Common Object Request Broker Architecture

An overview of the OMG way in Component Software

Matteo Matteucci
matteucc@elet.polimi.it
Component Programming

• Definition (ECOOP 96)
  – Standard interface (contractual)
  – Explicit context dependencies
  – Independent deployment and composition

• Motivations
  – User:
    • Integration of component independently developed
    • Reuse of common features
    • Wrapping of user applications (legacy)
  – Developer:
    • Modularization of application
    • Incremental development
    • Reuse

Object Management Group

• OMG (Object Management Group)
  – No-profit consortium of several hundred members founded in 1989 to promote the development and diffusion of object-oriented software
  – Definition of an extensible infrastructure to support services:
    • Interoperability
    • Security
    • Concurrency
    • ...

• Object-oriented and Component Programming
  – Objects or components as production, distribution and management units
  – The objects became also deployment units
  – Objects interoperate using a software bus sending requests and results in a distributed system
OMA and CORBA

- The problem to face:
  "How can distributed object-oriented systems implemented in different languages and running on different platforms interact?"

- The OMG solution
  - OMA (1992)
    - Reference architecture to guarantee component reuse and interoperability
  - CORBA (1995)
    - Open interconnection of languages implementations and platforms
    - Bus software for component wiring
    - Interface definition language

CORBA

- Reference architecture to ORB implementation
  - Specification of architecture
  - No implementation defined

- Middleware Object Oriented
  - Marshaling
  - Implementation hiding
  - References management
  - Localization hiding
  - OO RPC

- Object Request Broker
  - Communication infrastructure
  - Platform independent primitives
  - Implementation independent primitives
Object Request Broker

- Client Application
  - Written in a language with a binding to IDL
  - Not necessarily an object oriented language (es. VB, 4GL, …)
  - Client does not know the location of an object implementation
  - An object can act as a client and as a server at the same time
  - The client can call a method of a remote object by its reference and knowing its interface

- Object Reference
  - Identify univocally an object in distributed system based on an ORB
  - CORBA specifies the standard of IOR (Interoperable Object Reference) but not its implementation
  - How to map Object Reference is defined by the binding of IDL to a specific language (pointers in C++, object references in SmallTalk, …)
ORB: ORB Interface

- API to services implemented by the ORB used by the client and by the server
  - Allow operations on Object References
    - `get_interface`: to obtain the description of the interface of an object
    - `get_implementation`: to obtain the description of the object implementation of an object
    - `is_nil`: to verify if the Object Reference really identify an object

ORB: Repositories

- Interface Repository
  - Contains the description of the interfaces defined in the system
  - Can be queried to obtain information about interfaces, methods and parameters
  - Information can be used to dynamically compose the requests (DII)
  - Its services are described using IDL

- Implementation Repository
  - Contains information on the object implementation, object instances, identifiers and management information
ORB: IDL Stubs and DII (1)

• Stub (static)
  – Represents operations of the server that the client can invoke using its implementation language
  – It is compiled from the IDL description of the interface to the specific implementation language
  – From client point of view it is a local procedure call
  – The Stub codifies the parameters (marshaling), de-codifies the results and re-raises exception from the server

• Dynamic Invocation Interface
  – The client may know the interface of an object during the execution
  – The client can built dynamically the request for a service
  – Server cannot distinguish if the request came from a static stub or a dynamic invocation

ORB: IDL Stubs and DII (2)

• Stubs (static)
  – Can be used only if methods are known a-priori (compile time)
  – They are totally transparent to the programmer
  – They allow static type checking
  – They are efficient

• Dynamic Invocation Interface
  – They do not require to know objects’ methods (interface) before execution
  – Allow to write generic code
ORB: IDL Skeletons and DSI

- **Skeleton (static)**
  - Analogous to the Stub but from the server side
  - It is compiled from the IDL description of the interface to the specific implementation language
  - The Skeleton de-codifies the parameters (marshaling) passing them to the invoked method
  - Once executed the method, codifies the results and exceptions sending them to the client

- **Dynamic Skeleton Interface**
  - The server may know the interface of an object during the execution
  - The server receive the request also if it has not yet compiled the IDL interface
  - Client cannot distinguish if the result came from a static skeleton or a dynamic skeleton

ORB: Object Adapter (1)

- **Layer between the ORB and the object implementation**
  - Supply common operations on the objects (IDL)
    - Instances of new objects
    - Management of Object References
    - Routing of requests
    - Registration in the Implementation Repository

- **CORBA defines BOA a standard object adapter (Basic Object Adapter)**
ORB: Object Adapter (2)

- **Examples of adapters**
  - Basic Object Adapter (BOA): generation and management of object references, invocation of methods, request delivery, registration …
  - Library Object Adapter (LOA): can substitute BOA if client and server belong to the same process (ORB == Library), few services but optimized for the particular use (in-process)
  - Object-Oriented Database Adapter (OODA): implements a connection to an object-oriented database

- **Different types of BOA**
  - Shared server: actsives the server when requested and manages more objects
  - Unshared server: a different server for each object when requested
  - Server-per-method: a server for each request
  - Persistent server: a shared sever not managed by BOA

ORB: ORB Interoperability

- **CORBA 2.0 defines a standard to connect ORB on different (sub)systems by different developer**
  - General Inter-ORB Protocol (GIOP)
    - Defines the exchange format
    - Defines a common format for the data (Common Data Representation)
  - Internet Inter-ORB Protocol (IIOP)
    - Specifies messages of GIOP on TCP/IP
    - Allow to connect different ORBs using Internet
  - Environment Specific Inter-ORB Protocol (ESIOP)
    - Allow the use of different transport protocols
Interface Definition Language

• Separation among interface and implementation
• The language to define interfaces of OMA components
  – To use services implemented by an object a client application has to know
    its interface
  – Used to define objects in CORBA-compliant applications
• An interface specified by IDL can be implemented by different languages
  – CORBA specifies mapping to several languages

IDL: Main features

• Main features
  – Specification language (not programming)
  – Independent from implementation language
  – Multiple Inheritance
  – Static type checking for interfaces
  – Allow static and dynamic use
  – C++ like syntax
  – IDL compiler support pre-processing (#include)
  – An IDL specific includes
    • types
    • constants
    • exceptions
    • interfaces
    • modules
IDL: Interfaces (1)

- An interface defines types, constants, exceptions, attributes and operation supported by a server objects
- Any interface has a name and can be defined from one or more existing interfaces (multiple inheritance)
  - Derived interfaces can add new elements or redefine existing ones
  - It is not possible to derive from interface defining the same attributes or operations
  - Ambiguities are resolved by operator::
- Objects implements an interface if implements all the operations (may be more than one interface)
- The object reference supplied by the ORB and mapped on the specific implementation language feature

IDL: Interfaces (2)

- An example:

  ```
  interface BASE
  {
    const int N;
    ...
  }
  
  interface Derived: Base
  {
    const int N;
    typedef ...
  }
  ```
IDL: Modules

- Modules implements modularization of programming languages (modules, namespaces, packages ...)
- An example:

```cpp
module Global {
    typedef ...;
    interface B ...
}
Global::B ...
```

IDL: Types

- Built-in types:
  - [unsigned]short, long, long long
  - float, double, long double
  - char
  - octet
  - string
  - boolean
- It is possible to define C++ like types
  - enum
  - struct, union
  - array
  - sequence
- An example:
  ```cpp
typedef sequence<sequence<long>> DblSeq;
```
IDL: Attributes

- It is possible to declare constants data
- Attributes are declared in the interfaces
  - `attribute` is the keyword to declare an attribute
  - `readonly` the client cannot modify the attribute
  - `const` also the server cannot modify the attribute
  - attribute are public, any other value is private

- An example:
  ```
  const int MAX = M * N + LEN;
  attribute float radius;
  readonly attribute position_t position;
  ```

IDL: Exceptions & Operations

- Exceptions can be declared as structures
  - Can be raised by operation called by the caller
  - CORBA defines a set of standard exceptions
- An operation specifies a function in a C-like style
  - The operation syntax is:
    ```c
    [oneway] <resultName> <opName> ([|in|out|inout] par1, ..)
    [raises(exception1, ..)][context(c1, ..)];
    ```
  - `oneway` specifies that the result is not used by the client
  - Overloading is non supported
- An example:
  ```c
  exception NotFound {string <N> what};
  void search(in Code what, out Item el)
  raises(NotFound);
  ```
Object Management Architecture

- OMA objects:
  - Encapsulate data and operations
  - Have a defined interface
  - Univocally identifiable
  - Execute services requested by a client (not necessary an object)

OMA: CORBA services (1)

- Objects Services form the base to distributed applications
  - Creation of distributed objects
  - Access to object’s methods and attributes
  - Security, transactions ...

- CORBA services
  - OMA specifies the interfaces of objects implementing the Object Services
  - OMA does not specify any implementation

- Object Services implementation is the base for any CORBA platform
OMA: CORBA services (2)

- Naming Service
- Event Service
- Life Cycle Service
- Relationship Service
- Persistent Object Service
- Transaction Service
- Concurrency Service
- Externalization Service
- Licensing Service
- Query Service
- Property Service
- Security Service
- Time Service
- Collection Service
- Trader Service

OMA: Frameworks

- CORBA facilities
  - Components that supply common application functionality
    - Distributed help
    - DBMS access
    - e-mail services
    - ...
  - OMA defines their interfaces non the implementation (not necessary)
- Domain Interfaces
  - Specific services for particular domain
    - Medics
    - Finance
    - Manufactory
    - e-commerce
    - Telecommunications
    - ...

Matteucc@elet.polimi.it
www.elet.polimi.it/~matteucc
Bibliography and Resources

• C. Zypersky. *Component Programming (beyond OOP).*

• http://www.omg.org/corba
• http://www.cetus-links.org
• http://www.infosys.tuwien.ac.at/Research/Corba/OMG/cover.htm