Distributed Systems and Security

Further RMI

Graham White

Electronic Engineering and Computer Science
Queen Mary University of London

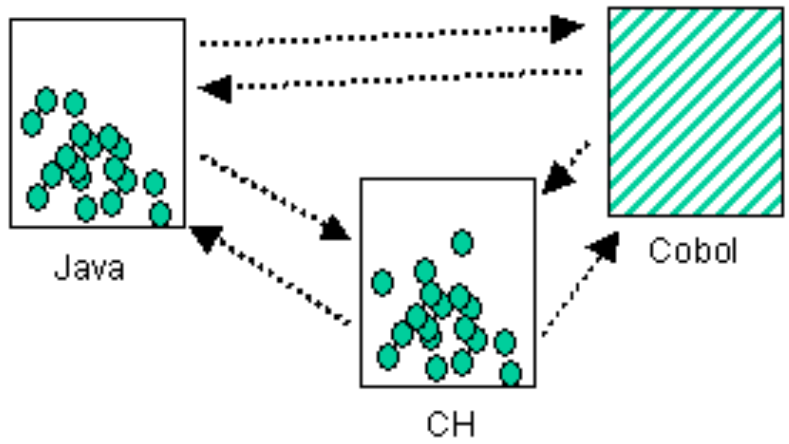
January 2016

Where we are

1 RMI in Multiple-Language Environments

RMI in Multiple-Language Environments

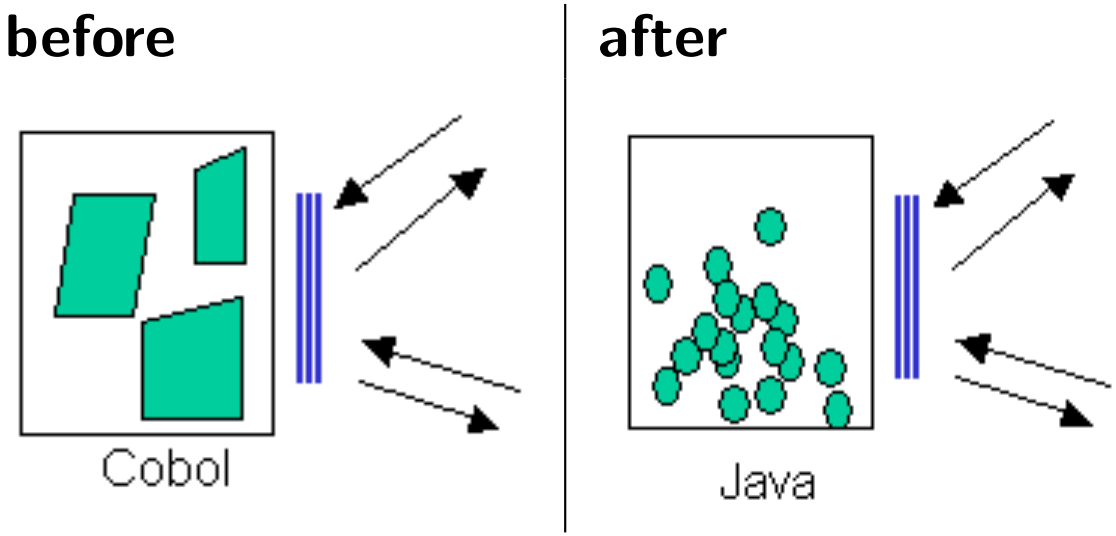
Distributed objects could, in principle, be written in *any* language, as in the picture:



- Can we get these objects to communicate with each other?
- If so, how can we specify interfaces of such objects independently of specific languages?

The Sofware Lifecycle 1

Such multi-language systems are important, because of what happens when we update software:



In this scenario, we update the implementation of an object (or group of objects), but the interface (i.e. the specification) of these objects does not change

The Software Lifecycle 2

- The previous slide showed an optimistic version of the software lifecycle, in which software did get updated
- In practice, a great deal of sofware *never* gets updated, because of
  1. lack of resources
  2. if the old software actually works, there is a risk that the new software may not work, or may work badly
- Consequently, a great deal of software remains in place, and new systems are implemented around it
- These new systems must communicate with the old ones
- RMI is a good way of handling this, especially since it can make non-OO software look object-oriented from the outside

Zombie Languages

- A great deal of very old technology is kept in being by such means.
- Very old programming languages are in a sort of zombie-like state, in which they are kept going only because of important software written in them
- For example, a lot of banks in 2000 still had software written in COBOL, which needed a lot of attention because of the millenium bug
- Remember: "The use of COBOL cripples the mind; its teaching should, therefore, be regarded as a criminal offense" (Dijkstra)
- Thus CORBA has support for COBOL and PL/1 (which are definitely zombie languages), for Tk (which may be a zombie language) and for Perl (which ought to be)

CORBA Summary

CORBA offers a language-independent RMI framework based on

IDL (the Interface Definition Language): a language independent interface definition language

CDR (the Common Data Representation): a common external data representation using

IOR the Interoperable Object Reference, which gives object refrences,

plus naming and other services which give the infrastructure for RMI

CORBA History

- OMG, the Object Management Group started the process in 1989
- the OMG it was a consortium of vendors (originally Data General, HP and Sun, but later far more)
- It was a rival to Microsoft's technologies of OLE and DCOM
- CORBA Version 1 emerged in 1991
- CORBA Version 2 in 1996.
- CORBA Version 3 some time later

Strategic Decisions

In order to know how CORBA does what it does, we need to know the answers to some strategic questions

- What object model does CORBA have, and how can this object model realise multiple-language RMI, even with non-OO target languages?
- How does IDL specify an interface, and how is the IDL interface translated into interfaces for specific languages?
- Linked with this is the question of how CORBA does marshalling and unmarshalling.

CORBA: the Object Model

This is as follows (see [CDK5 §8.3])

- In CORBA, "object" means "remote object interactions via an IDL interface"; that is, it is a concept of *local state* and *encapsulation*.
- There is no concept of a class such as Java has
- A non-object-oriented language, such as COBOL, can export a CORBA "object".
- Parameter-Passing:
  - All CORBA objects are passed by reference
  - All primitive data types and non-CORBA objects are passed by value
- Interaction is based on request-reply, with at-most-once semantics (you can specify maybe semantics by saying that a method is `oneway`

CORBA Interfaces

- Just like Java, CORBA has *interfaces* which provide specifications for objects
- These are especially important in a multi-language context.
  - Suppose we have an object `a` in language A, and a method `m` in language B, and we want to call `m` with parameter `a`
  - then, when we marshall and unmarshall `a` into language B, we want it to have the correct type for `m`
  - So if we have corresponding interfaces in both languages, this will work
  - and we achieve that by writing a single interface in IDL, and generating the interfaces in languages A and B from it

CORBA workflow

In CORBA, a remote object interface is written in language-independent Interface Definition Language (IDL), which is then compiled into the relevant target language or languages. So the workflow is as follows:

1. Write a remote interface in IDL
2. Compile it into (for example) Java using the Java tool `idlj`: there are a number of options for whether you generate client-side or server-side bindings, or both, and which Java CORBA framework you use.
3. Implement the interface with a java class

CORBA: the structure of the IDL

IDL has the following types

- primitive types, including `short`, `long`, `char`, `boolean` and `any`
- `sequence` (similar to `Vector` in Java), `string`, and `array`
- *Constructed types* such as `record`, `union` and `enumerated`
- `Objects` and interfaces, which are types for remote objects. We also have `valuetype`, which can contain (marshalled) objects from other languages.
- `exception` types.

CORBA IDL: An Example

The scenario should be familiar by now

```
interface Server {

  exception Exception{String reason};
  short deposit (short inc)
          raises (Exception);
  short withdraw (short dec)
          raises (Exception);
  short balance ()
          raises (Exception);

}
```

CORBA to Java

The above IDL is translated by `idlj` into:

```
public interface Server
        extends org.omg.corba.Object {

                int deposit (int inc)
                        throws testCORBAp
                int withdraw (int dec)
                        throws testCORBAp
                int balance ()
                        throws testCORBAp

}
```

CORBA: Method Invocation

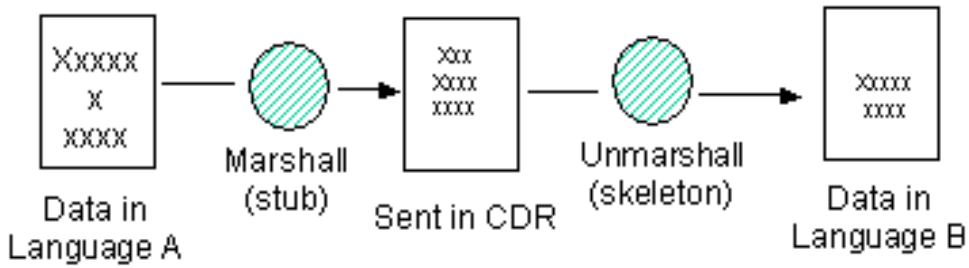
- CORBA uses stubs and skeletons, like Java RMI
- It uses an ORB (object request broker)
- ORB consists of the *ORB core*, which manages inter-site communication, and *Object Adaptor*, which manages remote object references and (local) invocation of remote objects.
- It also has *repositories* for locating running servers and interfaces

CORBA Remote Object Reference

- A remote object reference in CORBA is called an Interoperable Object Reference (IOR)
- it identifies the object by means of identifiers for the repository, for the object adaptor, and the name of the object given to it by the object adaptor.
- There are minor differences between
  1. a standard reference (called a *transient IOR*) for transient objects, and
  2. a *persistent IOR* for a persistent object, which corresponds to `Activatable` in Java RMI.

CORBA Common Data Representation

CORBA uses CDR (common data representation) for the uniform external (i.e. marshalled) data representation for RMIs.



Thus a CORBA implementation for a specific language only needs two mappings for communicating with arbitrary CORBA-enhanced sites

CORBA Services

CORBA provides various utility services, including

- Naming service (corresponds to `rmiregistry` in Java RMI]
- Interface repository (for dynamic interface discovery)
- Persistent object service (for persistent objects)
- Security enhancement (authentication/access control)
- Transaction service (commit/rollback)
- Event service

See [CDK §8.3.4] for details.

XML-based RPCs

The combination of XML and HTTP offers a simple framework for communicating among different applications over the network, because:

- XML offers a universal format for communicating structured data among different programming languages
- It has the same role as CDR in CORBA, but
  1. it tends to be more lightweight
  2. XML is used very extensively, so the software supporting it is very well tested and optimised
- So XML will probably be used more and more
- HTTP is an omnipresent Request-Reply protocol, and likewise has the advantages of wide use

Some Examples

XML-RPC and SOAP are two recent examples which use this combination.

- These XML technologies do not usually incorporate objects (though SOAP partly tries to do this), hence the name RPC.
- If XML-RPC/SOAP evolves into language independent RMI, basic elements of RMI mechanisms (especially treatment of remote objects) as found in Java RMI and CORBA become essential.
- There are also various technologies for serialising Java into XML (see the discussion here `http://stackoverflow.com/questions/35785/xml-serialization-in-java`)

Other Topics

Other topics in distributed objects:

- Event-based models.
  - In a *publish-subscribe* model, clients are *subscribed* to a *publisher* which sends *notifications* to subscribers when some event takes place.
  - *A server initiates interaction* ("push" rather than "pull")
  - Can be implemented using RMI and callbacks as well as asynchronous messages.
- Distributed objects with more complex behaviour including mobility ("agent" paradigm).
  - May use AI techniques.
  - Complex security issues.
  - Can use the existing object infrastructure.

Other Topics 2

- Distributed objects in ubiquitous computing and the Internet of Things
  - Millions of objects interacting in real-time. Many "objects" can be physical entities.
  - Use of the event-based paradigm.
  - Need of universal protocols (cf. Jini, UPnP).