CORBA

José M. Vidal

Mon Feb 23 10:58:17 EST 2004

This talk is based on:

- Douglas C. Schmidt[1]. CORBA Overview[2].
- Object Management Group. CORBA Overview[3] (Chaper 2) in CORBA 2.6 Formal Specification[4], 2002.
- Unknown tag=Steve Vinoski[5]. CORBA: Integrating Diverse Applications within Distributed Heterogeneous Environments.[6] *IEEE Communications,* (35)2, 1997.
- Dave Bartlett. OMG Interface Definition Language Definition Language[7], 2000.

1 Why CORBA?

- Distributed heterogeneous systems are the norm:
  1. There are always engineering tradeoffs with any technology.
  2. Consumers are not brand loyal. We use the best cheap stuff.
  3. Legacy systems seem to last forever.
- The Object Management Group (OMG[8]) was formed in 1989 to develop, adopt, and promote standards for the development of applications in these environments.
  - Largest software consortium in the world.
  - OMA (CORBA), UML, CWM, Model-Driven Architecture.
- OMG publishes the standards. There are many companies that implement them and sell ORBs.
- The OMG's goal is the "realization of a true commercial off-the-shelf software component marketplace". This vision drives the systems design.

2 Object Management Architecture

- The OMA is composed of object and reference models.
- **Object Model**: describes an object: "an encapsulated entity with a distinct immutable identity whose services can be accessed only through well-defined *interfaces*. Clients issue requests to perform services on their behalf. The implementation and location of each object are hidden from the client.
- **Reference Model**: describes architecture.

3 OMA Reference Model

Application Interface	Domain Interfaces	Common Facilities	Object Services
Object Request Broker			

- **Object Services** are domain-independent interfaces that are used by many distributed object programs.
  - Naming Service: find object based on name.
  - Trading service: find object based on properties.
- **Common Facilities** are end-user-oriented interfaces.
  - Distributed Document Component Facility.
- **Domain Interfaces** are application domain-oriented interfaces.
  - Product Data Management Enablers: for manufacturing domain.
- **Application Interfaces** are developed for specific applications. They are not standardized.

4 Object Frameworks

- An **object framework** is a domain-specific group of objects that interact to provide a customizable solution within that application domain.
- They are the "big architecture picture" which guides OMA development.
- Frameworks are composed of components. Each component implements a number of interfaces.
- The components communicate with each other in a peer-to-peer fashion.

5 CORBA

Client		Object Implementation
Dynamic Invocation	IDL Stub	IDL Skeleton Object Adapter
ORB Core		

- There is also an Interface Repository.

6 ORB Core

- The ORB hides from the client many things, including the following.
- **Object Location**
- **Object Implementation:** Programming language, operating system, hardware.
- **Object Execution State:** Active or inactive.
- **Object Communication Mechanism:** TCP/IP, shared memory, pipes, local call, etc.
  - An **object reference** is created when object is created. It always refers to the same object. It is immutable an opaque.
- **Object Creation:** There are three ways to get hold of an object:
  1. Client invokes a creation request on a factory object which returns an object reference.

2. Invoke a lookup service such as a Naming Service or a Trading Service which store existing object references.
3. Turn the reference into a string and back. These objects are called *stringified* and *destringified*.

– ORB provides a simple naming service which can store object references of more general naming services. `ORB.resolve_initial_reference("NameService")`.

7 OMG Interface Definition Language

- Because CORBA is language-independent a way to define interfaces had to be developed that is also language-independent.
- Interface in IDL gets automatically turned into code for your favorite programming language.
- IDL looks quite a bit like a C++ header file.

```
//OMG IDL
interface Factory {
  Object create();
};
```

- It supports `module`s, which are groups of `interface`s.
- It supports `exception`s.
- It supports `attribute`s, which are like data members.
- It supports many primitive data types.

7.1 IDL Types

- `long` (`signed` and `unsigned`)- 32-bit arithmetic types.
- `long long`(`signed` and `unsigned`)- 64-bit arithmetic types.
- `short`(`signed` and `unsigned`)- 16-bit arithmetic types.
- `float`, `double`, and `long`- IEEE 754-1985 floating point types.
- `char` and `wchar`- character and wide character types.
- `boolean`- Boolean type.
- `octet`- 8-bit value.
- `enum`- enumerated type.
- `any`- a tagged type that can hold a value of any OMG IDL type, including built-in types and user-defined types.
- `struct`- data aggregation construct, like in C.
- `union`- like in C.

7.2 IDL Template Types

- Similar to C++ templates.
- A template type is a type that takes an argument at declaration-time. The actual type is, therefore, only created at compile time.
- `string` and `wstring` can be bounded by providing a number argument:
  - `string<10>` defines a string type of maximum length 10.
- `sequence` is a dynamic-length linear container (like Java `Vector`) whose maximum length and element type can be specified in angle brackets (unlike in Java).
  - `sequence<Factory>` defines a sequence of factories.
  - `sequence<Factory,10>` only 10 factories allowed.
- `fixed`- a fixed-point decimal value with no more than 31 significant digits.
  - `fixed<5,2>` has a precision of 5 and scale of 2. e.g., 999.99

7.3 IDL Example

```
module EmployeeInfoServer {
  interface Employee;
  interface Department;

  exception EmployeeInfoException {
    string message;
  };

  interface Employee {
    unsigned long getId();
    Department getDepartment();
    float authorizeCommission(in float saleVolume)
      raises (EmployeeInfoException);
    attribute string name;
    attribute string ssn;
  };

  typedef sequence<Employee> EmployeeList;

  interface Department {
    unsigned long getId();
    attribute string name;
    EmployeeList employees();
  };
};
```

7.4 IDL Object Reference Types

- You can declare an IDL object reference by simply naming the desired interface type.

```
interface FactoryFinder {

  //define a sequence of Factory object references
  typedef sequence<Factory> FactorySeq;

  FactorySeq find_factories(in string interface_name);
};
```

7.5 IDL Interface Inheritance

- IDL supports interface inheritance.

```
interface Factory{
  Object create();
};

//Forware declaration of Spreadsheet interface
interface Spreadsheet;

//SpreadsheetFactory derives from Factory
interface SpreadsheetFactory : Factory {
  Spreadsheet create_spreadsheet();
};
```

- The `create` function is inherited from `Factory`.
- An object reference of a derived interface can be substituted anywhere object references from base interface are allowed.
- All interfaces are implicitly derived from the `Object` interface defined in the CORBA module.

7.6 IDL Language Mapping

- OMG has standardized language mappings.

OMG IDL Type	C++ Mapping Type
`long, short`	`long, short`
`float, double`	`float, double`
`enum`	`enum`
`char`	`char`
`boolean`	`boolean`
`octet`	`unsigned char`
`any`	`Any` class
`struct`	`struct`
`union`	class
`string`	`char*`
`wstring`	`wchar_t*`
`sequence`	class
`fixed`	`Fixed` template class
object reference	pointer or object
interface	class

- `module`s map to C++ `namespace`s.

- In C, since it does not have objects, objects are written as abstract data types.
- There is also an IDL to Java mapping[9]

8 Interface Repository

- Usually, applications use static knowledge of IDL types to compile.
- But, sometimes they need run-time knowledge (e.g., the interface changes and we do not want to recompile).
- The IR allows the OMG IDL type system to be accessed and written programmatically at runtime.
- Using the IR interface, applications can traverse an entire hierarchy of IDL information.
- Or, we can use `CORB::Object.get_interface()` which returns an `InterfaceDef` object. Since all objects inherit from `Object` they all define this function.

9 Stubs and Skeletons

- The IDL compiler generates client-side **stubs** and server-side **skeletons**.
- They are built into the application and have a priori knowledge of the IDL interfaces being invoked.
- Using stubs and skeletons to access CORBA object functions (dispatch) is often called **static invocation**.
- The stub works with the ORB to **marshal** the request. The receiving ORB unmarshals it.

10 Dynamic Invocation

- In addition to static invocation via stubs, CORBA also supports dynamic invocation via two interfaces.
- **Dynamic Invocation Interface** supports dynamic client request invocation.
- **Dynamic Skeleton Interface** provides dynamic dispatch to objects.
- They can be viewed as "generic stub" and "generic skeleton", respectively.

10.1 Dynamic Invocation Interface

- Using it, a client can invoke requests on any object without having compile-time knowledge of the object's interface.
- How? `CORBA::Object` interface implements `Request create_request()` function.
  1. Get a request pseudo-object.
  2. Set argument values on it.
  3. Invoke it.
- The invocation can be:
  - **Synchronous:** Block waiting for the response.

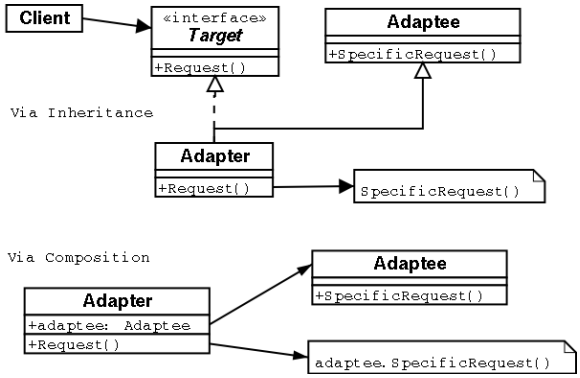
  - **Deferred Synchronous:** The client makes the call and continues processing, later collects the response.
  - **Oneway Invocation:** Make the request. There is no response.
- Using the DII is costly (time) since the ORB usually accesses the IR.

10.2 Dynamic Skeleton Interface

- The DSI allows servers to be written without having skeletons for the objects being invoked compiled statically into the program.
- A feature not often used.

11 Adapters

- The adapter design pattern[10] looks like this:



11.1 Object Adapter

- It is the glue between CORBA object implementations and the ORB itself. It is responsible for many things.
- **Object registration:** supplies operations that allow programming language entities to be registered as implementations for CORBA objects.
- **Object reference generation:** generates them.
- **Server process activations:** start it if need it.
- **Object activation:** activate them if they are not already active.
- **Request de-multiplexing:** cooperate with ORB to ensure that requests can be received over multiple connection.
- **Object upcalls:** dispatch requests to registered objects.

12 Inter-ORB Protocols

- Before CORBA 2.0 different vendors ORBs could not talk to each other.
- GIOP- General Inter-ORB protocol specifies transfer syntax and message formats for any connection-oriented transport.
- IIOP- Internet Inter-ORB Protocol specifies how GIOP is built over TCP/IP transports.
- IIOP is mandatory for 2.0 and later ORBs.

Notes

[1] http://www.cs.wustl.edu/ schmidt/
[2] http://www.cs.wustl.edu/ schmidt/corba-overview.html
[3] http://www.omg.org/cgi-bin/doc?formal/01-12-40
[4] http://www.omg.org/technology/documents/formal/corba_2.htm
[5] http://www.iona.com/hyplan/vinoski
[6] http://jmvidal.cse.sc.edu/library/vinoski97a.pdf
[7] http://www-106.ibm.com/developerworks/webservices/library/co-corbajct3.html
[8] http://www.omg.org
[9] http://www.omg.org/technology/documents/formal/java_language_mapping_to_omg_idl.htm
[10] http://www.wikipedia.org/wiki/Adapter_pattern

This talk is available at `http://jmvidal.cse.sc.edu/talks/corba`