WHITESTEIN
Technologies

From Distributed Objects to Multi-Agent Systems: Evolution of Middleware (1)

Giovanni Rimassa

Whitestein Technologies AG – (`gri@whitestein.com`)

Presentation Outline (1)

- Middleware Overview
  - What is Middleware
  - Why Middleware
  - Middleware and Models
  - Middleware Technologies and Standards
- Object Oriented Middleware
  - Mission: OOP for Distributed Systems
  - OOPrinciples
  - Bringing Objects to the Network
  - Overview of the CORBA Standard

Presentation Outline (2)

- Agent Oriented Middleware
  - Mission: Mainstreaming Agent Technology
  - What is an Agent?
  - Autonomy, Sociality and Other Agenthood Traits
  - Overview of the FIPA Standard
- JADE: A Concrete FIPA Implementation
  - Overview: The Software, the Project, the Community
  - JADE as a Runtime Support System
  - JADE as a Software Framework
  - JADE Internal Architecture

Middleware Overview

- 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?

- 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.

Middleware and Models (1)

- 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*.

Middleware Technologies

- 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, …
    - 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.

Middleware and Models (2)

- 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 being understandable and explainable through a coherent, limited set of concepts.

Middleware 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.
- Standardization efforts become critical to build momentum around an infrastructure technology.
  - Large standard consortia are built, which gather several industries together (OMG, W3C, FIPA).
  - Big industry players try to push their technology as *de facto* standards, or set up more open processes for them (Microsoft, IBM, Sun).

Middleware and Models (3)

- 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.

Middleware Discussion Template

- 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?

Middleware and Models (4)

- 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 FIPA agents.

Five OOPrinciples (2)

- **Open/Closed Principle (*OCP*).**
  - The language must allow the creation of modules *closed* for use but *open* for extension.
- **Single Choice Principle (*SCP*).**
  - Whenever there is a list of alternatives, *at most one module* can access it.

- The two principles above require Object-Orientation.
  - *OCP* requires *(implementation) inheritance*.
  - *SCP* requires *(inclusion) polymorphism*.

Distributed Objects

- Distributed systems need *quality software*, and they are a difficult system domain.
- OOP is a current *software best practice*.
- Question is:
  - 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.*

OOP Concept (1)

The fundamental concept of object-oriented programming is:

~~The Object~~

The Class

Back to Objects

- To describe the Distributed Objects model, let's review the basic OOP computation model.
  - The principles motivating OOP.
  - The central concept.
  - The central computation mechanism.
  - The central software evolution mechanism.

- "***Teach yourself OOP in 7 slides***".

OOP Concept (2)

- Def: ***Class***
  - *"An Abstract Data Type, with an associated Module that implements it."*

Type + Module = Class

Five OOPrinciples (1)

- **Modular Linguistic Units.**
  - The language must support modules in its syntax.
- **Embedded Documentation.**
  - A module must be self-documenting.
- **Uniform Access.**
  - A service must not disclose whether it uses stored data or computation.

- The three principles above are followed by OO languages, but also by Structured languages.

Distributing the Objects

- **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!
- …
- 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…

Modules and 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 and their properties.

(Distributed) Objects

The fundamental concept of Distributed Objects is:

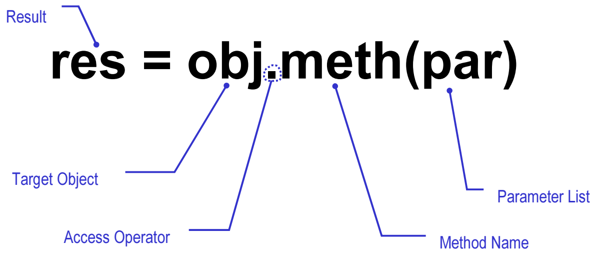
~~The Object~~

~~The Class~~

The Remote Interface

OOP Mechanism

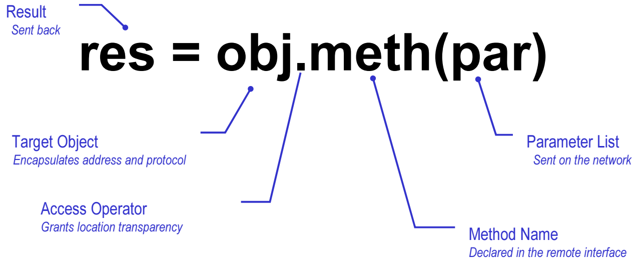
Fundamental OOP Computation Mechanism: ***Method Call***



res = obj.meth(par)

(Distributed) Objects

Fundamental Computational Mechanism: ***Remote Method Call***



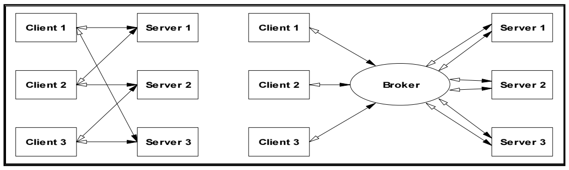
res = obj.meth(par)

Result
Sent back
Target Object
Encapsulates address and protocol
Access Operator
Grants location transparency
Method Name
Declared in the remote interface
Parameter List
Sent on the network

OOP Extensibility

- 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***.
- Inheritance: a partial order on modules.
  - A module grants special access to its sub-modules.
  - Allows to comply with the Open/Closed Principle.

Broker Architecture

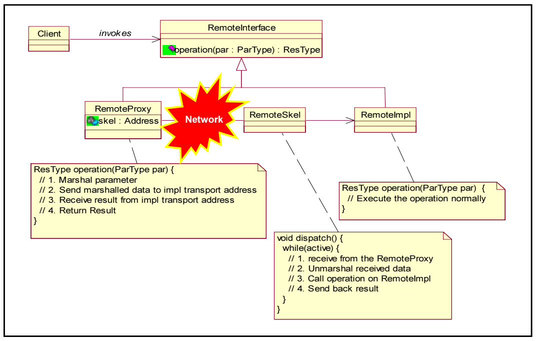


- *Broker* is an architectural pattern in [BMRSS96].
  - Stock market metaphor.
  - *Publish/subscribe* scheme.
  - Extensibility, portability, interoperability.
  - A broker reduces logic links from $N_c \bullet N_s$ to $N_c + N_s$.

Distributed (Objects)

Communication Mechanisms	Structured	Object Oriented
Explicit	**C Sockets**	**java.net.***
Implicit	**RPC**	**CORBA** **java.rmi.***

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
}
```

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

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

Distributed (Objects)

- The Distributed Objects communication model is *implicit*.
  - Transmission is implicit, everything happens through *stubs*.
  - The *stub* turns an ordinary call into an *IPC* mechanism.
  - One gains homogeneous handling of both local and remote calls (*location transparency*).

What's CORBA

- The word
  - An acronym for *Common ORB Architecture*.
  - ORB is an acronym again: *Object Request Broker*.
  - CORBA is a standard, not a product.
- The proponents
  - *Object Management Group* (OMG).
    - A consortium of more than 800 companies, founded in 1989.
    - Present all major companies.
      `http://www.omg.org`
    - The same institution that took up the *Unified Modeling Language* specification from its original creator, Rational Software Corp.

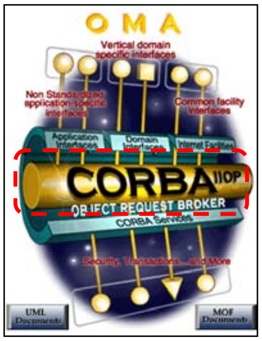
Distributed (Objects)

- The Distributed Objects communication model is *object oriented*.
  - Only *objects* exist, invoking *operations* on each other.
  - The 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*.

OMA - ORB Core

- Part of the OMA dealing with communication mechanisms.
- Allows remote method invocation regardless of:
  - Location and network protocols.
  - Programming language.
  - Operating System.
- The transport layer is hidden from applications using *stub* code.

Object Management Architecture

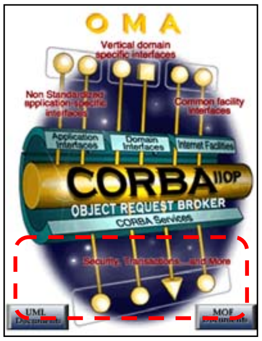

O M A
Vertical domain specific interfaces
Non Standardized application specific interfaces
Common facility interfaces
Application Interfaces
Domain Interfaces
Internet Facilities
CORBA IIOP
OBJECT REQUEST BROKER
CORBA Services
Security, Transactions.... and More
UML Documents
MOF Documents

- The OMA architecture was OMG overall vision for distributed computing.
  - The *Object Request Broker* is *OMA* backbone.
  - The *IIOP* protocol is the standard application transport that grants interoperability.
- Now, the *OMA* vision has been superceded by the *Model Driven Architecture*, almost a meta-standard in itself.

Remote invocation: Participants

- A *Request* is the closure of an invocation, complete with target object, actual parameters, etc.
- The *Client* is the object making the request.
- The *Object Implementation* is the logical object serving the request.
- The *Servant* is the physical component that incarnates the Object Implementation.
- The ORB connects Client and Servant.

Object Management Architecture

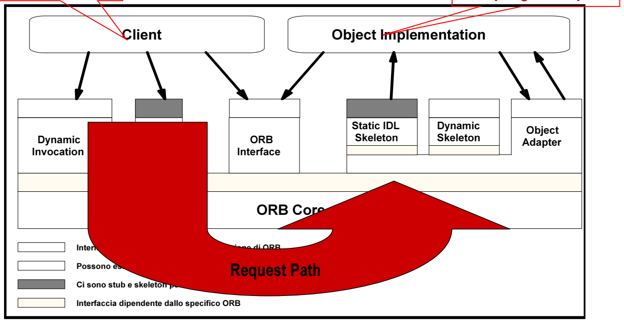

O M A
Vertical domain specific interfaces
Non Standardized application specific interfaces
Common facility interfaces
Application Interfaces
Domain Interfaces
Internet Facilities
CORBA IIOP
OBJECT REQUEST BROKER
CORBA Services
Security, Transactions.... and More
UML Documents
MOF Documents

- The *Common Object Services* serve as CORBA system libraries, bundled with the ORB infrastructure.
  - Naming and Trader Service.
  - Event Service.
  - Transaction Service.
  - ...

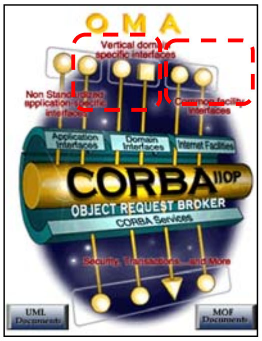
ORB Core Components

Invokes a method creating a request

Implements a method accepting the request



Object Management Architecture


O M A
Vertical domain specific interfaces
Non Standardized application specific interfaces
Common facility interfaces
Application Interfaces
Domain Interfaces
Internet Facilities
CORBA IIOP
OBJECT REQUEST BROKER
CORBA Services
Security, Transactions.... and More
UML Documents
MOF Documents

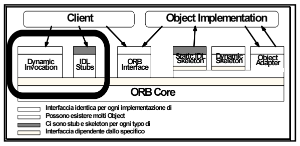
- The *Common Facilities* are frameworks to develop distributed applications in various domains.
  - *Horizontal Common Facilities* handle issues common to most application domains (GUI, Persistent Storage, Compound Documents).
  - *Vertical Common Facilities* deal with traits specific of a particular domain (Financial, Telco, Health Care).

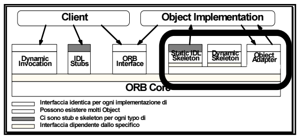
ORB Core Interfaces

- Static skeleton (IDL)
  - Corresponds to the Client Stub on Object Implementation side.
  - Automatically generated by compilation tools.
  - Builds parameters from network format (*unmarshaling*), calls the operation body and sends back the result.
- Dynamic Skeleton Interface (DSI)
  - Conceptually alike to Dynamic Invocation Interface.
  - Allows the ORB to forward requests to Object Implementations it does not manage.
  - Can be used to make *bridges* between different ORBs.

ORB Core Interfaces

- Client side interfaces:
  - Client Stub.
  - Dynamic Invocation Interface (DII).
- Server side interfaces:
  - Static Skeleton.
  - Dynamic Skeleton Interface (DSI).
  - Object Adapter (OA).
    - CORBA 2.0 → BOA.
    - CORBA 2.3 → POA.


Client
Object Implementation
Dynamic Invocation
IDL Stubs
ORB Interface
Static IDL Skeleton
Dynamic Skeleton
Object Adapter
ORB Core
Interfaccia identica per ogni implementazione di
Possono esistere molti Object
Ci sono stub e skeleton per ogni typo di
Interfaccia dipendente dallo specifico


Client
Object Implementation
Dynamic Invocation
IDL Stubs
ORB Interface
Static IDL Skeleton
Dynamic Skeleton
Object Adapter
ORB Core
Interfaccia identica per ogni implementazione di
Possono esistere molti Object
Ci sono stub e skeleton per ogni typo di
Interfaccia dipendente dallo specifico

ORB Core Interfaces

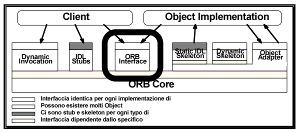
- Object Adapter (OA)
  - Connects the *Servant* (the component containing an Object Implementation) to the ORB.
  - In CORBA the Object Implementation is *reactive*.
    - The OA has the task of activating and deactivating it.
  - There can be many Object Adapters.
    - The CORBA 2.0 standard specifies the *Basic Object Adapter* (*BOA*).
    - The CORBA 2.3 standard specifies the *Portable Object Adapter* (*POA*).

ORB Core Interfaces

- Client (IDL) Stub.
  - Specific of each remote interface and operation, with *static typing* and *dynamic binding*.
  - Automatically generated by compilation tools.
  - Conversion of request parameter in network format (*marshaling*).
  - Synchronous, blocking invocation.

ORB Core Interfaces

- ORB Interface
  - Common interface for maintenance operations.
  - Initialization functions.
  - Bi-directional translation between Object Reference and strings.
  - Operations of this interface are represented as belonging to pseudo-objects.


Client
Object Implementation
Dynamic Invocation
IDL Stubs
ORB Interface
Static IDL Skeleton
Dynamic Skeleton
Object Adapter
ORB Core
Interfaccia identica per ogni implementazione di
Possono esistere molti Object
Ci sono stub e skeleton per ogni typo di
Interfaccia dipendente dallo specifico

ORB Core Interfaces

- Dynamic Invocation Interface (DII)
  - Generic, with *dynamic typing* and *dynamic binding*.
- Directly provided by the Object Request Broker.
- Both *synchronous* and *deferred synchronous* invocations are possible.
- Provides a *reflective* interface
  - Request, parameter, ...

OMA - Common Object Services

- *Design guidelines for CORBAservices*
  - Essential and flexible services.
  - Widespread use of multiple inheritance (*mix-in*).
  - Service discovery is orthogonal to service use.
  - Both local and remote implementations are allowed.
- ***CORBAservices are ordinary Object Implementations***.

OMA - Common Object Services

- *Naming Service*.
  - Handles name ⇔ Object Reference associations.
  - Fundamental as bootstrap mechanism.
  - Allows tree-like naming structures (*naming contexts*).
- *Object Trader Service*.
  - Yellow Page service for CORBA objects.
  - Enables highly dynamic collaborations among objects.

OMA - Common Object Services

- *Life Cycle Service*.
  - Object creation has different needs with respect to object use ⇒ the Factory concept is introduced.
  - *Factory Finders* are defined, to have location transparency even at creation time.
  - This service does not standardize Factories (they are class-specific), but ***copy***, ***move*** and ***remove*** operations.

CORBA Interoperability

- CORBA is heterogeneous for Operating System, network transport and programming language.
- With the 1.2 version of the standard, interoperation was limited to ORBs from the same vendor.
- In CORBA 1.2 two objects managed by ORBs from different vendors ***could not*** interact.
- CORBA 2.x grants interoperability among ORBs from different vendors.

CORBA Interoperability

- Recipe for interoperability
  1) **Communication protocols shared among ORBs.**
  2) **Data representation common among ORBs.**
  3) **Object Reference format common among ORBs.**

⇒ Only ORBs need to be concerned with interoperability.

CORBA Interoperability

- *Common communication protocols*
  - The standard defines the *General Inter-ORB Protocol* (GIOP), requiring a reliable and connection-oriented transport protocol.
  - With TCP/IP one has *Internet Inter-ORB Protocol* (IIOP).
- *Common data representation*
  - As part of GIOP the *CDR* (*Common Data Representation*) format is specified.
  - *CDR* acts at the *Presentation* layer in the *ISO/OSI* stack.
- *Common Object Reference format*
  - *Interoperable Object Reference* (IOR) format.
    - Contains all information to contact a remote object (or more).

The OMG IDL Language

Overall OMG IDL language features.

- Syntax and lexicon similar to *C/C++/Java*.
- Only expresses the *declarative* part of a language.
- Services are exported through *interfaces*.
- Support for OOP concept as inheritance or polymorphism.

Programming with CORBA

- The *Broker* architecture allows to build distributed applications, heterogeneous with respect to:
  - Operating System.
  - Network Protocol.
- The *OMG IDL* language allows to build distributed applications, heterogeneous with respect to:
  - Programming Language.
- But, the system will have to be implemented in some *real* programming languages at the end.
  - The IDL specification have to be cast into those languages

Programming with CORBA

- CORBA programming environments feature a tool called *IDL compiler*.
  - It accepts *OMG IDL* as input, and generates code in a concrete implementation language.
- With respect to a given IDL interface, a component may be a *client* and/or a *server*.
  - The *client* requests the service, the *server* exports it.
  - The IDL compiler generates code for both.

OMA - Common Object Services

- *Event Service*.
  - Most objects are *reactive*.
  - The Event Service enables notification delivery, decoupling the producer and the consumer with an *event channel*.
  - Supports both the *push* model (*observer*) and the *pull* model for event distribution.
  - Suitable administrative interfaces allow to connect *event supplier* and *event consumer* of *push* or *pull* kind.
- Notification Service
  - Improves the Event Service, with more flexibility.

OMA - Common Object Services

- *Transaction Service.*
  - Transactions are a cornerstone of business application.
  - A *two-phase commit* protocol grants *ACID* properties.
  - Supports *flat* and *nested* transactions.
- *Concurrency Control Service*.
  - Manages lock objects, singly or as part of groups.
  - Integration with the Transaction Service.
    - *Transactional lock* objects.

The OMG IDL Language

Motivation for an *Interface Definition Language*.

- *CORBA* is neutral with respect to programming languages.
- Different parts of an application can be written in different languages.
- A language to specify interactions across language boundaries is needed ⇒ *Interface Definition Language (IDL)*.

Objects and Metadata

- To increase system flexibility, one has to add a new level that:
  - Describes system capabilities.
  - Allows changing them at runtime.
- Data belonging to this second level are *"data about other data"*, that is they are *metadata* (e. g. the schema of a DB).
  - Systems have a (usually small) number of *meta-levels* (e.g. objects, classes and metaclasses in Smalltalk, ot the four-layer meta-model of UML).

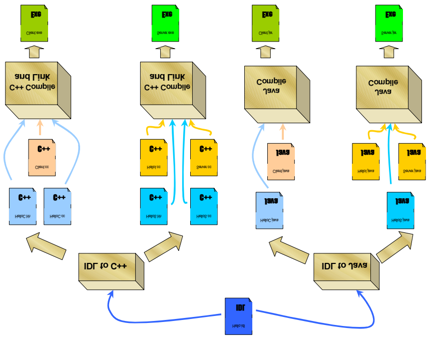
Objects and Metadata

- Object oriented software system were soon given metadata:
  - *Smalltalk* has *Metaclasses*.
  - *CLOS* (Common Lisp Object System) introduced the concept of *Meta-Object Protocol*.
  - *Java* has a *Reflection API* since version 1.1.
- In the book *"Pattern Oriented System Architecture: A system of Patterns"*, ***Reflection*** is an architectural pattern.

CORBA Metadata

- CORBA is an integration technology.
- Therefore, the issue of metadata and *Reflection* was given appropriate attention.
- In a distributed system, metadata have to be *persistent*, *consistent* and *available*.

Programming with CORBA



Programming with CORBA

- For each supported programming language, the CORBA standard specifies a *Language Mapping*:
  - How every *OMG IDL* construct is to be translated.
  - Programming techniques that are to be used.
- C++ Language Mapping.
- Java Language Mapping.
- Smalltalk Language Mapping.
- Python Language Mapping.

Objects and Metadata

- Compile-time vs. Run-time
  - In C++ and Java the state of an object can change at runtime, but its structure is carved by the compilation process.
  - Usually, the overall set of classes and functions belonging to the system is defined at compile time and cannot vary.
- With dynamic linking these rules can be overcome, but traditional systems tend to follow them anyway.

The Dynamic Invocation Interface

- To create a request, one uses the IDL:

```
module CORBA { // PIDL
  pseudo interface Object {
  typedef unsigned long ORBStatus;
  ORBStatus create_request(in Context ctx,
    in Identifier operation, // Operation name
    in NVList arg_list,  // Operation arguments
    inout NamedValue result, // Operation result
    out Request request, // Newly created request
    in Flags req_flags;  // Request flags);
  }; // End of Object pseudo interface
}; // End of CORBA module
```

The Dynamic Invocation Interface

- After creation, a request object can be used:

```
– module CORBA {
    typedef unsigned long Status;
    pseudo interface Request {
      Status add_arg(in Identifier name,
        in TypeCode arg_type,
        in any value, in long len,
        in Flags arg_flags);
      Status invoke(in Flags invoke_flags);
      Status delete(); // Destroy request object
      Status send(in Flags invoke_flags);
      Status get_response(in Flags response_flags);
    }; // End of Request interface
  }; // End of CORBA module
```

The Dynamic Invocation Interface

- The *DII*, through request objects, allows selecting the *rendezvous policy*:
  - *Synchronous* call with `invoke()`.
  - *Deferred synchronous* call with `send()`.
- With deferred synchronous invocations, a group of requests can be sent all at once.
- The new *Asynchronous Method Invocation* (*AMI*) specification of CORBA 2.4 also introduces asynchronous calls.

CORBA Metadata

- In the OMA architecture, metadata are used in several parts:
  - The *Dynamic Invocation Interface* allows to act on the remote operation invocation mechanism itself.
  - The *Interface Repository* allows runtime discovery of new *IDL* interfaces and their structure.
  - The *Trader Service* gathers services exported by objects into a yellow-page structure.

The Dynamic Invocation Interface

- Goals of the DII
  - The DII provides a complete and flexible interface to the remote invocation mechanism, around which CORBA is built.
  - The central abstraction supporting the DII is the *Request* pseudo-object, which *reifies* an instance of a remote call (see the *Command* design pattern in the *Gang of Four* book).

The Dynamic Invocation Interface

- IDL interfaces for the DII
  - Firstly, a request attached to a CORBA object needs be created.
  - The `create_request()` operation, belonging to the `Object` pseudo-interface (minimum of the inheritance graph), is to be used.
  - When a request is created, it is associated to its original *Object Reference* for its whole lifetime.

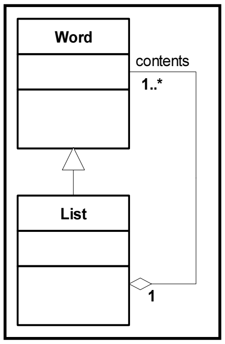
The Interface Repository

- Object oriented representation of the syntax of a language:
  - The formal grammar (e.g. in BNF notation) can be turned into a structure of classes and associations.
  - To do this, one defines a class for each non-terminal symbol of the given grammar.
- Approach followed by OO parser generators (*ANTLR, JavaCC*).
  - *Interpreter* design pattern from *Gang of Four* book.

The Interface Repository

- The *BNF* expression of a list of words (with right recursion) results in the *Composite* design pattern of the *Gang of Four* book:

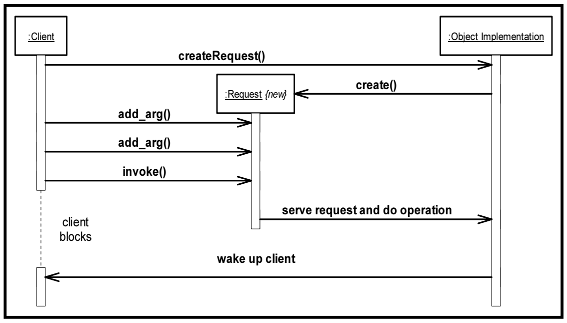
```
<list> ::=
  <word>
  | <list> <word>
```



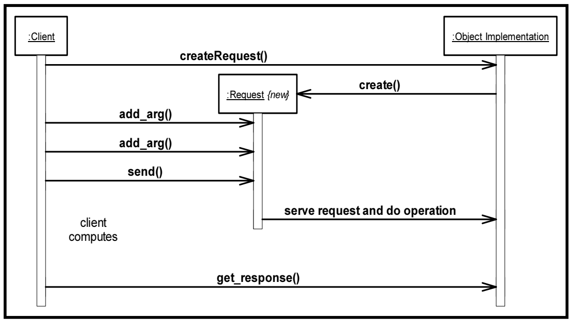
The Interface Repository

- The *OMG IDL* language representation:
  - A complete OO representation of the *IDL* language is stored within the *Interface Repository*.
  - The *IDL* BNF results in both *has-a* and *is-a* links in the objects structure.
- The `Repository` interface is the root of the containment hierarchy, whereas the `IRObject` interface is the root of the inheritance hierarchy.
- The two `Container` and `Contained` interfaces form a *Composite* structure.

Synchronous Call with the DII



Deferred Synchronous Call



The Interface Repository

- The *Interface Repository* keeps the descriptions of *all* the IDL interfaces available in a CORBA domain.
- Using the *Interface Repository*, programs can discover the structure of types they don't have the *stubs* for.
- The `TypeCode` interface provides an encoding of the *OMG IDL* type system.

Dynamic Collaboration

- CORBA objects are more adaptable than ordinary, programming language objects such as Java or C++ objects.
- Two CORBA objects **A** and **B**, initially knowing nothing about each other, can set up a collaboration.
  - Object **A** uses `get_interface()` to get an `InterfaceDef` describing **B**.
  - Browsing the *Interface Repository*, **A** discovers the syntax of **B** supported operations.
  - Using *DII*, **A** creates a request and sends it to **B**.

Dynamic Collaboration

- With CORBA, the *syntax* of the operations can be discovered at runtime.
- But the *semantics* of the operation is missing: *OMG IDL* lacks *preconditions*, *postconditions* and *invariants*.
- More complex systems (like *multi-agent systems*) need languages to describe the domain of the discourse (***ontologies***).

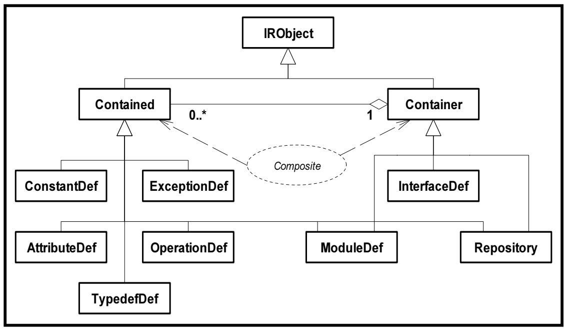
Summary on Distributed Objects

An impressive technology!

Extends OOP to Distributed Systems.
Hides DS programming complexity.
Supported by an open standard (OMG CORBA).
Integration across OSs, networks and languages.
A lot of free implementations available.

- Next in line: ***Multi-Agent Systems***
  - An emergent technology.
  - Can they do better than Distributed Objects?

The Interface Repository



The Interface Repository

- Using the *Interface Repository*:
  - Objects stored within the *Interface Repository* are an equivalent representation of actual *OMG IDL* source code.
  - Browsing the *Interface Repository*, one can even rebuild *IDL* sources back.
- With *Repository IDs*, more interface repositories can be federated.

The Interface Repository

- Every interface derived from `IRObject` supports two kinds of operations.
  - *Read Interface* to explore metadata (*Introspective Protocol*).
  - *Write Interface* to modify them and create new ones (*Intercessory Protocol*).
- Every interface derived from `Container` supports navigation operations, as well as new elements creation operations.

WHITESTEIN
Technologies

From Distributed Objects to Multi-Agent Systems: Evolution of Middleware (2)

Giovanni Rimassa

Whitestein Technologies AG – (`gri@whitestein.com`)

Summary on Distributed Objects

An impressive technology!

Extends OOP to Distributed Systems.
Hides DS programming complexity.
Supported by an open standard (OMG CORBA).
Integration across OSs, networks and languages.
A lot of free implementations available.

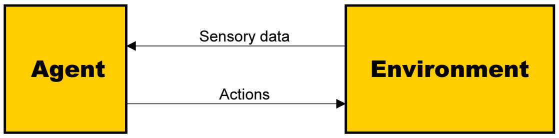
- Next in line: ***Multi-Agent Systems***
  - An emergent technology.
  - Can they do better than Distributed Objects?

Agent Middleware

- According to our previous discussion schema, an *Agent* middleware is supposed to:
  - Promote an *agent-oriented* ***Model***.
  - Realize an *agent-oriented* ***Infrastructure***.
- We will have to go through some steps:
  - Describe **what** agents and multi-agent system **are**.
  - Compare the agent/MAS model with the OO model.
  - Describe what kind of software components agents are.
  - Provide an infrastructure example: the FIPA standard.
  - Provide an implementation example: JADE.

What is a software agent?

- A *software agent* is a software system that can operate in dynamic and complex environments.
  - It can *perceive* its environment through *senses*.
  - It can *affect* its environment through *actions*.



Agenthood properties

- Fundamental features.
  - An agent is *autonomous*.
  - An agent is *reactive*.
  - An agent is *social*.
- Useful features.
  - An agent can be *proactive* (goal-*directed*).
  - An agent can be *mobile*.
  - An agent can be *adaptive* (or *learning*).

Autonomous Agents

Multi Agent Systems

Mobile Agents

Learning Agents

Intelligent Agents

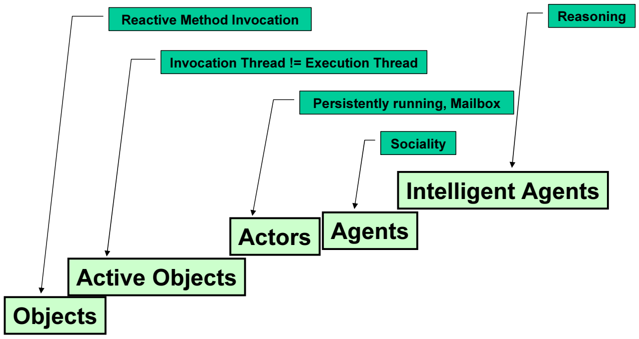
Application areas

- Information management.
  - Information Filtering.
  - Information Retrieval.
- Industrial applications.
  - Process control.
  - Intelligent manufacturing.
- Electronic commerce.
- Computer Supported Cooperative Work.
- Electronic entertainment.

Concurrent OOP

- Classical method invocation is a tight bond between caller and called object.
  - Not that this is always a bad thing (cohesion vs. coupling).
- However, in concurrent OOP things change a lot.
  - To exploit parallelism, other rendezvous policies are used, such as deferred synchronous or asynchronous.
  - In concurrent method invocation, *correctness preconditions* become *synchronization guard predicates*.
- The bond of classical *Design by Contract* is extremely loosened!

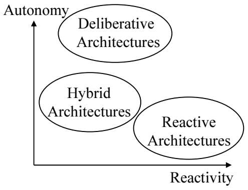
A Stairway to Agents


Reactive Method Invocation
Invocation Thread != Execution Thread
Persistently running, Mailbox
Sociality
Reasoning
Intelligent Agents
Actors
Agents
Active Objects
Objects

Building a single agent

- Various proposals for an agent architecture.

- Deliberative architectures
  - Explicit, symbolic model of the environment.
  - Logic reasoning.
- Reactive architectures
  - Stimulus ⇒ Response.
- Hybrid architectures
  - BDI, Layered, ...



Autonomy and Reactivity

- First fundamental trait of an agent: ***<u>autonomy</u>***.
  - An agent can act on the environment, on the basis of its internal evolution processes.
- Second fundamental trait: ***<u>reactivity</u>***.
  - An agent can perceive changes in the environment, providing responses to external stimuli.
- How do these qualities compare with objects?
  - Objects ***are*** reactive.
  - Objects ***are not*** autonomous.

Master and Servant (1)

- Fundamental computational mechanism of the OOP:
  - ***Method invocation***.
  - An object exposes its capabilities (*public methods*).
  - Then other objects exploit them how and when they like (they decide *when* to invoke the methods and *which parameters* to pass to them).
- An object decides its behaviour space, but does not further control its own behaviour.

- The object is **<u>servant</u>**, its caller is **<u>master</u>**.

Master and Servant (2)

- Method invocation follows *Design by Contract*:
  - It is a *synchronous* rendezvous, so the caller object has to wait until the called object completes its task.
  - The caller must ensure the correctness precondition of the method are verified before invoking it.
- Though the caller object chooses the method to invoke, then it surrenders itself (i.e. its thread of control) to code that it is controlled by the called.
- The object is **<u>master</u>**, its caller is **<u>servant</u>**.

Say What?

- An Agent Communication Language captures:
  - The speaker (sender) and hearer (receiver) identities.
  - The kind of speech act the sender is uttering.
  - This should be enough to understand the message.
- “I request that you *froznicate* the *quibplatz*”.
  - …
- There is more to the world than people and words.
  - There are also *things*.
  - A common description of the world is needed.
  - Describing *actions*, *predicates* and *entities*: ***ontologies***.

Sociality: From Agent To MAS

- Autonomy and Reactivity are about an agent and its environment.
- Sociality is about having more than one agent and they building relationships.
- The shift towards the *social level* marks the border between ***Agent*** research and ***Multi-Agent Systems*** (MAS) research.
  - This is the major trait differentiating (non-intelligent) agents from classical actors.

Interaction and Coordination

- A MAS is more than a bunch of agents.
  - In order to get something useful, some constraints have to be set on what agents can do.
  - Agents can represent different stakeholders.
- The *society* metaphor as a modeling tool.
  - *Social Role Model*: which parts can be played in the society (***static, structural model***).
  - *Interaction and Coordination Model*: which patterns conversation can follow (***dynamic, behavioral model***).
- Specifying *conversation patterns* with ***Interaction Protocols***.

Communication in MAS

- MASs need a richer, more loosely coupled communication model with respect to OO systems.
- Approach: trying to mimic human communication with natural language.
  - When people speak, they try to make things happen.
  - Listening to someone speaking, something of her internal thoughts is revealed.
  - When institutionalized, word **is** law (“*I pronounce you…*”).
- A linguistic theory results in a communication model.
  - *Speech Act Theory.*
  - *Agent Communication Languages* (*ACLs*).

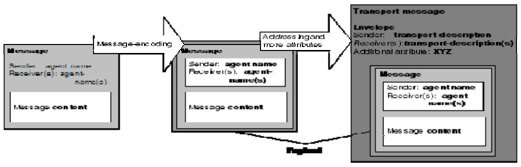
Standards for Agents

- To achieve interoperability among systems independently developed, a common agreement is needed.
- Several institutions are interested in building standards for agent technology.
  - Agent Society;
  - Foundation for Intelligent Physical Agents;
  - Internet Engineering Task Force;
  - Object Management Group;
  - World Wide Web Consortium.

Speech Act Theory and ACLs

- Theory of human communication with language.
  - Considers sentences for their effect on the world.
  - A *speech act* is an act, carried out using the language.
- Several categories of speech acts.
  - Orders, advices, requests, queries, declarations, etc.
- Agent Communication Languages use *messages.*
  - Messages carry speech act from an agent to another.
  - A message has *transport slots* (sender, receiver, …).
  - A message has a *type* (request, tell, query).
  - A message has *content slots.*

FIPA ACL Message Layers



- The previous message is a Speech-Act Level message.
- A Speech-Act Level message has an encapsulated *content*.
  - Expressed in a *content language*, according to an *ontology*.
- For transport reasons, it is encapsulated again.
  - An *envelope* is added, to form a *Transport-Level message*.

FIPA

Foundation for Intelligent Physical Agents

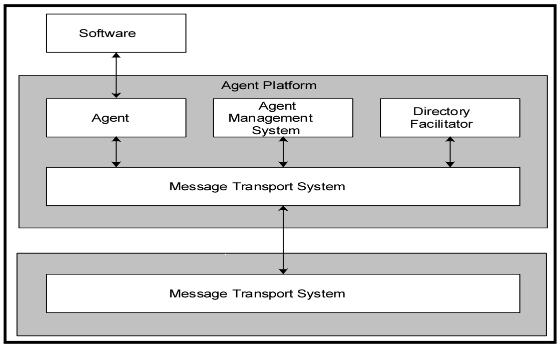
http://www.fipa.org

- FIPA is a world-wide, non-profit association of companies and organizations.
- FIPA produces specifications for generic MAS and agent technologies.
- Promotes agent-level and platform-level interoperability among MAS developed independently.

FIPA Ontologies and IPs

- FIPA specifications heavily rely on ontologies.
  - All significant concepts are collected in standard ontologies (`fipa-agent-management` , etc.).
  - An *Ontology Service* is specified for ontology brokering.
- A set of standard Interaction Protocols is provided.
  - Elementary protocols directly induced by the semantics of the single *communicative acts* (`fipa-request`, `fipa-query`, etc.).
  - More sophisticated negotiation protocols (`fipa-contract-net`, `fipa-auction-dutch`, etc.).

FIPA Platform Architecture



FIPA ACL

- The FIPA ACL complies with a *communication model*.
  - Based on the speech-act theory.
  - Speech acts correspond to *communicative acts* in FIPA.
  - FIPA CAs are gathered in the FIPA CA Library.
  - A formal semantics for each act is provided.

FIPA ACL Message

```
(REQUEST
 :sender  ( agent-identifier  :name da0)
 :receiver  (set ( agent-identifier  :name df) )
 :content  "((action (agent-identifier :name df )
            (register (df-agent-description
              :name (agent-identifier :name da0)
              :services (set ( service-description
                   :name sub-sub-df :type fipa-df
                   :ontologies (set fipa-agent-management )
                   :languages (set FIPA-SL)
                   :protocols (set fipa-request ) :ownership JADE ))
              :protocols (set ) :ontologies (set ) :languages (set )
            )) ) )"
 :reply-with  rwsub1234 :language  FIPA-SL0
 :ontology  FIPA-Agent-Management :protocol  fipa-request
 :conversation-id  convsub1234
)
```

FIPA ACL

- With speech acts, we follow the *communication as attempt* idea.
  - The speaker tells the world something about her mind (beliefs, intentions, …).
  - The hearer is not forced to react.
  - We can have pre-conditions for the speaker to speak, but **no** post-conditions.
  - We can infer the intentions of the speaker.

FIPA ACL

- Each CA semantics is expressed with a *modal logic* system.
  - Modal logics define a set of *modalities*, grouping logical formulas.
  - Within a modality, the usual first order logic applies.
  - There are axioms and rules to link modalities among each other.

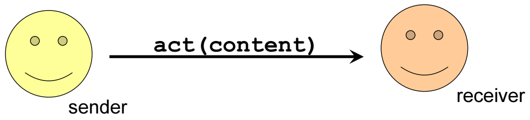
FIPA ACL

- The formal semantics of a FIPA communicative act comprises:
  - What must be true for the sender before sending a CA (***feasibility precondition***).
  - Which intentions of the sender **could** be satisfied as a consequence of sending the CA (***rational effect***).

FIPA ACL

- The modal logic used in FIPA ACL applies the BDI agent model.
  - **B**eliefs (what an agent thinks he knows now).
  - **D**esires (what an agent wishes to become true).
  - **I**ntentions (what an agent will try to make true).
- The BDI model adopts the *Intentional Stance*.

FIPA ACL



- Observer knows **`act`** has **`<FP, RE>`**.
  - It can deduce `FP(content)`.
  - It can deduce I_{sender}`(RE(content))`.
  - **Nothing** can be deduced about the receiver.

FIPA ACL

- The Intentional Stance is a way to model complex systems, whose details are unknown.
  - Attributing mentalistic traits to the system.
  - Explaining its behaviour with them.
- Example: a computer chess player.
  - Does it ‘want’ to win?
  - Does it ‘fear’ to lose?

FIPA ACL

- Content element: Predicate.
  - A logic formula, with zero or more *terms*, yielding a boolean value.
- Content element: Action.
  - An operation of an agent on its environment.
  - Has zero or more *terms*, yields no result.
  - Complex *action expressions* can be built with `;` and `|` operators.

FIPA ACL

Agent **i** believes φ to be true

Agent **j** desires that ψ be true

Agent **k** intends to make it so that θ be true

- Content term: Object Description.
  - Frame structure, with named slots.

  `(person :name Giovanni :age 32)`
- Content term: Variable.

  `?x`
- Content term: Modal operators.

  $B_i\varphi$ $C_j\psi$ $I_k\theta$

FIPA ACL

- Content term: Action operators.
  - They link actions with their premises and their consequences.
  - `Agent(i, a)` – Agent `i` is the one performing actions in action expression `a`.
  - `Feasible(a, p)` – Action `a` can be done, and predicate `p` will hold just after that.
  - `Done(a, p)` – Action `a` was done, and predicate `p` held just before that.
  - Both have the predicate defaulting to `true`.

FIPA ACL

- FIPA ACL is an intentional language for component communication.
  - Better suited for autonomous components.
- In Object-Oriented systems, Design by Contract is followed.
  - Better suited for passive components.
- How do they compare?

FIPA ACL

- With Design by Contract, a method has preconditions and postconditions.

$$\frac{\{\mathtt{pre(formals)}\}\mathbf{body}\{\mathtt{post(formals)}\}}{\{\mathtt{pre(actuals)}\}\mathbf{call}\{\mathtt{post(actuals)}\}}$$

- A FIPA ACL CA has FPs and REs.

$$\frac{\{\mathtt{FP(content)}\}\mathbf{CA}\{\mathtt{RE(content)}\}}{\{\mathtt{FP(content')} \wedge I_s\mathtt{(RE(content'))}\}\mathbf{send}\{\}}$$

FIPA ACL

- The FP and RE are predicates over the message content.
  - A content model is needed.
- Acts have different content types.
  - Some acts contain *predicates*.
  - Some other contain *actions*.
  - Content expressions can also hold *object descriptions* and several *operators*.

The `inform` CA

- The sender informs the receiver that a given proposition is true.
  - The content is a predicate.
  - The sender believes the content.
  - The sender wants the receiver to believe it.
- Formalizing `<s, inform(r, φ)>`:
  - FP: $B_s\varphi \wedge \neg B_s(B_r\varphi \vee B_r\neg\varphi)$
  - RE: $B_r\varphi$

The `request` CA

- The sender requests the receiver to perform some action.
  - The content is an action expression.
  - A CA is an action and can be requested.
- Formalizing `<s, request(r, a)>`:
  - FP: $\texttt{FP(a)[i/j]} \wedge B_s\ \texttt{Agent(r, a)} \wedge \neg B_s I_r\ \texttt{Done(a)}$
  - RE: `Done(a)`

The `query-if` CA

- The sender requests the receiver to tell whether a predicate is true.
- It is a composite act:

  `query-if(φ)` means:

  `request(inform(φ) | inform(¬φ))`
- Formalizing `<s, query-if(r, φ)>`
  - FP: Replace `a` with the two `inform` CAs.
  - RE: `Done(<r, inform(s, φ)> | <r, inform(s, ¬φ)>)`

FIPA ACL

- Content term: Identifying reference expression (*IRE*).
  - Used in the reponse to open questions.
  - Corresponds to logical quantifiers, but yields a value.

Universal:	`all ?x, φ(?x)`
Existential:	`any ?x, φ(?x)`
One and only one:	`iota ?x, φ(?x)`

FIPA ACL

- IRE vs. quantifier example.
  - To show the difference, let's use an example question.
- "What's the day today?"
  - Q1: $\exists!\ \texttt{?d,}\ B_{you}\texttt{today-is(?d)}$?
  - A1: "`Yes`".
  - Q2: $\texttt{iota ?d,}\ B_{you}\texttt{today-is(?d)}$?
  - A2: "`Today is Thursday`".

FIPA ACL

- The FIPA Communicative Act library specifies all FIPA CAs.
  - Each CA has an informal and formal (FP + RE) semantics.
  - An Appendix details the semantic model of CAs and their content.
  - FIPA Spec SC00037J.

Responder CAs

- A protocol has two roles:
  - ***Initiator*** role (triggers the protocol).
  - ***Responder*** role (receives initial triggers).
- There is a set of communicative acts dedicated to responders.
  - Agree.
  - Refuse.
  - Failure.
  - Accept-Proposal.

The `query-ref` CA

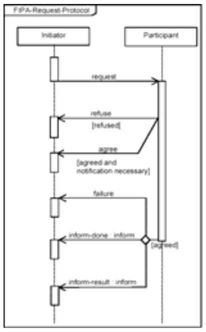
- The sender queries the receiver for the object(s) identified by an IRE.
  - The content is an IRE (any, iota or all).
  - It is a composite act:

query-ref(Ref$_x$φ(?x)) means:

request(inform-ref(Ref$_x$φ(?x)))

  - The `inform-ref` composite act means the disjunction of all possible `inform` acts over the range of the variable `?x`.

FIPA-Request

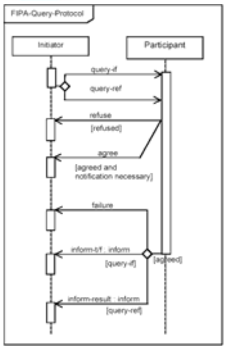


- The IP generated by the `request` CA.
  - An initial `request`.
  - An `agree`/`refuse` branch.
  - Actual action execution (not shown in the diagram).
  - Possible `failure` report.
  - Possible `inform` report.
    - Informing about completion.
    - Informing about action result.

Interaction Protocols

- Observing a single CA says nothing about the receiver.
  - No post-conditions outside sender's mind.
  - Messages can be lost (unreliable channel).
- To draw useful conclusions, we must move from utterances to conversations.

FIPA-Query



- The IP generated by the `query-if` or `query-ref` CA.
  - An initial query is sent.
  - An `agree`/`refuse` branch.
  - Possible `failure` report.
  - Possible `inform` report.
    - Informing whether (`query-if`).
    - Informing about query result (in the `query-ref` case).

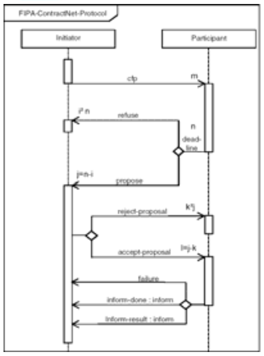
Interaction Protocols

- A rational agent tries to turn its intentions into its beliefs.
  - To do so, it must *act* on its environment, and then *perceive* the results.
  - It needs to both send and receive messages.
- FIPA specifies an *IP Library*, containing conversation templates.
  - IPs compose the semantics of single CAs.

JADE Family

- JADE has solved the basic MAS infrastructure problem.
  - Most new AgentCities nodes fire up JADE and go.
  - With *JADE-LEAP*, FIPA runs on wireless devices.
  - With *BlueJADE*, runs within J2EE app servers.
    - Palo Alto HP Labs OS spinoff project. (`http://sourceforge.net/projects/bluejade`).
- Users are moving on to higher level tasks.
    - Ontology design (Protegé plugin, WSDLTool).
    - Intelligent agents design (ParADE, Corese, JESS).

FIPA-Contract-Net



- More complex IP.
  - Does not follow simply from CAs semantics.
  - It embeds *policies*.
- One-to-many IP.
  - One *manager* agent.
  - N *contractor* agents.
  - A `cfp` is issued.
  - A contractor is selected among proponents.

JADE Features

- Distributed Agent Platform.
  - Seen as a whole from the outside world.
  - Spanning multiple machines.
- Transparent, multi-transport messaging.
  - Event dispatching for local delivery.
  - Java RMI for intra-platform delivery.
  - FIPA 2000 MTP framework.
    - IIOP protocol for inter-platform delivery.
    - HTTP protocol and XML ACL encoding.
  - Protocol-neutral, optimistic address caching.

FIPA and JADE

- FIPA is a world-wide, non-profit association of companies and organizations (`http://www.fipa.org`).
- FIPA produces specifications for generic MAS and agent technologies.
- Promotes agent-level and platform-level interoperability among MAS developed independently.

A FIPA 2000-compliant agent platform.
A Java framework for the development of MAS.
An Open Source project, © TI Labs, LGPL license.

JADE is a joint development of TI Labs and Parma University.
Project home page: `http://jade.cselt.it`.

JADE Features

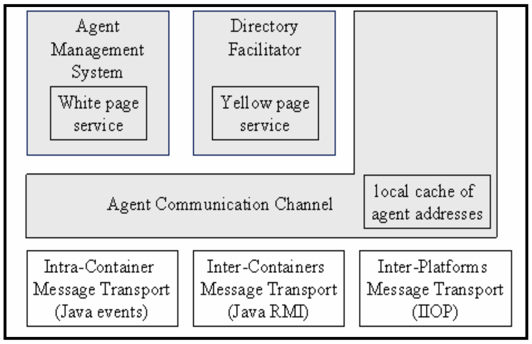
- Two levels concurrency model.
  - Inter-agent (pre-emptive, Java threads).
  - Intra-agent (co-operative, Behaviour classes).
- Object oriented framework for easy access to FIPA standard assets.
  - Agent Communication Language.
  - Agent Management Ontology.
  - Standard Interaction Protocols.
  - User defined Languages and Ontologies.

History of JADE

Welcome to the
Java Agent DEvelopment Framework

- Project started July 1998
- Present at both the first (Seoul, 1999) and the second (London, 2001) FIPA test.
- Many users worldwide.
  - 13 released versions.
  - Internet-based support.
  - Leading Open Source platform.

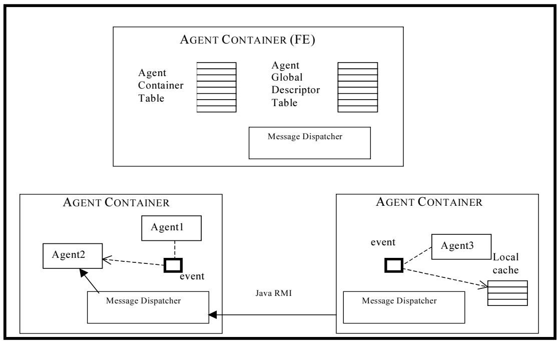
JADE Main Container


Agent Management System
White page service
Directory Facilitator
Yellow page service
Agent Communication Channel
local cache of agent addresses
Intra-Container Message Transport (Java events)
Inter-Containers Message Transport (Java RMI)
Inter-Platforms Message Transport (IIOP)

JADE Features

- User defined content languages and ontologies.
  - Each agent holds a table of its capabilities.
  - Message content is represented according to a meta-model, in a content language independent way.
  - User defined classes can be used to model ontology elements (*Actions*, *Objects* and *Predicates*).
- Agent mobility.
  - Intra-platform, *not-so-weak* mobility with on-demand class fetching.

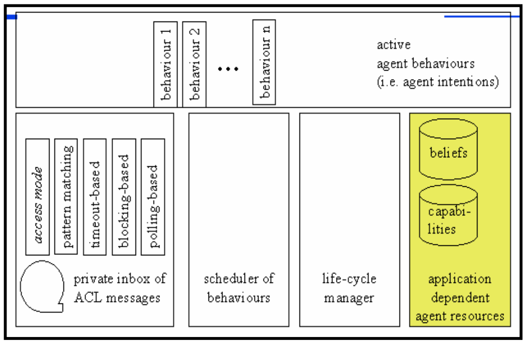
JADE Message Dispatching


AGENT CONTAINER (FE)
Agent Container Table
Agent Global Descriptor Table
Message Dispatcher
AGENT CONTAINER
Agent1
Agent2
event
Message Dispatcher
Java RMI
AGENT CONTAINER
event
Agent3
Local cache
Message Dispatcher

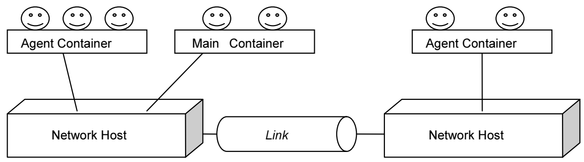
JADE Features

- Event system embedded in the kernel.
  - Allows observation of *Platform*, *Message*, *MTP* and *Agent* events.
  - Synchronous listeners, with lazy list construction.
- Agent based management tools.
  - *RMA*, *Sniffer* and *Introspector* agents use *FIPA ACL*.
  - Extension of `fipa-agent-management` ontology for JADE-specific actions.
  - Special `jade-introspection` observation ontology.

JADE Agent Architecture


behaviour 1
behaviour 2
...
behaviour n
active agent behaviours (i.e. agent intentions)
access mode
pattern matching
timeout-based
blocking-based
polling-based
private inbox of ACL messages
scheduler of behaviours
life-cycle manager
beliefs
capabi-lities
application dependent agent resources

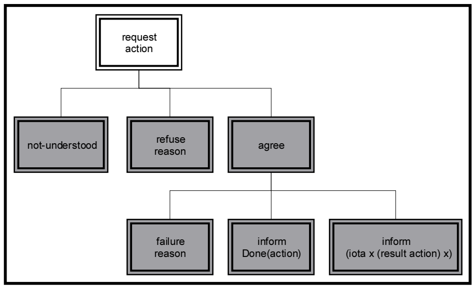
JADE Platform Architecture


Agent Container
Main Container
Agent Container
Network Host
Link
Network Host

- Software Agents are software components.
  - They are hosted by a runtime support called *Agent Container*.
  - Many agents can live in a single container (about 1000 per host).
- Selective Network Awareness and Flexible Deployment.
  - Any mapping between agents, containers and hosts.

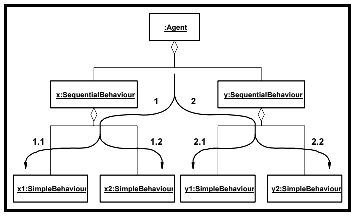
JADE Behaviours Example

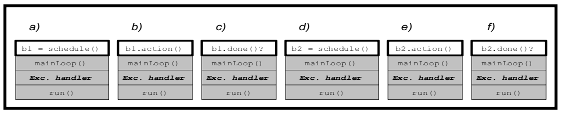
Fipa-Request interaction protocol (FIPA 97 spec).



JADE Concurrency Model

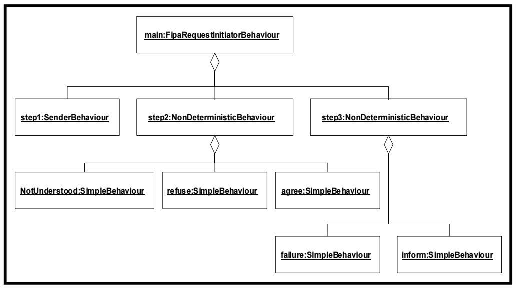
- Multithreaded inter-agent scheduling.
- Behaviour abstraction
  - *Composite* for structure
  - *Chain of Responsibility* for scheduling.
  - No context saving.





JADE Behaviours Example

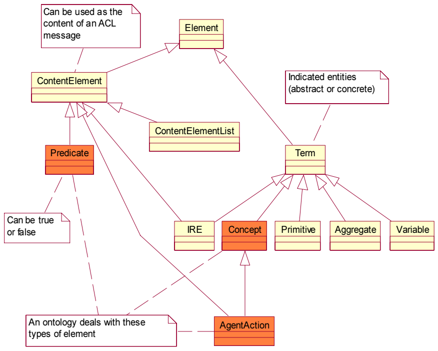
Object structure for *FipaRequestInitiatorBehaviour*.


main:FipaRequestInitiatorBehaviour
step1:SenderBehaviour
step2:NonDeterministicBehaviour
step3:NonDeterministicBehaviour
NotUnderstood:SimpleBehaviour
refuse:SimpleBehaviour
agree:SimpleBehaviour
failure:SimpleBehaviour
inform:SimpleBehaviour

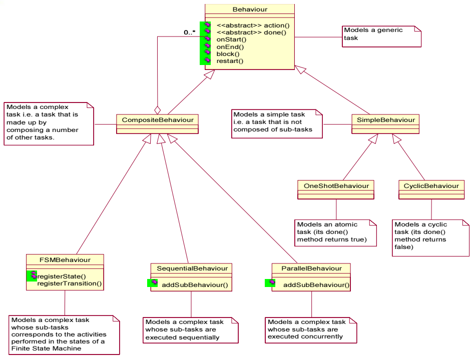
Behaviours and Conversations

- The behaviours concurrency model can handle many interleaved conversations.
  - Using the *Composite* structure, arbitrarily fine grained task hierarchies can be defined.
  - The new `FSMBehaviour` supports nested FSMs.
- FIPA Interaction protocols are mapped to suitable behaviours:
  - An *Initiator* Behaviour to start a new conversation.
  - A *Responder* Behaviour to answer an incoming one.

JADE Content Metamodel


Can be used as the content of an ACL message
Element
ContentElement
Indicated entities (abstract or concrete)
ContentElementList
Predicate
Term
Can be true or false
IRE
Concept
Primitive
Aggregate
Variable
An ontology deals with these types of element
AgentAction

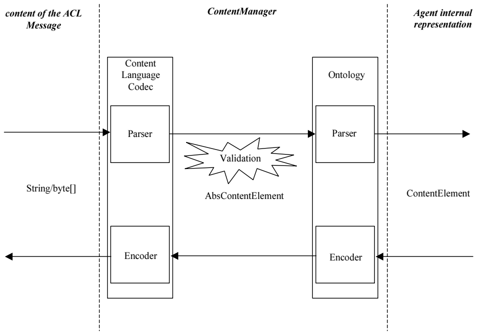
JADE Behaviours Model


Behaviour
<<abstract>> action()
<<abstract>> done()
onStart()
onEnd()
block()
restart()
0..*
Models a generic task
Models a complex task i.e. a task that is made up by composing a number of other tasks.
CompositeBehaviour
Models a simple task i.e. a task that is not composed of sub-tasks
SimpleBehaviour
OneShotBehaviour
CyclicBehaviour
Models an atomic task (its done() method returns true)
Models a cyclic task (its done() method returns false)
FSMBehaviour
registerState()
registerTransition()
SequentialBehaviour
addSubBehaviour()
ParallelBehaviour
addSubBehaviour()
Models a complex task whose sub-tasks corresponds to the activities performed in the states of a Finite State Machine
Models a complex task whose sub-tasks are executed sequentially
Models a complex task whose sub-tasks are executed concurrently

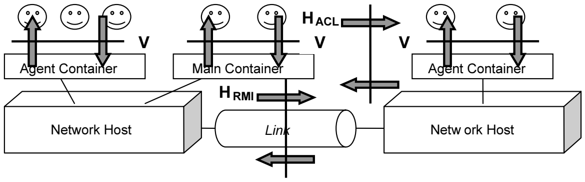
JADE Internals

- JADE is a MAS infrastructure.
  - Applications developed over JADE use agent-level modeling and programming.
  - Software components hosted by JADE exhibit agent-level features (they comply with the *weak agent* definition).
  - **JADE API is an agent-level API**.
- JADE is implemented in Java.
  - JADE applications integrate well with Java technology.
  - JADE runtime exploits object-oriented techniques.
  - **JADE API is an object-oriented API**.

JADE Content Processing



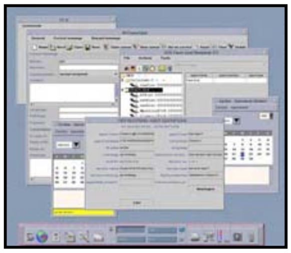
JADE Layered Architecture



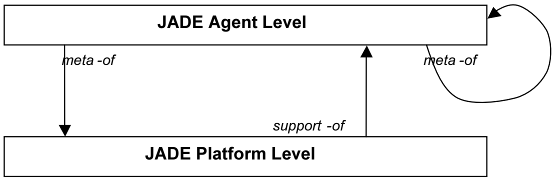
- JADE architecture is divided into ***two*** layers:
  - ***Platform layer*** (uses object-oriented concepts, distribution via RMI).
  - ***Agent layer*** (uses agent-level concepts, distribution via ACL).
- JADE architecture has two kind of interfaces:
  - ***Vertical interfaces*** (bidirectional connections between layers).
  - ***Horizontal interfaces*** (H_{RMI} at platform layer, H_{ACL} at agent layer).

JADE Support Tools

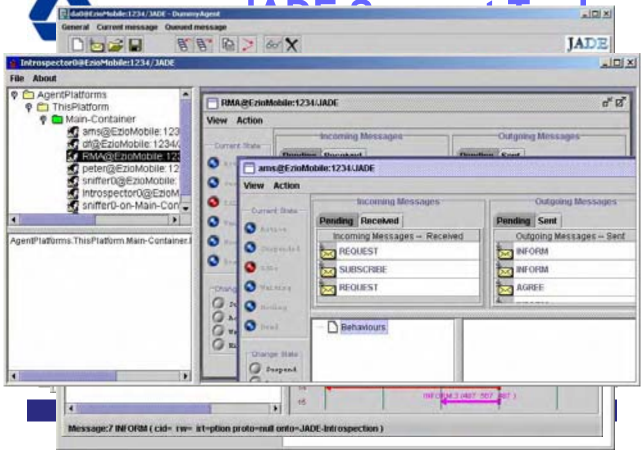
- Administration tools.
  - RMA Management Agent.
    - White pages GUI.
    - Agent life cycle handling.
  - Directory Facilitator GUI.
    - Yellow pages handling.
- Development tools.
  - DummyAgent.
    - Endpoint Debugger.
  - Message Sniffer.
    - Man-in-the-middle.



Inter-layer Relationships



- Def.: *X* ***meta-of*** *Y*: Layer X describes and possibly controls layer Y.
- Def.: *X* ***support-of*** *Y*: Layer X provides services to layer Y.
- Platform *support-of* Agent: It's the runtime system for agents.
- Agent *meta-of* Platform: Description with JADE ontologies.
- Agent *meta-of* Agent: It's a ***self describing*** layer.



Message:7 INFORM (cid= rw= irt=ption proto=null onto=JADE-Introspection)

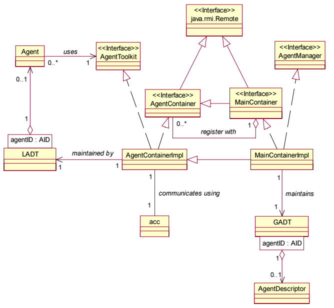
Summary on Multi-Agent Systems

An interesting technology!

Connects Artificial Intelligence and Distributed Systems.
Hides DS programming complexity.
Promotes loosely coupled, multi-authority systems.
Supported by an open standard (FIPA).
Integration across OSs, networks and languages.
A lot of free implementations available (e.g. JADE).

- Now, Agent Technology is *almost famous*.
  - Will it mainstream?
  - Will it replace Web Services? EJBs? .NET?

JADE Core Classes

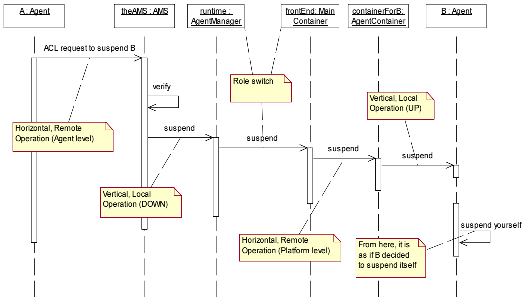


Any Order of Business

- Live Demo of JADE.
- Questions about JADE?
- …

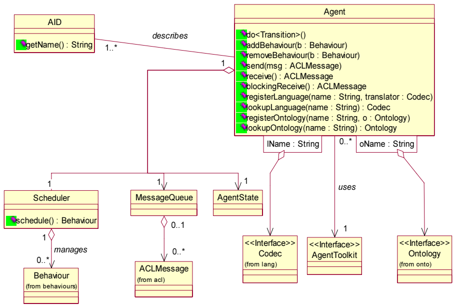
GAME OVER

Agent Suspension



A : Agent
theAMS : AMS
runtime : AgentManager
frontEnd: Main Container
containerForB: AgentContainer
B : Agent
ACL request to suspend B
verify
Role switch
Vertical, Local Operation (UP)
Horizontal, Remote Operation (Agent level)
suspend
suspend
suspend
suspend
Vertical, Local Operation (DOWN)
suspend yourself
Horizontal, Remote Operation (Platform level)
From here, it is as if B decided to suspend itself

JADE Agent Class



AID
getName() : String
describes
1..*
1
Agent
do<Transition>()
addBehaviour(b : Behaviour)
removeBehaviour(b : Behaviour)
send(msg : ACLMessage)
receive() : ACLMessage
blockingReceive() : ACLMessage
registerLanguage(name : String, translator : Codec)
lookupLanguage(name : String) : Codec
registerOntology(name : String, o : Ontology)
lookupOntology(name : String) : Ontology
lName : String
0..*
oName : String
uses
1
1
1
1
Scheduler
schedule() : Behaviour
MessageQueue
AgentState
1
manages
0..*
0..1
0..*
Behaviour
(from behaviours)
ACLMessage
(from acl)
<<Interface>>
Codec
(from lang)
<<Interface>>
AgentToolkit
<<Interface>>
Ontology
(from onto)