CORBA Asynchronous Messaging

Bill Binko
The Technical Resource Connection
cwbinko@trcinc.com
Presentation Preview

- CORBA Messaging Overview
- Asynchronous Method Invocation (AMI)
- Time Independent Invocation (TII)
- QoS Framework Overview
- MOM Product Integration Strategies
- Miscellaneous Issues
- Open Discussion and Q/A
CORBA Messaging is an integral part of version 2.4 of CORBA

Provides *strongly typed* asynchronous communications

Adds support for such QoS as store-and-forward and priority delivery

Allows administrative routing of requests
Background – RPC Vs. MOM

- CORBA originally based on RPC systems
  - Generated Stubs/Skeletons provide networking support
  - IDL Compiler handles converting the invocation’s parameters and return value into remote request/reply messages

- Remote calls “look like” local calls (Location Transparency)

- Provides Compile-Time Error Checking
  - Request and Reply Messages will always be in correct form (syntax, not semantics)
Background – RPC Vs. MOM

- CORBA’s RPC Issues
  - Forces a strict request/reply sequence (Blocking)
  - Very Connection-oriented
  - Tight Coupling (At several levels)
Background –
RPC Vs. MOM

- Message Oriented Middleware products use messages and queues
  - Allows decoupling of sender/receiver
  - Allows administration of routing and QoS
  - Provides little/no type safety
- Request and Replies are independently targeted
- Messages are self-contained
  - Contain all information needed for deliver and execution
  - Are often meaningful without the originator’s presence
Background – RPC Vs. MOM

- Does NOT look like local calls
  - Messages are built by hand and given to the MOM interface
  - XML Becoming standard format for Messages (JAXM, BizTalk, etc.)
Background –
RPC Vs. MOM

- Typing discussion
Background – Messaging

Goals

- Provide some of the benefits of MOM
  - QoS such as store/forward, priority, etc.
  - Administrative routing
  - Disconnected clients/servers
  - No P&S etc. (Notification covers that)

- Maintain type-safety

- Interoperate with existing CORBA servers

- Client-only changes (except for OTS Calls)
Background: Messaging QoS

- Priority: Controls the Request Priority while in Queues
- Timing: Controls various timeout aspects
  - Request Start Time, Timeout
  - Reply Start Time, Timeout
  - Round-Trip Timeout, etc.
- Routing: Controls Forwarding or Store/Forwarding
- Max Hops: Sets max hops before request dies
- Ordering: Sets priority/temporal/”deadline” sorting in queue
Usage Scenario: Client Disconnection

Buyers

Find Item
MarketOrder
Price, etc.

Exchange
Usage Scenario: Client Disconnection

Order Entry

NYSE  CBOE  NASD

<table>
<thead>
<tr>
<th>Symbol</th>
<th>AMZN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Quantity</td>
<td>1000</td>
</tr>
<tr>
<td>Price/Share</td>
<td>135</td>
</tr>
<tr>
<td>Order Type</td>
<td>Limit Order</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Order Id</th>
<th>Security</th>
<th>Req. Qty/$</th>
<th>Qty/$</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>OID3958</td>
<td>ORCL</td>
<td>100/$5</td>
<td>N/A</td>
<td>Pending</td>
</tr>
<tr>
<td>OID35682</td>
<td>MSFT</td>
<td>500/$52</td>
<td>500/$52 1/2</td>
<td>Complete</td>
</tr>
</tbody>
</table>
Usage Scenario: Client Disconnection
Usage Scenario: Enhanced QoS
Usage Scenario: Routing Negotiations
Usage Scenario:
Routing Negotiations
Usage Scenario: Registration (AMI)
Background – Problems

- GIOP (CORBA’s transport protocol) is very request/reply oriented
- GIOP Reply message has no destination field (cannot be routed)
- Clients have no identity in CORBA
- Users need an alternative to “normal” call semantics
Background – Problems

Blocking Calls

- **IDL**

  ```idl
  interface A {
    long f(in float a, out float b, inout float c);
  };
  ```

- **Java**

  ```java
  A ref = AHelper.narrow(someObjRef);

  FloatHolder bH = new FloatHolder();
  FloatHolder cH = new FloatHolder(1.2);

  int result = ref.f(2.0, bH, cH);
  //use result, bH, cH
  ```
Background – Approach

- **Decouple Request/Reply at App Level**
  - AMI: New invocation interface (peer to SII and DII)
  - Allows typed asynchronous calls (via Callbacks/Polling)

- **Decouple Request/Reply at Wire Level**
  - Specify Router Interfaces (TII)
  - Wraps GIOP Request/Reply in Routable Messages
  - Allows MOM-type queuing/routing of GIOP-level messages

- **Extensible QoS Policy Framework**
  - Provides control over the system
AMI: Asynchronous Method Invocation

- Allows the ORB to separate a request from its reply in a type-safe manner
- Two models: Callback and Polling
- Both rely on the IDL compiler for support (special AMI stubs)
- Like SII/DII – client-side only: server can’t tell which interface is used
  - When used with TII, Transactional Servers must know about it (more on that later)
For comparison, we will go over all of the available invocation interfaces:

- (Normal) Synchronous Call
- Oneway Call
- Deferred Synchronous Call
- Asynchronous Call (new)

Disclaimer: These are typical call-paths and are not complete. They are solely for illustration.
AMI Comparison: SII  
Synchronous Call

**IDL**

```idl
interface A {
    long f(in float a, out float b, inout float c);
};
```

**Java (C++, C, Smalltalk, Perl, COBOL, Others Available)**

```java
A ref = AHelper.narrow(someObjRef);

FloatHolder bH = new FloatHolder();
FloatHolder cH = new FloatHolder(1.2);

int result = ref.f(2.0, bH, cH);
//use result, bH, cH
```
AMI Comparison: SII
Synchronous Call

1: Normal Call (wait)
2: Create Request
3: Marshal Params
4: Send Request (wait)
5: Send Data (IIOP/GIOP/other) (wait)
6: Create
7: Deliver
8: invoke
9: Unmarshal Params
10: Normal Call
11: Marshal Results
12: Send Reply
13: Send Reply Data (#5 wakes)
14: Update w/Results (#4 wakes)
15: Unmarshal Results
16: Return
AMI Comparison: Oneway Call

- **IDL**
  ```
  interface A {
    oneway void g(in float a);
  }; 
  ```

- **Java**
  ```
  A ref = AHelper.narrow(someObjRef);
  // maybe set SYNC_SCOPE on ref

  ref.g(2.0);
  ```
AMI Comparison: SII Oneway Call

1: Normal Call (wait)
2: Create Request
3: Marshal Params
4: Send Request (wait?)
5: Send Data (IIOP/GIOP/other)
6: Create
7: Deliver
8: invoke
9: Unmarshal Params
10: Normal Call

Client Code

Generated Stub

Obj Adapter (POA)

Generated Skeleton

Servant

Server Request

Client ORB

Server ORB

SYNC NONE

SYNC_SERVER

SYNC_Target

SYNC_TRANSPORT

Request
AMI Comparison: DII
Deferred Sync Call

**IDL**

```idl
interface A {
    long h(in float a);
}
```

**Java**

```java
org.omg.CORBA.Request r = someObjRef._request("h");
r.set_return_type(
    _orb().get_primitive_tc(TCKind.tk_long));
Any s = r.add_in_arg();
s.insert_float(2.0);
r.send_deferred(); //Can do other stuff here
r.get_response();  //will wait for result
int result = r.return_value().extract_long();
//use result
```
AMI Comparison: DII Deferred Sync Call

1: Create Request
2: Marshal Params
3: Send Request (Don't Wait)
4: Send Data (IIOP/GIOP/other) (wait)
5: Create
6: Deliver
7: invoke
8: Unmarshal Params
9: Normal Call
10: Marshal Results
11: Send Reply
12: Send Reply Data (#5 wakes)
13: Update w/Results
14: (opt.)Get Results
AMI Comparison: Async Polling Model (new)

**IDL**

```java
interface A {
    long f(in float a, out float b, inout float c);
};
```

**Java**

```java
A ref = AHelper.narrow(someObjRef);
//ref is actually a stub
//only pass in/inout params
AMI_APoller poller = ref.sendp_f(2.0,1.2);
//Can do other stuff
FloatHolder bh = new FloatHolder();
FloatHolder ch = new FloatHolder();
int result = poller.f(/*timeout*/-1,bh,ch);
//Can use result, b, c
```
AMI Comparison: Async Polling Model (new)

- **Client Code**
  - 1: Sendp call
  - 6: return poller

- **Generated AMI Stub**
  - 2: Create Request
  - 3: Marshal Params

- **Request**
  - 4: Send Request
  - 7: Send GIOP Request

- **Server ORB**
  - 8: Normal Server Processing
  - 9: Send GIOP Reply
  - 10: update w/results

- **Type Specific Poller**
  - 5: create wrapper for request
  - 11: get results
  - 12: unmarshal results
  - 13: return results/error
AMI Comparison: Async Callback (new)

**IDL**

```idl
interface A {
    long f(in float a, out float b, inout float c);
};
```

**Java**

```java
A ref = AHelper.narrow(someObjRef);
//ref is actually a stub
//create handler (implicit act. On root POA)
AHandler_impl handler = new AHandler_Impl();
//only pass in/inout params
ref.sendc_f(handler._this(), 2.0, 1.2);
//handler’s f() method will be called
//back w/ inout/out/retval
```
AMI Comparison: AMI Callback Implied IDL

- **IDL**
  ```
  interface A {
    long f(in float a, out float b, inout float c);
  };
  ```

- **Implied Callback IDL**
  ```
  interface AMI_AHandler {
    f(in int retVal, in float b, in float c);
    f_excep(in AMI_AExceptionHolder holder);
  };
  ```
### Java Callback Implementation

```java
public class AHandler_Impl extends POA_AMI_AReplyHandler {
    public void f(int retVal, float b, float c) {
        // use values
    }
    public void f_excep(AMI_AExceptionHolder holder) {
        // handle exception on f() call
        System.err.println("It Failed: " + holder);
    }
}
```
AMI Comparison: Async Callback (new)

1: Sendc call with RH Reference
2: Create Request
3: Marshal Params
4: Send Request as Async w/RH

Note client continues execution here

5: Send GIOP Request
6: Normal Server Processing
7: Send GIOP Reply
8: Convert Reply to RH Request
9: deliver
10: invoke RH Method

Generated AMI Stub

Client Code

Request

Reply Handler

Obj Adapter (POA)

Unmarshalling not shown

Server ORB
Results of AMI?

- Part One Complete: Application Code now uses decoupled requests and replies
- Users of Deferred Sync now have a typed interface (actually two – polling/callback)
- Callbacks can be used on existing servers (a big plus!) without IDL change
- Ready to decouple the wire protocol via Routers
AMI Questions?
AMI Review

- Part One Complete: Application Code now uses decoupled requests and replies
- Users of Deferred Sync now have a typed interface (actually two – polling/callback)
- Callbacks can be used on existing servers (a big plus!) without IDL change
- Ready to decouple the wire protocol via Routers
TII & Routers – Overview

- TII = Time Independent Invocation
  - Second (Optional) section of Messaging
  - Consists of Interoperable Routing Protocol
  - Defines CORBA Messaging Router Interfaces for MOM Products (or native MOM implementations)

- Even with AMI, connections between Clients and Servers must be maintained due to the RPC nature of CORBA’s transport (GIOP)
TII – Overview

- Messaging uses Routers to move Requests/Replies as first-class messages
  - Client ORB wraps Request in a Message and sends it to the Initial Router
  - Target Router makes GIOP call on server
  - Reply is routed as a wrapped Request
TII - The Initial Router
(Callback Interface)

- Initial Routers Accept wrapped Requests
  - Client ORB determines Init. Router and whether to use TII from IOR and Policies
  - Client ORB passes ReplyHandler for return call
Initial Routers Hold ReplyHandlers For Pollers

- Client ORB creates Pollers that query PersistentRequest interface for results
- Client can disconnect since ReplyHandler is in Router

```
poller = sendp_f(params)
poller = sendp(Request)
create_persistent_request()
result = f()
getReply()
creates
```

Initial Router

- Client Application
- Poller
- Stub
- Client ORB
- PersistentRequest/ReplyHandler
- `create_persistent_request()`
TII: Simplest Case

- One Router is Initial and Target Router
  - Accepts Wrapped Requests
  - Invokes Server
  - Converts Reply to Request
  - Invokes ReplyHandler
**TII – RequestInfo Structs**

- RequestInfo structs holds:
  - Original Request Info (needed to invoke the request)
  - QoS info
  - ReplyHandler

RequestInfo Message is passed among Routers until Target is invoked.
TII – Benefits

- With Request/Reply as First Class Messages, most MOM QoS can work.
- Administrators can modify routing, etc.
AMI ⇒ TII: Changes Needed

- Almost none!
- Only need to set the Routing Policy from ROUTE_NONE to ROUTE_FORWARD or ROUTE_STORE_FORWARD
- Configure Client ORB to use Init. Router
- Optionally set RoutingList hints on server
- No other changes to code are necessary.
TII: TII Call Processing

1: Sendc call with RH Reference

2: Create Request

3: Marshal Params

4: Send Request as Async w/RH

5: Send RequestMessage

6: Send RM

7: Send RequestMessage

8: Send GIOP Request

9: Normal Server Processing

10: Send GIOP Reply

11: Convert Reply to RequestMessage

12: Send RequestMessage

13: SendRM

14: Send GIOP Request

15: deliver

16: invoke RH method

Note client continues execution here

Unmarshalling not shown
QoS Framework: Overview

- Provides mechanism for application to control the Messaging System
- Uses Policy interface already in place
- Is set at various points:
  - Default – at POA (Server Side)
  - Client ORB (Client Side)
  - Current Thread (Client Side)
  - Object Ref (Client Side)
QoS Framework: Messaging QoS Menu

- Priority: Controls the Request Priority while in Queues
- Timing: Controls various timeout aspects
  - Request Start Time, Timeout
  - Reply Start Time, Timeout
  - Round-Trip Timeout, etc.
- Routing: Controls Forwarding or Store/Forwarding
- Max Hops: Sets max hops before request dies
- Ordering: Sets priority/temporal/”deadline” sorting in queue
MOM Integration: Wrapping MOM Product

The diagram illustrates the integration of MOM (Message Oriented Middleware) with a GIOP (Generalized Inter-ORB Protocol) connection using an application server and an implementation (servant). The flow includes:

- **Client Application** sending requests through the Client ORB.
- **MOM Product** as the central component, handled by **Router 1** and **Router 3**.
- **Persistant Storage** for maintaining data.
- **GIOP Connection** between the server and client ORBs.
- **RequestInfo** and **ReplyHandler Request** for request and response handling.

The diagram highlights the integration points and the flow of messages within this MOM integration scenario.
MOM Integration: Wrapping JMS Product
MOM Integration:
MOM Supports TII

The diagram illustrates the integration of MOM (Message-Oriented Middleware) with TII (Transaction Information Infrastructure) using GIOP (General Inter-ORB Protocol). The process involves:

1. **Client Application** making a request through the **Client ORB**.
2. The request is handled by the **Router** and passed to the **MOM Product**.
3. The **MOM Product** processes the request and generates the **RequestInfo**.
4. The **MOM Product** then sends the request to the **IIOP Invoker**.
5. The **IIOP Invoker** handles the request and sends it to the **Server ORB**.
6. The **Server ORB** processes the request and sends the reply to the **MOM Product**.
7. The **MOM Product** sends the reply to the **MOM Queue**.
8. The **MOM Queue** stores the reply for further processing.

The process involves both **Standard GIOP calls** and **MOM-Specific routing** to handle requests and responses efficiently.
MOM Integration: MOM Bridges TII

The diagram illustrates the integration of MOM (Message Oriented Middleware) with the TRC (Technical Resource Connection) system. It shows how a client application interacts with the MOM product through standard GIOP (General Inter-ORB Protocol) calls into a router interface. The router converts these calls into SOAP (Simple Object Access Protocol) invocations, which are handled by the MOM product through its MOM queue. The SOAP server receives these invocations and processes them, sending replies back through the MOM queue.

Key components include:
- **Client Application**
- **Client ORB**
- **Client ORB**
- **Router**
- **MOM Product**
- **XML/SOAP Invoker**
- **SOAP SERVER**
- **MOM Queue**
- **GIOP/SOAP Converter**
- **MOM Protocol**

The diagram also highlights the roles of message handling and routing, including request and reply handling, and the conversion of GIOP calls into SOAP invocations.
Misc Issues

- AMI does not effect OTS: It works fine
- TII inherently breaks OTS
- Disconnected client cannot participate in 2PC
- New Transaction Mode: “Unshared”
  - New Transaction “Each Hop”
  - Used for STORE_AND_FORWARD and existing servers that “require” a transaction context
  - OTS-aware Servers can say whether they will participate in Unshared Transaction
Unshared Transaction Example
Availability

- Part of CORBA 2.4
- Currently in development by major vendors and third parties
- RTF complete: issues rolled into ORB Revision
  - Specification Doc: orbos/98-05-06
  - RTF Output is available