

OMAS v7 - Platform Features

Jean-Paul Barthès

BP 349 COMPIÈGNE
Tel +33 3 44 23 44 66
Fax +33 3 44 23 44 77

Email: barthes@utc.fr

N244
July 2009

Warning

This document presents the main features of the OMAS platform.

Keywords

Multi-agent platform, cognitive agents, OMAS

Revisions

Version	Date	Author	Remarks
1.0	July 09	Barthès	First Issue

OMAS documents

Related documents

- UTC/GI/DI/N185L - OMAS v7 - Defining Skills
- UTC/GI/DI/N187 - OMAS 5.0/ JADE 3.1 - A Comparison
- UTC/GI/DI/N195L - OMAS v7 - Goals
- UTC/GI/DI/N208L - OMAS v7 - Primer (Mac OSX)
- UTC/GI/DI/N210L - OMAS v7 - Net
- UTC/GI/DI/N211L - OMAS v5 - Primer (Windows XP)
- UTC/GI/DI/N212L - OMAS v7 - Coterie Structure
- UTC/GI/DI/N213L - OMAS v7 - Postman
- UTC/GI/DI/N215L - OMAS v7 - Tutorial (Windows XP)
- UTC/GI/DI/N216L - OMAS v7 - Internals
- UTC/GI/DI/N217L - OMAS v7 - Personal Assistant
- UTC/GI/DI/N226L - OMAS v7 - API and Low Level Functions
- UTC/GI/DI/N227L - OMAS v7 - Tests
- UTC/GI/DI/N229L - OMAS v7 - Programming Tutorial
- UTC/GI/DI/N230L - OMAS v7 - PA Interface
- UTC/GI/DI/N231L - OMAS v7 - Postier (in French)
- UTC/GI/DI/N232L - OMAS v7 - Postman Connection
- UTC/GI/DI/N233L - OMAS v7 - ACL
- UTC/GI/DI/N234L - OMAS v7 - Multilinguism
- UTC/GI/DI/N235L - OMAS v7 - Window Macros
- UTC/GI/DI/N238L - OMAS-PACPUS (in French)
- UTC/GI/DI/N240L - OMAS v7 - Foreign PA
- UTC/GI/DI/N244L - OMAS v7 - Platform Features

Readers unfamiliar with OMAS should read first N244 (Platform Features), N208 (primer Mac OSX) or N211 (primer Windows XP) or N215 (Tutorial Windows XP), then N212 (Coterie Structure), then N195 (Goals), and N185 (Defining Skills).

Programming information is contained in N185 (Defining Skills) and N226L (API and Low Level Functions).

Readers who want to implement Personal Assistants should read N217 (Personal Assistant) and N240 (Foreign PA) for a tutorial.

Readers interested in inter-platform communication should read N213 (Postman) and N232 (Postman Connection), and N238 (OMAS-PACPUS) for an example.

N187 presents a brief comparison with JADE.

Contents

1	Introduction	7
2	Main Design Decisions	7
2.1	The Nature of Agents	7
2.2	P2P Organization and Coteries	8
2.3	Non Mobility	8
2.4	Skills	8
2.5	Goals	8
2.6	Personal Assistants	8
2.7	Ontologies and Knowledge Bases	8
2.8	IDE	8
3	OMAS ACL Language	8
3.1	Message Structure	9
3.2	Types of Messages (Performatives)	10
3.3	Addressing Modes	10
3.3.1	Point-to-Point	10
3.3.2	Multicast	11
3.3.3	Broadcast	11
3.3.4	Conditional	11
3.4	Protocols	11
3.4.1	Simple-Protocol	11
3.4.2	Contract Net	11
3.5	Message Processing Mechanism	11
3.6	Sending Messages	11
4	OMAS Content Language	13
5	OMAS Agent Skills	13
6	OMAS Agent Goals	13
7	OMAS Ontologies	14
8	OMAS Agent Types	14
8.1	Service Agents	14
8.2	Transfer Agents (Postmen)	15
8.3	Personal Assistant Agents	15
9	OMAS IDE	15

1 Introduction

The OMAS (Open Multi-Agent System) platform has been developed over many years to let application designers build applications involving cognitive agents easily. It offers several models of agents and a middleware taking care of the traditional agent machinery like sending messages and applying skills. The goal was to develop a tool in which an application could be programmed by adding a minimum of code in a plugin style.

The set of OMAS documents give details on the various parts of the platform. Tutorial documents are also available for Windows and Mac environments.

2 Main Design Decisions

We took the following decisions:

- agents are totally independent and interact in a P2P fashion (there is no central record of agents nor of their skills, namely there are no white or yellow pages);
- agents have a long life;
- agents are multi-threaded and can do several tasks at the same time;
- agents can join or quit the platform at any time;
- agents *cannot* travel from machine to machine;
- agents are organized in coteries; a coterie is located on a single physical LAN loop;
- remote coteries may be connected trough a transfer agent (postman);
- agents have skills;
- agents have goals that are different from skills;
- agents communicate with the user through a personal assistant agent;
- agents have ontologies;
- agents can reason;
- there is an IDE for exercising agents.

The options are discussed in the following paragraphs.

2.1 The Nature of Agents

Our agents are complex agents (their complexity is of the order of an OS). Once they are created they stay alive until somebody or something kills them. Our vision is to allocate one machine for an agent or a small group of agents.

2.2 P2P Organization and Coterie

We wanted our agents to be completely free of joining and leaving the group of agents at any time without having to register anywhere. Therefore, inside a coterie (a group of agents leaving on the same LAN loop) low-level communication is done by broadcasting, using the UDP protocol, meaning that all agents can receive all messages. However, the middleware will filter messages, delivering only those targeted to an agent. A coterie is specified by its port of communication, which is the only piece of data an agent must know to join the coterie.

2.3 Non Mobility

OMAS agents are complex agents and cannot travel from a machine to another, although this could be implemented easily. The rationale for this decision was that OMAS agents are supposed to have a dedicated machine. Thus, it is not very meaningful for them to leave it.

2.4 Skills

Agents have a set of skills allowing them to perform tasks. If an agent receives a message but has not the corresponding skill, it simply ignores it.

2.5 Goals

Agents may have goals of different kinds: one-shot, or cyclic. Goals are different from skills, and are used for triggering a specific behavior when some conditions are verified.

2.6 Personal Assistants

Since communication with humans is an important features, OMAS provides Personal Assistant agents, that act as digital butlers. They can be programmed in the user's native language, and may use staff agents.

2.7 Ontologies and Knowledge Bases

Agents use ontologies for expressing tasks and domain knowledge. An ontology can be used to develop a knowledge base. Agents use the MOSS language for expressing the ontology and knowledge bases. The MOSS query mechanism can be used for reasoning.

2.8 IDE

An interactive developing environment is available for debugging purposes, allowing to examine the content of agents and to display their behavior.

3 OMAS ACL Language

The Agent Communication Language (ACL) specifies the structure of messages and the performatives that can be used to communicate (type of messages). In addition messages can be routed as point-to-point, multicast, broadcast, conditional, or ContractNet.

3.1 Message Structure

The structure of messages and performatives are described in the following mémo.

N233 OMAS v7 ACL, UTC/GI, dec 2008

Table 1 lists the various parameters that can be found in a message, comparing them with the parameters that are defined for a FIPA message. Some of the missing parameters on the OMAS side (e.g. language, encoding, ontology) are in practice included in the content/action/args parameters.

Table 1: FIPA and OMAS list of parameters

FIPA Parameter	OMAS Parameter	Category
-	name	Message description
-	date	Message description
performative	type	Type of communicative act
receiver	to	Participant in communication
sender	from	Participant in communication
reply-to	reply-to	Participant in communication
content	content, action, args	Content of message
language	-	Description of content
encoding	-	Description of content
ontology	-	Description of content
-	error-contents	Description of content
protocol	protocol	Control of conversation
conversation-id	task-id	Control of conversation
reply-with	-	Control of conversation
in-reply-to	task-id	Control of conversation
reply-by	time-limit	Control of conversation
-	acknowledgement	Control of conversation
-	repeat-count	Control of conversation
-	but-for	Control of conversation
-	strategy	Control of conversation
-	sender-ip	System Information
-	sender-site	System Information
-	thru	System Information
-	task-timeout	System Information
-	timeout	System Information
-	id	Message identifier

Because an agent usually does not answer if not interested, a lack of answer on a point to point request is ambiguous. I.e. the agent may not want to answer, or it may have left, or it may be down. Consequently, an **acknowledgment** parameter was added to force an agent that received a message to acknowledge it (similar to registered mail).

More specific parameters have been added, like **sender-site**, **sender-IP**, **thru**, to avoid looping when several coteries are interconnected. Indeed, any message is distributed to all coteries by means of transfer agents, which might lead to sending messages in a loop if one is not careful. However, the user does not have to deal with such parameters.

3.2 Types of Messages (Performatives)

The OMAS performative are somewhat different from the FIPA/KQML performative and are listed in Table 2.

Table 2: FIPA and OMAS list of parameters

FIPA performatives	OMAS performatives
-	Abort
Accept Proposal	-
-	Acknowledge
Agree	Grant
-	Answer
-	Bid with Answer
Cancel	Cancel
-	Cancel-Grant
Call for Proposal	Call for Bids
Confirm	-
Disconfirm	-
Failure	Error
Inform	Inform
Inform If	-
Inform Ref	-
-	Internal
Not Understood	-
Propagate	-
Propose	Bid
Proxy	-
Query If	-
Query Ref	Request
Refuse	-
Reject Proposal	-
Request	Request
Request When	-
Request Whenever	-
Subscribe	-
-	Sys-Inform

OMAS exchange of messages is more flexible than those specified by FIPA, since an OMAS agent does not have to answer if it does not want to.

3.3 Addressing Modes

OMAS messages have four addressing modes: point-to-point, multicast, broadcast and conditional.

3.3.1 Point-to-Point

As usual, the message is sent from a specific agent to another specific agent.

3.3.2 Multicast

A message is sent to a list of agents.

3.3.3 Broadcast

A message is sent to all agents of the connected coterie (address is ALL or ALL-AND-ME).

3.3.4 Conditional

Conditional messages contain a condition expressed as a MOSS query. Agents for which the query returns a non null answer will receive the message. The query acts as a semi-predicate. E.g.

```
(:_cond ("agent" ("eyes" ("eyes" ("color" :is "blue")))))
```

will deliver messages to all agents with *known* blue eyes.

3.4 Protocols

OMAS has only two types of protocols: simple and contract-net.

3.4.1 Simple-Protocol

The simple protocol sends the message in any addressing mode. If the mode is multi-cast or broadcast, it can select 3 strategies: *take-first-answer*, *collect-answers*, *collect-first-n-answers*.

The *take-first-answer* strategy will return as soon as an answer is received, The *collect-answers* strategy will collect answers until a timeout is reached, the *collect-first-n-answers* strategy will return as soon as n answers have been received or wait for a timeout if not enough answers are received.

A message may specify a time limit to the destination agent. A message has a timeout (defaulting to 1 hour).

3.4.2 Contract Net

When Contract Net is specified the Contract Net protocol is executed, sending first a Call-For-Bids, waiting for answers, and granting the task to some agent(s).

3.5 Message Processing Mechanism

Messages are processed differently according to their nature. When an agent is created, it has Two processes: *scan* and *mbox* (Fig.1).

Messages other than request are processed immediately by the *scan* process. Request messages are put into the agenda and processed by the *mbox* process.

A request message will lead to the creation of a new thread for processing the message and a thread for a time-limit as shown Fig.2. The figure also shows that if the task process spawns subtasks, the *scan* process will redirect answers to the proper thread. This is done by the middleware.

3.6 Sending Messages

Most messages will be sent using the *send-subtask* macro. Other types of messages are usually sent by the system as a consequence of doing some action. For example the answer message to a specific request will be built and sent by the middleware when a skill terminates.

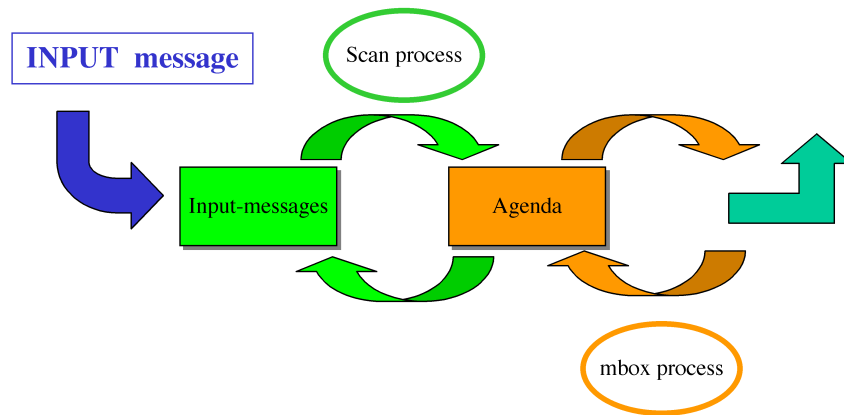


Figure 1: Initial Agent Processes: scan receives messages, mbox waits on agenda

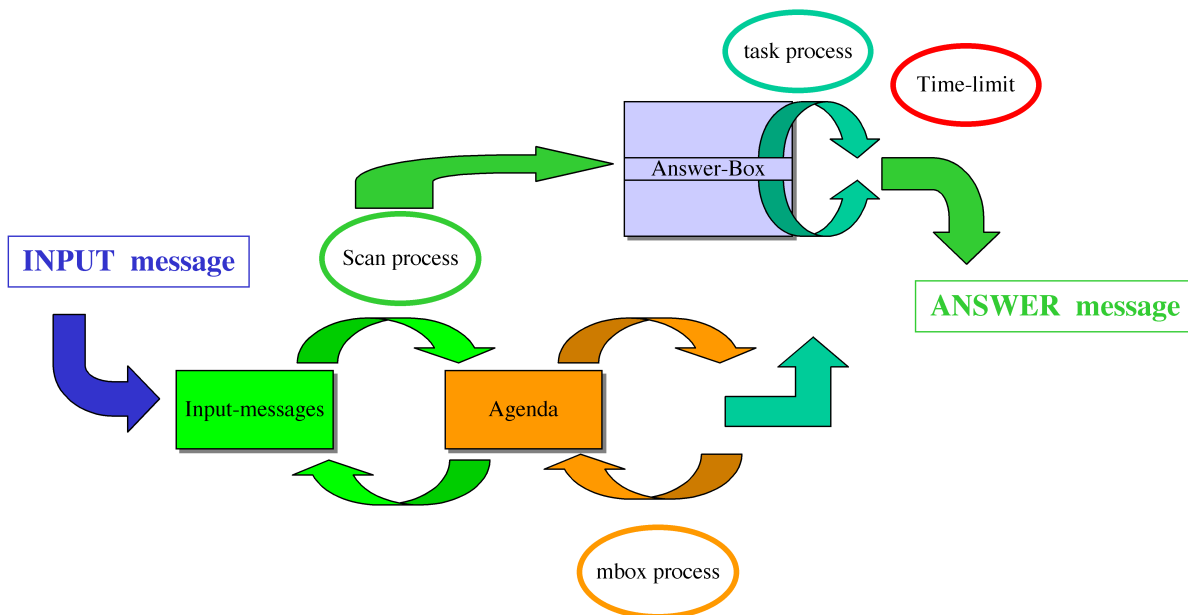


Figure 2: a request message creates a new task process and a time-limit timer

4 OMAS Content Language

The OMAS content language is not currently strictly specified. Thus, it is up to the application developer to use the language most appropriate for the application.

In practice, the action parameter specifies a task to be undertaken and the args parameter contains information related to such a task (usually formatted as an association list). Some exchanges between a Personal Assistant and the staff agent transfer pieces of natural language.

5 OMAS Agent Skills

A detailed description is done in the following document:

N185 OMAS v7 Defining Skills, UTC/GI, July 2008

Skills correspond to tasks that an agent can do. A skill is programmed as a function, the name of which is contained in the action parameter of the message. There are two types of skills: an atomic skill and a complex skill. Atomic skills are entirely executed by the agent without help. Complex skills spawn subtasks to be executed by other agents.

A skill may use following functions:

Table 3: OMAS skill functions

Function	Meaning
static-fcn	called when the skill is activated
dynamic-fcn	called when an answer is sent by a spawned subtask
timeout-handler	called when a timeout fires
preconditions	called before executing the skill to check preconditions
select-best-answer-fcn	called on broadcast or multicast to select answer
acknowledge-fcn	called when received message asks for acknowledgment

At least the static function must be provided when the skill is atomic, and the dynamic function when the skill is complex. Other functions are optional and depend on the protocol and adopted strategy. The other functions have a default behavior.

When a message is received by an agent OMAS calls the proper function automatically in the proper thread, according to the specified skill. Thus, the application designer is saved the trouble of waiting for messages (events), filtering them and distributing them to the proper thread.

Skills are defined by using the `defskill` macro.

6 OMAS Agent Goals

Goals are optional but may be used to make the agent pro-active. Details about goals can be found in the following document:

N195 OMAS v7 Goals, UTC/GI, July 2008

Parameters used to define a goal are shown Table 4. Although there are two types of goals (rigid and flexible) only the rigid goals are currently implemented. The flexible goal mechanism, corresponding to the propagation of an energy of activation, is not yet available. Goals may be one-shot or cyclic.

Table 4: OMAS skill functions

Function	Meaning
mode	activation mode (:rigid or :flexible)
type	type of goal (:permanent :cyclic :1-shot)
period	period for cyclic goals (default is 20s)
expiration-date	date at which the goal dies
expiration-delay	time to wait until we kill the goal
importance	on a 1-100 scale (not yet available)
urgency	on a 1-100 scale (not yet available)
activation-date	date at which the goal should fire (default is now)
activation-delay	time to wait before activating the goal
activation-level	on a 1-100 scale (default 50) (not yet available))
activation-threshold	on a 1-100 scale (default 50) (not yet available)
activation-change-fcn	optional function called at each cycle (not yet available)
status	waiting, active, dead,...
goal-enable-fcn	function checking the goal enabling conditions
script-fcn	a function returning a list of messages

An activation function decides whether the goal will fire or not. The goal consists in executing a list of messages sent to the agent as internal messages.

Goals are defined by using the `defgoal` macro.

7 OMAS Ontologies

Agents have an ontology that can be extended to suit the application purposes and used for instantiating a knowledge base. The ontology must be formalized using the MOSS language. The ontology can be multilingual if necessary. Details are given in the MOSS documentation and an extensive example can be found in:

N240 OMAS v7 Creating Foreign PA - Tutorial, UTC/GI, May 2009

8 OMAS Agent Types

OMAS offers three types of agents: Service Agent, Transfer Agent (Postman), or Personal Assistant.

8.1 Service Agents

Service Agents are regular agents that provide a set of services. They can be thought as Web Services with the difference that they may not answer even if they can do the job. Their structure is shown Fig.3.

The various parts of the structure of an agent are:

- net interface, handling message and log;
- skills, containing the skills corresponding to tasks the agent can do;
- world, containing a model of the world, namely a description of other agents obtained through the processing of messages;

- tasks, a model of the current tasks being active in the system (currently unused);
- ontology, containing the ontology and knowledge base;
- self, containing a self description, a description of skills and the agent memory;
- control, containing all the necessary internal structures for running the agent.

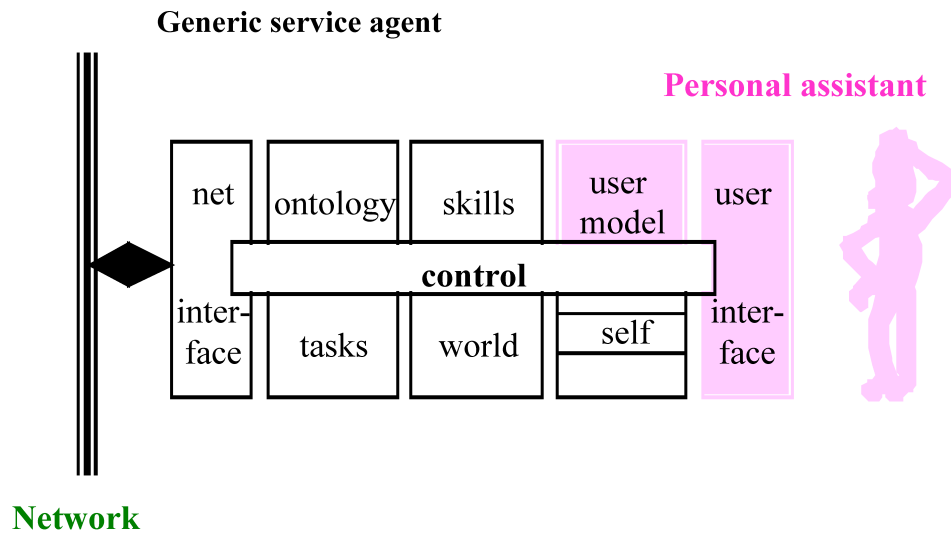


Figure 3: The structure of a Service Agent / Personal Assistant (with grey additions)

8.2 Transfer Agents (Postmen)

Transfer Agent (Postmen) are used as gateways between two remote coteries or for interfacing the platform with web services or for interfacing with other multi-agent platforms. They normally use a TCP protocol. An example for developing a postman between remote coteries is given in:

N232 OMAS v7 Postman Connection, UTC/GI, March 2009

8.3 Personal Assistant Agents

Personal Assistant agents are complex agents that can interface the user through a dialog in natural language in any language. They use ontologies and a complex dialog mechanism. How to build a Personal Assistant is described in:

N240 OMAS v7 Creating a Foreign PA, UTC/GI, May 2009

9 OMAS IDE

The OMAS IDE shown Fig.4 allows exercising the agent environment and is detailed in:

N215 OMAS v7 Tutorial, UTC/GI, June 2008

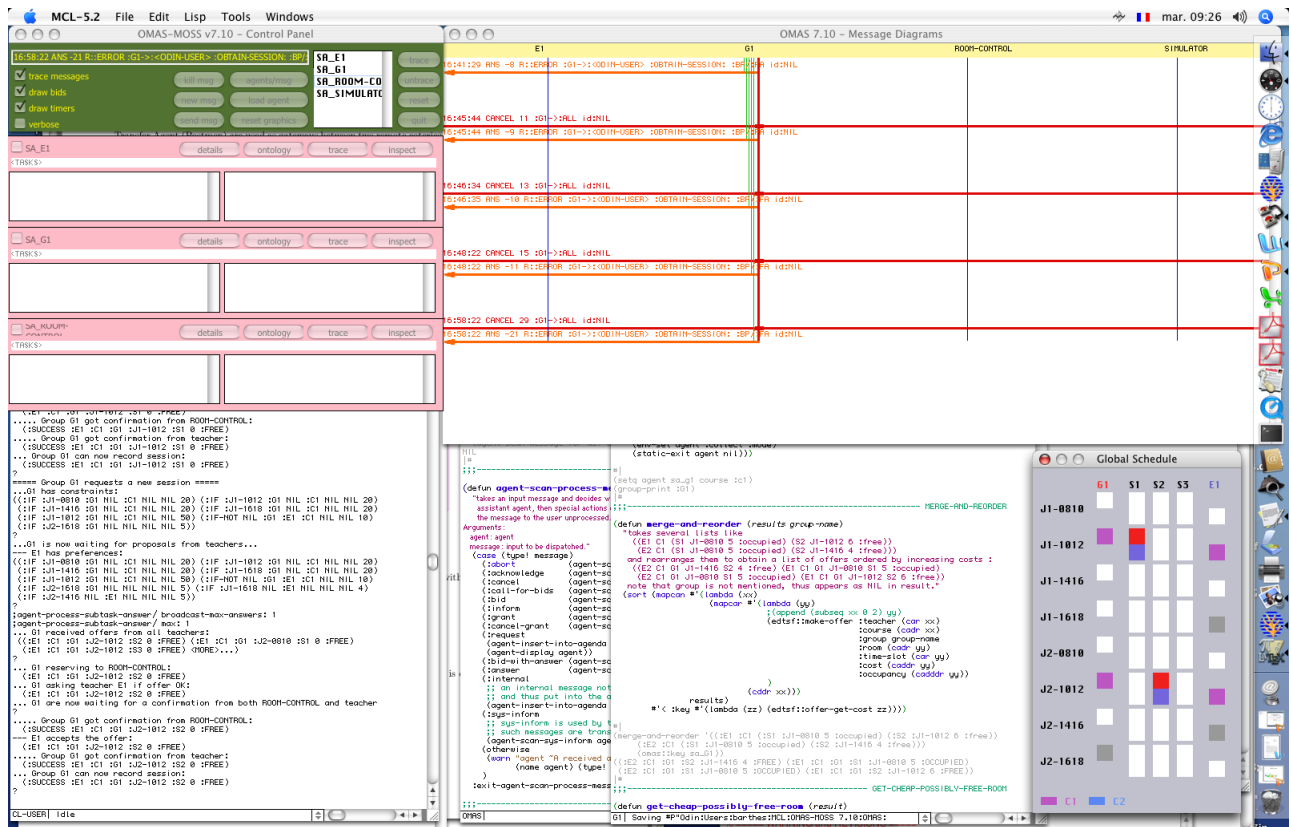


Figure 4: OMAS Interactive Development Environment