned by the system classes (denoted by the `(o)` indicator in the derivation hierarchy) are maintained by TADDM, and cannot be manipulated by any other component. TADDM maintains key attributes, common to all objects, such as GUID, ContextIp, CreatedBy, CDMSource, DisplayName and others. Naturally you can use these attributes to qualify resources in a query, but you cannot update them from a custom sensor, or a program.

If you look closely, you will also see attributes such as ConfigFile, Containers, Host, and OperatingSystem for the J2EEServer class. These are all superseded by a relationship name in parentheses, and the datatype is a CDM resource class. This means, that the attributes contains a pointer to another resource, and the implicit relationship this pointer denotes is of the specified type. These implicit relationships can be used creatively when querying the database. The Model Query Language allows you to traverse implicit relationships simply by using the unary operator (.). A query like this:

```
select name from J2EEServer where lower(host.fqdn) ends-with 'joe.com'
```

provides the names of all J2EEServers that are running on ComputerSystems that have a fully-qualified-domain-name which ends with `joe.com`. Understanding and using these implicit relationships is key to successfully creating rules to define your Business Applications.

If the cardinality of the relationship between one CI type and another allows for one-to-many or many-to-many definitions, an attribute may contain an array of pointers to related CIs. This is an important point to remember, since the Model Query Language (MQL) treats arrays differently from pointers.



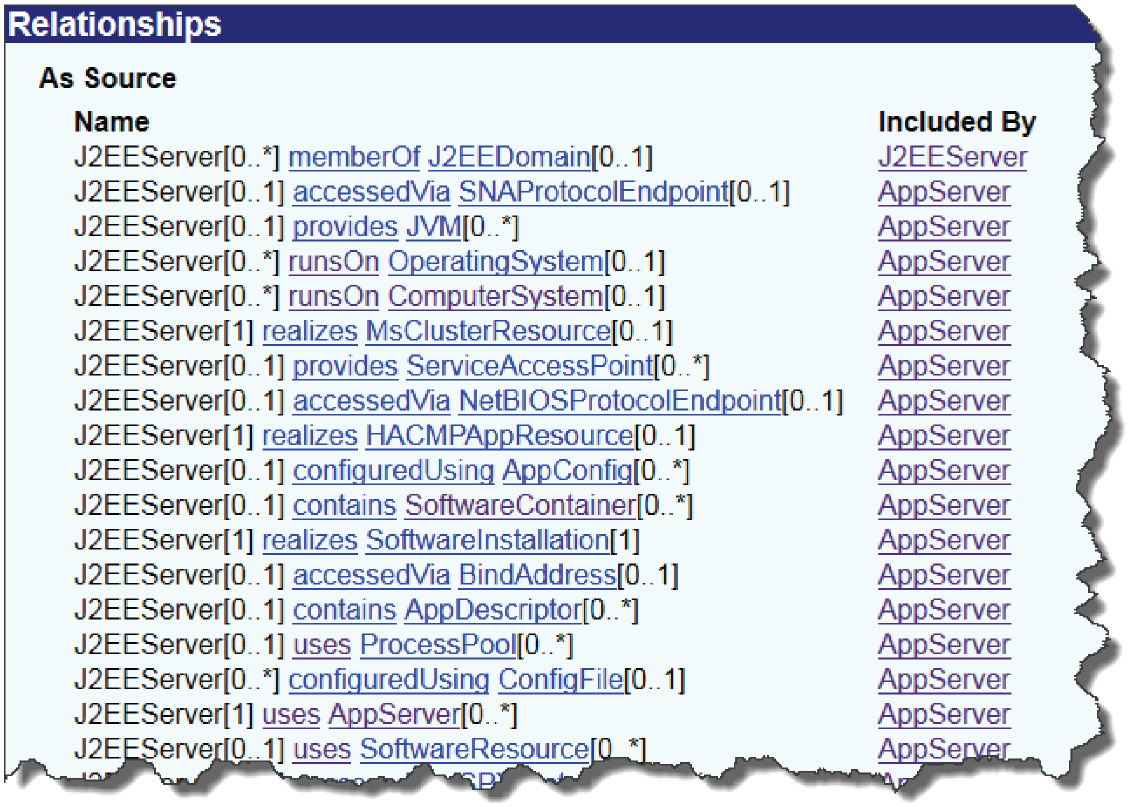
Attributes

In the following list, attributes that are listed with the name of a relationship in parentheses are *Roles*. Roles are distinct from attributes and must not be used as attributes; the value in the Role represents another object for a given relationship.

Name	Datatype	Included By
AdminState	AdminStateEnum	ModelObject
Admininfo (administers)	AdminInfo	ConfigurationItem
AppDescriptor (contains)	AppDescriptor	AppServer
DeployedObjectsRaw (deployed	J2EEDeployedObject	J2EEServer
Description	String	ModelObject
DisplayName	String	ModelObject
ExecutableName	String	AppServer
GeneralCIRole	String	ConfigurationItem
Guid	GUID	ModelObject
HACMPAppResource (realizes)	HACMPAppResource	AppServer
Host (runsOn)	ComputerSystem	AppServer
Jvms (provides)	JVM	AppServer
KeyName	String	AppServer
Label	String	ModelObject
LastAuditState	AuditStateEnum	ConfigurationItem
LastAuditTime	Datetime	ConfigurationItem
LastLifecycleStateTime	Datetime	ConfigurationItem
LastModifiedBy	String	ModelObject
LastModifiedTime	Datetime	ModelObject
Level	Integer	Version
LifecycleState	LifecycleStateEnum	ConfigurationItem
MajorVersion	Integer	Version
ManagedSystemName	String(32)	ManagedElement
ModelObjects (uses)	ModelObject	ModelObject
Modifier	Integer	Version
Modules (deployedTo)	SoftwareModule	AppServer
MsclusterResource (realizes)	MsClusterResource	AppServer
Name	String	AppServer
NetBIOSSAP (accessedVia)	NetBIOSProtocolEndpoint	AppServer
ObjectType	String	ModelObject
OperatingSystem (runsOn)	OperatingSystem	AppServer
Parent (memberOf)	J2EEDomain	J2EEServer
PrimaryOwner (owns)	OrganizationalEntity	ConfigurationItem
PrimarySAP (accessedVia)	BindAddress	AppServer
ProcessPools (uses)	ProcessPool	AppServer
ProductName	String	AppServer
ProductVersion	String	AppServer
Release	Integer	Version
Resources (uses)	SoftwareResource	AppServer
Roles (controlsAccess)	Role	ManagedElement
SNASAP (accessedVia)	SNAProtocolEndpoint	AppServer
SPXSAP (accessedVia)	SPXProtocolEndpoint	AppServer
ServicePack	String	Version
SoftwareInstallation (realizes)	SoftwareInstallation	AppServer
[illegible]	String	[illegible]

9.10 J2EEServer relationships

In addition to the attributes, for each class, the Common Data Model defines the relationships in which resources of the specific type can participate. These relationships are always unidirectional, so the CDM defines specifically if the class can be the source or the target of relationship. Some of the relationships in which the J2EEServer instances can be the source are:



Relationships

As Source

Name	Included By
J2EEServer[0..*] memberOf J2EEDomain[0..1]	J2EEServer
J2EEServer[0..1] accessedVia SNAProtocolEndpoint[0..1]	AppServer
J2EEServer[0..1] provides JVM[0..*]	AppServer
J2EEServer[0..*] runsOn OperatingSystem[0..1]	AppServer
J2EEServer[0..*] runsOn ComputerSystem[0..1]	AppServer
J2EEServer[1] realizes MsClusterResource[0..1]	AppServer
J2EEServer[0..1] provides ServiceAccessPoint[0..*]	AppServer
J2EEServer[0..1] accessedVia NetBIOSProtocolEndpoint[0..1]	AppServer
J2EEServer[1] realizes HACMPAppResource[0..1]	AppServer
J2EEServer[0..1] configuredUsing AppConfig[0..*]	AppServer
J2EEServer[0..1] contains SoftwareContainer[0..*]	AppServer
J2EEServer[1] realizes SoftwareInstallation[1]	AppServer
J2EEServer[0..1] accessedVia BindAddress[0..1]	AppServer
J2EEServer[0..1] contains AppDescriptor[0..*]	AppServer
J2EEServer[0..1] uses ProcessPool[0..*]	AppServer
J2EEServer[0..*] configuredUsing ConfigFile[0..1]	AppServer
J2EEServer[1] uses AppServer[0..*]	AppServer
J2EEServer[0..1] uses SoftwareResource[0..*]	AppServer

Many of the relationships represent implicit relationships between a source and a target. Implicit relationships are those relationships that are maintained and enforced by the model itself, through pointers. Implicit relationships are those that, in the model, are used as part of a pointer in an attribute definition. For example, for the J2EEServer, the Host (*runsOn*) attribute specifies that the value of the attribute is the resource for which a *runsOn* relationship exists, and the datatype of the attribute enforces that the resource pointer, the value of the attribute, must be a ComputerSystem. If you look at the relationship, in which the J2EEServer is the source, you see that the J2EEServer can be the source of several *runsOn* relationships, but since the datatype for the host attribute is ComputerSystem, you can narrow the possibilities down to one. The cardinality for the runsOn relationship between the J2EEServer and the ComputerSystem specifies a many-to-one relationship, so the CDM allows multiple J2EEServer instances to runs on a single ComputerSystem, and only allows a J2EEServer instance to run on a single ComputerSystem.

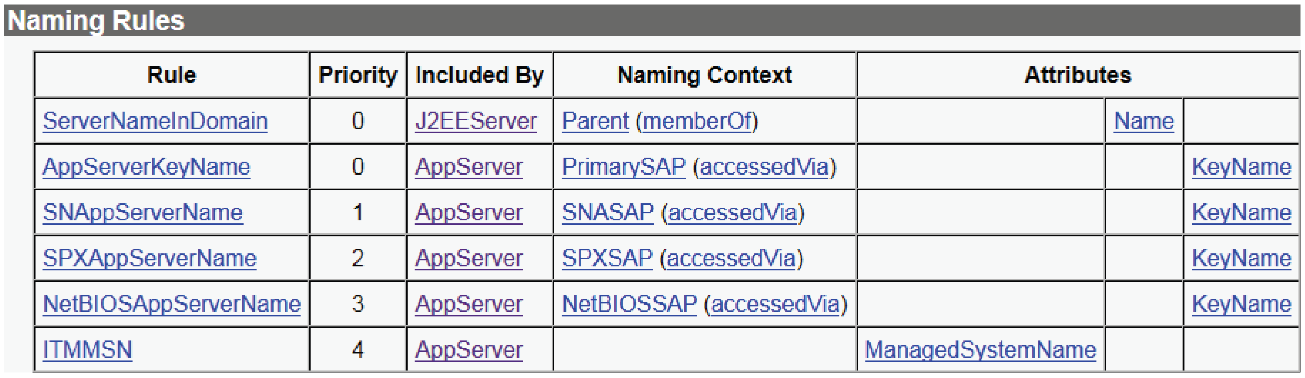
Relationships that are not used as part of a pointer attribute are explicit relationships. These are not enforced automatically by the values of the pointer attributes in the CDM, but must be provided and maintained programmatically. Explicit relationships are mainly used to provide generic metadata about a resource, which cannot be modeled. Using explicit relationships, you can maintain relationships between resources that are not otherwise related. The ultimate form of an explicit relationship is ConfigurationItem uses J2EEServer. This means that you can create a relationship between any resource in your IT infrastructure and the J2EEServer. The interpretation of this it relationship is totally up to you.

9.11 J2EEServer naming rules

For resources that are classified as J2EEServer resources, the Name and KeyName attributes have special significance. These are used in the naming rules to uniquely identify each instance of that particular resource type.

The CDM specifies, for all classes, a set of one or more naming rules. Each rule defines a unique set of attributes which is used to identify each instance. When new resources are created, the program that needs to store its information in accordance with the CDM, must provide enough attribute values to satisfy at least ONE naming rule. For most classes, the CDM defines multiple naming rules in order to provide flexibility, so the programs are not forced to provide a fixed set of attribute values.

The naming rules for the J2EEServer class are:



Naming Rules

Rule	Priority	Included By	Naming Context	Attributes		
ServerNameInDomain	0	J2EEServer	Parent (memberOf)		Name	
AppServerKeyName	0	AppServer	PrimarySAP (accessedVia)			KeyName
SNAppServerName	1	AppServer	SNASAP (accessedVia)			KeyName
SPXAppServerName	2	AppServer	SPXSAP (accessedVia)			KeyName
NetBIOSAppServerName	3	AppServer	NetBIOSSAP (accessedVia)			KeyName
ITMMSN	4	AppServer		ManagedSystemName		

When a new J2EEServer object is instantiated, the provided attribute values are matched with the naming rules from the lowest to the highest priority. So if, for example, the Parent and Name attributes are provided by the program, the very first naming rule, here named ServerNameInDomain, will be used to uniquely identify the new J2EEServer. If the first rule cannot be matched, for example because a value for the Name attribute is not provided, the next rule is validated. In this case, the second rule, AppServerKeyName, requires that attribute values for the KeyName and the PrimarySAP are populated.

For both naming rules, a pointer attribute is used. This implies that when creating new J2EEServer resources, the J2EEDomain resource referenced in the Parent attribute and the BindAddress used in the PrimarySAP must exist for the rules to be used. The relationship specification allows the CDM to use the cardinality of the relationship to prevent object creation if the cardinality does not allow a new relationship to be created, for example if the cardinality is 1:1 and a relationship already exists.

9.12 CDM summary

All resource types defined by the Common Data Model are characterized by:

- A number of attributes that contains the configuration information for the resource Attribute values can be:
    - A specific value
    - A pointer to another configuration item to which the resource has an implicit relationship
    - An array of pointers

- Relationships that specify how the resource can be related to other resources. Implicit relationships are enforced by the CDM, and explicit relationships are used to associate resources that otherwise have no logical relationship that can be modeled.
- Naming rules, combinations of attributes that are used to uniquely identify individual resources. In many instances, pointer attributes are used in naming rules.

9.13 Querying the TADDM database

In the previous two sections you have looked at how configuration information is organized in the TADDM CMDB, and how it is discovered so it can be stored. In this section, we will discuss how you can query the CMDB to extract specific details.

Naturally, TADDM provides a user interface from which you can access the data in the CMDB. This user interface includes a Query wizard that allows you to create simple reports by choosing a main resource type, and then, from a tree-like structure, select the attributes you want to see in the report. Attributes can be selected from the main resource type, or from any resources that are implicitly related to the main resource. After you have selected the information you want to see in the report, you can apply filters to include only information where certain conditions exist. Under the covers, the Query wizard creates the Model Query Language (MQL) definitions needed to extract the information from the CMDB in accordance with our specifications.

Naturally TADDM also provide various APIs, including a command-line API. Using this API, you can extract information from the CMDB by specifying your own MQL query. This allows for greater flexibility than using the Query wizard, but the disadvantage is, that you need to understand exactly how the data are organized in the CMDB. This is why the first section in this document discussed the Common Data Model, which documents all the details you need to know to manually build your own MQL queries.

Understanding how to query the TADDM CMDB is paramount to defining Business Applications. Business Applications are defined by specifying a number of queries, each of which will associate a specific set of resources with the business application. So, in order to get the results to want, you must understand the CDM and master the MQL language.

In addition to the Query wizard and the command-line API, TADDM also provides a RESTful State Transfer (REST) interface, which allows you to interact with the TADDM CMDB through a browser or from a script. Contrary to the Query wizard and the command-line API, the REST API can be used to both query and update the CMDB.

9.14 Using the Query wizard

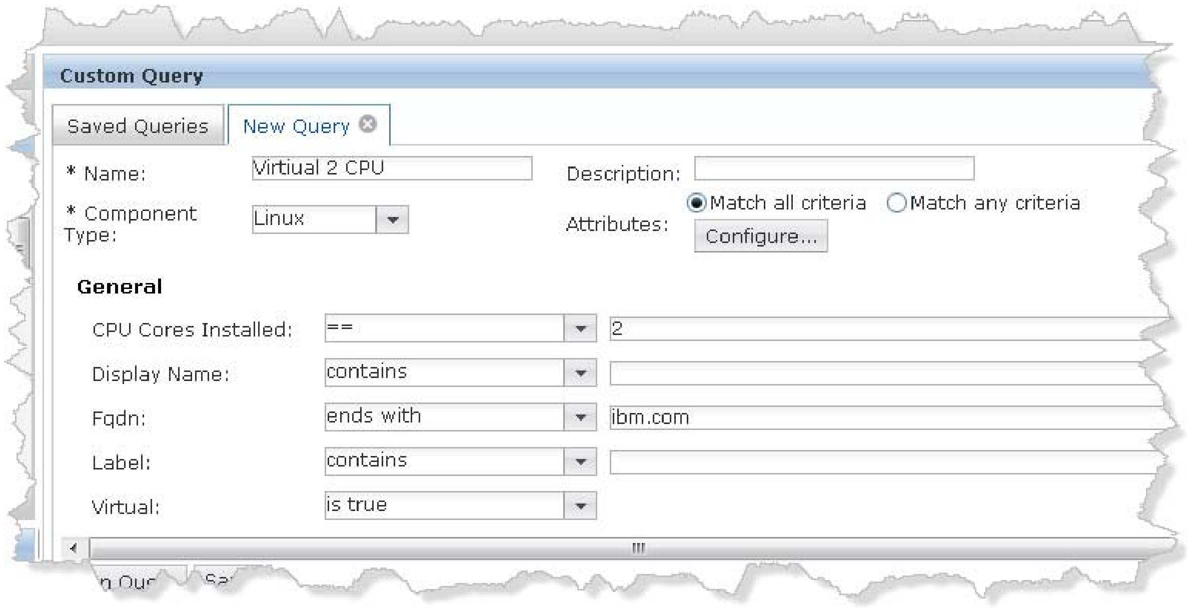
TADDM includes a Query wizard that enables you to easily create simple reports by querying the database. The queries you create can be saved, and used as templates when you define the rules that determine which resources to include in the functional groups of a business application.

The query wizard provides a graphical interface from which you can select objects and attributes to be included in the report. Once you have selected the attributes you want to include, you can apply filtering criteria to limit the resources shown in the report to the specific subset for which your criteria apply.

One restriction in the query wizard you have to be aware of is, that it only allows you to select resource types that are represented in the TADDM database. The implication of this is, you cannot select parent classes (such as ComputerSystem, or J2EEServer) since all your custom server templates and sensors register the discovered resources at the lowest (most qualified) level. This means that if you, for example, discover the resources of a Linux system hosting a TADDM primary storage server (using a custom server template named *TADDM Server* to find the process that listens on port 9430) TADDM will register a LinuxUnitaryComputerSystem, and a TADDM Server resource in the database. The LinuxUnitaryComputerSystem is a sub-class of ComputerSystem, and the TADDM Server resource is a sub-class of the class selected when defining the custom server template, most likely J2EEServer.

When you create a rule based on a saved query, you can see the MQL statement directly in the Grouping Composer. This makes it easy for you to understand how the MQL syntax works, and helps you quickly create the MQL statements for the rule. However, you must understand that rules only require the GUID of the selected resources, so you will not see the references to display attributes. Only the filtering criteria are shown.

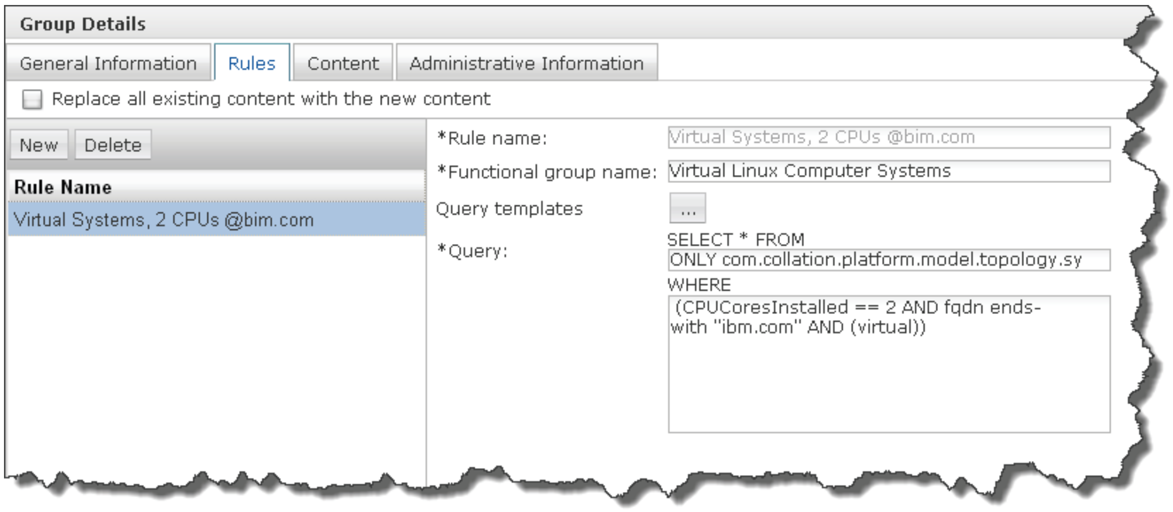
For example, to create a query that shows the virtual Linux systems in your infrastructure that has exactly two CPUs, and a fully qualified domain name (fqdn) that ends with *ibm.com*, you would use the Query wizard to produce a query that looks like this:



* Name: Virtiual 2 CPU
Description:
* Component Type: Linux
Attributes: [✓ Match all criteria] Match any criteria

CPU Cores Installed:	==	2
Display Name:	contains	
Fqdn:	ends with	ibm.com
Label:	contains	
Virtual:	is true	

When you use this query as a template for a rule, the rule displays the MQL where clause:


Group Details

General Information | Rules | Content | Administrative Information

Replace all existing content with the new content

New | Delete

Rule Name

Virtual Systems, 2 CPUs @bim.com

*Rule name: Virtual Systems, 2 CPUs @bim.com

*Functional group name: Virtual Linux Computer Systems

Query templates ...

*Query: SELECT * FROM ONLY com.collation.platform.model.topology.sy WHERE (CPUCoresInstalled == 2 AND fqdn ends-with "ibm.com" AND (virtual))

The query information in the rule can be translated to the following MQL statement:

```
select ONLY guid from LinuxUnitaryComputerSystem
where CPUCoresInstalled == 2
and fqdn ends-with "ibm.com"
and virtual
```

If you look up the LinuxUnitaryComputerSystem details in the Common Data Model, you will see that the LinuxUnitaryComputerSystem class is a sub-class of ComputerSystem, and that the three attributes used in the where clause (CPUs, fqdn, and virtual) are all inherited from the ComputerSystem class. This implies that you can change the LinuxUnitaryComputerSystem resource type in the query to *ComputerSystem* to include all types of virtual computer systems with two processors that are members of the ibm.com domain.

Since you are allowed to modify both the resource type and the where clause of the rule, you can easily imagine how you can use the command-line API to model your rule queries, and paste in specific sections from your command-line interface.

9.15 MQL Queries through the command-line API

No matter which API you use, the Model Query Language is used to access the data in the TADDM CMDB. This is also true when you work with business applications. Business applications are composed from one or more functional groups, and each functional group is populated through a query. When defining business applications in the Grouping Composer you provide the MQL WHERE clause directly in the Grouping Composer, so you need to understand how to create MQL queries in order to define business applications.

The Model Query Language is a query language that allows you to execute object-oriented queries to extract information from the TADDM CMDB. The structure and syntax is very similar to that of the Structured Query Language (SQL) which is the predominant query language to extract information from a relational database.

An MQL query is constructed by specifying the model object class, or other data sources, their attributes and a filter expression to limit the selected objects. In its simplest form, it names a single model class, such as: *J2EEServer*.

```
select * from J2EEServer
```

In this example, this query would return all instances of the J2EEServer resource type.

The class name may be fully qualified, as in *com.collation.platform.model.topology.sys.J2EEServer*. Or it may be of the short form: *J2EEServer*.

There are currently seven model classes whose short form is ambiguous: *AgentConfiguration*, *ControlledBy*, *Enumeration*, *SSLSettings*, *Template*, *WebModule* and *Zone*. Some model objects are not selectable as they are abstract, or simply because no instances exist. There are no queries to find all instances of all objects.

The structure of a query, as it applies to definition of business applications, is:

```
select attribute-list from data-sources [ WHERE clause ]
```

Where data-sources is a comma-separated list of model object class names, or external data sources. A data source may be fully qualified or the short form.

The WHERE expression looks like:

```
attr-name OP expression [ ... ]
```

Where attr-name may be any attribute of any selected resource type and may include attributes separated by the unary operator (.).

9.16 The Unary operator

The unary operator is used to follow the references stored in pointer attributes (implicit relationship) to access the referenced resource. An example:

```
select displayName, host.name from J2EEServer where
host.OSRunning.OSName == 'Linux' and host.numCPUs > 3
```

In this example, host is an attribute of the J2EEServer type (inherited from the AppServer class).The host attribute contains a reference to a ComputerSystem instance. OSRunning is an attribute of the ComputerSystem resource type which contains a reference to an instance of an OperatingSystem instance. OSName is an attribute of the OperatingSystem resource type. numCPUs is also an attribute of the ComputerSystem type. Therefore, the query above returns the displayName of the J2EEServer and the name of the ComputerSystem hosting the J2EEServer for all J2EEServers hosted on a ComputerSystems with more than 3 CPUs and on which a Linux operating system is running.

Memberships

As mentioned, some attributes may contain multiple resource references. Commonly, this type of attributes is referred to as arrays.

In an MQL query, you can use the membership of an array to qualify the result set using the EXISTS operator. For example:

```
select * from J2EEDomain where EXISTS (resources.name contains 'trade'
and resources.objectType contains 'JDBC' )
```

The EXISTS expression can be any expression which refers to any array attributes of a class in the FROM list. At least one attribute must be an array in order to use the EXISTS expression.

Boolean values

When querying on boolean attributes all you need to provide is the name of the attribute, if you want to filter in resources for which the attribute value is *true*. To test for a value of *false*, you must append the is-null operator.

The following example fetches all virtual ComputerSystems:

```
select * from ComputerSystem where virtual
```

and this query returns all physical ComputerSystems:

```
select * from ComputerSystem where virtual is-null
```

Joins

Inner joins are supported against other resources. These joins can be used to test conditions for which there are no implicit or explicit relationships. Other join types, such as combinations of right outer or left outer are not currently supported. Only one type of model object will be returned from a join. For example:

```
select * from J2EEServer, FunctionalGroup where Db2Server.port ==
OracleInstance.port
```

This returns all Db2Server model objects where the port number from Db2Server and the port number from OracleInstance are equal.

The relationship resource type

Explicit relationships are stored as Relationship resources. All relationships have a type, as well as a source and a target. You can use these relationships identify resources that are explicitly related to each other by using the relationship attributes in joins.

For example, assuming that you want to identify all database instances that are used by the J2EEServers in a functional group named *joe*, you would use a query similar to this

```
select guid from DatabaseServer, FunctionalGroup, Relationship where
DatabaseServer.guid == Relationship.target and
FunctionalGroup.displayName == 'joe' and
exists( FunctionalGroup.members.guid == Relationship.source)
```

In this query you want to return the guid of all DatabaseServers, that is why the resource type DatabaseServer appears first in the list of resource types to inspect. Then you specify that the target in the relationship is the database, because you want to find the databases that are used by other resources. In order to identify the resources that uses the database, you qualify the specific FunctionalGroup that contains members that might communicate with a database. Finally, in the EXISTS section, you pick only the members of the functional group for which an explicit relationship, having the member as the source, exists.

Naturally, if you want to identify specific types of database servers, you can replace the DatabaseServer resource type with one of its subclasses - for example OracleInstance.

Agreed, you could restructure this query to make it more intuitive to read. However, tests have revealed, that to build reliable MQL queries the EXISTS clause should always be the last clause in an MQL query.

9.17 MQL Tips

When developing queries to identify members of a business application, you will most likely use the command-line API to model and refine your queries. In this process you may find these tips helpful.

The Model Query Language allows you to limit the number of records to be selected. By appending FETCH FIRST x ROWS to the query you can limit the number of records returned from the query. When developing queries for business application mapping, you are often more focused on whether or not your filtering criteria works as expected, that on the actual data that are returned.

For example, to see only the label of a single J2EEServer, you can issue a query like this:

```
select label from J2EEServer FETCH FIRST 1 ROWS
```

The command-line API find method supports the addition of a `--count` (or `–c`) parameter. You can use this to see how many resources will be returned by a specific query, but also the quickly validate the query without fetching all the data.

For example, to select see how many ComputerSystems that are represented in the CMDB, you can issue this command from the location where you have installed the TADDM SDK:

```
api.sh –u administrator –p collation find --count ComputerSystem
```

Another point to remember when developing queries for business application mapping is, that for that specific purpose all you need is the guids of the resources you want to include in the business application. Specific attributes such as displayName or object types are not used by the grouping composer. Unless you need to see specific attributes in order to verify the operation of the query, there is no need for you to spend time on specifying them.

9.18 MQL Query summary

The Model Query Language is used to define queries that can be used to extract information from the TADDM CMDB. The Query wizard helps you define queries without knowing the intimate details of the MQL language, but it has some limitations. However as a starting point, to get familiar with the MQL language, you can use the Query wizard to define rudimentary queries that you can expand later on.

The main interface for developing queries for business application mapping is the command-line API. Here you create and refine your queries, and can copy-and-paste them into the appropriate business application rules in order to populate specific functional groups.

9.19 Model Query Language Specification

SELECT statement grammar:

```
statement :=  SELECT attribute_list [ EXCLUDING attribute_list ] FROM [ ONLY
] class_list { WHERE [expression | exists_expr] }
[ FETCH FIRST n { ROW | ROWS } [ ONLY ]] [ ORDER BY order_list ]

attribute_list := attrib {, attrib}* | *

attrib := {class .} [<an attribute of a class>{.embedded_attribute} | * ]

embedded_attribute := [<embedded attribute>{.embedded_attribute} | *]

class_list := domain_class {, domain_class }*

domain_class := {domain_list} class

class := <a model object class>

domain_list := domain {, domain}* :

domain := <the server from which to pull data from, default: local database>

exists_expr := exists( array_attrib op value )

expression := [ attrib op value | attrib post-op | pre-op ( attrib ) | attrib
[ NOT ] IN ( expression [, ...] ) ] {logical_op expression}*

value := <data value>

array_attrib := <series of attributes where at least the second to last
attribute is an array >

op := != | == | > | < | >= | <= | contains | starts-with | ends-with | equals
| not-equals

logical_op := AND | OR | && | ||

post_op := is-null | is-not-null

pre_op := lower | upper

all := *

order_list := attrib [ ASC | ASCENDING | DESC | DESCENDING ] [ , order_list ]
```

Note: **-** All keywords (SELECT, FROM, WHERE, etc.) are not case-sensitive.
- Attributes can contain wildcard characters.

The currently supported expression operators are:

Remarks	Operator-name	precedence
.	dot selection	5
!	unary not	4
upper()	function	4
lower()	function	4
exists	array contains	4
in	list membership	4
()	parentheses	4
is-null	is null	3
is-not-null	is not null	3
equals	equals	2
not-equals	not equals	2
starts-with	starts-with	2
ends-with	ends-with	2
contains	contains-with	2
==	equals	2
!=	not equals	2
>	greater than	2
>=	greater or equal	2
<	less than	2
<=	less or equal	2
instanceof	instance of a class	2
and	logical and	1
or	logical or	0

Higher precedence values have greater precedence.

Other SQL SELECT operators or features, such as GROUP BY, HAVING, DISTINCT, nested SELECTS, IN, BETWEEN, aggregates, etcetera, are not supported.

Notices

Trademarks