Object-Oriented Middleware for Distributed Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini *after Giovanni Rimassa*
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2011/2012
Outline

1. Middleware Overview
2. Object-Oriented Middleware
3. CORBA & OSGi
These Slides...

... are derived from a Presentation by Giovanni Rimassa, which we warmly thank.

Slides were made kindly available by the author.

- Every problem or mistake contained in these slides, however, should be attributed to the sole responsibility of the teacher of this course.
Outline

1. Middleware Overview
2. Object-Oriented Middleware
3. CORBA & OSGi
What is Middleware?

Traditional definition

- What is middleware?
  - The word suggests something belonging to the middle
  - But middle between what?

- The traditional middleware definition
  - The middleware lies in the middle between the Operating System and the applications

- The traditional definition stresses *vertical* layers
  - Applications on top of middleware on top of the OS
  - Middleware-to-application interfaces (*top interfaces*)
  - Middleware-to-OS interfaces (*bottom interfaces*)
Why Middleware?

Behind middleware

- Problems of today
  - Software development is hard
  - Experienced designers are rare (and costly)
  - Applications become more and more complex

- What can middleware help with?
  - Middleware is developed once for many applications
  - Higher quality designers can be afforded
  - Middleware can provide services to applications
  - Middleware abstracts away from the specific OS
A key feature of middleware is interoperability.
- Applications using the same middleware can interoperate.
- This is true of any common platform (e.g., OS file system).

But, many incompatible middleware systems exist.
- Applications on middleware A can work together.
- Applications on middleware B can work together, too.
- But, A-applications and B-applications cannot!

The Enterprise Application Integration (EAI) task.
- Emphasis on horizontal communication.
- Application-to-application and middleware-to-middleware.
Software development does not happen *in vacuum*

- Almost any software project must cope with past systems
- There is never time nor resources to start *from scratch*
- Legacy systems were built with their own approaches

System integration is the only way out

- Take what is already there and add features to it
- Try to add without modifying existing subsystem

First casualty: Conceptual Integrity

- The property of a system of being understandable and explainable through a coherent, limited set of concepts
Real systems are heterogeneous
- Piecemeal growth is a very troublesome path for software evolution
- Still, it is very popular – being asymptotically the most cost effective when development time goes to zero

Middleware technology is an integration technology
- Adopting a given middleware should ease both new application development and legacy integration
- To achieve integration while limiting conceptual drift, middleware tries to cast a model on heterogeneous applications.
Integration middleware

- Before: you have a total mess
  - A lot of systems, using different technologies
  - Ad-hoc interactions, irregular structure
  - Each piece must be described in its own reference frame

- Then: the Integration Middleware (IM) comes
  - A new, shiny model is supported by the IM
  - Existing systems are re-cast under the Model
  - New model-compliant software is developed

- After: you have the same total mess
  - But, no, now they are CORBA objects, or TuCSoN agents
Abstract vs. concrete middleware

- Abstract middleware: a common *model*
- Concrete middleware: a common *infrastructure*
- Example: Distributed Objects
  - Abstractly, any middleware modeling distributed systems as a collection of network reachable objects has the same model: OMG CORBA, Java RMI, MS DCOM, OSGi Architecture...
    - Actually, even at the abstract level there are differences...
  - Concrete implementations, instead, aim at actual interoperability, so they must handle much finer details
    - Until CORBA 2.0, two CORBA implementations from different vendors were not interoperable
    - OSGi easily provides you with specifications—technology not so easy to find
The role of standards

- Dealing with infrastructure, a key-issue is the so-called *network effect*. The value of a technology grows with the number of its adopters.
- Standardisation efforts become critical to build momentum around an infrastructure technology.
  - Large standard consortia are built, which gather several industries together (OMG, W3C, FIPA, OSGi).
  - Big industry players try to push their technology as de facto standards, or set up more open processes for them (Microsoft, IBM, Sun).
How to (re)present a middleware

- Presentation and analysis of the model underlying the middleware
  - What do they want your software to look like?
- Presentation and analysis of the infrastructure created by widespread use of the middleware
  - If they conquer the world, what kind of world will it be?
- Discussion of implementation issues at the platform and application level
  - What kind of code must I write to use this platform?
  - What kind of code must I write to build my own platform?
Distributed Objects

From OO to Distributed OO

- Distributed systems need quality software, and they are a difficult system domain
- OOP is a current software best practice
- Questions are
  - Can we apply OOP to Distributed Systems programming?
  - What changes and what stays the same?
- Distributed Objects apply the OO paradigm to Distributed Systems
  - Examples: CORBA, DCOM, Java RMI, JINI, EJB, OSGi
What is the fundamental concept of OOP?

- From the very name of object-oriented programming, could it be **The Object**?
- Definitely not—and *you should know this*!
- The fundamental concept of object-oriented programming is **The Class**!
Class: A definition

- A class is an **abstract data type**, with an associated **module** that implements it.
- Writing this as a conceptual equation *à la* Wirth,

\[
\text{Type} + \text{Module} = \text{Class}
\]
Modules vs. Types

- Modules and types look very different
  - Modules give structure to the implementation
  - Types specifies how each part can be used
- But they share the interface concept
  - In modules, the interface selects the *public* part
  - In types, the interface describes the allowed *operations* as well as their *properties*
- As a result, the interface is at the very core of the notion of class
The fundamental OOP computation mechanism:

```
res = obj.meth(par)
```
OOP Extensibility

**Subclassing**

Subclassing is the main OOP extension mechanism, and it is affected by the dual nature of classes

- Type + Module = Class
- Subtyping + Inheritance = Subclassing

**Subtyping** — a partial order on types

- A valid operation on a type is also valid on a subtype
- Liskov Substitutability Principle: If S is a subtype of T, then replacing objects of type T with objects of type S does not alter the properties of a program

**Inheritance** — a partial order on modules

- A module grants special access to its sub-modules
- Open/Closed Principle: An OO language must allow the creation of modules closed for use but open for extension
How to?

Q How can we extend OOP to a distributed system, preserving all its desirable properties?

A Just pretend the system is not distributed, and then do business as usual!

This is called *transparency*

- As crazy as it may seem, it works!
- Well, up to a point at least, but generally enough for a lot of applications.

Problems arise from failure management

- In reliable and fast networks, things run smooth...
- Whenever a failure comes from what we abstracted away – e.g., a network failure –, we are just plain dead
What is the fundamental concept of Distributed OOP?

- Could it be The Object

  or, again, The Class

- Clearly not

- The fundamental concept of distributed OOP is The Remote Interface!
Distributed OOP Mechanism

Remote Method Call

The fundamental Distributed OOP computation mechanism

\[ res = \text{obj}.\text{meth}(\text{par}) \]

- **Result**: Sent back
- **Target Object**: Encapsulates address and protocol
- **Parameter List**: Sent on the network
- **Access Operator**: Grants location transparency
The Distributed Objects communication model...

- ... is *implicit*
  - Transmission is implicit, everything happens through stubs
  - The stub turns an ordinary call into an Inter-Process Communication (IPC) mechanism
  - As a result, both local and remote calls are handled homogeneously—*location transparency*

- ... is *object-oriented*
  - Only *objects* exist, invoking operations on each other
  - Interaction is client/server with respect to the individual call—micro C/S, not necessarily macro C/S
  - Each call is attached to a specific target object: the result can depend on the target object state
  - Callers refer to objects through an object reference
Broker Architecture

Broker architectural pattern [Buschmann et al., 1996]

- Stock market metaphor
- Publish/subscribe scheme
- Extensibility, portability, interoperability
- A broker reduces communication channels from $N_c \times N_s$ to $N_c + N_s$
Proxy and Impl, Stub and Skeleton

ResType operation(ParType par) {
  // 1. Marshal parameter
  // 2. Send marshalled data to impl transport address
  // 3. Receive result from impl transport address
  // 4. Return Result
}

void dispatch() {
  while(active) {
    // 1. receive from the RemoteProxy
    // 2. Unmarshal received data
    // 3. Call operation on RemotelImpl
    // 4. Send back result
  }
}

ResType operation(ParType par) {
  // Execute the operation normally
}
Outline

1 Middleware Overview

2 Object-Oriented Middleware

3 CORBA & OSGi
Many thanks... 
... to Giovanni Rimassa for his slides
Many thanks…

… to Marcel Offermans for his slides
Conclusions

Summing Up

Object-oriented Middleware...

- ... provides a coherent framework for Distributed OOP, both conceptually and technologically
- ... extends OOP to Distributed Systems
- ... hides the complexity of programming DS
- ... is supported by open standards—such as OMG CORBA and OSGi
- ... promotes integration across OSs, networks and languages
- ... counts on a lot of free implementations available

Does it solve everything?

- Of course not.
- That is why why have a course on Multi-agent Systems, then!
Object-Oriented Middleware for Distributed Systems

Distributed Systems
Sistemi Distribuiti

Andrea Omicini *after* Giovanni Rimassa
andrea.omicini@unibo.it

Ingegneria Due
Alma Mater Studiorum—Università di Bologna a Cesena

Academic Year 2011/2012