UNIVERSITE DE TECHNOLOGIE DE COMPIÈGNE
Département de Génie Informatique
CNRS UMR 7253 Heudiasyc

OMAS v10 - 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
http://www.utc.fr/∼barthes

N244
February 2013

Warning

This document presents the main features of the OMAS v10 platform.

Keywords

Multi-agent platform, cognitive agents, OMAS

Revisions

Version	**Date**	**Author**	**Remarks**
1.0	July 09	Barthès	First Issue
2.0	February 13	Barthès	v10

Contents

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 OMAS User Manual (UTC/GI/DI/N260L - OMAS v10 User Manual) gives details on the various parts of the platform. Tutorial are also included.

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 log 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;
- coteries are organized in platforms; a platform may span several coteries all over the world;
- remote coteries are connected through a transfer agent (postman);
- agents have skills;
- agents have goals that are distinct 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.

This has been recently extended:

- personal assistants can use voice interaction;
- service agents can be persistent;
- agents can use HTTP protocols;
- users can communicate with a personal assistant using the Web;
- OMAS has a limited FIPA interface.

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 Coteries

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 living 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 filters 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 the message.

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 feature, OMAS provides Personal Assistant agents, that act as digital butlers. They can be programmed in the user's native language, and may use more technical 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.

2.9 Voice Interaction

It should be possible to communicate with the agents through a personal assistant using voice and natural language.

2.10 Persistency

Service agent should elect to be persistent to keep state if their machine crashes. This should not make the programming overly complex.

2.11 HTTP Protocol

Agent should be able to use the HTTP protocol to simplify access through firewalls.

2.12 Web Access

If a remote machine has no OMAS environment it should be possible to communicate with one's PA using a web browser.

2.13 FIPA Compliance

Because FIPA is currently the only set of standards for agent platforms, OMAS should have some sort of FIPA interface.

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 memo:

`N233 OMAS v7 ACL, UTC/GI, dec 2008`

Table **??** 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.

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 dead. 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 **??**.

Again note that 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.

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

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

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 coteries, i.e. the platform (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.**??**).

Messages other than requests 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 setting up a time-limit as shown Fig.**??**. 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.

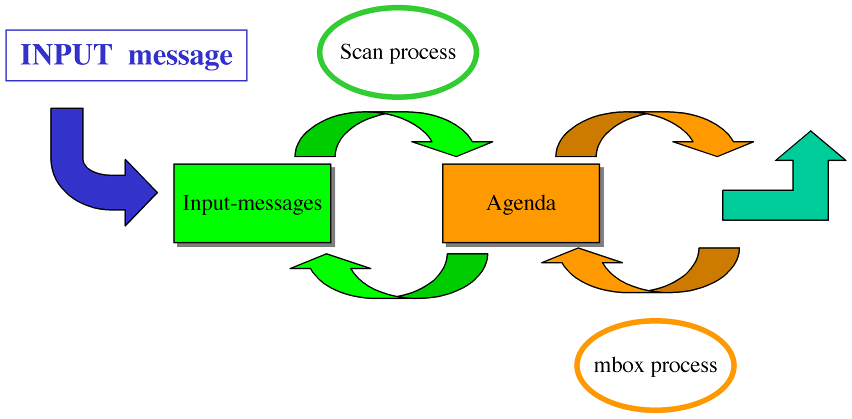


Figure 1: Initial Agent Processes: scan receives messages, mbox waits on agenda

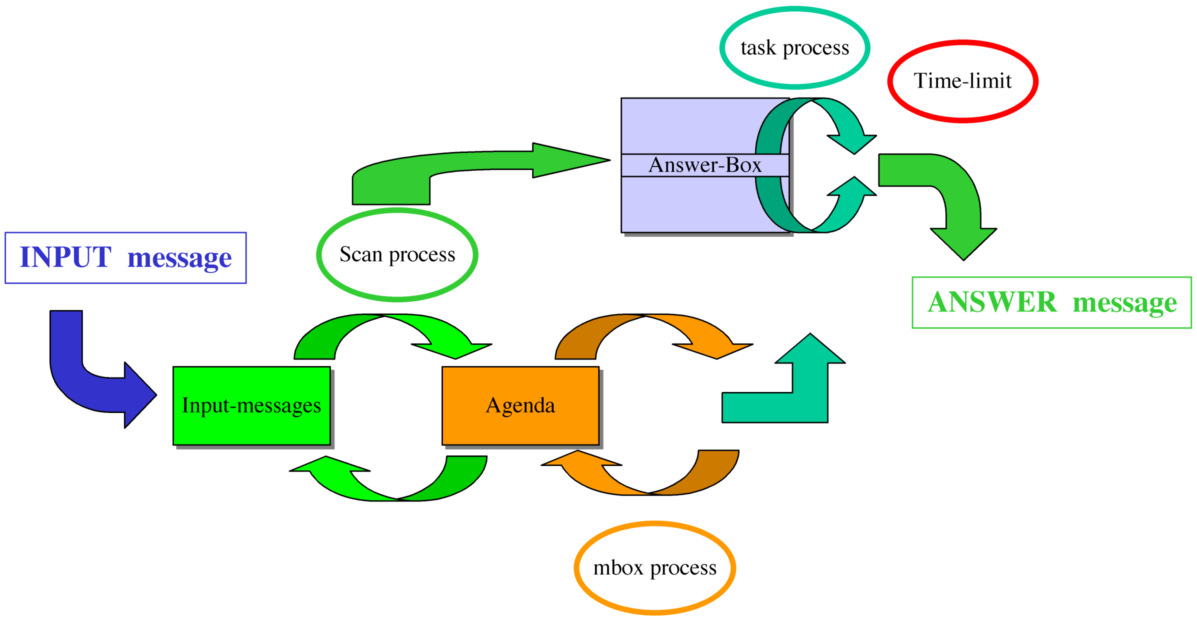


Figure 2: a request message creates a new task process and a time-limit timer

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.

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). Currently, some exchanges between a Personal Assistant and the staff agent transfer pieces of natural language.

A recommended format for the :args field of a message for exchanging information between a personal assistant agent and other agents is the following:

```
((:data . <expr specifying the data>) (:language . <language tag>)
 (:pattern . <pattern>) ...)
```

with the following meaning:

:data, the field usually contains a string in natural language;

:language, the field contains a language marker, e.g. :FR, :EN, :JP, ...

:pattern, the field uses MOSS and the ontology to request a structured answer. If no pateern in given, the answer will consist of a string literal. If a patter is specified, it should obey a MOSS description, using concepts and properties from the agent ontology (which may be different from that of the agent that will receive the message:

<**other tags**>, defined as needed by the application, e.g. for specifying a gate for synchronizing data in web applications.

5 OMAS Agent Skills

A detailed description is done in the User Manual.

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:

At least the static function must be provided when the skill is atomic, and also 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.

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

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 User Manual.

Parameters used to define a goal are shown Table **??**. 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.

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.

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

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.

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.??.

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.

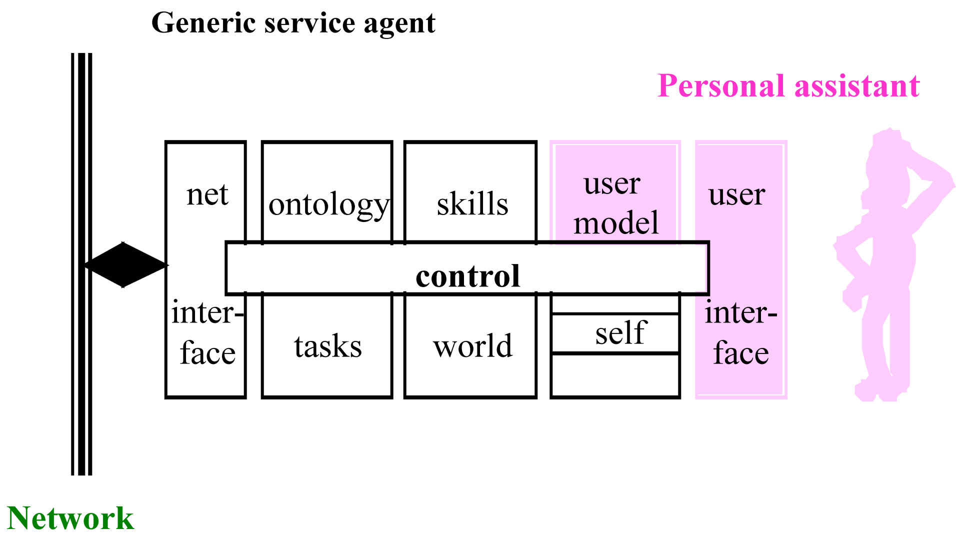


Figure 3: The structure of a Service Agent / Personal Assistant (with grey additions)

8.2 Transfer Agents (Postmen)

Transfer Agents (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:

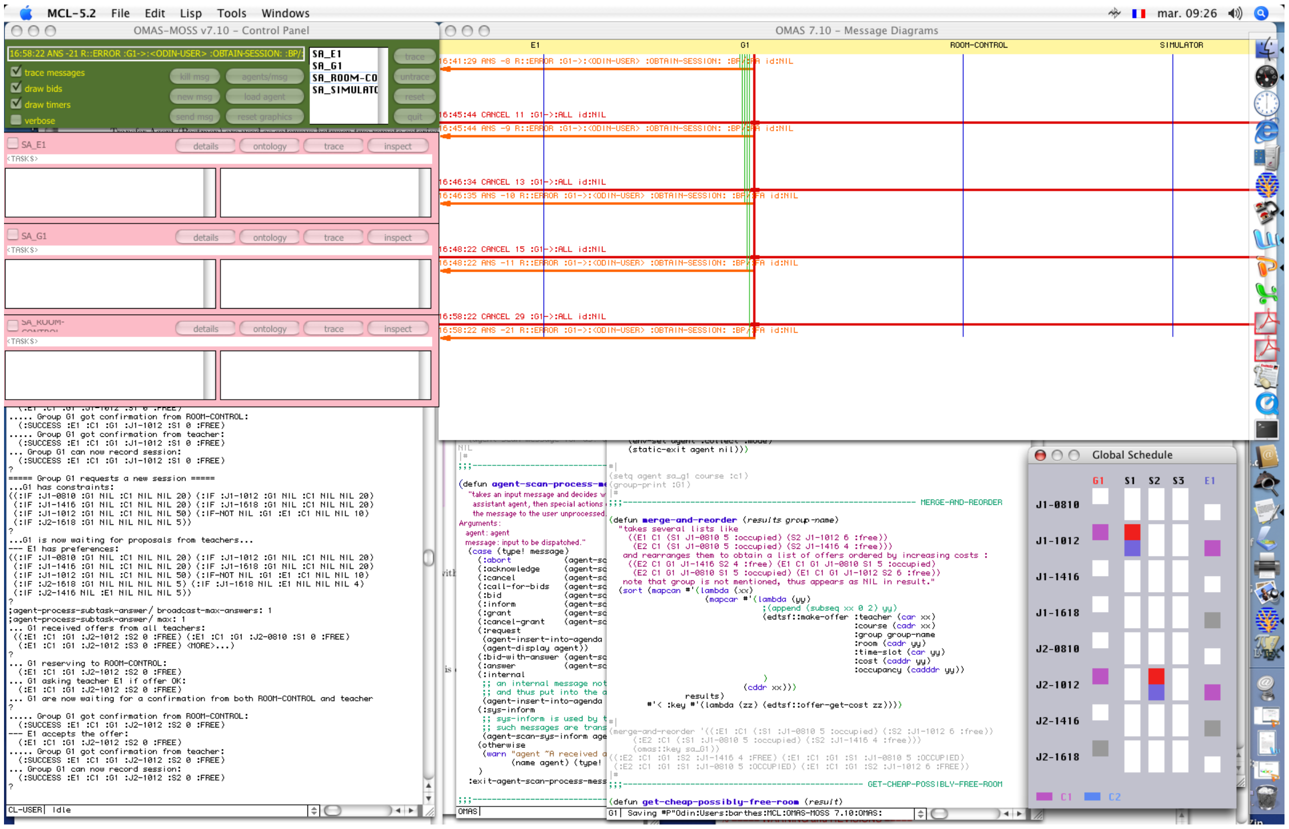
`N232 OMAS v7 Postman Connection, UTC/GI, March 2009`

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 the OMAS User Manual. Furthermore, the exchange can be done using voice with a spoken feedback.

9 OMAS IDE

The OMAS IDE shown Fig.?? allows exercising the agent environment and is detailed in the OMAS User Manual.



```
16:41:29 ANS -8 R::ERROR :G1->:<ODIN-USER> :OBTAIN-SESSION: :BP/:FA id:NIL
16:45:44 CANCEL 11 :G1->:ALL id:NIL
16:45:44 ANS -9 R::ERROR :G1->:<ODIN-USER> :OBTAIN-SESSION: :BP/:FA id:NIL
16:46:34 CANCEL 13 :G1->:ALL id:NIL
16:46:35 ANS -10 R::ERROR :G1->:<ODIN-USER> :OBTAIN-SESSION: :BP/:FA id:NIL
16:48:22 CANCEL 15 :G1->:ALL id:NIL
16:48:22 ANS -11 R::ERROR :G1->:<ODIN-USER> :OBTAIN-SESSION: :BP/:FA id:NIL
16:58:22 CANCEL 29 :G1->:ALL id:NIL
16:58:22 ANS -21 R::ERROR :G1->:<ODIN-USER> :OBTAIN-SESSION: :BP/:FA id:NIL
```

```
..... Group G1 got confirmation from ROOM-CONTROL:
  (:SUCCESS :E1 :C1 :G1 :J1-1012 :S1 0 :FREE)
..... Group G1 got confirmation from teacher:
  (:SUCCESS :E1 :C1 :G1 :J1-1012 :S1 0 :FREE)
... Group G1 can now record session:
  (:SUCCESS :E1 :C1 :G1 :J1-1012 :S1 0 :FREE)
?
===== Group G1 requests a new session =====
...G1 has constraints:
((:IF :J1-0810 :G1 NIL :C1 NIL NIL 20) (:IF :J1-1012 :G1 NIL :C1 NIL NIL 20)
 (:IF :J1-1416 :G1 NIL :C1 NIL NIL 20) (:IF :J1-1618 :G1 NIL :C1 NIL NIL 20)
 (:IF :J1-1012 :G1 NIL :C1 NIL NIL 50) (:IF-NOT NIL :G1 :E1 :C1 NIL NIL 10)
 (:IF :J2-1618 :G1 NIL NIL NIL NIL 5))
?
...G1 is now waiting for proposals from teachers...
--- E1 has preferences:
((:IF :J1-0810 :G1 NIL :C1 NIL NIL 20) (:IF :J1-1012 :G1 NIL :C1 NIL NIL 20)
 (:IF :J1-1416 :G1 NIL :C1 NIL NIL 20) (:IF :J1-1618 :G1 NIL :C1 NIL NIL 20)
 (:IF :J1-1012 :G1 NIL :C1 NIL NIL 50) (:IF-NOT NIL :G1 :E1 :C1 NIL NIL 10)
 (:IF :J2-1618 :G1 NIL NIL NIL NIL 5) (:IF :J1-1618 NIL :E1 NIL NIL NIL 4)
 (:IF :J2-1416 NIL :E1 NIL NIL NIL 5))
?
;agent-process-subtask-answer/ broadcast-max-answers: 1
;agent-process-subtask-answer/ max: 1
... G1 received offers from all teachers:
 ((:E1 :C1 :G1 :J2-1012 :S2 0 :FREE) (:E1 :C1 :G1 :J2-0810 :S1 0 :FREE)
  (:E1 :C1 :G1 :J2-1012 :S3 0 :FREE) <MORE>...)
?
... G1 reserving to ROOM-CONTROL:
  (:E1 :C1 :G1 :J2-1012 :S2 0 :FREE)
... G1 asking teacher E1 if offer OK:
  (:E1 :C1 :G1 :J2-1012 :S2 0 :FREE)
... G1 are now waiting for a confirmation from both ROOM-CONTROL and teacher
?
..... Group G1 got confirmation from ROOM-CONTROL:
  (:SUCCESS :E1 :C1 :G1 :J2-1012 :S2 0 :FREE)
--- E1 accepts the offer:
  (:E1 :C1 :G1 :J2-1012 :S2 0 :FREE)
..... Group G1 got confirmation from teacher:
  (:SUCCESS :E1 :C1 :G1 :J2-1012 :S2 0 :FREE)
... Group G1 can now record session:
  (:SUCCESS :E1 :C1 :G1 :J2-1012 :S2 0 :FREE)
?
CL-USER| Idle
```

```
NIL
|#
;;;----------------------------

(defun agent-scan-process-me
  "takes an input message and decides w
   assistant agent, then special actions
   the message to the user unprocessed.
Arguments:
  agent: agent
  message: input to be dispatched."
  (case (type! message)
    (:abort              (agent-sc
    (:acknowledge        (agent-sc
    (:cancel             (agent-sc
    (:call-for-bids      (agent-sc
    (:bid                (agent-sc
    (:inform             (agent-sc
    (:grant              (agent-sc
    (:cancel-grant       (agent-sc
    (:request
     (agent-insert-into-agenda
     (agent-display agent))
    (:bid-with-answer (agent-sc
    (:answer             (agent-sc
    (:internal
     ;; an internal message not
     ;; and thus put into the a
     (agent-insert-into-agenda
    (:sys-inform
     ;; sys-inform is used by t
     ;; such messages are trans
     (agent-scan-sys-inform age
    (otherwise
     (warn "agent ~A received a
           (name agent) (type!
    )
  :exit-agent-scan-process-mess

;;;----------------------------
OMAS|
```

```
    (static-exit agent nil)))

#|
(setq agent sa_g1 course :c1)
(group-print :G1)
|#
;;;---------------------------------------------------------------- MERGE-AND-REORDER

(defun merge-and-reorder (results group-name)
  "takes several lists like
    ((E1 C1 (S1 J1-0810 5 :occupied) (S2 J1-1012 6 :free))
     (E2 C1 (S1 J1-0810 5 :occupied) (S2 J1-1416 4 :free)))
   and rearranges them to obtain a list of offers ordered by increasing costs :
    ((E2 C1 G1 J1-1416 S2 4 :free) (E1 C1 G1 J1-0810 S1 5 :occupied)
     (E2 C1 G1 J1-0810 S1 5 :occupied) (E1 C1 G1 J1-1012 S2 6 :free))
   note that group is not mentioned, thus appears as NIL in result."
  (sort (mapcan #'(lambda (xx)
                    (mapcar #'(lambda (yy)
                                ;(append (subseq xx 0 2) yy)
                                (edtsf::make-offer :teacher (car xx)
                                                   :course (cadr xx)
                                                   :group group-name
                                                   :room (cadr yy)
                                                   :time-slot (car yy)
                                                   :cost (caddr yy)
                                                   :occupancy (cadddr yy))
                                )
                      (cddr xx)))
                results)
        #'< :key #'(lambda (zz) (edtsf::offer-get-cost zz))))

#|
(merge-and-reorder '((:E1 :C1 (:S1 :J1-0810 5 :occupied) (:S2 :J1-1012 6 :free))
     (:E2 :C1 (:S1 :J1-0810 5 :occupied) (:S2 :J1-1416 4 :free)))
     (omas::key sa_G1))
((:E2 :C1 :G1 :S2 :J1-1416 4 :FREE) (:E1 :C1 :G1 :S1 :J1-0810 5 :OCCUPIED)
 (:E2 :C1 :G1 :S1 :J1-0810 5 :OCCUPIED) (:E1 :C1 :G1 :S2 :J1-1012 6 :FREE))
|#
;;;---------------------------------------- GET-CHEAP-POSSIBLY-FREE-ROOM

(defun get-cheap-possibly-free-room (result)
G1| Saving #P"Odin:Users:barthes:MCL:OMAS-MOSS 7.10:OMAS:
```

Figure 4: OMAS Interactive Development Environment

Using the IDE it is possible to visualize messages, to see the messages received and sent by an agent, to trace a dialog, to open or close connections with remote coteries, etc.

10 Persistency

Persistency is only possible for service agents. One simply adds a persistency option to the macro call creating the agent. The rest is automatic. What is saved is the content of the ontology and the associated knowledge base. To save data one needs simply to wrap the creating code in a `with-transaction` command. Details are given in the User Manual.

11 Web Access

Since for some applications, users not always have an OMAS environment deployed on their local machine, the possibility of accessing applications through a web server has been implemented. Thus, dialogs can be carried out using a web page, or forms can be posted to enter data easily.

12 FIPA Compliance

The OMAS platform has currently some sort of FIPA compliance, although the OMAS architecture is quite different from the FIPA architecture. However, a special transfer agent or postman allows accessing OMAS agents from outside, and outside agent from within OMAS.

OMAS was interfaced with the JADE platform that is a popular FIPA compliant platform written in Java.