Your systems. Working as one.

Remote Procedure Call over DDS

RTI Revised Submission

mars/2013-06-21

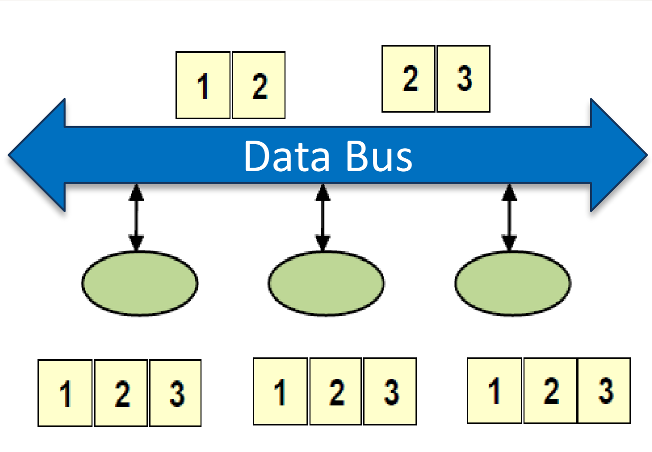
Sumant Tambe, Ph.D.
Senior Software Research Engineer,
Real-Time Innovations, Inc.
sumant@rti.com

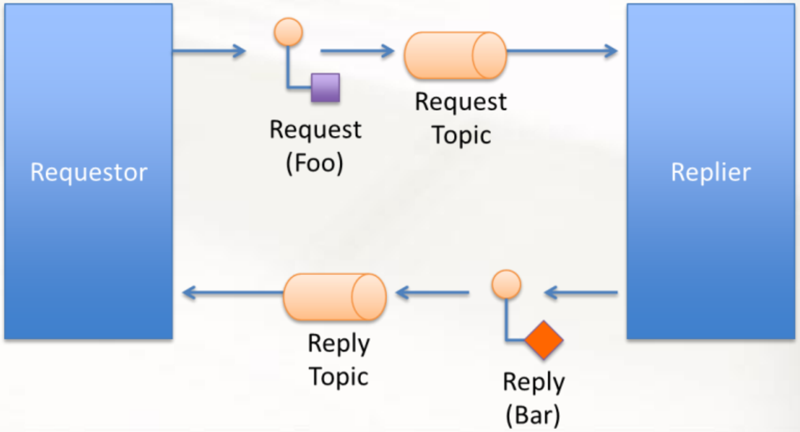
Outline

- Background
  - Advantages of RPC over DDS
- RPC over DDS – Intent
- RPC over DDS – Key Considerations
- RTI's Initial Submission
  - Specification
  - Architecture
  - Mapping of Services to Topics
  - Mapping of Services to Types

Background

- DDS excels at one-to-many communication
- Remote Procedure Call (RPC) implies Request/Reply semantics (bidirectional)
- Cumbersome using vanilla DDS
  - No function call semantics (only write/read)
  - No clear definition of an “invocable service”
  - Setup request topic and type
  - Setup reply topic and type
  - Filter unwanted requests
  - Filter unwanted replies
  - etc…





Advantages of RPC over DDS

- Client and Servers are decoupled
  - No startup dependencies
- QoS enforcement to support service SLAs
  - Ownership–Redundant pool of servers
  - Lifespan–request is only valid for the next N sec
  - Durability-request/replies will be eventually received and processed
  - Cancellation– after a good quality response has arrived
- Data-Centricity
  - Explicit trace-ability
  - Interaction state can be Monitored, Logged, Audited, Stored, Manipulated
    - Watch by subscribing to requests and/or responses (wire-tap)
- One middleware – leverage DDS infrastructure
  - Suited for real-time systems
  - Multiplatform, Multilanguage, Interoperable

RPC over DDS—Intent

- Use the same middleware for pub/sub as well as RPC
  - Reduce cost
  - Flatter learning curve
- Build on top of DDS
  - RPC over DDS—just a special case of DDS
- Leverage the power of DDS for request/reply
  - Data-centric
    - Interaction, datatypes, parameters , exceptions all well-described and visible
  - Reliable QoS for command-response
  - Ownership for active replication of services
- Not to replace CORBA
  - CORBA, in its entirety, is complex
- Maintain interoperability
  - Make RPC over DDS interoperable across vendor implementations

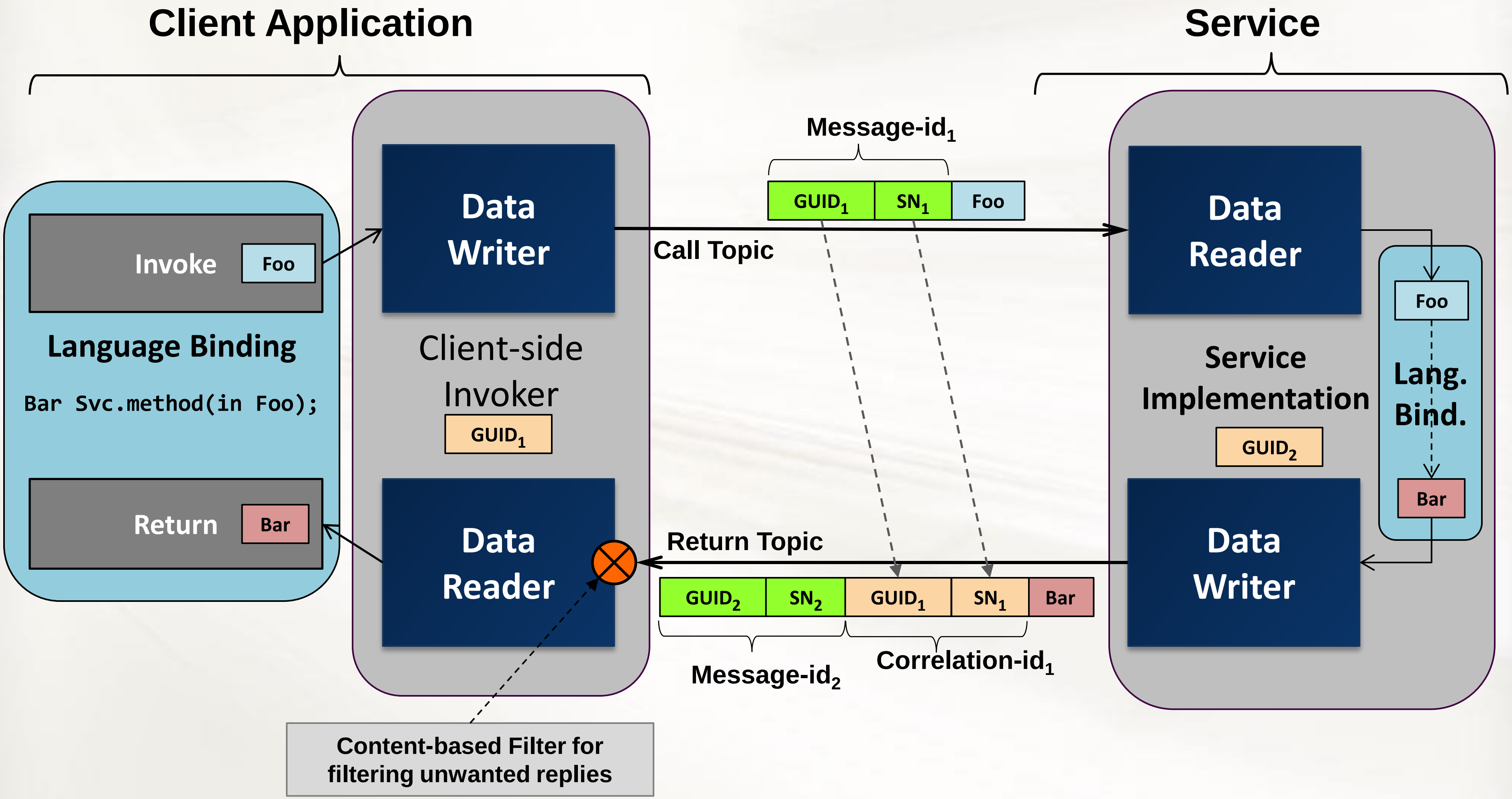
RPC over DDS – Key Considerations

- Scalability
  - Each operation maps to a pair of topics?
    - Consequence: Twice as many topics and four times as many DDS entities (not scalable)
  - Each service maps to a pair of topics?
    - How about 1000s of rarely used services?
  - Many services map to a few topics?
    - Implementable using multiple inheritance (most scalable)
- Data-Centricity
  - How does the request/reply topic types look like?
    - Precisely capture the semantics of interface, method invocation, parameters and return
  - How to model operations?
    - Capture *in*, *out*, *inout* parameters
    - Absence and presence of the return type
    - Polymorphic parameter types
  - Subscribe to requests and responses
    - Highly desirable for logging, auditing, monitoring, etc.

RPC over DDS – Key Considerations

- Discovery
  - Is a service a first-class entity or just a DDS process with two topics?
  - How are the services discovered?
    - Simply add some metadata to the built-in discovery topics
    - Or Publish an instance on a *new* well-known topic (new topic, entities, is that necessary?)
- Service Description
  - RFP requires IDL and Java
- Language Binding and Code Generation
  - Is function call/return semantics integral to RPC?
  - Function call semantics
    - If the data model is right, function-call semantics is a purely *local* concern
    - Must address (complex) language binding issues
  - Request/reply semantics
    - Lower level than function call semantics
    - Typed entities for sending/receiving request/replies
    - More like DDS `read/write` than `call/return` from a function

Architecture

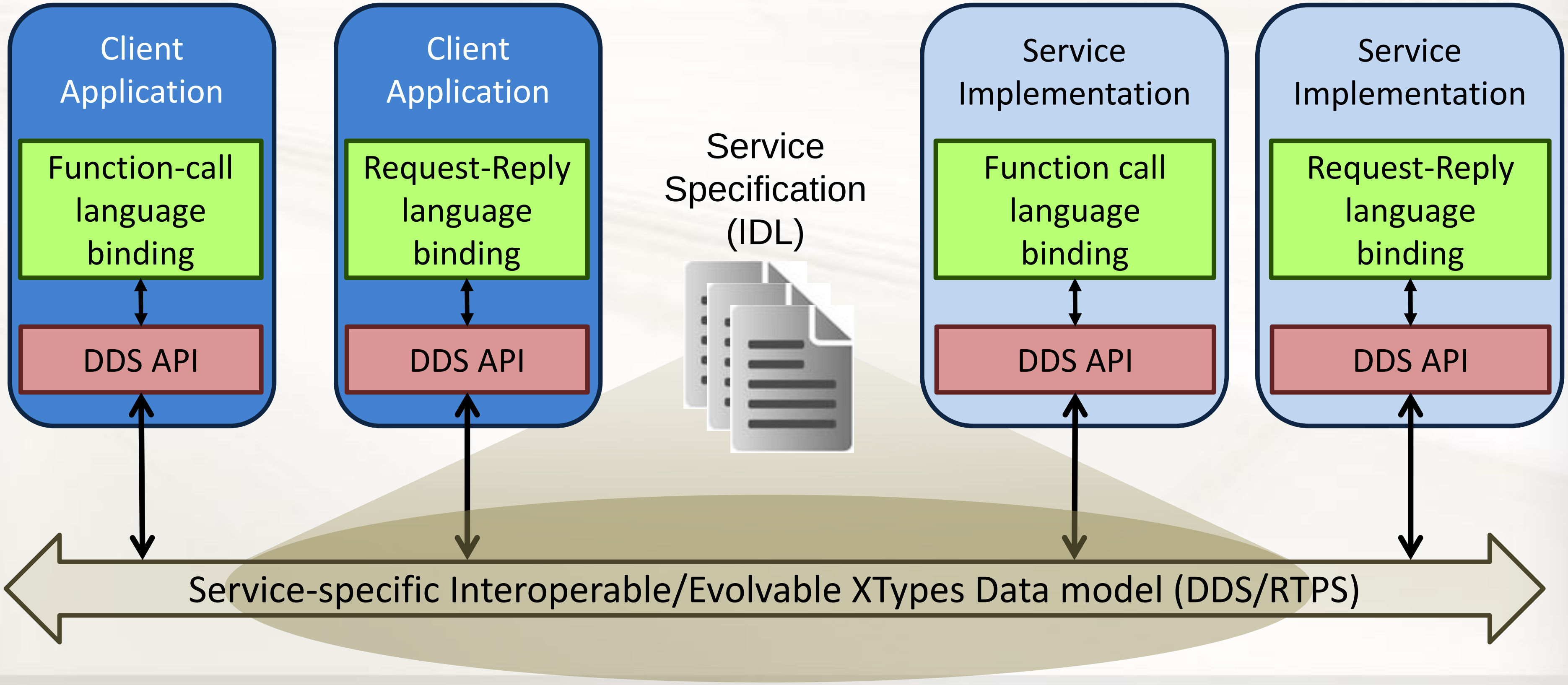


Language Bindings

- Request/Reply semantics
  - Typed API akin to DDS `read/write`
  - Higher-level entities to simplify programming
    - E.g., `Requester<Treq, TRep> , Service<TReq, TRep>`
  - Programmer must create the request and reply samples at the client and service side respectively
  - Return value is just a special member (`_return`)
  - User-defined exceptions are just members
  - System exceptions are captured with a special exception code field
- Function-call semantics
  - Look and feel like a local function-call
  - Bindings that mimic valuetypes will thwart interface evolution
    - For Example, `long` → `int`
  - Special bindings to support optionality when operations are marked `@MutableOperation`

Language Bindings

- RTI's revised submission anticipates two types of language bindings
  - High-level binding that provide *function-call* semantics
  - Low-level binding that are akin to `send/receive` (but still higher level than raw DDS `read/take/write`)
- Strong separation between the data model and language binding


Client Application
Function-call language binding
DDS API
Client Application
Request-Reply language binding
DDS API
Service Specification (IDL)
Service Implementation
Function call language binding
DDS API
Service Implementation
Request-Reply language binding
DDS API
Service-specific Interoperable/Evolvable XTypes Data model (DDS/RTPS)

RTI’s Revised Submission

RTI’s Revised Submission

- Definition of a Service according to the OASIS Service-Oriented Architecture Reference Model

  > *“A mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description.”*

- Specification of interface → IDL
- Specification of constraints → Annotations

```
struct Region { ... }
struct TooFast { ... }
@DDSService
interface RobotControl {
   @oneway void start();
   @oneway void stop();
           bool swapSpeed(inout long s) raises TooFast;
           long getSpeed();
           void setRegion(in Region r);
           Region getRegion();
};
```

Service Specification

- IDL and Java
- Defined using EBNF grammar

```
<specification>   =  <definitions>
<definitions>     =  { <definition> }
<definition>      =  { <interface-dcl> |  <module> }
<module>          =  'module' <identifier> '{' <definitions> '}'
<interface-dcl>   =  <interface-header> '{' <interface-body> '}'  <semicolon-opt>
<interface-header> =  <annotations> ‘interface’ <inherit-spec>
<inherit-spec>    =  [ ‘:’ <identifier> ] <identifiers>
<annotations>     =  { ‘@’ ‘oneway’     |
                       ‘@’ ‘async’      |
                       ‘@’ <identifier> |
                       ‘@’ <identifier> ‘(‘ ( ‘true’ | ‘false’) ‘)’ |
                       ‘@’ <identifier> ‘(‘ <name-value-assign> ‘)’ }
<interface-body>  =  <operation>  { <operation> }
<operation>       =  <annotations> <return-type> <identifier> ‘(‘ <parameter-list> ‘)’ <ex-spec>
<return-type>     =  ‘void’ | <identifier>
<parameter-list>  =  [ <parameter> <parameters> ]
<parameters>      =  { ‘,’  <parameter> }
<parameter>       =  <parameter-attributes> <type> <identifier>
<parameter-attributes> =  ‘in’  |  ‘out’   |  ‘inout’
<ex-spec>         =  [ ‘raises’  ‘(‘  <exception-list> ‘)’ ]
<exception-list>  =  [ <identifier>  <exceptions>  ]
<exceptions>      =  <identifiers>
<identifiers>     =  { ‘,’ <identifier>  }
<name-value-assign> =  { <identifier> ‘=’ <value> }
<value>           =  <digits> | <string>
<semicolon_opt>   =  [ <semicolon> ]
<type>            =  <identifier>
<identifier>      =  <string>
<string>          =  <alpha> { <alpha-numeric> }
<alpha-numeric>   =  <digit> | <alpha>
<digits>          =  <digit>  { <digit> }
<digit>           =  "0" | "1" | "2" | "3" | "4" | "5" | "6" | "7" | "8" | "9"
<alpha>           =  ‘A’ … ‘Z’ | ‘a’ … ‘z’
```

Mapping A Service to Topics

- Each interface is mapped to two DDS topics
  - Request topic
  - Reply topic
- Three ways to specify topic names
  1. Append “Call” and “Return” to the service name
  2. Use Annotations

```
@DDSService
@CallTopic(name=“RobotControlIn”)
@ReturnTopic(name=“RobotControlOut”)
interface RobotControl { ... }
```

  3. Topic names can also be specified at run-time

Mapping of Interfaces to Types

```
struct Region { ... }
struct TooFast { ... }
@DDSService
interface RobotControl {
   @oneway void start();
   @oneway void stop();
           bool swapSpeed(inout long s) raises TooFast;
           long getSpeed();
           void setRegion(in Region r);
           Region getRegion();
};
```

- Mapping of Interfaces and Operations
- Both rely on XTypes

Mapping of Operations

- A pair of Call/Return structures for each operation

```
struct Region { ... }
struct TooFast { ... }
@DDSService
interface RobotControl {
   @oneway void start();
   @oneway void stop();
           bool swapSpeed(inout long s) raises TooFast;
           long getSpeed();
           void setRegion(in Region r);
           Region getRegion();
};
```

```
@Empty @Final struct RobotControl_start_call { };
@Empty @Final struct RobotControl_start_call { };
@Final struct RobotControl_swapSpeed_call {
  long s;
};
@Empty @Final struct RobotControl_getSpeed_call { };
@Final struct RobotControl_setRegion_call {
  Region r;
}
@Empty @Final struct RobotControl_getRegion_call { };
```

Mapping of Operations

- A pair of Call/Return structures for each operation

```
@Empty @Final struct RobotControl_start_call { };
@Empty @Final struct RobotControl_start_call { };
@Final struct RobotControl_swapSpeed_call {
  long s;
};
@Empty @Final struct RobotControl_getSpeed_call { };
@Final struct RobotControl_setRegion_call {
  Region r;
}
@Empty @Final struct RobotControl_getRegion_call { };
```

```
typedef long SystemExceptionCode;

@Choice
struct RobotControl_start_return {
  @md5id @Empty @Final struct Out { } _out;
  @md5id SystemExceptionCode _sysx;
};

@Choice
struct RobotControl_stop_return {
  @md5id @Empty @Final struct Out { } _out;
  @md5id SystemExceptionCode _sysx;
};
```

```
@Choice
struct RobotControl_swapSpeed_return {
  @md5id @Final struct Out {
    long s;
    bool _return;
  } _out;
  @md5id SystemExceptionCode _sysx;
  @md5id TooFast toofast;
};

@Choice
struct RobotControl_getSpeed_return {
  @md5id @Final struct Out {
    long _return;
  } _out;
  @md5id SystemExceptionCode _sysx;
};

@Choice
struct RobotControl_setRegion_return {
  @Empty @md5id @Final struct Out { } _out;
  @md5id SystemExceptionCode _sysx;
};

@Choice
struct RobotControl_getRegion_return {
  @md5id @Final struct Out {
    Region _return;
  } _out;
  @md5id SystemExceptionCode _sysx;
};
```

Annotations

- @Empty
  - Operations with empty parameter list become empty structures
  - But, IDL does not support empty structures
  - The annotation adds an exception
  - Eliminating empty structures is undesirable because two or more operations may take no parameters. It would be ambiguous.
  - Help maintain consistency between call/return members
- @Choice
  - Capture the semantics of a `union` without using a discriminator
  - Operations in an interface have set semantics and have no ordering constraints. Unions, however, enforce strict association with discriminator values, which are too strict for set semantics.
  - Implies `@Extensibility(MUTABLE_EXTENSIBILITY)` and all members `@Optional`
  - Exactly one `@optional` member must be active at any given time.
  - Examples
    - Exactly one operation (out of many in an interface) is invoked at a time
    - Exactly one out of normal return, user-defined exception, and system exception is possible in a reply

Annotations

- @md5id
  - Using ordered ids is brittle for the same reasons as unions are brittle
  - In case of multiple inheritance of interfaces, ordered ids are ambiguous
  - @md5id implies the id of the field is the same as the md5sum of the name of the field
  - IDL guarantees that no two operations have the same name (no overloading)
- @Final
  - The `Out` nested structure in the `Return` structure is @Final to disallow incompatible interface evolution

Mapping of Interfaces

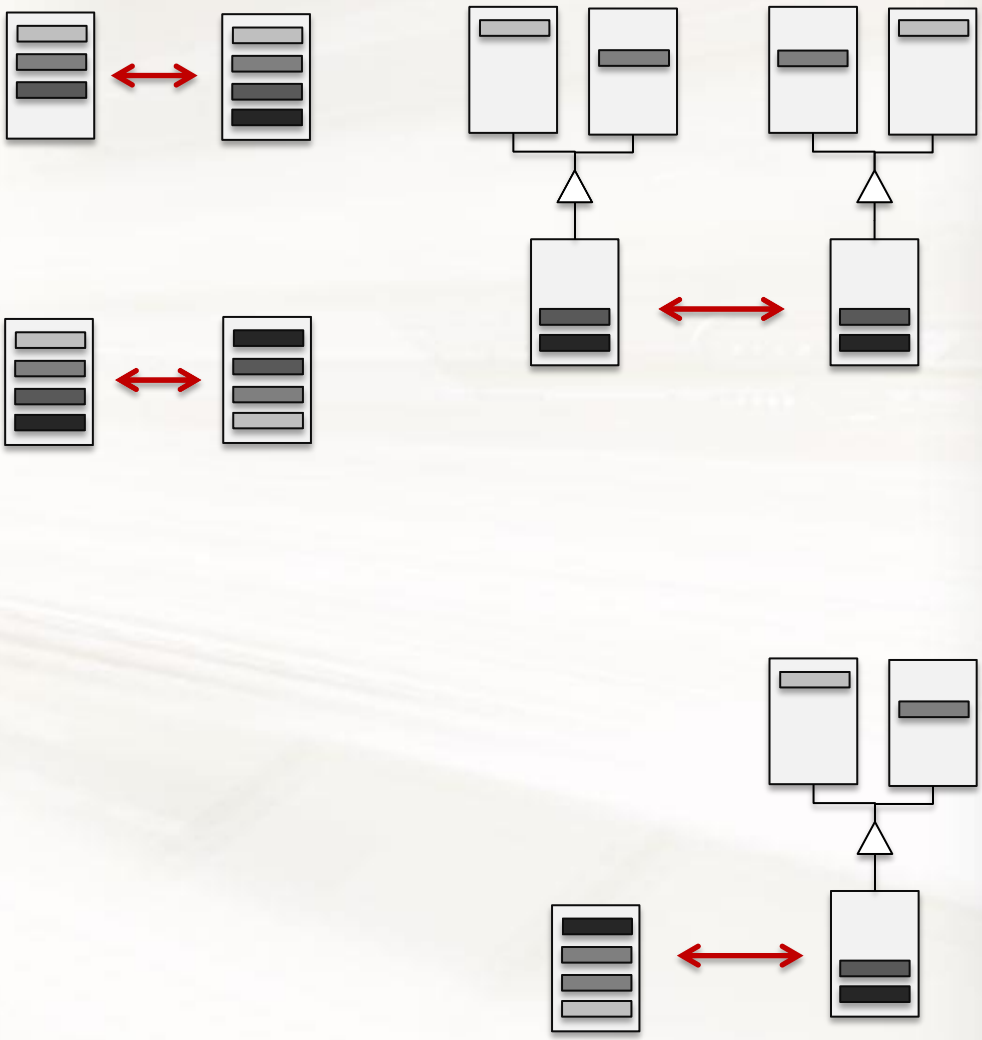
- A pair of Request/Reply structures for each interface

```
@Choice
struct RobotControl_request {

  @md5id
  RobotControl_start_call start;

  @md5id
  RobotControl_stop_call stop;

  @md5id
  RobotControl_swapSpeed_call swapSpeed;

  @md5id
  RobotControl_getSpeed_call getSpeed;

  @md5id
  RobotControl_setRegion_call setRegion;

  @md5id
  RobotControl_getRegion_call getRegion;
};
```

```
@Choice
struct RobotControl_reply {

  @md5id
  RobotControl_start_return start;

  @md5id
  RobotControl_stop_return stop;

  @md5id
  RobotControl_swapSpeed_return swapSpeed;

  @md5id
  RobotControl_getSpeed_return getSpeed;

  @md5id
  RobotControl_setRegion_return setRegion;

  @md5id
  RobotControl_getRegion_retun getRegion;
};
```

Interface Evolution

- Supported
  - Adding an operation
  - Removing an operation
  - Reordering operations
  - Reordering Base classes
  - Adding a base class
  - Removing a base class
  - Duck typing
- Hierarchy evolution is supported only when the entire hierarchy uses the same topic name.
  - Use `@RequestTopic`, `@ReplyTopic` annotations



Interface Evolution and XTypes

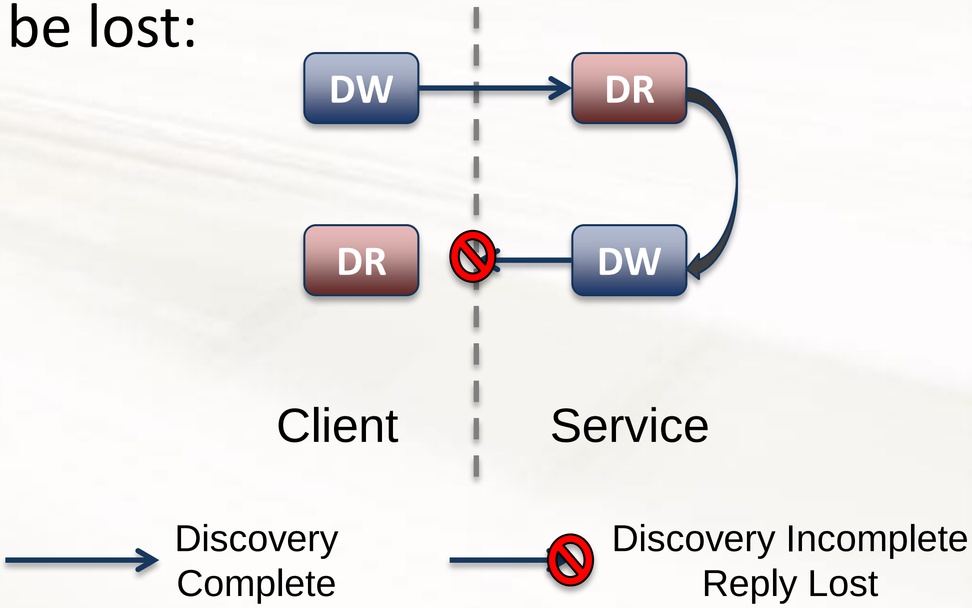
- XTypes takes care of evolution
  - The assignability rules and `@md5id` do the right thing
- When an operation isn't implemented by a service
  - The target type will not contain an operation that is present in the source.
  - All member pointers will be `NULL` in an `@choice` structure
    - Raise NOT_SUPPORTED system exception
- If evolution is not the intent, use a different topic
  - This is the default
  - Note: By default the topic name for each interface is different

Operation Evolution

- Supported
  - Changing the type of a parameter including the return value (except to/from `void`)
  - Adding and removing an exception
    - Each requester must be able to handle `UNEXPECTED_EXCEPTION`
    - Example, an old service may throw an exception that new client does not understand
- Not Supported (by default)
  - Changing the name of parameters
  - Adding/removing parameters
  - Reordering parameters
  - Adding/removing return type
- Semantics enforced using XTypes `@Final` annotation
  - Note: *Call* and *Out* structures are `@Final` by default
  - `@Final` implies member names, id, optionality, and order must match and types must be strongly assignable
- Possible to override the default using `@MutableOperation` annotation
  - `@Extensibility(MUTABLE_EXTENSIBILITY)` instead of `@Final`
  - All parameters become `@Optional` in the *Call* and *Out* structures
  - Will likely affect language binding to reflect *Optionality*

Service Discovery

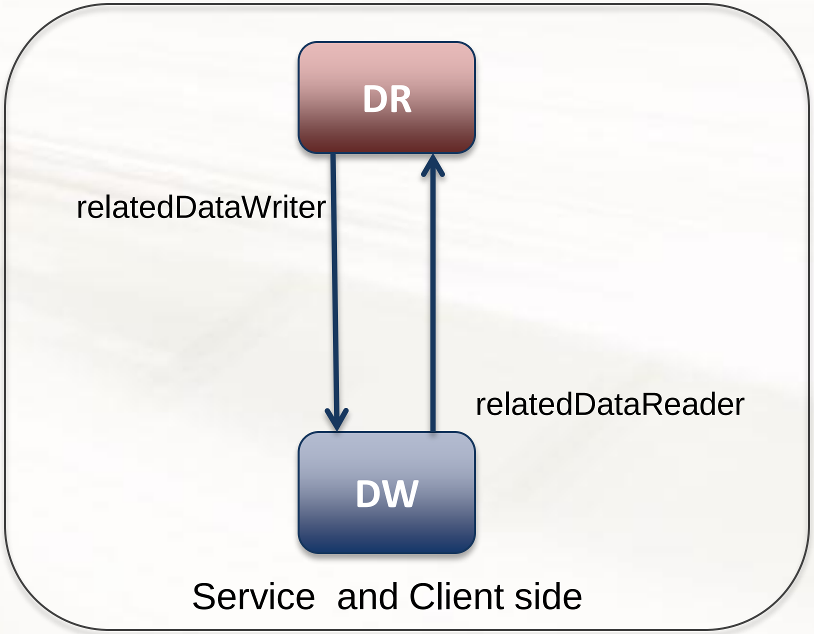
- Uses standard DDS discovery for client and service DataReaders/DataWriters
- Additionally specifies ordering constraints to avoid lost replies during transient discovery states
  - Ensure complete discovery of “Reply” topic entities before “Request” topic entities
- Example of why replies may be lost:



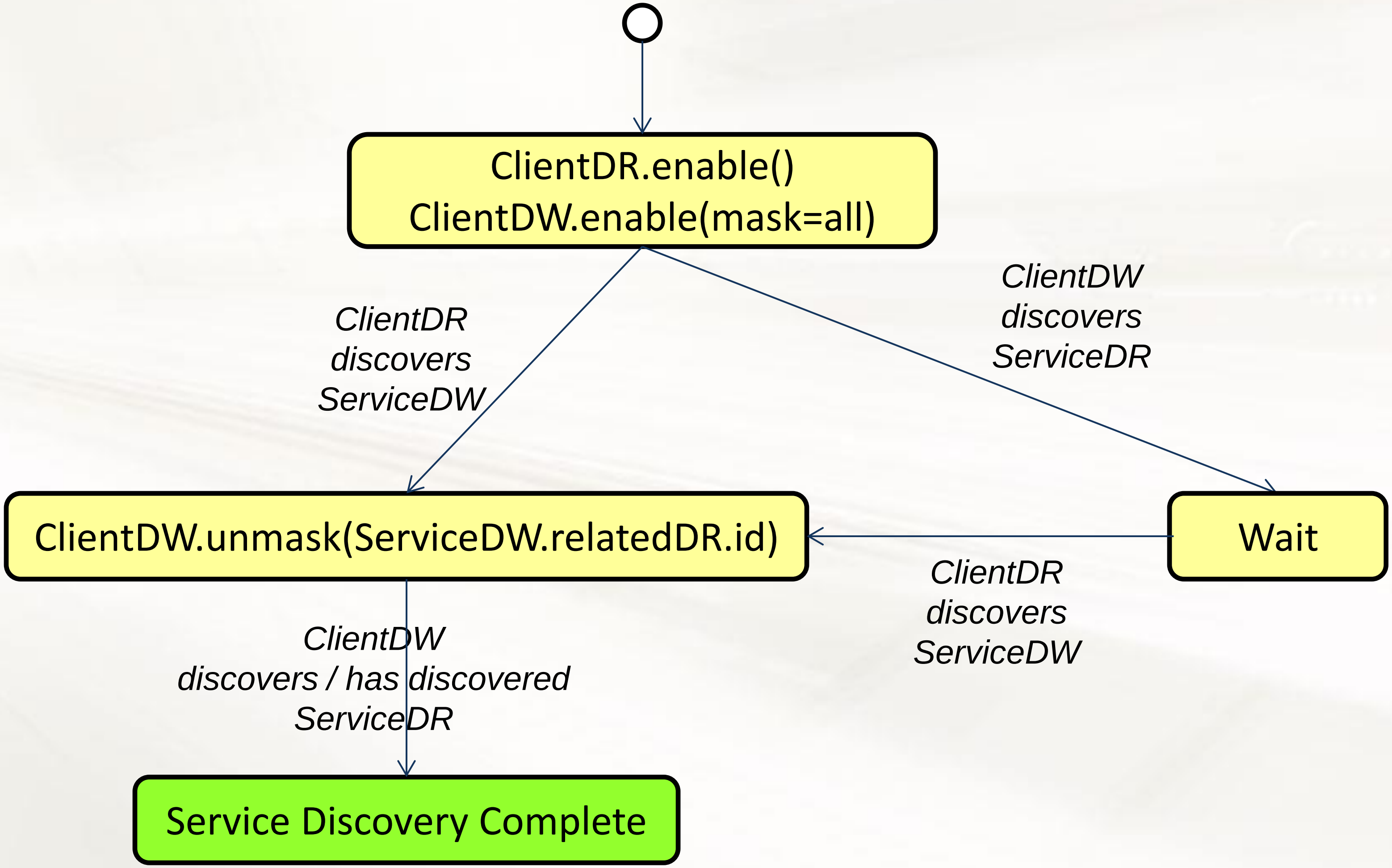
Service Discovery

- Service DataReader and DataWriter refers each other using globally unique IDs
  - Extended Publication and Subscription built-in topic data
- `Instance_name` should be unique but not enforced
  - Used only at the Service-side

```
struct PublicationBuiltinTopicData {
  //...
  BuiltinTopicKey_t relatedDataReader;
  string instance_name;
  //...
}


struct SubscriptionBuiltinTopicData {
  //...
  BuiltinTopicKey_t relatedDataReader;
  string instance_name;
  //...
}
```



Service Discovery



ClientDR.enable()
ClientDW.enable(mask=all)

ClientDR discovers ServiceDW

ClientDW discovers ServiceDR

ClientDW.unmask(ServiceDW.relatedDR.id)

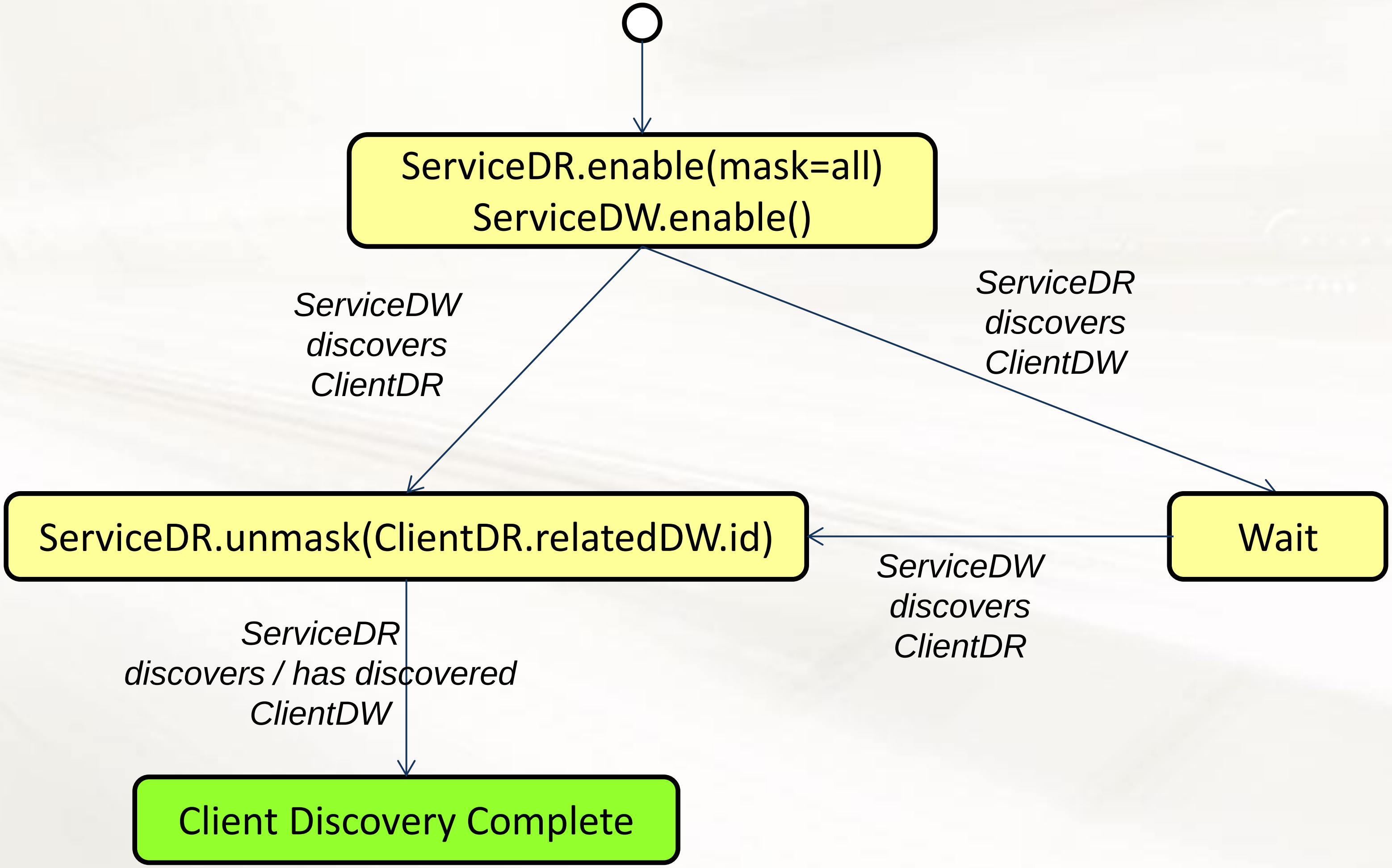
Wait

ClientDR discovers ServiceDW

ClientDW discovers / has discovered ServiceDR

Service Discovery Complete

Client Discovery



ServiceDR.enable(mask=all)
ServiceDW.enable()

ServiceDW discovers ClientDR

ServiceDR discovers ClientDW

ServiceDR.unmask(ClientDR.relatedDW.id)

Wait

ServiceDW discovers ClientDR

ServiceDR discovers / has discovered ClientDW

Client Discovery Complete

Request/Reply Language Bindings

- Request/Reply semantics
  - Typed API akin to DDS `read/write`
  - Higher-level entities to simplify programming
    - `Requester<Treq, TRep> , Service<TReq, TRep>`
  - Programmer must create the request and reply samples at the client and service side respectively
  - Return value is just a special member (`_return`)
  - User-defined exceptions are just members
  - System exceptions are captured with a special exception code field

Request/Reply style language binding in C++

- Requester

```
template <class TReq, class TRep>
class Requester {
public:
  // Creates a Requester with the minimum set of parameters.
  Requester (DomainParticipant *participant, const std::string &service_name);

  // Creates a Requester with parameters.
  Requester (const RequesterParams &params);

  void call(TRep &, const TReq &, Duration);

  void call(Sample<TRep> &, const TReq &, Duration);

  LoanedSamples<TRep> call(const TReq &, Duration);

  dds::future<Sample<TRep>> call_async(const TReq &);

  void call_oneway (const TReq &);

  bool bind(const std::string & instance_name);

  bool unbind();

  bool is_bound();

  void get_service_info(std::string & service_name, std::string & instance_name);
};
```

Request/Reply style language binding in C++

- Service

```
template <class TReq, class TRep>
class Service {
public:
  // Creates a Service with the minimum set of parameters.
  Service (DomainParticipant *participant,
           const std::string &service_name,
           const std::string & instance_name);
  // Creates a Service with parameters.
  Service(const ServiceParams &params);
  // blocking take
  void receive(TReq &, RequestIdentity &, Duration);
  // blocking take with sampleinfo
  void receive(Sample<TReq> &, RequestIdentity &, Duration);
  // blocking take
  // SampleInfo will contain the request identity.
  LoanedSamples<TReq> receive(Duration);
  bool wait(Duration);
  // non-blocking take
  bool take_request(TReq &, RequestIdentity &);
  // non-blocking take with sampleinfo
  bool take_request(Sample<TReq> &, RequestIdentity &);
  // read-blocking take
  bool read_request(TReq &, RequestIdentity &);
  // non-blocking take with sampleinfo
  bool read_request(Sample<TReq> &, RequestIdentity &);
  bool reply(const TRep &, const RequestIdentity &);
};
```

Request/Reply style language binding in C++

- Listeners

```
template <class TReq, class TRep>
class ServiceListener {
public:
  virtual TRep * process_request(const Sample<TReq> &, const RequestIdentity &) = 0;
  ~ServiceListener();
};


template <class TReq, class TRep>
class AsyncServiceListener {
public:
  virtual void process_request(Service<TReq, TRep> &) = 0;
  ~AsyncServiceListener();
};
```

Consuming/Implementing DDS Services using Function-Call Syntax

- Optional conformance point
- A purely local concern and not necessary for interoperability
- Built on top of request-reply style bindings (at least conceptually)
- The specification describes PIM for `ServantBase`, `ServiceProxy`, and `RPCRuntime`

Function-call Style C++ Bindings

- Use IDL to C++11 mapping modulo C++11-only stuff (e.g., rvalue references, some traits)
- Generate “stub” classes that proxy the remote service
- Generate “skeleton” classes to help implement the service interface using callbacks
- API
  - `ServantBase`: base class of all services implementation
  - `ServiceProxy<T>`: Handle to invoke services (for clients)
    - `T` is the C++ abstract base class of the interface defined in IDL
  - `RPCRuntime`: Bootstrap class to create services and client-side handles to the services

Example: Function-call Style C++

IDL

```
module Bank {

struct Address {
  String line1;
  String line2;
}
exception NotEnoughFunds {}

@DDSService
interface Account
{
  double  balance();
  boolean deposit(double amount);
  boolean withdraw(double amount)
      raises (NotEnoughFunds);
}

@DDSService
interface SavingsAccount : Account
{
  double interest_rate();
}

@DDSService
interface CheckingAccount : Account
{
  void order_checks(long howmany,
                    Address mail);
}

}
```

Client

```
int main(void)
{
  // Creating a client
  ServiceProxy<Bank::CheckingAccount> cap =
      rpcruntime.create_client<Bank::CheckingAccount>
      (“SomeBankService”);
  cap.balance();
  cap.bind(“BankInstance”);
  cap.order_checks(50, some_address);
}
```

Service

```
namespace mybank {

// Service implemenatation
class MyCheckingAccountImpl : public
dds::rpc::ServiceImpl<Bank::CheckingAccount>
{
public:
  virtual double  balance() {
    // my implementation body
  }
  virtual bool deposit(double amount) {
    // my implementation body
  }
  virtual bool withdraw(double amount) {
    // my implementation body
  }
    virtual void order_checks() {
    // my implementation body
  }
};
} // namespace mybank
```

Example: Function-call Style C++

Common

```
namespace Bank {
struct Address {
  std::string line1; // notional
  std::string line2; // notional
};

// Generated by IDLGEN from Bank.idl
class Account {
public:
  virtual double balance()=0;
  virtual bool deposit(double amount)=0;
  virtual bool withdraw(double amount)=0;
};

// Generated by IDLGEN from Bank.idl
class SavingsAccount : public virtual Account {
public:
  virtual double interest_rate()=0;
};

// Generated by IDLGEN from Bank.idl
class CheckingAccount : public virtual Account {
public:
  virtual void order_checks(int howmany,
                            const Address &)=0;
};

} // namespace Bank
```

Example: Function-call Style C++

Client-side Stubs

```
namespace dds {
  namespace rpc {
// Generated by IDLGEN from Bank.idl
template <>
class ServiceProxy<Bank::Account>
  : public ServiceProxyBase
{
public:
  typedef Bank::Account service_type;
  ServiceProxy(const ServiceProxy &);

  virtual double balance();
  virtual bool deposit(double amount);
  virtual bool withdraw(double amount);
};

// Generated by IDLGEN from Bank.idl
template <>
class ServiceProxy<Bank::CheckingAccount>
  : public virtual ServiceProxy<Bank::Account>
{
public:
  typedef Bank::CheckingAccount service_type;
  ServiceProxy(const ServiceProxy &);

  virtual void order_checks(int howmany, const
Bank::Address &);
};
```

```
// Generated by IDLGEN from Bank.idl
template <>
class ServiceProxy<Bank::SavingsAccount>
  : public virtual ServiceProxy<Bank::Account>
{
public:
  typedef Bank::SavingsAccount service_type;
  ServiceProxy(const ServiceProxy &);

  virtual double interest_rate();
};

} // namespace rpc
} // namespace dds
```

Example: Function-call Style C++

Service-side Skeleton

```
namespace rpc { namespace dds {
// Generated by IDLGEN from Bank.idl
template <>
class ServiceImpl<Bank::Account>
  : public virtual Bank::Account,
    public virtual ServantBase
{
public:
  typedef Bank::Account interface_type;

  virtual double  balance() {
    // empty implementation body
  }
  virtual bool deposit(double amount) {
    // empty implementation body
  }
  virtual bool withdraw(double amount) {
    // empty implementation body
  }
};

template <>
class ServiceImpl<Bank::SavingsAccount>
  : public virtual Bank::SavingsAccount,
    public virtual ServiceImpl<Bank::Account>,
    public virtual ServantBase
{
public:
  typedef Bank::SavingsAccount interface_type;

  virtual double interest_rate() {
    // empty implementation body
  }
};
```

```
template <>
class ServiceImpl<Bank::CheckingAccount>
  : public virtual Bank::CheckingAccount,
    public virtual ServiceImpl<Bank::Account>,
    public virtual ServantBase
{
public:
  typedef Bank::CheckingAccount interface_type;

  virtual void order_checks(int howmany,
                            Address address) {
    // empty implementation body
  }
};

} // namespace rpc
} // namespace dds
```

Stub and Skeleton helpers (base)

```
class ServantBase {
public:
  DomainParticipant get_domain_participant() const;
  std::string get_service_name() const;
  std::string get_instance_name() const;
  DataReaderQos get_request_datareader_qos() const;
  DataWriterQos get_reply_datawriter_qos() const;
};
```

```
class ServiceProxyBase
{
public:
  ServiceProxyBase(const ServiceProxyBase &);

  bool is_bound();
  bool get_service_info(std::string & service_name,
                        std::string &
bound_instance);

  std::string get_service_name();
  void bind(const std::string & instance_name);
  void unbind();
};

template <class T>
class ServiceProxy; // only a declaration
```

QoS Mapping

- Default strict reliable (request and reply)
  - RELIABLE reliability
  - KEEP_ALL history
  - VOLATILE durability
- Can be changed by the user

Thank you!