IBM Software Group

An Introduction to the J2EE Connector Architecture

David Currie, IT Specialist
EMEA Software Lab Services

Agenda

- **Introduction**
- **Contracts**
- **Development**
- **Runtime**
- **Future direction**

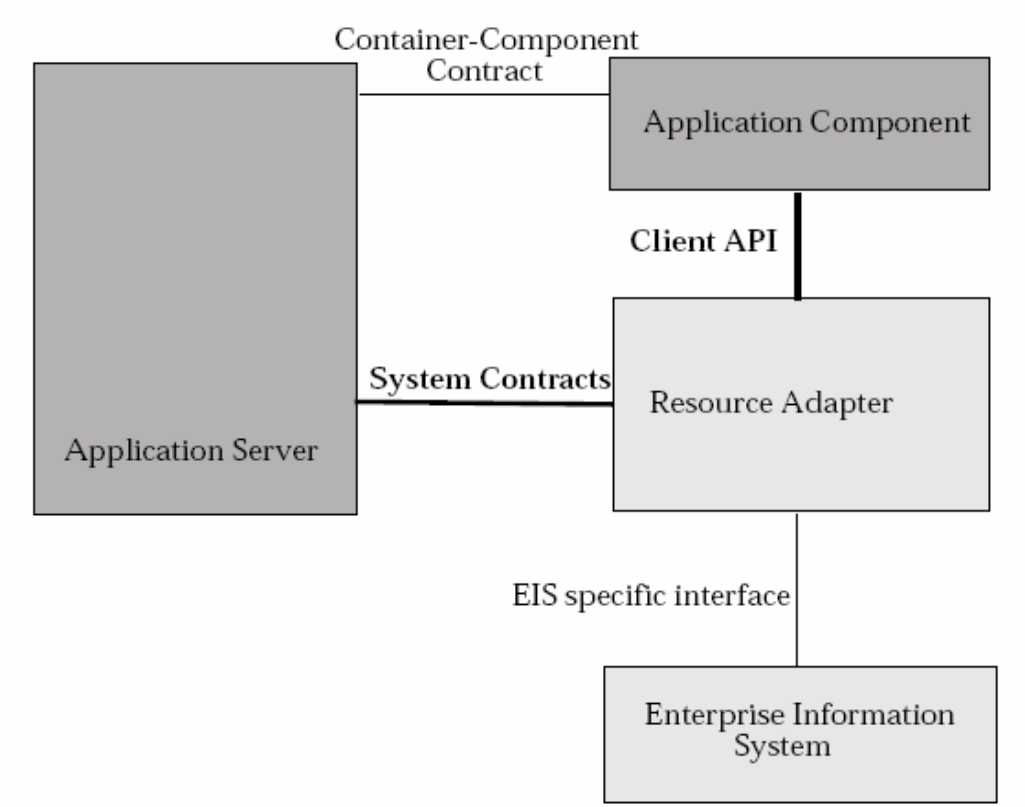
Agenda

- **Introduction**
- Contracts
- Development
- Runtime
- Future direction

Introduction

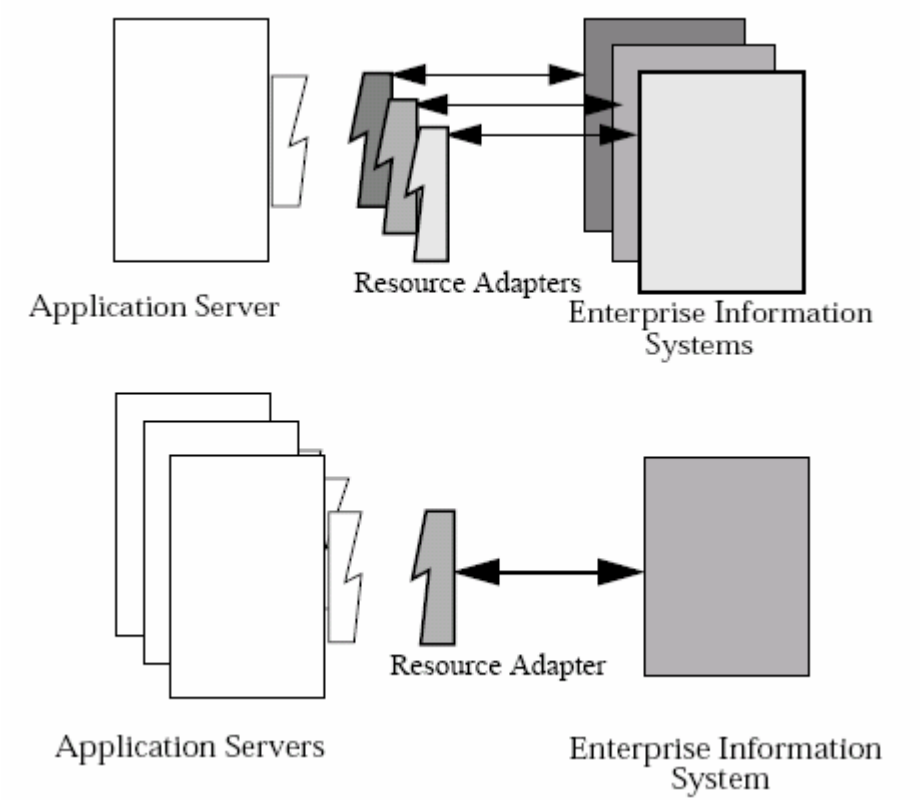
- **Defines standard contracts permitting bi-directional communication between enterprise applications and Enterprise Information Systems (EIS) e.g. ERP, TP, DB**
- **Defines SPIs between EIS and application server**
- **Defines optional API between application and EIS**

Architecture Overview



Application Server – EIS Integration

- **m x n problem becomes an m + n problem**



History

- **1998 – IBM Common Connector Framework in VAJ 2.0**
- **2001 – JCA 1.0 part of J2EE 1.3 and supported by WAS 5**
  - Packaging
  - Lifecycle management
  - Connection management
  - Security
  - Transaction management
- **2003 – JCA 1.5 part of J2EE 1.4 and supported by WAS 6**
  - Work management
  - Transaction inflow
  - Message inflow

Agenda

- Introduction
- **Contracts**
- Development
- Runtime
- Future direction

Packaging

- **Packaged in ResourceAdapter Archive (RAR) file**
- **JAR file with a .rar suffix**
- **Example contents:**
  - META-INF/ra.xml
  - howto.html
  - images/icon.jpg
  - ra.jar
  - cci.jar
  - win.dll
  - solaris.so

Lifecycle Management

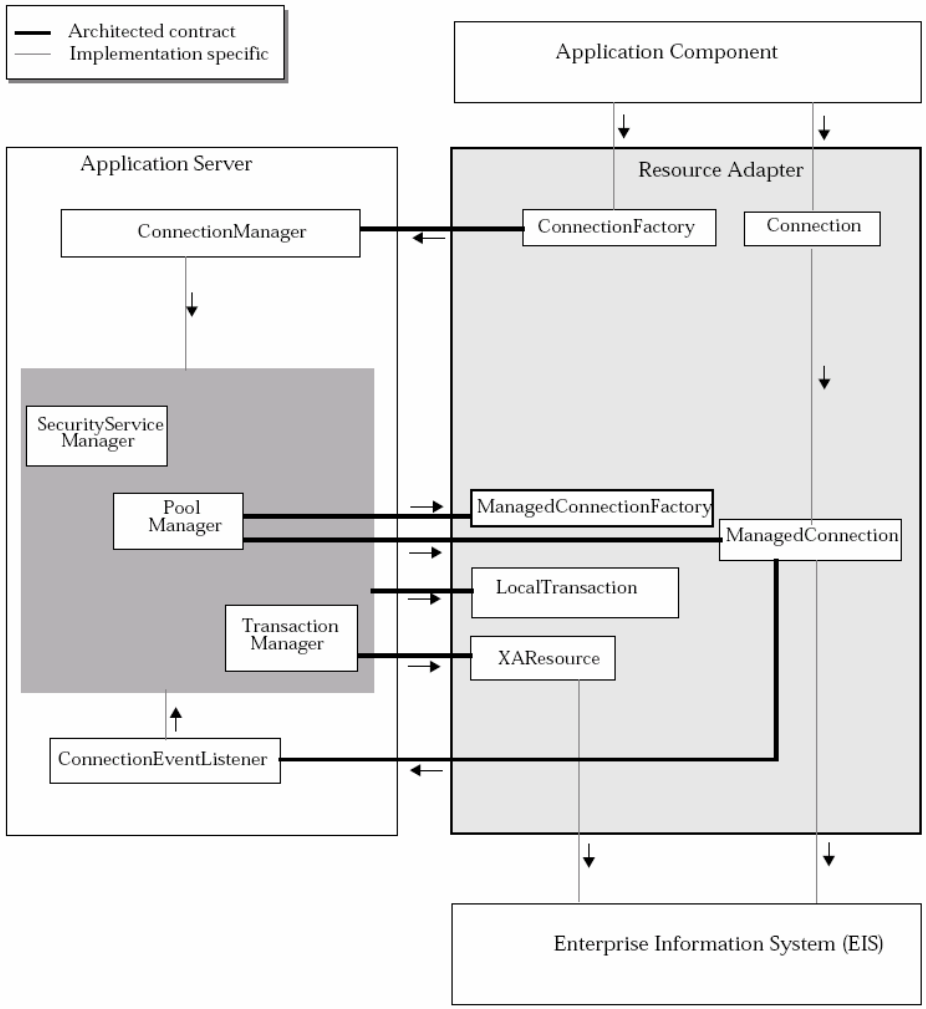
- **New in JCA 1.5**
- **Resource adapter implements *ResourceAdapter* interface**
- **Provides bootstrap mechanism during resource adapter deployment or at application server startup**
- ***BootstrapContext* implemented by application server provides access to timer, work management and transaction inflow support**
- **Notifies resource adapter during undeployment or at application server shutdown**

```
public interface ResourceAdapter {
  void start(BootstrapContext);
  void stop();
  ...
}
```

Connection Management

- **Application uses connection factory to access connection instance**
- **Connection then used to access EIS instance**
- **Resource adapter obtains connections via connection manager**
- **Application server manages connections via managed connection factory and managed connection interfaces**

Connection Management Architecture

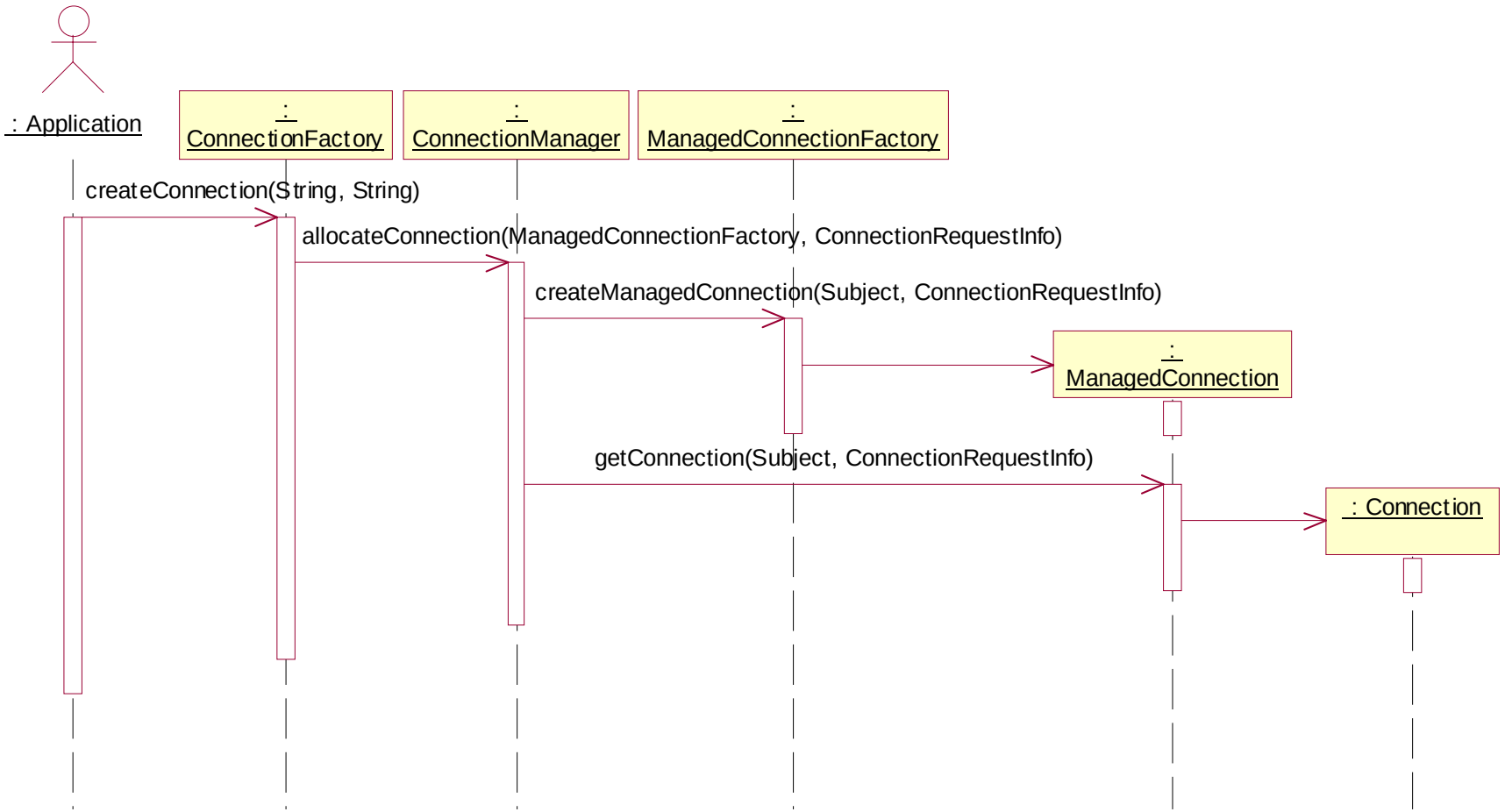


Obtaining a Connection Factory

- **ra.xml contains name of implementation class for managed connection factories along with names, types and default values for properties for the connection factory**
- **Application server system management used to configure properties**
- **During lookup, application server uses reflection to create managed connection factory and set properties**
- **Application server calls *createConnectionFactory*, passing in connection manager and returns result to application**

```
public interface ManagedConnectionFactory {
  …
  Object createConnectionFactory(ConnectionManager connectionManager);
  …
}
```

Creating a Connection



Security Credentials

- **For a res-auth of *Container:* connection manager passes *Subject* on create managed connection call containing *PasswordCredential* or *GSSCredential* depending on support declared in ra.xml**
- **For a res-auth of *Application:* connection factory may have placed application credentials in *ConnectionRequestInfo* otherwise default values set on managed connection factory**

```
public interface ManagedConnectionFactory {
  …
  ManagedConnection createManagedConnection(Subject subject,
    ConnectionRequestInfo cri);
  …
}
```

Connection Sharing

- **res-sharing-scope set to *Shareable***
- **On connection creation, connection manager may detect it is in a global transaction and that the request parameters are the same as those for a managed connection already in use in that transaction**
- **Connection manager simply calls *getConnection* a second time on the managed connection**
- **If the application already has a connection handle and is just crossing into a transaction where a managed connection already exists, connection handle can be re-associated with that managed connection**

```
public interface ManagedConnection {
  …
  void associateConnection(Object connection) ;
  …
}
```

Lazy Association

- **New in JCA 1.5**
- **Default behaviour is that a *Shareable* connection is re-associated every time it crosses a method or transaction boundary**
- **If managed connection implements *DissociatableManagedConnection* interface, connection manager dissociates all connection handles at end of method/transaction**
- **Connection handles may re-associate themselves as and when required using the *associateConnection* method on the *LazyAssociatableConnectionManager* interface**

```
public interface DissociatableManagedConnection {
  void dissociateConnections();
}
```

Connection Pooling

- **Connection manager may keep a pool of free managed connections**
- **On call to *allocateConnection* connection manager passes possible set of managed connections to managed connection factory**
- **Managed connection factory may select an appropriate managed connection or reject them all**
- **Managed connection has *cleanup* and *destroy* methods to manage its lifecycle**

```
public interface ManagedConnectionFactory {
  …
  ManagedConnection matchManagedConnections(
    Set connections, Subject subject, ConnectionRequestInfo cri);
  …
}
```

Connection Event Notification

- **Connection manager registers with managed connection to receive connection events**
- ***connectionClosed* used to return managed connections to free pool**
- ***connectionErrorOccurred* used to *destroy* broken connections**
- **Local transaction events used to police transaction interleaving**

```
public interface ConnectionEventListener {
  void connectionClosed(ConnectionEvent event);
  void connectionErrorOccurred(ConnectionEvent event);
  void localTransactionStarted(ConnectionEvent event);
  void localTransactionCommitted(ConnectionEvent event);
  void localTransactionRolledback(ConnectionEvent event);
}
```

Error Logging and Tracing

- **Managed connection factory and managed connection passed *PrintWriter* by application server which they may use for logging and tracing to the application server's log file**

```
public interface ManagedConnectionFactory {
  …
  public void setLogWriter(PrintWriter out);
  public PrintWriter getLogWriter();
  …
}
```

```
public interface ManagedConnection {
  …
  public void setLogWriter(PrintWriter out);
  public PrintWriter getLogWriter();
  …
}
```

Transaction Management

- **Resource adapter declares transactional capability in ra.xml of *XATransaction*, *LocalTransaction* or *NoTransaction***
- **Managed connection provides accessors for *XAResource* and *LocalTransaction***

```
public interface ManagedConnection {
  …
  XAResource getXAResource();
  LocalTransaction getLocalTransaction();
  …
}
```

Global Transaction Management

- **Connection manager enlists *XAResource* with transaction manager**
- **Default behaviour is to enlist when connection is created or transaction is begun**
- **If managed connection implements *LazyEnlistableManagedConnection* connection manager may defer enlistment until managed connection signals that it is required using *lazyEnlist* method**
- **During transaction recovery, *XAResource* is also obtained via managed connection**

```
public interface LazyEnlistableConnectionManager {
  void lazyEnlist(ManagedConnection mc);
}
```

Local Transaction Management

- ***LocalTransaction*** **interface used by WAS to support Local Transaction Containment Scope**
- **For resolution-control of *Application*: if *connectionStarted* event received but no corresponding completion event, *LocalTransaction* used to drive appropriate unresolved-action (commit or rollback)**
- **For resolution-control of *Container*: *begin* called on *LocalTransaction* at start of scope and *commit* or *rollback* at end**

```
public interface LocalTransaction {
  void begin();
  void commit();
  void rollback();
}
```

Work Management

- **Resource adapter may obtain *WorkManager* from *BootstrapContext***
- ***Work* instances may be submitted for processing**
  - *doWork* blocks for work to complete
  - *startWork* blocks until work begins
  - *scheduleWork* does not block
- ***Work* instances implement *Runnable* and have a *release* method for the *WorkManager* to indicate that the thread should complete**
- **Resource adapter may specify timeout for start of work and pass *WorkListener* to be informed of progress of the work**
- ***Timer* may be obtained from *BootstrapContext* for performing periodic work**

Transaction Inflow

- **Permits a resource adapter to import a transaction into the application server using *ExecutionContext* passed when work submitted**
- **Resource adapter responsible for transmitting transaction completion and crash recovery calls to the application server using *XATerminator* obtained from *BootstrapContext***

```
public interface XATerminator {
  void commit(Xid xid, boolean onePhase);
  void forget(Xid xid);
  int prepare(Xid xid);
  Xid[] recover(int flag);
  void rollback(Xid xid);
}
```

Message Inflow

- **Resource adapter declares the MDB interfaces that it supports in ra.xml along with name of the *ActivationSpec* implementation class**
- **Introspection of *ActivationSpec* JavaBean used to determine properties required by resource adapter for message inflow**
- ***ActivationSpec* properties configured in system management for each deployed MDB**
- **On application startup application server passes configured *ActivationSpec* and *MessageEndpointFactory* to resource adapter**

```
public interface ResourceAdapter {
  ...
  void endpointActivation(MessageEndpointFactory mef, ActivationSpec spec);
  void endpointDeactivation(MessageEndpointFactory mef, ActivationSpec spec);
  XAResource[] getXAResources(ActivationSpec[] specs);
  ...
}
```

Message Delivery

- **On message receipt, resource adapter creates *MessageEndpoint* optionally passing in an *XAResource***
- **Resource adapter casts endpoint to appropriate MDB interface and delivers message**
- **Resource adapter releases endpoint again**

```
public interface MessageEndpointFactory {
  MessageEndpoint createEndpoint(XAResource xaResource);
  boolean isDeliveryTransacted(Method method);
}

public interface MessageEndpoint {
  void beforeDelivery(Method method);
  void afterDelivery();
  void release();
}
```

Administered Objects

- **ra.xml may define interfaces, implementation classes and properties (names, types and default values) for administered objects**
- **Application server system management used to configure instances of administered objects which may then be bound into JNDI**

Common Client Interface (CCI)

- **Specifies an optional set of client APIs**
- **Solves the *m x n* problem for tooling vendors**
- **Interfaces for:**
  - *ConnectionFactory, Connection, ConnectionSpec, LocalTransaction*
  - *Interaction, InteractionSpec*
  - *RecordFactory, Record, MappedRecord, IndexedRecord, Streamable, ResultSet*
  - *ConnectionMetaData, ResourceAdapterMetaData, ResultSetInfo*
  - *MessageListener*

Agenda

- Introduction
- Contracts
- **Development**
- Runtime
- Future direction

Reasons to use JCA (or not)

- **Open sockets ✘**
- **Accessing file system ✘**
- **Multi-threading ?**
- **Packaging ?**
- **Embedded EIS ?**
- **Connection pooling and sharing ✓**
- **Transaction support ✓**
- **Obtaining security credentials ✓**
- **Work initiated other than by HTTP/JMS/RMI ✓**
- **Administrative access to resources ✓**

Initial design decisions

- **Inbound and/or outbound resource adapter**
- **CCI or custom API**
- **Transaction support**
  - *NoTransaction*
  - *LocalTransaction*
  - *XATransaction*
- **Authentication mechanism**
  - *BasicPassword*
  - *KerbV5*
- **Reauthentication**
- **Administered objects**
- **Properties**

Outbound resource adapter development

- **Implement default *ConnectionManager* (*CM*)**
  - *allocateConnection* calls *createManagedConnection* followed by *getConnection*
- **Start to implement *ManagedConnectionFactory* (*MCF*)**
  - accessors for properties
  - implement *ResourceAdapterAssociation* if required
  - *equals/hashCode* important
  - *createConnectionFactory* passes *CM* and *MCF* to connection factory constructor
- **Implement *ConnectionRequestInfo* (*CRI*)**
  - fields for connection specific properties
  - *equals/hashCode* important

Outbound resource adapter development cont'd

- **Implement connection factory**
  - *getConnection* or equivalent constructs *CRI* from parameters and calls *allocateConnection* on *CM*
- **Complete *MCF* implementation**
  - *createManagedConnection* constructs *ManagedConnection* passing *Subject* and *CRI*
  - *allocateManagedConnection* iterates over connection set and returns first *ManagedConnection* that matches given *Subject* and *CRI*
  - Implement *ValidatingManagedConnectionFactory* if required

Outbound resource adapter development cont'd

- **Implement *ManagedConnection***
  - constructor extracts credentials from *Subject*, *CRI* or *MCF* and creates physical connection using properties from *CRI* and *MCF*
  - *getConnection* should check credentials, reauthenticate if necessary, return a new connection passing *ManagedConnection* to constructor and add the connection to the associated set
  - *associateConnection* should set this *ManagedConnection* on the connection and add the connection to the associated set
  - *cleanup* should remove all associated connections
  - *destroy* should close the physical connection
- **Implement *ManagedConnectionMetaData***

Outbound resource adapter development cont'd

- **Implement connection**
  - connection is associated with *ManagedConnection* on construction
  - methods delegate to physical connection via *ManagedConnection*
  - when closed, associated *ManagedConnection* should send *connectionClosed* to *ConnectionEventListeners*
  - if *DissociatableMangedConnection* implemented then connection may be dissociated from *ManagedConnection* in which case *associateConnection* called on *LazyAssociatableConnectionManager* when needed

Outbound resource adapter development cont'd

- **If required, implement *LocalTransaction***
  - methods should send *LocalTransactionStarted*, *Committed* and *Rolledback* events
- **If required, implement *XAResource***
  - requires good understanding of JTA including transaction recovery
  - If implementing *LazyEnlistableManagedConnection* interface, prior to performing any work, if not yet enlisted and *CM* implements *LazyEnlistableConnectionManager*, call *lazyEnlist*

Inbound resource adapter development

- **Implement *ActivationSpec* (*AS*)**
  - accessors for properties (detected via introspection) plus validation
  - required properties defined in deployment descriptor
- **Implement *ResourceAdapter* (*RA*)**
  - accessors for properties
  - start/stop for initialization and cleanup
  - *endpointActivation* constructs *EndpointActivation* class passing *MessageEndpointFactory* (*MEF*), *AS* and *RA*, and adds to map keyed off *MEF*
  - *endpointDeactivation* locates *EndpointActivation* from map and deactivates it
  - if transactional, implement *getXAResources*

Inbound resource adapter development cont'd

- **Implement *EndpointActivation***
  - constructor typically creates *Timer* from *BootstrapContext* and schedules *EndpointTask* to poll for messages
  - deactivate cancels timer
- **Implement *EndpointTask***
  - when run finds messages, obtain *WorkManager* from *BootstrapContext* and start *EndpointWork* for asynchronous processing
- **Implement *EndpointWork***
  - run obtains *MessageEndpoint* from *MEF*, casts to required interface, invokes method and then releases endpoint
  - if transactional, pass *XAResource* when creating *MessageEndpoint*
  - use *before/afterDelivery* if processing requires MDB classloader or transaction
  - if batch processing, release method should halt batch

Transaction inflow

- **Set *Xid* and transaction timeout on *ExecutionContext* passed when starting *ExecutionWork***
- **When resource adapter receives notification of transaction flow, obtain *XATerminator* from *BootstrapContext* and invoke appropriate method**

Administered object development

- **Interface and implementation class defined in deployment descriptor**
- **Deployment descriptor names properties along with types and default values**
- **Beware, WAS will also perform introspection**

IBM WebSphere Adapter Toolkit

- **Plugin for Rational Application Developer or WebSphere Integration Developer**
- **J2C Resource Adapter Project Wizard**
- **Resource Adapter Deployment Descriptor Editor**
- **Framework classes for adapters using SDO and SDO running in WebSphere Process Server**

Agenda

- Introduction
- Contracts
- Development
- **Runtime**
- Future direction

Installing resource adapters

- **Install RAR file via administration console or installResourceAdapter command**
  - RA is always installed at node level
  - Scope defines visibility of resources
  - Options for classpath, native path and thread pool
- **Install in to client container using clientRAR**
- **Embed RAR file in EAR**
  - Specify target server/cluster when mapping modules to servers
  - Appears in administration console under application

Defining connection factories

- **Create connection factories using administration console or *createJ2CConnectionFactory* command**
- **For security, component-managed authentication alias defined on resource reference should be used with XA recovery alias if necessary**
- **Configure connection pool properties (in particular, default maximum is 10)**
- **Direct lookups are deprecated in WAS 6**
- **For client container, use *clientConfig* to define connection factories**

Defining administered objects

- **Create administered objects via administration console or *createJ2CAdminObject* command**
- **For client container, use *clientConfig* to define administered objects**

Defining activation specifications

- **Create activation specification via administration console or *createJ2CActivationSpec* command**
- ***ActivationSpec* is bound in to JNDI and referenced by application**
- ***UserName/Password* should be specified as authentication alias in MDB bindings with separate XA recovery alias if necessary**
- **Special processing for “destination” property to enable use of message-destination-link**

Runtime administration

- **“Manage state” option on connection factory in administration console provides ability to pause resource**
- ***J2CResourceAdapter* MBean also enables entire resource adapter to be paused and stopped (including inbound resources)**
- **PMI statistics for every conceivable event**

Agenda

- Introduction
- Contracts
- Development
- Runtime
- **Future direction**

Current and future direction

- **JCA 1.5 explicitly states pluggability of JMS providers as a goal**
- **Programming model but no deployment model for client environment**
- **Improvements to configuration model**
- **Security inflow**
- **Ability to embed the entire EIS, not just a client, within the application server**


Questions

Summary

- **Defines standard contracts permitting bi-directional communication between enterprise applications and Enterprise Information Systems (EIS) e.g. ERP, TP, DB**
- **Defines optional API between application and EIS**
- **Defines SPIs between EIS and application server**
- **WebSphere Application Server v6 fully supports JCA 1.5**

References

- **Specification**
  - http://java.sun.com/j2ee/connector/download.html
- **Introduction to J2EE Connector Architecture**
  - http://www.ibm.com/developerworks/java/edu/j-dw-javajca-i.html
- **J2EE Connector Architecture 1.5**
  - http://www.ibm.com/developerworks/java/library/j-jca3/
  - http://java.sun.com/developer/EJTechTips/2004/tt1123.html#1
- **IBM WebSphere Adapter Toolkit**
  - http://www.ibm.com/developerworks/websphere/downloads/wat/


Thank
You