



# SENG 697 Agent-based Software Engineering

## Session 4 : Agent-based System Architecture and Organization

**Behrouz Far**  
Schulich of Engineering, University of Calgary  
far@ucalgary.ca  
<http://www.enel.ucalgary.ca/People/far/>



## Course Curriculum

### Overview of agent-based SE

Methodologies for agent-based  
analysis and design

Agent communication &  
knowledge sharing

Agent-based System  
Architecture & Organization

FIPA: Foundation for  
Intelligent Physical Agents

Principles of  
Object Technology

Other topics:  
Agent Interaction,  
Infrastructure, APIs,  
Performance metrics,  
Learning,  
Self-organizing systems  
etc.



## Contents

- **Session 4 : Agent-based System Architecture and Organization**
  - The Open Agent Architecture: A Framework for Building Distributed Software Systems
  - FIPA-OS Agent Development Framework
  - JADE: Java Agent Development Framework



## Agent Platforms

- **Question:** Do we really need an agent platform to implement a MAS?
- **Answer:** No. But it is good to have one because agent platforms provide *run-time environment* (i.e. implementation of infrastructure and services) needed for successful creation and execution of a MAS.



## The Open Agent Architecture: A Framework for Building Distributed Software Systems

D.L. Martin    A.J. Cheyer    D.B. Moran

Artificial Intelligence Center  
SRI International



## Review: Agent Properties

- Three properties of agents play a key role in agent-based analysis/design/simulation:
- The first is an agent's ability to be *autonomous* and independent.
- The second is *asynchronous message passing* in agents.
- The third is *social activity* and *reactivity*.





# Open Agent Architecture

- A distributed cooperative computing model
- The Open Agent Architecture (OAA) is a framework in which a community of software agents running on distributed machines can work together on tasks assigned by human or non-human participants in the community.
- Two ideas central to OAA
  - Distributed cooperation through delegation of tasks, data requests and triggers
  - High-level communication



# OAA: Features

- **Open:** agents can be created in multiple programming languages and interface with existing legacy systems.
- **Extensible:** agents can be added or replaced individually at runtime.
- **Distributed:** agents can be spread across any network-enabled computers.
- **Parallel:** agents can cooperate or compete on tasks in parallel.
- **Mobile:** lightweight user interfaces can run on handheld PDA's or in a web browser using Java or HTML and most applications can be run through a telephone-only interface.
- **Multimodal:** When communicating with agents, hand-writing, speech, pen gestures and GUIs can be combined in a natural way.



# What is an OAA Agent?

- In the Open Agent Architecture, an *agent* is defined as being *a software which conforms to the communication and functional standards imposed by the OAA.*
- In order for multiple agents to be able to cooperate, OAA agents can delegate and receive work requests through a *facilitator agent.*
- These requests are expressed in an *inter-agent communication language (ICL)* common to all agents.
- OAA agents possess
  - A dedicated set of functionalities based on their roles in the agency
  - A common set of functionalities, such as the ability to install local or remote triggers that is offered by the OAA platform.



# What is OAA Good for?

- Offering the ability to create programs from a *dynamic, virtual communities of services* which cooperate and interact in a flexible manner.
- Offering the ability to *build complex systems* in which there are many heterogeneous components, and in which flexibility and extensibility are important.
- Offering *delegation-based agent architecture* which is a solution for adaptable, incrementally extendable systems.
- Offering *advanced human-computer interaction* in which humans communicate easily and naturally with the community of agents.



# Distributed Computing Models

- Distributed computing models permit computing components to be spread across multiple platforms.
- **Various distributed computing models:**
  - Distributed Objects, Conversational (e.g., KQML based), Jini, Blackboard, Mobile objects, Publish and Subscribe, Delegated Computing Model, Cloud Computing, etc.
- Two areas of distinction among these models:
  - Are *interactions* among components are preconfigured (hard-wired)?
  - Where *control* for using the component or service lies (e.g., requester/client, provider/server, mediator)?



# 1. Distributed Objects

- Common distributed object technologies are
  - OMG's Common Object Request Broker Architecture (CORBA)
  - Microsoft's DCOM and its variations
- In both, control among components lies solely with the requesting object, that is, an explicit method call from a requesting object using a predefined interface specification initiates service access
- The service is provided by a remote object through a registry that finds the object, and then mediates the request and its response





# Distributed Objects: CORBA

## Common Object Request Broker Architecture (CORBA)

CORBA allows objects find each other and coordinate their behavior on a **common object bus**

### Advantages:

- Client-server architecture middleware
- Using objects as a metaphor for using existing applications (encapsulation)
- Ideal for component-based applications



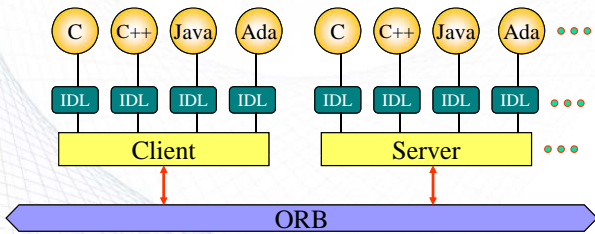
# Distributed Objects: Terminology

- OMG:** Object Management Group
- IDL:** Interface Definition Language
- ORB:** Object Request Broker
- CORBA:** Common Object Request Broker Architecture
- Component (distributed object):** An object with its interface defined in IDL
- IIOP:** Internet Inter ORB Protocol



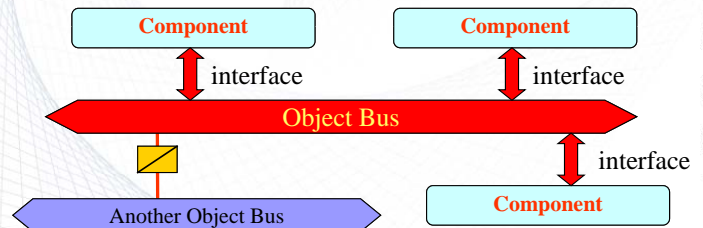
# Distributed Objects: CORBA

- IDL defines components' boundary, i.e., interface with other components
- All components on ORB use IDL as their interface
- Every object (distributed object) is a component



# Distributed Objects: CORBA

- An architecture for interoperability: Allowing components (object + IDL) find each other on a common bus



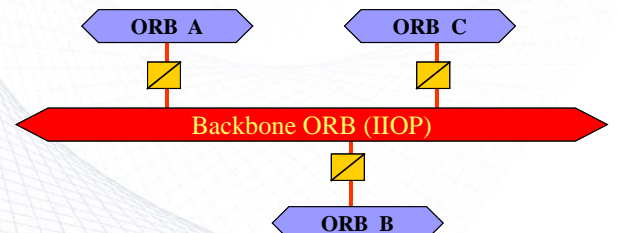
# Distributed Objects: CORBA

- OMG providers supply **components + description of interface** of their products
- An object requesting something from another object knows the target object's **interface**
- All the interfaces are defined and stored in CORBA **interface repository**. The definition helps objects find each other dynamically
- The OMG providers, specify which objects and methods are available, and which interface adapters are supported



# Distributed Objects: CORBA

- IIOP:** Internet Inter ORB Protocol
- IIOP is basically TCP/IP with some CORBA-defined message exchanges
- Any ORB can connect with the universe of ORBs by translating requests to and from the IIOP backbone

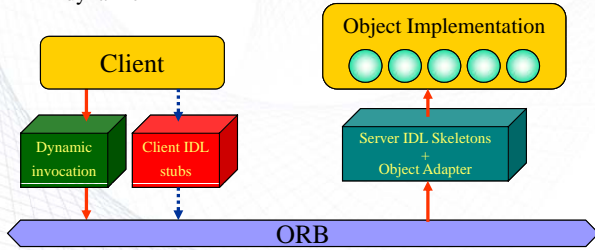




## Distributed Objects: CORBA

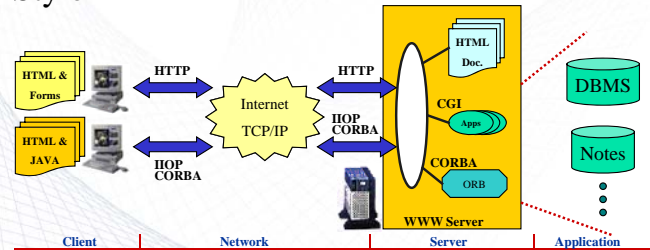
### Static and Dynamic methods

- Two types of client-server invocation: *static* and *dynamic*
- The server is not aware whether the invoked method is static or dynamic



## Distributed Objects: CORBA

- Object Request Broker (ORB)
- Client-Server Computing: IIOP, CORBA Style



## Distributed Objects: Critics

- Advantage:** Distributed object models offer a powerful paradigm for creating networked applications composed of objects potentially written in different programming languages
- Disadvantage:**
  - Service access control lies solely with the client
  - Hard-coded communication interactions make it difficult to reuse an object in a new application without bringing along all services on which it is dependent, and reworking the system to incorporate new services that were not initially foreseen is a complex task



## 2. Conversational Agents

- Conversational Agent technologies model communication and cooperation among autonomous entities through message exchange based on variations of the *speech act theory*.
- Representative technology:** Knowledge Query and Manipulation Language (KQML), in conjunction with the Knowledge Interchange Format (KIF).



## 2. Conversational Agents (cont'd)

- What is needed to interact and interoperate?
- A representation language → **KIF**
- A communication language → **KQML**
- Sharing content requires more than a representation language and a communication language**
- Sharing the content → **Ontology**



## Conversational Agents: Critics

- Service access control lies with the client, which requests a service from a service broker (facilitator, name server, yellow pages) and then initiates peer-to-peer communication with the provider at an address provided by the broker
- Although language-enriched interchanges occur, the agents suffer from the same restriction as distributed objects in that the interactions among components are hard-coded in the requester, thus making services inflexible and difficult to reuse and extend





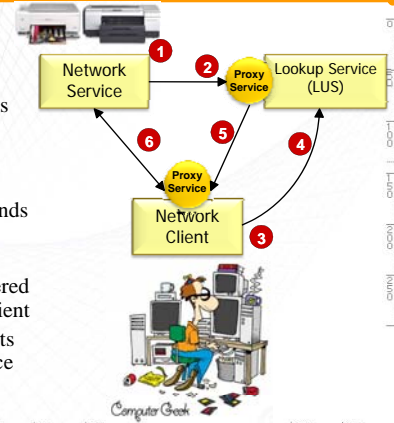
### 3. Jini

- Provides reliable utilization of services in an unreliable environment and allows users to easily access services available on the network
- Architecture for easy addition and removal of services
- Common access framework for users of services.
- Service provider specified communication protocol between users and services



### 3. Jini (cont'd)

- Discover:** Network service discovers available lookup services (LUS)
- Join:** Network service sends Service proxy to LUS
- Discover:** Network client discovers available LUS
- Lookup:** Network client sends a request to LUS to find desired services
- Receive:** LUS sends registered service proxy to network client
- Use:** Network client interacts directly with network service via service proxy



### Jini: Critics

- Control for resource access lies with the client who requests a service based on type and attributes from a Lookup Service (LUS) that holds a collection of Service Object proxies (Java object and methods) and attributes posted by providers.
- Clients filter responses from the LUS, download the proxy for the selected service, and invoke remote methods within the provider to obtain the service.
- Although the affordance of downloading the interface between service requester and provider permits dynamic and extensible assembly of resources, Jini's model still places burden and responsibility for selecting, acquiring, and managing access with the client. ← possible problem for light weight clients

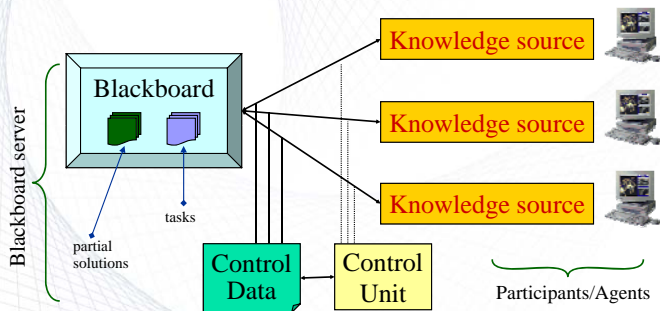


### 4. Blackboard

- Blackboard approaches allow multiple participants (agents) to communicate by reading requests from and writing information to a global data store.
- Each participant agent has expertise in its own field, and has a kind of problem solving knowledge (*knowledge source*) that is applicable to a part of the problem, i.e., the problem cannot be solved by an individual participant only
- Agents communicate strictly through a common *blackboard* whose contents is visible to all agents
- When a partial problem is to be solved, candidate agents that can possibly handle it are listed up
- A *control unit* is responsible for selecting among the candidate agents to assign the task to one of them



### 4. Blackboard (cont'd)



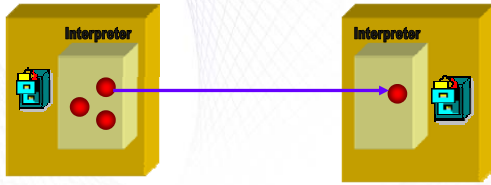
### 4. Blackboard (cont'd)

- Advantages:**
  - Suitable when there are diverse sources of input data
  - Suitable for physically distributed environments
  - Suitable for scheduling and postponement of tasks and decisions
  - Suitable for team problem-solving approaches as it can be used for posting problem subcomponents and partial results
- Drawbacks:**
  - Expensiveness
  - Difficult to determine partitioning of knowledge
  - Control unit can be very complex





## 5. Mobile Objects (Agents)



- Objects move under their own authority (e.g., Web robots, Voyager, Aglets)



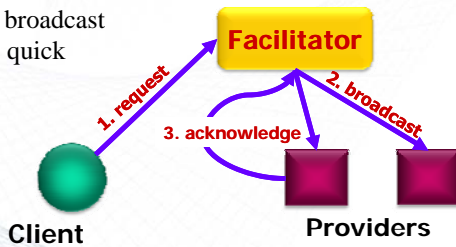
## 5. Mobile Objects (Agents)

- Advantages:**
  - Network bandwidth for certain classes of problems
  - Parallelism - many objects can be spawned
- Drawbacks:**
  - Programmatically specify where to go and what to do, through a known interface
  - Little automated support for inter-object cooperation
  - Programming language specific (non-heterogeneous)



## 6. Publish & Subscribe

- Publish and Subscribe approaches use a **centralized broker** (i.e., facilitator) as a clearinghouse for requests and information. Clients issue a request to the broker that broadcasts it to available providers; their responses are reflected through the broker to the client. This approach is well-suited to time-critical problems, as the broadcast model facilitates quick responses.

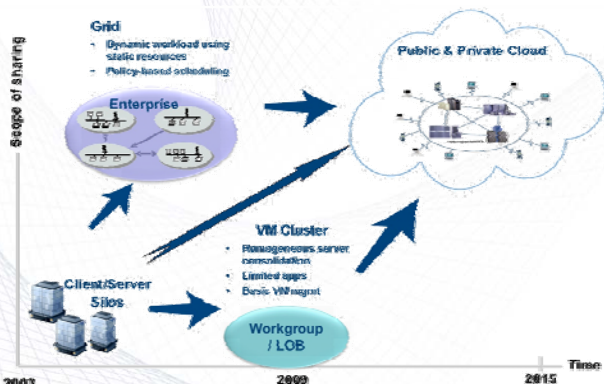


## 6. Publish & Subscribe: Critics

- Advantages:** ability to enable dynamic and flexible composition of distributed components: **interaction among components is not predefined at code time or tightly bound at runtime.**
- Drawbacks:** with this flexibility comes a potential inherent disadvantage because it does not provide programmatic control for guiding operation, and at times this control is needed or desired (e.g., to take a provider that best meets known requirements).

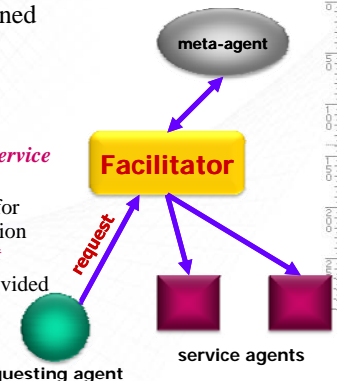


## 7. Cloud Computing



## OAA Approach

- Agent interactions are determined by a cooperative process that combines:
  - advice or constraints from the **requesting agent**
  - specifications provided by the **service agents**
  - domain-independent strategies for coordinating interagent cooperation provided by the **facilitator agent**
  - domain-relevant knowledge provided by **meta-agents**





## OAA Compared to CORBA

- OAA's claim is its use of a **delegated computing** model, where the control for how components interact with each other is the result of a cooperative process among at least four processes:
  - The **requester** can choose to partially specify how a goal should be accomplished by the community in a rich tasking language. Control parameters not explicitly stated by the requesting agent will be filled in by other members of the community.



## OAA Compared to CORBA (cont'd)

- Service providers**, during the registration process with a facilitator agent, can specify test-condition code and performance estimation values which will both impact how and under what circumstances the agent receives work.
- The **facilitator** agent possesses domain-independent routing strategies for parceling out work to the agent community based on an incoming request and service providers' capabilities descriptions.
- Collections of **meta-agents** (domain experts) can collaborate to provide the facilitator with domain-specific knowledge that guides how tasks are performed by the overall community.



## OAA & Distributed Objects

- At least two characteristics distinguish **OAA agents** from **distributed objects**:
  - Higher level communication**: Programs accessing distributed objects must use low-level remote procedure call or method invocation, agent communication takes place at a higher-level. The language used by the agents to communicate must be powerful enough to describe natural language requests, so that humans can communicate easily with other agents.



## OAA & Distributed Objects

- At least two characteristics distinguish **OAA agents** from **distributed objects**:
  - Proactivity**: Distributed objects are generally passive sets of data and library routines. Agents may have their own goals and may **reason about the interactions** with the other agents and provide information when appropriate. In addition, agents can receive or make requests to install **triggers** or **monitors** of various kinds, keeping track of events for another agent or human peer.



## OAA & Distributed Computing

- OAA is similar to the distributed computing models in that it encourages creation of networked applications like Distributed Objects, permits rich and complex interactions like Conversational Agents, and enables building dynamic, flexible, and extensible communities of components like Jini, Blackboard, and Publish and Subscribe.
- A distinguishing feature of OAA is its **delegated computing model** that enables both human users and software agents to express their requests in terms of **what** is to be done without requiring specification of **who** is to do the work or **how** it should be performed.



## OAA & Distributed Computing

- Example**  
*"When a message arrives about security, notify me immediately."*  
 A requester delegates control for meeting a goal to the Facilitator agent (a specialized service provider agent within OAA) that coordinates the activities of agents for the purpose of achieving higher-level, often complex problem-solving objectives.
- OAA's delegation model relieves human and software agents from some of the responsibilities of interfacing, task planning, and execution monitoring.





## OAA & Distributed Computing

- OAA's delegation model has several benefits:
  - Reducing complexity** for users and agents. Requesters need only specify the work to be done and advice on its execution. Agents then focus on performing their specialized task, not on coordinating execution and results.
  - Precipitating a more **open and dynamically extensible computing style** wherein agents written in many languages and styles can work together. New or different agents can be added or replaced on the fly without requiring reprogramming to take advantage of their capabilities.
  - Encouraging reuse** across applications and domains because interagent interactions are not pre-defined and their interfaces are not hard-coded.

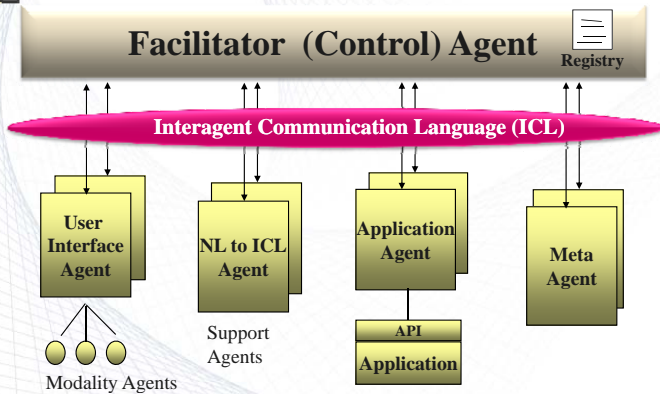


## OAA & Distributed Computing

- The Facilitator meets these objectives by making use of knowledge distributed in four locations in OAA:
  - the requester, which specifies a goal to the Facilitator and provides advice on how it should be met,
  - providers, who register their capabilities with the Facilitator, know what services they can provide, and understand limits on their ability to do so,
  - the Facilitator, which maintains a list of available provider agents and a set of general strategies for meeting goals
  - meta-agents, which contain domain- or goal-specific knowledge and strategies that are used as an aid by the Facilitator
- This knowledge is employed to foster cooperation among a set of OAA agents. The Facilitator matches a request to an agent or agents providing that service, delegates the task to them, coordinates their efforts, and delivers the results to the requester. In addition to delegation, OAA also provides the ability to make direct calls to a specific agent (like Distributed Objects and Conversational Agents) and to broadcast requests (like Publish and Subscribe).



## OAA Architecture



## OAA Architecture (cont'd)

- The facilitator is a specialized agent that is responsible for coordinating agent communications and cooperative execution of tasks.
- Client (non-facilitator) agents may be categorized as follows:
  - User interface agents** accept multimodal input and present results.
  - Natural language agents** translate users' requests into ICL.
  - Application agents** provide services. Application Agents are frequently wrapped versions of legacy programs.
- Meta agents** apply domain knowledge to help coordinate the efforts of other agents.
- All OAA agents publish their capabilities and communicate requests among themselves using the **Inter-agent Communication Language (ICL)**.



## Facilitator agent

- A **Facilitator** is an agent whose capabilities involve distributing and coordinating tasks among a number of client agents.
- A Facilitator performs three tasks:
  - Maintaining a registry** of agent service and data declarations.
  - Planning an execution strategy** for a given query.
  - Executing the plan**, that is, breaking tasks into subtasks, matching subtasks to service providers, and routing and collecting information among the participants.



## Facilitator agent: Properties

- Facilitator agent is domain independent**, i.e., the same facilitator can be used for all applications (although different facilitators may exist, providing varying levels of sophistication in inter-agent management).
- Facilitator agent **possesses a number of services and domain independent strategies** for coordinating distributed processing.
- Domain-specific knowledge can be encoded in meta-agents or mediator agents, which apply whatever "intelligence" they possess (e.g. neural networks, learning, planning, rules) to help the facilitator make key decisions.





## Agents Communication

- All inter-agent communication is expressed using an OAA-specific language called the Inter-agent Communication Language (ICL).
- For OAA's ICL, has a logic-based language similar to Prolog, because for the research purposes, it was important that the language the agents speak could represent and easily translate back and forth to human (or natural) language.
- The most important characteristic of the ICL is that it separates out the "what" from the "how".
- Control parameters specifying how the task should be routed, who should perform the task, and so forth, are optional and separate from the task description itself.



## Triggers

- Triggers in OAA (also called "monitors" or "watchdogs") allow agents to persistently keep track of different types of events in agent community and take some action if a condition becomes true.
- There are four types of triggers:
  - Task triggers:** watches agent-supplied conditions, such as "when email arrives about XYZ, notify me."
  - Communication triggers:** looks for specific low-level agent-communication events, such as "when an agent-disconnect event is sent, record time."
  - Data triggers:** keeps track of changes to information stored, such as "when the cash amount becomes minus, notify me."
  - Time triggers:** time-based events, such as "every Thursday at 2PM from now until Christmas, send me a meeting reminder."



## Support Agents

- The only agent used in every agent application is the facilitator.
- A number of support agents exist that are domain independent and can be useful as participants in a given application domain.
- Examples:**
  - Natural Language Parsing Agent; Speech Recognition Agent; Natural language generation agent; Text-To-Speech Agent
  - Database Agent
  - Phone Agent to permit remote mobile access to speech recognition
  - Monitor agent that graphically displays and records an agent community and their communications.
  - Debug Agent which is a generic user interface that lets a programmer send messages to the agent community using either ICL or natural language.



## Creating New Agents

- To create a new agent, an agent developer usually performs the following tasks.
  - Specify what services an agent can provide and what services an agent needs, expressed in the syntax of the Inter-agent Communication Language (ICL).
  - Write the code for the agent, implementing the capabilities declared solvable by the agent.
  - Document the agent's functionalities, usually in Internet-readable form so as to provide easy access to the documentation through Web browsers and Internet search tools.
  - Add vocabulary for the agent to the appropriate service agent.
  - Include the agent as a community member for agent-based systems which might make use of its services.



## Agent Development Tool

- The Agent Development Tools (ADT) attempts to guide an agent programmer through the steps required for developing an agent, automating the process as much as possible.
- ADT has yet to be updated to work well with OAA version 2.x.
- The interface will be ported to Java, to give the ADT a more web-centric approach, and integrate ADT with the OAA Community pages.



## Agent Development Tool

Tools are implemented themselves in OAA  
Guide user through process of creating an agent:

- Definition of capabilities
- Documentation management (publication on Web)
- Code generation of agent template
- Definition of NL vocabulary
- Update NL & speech recognition systems
- Assembly of multiagent projects

Runtime tool for launching and monitoring agent communities





## Platforms

- The OAA 2.x Facilitator is distributed in binary form for Windows, Solaris, and Linux platforms.
- In addition, user interface agents can be written using JAVA or HTML, providing platform independent access to agent applications.
- Currently, the Facilitator agent, which is required for all agents systems, runs only on UNIX workstations or PCs, so at least one of these machines is required for OAA development and deployment.



## Development Library

- In OAA 1.0, agent libraries and tools exist to create agents in the following programming languages:  
Prolog, C, C++, Perl, Lisp, Visual Basic, Delphi, JAVA
- OAA 2.x supports Prolog, ANSI C/C++ and Java
- January 2007 : OAA 2.3.2 released.



## Future Directions

- Many interesting technologies have already been integrated into the OAA framework, and there are still many yet to incorporate.
- A number of systems have been built to demonstrate the capacities of these technologies.
- The current directions of the OAA group has been focused on bringing together advanced computer technologies that have currently existed only in isolation, and through the combination of these technologies, making computers smarter, easier and friendlier for use for the average human being.



## Demonstration Systems

- Unified Messaging
- Automated Office



## Developing Tools : OAA

### OAA

Open Agent Architecture (OAA) from SRI enables the construction of agents that is based on the logic based InterAgent Communication Language and runs on top of CORBA

Description	Property
Communication Infrastructure	Message based Connected Point-to-point Push Synchronous
Directory service	ORB
Message Protocol	IIOIP
Mediation Service	Facilitator
Security Service	None



## More info

- See the following tutorial:
  - <http://www.ai.sri.com/~oaa/sri-private/doc/tutorial/index.html>
- OAA Technical White Paper:
  - <http://www.ai.sri.com/~oaa/whitepaper.html>





# want a break?



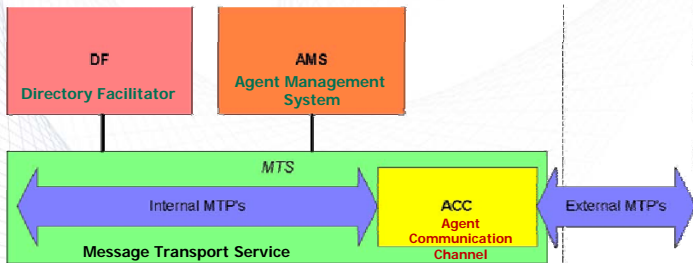
## FIPA Agent Platform: FIPA-OS

# FIPA-OS v2.2.0



## FIPA-OS Agent Framework

- The FIPA reference model illustrates the core components of the FIPA-OS distribution.
- AMS, MTS and DF form the Agent Platform (AP). These are mandatory, normative components of the model.



## FIPA-OS Agent Framework

- Directory Facilitator (DF)** provides naming (yellow pages) services to other agents.
- Agent Management System (AMS)** provides platform management functionality, such as monitoring agent lifecycles and ensuring correct behavior of entities within, and upon, the platform.
- Agent Communication Channel (ACC)** is a lower-level entity that is part of the MTS. ACC supports interoperability both within and across different platforms; therefore, it is viewed as a component of the MTS.
- Message Transport Service (MTS)** provides a message routing service for agents on a particular platform.



## Starting the Agent Platform

- Use Agent Loader to start all agents. This enables agents to be managed by a human user as required.
- To start the Agent Loader (plus naming services and ACC, dependant upon your configuration) use:  
G:\ >StartFIPAOS
- This will result in the Agents that have been specified in the loader profile being started and a loader GUI being displayed. This will show the Agents that are running as a list on the left-hand side of the window, and a list of ready-to-start Agents on the right.
- The FIPA-OS build includes an IOTestAgent, which enables ACL messages to be manually composed and sent.



## Agent Loader

- This will show the Agents that are running as a list on the left-hand side of the window, and a list of ready-to-start Agents on the right:





## DF Cross Registration

- When the **Directory Facilitator (DF)** starts, it creates a GUI to enable registration of the DF with other DF's and vice-versa. This enables federated searches to be propagated by and to the DF.



## Swing DF Interface Agent

- Swing DF interface agent is an agent that will give a user a way to interact with DFs. Swing DF GUI supports following:
  - Subscribing to a home platform FIPA-OS DFs, so that the GUI will always show all the agents registered on those DFs.
  - Querying remote FIPA-OS DFs – the information will be updated when Refresh is invoked.
  - Viewing agents' DF agent descriptions.



## User Constructed Agents

- In addition to the mandatory components of the FIPA Reference Model, the FIPA-OS distribution includes an empty template for an agent, i.e., **Agent Shell**.
- Multiple agents can be implemented, using the Agent Shell as a template.
- These agents can communicate with each other using the FIPA-OS support facilities.



## User Constructed Agents

- How to create user agents?
- Use a built-in empty template for an agent, i.e., **Agent Shell**.
- Use **JESS Agent**: an special Agent Shell that incorporates the reasoning abilities provided by JESS (Java Expert System Shell).

(<http://www.jessrules.com/>)

latest version (2008): ver. 7.1



## Extending the Generic Agent

- Start with a template for a Generic Agent.
- The template is provided.
- Extend it with defining and adding
  - Task
  - Listener Task
  - Decompose Task (if needed)



## 1. Task

- A Task encapsulates some functionality that forms a logical unit of work (i.e. search the DF, conduct a negotiation with another agent, or wait for an amount of time). Tasks can also be composed of sub-tasks (child-Tasks) to enable more complex “units of work” to be subdivided into smaller chunks of functionality.
- Tasks are based upon event-based processing, where events are delivered by dynamic invocation of various methods. Once a Task has been initialized, its `startTask()` method is invoked.





## 2. Listener Task

- Every agent that uses Tasks should have a `ListenerTask()`.
- The `ListenerTask()` will deal with any incoming messages which do not form part of a conversation that is being dealt with by another Task.

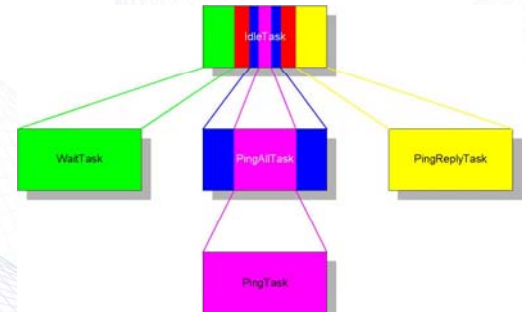


## 3. Decompose Tasks

- Decomposing a task allows for implementing more complex behaviour.

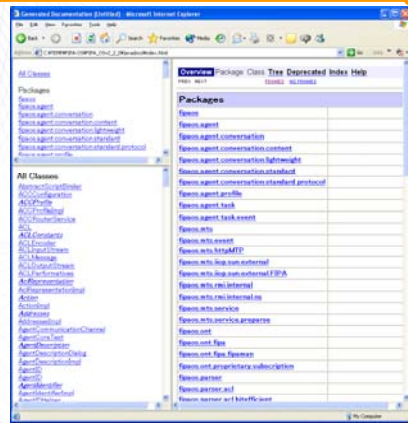
### Example:

Ping task is decomposed to 5 child-tasks.



## FIPA-OS Java Docs

- All the classes and packages are documented as Java standard documentation.



## Example: eMarket Agency

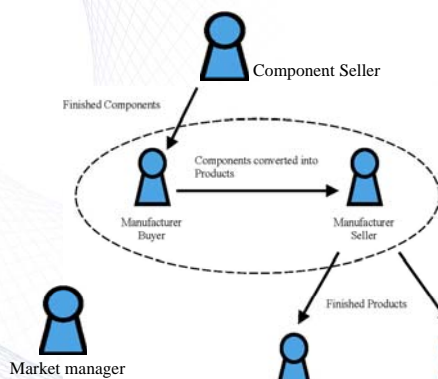
### Scenario:

- The eMarket scenario is a 3-tiered marketplace that contains the following agent types:
  - Component Seller Agent** constructs finished components from raw materials and sells these components to a Manufacturer Buyer Agent.
  - Manufacturer Buyer Agent** purchases finished components from a Component Seller Agent.
  - Manufacturer Seller Agent** uses finished components to create finished products, which it sells to a Retail Buyer Agent.
  - Retail Buyer Agent** purchases the finished products from a Manufacturer Seller Agent in order to sell them to consumers.
  - Market Manager Agent** is responsible for controlling the marketplace.



## eMarket Agency (cont'd)

- Agent types based on identified roles



## eMarket Agency (cont'd)

### Assumptions:

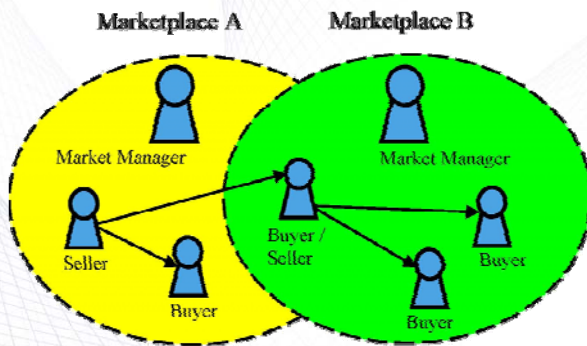
- A subscription-based framework for creating communities of agents.
- When an agent joins the marketplace, it is initially informed about all of the agents that are already registered in the marketplace.
- When another agent joins or leaves the marketplace, the **Market Manager Agent** informs all subscribed agents of this event so that subscribers always have an up-to-date view of the marketplace.
- When an agent leaves a marketplace, it forgets all knowledge of agents in the marketplace.
- An agent can be subscribed to more than one marketplace and may have different roles in different marketplaces.
- Each marketplace is managed by a different **Market Manager Agent**.





## eMarket Agency (cont'd)

- Marketplace model (added roles)



## eMarket Agency (cont'd)

- Next we must define
  - Goals (objectives)
  - Strategies
  - Tasks
 for each agent.



## 1. Goals

### Example:

- The goal of a **Buyer agent** is to develop a buying strategy to purchase enough items to achieve its target and still remain within its budget of credits. A successful **Buyer agent** will contain strategies that minimize a buyer's average price paid per item.
- The goal of a **Seller agent** is to develop a selling strategy to sell enough items to achieve its target number of credits. A successful **Seller agent** will contain strategies that maximize a seller's average price received per item.



## 2. Strategies

### Example:

- Never allow buyer agents to spend more credits than they own.
- Never allow seller agents to sell more items than they own.



## eMarket Agency (cont'd)

- FIPA-OS Tutorials:  
Source code and detailed implementation of sample projects are provided



## JADE: Java Agent Development Framework

<http://jade.tilab.com/>  
Telecom Italia Lab

JADE: A White Paper  
F. Bellifemine, G. Caire, A. Poggi, G. Rimassa



## What is JADE?

- JADE is the middleware developed by TILAB for the development of distributed multiagent applications based on the peer-to-peer communication architecture.
- Intelligence, information, resources and control can be fully distributed on mobile or fixed terminals.
- The environment can evolve dynamically with peers (agents), that appear and disappear in the system according to the needs and the requirements of the application environment.
- JADE is free and open-source software.



## What is JADE? (cont'd)

- JADE is developed in Java and is based of the following driving principles:
  - Interoperability:** JADE is compliant with the FIPA specifications and can interoperate with other FIPA compliant agents.
  - Portability:** JADE provides a set of APIs that are independent from the underlying network and Java version. JADE run-time provides the same APIs both for the J2EE, J2SE and J2ME environment. Developers can decide the Java run-time environment at deploy-time.
  - Easy to use:** The complexity of the middleware is hidden behind a simple and intuitive set of APIs.
  - Pay-as-you-go philosophy:** Programmers do not need to use all the features provided. Features that are not used do not require programmers to know anything about them, neither add any computational overhead.



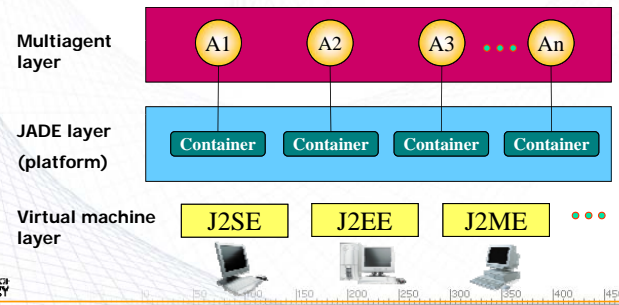
## What is JADE? (cont'd)

- JADE includes both the *libraries* (i.e. the Java classes) required to develop application agents and the *run-time environment* that provides the basic services and that must be active on the device before agents can be executed.
- Each instance of the JADE run-time is called **container** (since it contains agents). The set of all containers is called **platform** and provides a layer that hides to agents (and to application developers also) the complexity and the diversity of the underlying tiers (hardware, operating systems, types of network, JVM).



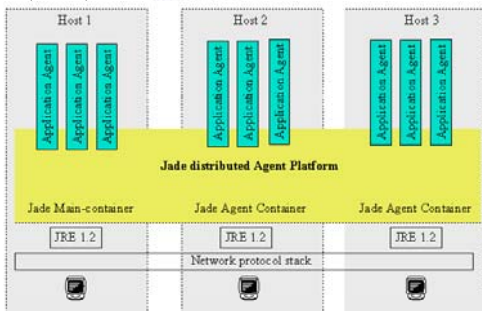
## JADE Architecture /1

- JADE architecture is based on coexistence of several VMs and communication relies on Java RMI between different VMs, event signaling within a single VM, and IIOP between different agent platforms.



## JADE Architecture /2

- The agent platform can be distributed on several hosts. Only one Java application, and therefore only one Java Virtual Machine (JVM), is executed on each host.



## JADE: Agent Communication

- All agent communication is performed through message passing.
  - Lightweight transport of ACL messages inside the same agent platform:** as messages are transferred encoded as Java objects, rather than strings.
  - FIPA compliant message passing:** When sender or receiver do not belong to the same platform, the message is automatically converted to/from FIPA compliant string format. This conversion is hidden to the agent programmers.





## JADE Features

- **FIPA-compliant agent platform:** It includes the AMS (Agent Management System), the DF (Directory Facilitator), and the ACC (Agent Communication Channel), all automatically activated at the agent platform start up.
- **Distributed agent platform:** The agent platform can be split on several hosts (provided that there is no firewall between them). Only one Java application, and therefore only one Java Virtual Machine, is executed on each host. Agents are implemented as one Java thread and Java events are used for communication between agents on the same host. Parallel tasks can be still executed by one agent, and JADE schedules these tasks efficiently.



## JADE Features (cont'd)

- **FIPA-compliant DF:** is provided.
- **Programming interface:** to simplify registration of agent services with one, or more, domains (i.e. DFs).
- **Transport mechanism:** and interface to send/receive messages to/from other agents.
- **FIPA-compliant MTPs (Message Transport Protocol):** to connect different agent platforms.
- **Library of FIPA interaction protocols:** is provided.



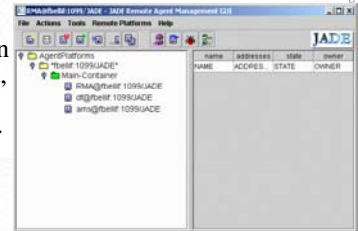
## JADE Features (cont'd)

- **Automatic registration:** of agents with the AMS.
- **FIPA-compliant naming service:** at start up agents obtain their GUID (Globally Unique Identifier) from the platform.
- **Graphical user interface:** to remotely manage the life-cycle of agents and agent containers.
- **Graphical tools:** for debugging the multi-agent system, like the *dummy agent* to send/receive ACL messages and the *sniffer agent* to monitor the ongoing communication between several agents.



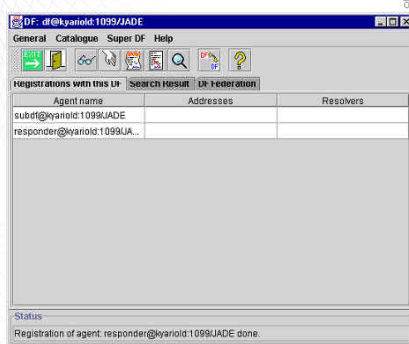
## Agent Platform

- The agent platform provides a GUI for the remote management, monitoring and controlling of the status of agents.
- The GUI allows also to create and start the execution of an agent on a remote host, provided that an agent container is already running.
- The GUI itself has been implemented as an agent, called RMA (Remote Monitoring Agent).
- All the communication between agents and this GUI is done through ACL via an ad-hoc extension of the Fipa-agent-management ontology.



## Directory Facilitator (DF)

- A GUI of the DF can be launched from the Tools menu. By using this GUI, the user can interact with the DF: view the descriptions of the registered agents, register and deregister agents, modify the description of registered agent, and search for agent descriptions.



## Languages Support

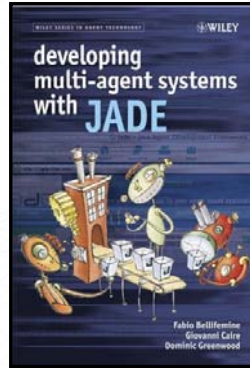
- JADE agents can be written in Java 1.2 or better
- JADE agents can interoperate with JESS agents
- It is possible to implement applications with JADE 3.x and Microsoft .NET





## More Information

- URL: <http://jade.tilab.com/>
- The latest version of JADE is JADE 3.7 + WADE 2.5 (released on July 2009)
- “Developing Multi-Agent Systems with JADE” Fabio Luigi Bellifemine, et al, (April 2007)



## Other Development Platforms



## A Few FIPA Compliant Platforms

Publicly available implementations of FIPA standards

- Agent Development Kit
- April Agent Platform (AAP)
- Comtec Agent Platform
- FIPA-OS
- Grasshopper
- JACK Intelligent Agents
- Java Agent Development Environment (JADE)
- Lightweight Extensible Agent Platform (LEAP)
- ZEUS
- Etc.



## Some Non-FIPA Compliant Platforms

- IBM’s Aglets
- Agent Building Environment (**ABE**)
- Java Agent Template (**JAT**)
- Java Expert System Shell (**JESS**)
- **Voyager** Object Request Broker for Java agents
- And many others



## Developing Tools : ABE

### ■ ABE

Agent Building Environment (ABE) from IBM written in C++ and Java. Agents in ABE have rule-based reasoning and interfaces to HTTP, NNTP and SMTP

Description	Property
Communication Infrastructure	Message based Connectionless Multicast Push Asynchronous
Directory service	None
Message Protocol	HTTP, NNTP, SMTP
Mediation Service	None
Security Service	None



## Developing Tools : JAT

### ■ JAT

The Java Agent Template (JAT) and JAT-Lite from Stanford University enable Java agents to communicate over a LAN via KQML

Description	Property
Communication Infrastructure	Message based Connected Point-to-point Push Synchronous
Directory service	Nameserver
Message Protocol	KQML
Mediation Service	None
Security Service	None





## Developing Tools : JESS

### JESS

Java Expert System Shell (JESS) is basically CLIPS in Java; it enables solitary reasoning agents to be constructed

Description	Property
Communication Infrastructure	None
Directory service	None
Message Protocol	None
Mediation Service	None
Security Service	None



## Developing Tools : VOYAGER

### VOYAGER

Voyager from ObjectSpace Inc. provides an Object Request Broker (ORB) for Java agents

Description	Property
Communication Infrastructure	Message based Connectionless Point-to-point Push Asynchronous
Directory service	ORB
Message Protocol	IOP
Mediation Service	None
Security Service	None



Agents!

