



java.com.sun/javaone

JavaOne™

Java™ EE Connector architecture 1.6 overview

Binod PG and Sivakumar Thyagarajan
Sun Microsystems

BOF 5634, Hall E 133, 6th May 2008



Learn what is planned for the next version of the
Java EE Connectors Architecture technology

GOAL

Caveat

- These are **proposed** feature discussions
 - Work in progress
 - Features/APIs subject to change
- Please wait for the early draft

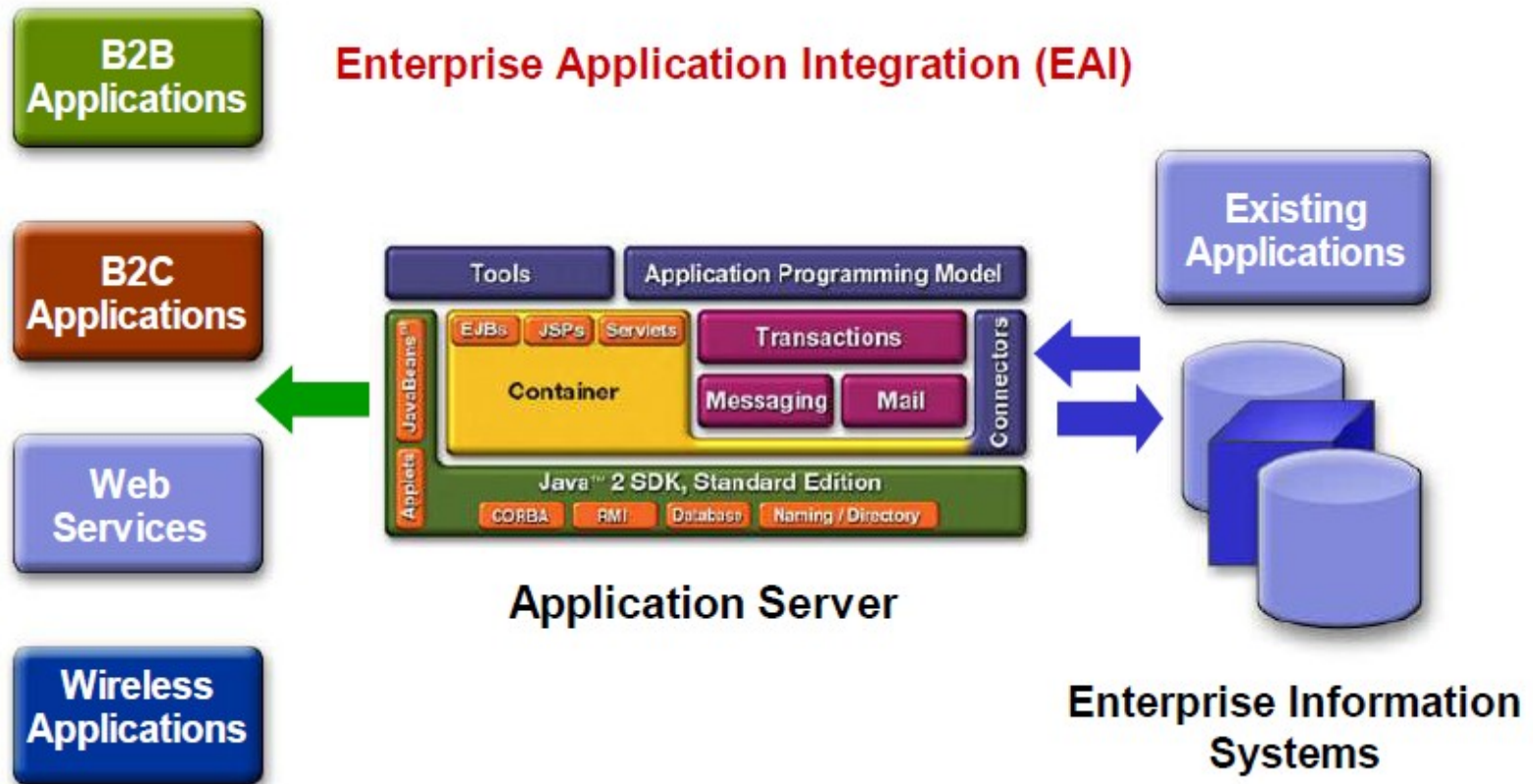
Agenda

- Introduction
- Connectors 1.5 capabilities
- Major themes for Connectors 1.6
 - Generic Inflow context
 - Security Inflow contract
 - Ease of development and use
 - Misc. improvements to the spec
- Summary
- Q & A

Agenda

- **Introduction**
- **Connectors 1.5 capabilities**
- **Major themes for Connectors 1.6**
 - Generic Inflow context
 - Security Inflow contract
 - Ease of development and use
 - Misc. improvements to the spec
- **Summary**
- **Q & A**

Java EE Connector architecture overview



Connectors 1.5 capabilities

- Lifecycle Management
- Outbound communication
 - Mechanism for Java EE components to connect to external systems
 - Connection Management
 - Lifecycle management/Pooling/Sharing
 - Security contract
 - Exporting transaction and security context to EIS
- Inbound messaging
 - Mechanism for EIS to call message endpoints (MDBs)
 - Import transaction context from EIS
- Work Management

Agenda

- Introduction
- Connectors 1.5 capabilities
- **Major themes for Connectors 1.6**
 - Generic Inflow context
 - Security Inflow contract
 - Ease of development and use
 - Misc. improvements to the spec
- Summary
- Q & A

Connectors 1.6

- Work in progress by the expert group of JSR 322
- Part of Java EE 6
- Started work in Jan 2008
- Expect early draft to be available in June 2008
- Major themes
 - Generic Inflow context/Security Inflow contract
 - Ease of Development
 - Misc. improvements to the specification

Agenda

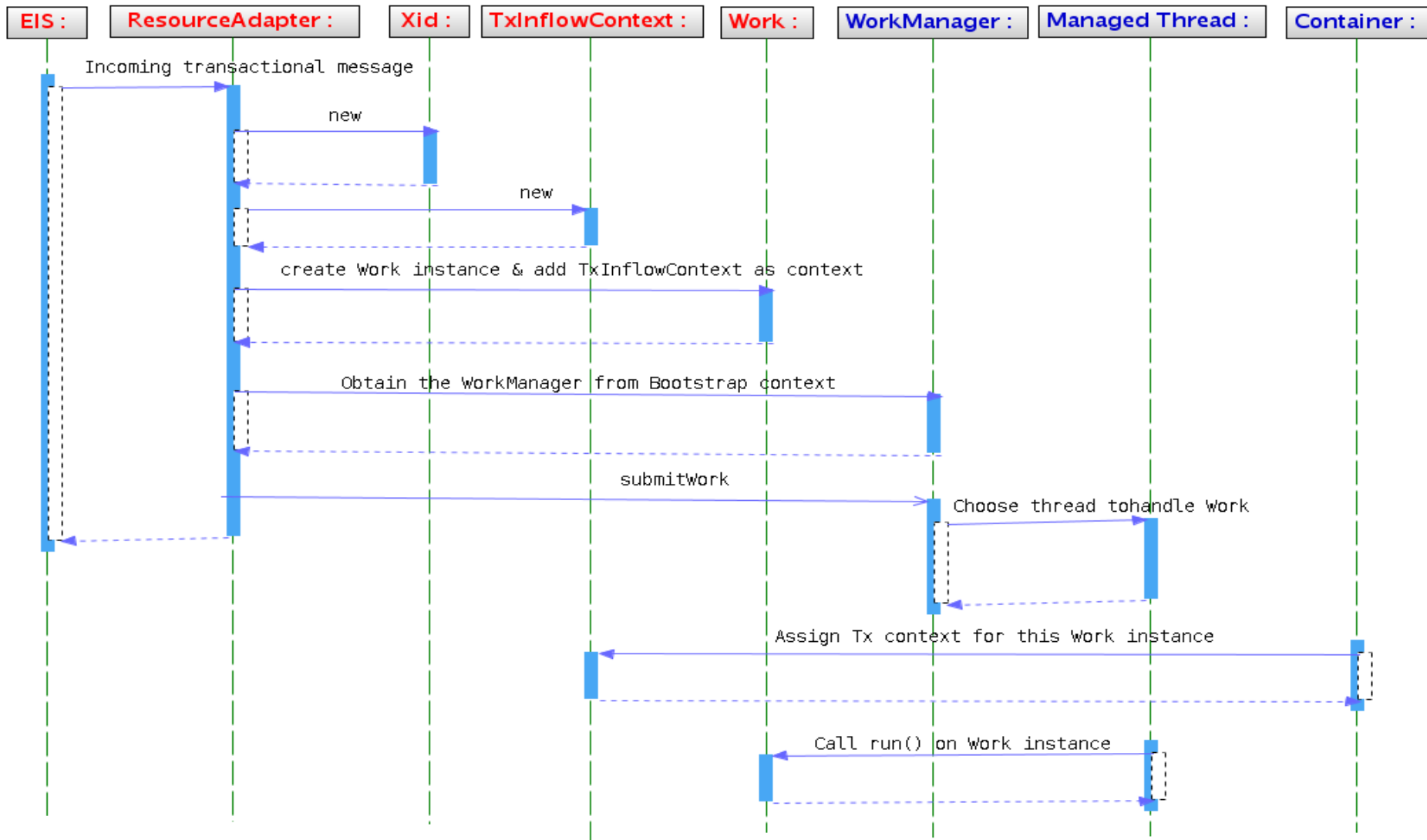
- Introduction
- Connectors 1.5 capabilities
- **Major themes for Connectors 1.6**
 - Generic Inflow context
 - Security Inflow contract
 - Ease of development and use
 - Misc. improvements to the spec
- Summary
- Q & A

Generic Inflow context

➤ Rationale

- Propagate other contextual info from EIS
 - Security, Conversational context, Availability/QoS etc
 - Existing Transaction Inflow contract tightly coupled with Connector WorkManager contracts
-
- Allow a resource adapter(RA) to flow in an “extensible” context to an application server
 - Enables an application server to support new message inflow and delivery schemes
 - Provides a richer contextual *Work* execution environment to the RA

Generic Inflow context – Sequence diagram



Generic Inflow context model

```
public interface InflowContextProvider {  
    InflowContexts getInflowContexts();  
}
```

```
public interface InflowContext {...}
```

```
public class TransactionInflowContext  
    extends ExecutionContext  
    implements InflowContext {...}
```

Generic Inflow context – illustrative example

```
//Resource adapter
public class MyResourceAdapter
    implements ResourceAdapter {
    ...
    {
        WorkManager workManager =
myRA.bootstrapCtx.getWorkManager();
        workManager.submitWork(new MyWork());
    }
}
```

Generic Inflow context – illustrative example

```
//Work implementation
class MyWork implements Work,
                        InflowContextProvider {
    ...
    InflowContexts getInflowContexts() {
        TransactionInflowContext txIn = new
            TransactionInflowContext();
        txIn.setXid(xid);
        return new InflowContexts(txIn);
    }
    void run() {
        //deliver message to MessageEndpoint;
    }
}
```

Generic Inflow context model

- Compatible with the Connectors 1.5 Work submission and context assignment model
- Independent of the Connectors Work Management contract
 - Enables the RA to use these in other asynchronous task executors
- Standardize Transaction Inflow and Security Inflow contexts
- *InflowContextLifecycleListener*
- Ongoing discussion on
 - Reconciling app-server's support and RA's requirement of an *InflowContext* type
 - Optional inflow context support

Agenda

- Introduction
- Connectors 1.5 capabilities
- **Major themes for Connectors 1.6**
 - Generic Inflow context
 - **Security Inflow contract**
 - Ease of development and use.
 - Misc. improvements to the spec
- Summary
- Q & A

Security Inflow contract

➤ Rationale

- Security inflow context absent in Connectors 1.5
- RAs not being able to deliver application level security while
 - executing a task in a *Work* instance
 - delivering a message to a *MessageEndpoint*

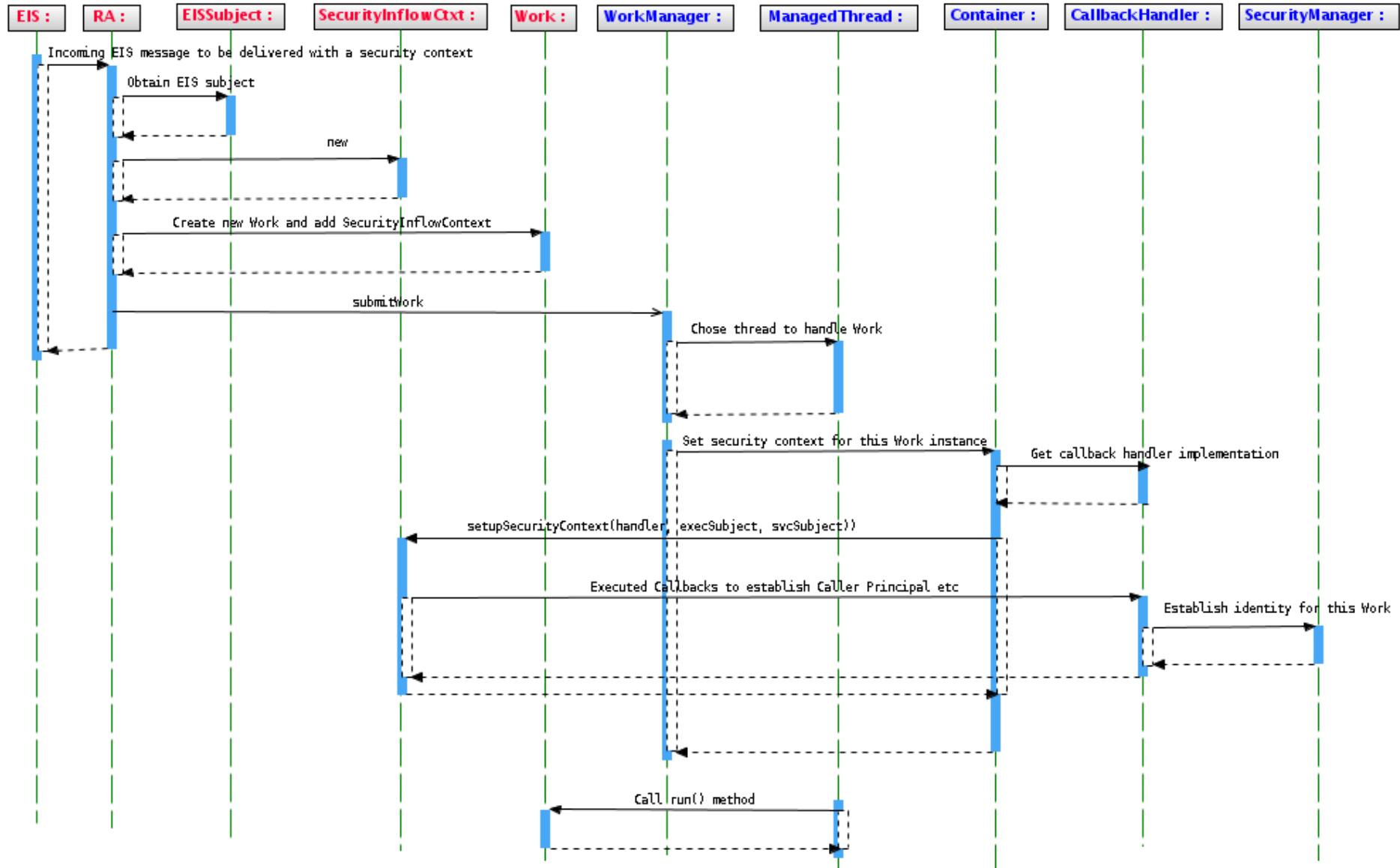
➤ Goal

- Enable an end-to-end security model for Java EE application-EIS integration
- Support the execution of a *Work* instance in the context of an established identity.
- Support the propagation of *Principal* information from an EIS to a *MessageEndpoint* during Message Inflow

Security Inflow contract

- Leverage the work done in JSR 196: Java Authentication Service Provider Interface for Containers
- Define a standard *SecurityInflowContext*
 - Appserver establishes security context for the MDB/Work during context assignment of *SecurityInflowContext*
- RA identifies security context to be used and uses the various JSR-196 *Callbacks* to establish caller identity

Security Inflow contract – Sequence diagram



Security Inflow contract API

```
public abstract class SecurityInflowContext
    implements InflowContext {
    ...
    public abstract void
        setupSecurityContext(
            CallbackHandler handler,
            Subject executionSubject,
            Subject serviceSubject);
}
```

Security Inflow contract

- Handling EIS and application server security domain differences
 - Case 1: No translation required
 - Appserver uses the initiating principal as the security context
 - Case 2: Translation required from the EIS to the appserver security domain (not discussed in this session)

- CallbackHandler allows
 - CallerPrincipalCallback
 - GroupPrincipalCallback
 - ... and others

Security Inflow contract – illustrative example

```

//Resource adapter
public class XMPPResourceAdapterImpl
    implements ResourceAdapter {
    ...
    {
        WorkManager workManager =
myRA.bootstrapCtx.getWorkManager();
        workManager.submitWork(new
            XMPPMessageDeliveryWork());
        ...
    }
}

```

Security Inflow contract – illustrative example

```

//Work implementation
public class XMPPMessageDeliveryWork
    implements Work, InflowContextProvider {
    ...
    InflowContexts getInflowContexts() {
        SecurityInflowContext scIn = new
            XMPPSecurityInflowContext();
        return new InflowContexts(scIn);
    }
    void run() {
        //deliver the message from the
        //XMPP user to the MessageEndpoint
    }
}

```

Security Inflow contract – illustrative example

```
//Case #1: No mapping required
class XMPPSecurityInflowContext
    extends SecurityInflowContext {
    @Override
    public void setupSecurityContext(CallbackHandler
        h, Subject execSubj, Subject svcSubj) {
        //get username from EIS
        String userName = ...;
        CallerPrincipalCallback cpCallback = new
            CallerPrincipalCallback (execSubj, userName);
        h.handle(new Callback[] {cpCallback});
    }
}
```

Agenda

- Introduction
- Connectors 1.5 capabilities
- **Major themes for Connectors 1.6**
 - Generic Inflow context
 - Security Inflow contract
 - **Ease of development and use**
 - Misc. improvements to the spec
- Summary
- Q & A

Ease of development

- Goal : "Configuration by exception" and ease of use(EoU)/development(EoD) of RAs
- EoU: Already available via @Resource and activationConfig in @MessageDriven
- Only focussed EoD improvements in this version
- For example, ideas include:
 - New metadata annotations
 - Candidates: ResourceAdapter, ActivationSpec, ManagedConnectionFactory, AdminObject
 - Specify defaults if possible
 - Remove required elements in ra.xml
 - Allow sparse ra.xml
 - Dependency injection
 - Candidates: ResourceAdapterAssociation, BootstrapContext

Agenda

- Introduction
- Connectors 1.5 capabilities
- **Major themes for Connectors 1.6**
 - Generic Inflow context
 - Security Inflow contract
 - Ease of development and use
 - **Misc. improvements to the spec**
- Summary
- Q & A

Misc. improvements to the spec

- **Quality of service improvements**
 - Connection validation, handling connection failures
- **Configuration property processing**
 - Alignment between inbound and outbound, validation, dynamic reconfiguration support
- **TransactionSynchronizationRegistry availability**
- **Dynamic transactional support determination**
- **Standalone Connectors container semantics**
- **... and others**

Agenda

- Introduction
- Connectors 1.5 capabilities
- Major themes for Connectors 1.6
 - Generic Inflow context
 - Security Inflow contract
 - Ease of development and use.
 - Misc. improvements to the spec
- **Summary**
- Q & A

Summary

- **Connectors 1.6 specification (JSR-322)**
- **Part of Java EE 6**
- **Goals**
 - **Generic/Security Inflow**
 - **Misc improvements to the spec**
 - **Ease of development**

For More Information

➤ JSR 322 details

- home <http://jcp.org/en/jsr/detail?id=322>
- Send comments to jsr-322-comments@jcp.org

➤ Our blogs

- Binod <http://weblogs.java.net/blog/binod>
- Siva <http://blogs.sun.com/sivakumart>

➤ Reference implementation – Project GlassFish

- <http://GlassFish.dev.java.net>

THANK YOU



Binod PG and Sivakumar Thyagarajan
Sun Microsystems

BOF 5634, Hall E 133, 6th May 2008



Security Inflow contract – illustrative example

```
//Case #2: Mapping required
class XMPPSecurityInflowContext extends SecurityInflowContext {
    public void setupSecurityContext(CallbackHandler h ...) {
        //Map principal from EIS
        String eisPrincipal = ...;
        Map hm = new HashMap(); hm.put(eisPrincipal, null);
        PrincipalMappingCallback pmCallback =
            new PrincipalMappingCallback(false, hm);
        handler.handle(new Callback[] { pmCallback });
        String mappedPrin = hm.get(eisPrincipal);

        //Establish the mapped Principal as the caller principal
        CallerPrincipalCallback cpCallback =
            new CallerPrincipalCallback(execSubj, mappedPrin);
        handler.handle(new Callback[] { cpCallback });

        ...
    }
}
```