Communication Framework

G.Riviere
Guillaume.Riviere@inrialpes.fr
Contents

- RMI Overview
- Multi-Protocol with RMI-IIOP
- Multi-Name Services with JNDI
- RMI Context Propagation
- Conclusion
Contents

† RMI Overview
  ➤ RMI JRMP, the Java RPC
  ➤ RMI IIOP, the Java RPC on the top of CORBA
† Multi-Protocol with RMI-IIOP
† Multi-Name Services with JNDI
† RMI Context Propagation
† Conclusion
RMI JRMP

† RPC with Java serialization
† JRMP remote object generation
  ✤ Implements Remote Interface
  ✤ RMI Classes generation
    • V1.1 one Stub and one Skeleton
    • V1.2 only one Stub
† JRMP remote object server instantiation
  ✤ Exported by UnicastRemoteObject
    • Extends UnicastRemoteObject, export in constructor
    • UnicastRemoteObject static method
  ✤ Binded in a Name Service Java: Registry
† JRMP remote object client access
  ✤ Lookup in a Name Service
  ✤ Stub visible in the Classpath
  ✤ PortableRemoteObject.narrow (J2EE)
JRMP overview

Server

JNDI

SPI InitialContext

API InitialContext

Registry bind

Registry

InitialContext bind

JRMP export

lookup

Remote Method call

JRMP Client
JRMP 1.1 method call

- **Remote Object**
  - **Skeleton Impl**
  - **Delegation**
  - **Server Ref**

- **Client**
  - **Stub Impl**
  - **Delegation**
  - **Remote Ref**

- **Stub Methods**
  - **Skeleton with Object Reference**
  - **Remote Ref with Server Ref port address**
  - **Server Ref with port address**

- **Java Serialization**
JRMP 1.2 method call

Remote Object

Reflection

Server Ref

Java Serialization

Remote Ref

Stub

Delegation

Stub Impl

Client
Contents

† RMI Overview
   ✧ RMI JRMP, the Java RPC
   ✧ RMI IIOP, the Java RPC on the top of CORBA
† Multi-Protocol with RMI-IIOP
† Multi-Name Services with JNDI
† RMI Context Propagation
† Conclusion
† RPC with IIOP (GIOP on TCP/IP) serialization
† IIOP remote object generation
   ✶ Implements Remote Interface
   ✶ RMI Classes generation
      • one Stub and one Tie (CORBA Skeleton)
† IIOP remote object server instantiation
   ✶ Exported by PortableRemoteObject
      • Extends PortableRemoteObject, export in constructor
      • PortableRemoteObject static method
      • PortableRemoteObject Delegation Mechanism
   ✶ Binded in a Name Service CORBA: CosNaming
† IIOP remote object client access
   ✶ Lookup in a Name Service
   ✶ Stub visible in the Classpath
   ✶ PortableRemoteObject.narrow (J2EE)
IIOP overview

IIOP export

JNDI

Server

Remote Method call

InitialContext bind

lookup

SPI InitialContext

CosNaming bind

API InitialContext

CosNaming

IIOP Client
IIOP method call

Remote Object -> Skeleton (Tie) -> CORBA Bus -> IIOP Serialization -> Stub

Client
JNDI

SPI InitialContext

API InitialContext

CosNaming

IIOP export

Remote Method call

IIOP Client
IIOP PortableRemoteObject

Server, Remote Object

IIOP export, narrow, toStub, connect, unexport

PortableRemoteObject

Stub

Util

Delegation

P.R.O. Delegate

Stub Delegate

Util Delegate

javax.rmi.CORBA.PortableRemoteObjectClass

javax.rmi.CORBA.StubClass

javax.rmi.CORBA.UtilClass

ORB ORBSingleton

javax.omg.CORBA.ORBClass

javax.omg.CORBA.ORBSingletonClass

www.objectweb.org
IIOP Conclusion

- We have the control of the PortableRemoteObjectDelegate
- Bind, Lookup => Stub propagation
- We use CORBA mechanism in RMI IIOP
  - CORBA ServiceContext
  - Portable Interceptor (CORBA 2.4)
Contents

RMI Overview

Multi-Protocol with RMI-IIOP

🔗 Multi protocol overview
🔗 RMI IIOP architecture for multi protocol

Multi-Name Services with JNDI

RMI Context Propagation

Conclusion
Multi Protocol

† Server Protocol Independent
   ✗ Use of RMI IIOP for root node in the protocol tree
   ✗ No impact on client conception/coding
   ✗ No impact on server conception/coding (standard RMI IIOP)

† Remote object accessible on more than one protocol at Runtime, by configuration
   ✗ RMI IIOP (David, JDK …)
   ✗ RMI JRMP
   ✗ Jeremie
   ✗ RMI over XML (later)

† Standard clients for each protocol
Multi Protocol overview

JNDI

Server

InitialContext bind

SPI InitialContext

API InitialContext

CosNaming

Registry

CosNaming bind

Registry bind

JRMP Client

IIOP Client

Remote Method call

Remote Method call

lookup

Rmi JRMP and IIOP export

CosNaming bind

Registry bind
Contents

- RMI Overview
- Multi-Protocol with RMI-IIOP
  - Multi protocol overview
  - RMI IIOP architecture for multi protocol
- Multi-Name Services with JNDI
- RMI Context Propagation
- Conclusion
One Portable Remote Object Delegate (PROD) for each protocol

- **RMI IIOP PROD**: Standard in rmi iiop
- **RMI JRMP PROD**: Mapping of the UnicastRemoteObject

A system property to set our Multi PROD

A multi-protocol configuration mechanism (for example XML file)
Contents

- RMI Overview
- Multi-Protocol with RMI-IIOP
- Multi-Name Services with JNDI
  - Problem
  - Current solution
  - Lookup, toStub problem
- RMI Context Propagation
- Conclusion
Name services problem

- JNDI
- InitialContext bind
- Server
- JRMP Client
- IIOP Client

- SPI InitialContext
- API InitialContext
- Registry bind
- CosNaming bind
- Registry
- CosNaming
Contents

RMI Overview
Multi-Protocol with RMI-IIOP
Multi-Name Services with JNDI
  Problem
  Current solution
  Lookup, toStub problem
RMI Context Propagation
Conclusion
Current solution

- MultInitialContext System
- SPI MultInitialContext Factory
  - Server system property –Djava.naming.factory.initial sets to this MultInitialContext Factory
  - Standard new InitialContext()
- This Multi InitialContext calls the InitialContext for each protocol
  - jndi api/spi for each protocol
Current solution

JNDI

SPI InitialContext

API InitialContext

Registry

CosNaming

Server

InitialContext bind

CosNaming bind

Registry bind
Current solution

JNDI API InitialContext

SPI MultiFactory

-Djava.naming.factory.initial=<MultiFactory>

MultInitialContext

New InitialContext()

bind

Server

Configuration Mechanism XML File

CosNaming bind

Registry bind

New InitialContext(IIOP (IIOP properties))

New InitialContext(JRMP (JRMP properties))

JNDI API InitialContext

SPI IIOP

API InitialContext

SPI MultiFactory

Registry

CosNaming

IIIOP I.C.

JRMP I.C.

www.objectweb.org
In the Server, with Multi-Protocol object access, if a method (like lookup) returns a stub:

- On which PROD build this Stub (toStub method) ?
- On which NameService find this Stub (lookup method) ?

We can define a default protocol, but there is another problem:

- If the default protocol is RMI IIOP
  - A RMI JRMP client calls a method that returns a Remote Object
  - This method returns a Stub for RMI IIOP and not for RMI JRMP!

So for this problem we can also make for each protocol

- An Interceptor that sets a Current Thread to the right protocol
Lookup, toStub problem

Configuration Mechanism XML File

JNDI
- SPI InitialContext
- API InitialContext

CosNaming
- Registry

ThreadLocal

Current: JRMP

Server
- getProtocol
- lookup
- setJrmpProtocol

JRMP Server Interceptor

Remote Method call

JRMP Client

www.objectweb.org
Contents

- RMI Overview
- Multi-Protocol with RMI-IIOP
- Multi-Name Services with JNDI
- RMI Context Propagation
  - CORBA Portable Interceptor
    - Portable Interceptor Overview
    - ServiceContext static final Class
  - Interceptor: A solution also for JRMP
    - Custom UnicastRemoteObject, ServerRef and ClientRef for Portable Interceptor integration in JRMP
- Conclusion
Portable Interceptor Overview

Portable Interceptor (P.I.)

- CORBA 2.4+, JDK 1.4+
- Package org.omg.PortableInterceptor
- A standard way for ServiceContext propagation

Mechanism overview

- Interface ORBInitialisation for P.I. Configuration
  - post_init and pre_init methods
  - System property (org.omg.CORBA.ORBInitiliazerClass)

- Interface ServerRequestInterceptor
  - Server Interceptor
  - receive_request, send_reply, send_exception interception

- Interface ClientRequestInterceptor
  - Client Interceptor
  - Send_request, receive_reply, receive_exception interception
Portable Interceptor Overview

- **Server Interceptor**
  - Get ServiceContext
  - ServiceContext Array

- **Client Interceptor**
  - Add ServiceContext
  - ServiceContext Array

**CORBA Bus**

**IIOP Serialization**

**Method call**
Static final class ServiceContext

- Simple Class
  - One int field: the ServiceContext id
  - One byte [ ] field: the ServiceContext

- Static final class
  - No inheritance mechanism

- This class is not marshaled by the ORB
  - the orb only sends an int and a byte [ ]
Contents

† RMI Overview
† Multi-Protocol with RMI-IIOP
† Multi-Name Services with JNDI
† RMI Context Propagation
   ☞ CORBA Portable Interceptor
      • Portable Interceptor Overview
      • ServiceContext static final Class
   ☞ Interceptor: A solution also for JRMP
      • Custom UnicastRemoteObject, ServerRef and ClientRef for Portable Interceptor integration in JRMP
† Conclusion
The Server knows, by configuration, the client and the server interceptors classes

The ServerRef sends the RemoteRef with the client interceptor

- There is no stub/skeleton modification
- Configuration only in the server side
- Client interceptors classes in the client classpath
JRMP 1.1 Portable Interceptor

Remote Object

Skeleton Impl

Delegation

Server Interceptor

Server Ref

Skeleton

UnicastRemoteObject

getServerRefClass

Java Serialization

getRemoteRefClass()

Client

Stub Impl

Delegation

Client Interceptor

Remote Ref

Client

Stub
JRMP 1.2 portable Interceptor

Remote Object

Reflection

Server Interceptor

Server Ref

Java Serialization

Remote Ref

Stub

Delegation

Stub Impl

Client Interceptor

Client

Client Ref
JRMP Portable Interceptor Overview

Server Interceptor

Get ServiceContext

ServiceContext Array

Client Interceptor

Add ServiceContext

ServiceContext Array

ServerRef, RemoteRef architecture
Contents

† RMI Overview
† Multi-Protocol with RMI-IIOP
† Multi-Name Services with JNDI
† RMI Context Propagation
† Conclusion
Conclusion

† Configuration problem
  ✤ Xml file, properties … ?

† Jonathan Portable Interceptor Problem
  ✤ End of the summer

† Referenceable Problem
  ✤ The CosNaming doesn’t accept Referenceable binding

† Interceptor Problem
  ✤ org.omg.PortableInterceptor.* interfaces for JRMP Problem

† Stub jar location problem
  ✤ One stub jar for each protocol

† The prototype works fine !
  ✤ Both RMI IIOP and RMI JRMP access
  ✤ XML file configuration
  ✤ JRMP Context propagation with JRMP Portable Interceptor
  ✤ IIOP Context propagation with CORBA Portable Interceptor
  ✤ MultiInitialContext