USD

Odense
09 August 2010

MAS Course 4

Yves Demazeau
Yves.Demazeau@imag.fr

SCHEDULE OF THE COURSE + EXAMINATION

MAS 01	04 Aug.	Introduction, Methodology, Agents
MAS 02	05 Aug.	Agents, Environments, Interactions
MAS 03	06 Aug.	Dynamics, Organisations, Example
MAS 04	09 Aug.	Development, Deployment, Example
MAS 05	10 Aug.	
MAS 06	11 Aug.	

attendance ; handouts ; individual work
[Ferber 95] [HERMES 01] [OFTA 04]

MAS Ex.	13 Aug.	Written Control and Oral Delivery

DEVELOPMENT

Evolution of Programming Paradigms

1950's
- Machine and assembly language

1960's
- Procedural programming

1970's
- Structured programming

1980's
- Object-Based programming, Declarative programming

1990's
- Frameworks, design patterns, scenarios, and protocols

2000's
- Agents… Multi-Agent Systems…

…

Features of Languages and Paradigms

Concept	Proc. L.	Object L.	Agent L.
abstraction	type	class	society
block	data	object	agent
model	procedure call	method message	perceive reason / act
paradigm	tree of procedures	interaction patterns	cooperative interaction
architecture	functional decomposition	inheritance polymorphism	managers assistants, peers
modes of	coding	designing and using	enabling and enacting
terminology	implement	engineer	activate

Agent Oriented Programming [Shoham 93]

A complete AOP system will include three primary components

- a restricted formal language with clear syntax and semantics for describing mental state: the mental state will be defined uniquely by several modalities, such as belief and commitment
- an interpreted programming language in which to define and program agents, with primitive commands such as REQUEST and INFORM: the semantics of the language will be required to be faithful to the semantics of the mental state
- an "agentifier", converting neutral devices into programmable agents.

AOP vs. OOP : introduction

- The use of mental states is to design the computational system
- From an engineering point of view, AOP can be viewed as a specialization of OOP, in the original sense of Hewitt's Actors model
- OOP proposes viewing a computational system as made up of modules that are able to communicate with one another and that have individual ways of handling incoming messages
- AOP specializes the framework by fixing the state (mental state) of the modules (agents) to consist of components such as beliefs (about the world, about themselves, about one another), capabilities, and decisions, each of which enjoys a precisely defined syntax.

Components of mental state

Components of mental state
- the future is determined by two factors : past history and current actions of agents.

The actions of an agent are determined by its decisions, or choices

Decisions are logically constrained, though not determined, by the agent's beliefs
- These beliefs refer to the state of the world, to the mental state of other agents, and to the capabilities of this and other agents

Basics : beliefs, capabilities, obligations (decision is simply an obligation to oneself)

A language for belief, obligation, and capabilities

time / action	holding(robot,cup)^t
belief	$B^t_a\varphi$ φ a (recursive) sentence
obligation	$OBL^t_{a,b}\varphi$
decision	$\equiv_{def}$ $OBL^t_{a,a}\varphi$
capability	$CAN^t_a\varphi$
	$ABLE_a\varphi \quad =_{def} CAN^{time(\varphi)}_a\varphi$ time(φ) the outermost time occurring on

Properties of the various components

internal consistency	$\forall a,t \{\varphi:B^t_a\varphi\}$ is consistent $\forall a,t \{\varphi:OBL^t_{a,b}\varphi\}$ is consistent
good faith	$\forall t,a,b,\varphi\ OBL^t_{a,b}\varphi \supset B^t_a((ABLE_a\varphi)\wedge\varphi)$
introspection	$\forall t,a,b,\varphi\ OBL^t_{a,b}\varphi \equiv B^t_a OBL^t_{a,b}\varphi$ $\forall t,a,b,\varphi\ \neg\ OBL^t_{a,b}\varphi \equiv B^t_a \neg\ OBL^t_{a,b}\varphi$
persistence	so are mental state, obligations
capability	capabilities do not fluctuate widely

Agent0 : basic loop

The role of agent programs is to control the evolution of an agent's mental state

- actions occur as a side-effect of the agent's being committed to an action whose time has come

Each agent iterates the following two steps at regular intervals

- read current messages, and update your mental state, including your beliefs and commitments (the agent program is crucial for this update)
- execute the commitments for the current time, possibly resulting in further belief change (this phase is independent of the agent's program)

Agent0 : syntax

fact statements	**atomic objective sentences**
variables	**?x**
private statements	**(DO t p-action)**
communicative st.	**(INFORM t a fact)** **(REQUEST t a action)** **(UNREQUEST t a action)** **(REFRAIN action)**
conditional st.	**(IF mntlcond action)** **mental conditions may contain logical connectives**
commitment rules	**(COMMIT msgcond mntlcond (agent action) *)**

Agent0 : interpreter (1)

Since it is an instance of the generic interpreter, the AGENT-0 interpreter inherits its two-step loop design

The first loop may be specialized as follows

- update the beliefs : Agent0 imposes an extreme restriction, which is to disallow logical connectives other than negation. This makes the consistency checking trivial, at most linear in the size of the database.
- update the commitments : existing commitments are removed either as a result of the belief change, or as a result of UNREQUEST messages.

Agent0 : interpreter (2)

Adding commitments is performed as follows

for each program statement
(COMMIT msgcond mntlcond (a_i action$_i$) *)

if

- msgcond holds the new incoming message,
- mntlcond holds the current mental state
- $\forall$ i, a_i is not currently capable of the action$_i$, and
- $\forall$ i, a_i is not committed to REFRAIN action$_i$, and, if action$_i$ is itself of the form REFRAIN action$_i$,'the agent is not committed to action$_i$',

then

- $\forall$ i, commit to a_i to perform action$_i$

Interaction Oriented Programming [Huhns 96]

Motivations

- errors will always be in complex systems;
- Error-free code can be a disadvantage;
- Where systems interact with the real world, there is a power that can be exploited

Example : children forming a circle

- conventional approach: create a C++ class for each type of object, write a control program that uses trigonometry to compute the location of each object
- interaction-oriented approach: children approach is robust due to local intelligence and autonomy, write the program based on objects having attitudes, goals, agent models

IOP : Active modules, declarative specification, modules that volunteer, modules holdbelief about the world, especially about themselves and others

Organisation Oriented Programming [Lemaitre 98]

Designing, Maintaining, Using MAS utilize different integrative frameworks that include features to deal with agents, interactions, environments, ... MAS programming itself follows history of programming.

The most well-known effort towards MAOP is AOP [Shoham 93] ... IOP [Huhns 97] is an alternative...

OOP is another one [Lemaitre 98] ... EOP does not actually exist as a trend but looks like Artificial Life.

These approach respectively focus on Agents, on Interactions, on Organisations, on Environments, as being the respective basic bricks at the disposal of the designer / MAS / user...

(((A + I) + O) + E) OP : Populations [Demazeau 96]

The Population structure is the set of agents, the set of possible behaviors of the agents, and the set of all interaction processes between agents

Pop = (Ag, Bh, Ip; bc, ic)

Ag : set of agents
Bh : set of behaviors agents are able to perform
Ip : set of interaction processes
bc : Ag ---> P(Bh), behavioral capability,
bc(a), set of behavior a is able to perform
ic : Ag x Ag ---> P(Ip), interaction capability,
ic(a1,a2), set of interaction processes agents a1 and a2 may perform together

(((A + I) + O) + E) OP : Organisations [Demazeau 96]

The Organization structure is composed of organizational roles and organizational links

Org = (Ro; Li)

Ro is defined in a relational way

- e.g. Ro $\subseteq$ Lp x Gp : global processes (Gp) and local processes (Lp), the role is the part of agent's behavior that is integrated in the global process.
- e.g. Ro $\subseteq$ Fo x Lv : foci of interest (Fo), representation levels (Lv), the role is the agent's behavior for a given focus at a given level.

Li $\subseteq$ Ro x Ro

(((A + I) + O) + E) OP : Pop $\Re$ Org [Demazeau 96]

The suitable relation between the Pop and the Org is the system's organization implementation

It is any relation imp = Pop $\Re$ Org, on (Ro x Ag) $\cup$ (Li x Ip), Pop = (Ag, Bh, Ip; bc, ic), Org = (Ro; Li).

- if (r,a) $\in$ imp, r is said to implemented by a
- if (l,p) $\in$ imp, l is said to implemented by p

imp is said "proper" iff $\Re$ is an homomorphism.

- $\forall$ r $\in$ Ro, $\exists$ a $\in$ Ag / (r,a) $\in$ imp, and r is properly implemented by some behavior b $\in$ bc(a)
- $\forall$ l = (l1,l2) $\in$ Li, $\exists$ ip $\in$ Ip / { (l,ip) $\in$ imp $\wedge$ $\exists$ (a1,a2) $\in$ Ag x Ag / ip $\in$ ic(a1,a2), (r1,a1) $\in$ imp, (r2,a2) $\in$ imp, and r1, r2 are properly implemented by the behaviors of a1 and a2, respectively }

(((A + I) + O) + E) OP : PopOrgs [Demazeau 96]

The Interior (= Population + Organisation) of a time-invariant multi-agent system is captured by a population-organization structure PopOrg = (Pop, Org; imp), where

- Pop = (Ag, Bh, Ip; bc, ic) is a population structure
- Org = (Ro; Li) is a organization structure
- imp $\subseteq$ (Ro x Ag) $\cup$ (Li x Ip) is an organization implementation relation as defined previously

Changes in population and organization structures

The three main kinds of changes in the population structure Pop = (Ag, Bh, Ip; bc, ic)

- change of bc(a) of some agent a
- change in ic(a1,a2) of some agents (a1,a2)
- change in the population set Ag

The four main kinds of changes in the organization structure Org = (Ro; Li)

- change in a role r from Ro
- change in a link l from Li
- change in the set of roles Ro
- change in the set of roles Li

Multi-Agent Oriented Programming

Not Object-Oriented Programming

- S = Objects + Message passing

Not Logic nor Expert Systems Programming

- S = Knowledge + Inference Mechanism

Not Ontology-Oriented Programming

- S = Knowledge + Problem Solving Methods

But Agent-Oriented Programming

- S = BDI Agents + KQML (Interactions)

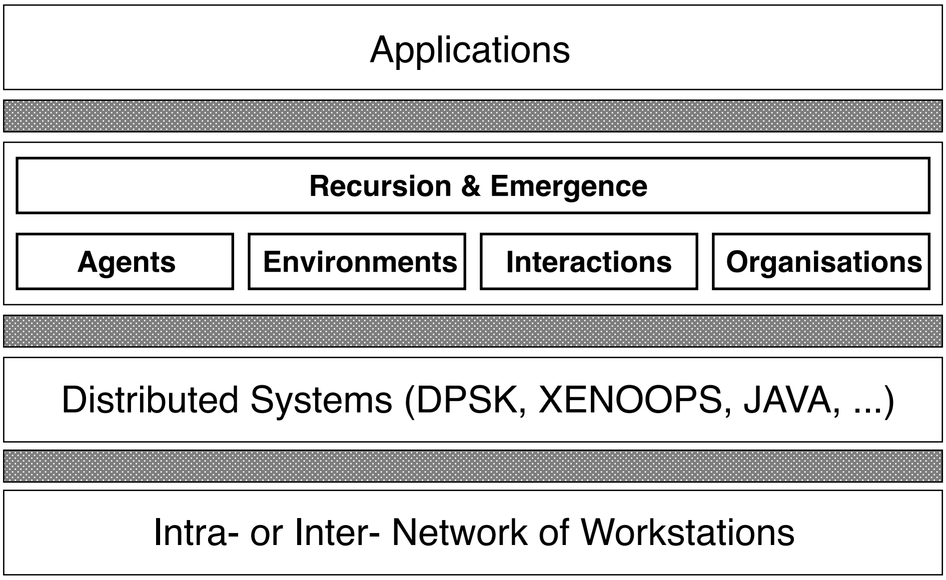
But (((A + I) + O) + E)-Oriented Programming

- S = ((A + I) + O) + E)

But VOWELS Programming

- S = [A*; E*; I*; O*] + (Recursion & Emergence) Mechanism

Programming with the MAGMA approach



VOWELS Oriented Programming [Demazeau 97]

We defend an instance of MAOP, the VOWELS framework in which :

- 1/ to express the problem to solve independently of the domain
- 2/ to "vowellify" the problem in terms of A E I O, ...
- 3/ to choose understood frames of A, E, I, O, dynamics, and recursion
- 4/ to leave VOWELS "emergence engine" complete the missing bricks by itself and build the appropriate MAS...
- 5/ ... to be deployed as self on a distributed settling...
- 6/ ... to be settled and used interactively

VOWELS Oriented Programming [Demazeau 97]

The Declarative Principle

MAS = A + E + I + O

The Functional Principle

Function(MAS) = ∑ Function(entities)
+ Emergence Function

The Recursive Principle

entity = basic entity I MAS

The Programming Principle

MAS = [A*; E*; I*; O*]
+ (Recursion & Emergence) Mechanism

Y. Demazeau, “Steps towards Multi-Agent Oriented Programming” (slides Workshop), 1st International Workshop on Multi-Agent Systems, IWMAS '97, Boston, October 1997.

MAOP and Domain Orientation (1)

(((A + E) + I) + O)	**Robotics Science**
(((A + I) + O) + E)	**Social Science**
((E + A) + (I + O))	**Life Science**
(((I + O) + A) + E)	**Military Science**
(((O + I) + E) + A)	**Economic Science**

The purpose of the domain drives the design of the system

MAOP and Domain Orientation (2)

MAS are not always A-centered !

(((A + E) + I) + O)	Robotics Science
(((A + I) + O) + E)	Social Science
((E + A) + (I + O))	Life Science
(((I + O) + A) + E)	Military Science
(((O + I) + E) + A)	Economic Science

The purpose of the domain drives the design of the system

MAOP and Domain Orientation (3)

PhD Boissier	(A + I) + O	ASIC
PhD Sichman	A + O	DEPNET
PhD Ferrand	((A + I) + O) + E	SANPA
PhD Baeijs	((A + E) + I) + O	SIGMA
PhD Van Aeken	O + A	SMAMS
PhD Ribeiro	I + A	DIM
PhD Ricordel	Development	VOLCANO

The purpose of the domain drives the design of the system…

The Domain Orientation and the User Orientation

PhD Boissier	(A + I) + O	ASIC
PhD Sichman	A + O	DEPNET
PhD Ferrand	((A + I) + O) + E	SANPA
PhD Baeijs	((A + E) + I) + O	SIGMA
PhD Van Aeken	O + A	SMAMS
PhD Ribeiro	I + A	DIM
PhD Ricordel	Development	VOLCANO

The purpose of the domain drives the design of the system…

But how far the user does the user drive the design of the system ? In fact, in all this, where is the User ?

MAOP and User Orientation (1)

The traditional place of the « User » is the « End- »

((((A + E) + I) + O) +U)	Robotics Science
((((A + I) + O) + E) +U)	Social Science
(((E + A) + (I + O)) +U)	Life Science
((((I + O) + A) + E) +U)	Military Science
((((O + I) + E) + A) +U)	Economic Science

MAOP and User Orientation (2)

In fact, in all this, where is the User ? With the A ?

((((U + A) + E) + I) + O) **Robotics Science**

((((U + A) + I) + O) + E) **Social Science**

((E + (U + A)) + (I + O)) **Life Science**

(((I + O) + (U + A)) + E) **Military Science**

(((O + I) + E) + (U + A)) **Economic Science**

But in fact MAS are not always ((U + A)-centered !

Interactive Games

General
- ---> A to be replaced by (U + A)

Handling
- The goal is to master emergence, to optimize a cost
- The understanding of the whole system has to be easy
- The (E)nvironment has to be as realistic as possible
- ---> E to be replaced by (U + E)

Strategy
- The goal is to use and to plan the use of resources
- The visualisation of the interactions is highly desirable
- The key parameters of the game are (I)nteractions
- ---> I to be replaced by (U + I)

Role
- The goal in to increase the competences of the User
- Competences of the characters have to be visualized
- The (O)rganisation constitutes the entry of the game
- ----> O to be replaced by (U + O)

MAOP and User Orientation (3)

From U as consumer…

((((A + E) + I) + O) + U)	Robotics Science
(((O + I) + E) + (U + A))	Economic Science

To U as a partner…

(((I + O) + (U + A)) + E)	Military Science
((E + (U + A)) + (I + O))	Life Science
((((U + A) + I) + O) + E)	Social Science

Towards U as a creator…

((((U + A) + E) + I) + O)	Robotics Science
(((U + (O + I)) + E) + A)	Economic Science

VOWELS A E I O U Decomposition

Agents
- internal architectures of the system processing entities

Environment
- domain-dependent elements for structuring external interactions between entities

Interactions
- elements for structuring internal interactions between entities

Organisations
- elements for structuring sets of entities within the MAS

Users
- internal architectures of the end-user processing entities

VOWELS Oriented Programming

The Declarative Principle

MAS = A + E + I + O + U

The Functional Principle

Function(MAS) = ∑ Function(entities)
+ Emergence Function

The Recursive Principle

entity = basic entity I MAS

The Programming Principle

MAS = [A*; E*; I*; O*; U*]
+ (Recursion & Emergence) Mechanism

Y. Demazeau, “Créativité Emergente Centrée Utilisateur” (keynote), 11èmes Journées Francophones sur les Systèmes Multi-Agents, pp. 31-36, Hermès, Hammamet, Novembre 2003.

Vowels Oriented Programming

MAOP subsumes AOP, IOP, OOP-like OP...

We defend an instance of MAOP, the VOWELS framework in which :

- 1/ to express the problem to solve independently of the domain
- 2/ to "vowellify" the problem in terms of A E I O U, ...
- 3/ to choose understood frames of A, E, I, O, U, dynamics, and recursion
- 4/ to leave VOWELS "emergence engine" complete the missing bricks by itself and build the appropriate MAS...
- 5/ ... to be deployed as self on a distributed settling...
- 6/ ... to be settled and used interactively

The orientation of the domain and of the user

PhD Boissier	(A + I) + O	ASIC
PhD Sichman	A + O	DEPNET
PhD Ferrand	((A + I) + O) + E	SANPA
PhD Baeijs	((A + E) + I) + O	SIGMA
PhD Van Aeken	O + A	SMAMS
PhD Ribeiro	I + A	DIM
PhD Ricordel	Development	VOLCANO
PhD Tavares	Planning	
PhD Deguet	Emergence	
PhD Piolle	((U + I) + O) + E	PAW
PhD Joumaa	Evaluation	MASPAJE
PhD Crepin	((U + O) + I) + E	HIPPO
PhD Lacomme	Dynamics	

The purpose of the user drives the design of the system…

How MAS Methodology is specific ?

= Approach + Model + Tools + Problem + Domain
= Analysis + Design + Development + Deployment

It provides a new analysis and design approach

It is supported by existing formalisms,

It integrates existing programming paradigms,

...

DEPLOYMENT

Object-Based (Concurrent) Languages

Actors-like Languages

- Massive Parallelism
- Every interaction is made by passing of asynchronous buffered messages
- Use of the local continuity
- Object-oriented design
- Full distributed control

Actors [Hewitt]

- An actor is composed of acquaintances, scripts, ...
- A message is an actor containing method, continuation, and complaint actors
- A method is a set of actions : create a new actor, send a message, modify actor's behaviour

Examples of OBCL

PLASMA [Hewitt, Sallé]
- the first actor language

ORIENT-84/K [Tokoro]
- integrates logics, objects, and control monitor

MACE [Gasser]
- actor-based / large-grain

KNOS [Tsichritzis]
- persistent, mobile objects, dynamic acquisition of methods

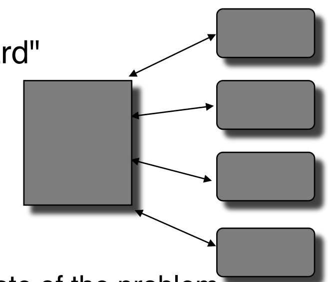
MERING IV [Carle, Ferber]
- actor-based reflective language / multi-grain
- multiple metaobjects and circular interpreters

ACTALK [Briot]
- actors in Smalltalk
- model view controller user interface

...

(Distributed) Blackboard Systems

BB-Like frameworks
- Interactions through the "blackboard"
- Opportunistic activation of the knowledge sources
- Initially Centralized Control



DBB
- Blackboard
  - ✓ A global database representing state of the problem
  - ✓ Hierarchically organised into levels of granularity
- Knowledge Sources
  - ✓ Produce changes (hypotheses) onto given levels of BB
  - ✓ Contain particular subset of domain knowledge
- Centralized Control
  - ✓ Evaluates the status of the problem
  - ✓ Controls the knowledge sources

Examples of Blackboard Systems

GBB [Corkill]
- flexible control regimes / control blackboard
- multi-level problem spaces

BB1/ BB* [Hayes-Roth]
- flexible metalevel control / control blackboard

CAGE/POLIGON [Nii]
- parallel at numerous levels

ATOME [Laasri]
- hierarchical control

TRANSACTIONAL B-BOARDS [Ensor, Gabbe]
- database consistency controls and transaction management integrated with parallel blackboards

...

Characteristics of the Integrative Environments

Language for constructing agents
- Different agent architectures
- Langage for knowledge representation
- Mechanisms for reasoning, deciding, controlling

Representing and dealing with the environment
- Representation of the environment and its evolution
- Implementing the perception of the environment
- Implementing the actions of the agents in the environment

Representing and dealing with other agents
- Primitives, protocols, message processing
- Implementing communication between agents

Development Interface
- Visualisation, trace, inputs-outputs

Application Interface

Interface with Target (Distributed or not) System

Examples of Integrative Environments

Cognitive MAS

- **ABE** [Erman]
- **AOP** [Shoham]
- **MASK** [Demazeau et al.]
- **MECCA** [Steiner]
- **MICE** [Durfee]
- **PACT** [Cutkosky]
- **PHOENIX** [Cohen]
- ...

Reactive MAS

- **ECO** [Drogoul]
- **LYDIA** [Connah]
- **PACO** [Demazeau et al.]
- **RDL** [Steels]
- ...

MICE [Durfee]

Michigan Intelligent Coordination Experiment

Describe environment in which agents act and interact

- 2-D grid model of the world
- does not restrict agent implementation

Variety of organisation structures and cooperation methods built-in

- Contract Net, Greedy Search, Game Theory

Pursuit Problem

ftp'able (freebie.engin.umich.edu: /pub/Mice)

- Built on Allegro Common Lisp

PACT [Cutkosky]

Palo Alto Collaborative Testbed

Computer-Aided Concurrent Engineering

- Stanford, Lockheed, Hewlett-Packard, Enterprise Integration Technologies
- Have integrated four pre-existing CE systems into common distributed framework supporting aspects of small robot

Uses quasi-standards

- Knowledge Interchange Format (KIF) [DARPA-92]
- Knowledge Query & Manipulation Language (KQML) [Finin-92]
- Product Data Exchange Specification (PDES)

MECCA [Steiner]

Multi-Agent Environment for Constructing Cooperative Applications

Domain-independent for integrating wide variety of predefined software systems and humans

Supports agent programming, communication framework, cooperation concepts

- Interfaces are agents
- MAS editing tools, e.g. monitor, debugger ..., Distributed User Interfaces

Based on ESPRIT Project IMAGINE

- Implemented on distributed Parlog+Prolog platform

MASK [Occello]

Applications (.. MAGIC, GEOMED, SMAALA, SIGMA, Le Salon, SPANS..)
(Boissier-94)(Ferrand-95)(Baeijs-95)(Van Aeken 96)

Agents	Environments	Interactions	Organisations
(Boissier-93) (Ferrand-95) (Sichman-95) (Occello-96)	(Demazeau-90) (Ferrand-94) (Baeijs-95)	(Koning-94) (Demazeau-95) (Ferrand-96) (Pesty-96)	(Baeijs-95) (Demazeau-96) (Kozlak-96) (Van Aeken-96)

XENOOPS (Joosens-94) JAVA (Sun™-95)

Network of Sun-Like Workstations

Todays advanced Web offer

Academics

- Firefly (MIT before Microsoft) (no more accesible)
- MadKit (LIRMM Montpellier - Ferber's group)
- Simula (II Porto Alegre - Alvares's group)
- dMARS (-> Jack, by Agent Oriented Software)
- ...

Industrials

- Voyager (ObjectSpace) - freeware (linked with OMG)
- JINI (Sun) - freeware
- Aglets (IBM) - freeware
- Javabeans (Sun) - freeware (based on components)
- Agentbuilder (Reticular) - freeware + product (AOP based)
- ZEUS (BT) - freeware product (FIPA compliant)
- ...

Qualification criteria

Four *qualities* for each stages:

- Completeness: quantity & quality
- Applicability: scope, restrictions
- Complexity: competence required, workload
- Reusability: reuse of previous work

16 criteria + availability & support

	Analysis	Design	Development	Deployment
Completeness				
Applicability				
Complexity				
Reusability				

Selected platforms

Platforms requirements :

- based on a strong academic model
- high quality software, well maintained
- cover as many aspects as possible of MAS
- cover the four methodological stages

Madkit, Jack, Zeus, AgentBuilder

- Evaluation as of first semester 2000
- Detailed presentation for MadKit (2006)
- Detailed presentation for Jack (2007)

MadKit™

Developed by O. Gutknecht & J. Ferber, LIRMM

Based on the AALAADIN organisational model
Graphical multi-agent runtime engine
Good versatility
Light methodology, no BDI

	Analysis	Design	Development	Deployment
Completeness	none	Aalaadin	Pure Java	G-Box
Applicability	n / a	broad range	simple A	small large MAS
Complexity	n / a	intuitive	few code base	GUI
Reusability	n / a	design patterns	classes	dynamic reconf.

CASSIOPEE : Abstraction Levels

Agents
- Which architecture to choose to implement the agents ?
- Which scope of knowledge and how to best use it ?
- Which competences and how are they distributed ?

Interactions
- How do agents communicate ?
- Which content ?
- Can agents influence / alterate other's behaviour ?

Organisations
- How do the agents cooperate ?
- Is there a global goal, how to build a plan to reach it ?
- Which structure to organize, which evolution of the structure ?

CASSIOPEE : General Issues

From the Analysis of natural organisations to the Design of artificial organisations

Based on several applications and experiments

Three Abstraction Levels

- individual agents, interactions, organizations

Agents is defined as a set of Roles

- individual roles, interactional roles, organizational roles

Lacks of Models and Tools

The AGR Model: Agent-Group-Role

Agent An autonomous and communicating entity

- ✓ **The agent plays roles in groups**
- ✓ **An agent may have different roles and may belong to several groups**

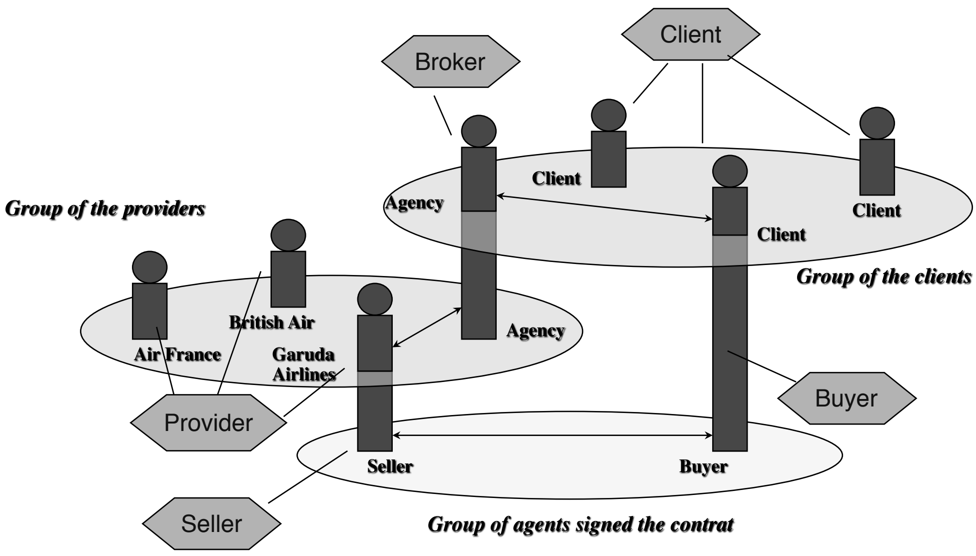
Group A set of agents sharing a certain characteristics

- ✓ **Two agents can communicate only if the belong to the same group**

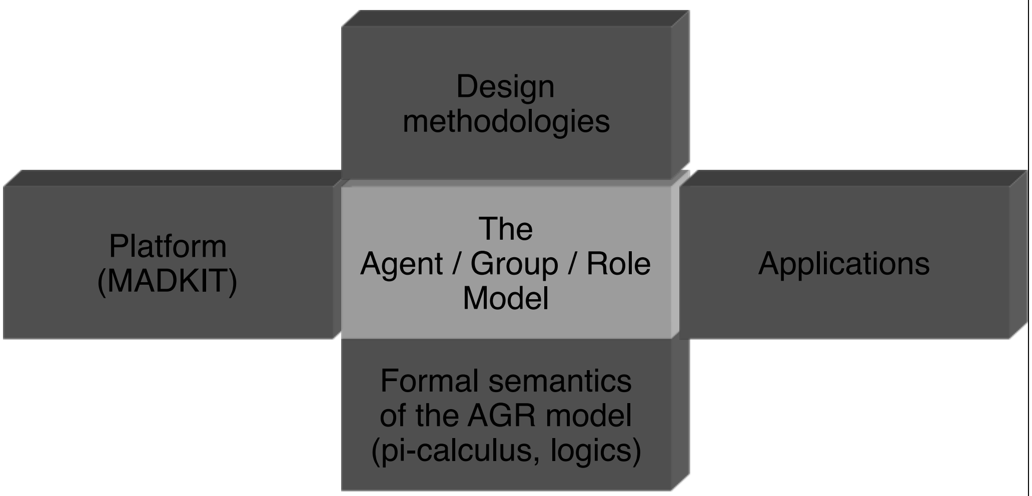
Role Fonction and interaction center

- ✓ **Roles of an agent are local to the groups, and a role has to be requested by the agent**
- ✓ **A role can be played by different agents**

The AGR Model: example of an organisation



The AALAADIN project



MadKit™ : Introduction

MadKit : meta-platform to develop and execute multi-agent systems based on the AGR model

The platform is distributed as open source (GPL/LGPL)

Applications:

- Multi-agent simulation and complex systems modelling
- Collective robotics
- « Intelligent » distributed applications

Several thousands of downloads per year

Very used in research and teaching environments

MadKit™ : caracteristics

Does not impose a particular way to program the agents

- Several programming languages are available: Java, Jess (rule based programming), Python, Scheme

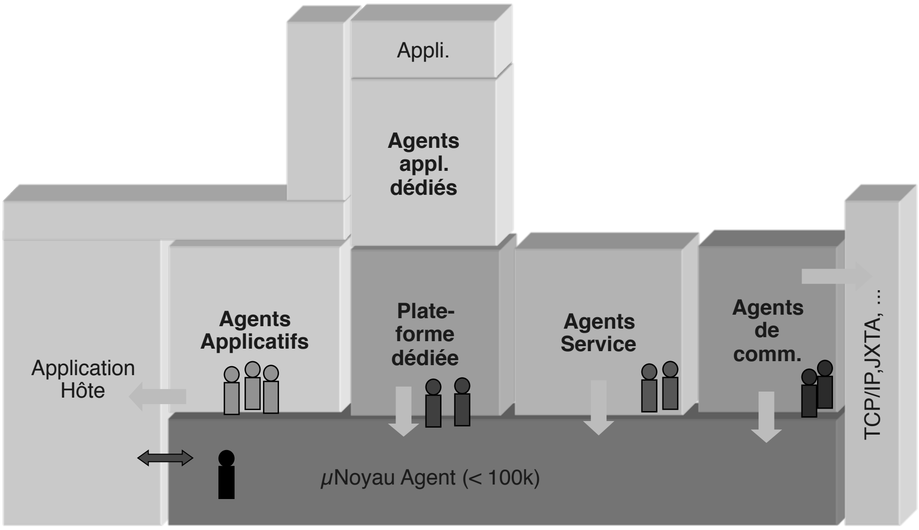
Enables to have a high heterogeneity of agents (from very ants to more cognitive agents) running at the same time

Peer to peer distribution model of the applications

- A MadKit application can be directly distributed onto several kernels.
- It enables to make several multi-agent applications run together in a distributed and concurrent way.

User-friendly interface, including several tools to debug and finalize MAS applications

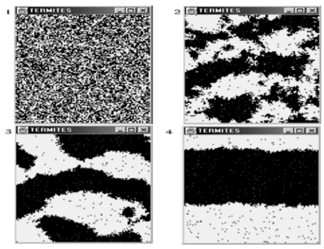
MadKit: General architecture

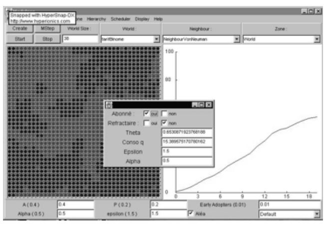


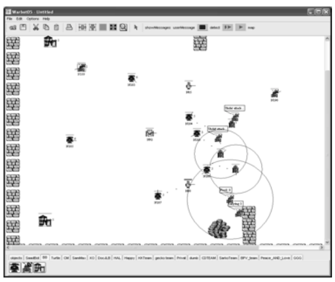
MadKit : meta-platform for development

Dedicated contexts:

- **TurtleKit**: Simple situated MAS to study emergent phenomena (Lirmm)
- **Warbot**: Study of cooperation principles between agents (Lirmm)
- **Moduleco**: Multi-Agent systems for modeling economical phenomena (Phan, Univ. Brest)







Jack™

Developed by Agent Oriented Software Pty.

Including the dMARS BDI model
Great versatility
Focus on the development stage

	Analysis	Design	Development	Deployment
Completeness	none	ident. of classes	extended Java	manual
Applicability	n / a	Jack BDI A	Any MAS	n / a
Complexity	n / a	Jack BDI A	Java & Logic P.	n / a
Reusability	n / a	difficult	classes	n / a

Zeus

Developed by British Telecom

All stages covered, from analysis to deployment
Methodological and Software tools
Limited to a single agent model

	Analysis	Design	Development	Deployment
Completeness	role modelling	finding solutions	5 activities	tools docs
Applicability	role o. MAS	task o. A	Zeus A model	debug visualis.
Complexity	UML	design skills	GUI tools	GUI
Reusability	role models	reusable formal.	partial	A reconf.

AgentBuilder®

Developed by Reticular Systems Inc.

Grounded on Agent0/Placa BDI architecture
Almost all stages covered
Complete graphical tools
Limited to a single agent model

	Analysis	Design	Development	Deployment
Completeness	ontology	A definition	behavoural rules	RT A engine
Applicability	universal	cognitive A	AgentBuild. BDI	small societies
Complexity	OO GUI	MAS design GUI	Logic P. GUI	GUI
Reusability	ontology	protocols	A	none

How MAS Methodology is specific ?

= Approach + Model + Tools + Problem + Domain
= Analysis + Design + Development + Deployment

It provides a new analysis and design approach

It is supported by existing formalisms,

It integrates existing programming paradigms,

It is striving towards industrial quality,

…