RPC over DDS
eProsima Revised Submission

OMG Tech Meeting, Berlin 2013
19/06/2013

Jaime Martin Losa
CTO eProsima
JaimeMartin@eProsima.com
+34 607 91 37 45

www.eProsima.com
Agenda

- Proposed Solution
- Definition of the Services
- Use of DDS Infrastructure: Topic Mapping
- API
- Exceptions
- QoS
- Proposed Interface/Operation QoS Annotation
- C++ 11 and Java 5 Support
Proposed Solution
Proposed solution: Manual

```java
interface MyInterface {
    funReturnValueType Fun(in MyType1 m1, in MyType2 m2, ...,
                           out MyTypeO1 n1, out MyTypeO2 n2, ...,
                           inout MyType1 l1, inout MyType2 l2 );

    /* ... */
}
```

![Diagram of global data space with client proxy, server skeleton, and topic fun requests/replies]
Proposed solution: Automatic
Definition of the Services
Definition of the services

- OMG IDL 3.5 will be used
- “Any” and “valuetype” not supported
  - DDS does not support these
- Exceptions supported
- Attributes in the interfaces ignored
- Oneway invocations supported through the IDL oneway keyword
Use of DDS Infrastructure. Topic Mapping
Topic Mapping

- For each Operation a Request/Reply Topic:
  - [InterfaceName][OperationName][Request/Reply]

- Advantages:
  - Decouples operations. A different channel per operation.
  - Straightforward. Easy to understand.
  - Protocol dissectors friendly.
    - The user can understand easily the contents of the messages.

- Disadvantages:
  - Scalability: Proliferation of topics
Topic Mapping: Alternative

- For each Service and operation:
  - [ServiceName]_[InterfaceName]_[OperationName][Request/Reply]

- Advantages:
  - Same as before, plus better performance
    - The server does not have to filter the requests per service

- Disadvantages:
  - Scalability: Proliferation of Topics
# Mapping: Request & Reply Topics

**MyInterface_FunRequest Topic**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientId</td>
<td>GUID_t</td>
</tr>
<tr>
<td>RemoteServiceName</td>
<td>string&lt;255&gt;</td>
</tr>
<tr>
<td>RequestSequenceNumber</td>
<td>SequenceNumber_t</td>
</tr>
<tr>
<td>In &amp; InOut FunParameters</td>
<td>FunParametersType</td>
</tr>
</tbody>
</table>

**MyInterface_FunReply Topic**

<table>
<thead>
<tr>
<th>Field</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>ClientId</td>
<td>GUID_t</td>
</tr>
<tr>
<td>RequestSequenceNumber</td>
<td>SequenceNumber_t</td>
</tr>
<tr>
<td>Union</td>
<td></td>
</tr>
<tr>
<td>Out Parameters &amp; Return value</td>
<td></td>
</tr>
<tr>
<td>InOut &amp; Out FunParameters</td>
<td>FunParametersType</td>
</tr>
<tr>
<td>FunReturnValue</td>
<td>FunReturnValueType</td>
</tr>
<tr>
<td>System Exception</td>
<td>SystemException_t</td>
</tr>
<tr>
<td>User Exception 1</td>
<td>UserException1Type</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
<tr>
<td>User Exception n</td>
<td>UserExceptionnType</td>
</tr>
</tbody>
</table>
Mapping Details

- The **clientId** is sent to identify the client. The server will publish replies include the clientId, thus the client can identify the replies directed to him.
- The **RemoteServiceName** is published to indicate the desired service implementation. The corresponding server service implementation should filter its request by its service name.
- The **RequestSequenceNumber** is used to correlate the client request with the server reply. The server will include this sequence name in the reply.
- Besides the clientId and RequestSequenceNumber to correlate the reply with the corresponding request, the Reply Topic will contain an union containing either the InOut/Out parameters and the procedure return value, or an exception, system or user generated.
- In the case of **oneway procedures**, no Reply is sent, thus there is no guarantee the call succeeds.
Why use Topic per operation mapping?

- Avoid operation **head of line blocking**
- Natural and transparent use of DDS
- Interoperable
interface MyInterface {
    funReturnValueType Fun(in MyTypeI1 m1, in MyTypeI2 m2, ..., 
                           out MyTypeO1 n1, out MyTypeO2 n2, ..., 
                           inout MyType1 l1, inout MyType2 l2);
    /* ... */
};
```cpp
interface MyInterface {
    funReturnValueType Fun(in MyTypeI1 m1, in MyTypeI2 m2, ..., 
    out MyTypeO1 n1, out MyTypeO2 n2, ..., 
    inout MyType1 l1, inout MyType2 l2 );
    /* ... */
};
```

### MyInterfaceProxy

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Returns/Type</th>
<th>Raises</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>MyInterfaceProxy</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
<tr>
<td></td>
<td>RemoteServiceName</td>
<td>string</td>
<td></td>
</tr>
<tr>
<td></td>
<td>domain_id</td>
<td>DomainId_t</td>
<td></td>
</tr>
<tr>
<td></td>
<td>timeout</td>
<td>Duration_t</td>
<td></td>
</tr>
<tr>
<td>enable</td>
<td>void</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
<tr>
<td>disable</td>
<td>void</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
<tr>
<td>Fun</td>
<td>FunReturnValueType</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
<tr>
<td>FunParameters</td>
<td>FunParametersType</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fun_async</td>
<td>void</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
<tr>
<td>handler</td>
<td>Fun_CallBackHandler</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
<tr>
<td>In &amp; InOut FunParameters</td>
<td>FunParametersType</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Generated Proxy: Async calls

```java
interface MyInterface {
    funReturnValuetype Fun(in MyTypeI1 m1, in MyTypeI2 m2,...,
                        out MyTypeO1 n1, out MyTypeO2 n2,...,
                        inout MyTypeI1 l1, inout MyTypeI2 l2);
    /* ... */
};
```

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Returns/Type</th>
<th>Raises</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fun_async</td>
<td>void</td>
<td></td>
<td></td>
</tr>
<tr>
<td>handler</td>
<td>Fun_CallBackHandler</td>
<td>In &amp; InOut FunParameters</td>
<td>FunParametersType</td>
</tr>
</tbody>
</table>

**Fun_CallBackHandler**

- no attributes
- operations

```
<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Returns/Type</th>
<th>Raises</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fun</td>
<td>FunReturnValueType</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
<tr>
<td>on_exception</td>
<td>InOut &amp; Out FunParameters</td>
<td>void</td>
<td></td>
</tr>
<tr>
<td>exception</td>
<td>exception</td>
<td>DDSRPC::Exception</td>
<td></td>
</tr>
</tbody>
</table>
```
**Generated Server**

```java
interface MyInterface {
    funReturnValueType Fun(in MyTypeI1 m1, in MyTypeI2 m2, ..., 
                            out MyTypeO1 n1, out MyTypeO2 n2, ..., 
                            inout MyType1 l1, inout MyType2 l2 );
    /* ... */
}
```

### MyInterfaceServer

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Returns/Type</th>
<th>Raises</th>
</tr>
</thead>
<tbody>
<tr>
<td>create</td>
<td>ServiceName, strategy, domain_id</td>
<td>MyInterfaceServer, DDSRPC::ServerStrategy, DomainId_t</td>
<td>DDSRPC::Exception</td>
</tr>
<tr>
<td>serve</td>
<td></td>
<td>void</td>
<td>DDSRPC::Exception</td>
</tr>
<tr>
<td>stop</td>
<td></td>
<td>void</td>
<td>DDSRPC::Exception</td>
</tr>
</tbody>
</table>

no attributes
operations
Generated Server: Skeleton

```java
interface MyInterface {
    funReturnValueType Fun(in MyTypeI1 m1, in MyTypeI2 m2,...,
                          out MyTypeO1 n1, out MyTypeO2 n2,...,
                          inout MyType1 l1, inout MyType2 l2);
    /* ... */
}
```

### MyInterfaceServerImpl

<table>
<thead>
<tr>
<th>Name</th>
<th>Parameters</th>
<th>Returns/Type</th>
<th>Raises</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fun</td>
<td></td>
<td>FunReturnValueType</td>
<td>DDSRPC::Exception</td>
</tr>
<tr>
<td></td>
<td>FunParameters</td>
<td>FunParametersType</td>
<td></td>
</tr>
</tbody>
</table>
Exceptions
Interface Example

The user can define his own Exceptions:

```java
interface MyInterface {
    exception MyException {
        MyTypeE1 exAttrib1;
        ...
        MyTypeEn exAttribn;
    }

    funReturnValueType Fun(in MyTypeI1 m1, in MyTypeI2 m2, ..., out MyTypeO1 n1, out MyTypeO2 n2, ..., inout MyTypeI1 l1, inout MyTypeI2 l2, ...) raises(MyException);
    /* ... */
}
```

The user can define his own Exceptions:
• Besides the user exceptions, a call to a remote procedure can raise system exceptions:
  • **ClientInternalException**: local Internal exception.
  • **ServerInternalException**: remote internal exception.
  • **ServerTimeoutException**: The call timeout has expired.
  • **ServerNotFoundException**: No server discovered.
• The creation of a proxy or a server can raise an **Initialize Exception**.
All system exceptions inherit from DDSRPC::SystemException:

<table>
<thead>
<tr>
<th>SystemException</th>
<th>attributes</th>
</tr>
</thead>
<tbody>
<tr>
<td>Name</td>
<td>Type</td>
</tr>
<tr>
<td>ExceptionId</td>
<td>Short</td>
</tr>
<tr>
<td>Message</td>
<td>String&lt;255&gt;</td>
</tr>
<tr>
<td>no operations</td>
<td></td>
</tr>
</tbody>
</table>
Qos
Qos

- Request and Reply Topics Qos:
  - reliability.kind=DDS_RELIABLE_RELIABILITY_QOS
  - history.kind=DDS_KEEP_ALL_HISTORY_QOS
  - durability.kind=DDS_VOLATILE_DURABILITY_QOS
Proposed Interface/Operation
QoS annotation
Proposed Interface/Operation QoS annotation.

- To set the specific Qos for an operation, the IDL annotation extension presented in the “Extensible and Dynamic Topic Types for DDS” specification should be used.

- The annotation can be applied at interface and/or operation level. The Qos profiles are XML Qos Profiles as defined in the “DDS for lightweight CCM” specification.
Proposed Interface/Operation QoS annotation: Example

```
@Annotation
local interface operationQos {
    attribute string RequestProfile; // “MyRequestProfile”
    attribute string ReplyProfile; // “MyReplyProfile”
}

@operationQos(“MyRequestProfile1”, “MyReplyProfile1”)
interface MyInterface {
    exception MyException {
        MyTypeE1 exAttrib1;
        ...
        MyTypeEn exAttribn;
    }
}

@operationQos(“MyRequestProfile2”, “MyReplyProfile2”)
funReturnValueType Fun(
in MyTypeI1 m1, in MyTypeI2 m2,...,
    out MyTypeO1 n1, out MyTypeO2 n2,...,
    inout MyType1 l1,inout MyType2 l2,... )
    raises(MyException);
/* ... */
```
C++ 11 & Java 5 Support
public class MyException1 extends Exception{
    MyTypeE1 exAttrib1;
    ...
    MyTypeEn exAttribn;
}

@RPC
@operationQos(RequestProfile = "MyRequestProfile1", ReplyProfile="MyReplyProfile1")
public interface MyInterface {

    @operationQos(RequestProfile = "MyRequestProfile1", ReplyProfile="MyReplyProfile1")
    funReturnValueType Fun(@in MyTypeI1 m1,@in MyTypeI2 m2,...,
                            @out MyTypeO1 n1, @out MyTypeO2 n2,...,
                            @inout MyType1 l1,@inout MyType2 l2,... )
        throws MyException1,...;

    /* ... */

};

/* ... */
C++ 11 and Java 5 support

- For C++ 11 the IDL2C++11 spec will be used.
- For Java 5 the IDL to Java specification applies. No specific Java 5 constructs are necessary.
Hands On: Calculator

- Create Calculator.idl

```cpp

// KIARA Webinar Example //

interface Calculator {
    float sum (in float x, in float y); // x+y
    float substract (in float x, in float y); // x-y
    float multiply (in float x, in float y); // x*y
}
```

- Generate Interface Support :

`rpcddsgen -ppDisable -example x64Win64VS2010 Calculator.idl`
Hands On: Calculator

- Edit the Example:
  - Client.cxx

```cpp
// Call to remote procedure "sum".
try {
    sum_ret = proxy->sum(2, 2);
    std::cout << "Server Says 2+2=" << sum_ret << std::endl;
}
```

- CalculatorServerImpl.cxx

```cpp
DDS_Float CalculatorServerImpl::sum(/*in*/ DDS_Float x, /*in*/ DDS_Float y) {
    DDS_Float sum_ret = 0;
    sum_ret = x + y;
    return sum_ret;
}
```
Thank you!

Jaime Martin Losa
CTO eProsima
JaimeMartin@eProsima.com
+34 607 91 37 45

www.eProsima.com