IBM Software Group

IBM WebSphere® Technical Conference
Featuring WebSphere Portal, WebSphere MQ, CICS® and Rational®

CT07 JCA and CICS: An Introduction to the J2EE Connector Architecture

nigel_williams@uk.ibm.com

IBM Design Center for e-business on demand

WebSphere software

@business on demand software

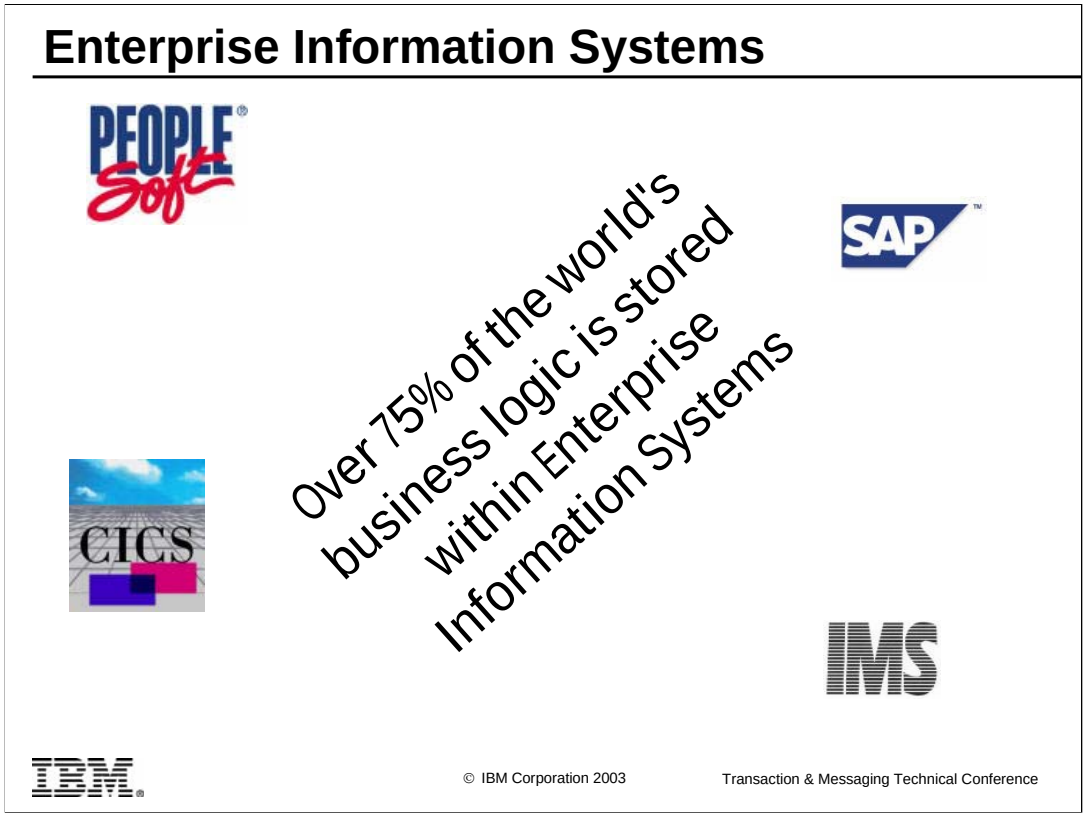
Agenda

- Introduction
  - JCA specification
  - Resource adapters
- Common Client Interface
  - API overview
  - CCI development tools
- System Level Programming Interface
  - System contracts
  - Managed environments
- Resource adapter packaging & deployment
  - Deployment process

Part One

Part One

Introduction

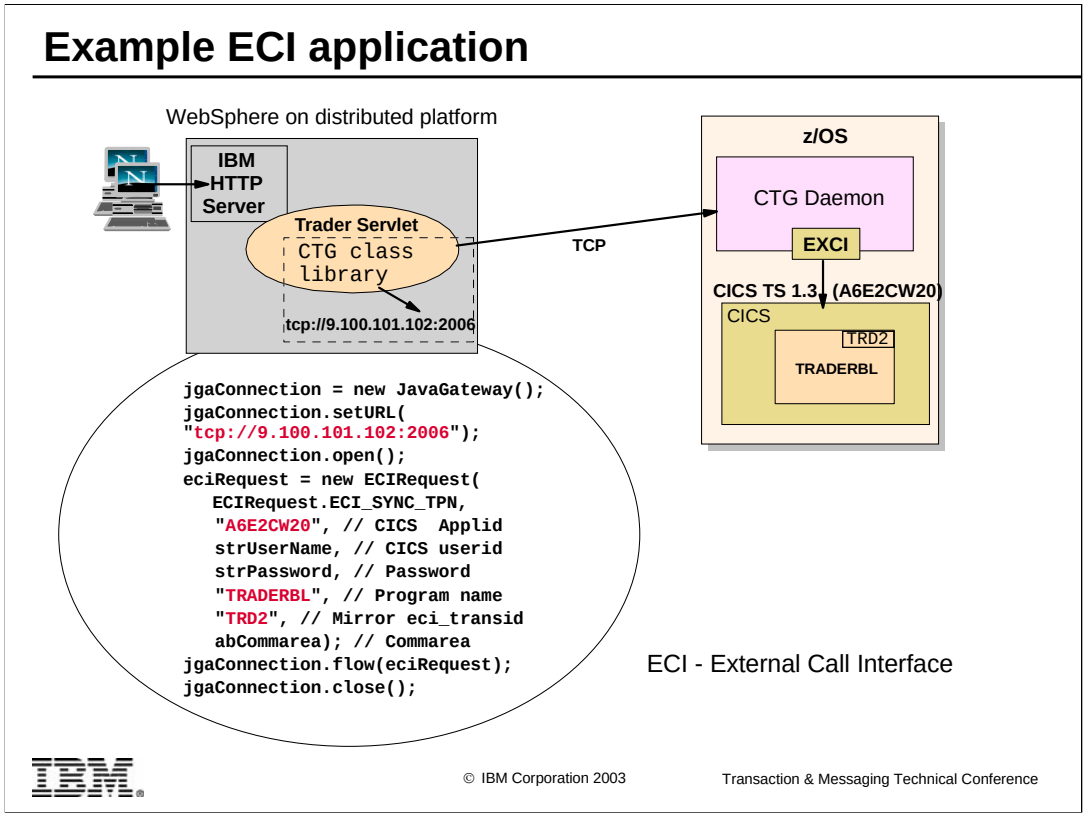


Over 75% of the world's business logic is stored within Enterprise Information Systems like CICS. Therefore we need an efficient way of accessing these applications from new applications which are written to the J2EE specification and will most likely be running in WebSphere Application Server.

Connector Solutions - background

- EIS specific
  - CICS Transaction Gateway APIs
    - Base ECI API, Base EPI API, EPI Support Classes
  - IMS Connect
- IBM Common Connector Framework (CCF)
  - Common API for all EISs supported by the framework
  - Tools to generate connector code
    - **VisualAge® for Java® Enterprise Access Builder**

The JCA is really an evolution based on the CCF . The key difference is that the CCF is an IBM standard whilst the JCA is an open standard.



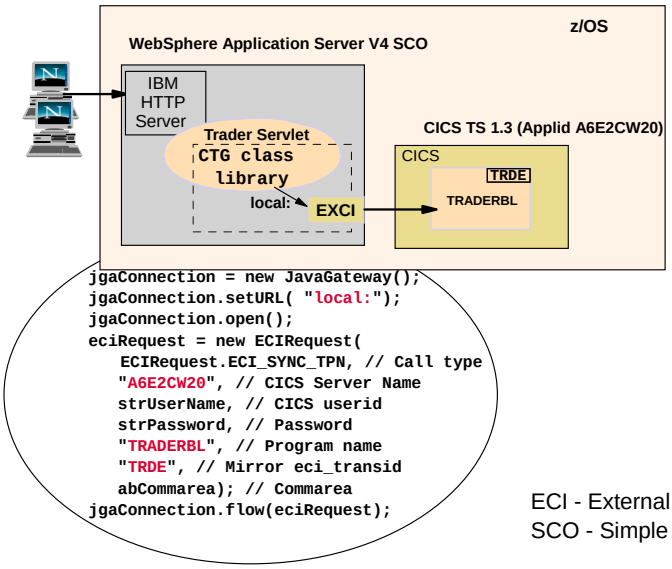
This Chart shows

- The Trader servlet running on distributed platform
- The JavaGateway specifies the tcp address and port of the CICS TG daemon
- A TCP/IP connection is established to a CICS TG daemon
- The CICS TG daemon uses EXCI to pass the request onto CICS

This type of application works well and has been deployed very widely, however:

- The ECI API is specific only to CICS and if a developer has to connect to different EISs he will need to learn different APIs
- The application itself has to manage such things as connections, security and transactions – it gets no help from the application server

ECI application (local connection)




```
jgaConnection = new JavaGateway();
jgaConnection.setURL( "local:");
jgaConnection.open();
eciRequest = new ECIRequest(
   ECIRequest.ECI_SYNC_TPN, // Call type
   "A6E2CW20", // CICS Server Name
   strUserName, // CICS userid
   strPassword, // Password
   "TRADERBL", // Program name
   "TRDE", // Mirror eci_transid
   abCommarea); // Commarea
jgaConnection.flow(eciRequest);
```

ECI - External Call Interface
SCO - Simple Configuration Option

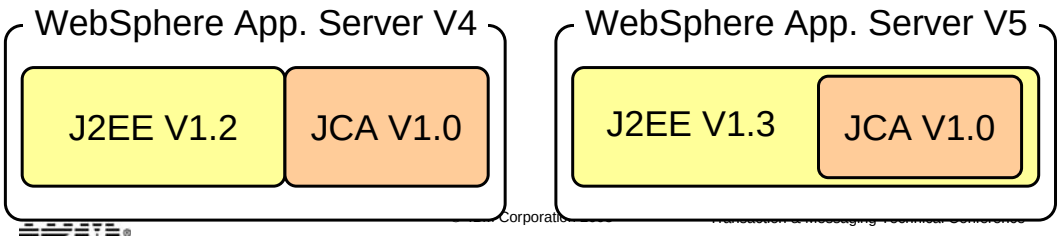
This Chart shows the Trader servlet running on WebSphere z/OS using a local connection to CICS:

Local mode is specified for the JavaGateway - this means that CTG native code libCTGJNI.so is called directly

The CICS Transaction Gateway daemon is not usually required when running servlets within the WebSphere Application Server on z/OS since the CTG local: protocol can be used to directly invoke the native functions in the underlying EXCI. The EXCI passes the request onto the attached CICS region.

J2EE Connector Architecture (JCA)

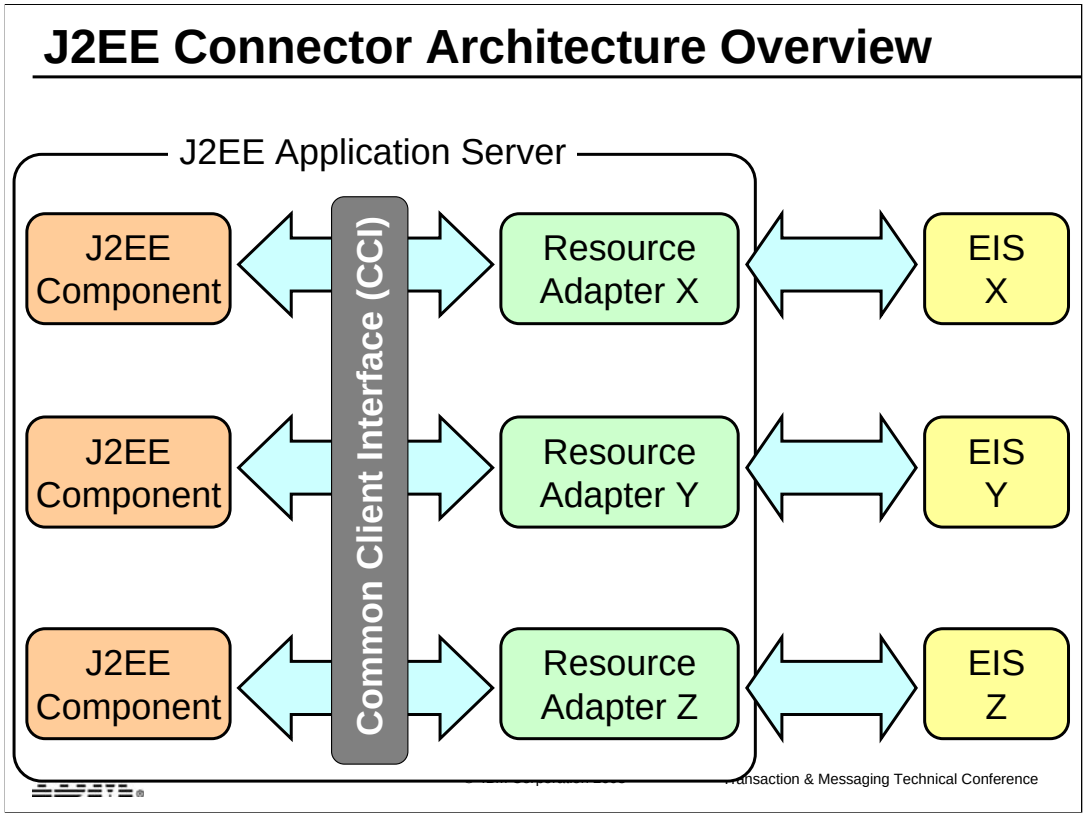
- Provides a uniform and simplified connectivity to heterogeneous Enterprise Information Systems (EIS)
- J2EE 1.3 standard introduced the J2EE Connector Architecture V1.0 (JCA)
- Many J2EE 1.2 application servers have been extended to include support for the J2EE Connector Architecture,
  - Including WebSphere Application Server V4



The J2EE Connector Architecture (JCA) is a standard way for a Java component to connect to an EIS. It is part of the J2EE standard, introduced in the J2EE V1.3 specification. WebSphere Application Server V5 is J2EE V1.3 compliant and therefore supports this new connector architecture.

WebSphere Application Server V4 is J2EE 1.2 compliant but it also contains support for the JCA.

See differences between WebSphere V4 and V5 support later.

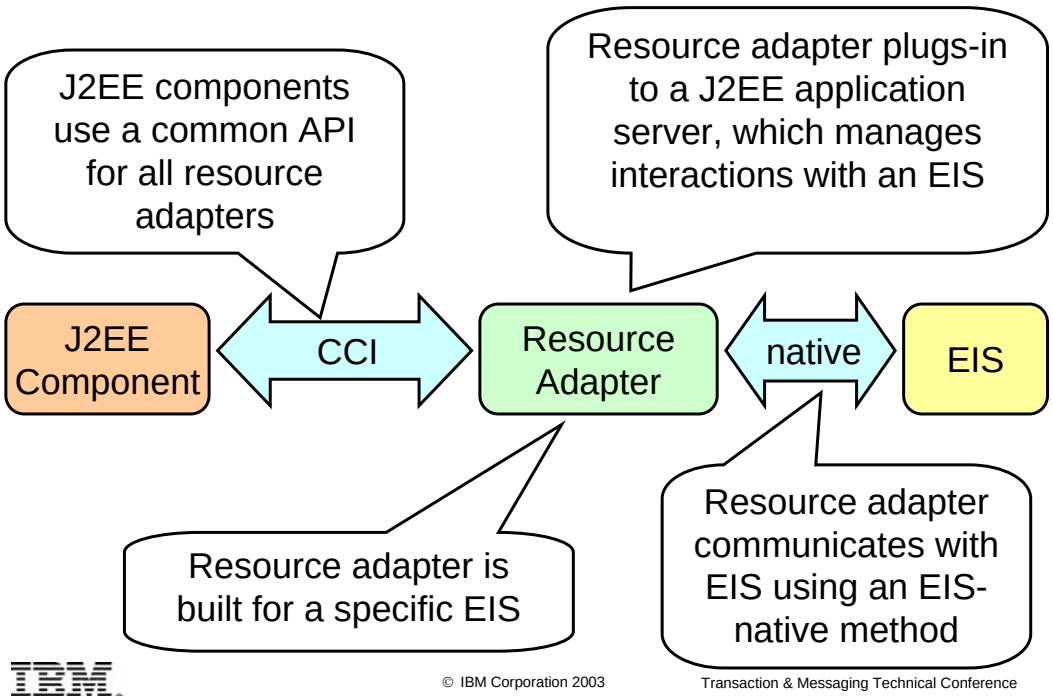


The JCA specification states that the way an application server communicates with an EIS is through a resource adapter. A resource adapter is written to support a specific EIS, but all resource adapters expose common interfaces:

A common API for a Java component to communicate with a resource adapter. This API is called the Common Client Interface (CCI).

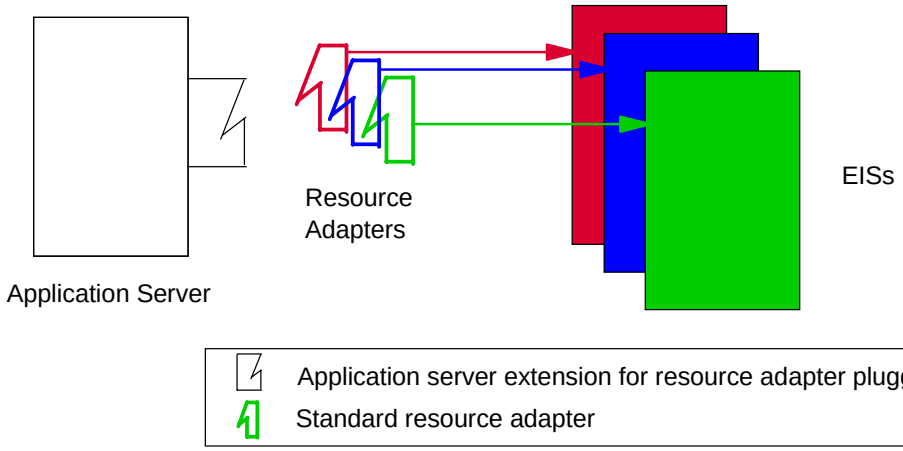
A common set of system contracts that allow the application server to manage connections, transactions, and security propagation with the resource adapter.

Resource Adapters



JCA rationale

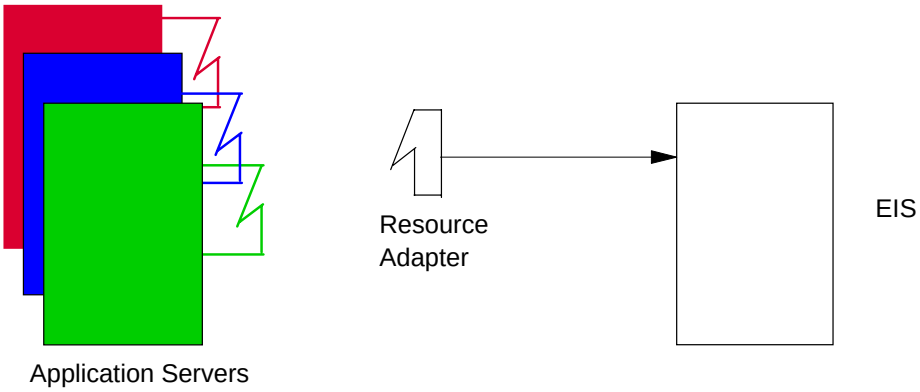
An application server extends its system only once to support the connector architecture and is then assured of connecting to multiple EISs.



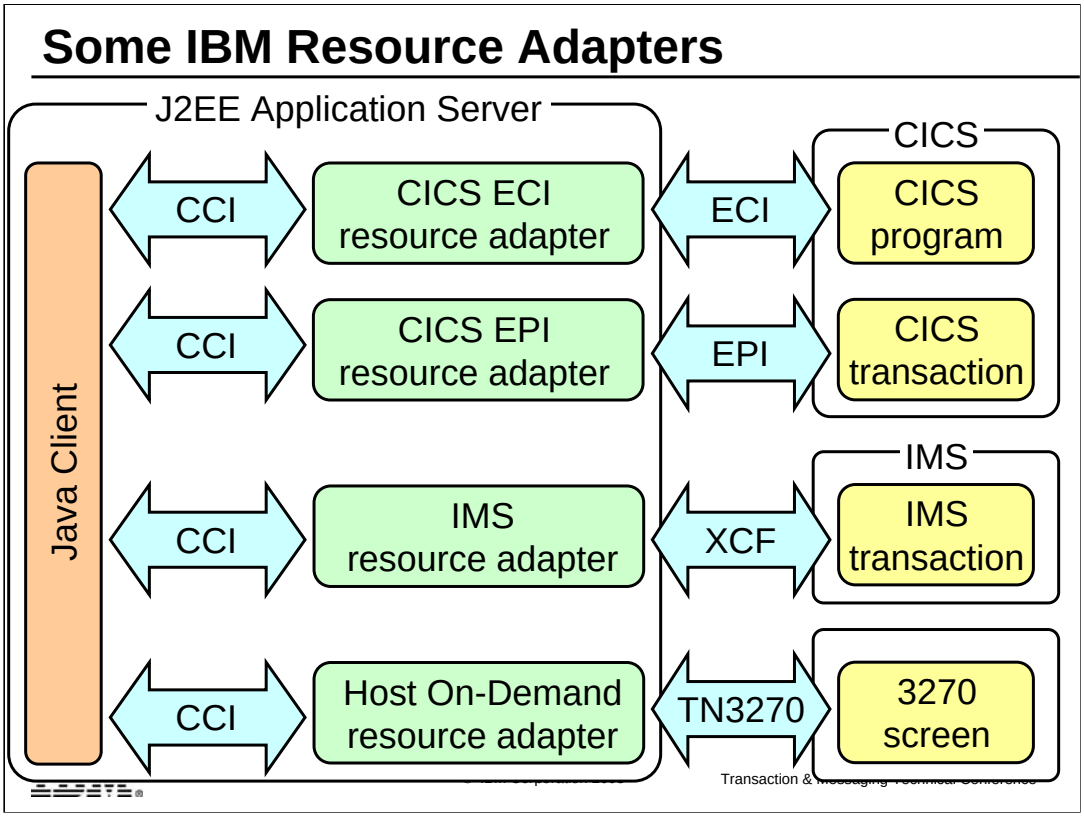
A major rationale for the JCA is to simplify the job of the EIS provider so that he has to provide a single resource adapter which will be able to be plugged into any applications server.

JCA rationale cont …

An EIS vendor provides one standard resource adapter and it has the capability to plug in to any application server that supports the connector architecture.



JCA reduces the scope of the integration problem



IBM ships the following four resource adapters

CICS ECI (cicseci.rar)

CICS EPI (cicsepi.rar)

IMS (ims.rar)

Host On-Demand (j2hod3270.rar)

J2EE Connector Architecture Specification

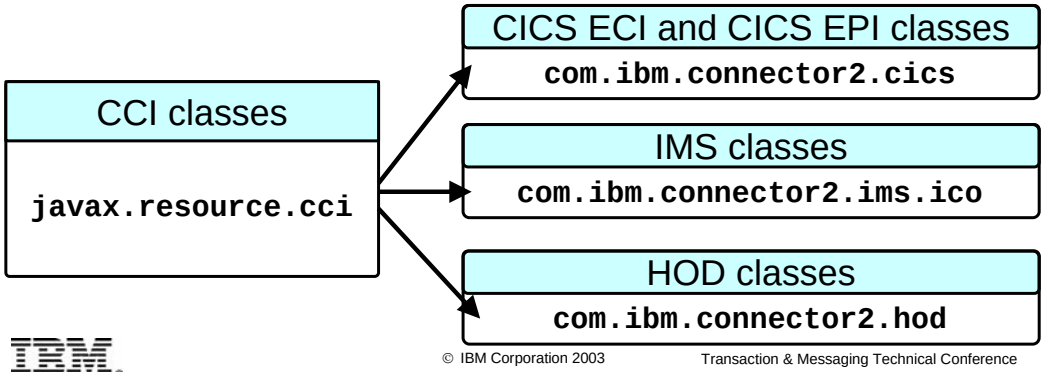
- The J2EE Connector Architecture V1.0 specification defines:
  - Common Client Interface (CCI)
    - A common API for interacting with resource adapters
    - Largely independent of a specific EIS
  - System Level Programming Interface (SPI)
    - A set of system-level contracts between a J2EE application server and EIS to provide services through the resource adapter such as connection pooling and transactions
  - Resource adapter deployment and packaging
    - A Resource Adapter Module used to deploy a resource adapter into a J2EE application server

Part Two

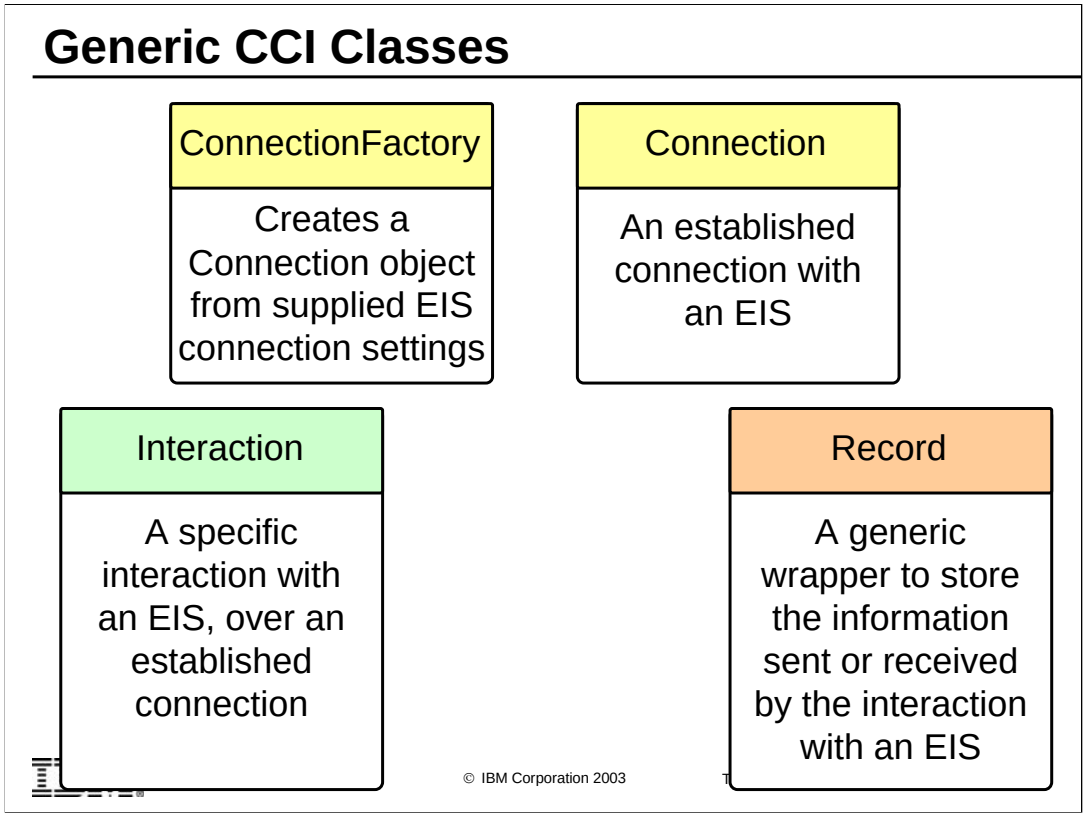
Common Client Interface

Common Client Interface

- The Common Client Interface API is used by a J2EE component to communicate with an EIS through a resource adapter
- Each resource adapter extends certain CCI classes with EIS specific methods



A program which uses the ECI resource adapter needs to import javax.resource.cci.* and com.ibm.connector2.cics.*

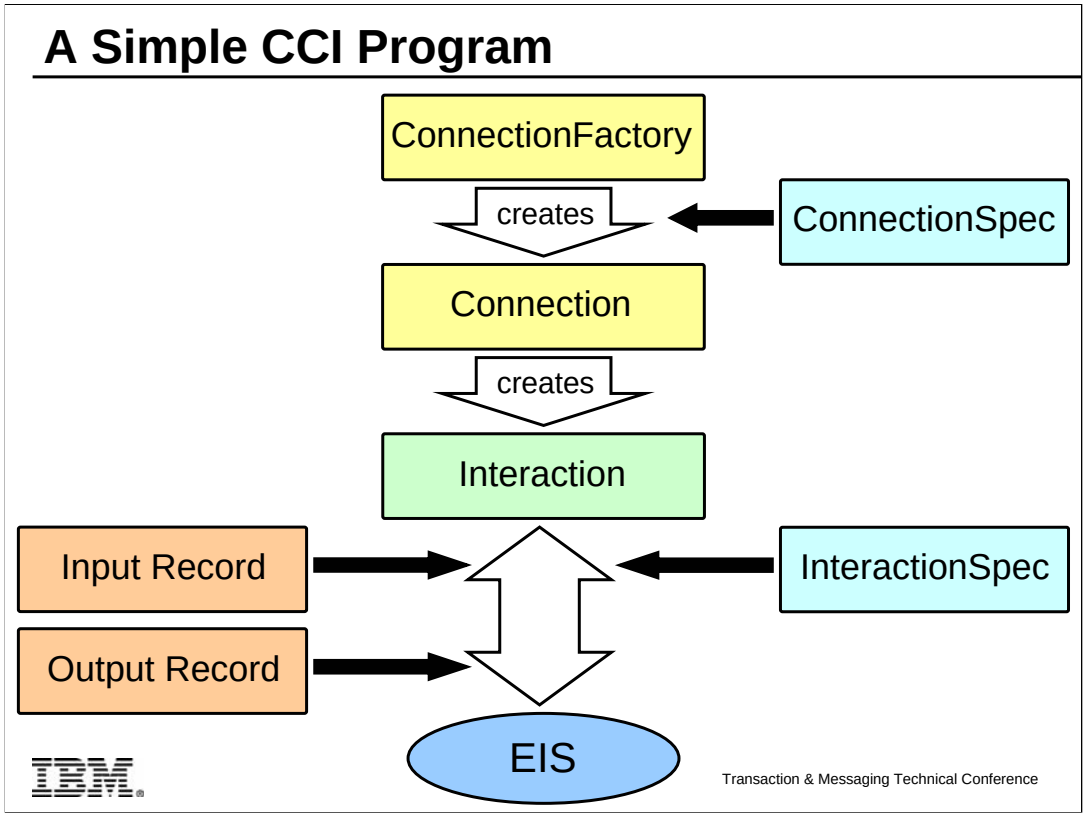


These generic classes are provided in the javax.resource.cci.* package

EIS Specific Classes

ConnectionSpec	InteractionSpec
EIS specific connection information (often overrides values set in the ConnectionFactory)	EIS specific interaction information, such as EIS transaction/program to call
ECIConnectionSpec **EPIConnectionSpec** **IMSConnectionSpec** **J2HODConnectionSpec**	**ECIInteractionSpec** **EPIInteractionSpec** **IMSInteractionSpec** **J2HODInteractionSpec**

ECI and EPI classes are provided in the com.ibm.connector2.cics.* package

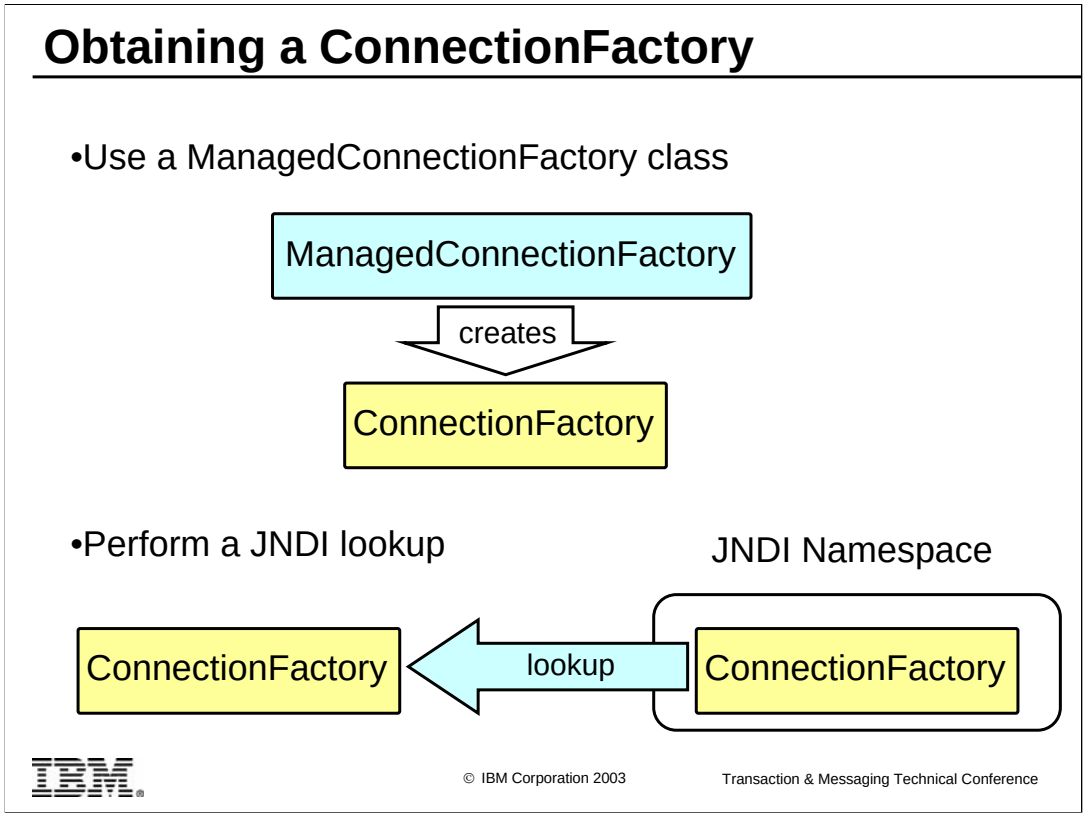


Start with a connection factory (something that makes things – in this case in makes connections) . We will see after how a connection factory itself is created.

A Connection Factory provides a getConnection method which is used to create a Connection. The ConnectionSpec class can be used to overide values set on the ConnectionFactory

The Connection class provides a createInteraction method which is used to create an Interaction. The InteractionSpec class can be used to specify interaction information such as the EIS program name and transaction identifier.

The call is made using the execute method of the Interaction class, passing the Interaction object, an input record and output record. In the case of CICS the input and output records are likely to be the same.



There are two ways to obtain a *ConnectionFactory* object:

1. Manually create a *ConnectionFactory* object:
– Instantiate a *ManagedConnectionFactory* object.
– Populate the *ManagedConnectionFactory* object with deployment values.
– Use the *ManagedConnectionFactory* createConnectionFactory() method
to create a *ConnectionFactory* object.

2. Use JNDI to lookup a *ConnectionFactory* object, which has earlier been created following the same steps as above, then bound into the JNDI name space.

The JCA spec recommends using JNDI to lookup connection factory.

Using the CCI

```
//Create and set values for ECI managed connection factory
ECIManagedConnectionFactory mcf=new ECIManagedConnectionFactory();
mcf.setConnectionURL("tcp://9.100.101.102");
mcf.setPortNumber("2006");
mcf.setServerName("A6E2CW20");
//Create a connection factory connection object
ConnectionFactory cxnf=(ConnectionFactory)mcf.createConnectionFactory();
Connection cxn=cxnf.getConnection();
//create an interaction with CICS to start program TRADERBL
Interaction ixn=cxn.createInteraction();
ECIInteractionSpec ixnSpec=new ECIInteractionSpec();
ixnSpec.setInteractionVerb(ixnSpec.SYNC_SEND_RECEIVE);
ixnSpec.setFunctionName("TRADERBL");
//Create a new record for handling the COMMAREA byte array
GenericRecord record =new GenericRecord(("abcde").getBytes("IBM037"));
//Finally execute and flow the request to CICS
ixn.execute(ixnSpec,record,record);
//Close the interaction and the connection
ixn.close();
cxn.close();
```


The following parameters of the ECIManagedConnectionfactory are set:

tcp://9.100.101.102 to be used as the Connection URL (note that the only option for this setting when deploying in WAS for z/OS is to use local:)

2006 for the port number (this is ignored if the protocol is local)

A6E2CW20 as the CICS server name -

In addition, the TRADERBL program is specified as the function name on the ECIInteractionSpec.

JavaStringRecord class in C:\Program Files\IBM\IBM CICS Transaction Gateway\samples\java\com\ibm\ctg\samples\j2eesamples is a sample CCI compliant record.

Note that, in this example, the connection factory is not retrieved from a JNDI name server, but is defined manually using the ManagedConnectionFactory class.

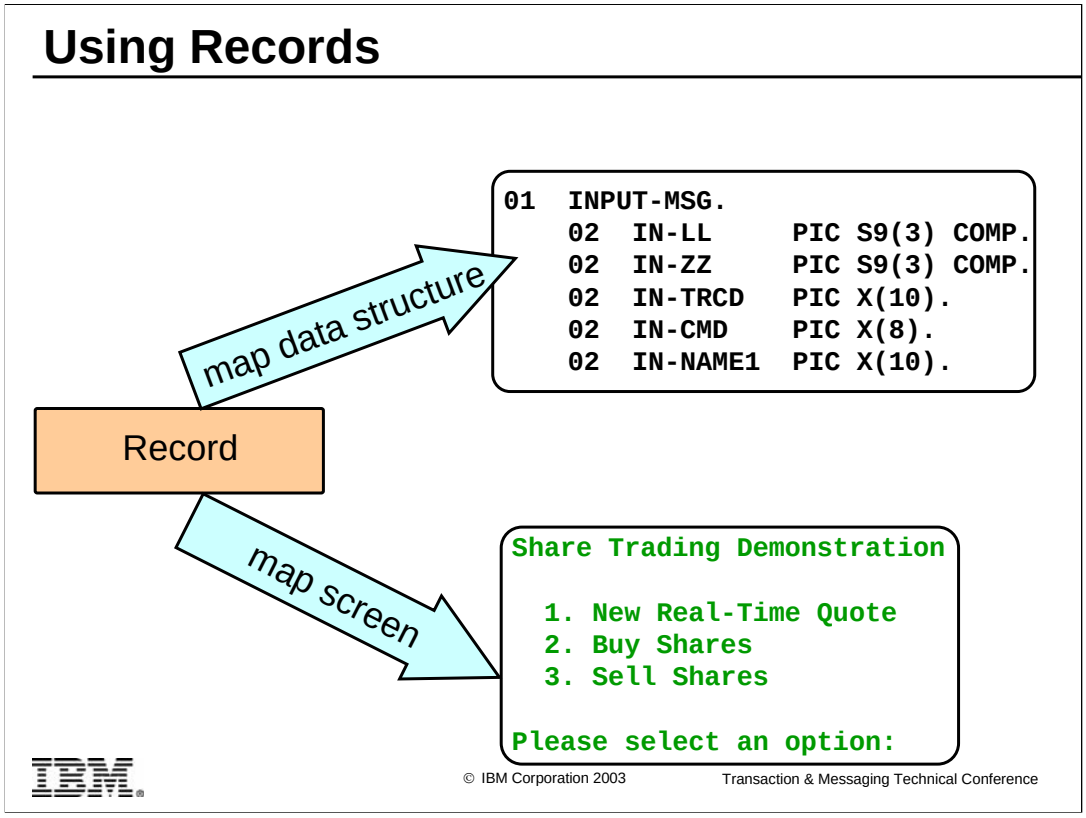
Using the CCI with JNDI lookup

```
//Obtain connection factory using JNDI
Context ic =new InitialContext();
cxnf =(ConnectionFactory)ic.lookup("java:comp/env/eis/ECICICS");
//Create connection from connection factory
cxn =cxnf.getConnection();
//Create an interaction from the connection
ixn =cxn.createInteraction();
//Create an InteractionSpec
ECIInteractionSpec ixnSpec=new ECIInteractionSpec();
ixnSpec.setInteractionVerb(ixnSpec.SYNC_SEND_RECEIVE);
ixnSpec.setFunctionName("TRADERBL");
//Create a new record for handling the COMMAREA byte array
GenericRecord record =new GenericRecord(("abcde").getBytes("IBM037"));
//Finally execute and flow the request to CICS
ixn.execute(ixnSpec,record,record);
//Close the interaction and the connection
ixn.close();
cxn.close();
```


This chart shows the same example but this time the Connection factory is created by doing a lookup of the resource reference java:comp/env/eis/ECICICS

An EJB uses a resource reference to represent a connection factory object. The JNDI string used here must match with a resource reference name that is declared in the EJB JAR deployment descriptor.

We will see later how we define Connection factories in WebSphere.



Records store the input and output data to be used during the interaction with an EIS.

The input record will contain data to pass to the EIS, and the output record will be used to store data generated by the EIS.

Instead of managing records yourself, it is typical to use the EAB feature of VisualAge for Java to generate Records using the *Import COBOL SmartGuide*. This provides getters and setters for accessing each record within the COMMAREA, and in built data conversion for all numeric and character data types.

WebSphere Studio Application Developer Integrated Edition provides similar function.

Application Development Tools

- J2EE Connector Architecture recommends using a development tool to generate CCI code
- Tools can be particularly useful for Record generation
- IBM tools supporting JCA:
  - VisualAge for Java Enterprise Edition V4
    - Uses the Enterprise Access Builder component
    - Creates Record and Command beans
  - WebSphere Studio Application Developer Integration Edition V5
    - Part of the WebSphere Studio family
    - Wizard to create Enterprise Services which can be deployed as a session bean and accessed as a Web service
    - Ships with the CICS ECI resource adapter
    - The strategic tool

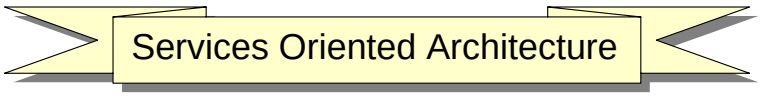
Why use Integration Edition to create a J2C enterprise service for CICS ECI when you could just code applications which use the CCI directly ?

Integration Edition has two advantages over using the CCI classes in a servlet or EJB:

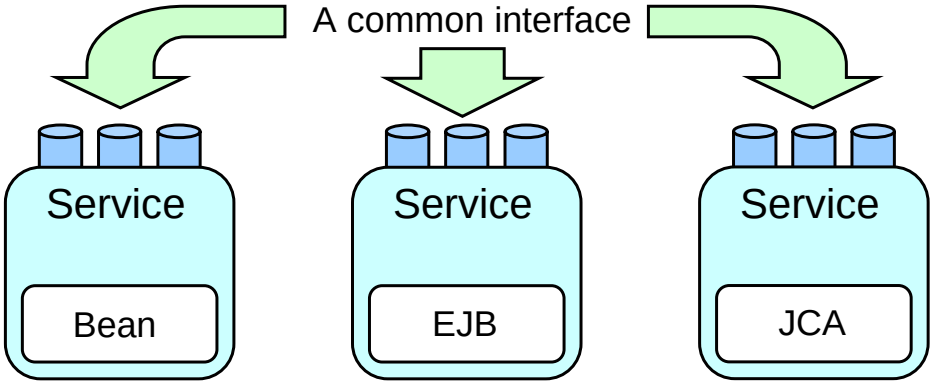
1. Integration Edition can be used to create a CICS ECI enterprise service that exposes a CICS program as a service with a common service interface. The service consumer can use a variety of protocols to invoke the service (IIOP, SOAP, JMS) using this common interface.

2. Integration Edition maps the COBOL or C defined COMMAREA data structure used by the CICS program to XSD types that are used in the enterprise service. To perform the same function using CCI classes, you would need to manually perform the COBOL/C to Java data conversions, as well as the ASCII to EBCDIC code page conversions, in a CCI Record object. Integration Edition automates this entire process.

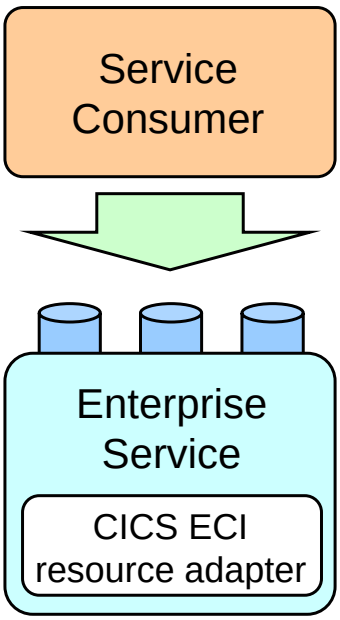
Service Oriented Architecture



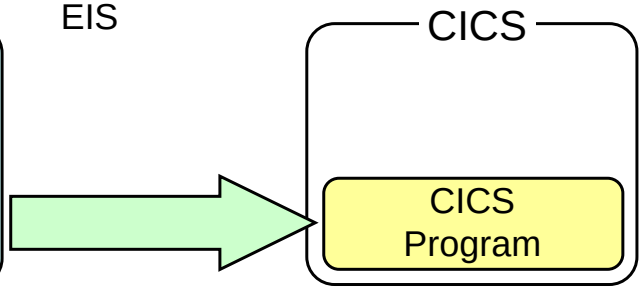
Takes existing software components residing on the network and allows them to interoperate with each other



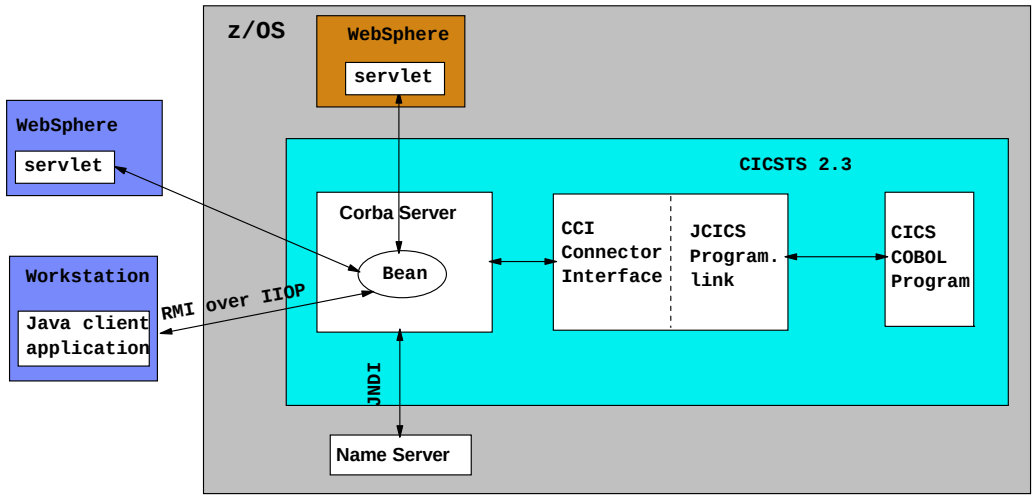
Enterprise Services and Resource Adapters



- An enterprise service uses a resource adapter to interact with an enterprise information system
- A service consumer binds to this enterprise service and exploits the functionality of the EIS without having to explicitly handle connectivity to the EIS



The CCI Connector for CICS – what is it ?



New in CICS TS V2.3

The CCI Connector for CICS (cont…)

- Enables a Java program or enterprise bean running on CICS TS 2.3 to link to a CICS server program using the CCI
- Uses a JCICS Program.link() call to access a back-end server
  - LINK
  - DPL
- Is highly optimized for execution within CICS (very little overhead involved in using it rather than a JCICS Program.link() call)
- The CICS server program
  - May be written in any of the CICS-supported languages
  - Must use a suitable COMMAREA
  - Must not do any terminal I/O

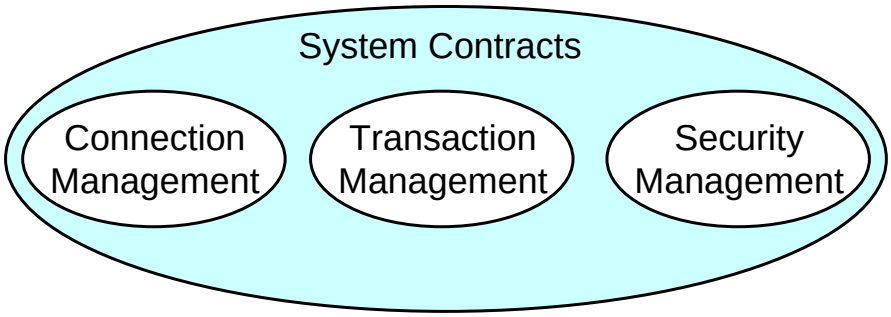
Part Three

Part Three

System Level Programming Interface

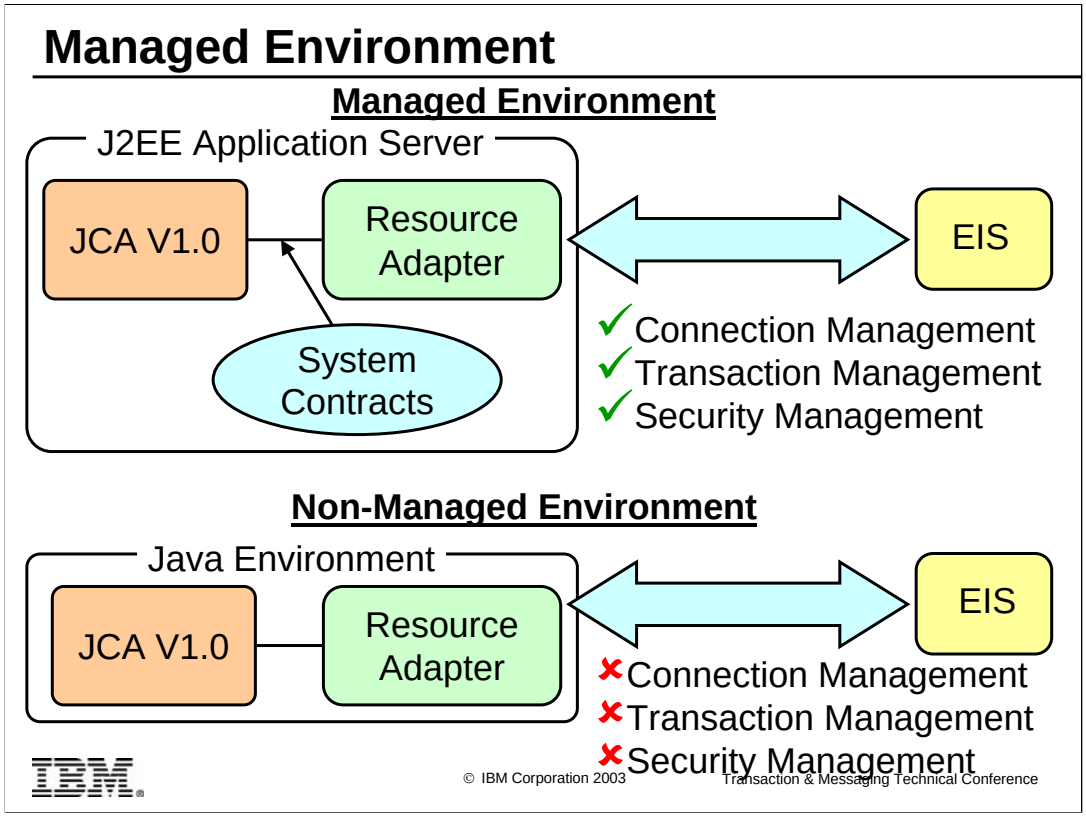
System Level Programming Interface

- Defines an API for communication between the resource adapter and application server
- Services offered through using the system level programming interface are known as **System Contracts**



To achieve the ease of interaction between the application server and EIS, the J2EE Connector Architecture defines a set of system contracts. The application server uses a resource adapter to support these contracts. The resource adapter implements the system contracts to collaborate with the application server and

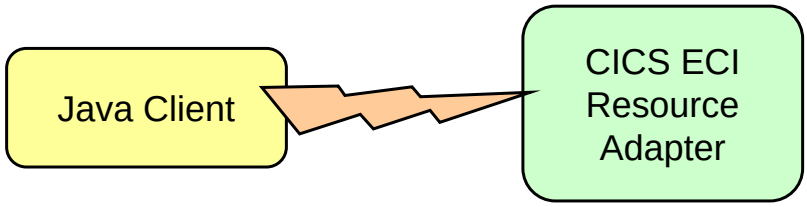
uses an EIS specific API to communicate with the EIS. Thus a resource adapter is specific to an EIS but, because it implements the system contracts, can be plugged into any J2EE compliant application server.



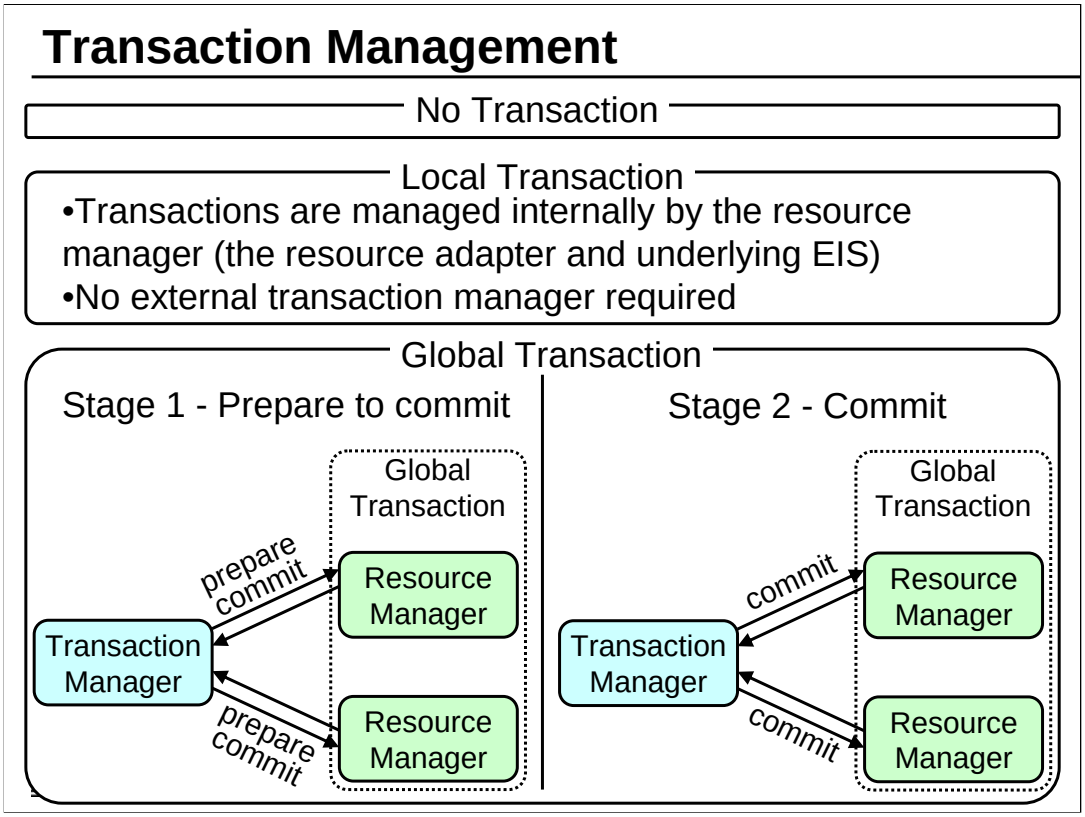
A resource adapter can be managed by any J2EE application server that supports JCA. The application server can manage the connections made to the resource adapter, the transactional support provided by the resource adapter, and the security propagated through the resource adapter.

Connection Management

- Gives an application client a connection to an EIS
- Connection pooling is supported
- An application server uses a Pool Manager to implement a connection pooling mechanism in its own implementation specific way



Connection pooling can be a significant performance benefit (see the CT08 presentation 'Using the J2EE Connector Architecture to access CICS from WebSphere Application Server for z/OS V5')



A resource manager has three options for supporting transactions:

No support

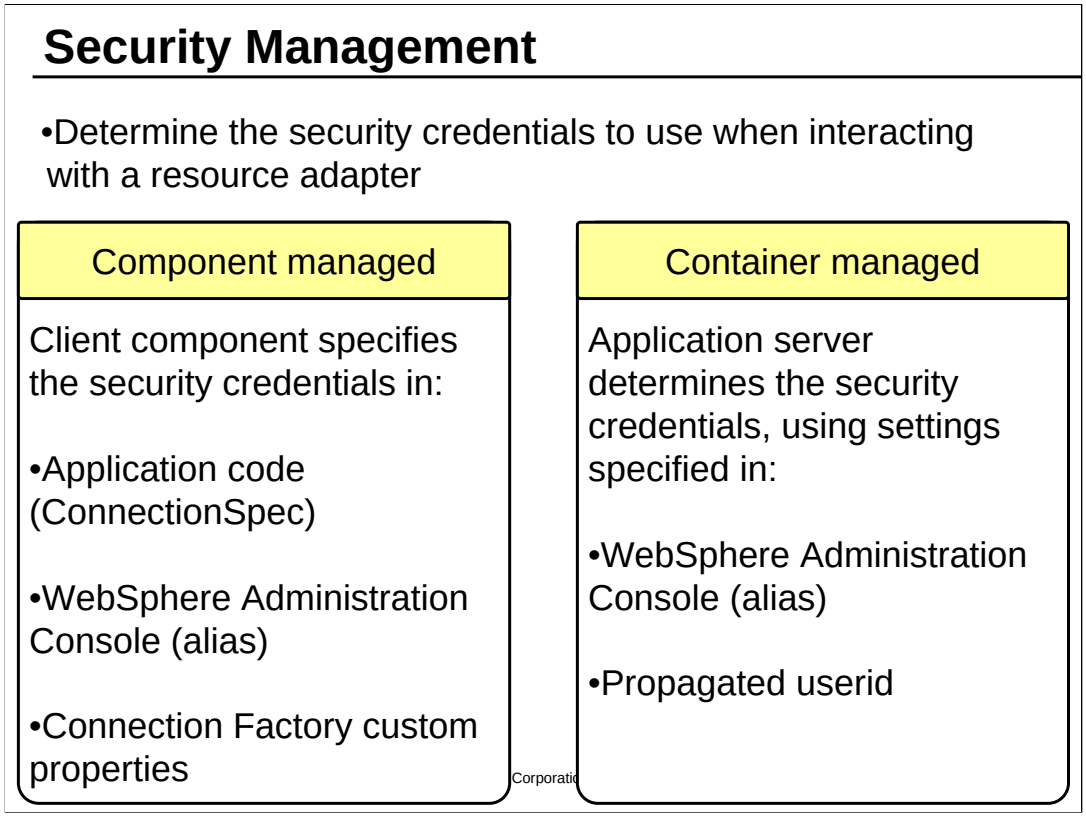
The resource manager does not support transactions.

Local transactions

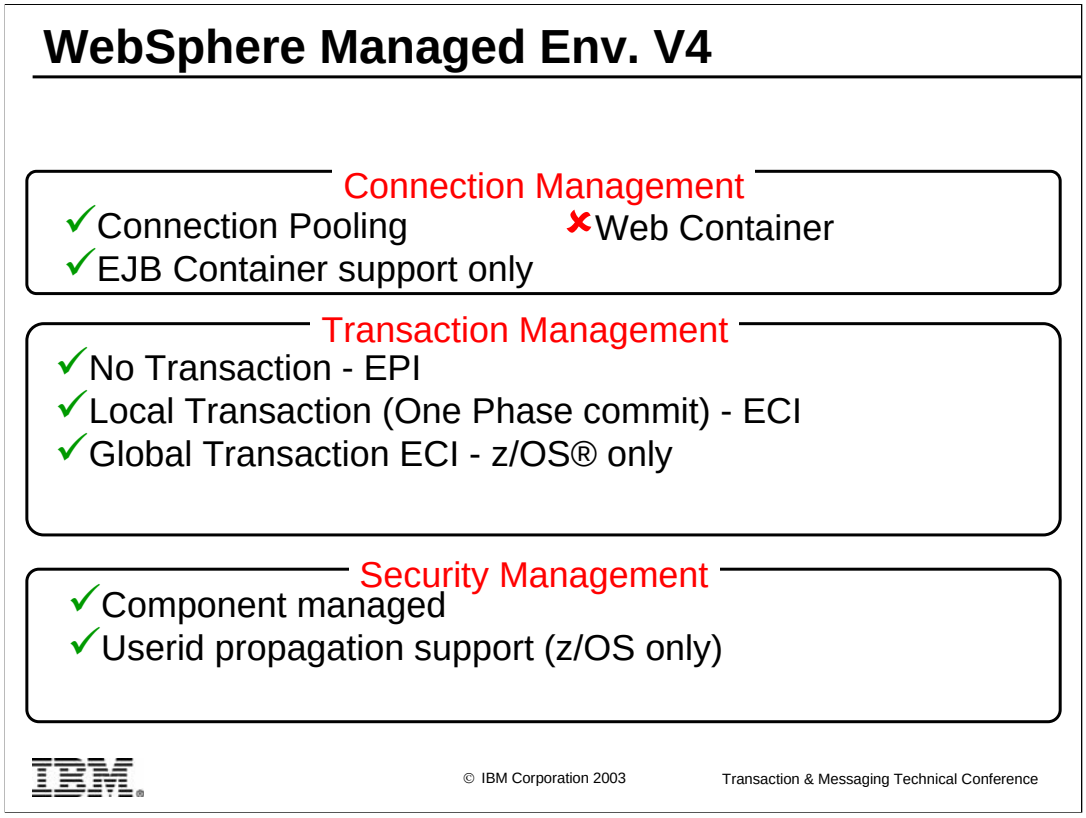
Transactions that are managed internally by the resource manager. The coordination of such transactions involves no external transaction manager. Local transactions only support one phase commit because they only reference one EIS. To support local transactions the resource adapter must implement the javax.resource.spi.LocalTransaction interface.

Global transactions

There are multiple resource managers involved and an external transaction manager must be used to coordinate the transaction using two phase commit processing. These transactions are also referred to in the J2EE Connection Architecture specification as JTA transactions, and are supported by the resource adapter implementing the javax.transaction.xa.XAResource interface.



In WebSphere Application Server V5, component and container managed authentication aliases can be specified using a Java Authentication and Authorization Service (JAAS) alias.



The following managed environment is offered by WebSphere Application Server V4:

Connection management

Connections to the CICS Transaction Gateway (often over a TCP/IP connection) are pooled for the EJB container.

Transaction management

No transaction support for the EPI resource adapter.

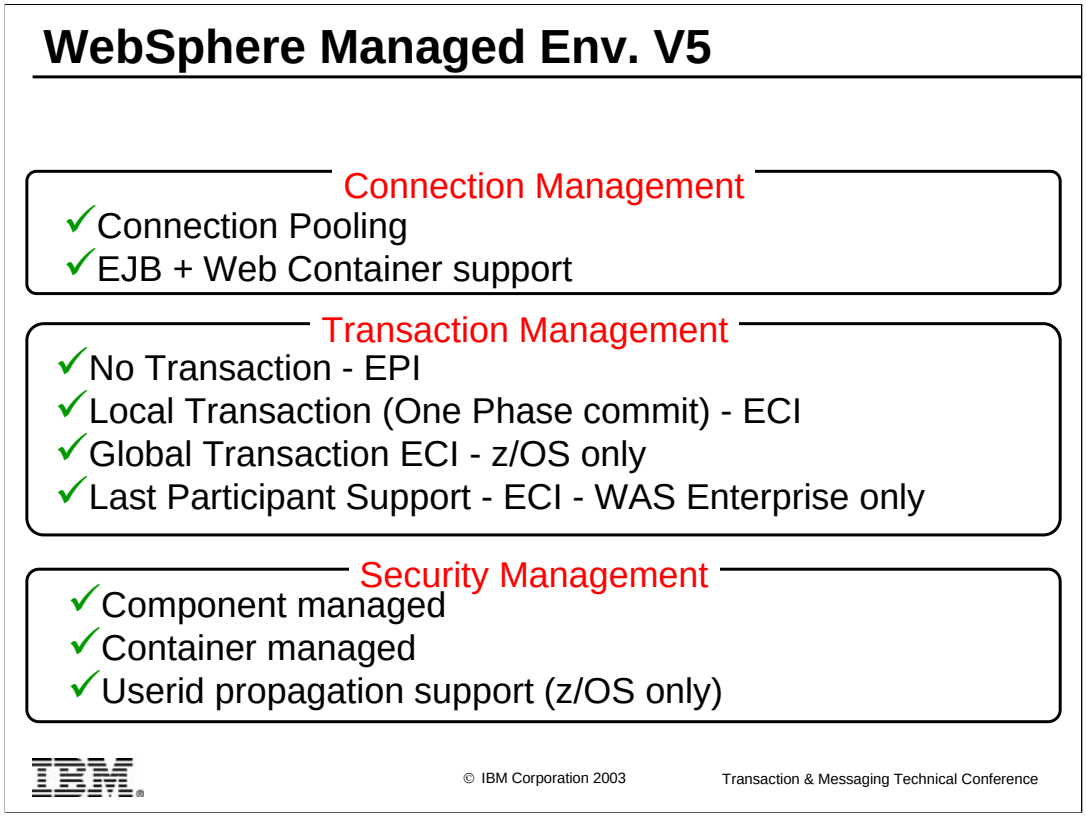
Local Transaction support for the ECI resource adapter on the distributed platforms permits multiple interactions to a CICS server to be part of a single unit of work. This unit of work is then committed or rolled backed, and all recoverable resources are modified accordingly.

Transaction management for the ECI resource adapter on z/OS is a two phase capability provided through RRS (Resource Recovery Services).

Security management

The security credentials to propagate through the CICS Transaction Gateway can be determined by the application (in the code or in the connection factory definition).

A unique solution provided by WebSphere z/OS is that it is possible to automatically propagate the WebSphere authenticated userid to CICS.



The following additional functions are added by the managed environment of WebSphere Application Server V5:

Connection management

Connection pooling in the Web container.

Transaction management

Last Participant Support (LPS) is provided when using the ECI resource adapter with WebSphere Application Server Enterprise Edition V5. LPS allows a resource adapter that does not implement the XAResource interface to participate in a global transaction if it is the only resource manager (which does not implement the XAResource interface) included in the global transaction. With LPS, in some circumstances therefore it is possible to coordinate the updates of a CICS program with other updates made by an EJB running in WebSphere.

Security management

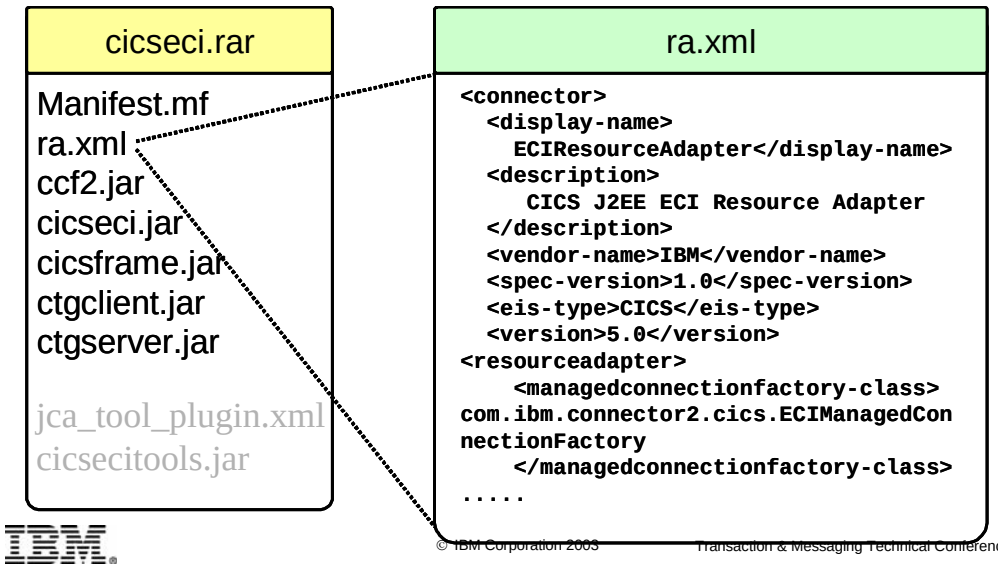
The security credentials to propagate through the CICS Transaction Gateway can be determined by the container. Container managed signon allows the application server to determine the security context which is passed to CICS – in practice, on the distributed platforms, this is somewhat limited support since it is provided by specifying a container managed JAAS alias which is used for all connections for a particular connection factory.

Part Four

Resource Adapter Deployment & Packaging

Resource Adapter Deployment & Packaging

•Resource Adapters are packaged in a Resource Adapter Archive (RAR file)

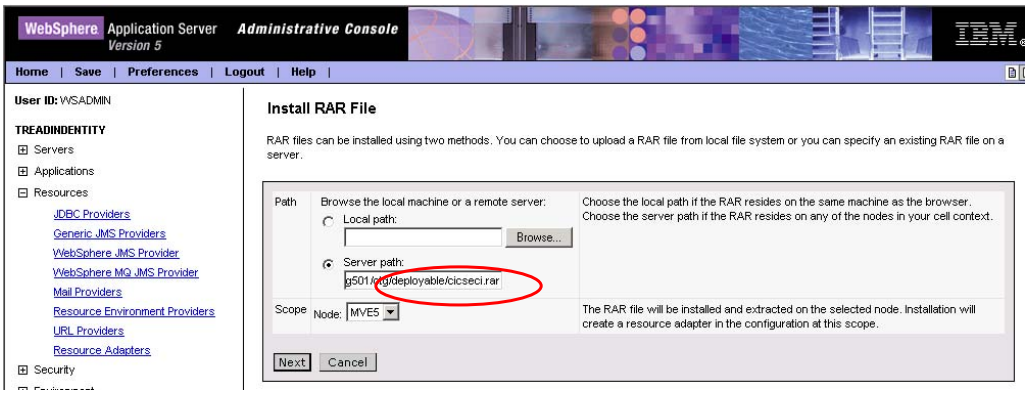


This chart shows the ra.xml file for the CICS ECI resource adapter for z/OS.

The ra.xml file is a deployment descriptor for the resource adapter i.e it defines the names of the classes as well as what type of support is provided for the system contracts.

Installing the resource adapter

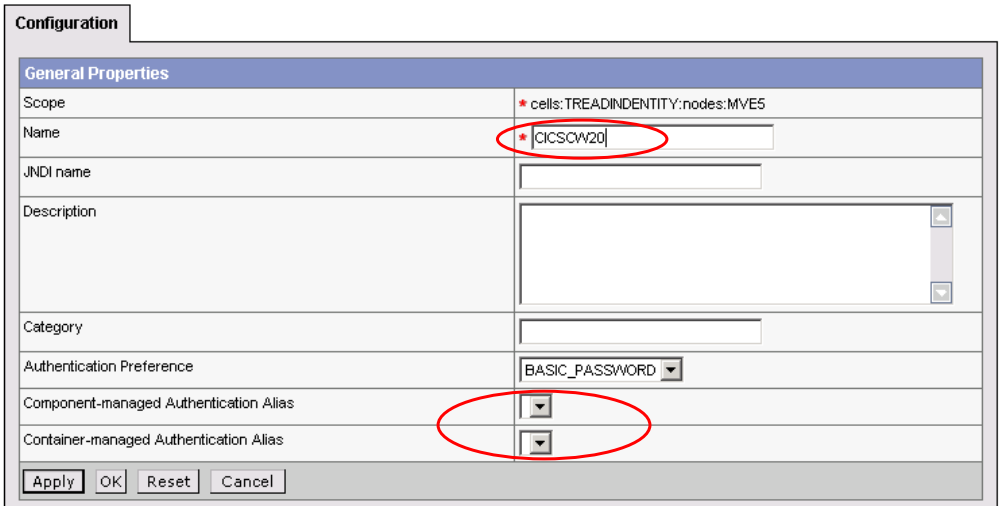
- Install the RAR file directly using the WebSphere Admin Console



The following sequence of charts show how the ECI resource adapter is installed, how connection factories are created and how an application is deployed to use a connection factory. More detail is provided in the presentation CT08 ‘Using the J2EE Connector Architecture to access CICS from WebSphere Application server for z/OS V5’.

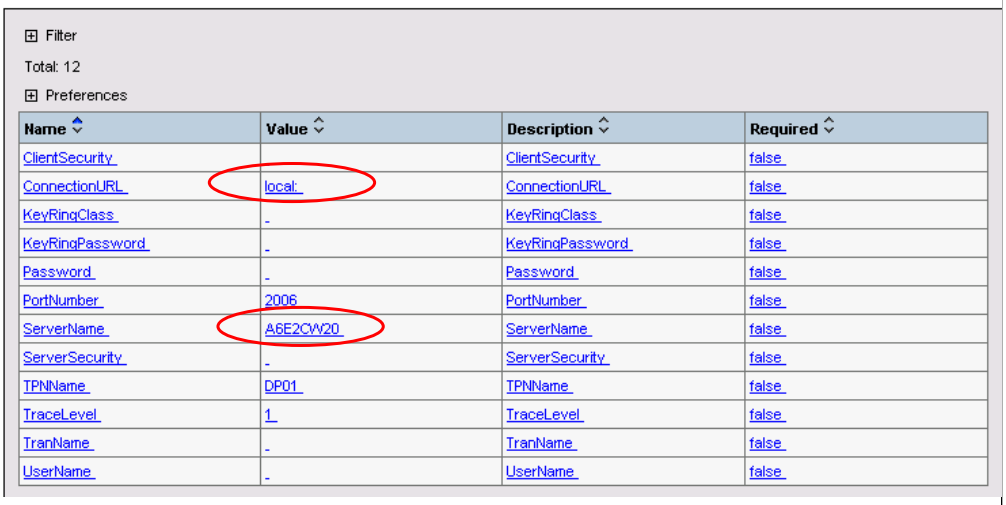
Create a Connection Factory

- Specify connection factory name (CICSCW20)
- Container and component-managed aliases control what userid is passed to CICS



Specify Connection custom properties

- Specify connection protocol (can be local or remote connection)
- Specify CICS applid, CICS transaction etc.



⊞ Filter

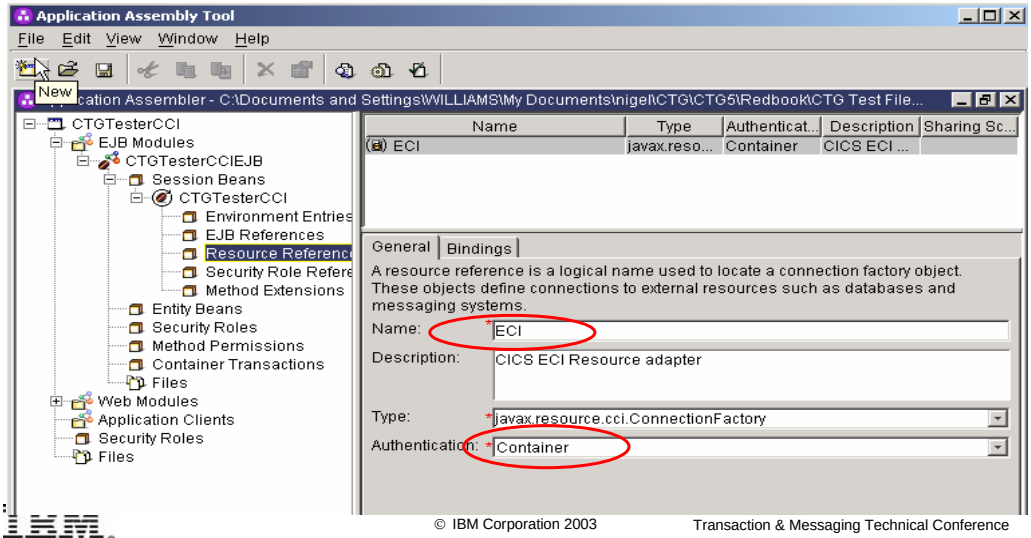
Total: 12

⊞ Preferences

Name	Value	Description	Required
ClientSecurity		ClientSecurity	false
ConnectionURL	local:	ConnectionURL	false
KeyRingClass	-	KeyRingClass	false
KeyRingPassword	-	KeyRingPassword	false
Password	-	Password	false
PortNumber	2006	PortNumber	false
ServerName	A6E2CW20	ServerName	false
ServerSecurity	-	ServerSecurity	false
TPNName	DP01	TPNName	false
TraceLevel	1	TraceLevel	false
TranName	-	TranName	false
UserName	-	UserName	false

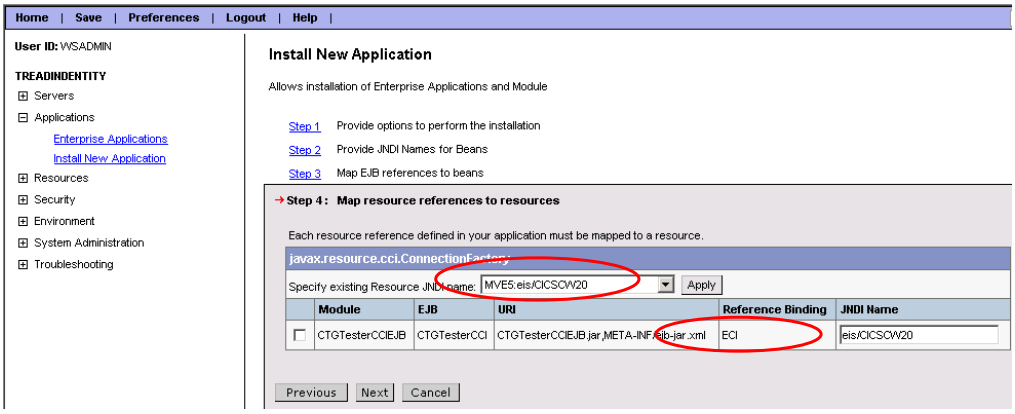
Deploying an application

- Use the Application Assembly Tool (AAT) to prepare the enterprise application (EAR file) for WebSphere
- Create a resource reference for the Connection Factory



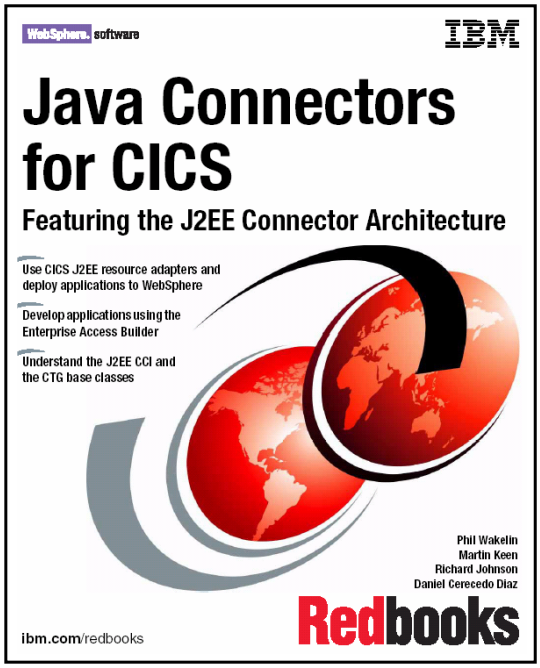
Deploying the application to WebSphere

- Use the WebSphere Admin console to deploy the application
- Map the application Connection Factory resource ('ECI') reference to the previously created Conection factory ('MVE5:eis/CICSCW20')



More Information

- Java Connectors for CICS –SG24-6401
- CICS Transaction Gateway V5, The WebSphere Connector for CICS - SG24-6133-01
- Exploring WebSphere Studio Application Developer Integration Edition V5 - SG24-6200
- Download at www.ibm.com/redbooks



Trademarks

- The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:
  - AIX
  - CICS
  - CICS Transaction Gateway
  - DB2
  - DFSMS
  - IBM
  - IMS
  - Language Environment
  - OS/390
  - RISC System/6000
  - RACF
  - RMF
  - S/390
  - VisualAge
  - WebSphere
  - z/OS
  - zSeries
- Java and all Java based trademarks and logos are trademarks or registered trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.
- SUN is a registered trademark of Sun Microsystems, Inc. in the United States, other countries, or both.
- UNIX is a registered trademark of The Open Group in the United States and other countries.
- Linux is a registered trademark owned by Linus Torvalds.
- Microsoft, Windows, Windows NT, Windows 2000 and the Windows logo are trademarks of Microsoft Corporation in the United States, other coutries, or both.
- Other company, product or service names may be trademarks or service marks of others.