eProsima The Middleware Experts

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

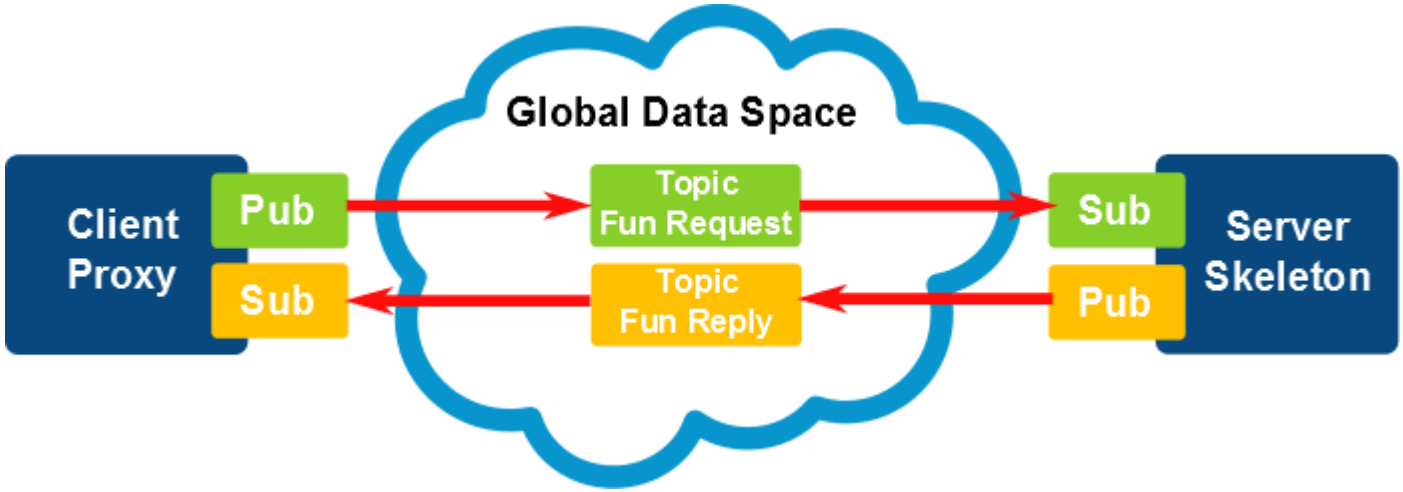
- Proposed Solution
- Definition of the Services
- Use of DDS Infrastructure: Topic Mapping
- API
- Exceptions
- QoS
- Proposed Interface/Operation Qos Annotation
- C++ 11 and Java 5 Support

Proposed Solution

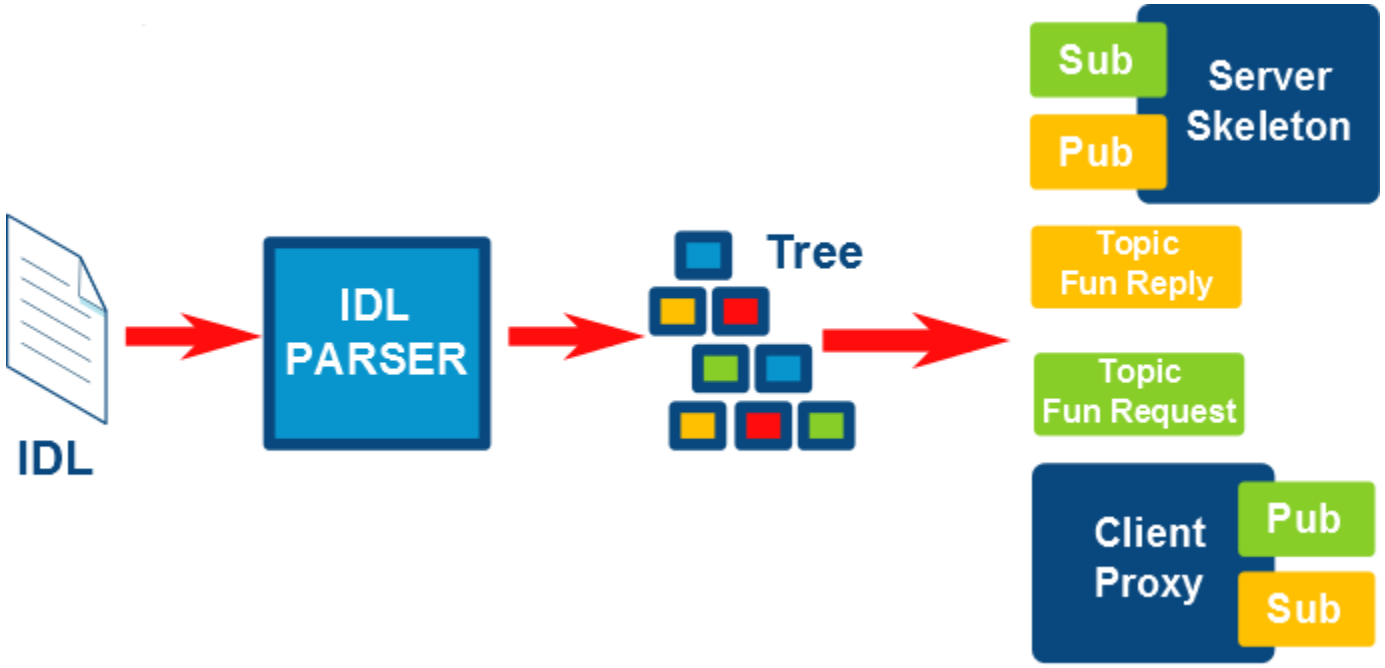
Proposed solution: Manual

```
interface MyInterface {
        funReturnValueType Fun(in MyTypeI1 m1,in MyTypeI2 m2,...,
                               out MyTypeO1 n1, out MyTypeO2 n2,...,
                                   inout MyType1 I1,inout MyType2 I2 );

        /* ... */
};
```

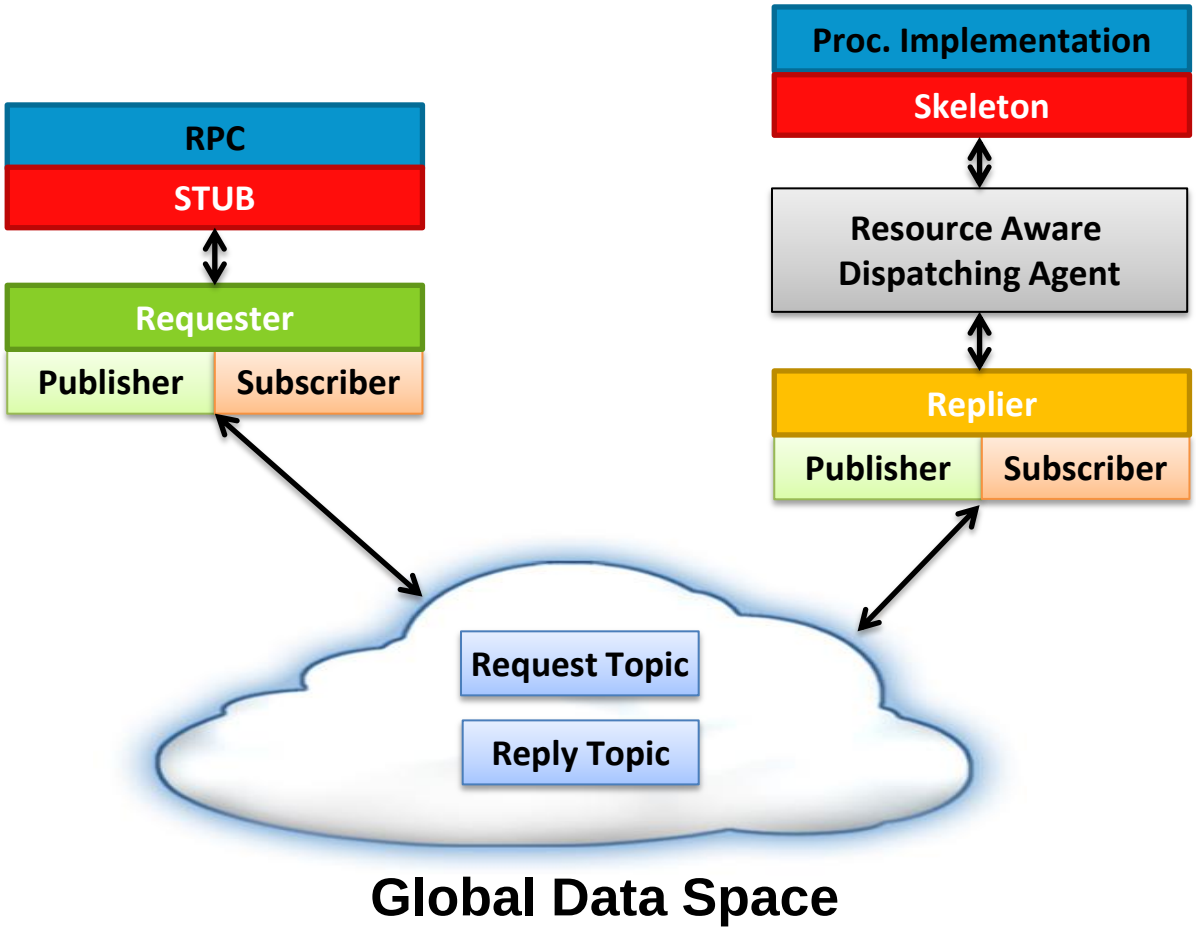


Proposed solution: Automatic



eProsima

Architecture



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
- Atributes in the interfaces ignored
- Oneway invocations supported throught the IDL oneway keyword

Use of DDS Infrastructure. Topic Mapping

Topic Mapping

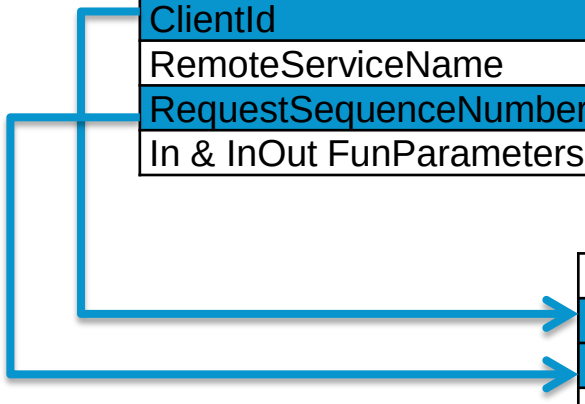
- For each Operation a Request/Reply Topic:
  - [InterfaceName]_[OperationName][Request/Reply]
- Advantages:
  - Decouples operations. A different channel per operation.
  - Straightforward. Easy to understand.
  - Protocol disectors friendly.
    - The user can ununderstand easily the contents of the 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	
ClientId	GUID_t
RemoteServiceName	string<255>
RequestSequenceNumber	SequenceNumber_t
In & InOut FunParameters	FunParametersType



MyInterface_FunReply Topic	
ClientId	GUID_t
RequestSequenceNumber	SequenceNumber_t
Union	
Out Parameters & Return value	
InOut & Out FunParameters	FunParametersType
FunReturnValue	FunReturnValueType
System Exception	SystemException_t
User Exception 1	UserException1Type
…	
User Exception n	UserExceptionnType

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

API

Interface example

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

Generated Proxy

```
interface MyInterface {
        funReturnValueType Fun(in MyTypeI1 m1,in MyTypeI2 m2,...,
                               out MyTypeO1 n1, out MyTypeO2 n2,...,
                                   inout MyType1 I1,inout MyType2 I2 );
        /* ... */
};
```

MyInterfaceProxy			
no attributes			
operations			
Name	**Parameters**	**Returns/Type**	**Raises**
create		MyInterfaceProxy	DDSRPC::Exception
	RemoteServiceName	string	
	domain_id	DomainId_t	
	timeout	Duration_t	
enable		void	DDSRPC::Exception
disable		void	DDSRPC::Exception
Fun		FunReturnValueType	DDSRPC::Exception
	FunParameters	FunParametersType	
Fun_async		void	
	handler	Fun_CallBackHandler	
	In & InOut FunParameters	FunParametersType	

Generated Proxy: Async calls

```
interface MyInterface {
        funReturnValueType Fun(in MyTypeI1 m1,in MyTypeI2 m2,...,
                               out MyTypeO1 n1, out MyTypeO2 n2,...,
                                   inout MyType1 I1,inout MyType2 I2 );
        /* ... */
};
```

MyInterfaceProxy			
Name	**Parameters**	**Returns/Type**	**Raises**
.....			
Fun_async		void	
	handler	Fun_CallBackHandler	
	In & InOut FunParameters	FunParametersType	

Fun_CallBackHandler			
no attributes			
operations			
Name	**Parameters**	**Returns/Type**	**Raises**
Fun		FunReturnValueType	DDSRPC::Exception
	InOut & Out FunParameters	FunParametersType	
on_exception		void	
	exception	DDSRPC::Exception	

Generated Server

```
interface MyInterface {
        funReturnValueType Fun(in MyTypeI1 m1,in MyTypeI2 m2,...,
                               out MyTypeO1 n1, out MyTypeO2 n2,...,
                                   inout MyType1 I1,inout MyType2 I2 );
        /* ... */
};
```

MyInterfaceServer			
no attributes			
operations			
Name	**Parameters**	**Returns/Type**	**Raises**
create		MyInterfaceServer	DDSRPC::Exception
	ServiceName	string	
	strategy	DDSRPC::ServerStrategy	
	domain_id	DomainId_t	
serve		void	DDSRPC::Exception
stop		void	DDSRPC::Exception

Generated Server: Skeleton

```
interface MyInterface {
        funReturnValueType Fun(in MyTypeI1 m1,in MyTypeI2 m2,...,
                               out MyTypeO1 n1, out MyTypeO2 n2,...,
                                   inout MyType1 I1,inout MyType2 I2 );
        /* ... */
};
```

MyInterfaceServerImpl			
no attributes			
operations			
Name	**Parameters**	**Returns/Type**	**Raises**
Fun		FunReturnValueType	DDSRPC::Exception
	FunParameters	FunParametersType	

Exceptions

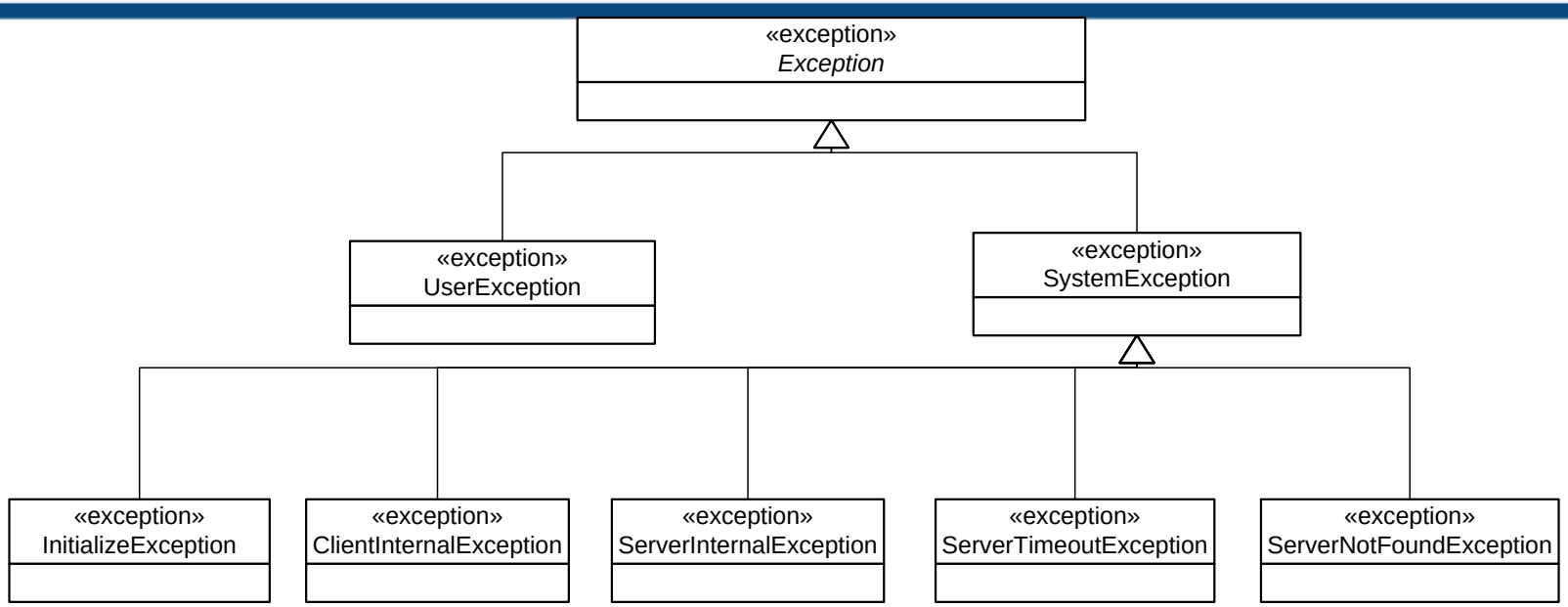
Interface Example

The user can define his own Exceptions:

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

        funReturnValueType Fun(in MyTypeI1 m1,in MyTypeI2 m2,...,
                                out MyTypeO1 n1, out MyTypeO2 n2,...,
                                 inout MyType1 I1,inout MyType2 I2,… )
                raises(MyException);
        /* ... */
};
```

Exceptions inheritance



- 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**.

System Exceptions

All system exceptions inherit from DDSRPC::SystemException:

SystemException	
attributes	
Name	**Type**
ExceptionId	Short
Message	String<255>
no operations	

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 I1,inout MyType2 I2,… )
                raises(MyException);
        /* ... */
};
```

C++ 11 & Java 5 Support

Defining an Interface in Java

```
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 I1,@inout MyType2 I2,… )
                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.

Example

Hands On: Calculator

- Create Calculator.idl

```
////////////////////////////
// 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

eProsima

Hands On: Calculator

- Edit the Example:
  - Client.cxx

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

  - CalculatorServerImpl.cxx

```
DDS_Float CalculatorServerImpl::sum(/*in*/ DDS_Float x, /*in*/ DDS_Float y)
{
    DDS_Float  sum_ret = 0;
    sum_ret  = x+y;

    return sum_ret;
}
```

eProsima

The Middleware Experts

Thank you!

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

www.eProsima.com