IBM Software Group

IBM® WebSphere® Application Server V6

Java™ 2 Connector Architecture (JCA) 1.5

e business on demand.

Updated January 24, 2005

Goals

- Discuss the existing features in the first version (1.0) of the Java 2 Connector Architecture (JCA)
- Provide an overview of the new features of the JCA 1.5 specification
- Understand how the new JCA (1.5) architecture extends the functionality of message-driven beans (MDB) to support JMS and non-JMS messaging systems

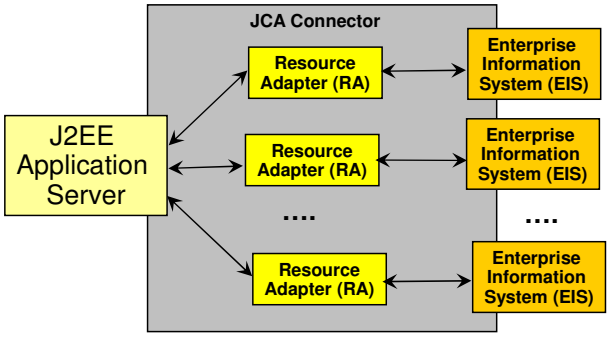
Agenda

- JCA – Quick Overview
- JCA 1.5 - New Features
  - Inbound Communication
  - Message Inflow
  - Transaction Inflow
  - Lifecycle Management
  - Work Management
- Summary and References

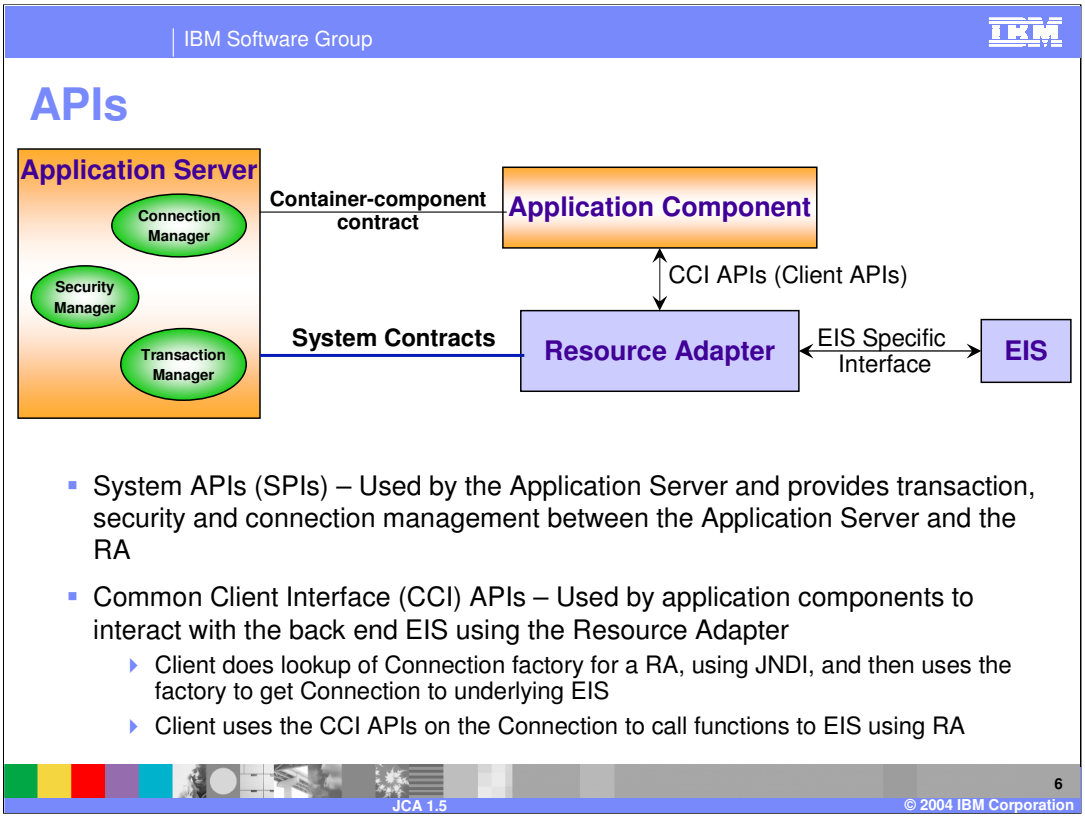
Section

JCA 1.0 Overview

Quick Overview



- JCA defines a standard that allows J2EE applications to interact with external Enterprise Information Systems (EIS)
- It provides a set of APIs that application components can use to retrieve information from a back end EIS
- To integrate with a J2EE Application Server, each EIS has its own Resource Adapter (RA) that is installed on the Application Server.



1. The Resource Adapter is installed in the Application Server using its deployment descriptor. To clearly demonstrate the various contracts that JCA defines, the resource adapter and application component are being shown (logically) outside the Application Server.

2. Connection management: The connection management contract enables an application server to pool connections to an underlying EIS, and enables application components to connect to an EIS
3. Transaction management: The Transaction Management contract between the Transaction Manager and an EIS enables an Application Server to manage transactions across multiple resource managers
4. Security: The Security contract enables a secure access to an EIS.

Sample Client Code – Getting Connection

- **Deployment Info:**
  res-ref-name: **eis/MyEIS**
  res-type: **javax.resource.cci.ConnectionFactory**
  res-auth: **Application or Container**

- **Client Application code:**

```
// obtain the initial JNDI Naming context
Context initctx = new InitialContext();

// perform JNDI lookup to obtain the connection factory
javax.resource.cci.ConnectionFactory cxf = (javax.resource.cci.ConnectionFactory)
    initctx.lookup("java:comp/env/eis/MyEIS");

// Get Connection from the factory
javax.resource.cci.Connection cx = cxf.getConnection();

// Client uses CCI APIs on the Connection class to access EIS functions
....
// close connection
cx.close();
```

Section

JCA 1.5

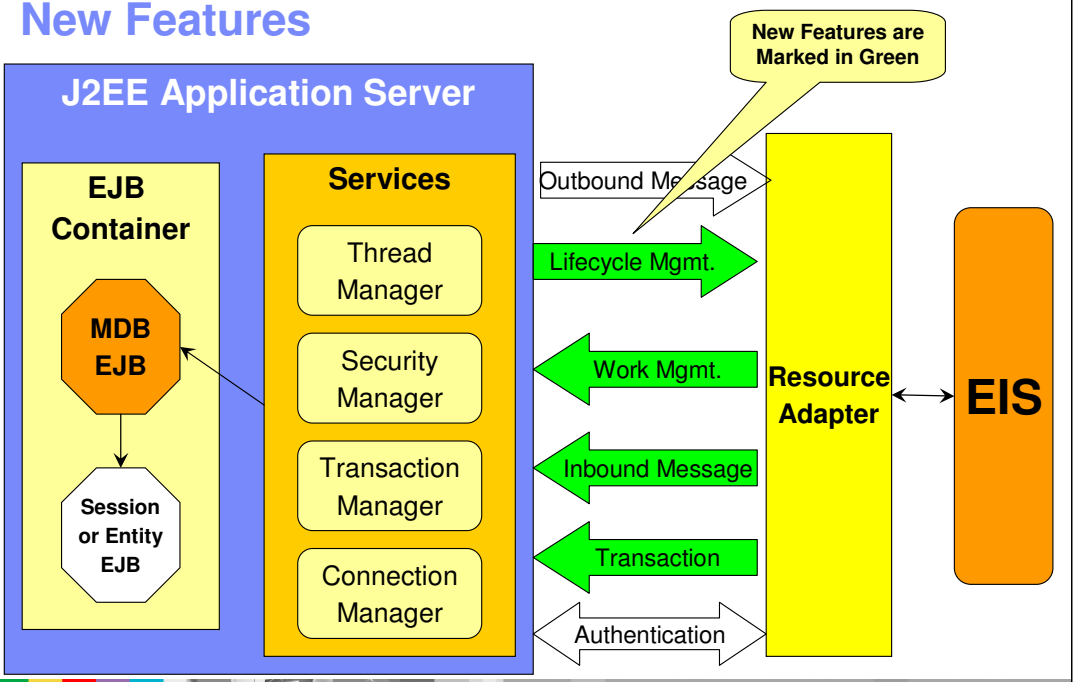
JCA 1.5 – New Features

- **Inbound Communication**
  - Allows external EIS systems to communicate with the application components deployed on the Application Server
- **Transaction Inflow**
  - Allows a Resource Adapter to propagate an existing two phase transaction into an Application Server
- **Message Inflow**
  - Generic contract to plug-in a variety of Message Providers into an Application Server

JCA 1.5 – New Features (cont.)

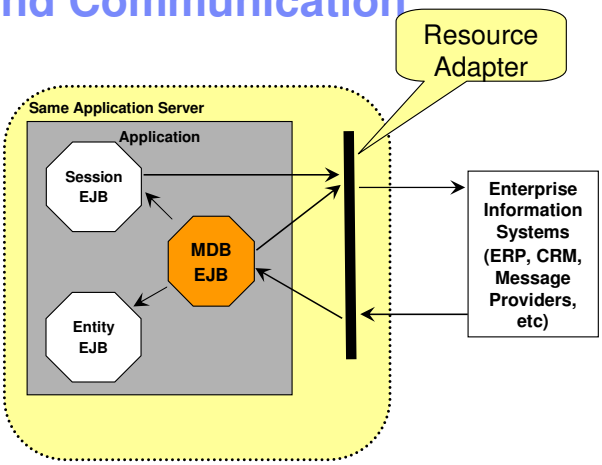
- **Lifecycle Management**
  - This feature provides the J2EE Application Server with the ability to control the life cycle of the Resource Adapter deployed in it's environment.
- **Work Management**
  - External EIS systems can now submit work to the Application Server and the submitted work is executed in the Application Server's address space

New Features



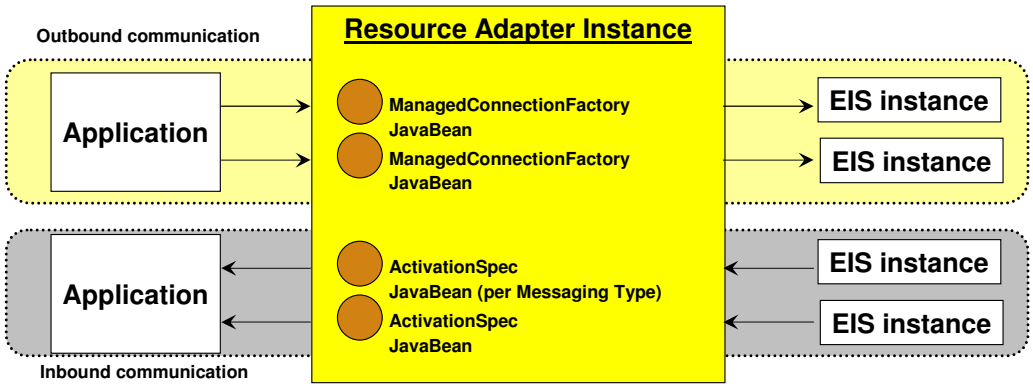
Inbound and Outbound Communication

- Introduces inbound and outbound communication between EIS and EJB Container
  - JCA 1.0 supported only outbound communication (EJB to EIS)
- Allows EIS to inject work into the J2EE container with a transactional context initiated by the EIS



Inbound and Outbound Communication

- Single RA instance may provide bi-directional connectivity to multiple EIS instances
  - ManagedConnectionFactory JavaBean for outbound connectivity to a single EIS instance
  - ActivationSpec JavaBean for inbound connectivity from an EIS instance



•When a ManagedConnectionFactory JavaBean is created, it may inherit the ResourceAdapter JavaBean (which represents the resource adapter instance) configuration information, and overrides specific global defaults, if any, and may add other configuration information specific to outbound connectivity.

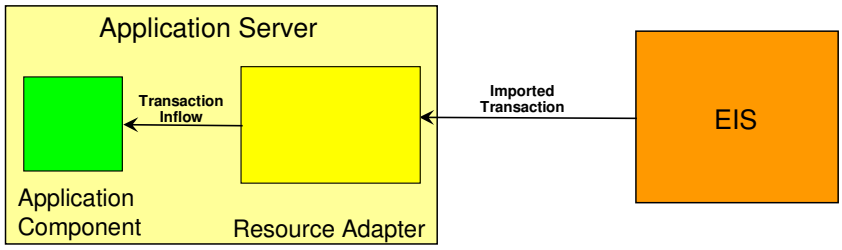
•Outbound communication is initiated by an application and the communication occurs in the context of an application thread, even though resource adapter threads may be involved in the interaction. RA may use the work management contract to request threads to do work.

•When an ActivationSpec JavaBean is created, it may inherit the RA JavaBean (which represents the RA instance) configuration information, and overrides specific global defaults, if any, and may add other configuration information specific to inbound connectivity.

•Inbound communication is initiated by an EIS instance and the communication occurs in the context of a resource adapter thread. There are no application threads involved. Note, a resource adapter may use the work management contract to request threads to do work

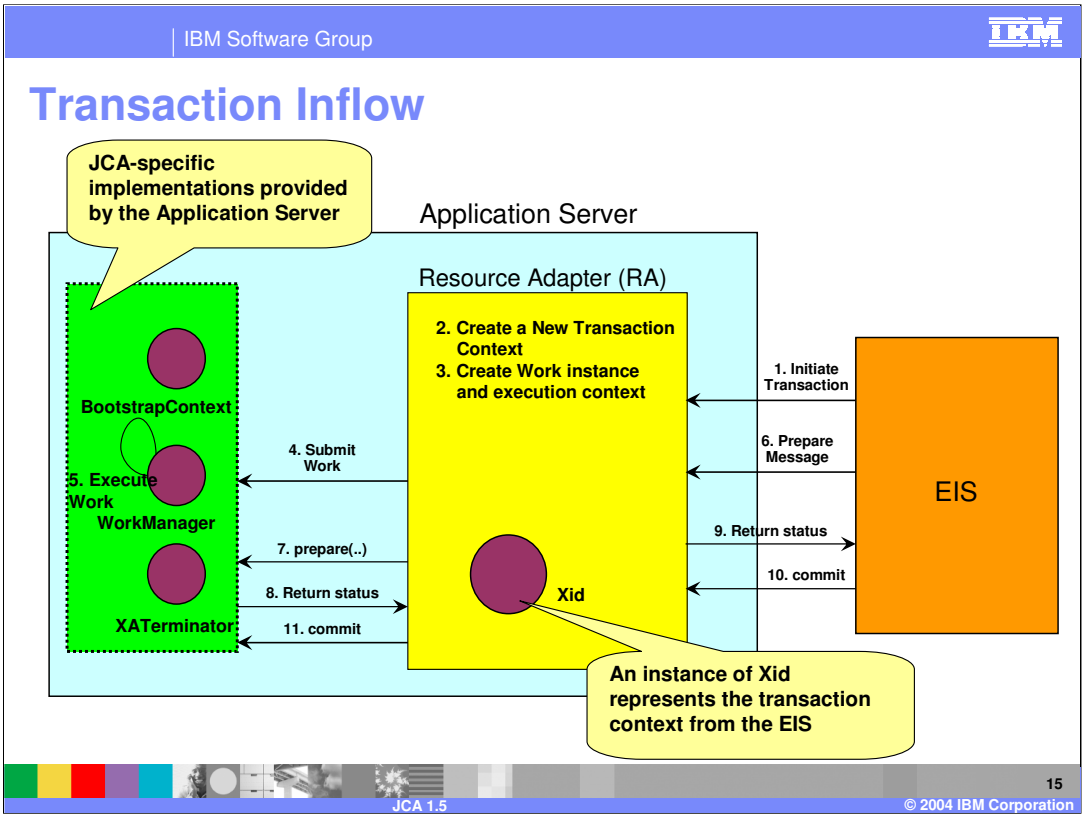
Transaction Inflow

- This feature allows an EIS to initiate a transaction and propagate the same to the Application Server using the Resource Adapter
- Defines the standard form to represent the transaction context imported by the Resource Adapter
- Transaction Inflow associates the work done by the Resource Adapter as part of the imported transaction (from EIS)



the resource adapter has to be able to “understand” the specific format and wire protocol used by the EIS to represent and transmit the transactional context.

Additionally, the resource adapter has to be able to translate that EIS-specific transactional context into a format that is compatible with the J2EE standards,



The resource adapter constructs a Work instance, which is expected to do work as part of the transactional message, and also creates an ExecutionContext instance containing the constructed Xid. It then submits the Work instance along with the ExecutionContext instance to the Application Server's WorkManager for execution.

The Application Server's WorkManager accepts the submitted Work instance and recreates the execution context for the Work instance. That is, the work to be done is enlisted as part of the imported transaction. It then calls the run method on the Work object.

Note, however, all the work done by the Work object may not be part of the transaction. For example, the Application Server may suspend the imported transaction depending on the transaction preference of the bean method that may be invoked.

The EIS sends a prepare message for a particular transaction. The resource adapter obtains an XATerminator instance from the Application Server via the getXATerminator method of the BootstrapContext instance.

Note, this step may be done at any time, and the obtained XATerminator instance may be used for transaction completion flows across multiple imported transactions.

The XATerminator implementation should be thread-safe and reentrant. The resource adapter calls the prepare method of the XATerminator instance with an appropriate Xid instance, and returns the outcome of the prepare operation to the EIS.

When the EIS sends a commit message for the transaction, the resource adapter calls the commit method of the XATerminator instance with an appropriate Xid instance.

Transaction Inflow – Crash Recovery

- When the Application Server recovers from the crash, it should recover the state of all transactions that were successfully prepared before the crash, and complete them upon receiving a commit method or rollback method call
- If the RA detects the failure of the EIS while the transaction is active (transaction completion has not begun), it cancels all active transactions that originated from the EIS
- If the RA detects the failure of the EIS while the transaction is in-doubt (the transaction has already been prepared), it waits for the EIS to recover. When the EIS recovers, it queries the EIS for a list of in-doubt transactions. It then completes the in-doubt transactions

Message Inflow

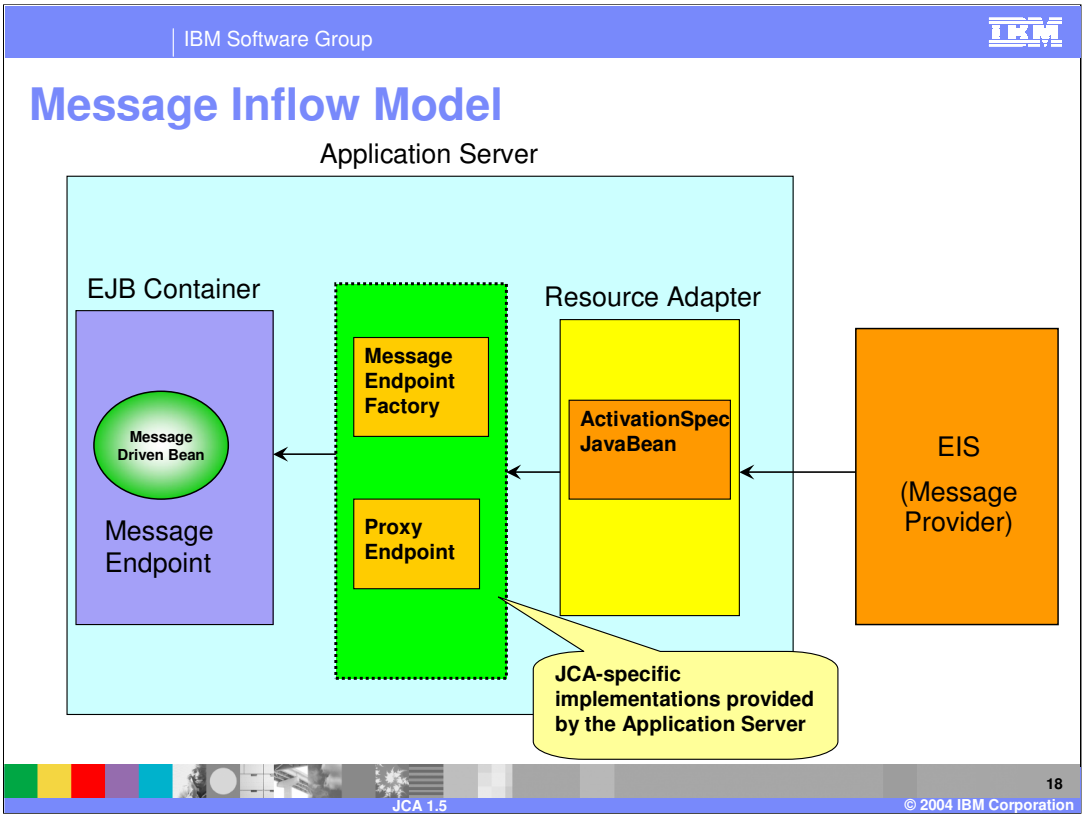
- Message Inflow defines a generic contract that allows an RA instance to asynchronously deliver messages to *message endpoints* residing in the Application Server
- In EJB terminology, a message endpoint is a message-driven bean that is deployed on a J2EE Application Server
- Message-Driven beans can receive messages from various types of message sources:
  - JMS based message providers
  - Non JMS-based message providers (SMTP e-mail server)

JCA 1.5 defines a generic interface to integrate various messaging systems with a J2EE Application Server.

These messaging systems can invoke the application components by providing an ActivationSpec Java Bean.

With the extended definition of message-driven and the new JCA architecture, message-driven beans can be

configured to accept messages from JMS and non-JMS messaging systems



1. At startup, the Application Server invokes the endpointActivation on the Resource Adapter.
2. The ActivationSpecBean and the MessageEndPointFactory instances are passed as parameters
3. When there is an incoming message, the RA uses the factory to create an endpoint instance
4. It then invokes the endpoint interface method that the MDB implements

Message Inflow

- The Resource Adapter (RA) for the messaging system provides
  - List of message listener types (fully qualified class names) it supports
  - ActivationSpec Java Bean, one for each supported message listener type
    - ActivationSpec contains a set of configurable properties that is used to specify endpoint activation configuration information during endpoint deployment
- The Application Server provides the MessageEndPointFactory implementation. The RA instance uses it to create message end point instances to deliver messages

•The Resource Adapter supports methods used for endpoint activations and deactivations

•The endpointActivation method (in the RA instance) is called by the Application Server when the message endpoint (MDB) is activated

•With the method call, the Application Server passes the ActivationSpec JavaBean and MessageEndPointFactory instances to the RA

•IMPORTANT:

•When a new message arrives, the Resource Adapter uses the MessageEndPointFactory implementation to create a Endpoint instance (MDB)

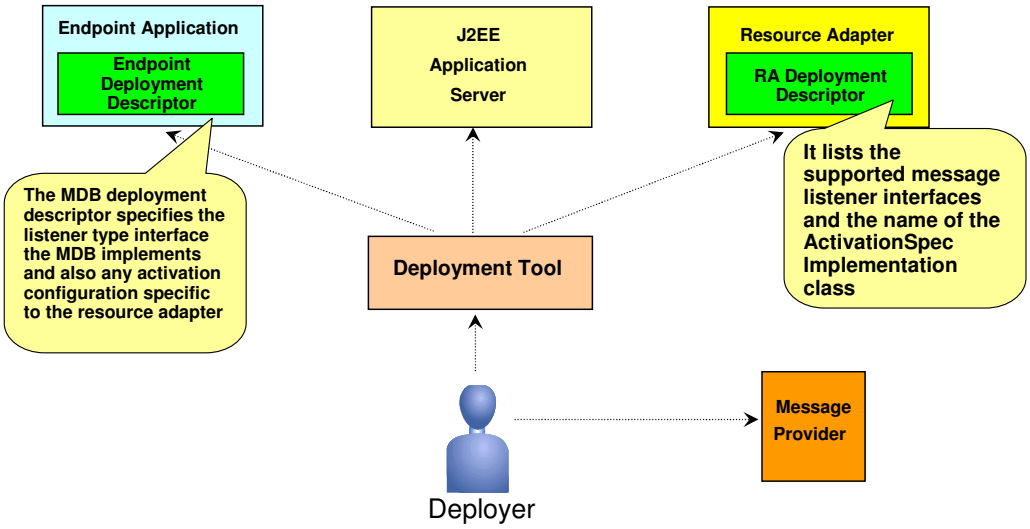
•Checks if the endpoint implements the RA specific interface

•If yes, invokes the appropriate method implemented by the MDB

Message Inflow – Various Perspectives

- Developers perspective
  - Developer provides the ***message endpoint*** – MDB that implements one of the message listener interfaces supported by the Resource Adapter
- Deployer perspective
  - The deployment descriptor of the MDB specifies the message listener interface that it implements and also any activation configuration information for the EIS
  - The RA deployment descriptor specifies the name of the ActivationSpec JavaBean class and the supported listener interfaces
  - The deployer uses a deploy tool to create an instance of ActivationSpec JavaBean and populate it with the activation configuration specified in the MDB's deployment descriptor

Message Endpoint Deployment



Sample Deployment Descriptors

```
...
<message-driven id="mdbCMT_Tx_Required_Bnd">
    <ejb-name>mdbCMT_Tx_Required</ejb-name>
    <ejb-class>mysample.mdbCMT_Tx_RequiredBean</ejb-class>
    <messaging-type>javax.jms.MessageListener</messaging-type>
    <transaction-type>Container</transaction-type>
    <activation-config>
        <activation-config-property>
            <activation-config-property-name>name</activation-config-property-name>
            <activation-config-property-value>mdbCMT_Tx_Required</activation-config-property-value>
        </activation-config-property>
    </activation-config>
    ....
</message-driven>
...
```

ejb-jar.xml (EJB Depl. Desc.)

```
...
<inbound-resourceadapter>
   <messageadapter>     <messagelistener>
        <messagelistener-type>javax.jms.MessageListener</messagelistener-type>
        <activationspec>
            <activationspec-class>fvt.adapter.ActivationSpecImpl</activationspec-class>
            <!--Required Property-->
            <required-config-property>
                <config-property-name>name</config-property
            </required-config-property>
        </activationspec>
    </messagelistener>     </messageadapter>
</inbound-resourceadapter
....
```

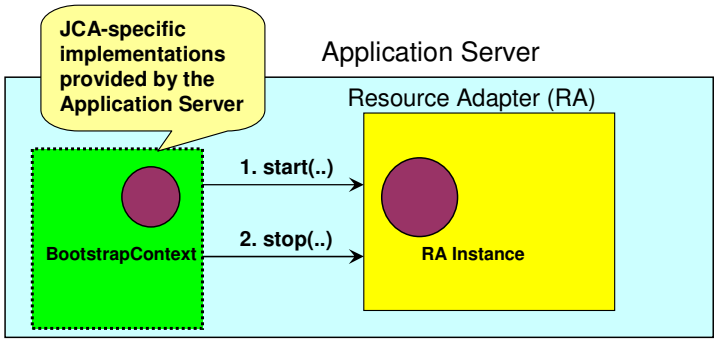
Message Listener Type

Name of the ActivationSpec JavaBean class

ra.xml (RA Depl. Desc.)

Lifecycle Management

- Provides a mechanism for an Application Server to manage the lifecycle of a RA instance
  - The Application Server can bootstrap a RA instance (in Application Server address space) during its deployment or server startup
  - To notify the RA instance during its undeployment or during a server shutdown



•The implementation class name of the ResourceAdapter interface is specified in the resource adapter deployment descriptor.

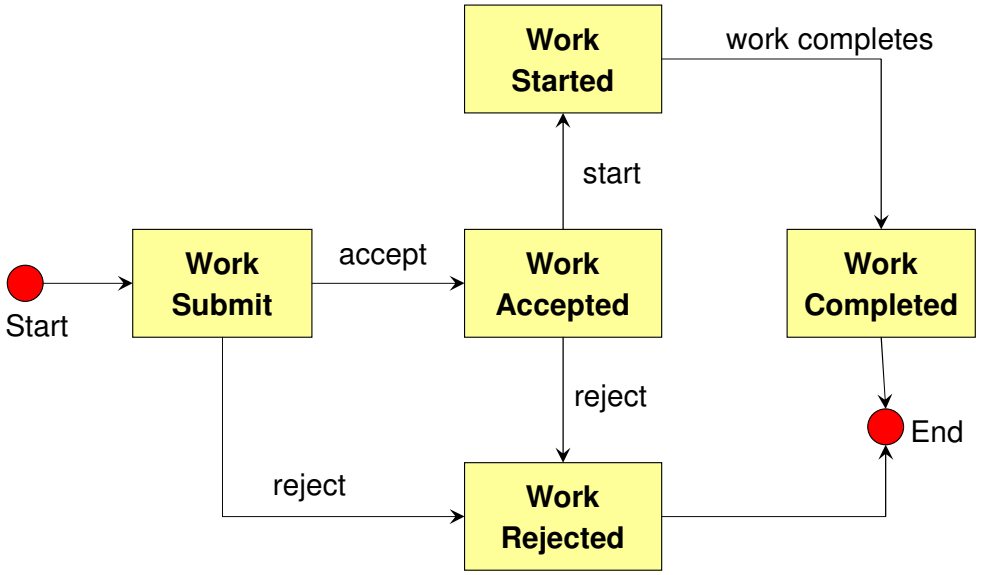
•At server startup, the Application Server bootstraps the RA instance by invoking the start method on it. It passes a handle to the BootstrapContext instance on the Application Server.

•The RA instance can use the BootstrapContext handle to retrieve the WorkManager implementation on the Server and submit work.

Work Management

- The new specification defines a mechanism for a Resource Adapter (RA) instance to submit work to the Application Server
- JCA Objects involved with Work Management:
  - WorkManager
  - Work
  - WorkListener
  - WorkEvent
  - WorkException, WorkRejectedException and WorkCompletedException
- The Application Server can either accept or reject the submitted work
  - If work is accepted, the Application Server dispatches threads to execute the Work instance

Work Progress



Section

Summary

Summary

- JCA 1.5 now supports inbound communication with transaction information from an Enterprise Information Systems to an Application Server via a Resource Adapter
- Lifecycle Management provides a mechanism for an Application Server to manage the lifecycle of a Resource Adapter instance
- Work Management support allows a Resource Adapter instance can submit work to the Application Server

Trademarks, Copyrights, and Disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	CICS	IMS	MQSeries	Tivoli
IBM(logo)	Cloudscape	Informix	OS/390	WebSphere
e(logo)business	DB2	iSeries	OS/400	xSeries
AIX	DB2 Universal Database	Lotus	pSeries	zSeries

Java and all Java-based trademarks are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are registered trademarks of Microsoft Corporation in the United States, other countries, or both.

Intel, ActionMedia, LANDesk, MMX, Pentium and ProShare are trademarks of Intel Corporation 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 of Linus Torvalds.

Other company, product and service names may be trademarks or service marks of others.

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements and/or changes in the product(s) and/or program(s) described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (e.g., IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.