A User Level View of Babel RMI.

Jim Leek  
Gary Kumfert  
Tom Epperly  
Tamara Dahlgren

UCRL-PRES-218401
Contents

Goals

User overview
- Register a protocol
- Built in Functions
- Taste the Difference
- General UI Changes

Language Specifics
- Language examples

The FUTURE
- Non Blocking/One way calls
- Structs
- Other protocols
The Goals

- **Transparency**
  
  interchangeability with classic Babel code
  
  - Mostly successful
    
    - Very minor changes required in client code
    
    - No change required in server code

- **Flexibility**

  allow users to use a variety of protocols
  
  - Totally successful
    
    - any protocol that implements the Babel RMI API, OK!
Register a protocol

The first thing a client need to do to use RMI is to add a protocol to use

For example:

```java
sidl.rmi.ProtocolFactory.addProtocol("simhandle","sidlx.rmi.SimHandle")
```

This registers a “short name” to be used in URLs

```
simhandle://faraway.com:9999
```
New builtin functions

Simple Builtins

bool _IsLocal() / bool _isRemote()
  - Returns true if the object is local/remote

string _getURL()
  - Returns the URL of the object
  • If the object is local, requires a local ORB
New builtin functions
The heart of RMI

void _exec(string name,
           Deserializer inArgs,
           Serializer outArgs)

- Method dispatch by name.
- Passes args by serializer
RMI/Classic Differences
Remote Creation

Concrete objects can be remotely created with:

_create[Remote](string URL)

- Creates an object on the server given by the URL.
- The URL is protocol specific.
- Example URL: simhandle://foo.com:9999/1000
Objects that exist on a remote server can be connected to with:

_connect(string URL)

- The URL must include a object ID string
- Can connect as either an object and an interface
- Example URL simhandle://foo.com:9999/1000
RMI/Classic Differences
Passing objects/arrays remotely

foo.Bar method(foo.Quux x)
- Will pass objects by reference

copy foo.Bar method(copy foo.Quux x)
- Will serialize objects
  - The objects must implement sidl.rmi.Serializable

Arrays are always passed by serialization
RMI/Classic Differences
Passing local objects to remote servers

Passing an local object remotely by reference requires a local ORB

url = "simhandle://localhost:"+port;
orb = sidl.rmi.SimpleOrb._create();
orb.init(url, 1);
long tid = orb.run();
sidl.rmi.ServerInfo si = orb;
sidl.rmi.ServerRegistry.registerServer(si);
General UI Changes

Exceptions

Any remote call may throw an exception, so all calls now throw RuntimeException

- New RuntimeException includes:
  - NetworkException
  - MemoryException
  - Pre/Post Exception
  - IOException
As of Babel 0.11, cast addrefs the object being cast. Let's see why RMI needs it.

```java
package example Version 0.1 {
    interface Foo {}
    class Bar implements-all Foo {}
    class Quux extends Bar {}
}
```
General UI Changes
Cast _addrefs

Example of RMI object structure:
General UI Changes
Cast _addrefs

How did we get this funky construct?

![Diagram](image-url)
General UI Changes
Cast _addrefs

First we remotely connect Quux as a Foo
- quux may have been passed remotely as a Foo
- Or, example.Foo._connect(quuxURL) was called
General UI Changes
Cast _addrefs

Cast Foo to a Bar. We need a new stub.

2 things could happen.

1) We destroy the old stub, in which case foo.deleteRef() will seg fault
General UI Changes
Cast _addrefs

Or

2) We keep both stubs, and addref. Now the user must deleteRef both stubs

We chose option 2
Contents

Goals

User overview
- Add a protocol
- Built in Functions
- RMI/Classic Differences
- General UI Changes

Language Specifics
- Language examples

The FUTURE
- Non Blocking/One way calls
- Structs
- Other protocols

Annyoung!
Language Specfics

C

Each language has pretty much the same interface, but there are small differences.

foo_Bar__createRemote(URL, exception);
foo_Bar__connect(URL, exception);
foo_Bar__isLocal(obj, exception);
foo_Bar__getURL(obj, exception);

foo_Bar__exec(obj, inArgs, outArgs, exception);
Language Specifics
UCxx

foo::Bar::_create(URL);
foo::Bar::_connect(URL);
obj._isLocal();
obj._getURL();
obj._exec(inArgs, outArgs, exception);
Language Specifics

F77

call foo_Bar__createRemote_f(obj, URL, ex);
call foo_Bar__connect(obj, URL, ex);
call foo_Bar__isLocal(obj, isloc, ex);
call foo_Bar__getURL(obj, isret, ex);
call foo_Bar__exec(obj, inArgs, outArgs, ex);
call new(obj, URL, ex);
call connect(obj, URL, ex);
call isLocal(obj, isloc, ex)
call getURL(obj, isret, ex)
call exec(obj, inArgs, outArgs, exception);
new foo.Bar(URL);
foo.Bar._connect(URL);
obj._isLocal()
obj._getURL()
obj._exec(obj, inArgs, outArgs, exception);
foo.Bar.Bar(url = "URL");
foo.Bar._connect(URL);
obj._isLocal()
obj._getURL()
obj._exec(obj, inArgs, outArgs, exception);
Contents

Goals

User overview
- Add a protocol
- Built in Functions
- RMI/Classic Differences
- General UI Changes

Language Specifics
- Language examples

The FUTURE
- Non Blocking/One way calls
- Other protocols
- Structs
The FUTURE in 0.11.2

Publishing Objects
- Give your local object a specific object ID

Non-blocking and Oneway
- Although no protocol that supports it actually exists yet...
- Oneway looks just like a blocking call
- Non-blocking functions are of the form:
  - `sidl.rmi.Ticket obj.foo_send([inargs])`
  - `retval obj.foo_recv(sidl.rmi.Ticket,[outargs])`
New protocols

- A number of protocols are under development
  - SARS
    - Non-blocking high performance computing
  - BXSA
    - Scientific Binary XML
  - RMIX
    - Part of MOCCA
  - Tech-X
    - CORBA compatible
  - Psuedo-Protocol
    - Fake protocol for inprocess _exec use
Always useful for sending clean remote messages, structs!

- Gary Kumfert is prototyping this now
- I have no idea when this will arrive
  - (Not 0.11.2 in any case.)
Conclusion

In conclusion, you should use Babel RMI for all your remoteable needs.