MASK : An AEIO Toolbox to Design and Build Multi-Agent Systems

Michel Occello, Christof Baeijs, Yves Demazeau, Jean-Luc Koning

MASK: An AEIO Toolbox to Design and Build Multi-Agent Systems

Michel OCCELLO, Christof BAEIJS, Yves DEMAZEAU, Jean-Luc KONING

LEIBNIZ-IMAG-CNRS,
46 Avenue Félix Viallet, 38031 Grenoble, France
http://www-leibniz.imag.fr/MAGMA/

Abstract. This article presents a development platform for designing and building multi-agent systems called MASK. The platform is organised along a set of four toolboxes, each of them covering one aspect of the AEIO (or "VOWELS") approach. This approach decomposes a multi-agent system into four different bricks: the agents (A), the environments (E), the interactions (I) and the organisations (O). The MASK platform offers for each of these basic entities the possibility to reuse existing models and software components, as well as the possibility to design and build new ones. Throughout this article we present the existing models for the AEIO bricks, we show how they have been integrated in the MASK development platform, and we partly illustrate our AEIO toolbox approach with a concrete example in collective robotics.

1. Designing and Building Multi-Agent Systems

1.1. A MAS platform

MASK (Multi-Agent System Kernel) is a software package to design and build multi-agent systems. This toolbox is used after the analysis stage in which the conceiver works out a detailed solution without taking into consideration the resources eventually required. Multi-agent analysis consists in breaking up a problem into a multi-agent solution. It attempts to specify conceptual agents and determines their skills and knowledge. The analysis phase can be tackled starting from the agents, the interactions, the organisations or the environment. The design stage's purpose is then to lead to an implementation of the envisioned application and to choose how to make the chosen models operational. Multi-agent-oriented design aims at building a Multi-Agent System (MAS) once what the agents have to do is known. A design approach is meant for leading to an operational MAS, i.e., integrating the agents, environment, interactions, and organisation within a MAS starting from the global specification drawn from the analysis stage. The main goal of the MASK platform is to provide the multi-agent system designer with a number of utilities packages embedded in a single software environment.

1.2. An AEIO kernel

MASK (Multi-Agent System Kernel) is the first software package associated to the AEIO (or VOWELS) [1] [2] approach. The MASK platform (cf. figure 1) is composed of packages covering different aspects of the multi-agent paradigm:

- The Agent package provides the user pre-defined agent models or allows the definition of new ones.
- The Environment package provides the user functions to define and to work with a new (simulated) environment, or allows the user to use pre-defined ones,
- The Interaction package is responsible for providing functions to use interaction mechanisms.
- The Organisation package provides the user functions to establish the whole organisation of a multi-agent system.

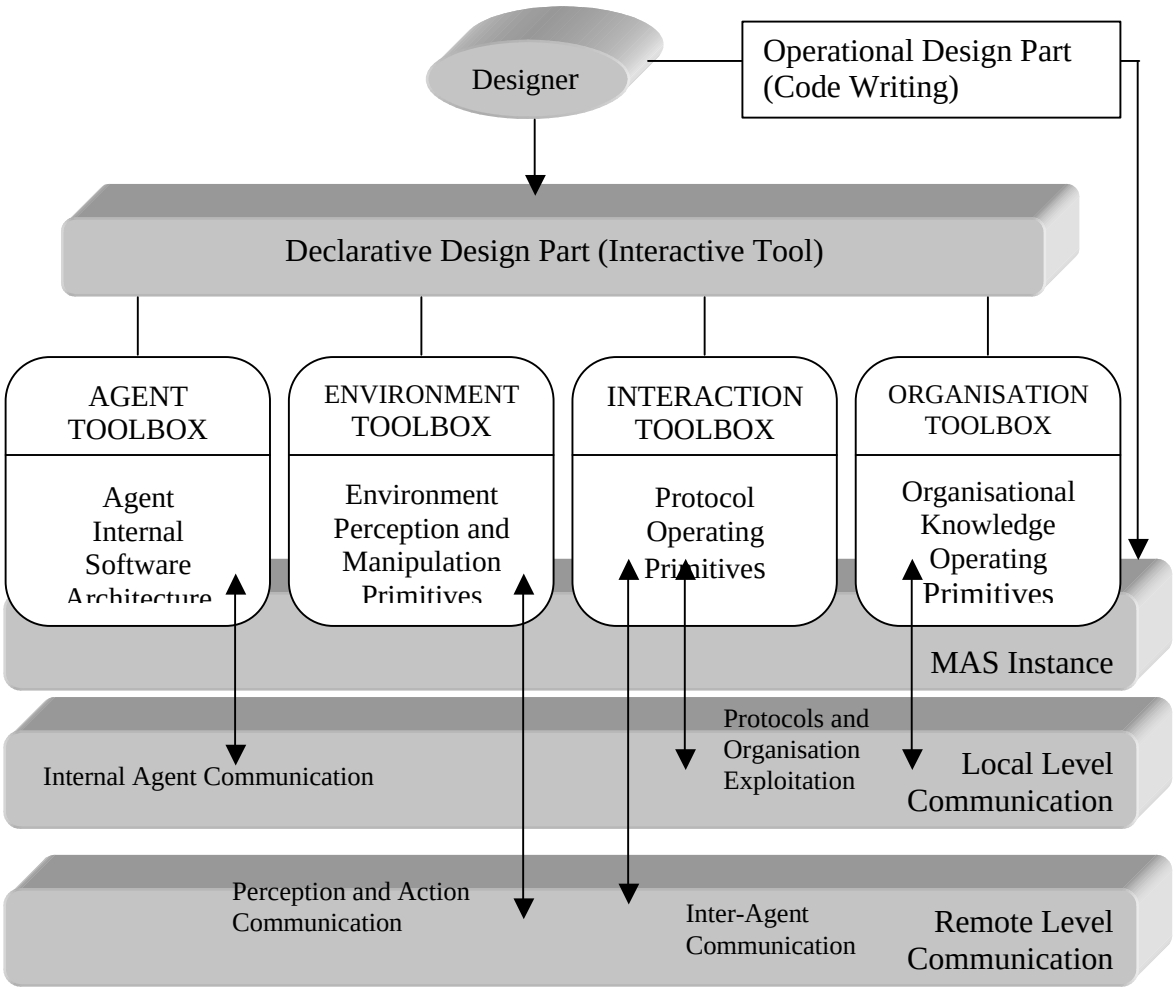

Figure 1: MASK general structure

For each of the MAS notions, the specification can be separated into declarative and operational parts. Declarative parts consist of static knowledge about the agents, the environment, the interactions, and the organisations; it is specified according to the nature of each concept. Dynamic features, such as exploitation of knowledge processes or

decision-making processes, are defined as primitives manipulating declarative data. Each of the four packages supply the following features:

- Editors to define new models of the agents, environment, interactions and organisations,
- Editors to create declarative parts of interactions and organisations,
- Editors to create declarative parts of agents and to integrate interactions, environment and organisation capabilities,
- Libraries to support operational parts of the agents' capabilities

MASK allows building independently each of the basic elements contributing to the operational MAS, according to the different types of approach adapted to the MAS nature. (reactive or cognitive, situated or not ...).

1.3. An open toolbox

A diversity of domains and problems can be addressed using MAS. This diversity does not allow to reasonably think that a single model will satisfy every single requirement. This diversity implies to use a big variety of models for agents, environments, interactions, and organisations:

- Libraries to support operational parts of the agents' capabilities
- BDI, reactive, cognitive or hybrid agents,
- Signals, stimuli, forces, messages with or without protocols or conversational interaction,
- Social or functional organisations,
- Spatialised or not, symbolic or numeric environments.

In each category, we can find several architectures especially suited to solve particular problems. Models can be combined in a variety of ways; the same agent model, being able to involve a set of interaction models, may be necessary for a given application. Choosing a toolbox approach seems thus a better alternative to build systems where components are seen as active cohabiting entities. Our approach is an alternative to a unique customisable multi-function model: it privileges a "multiple unique paradigm models" versus a "unique multiple paradigm model". In terms of the development tool, this idea leads to a very open software environment with which a user can define models developed on demand for a particular problem. The platform is thus in perpetual evolution, and each toolbox is enriched regularly by the integration of new models. For each additional model, the designer must specify with which models of other toolboxes it can be combined. The designer builds the different elements in the order he prefers. The agent (society) window proposes, through the building process of the agents, the integration of the other boxes. In some cases (for example, in a reactive approach when agents are very numerous) the agent building task can be automated. The MASK concept is very powerful as it is completely open and constantly evolving. Up to now a quite reduced set of the available and existing models is installed in the platform:

- (A) : an hybrid agent model ASTRO, a reactive agent model PACORG

- (I): an interaction by message model ILAPI, a force model PACO
- (O): a meta-model of organisation RESO.

New models are currently developed and are going to be integrated, such as a conversational interaction model (I), SMAM a minimal multi-agent model (O), or MID a dynamic interaction model (I).

2. Environments Interactions Organisation Agents

In the following sections, we describe the different toolboxes embedded in the current MASK platform. For each of them we introduce the models behind these basic bricks as well as the operational entity that has been included in our multi-agent system development platform.

2.1. The Environments Toolbox

The environment toolbox is currently certainly the most poorly populated and furnished part of the toolbox of the MASK development platform. The fact that in its actual shape this toolbox is left empty, can be explained by several reasons. First of all, the MAGMA group has not yet conducted any theoretical research on the environmental aspects of multi-agent systems. Furthermore, we do believe that the modelling of the environment highly depends on the application domain. Therefore, as we do not believe in some kind of universal multi-agent system independently from an application and problem domain, the environment can often be considered as the carrier of the data expressing the problem to be solved. This implies that the environment can have its own dynamics and its proper (passive – from a multi-agent perspective) entities. Although there is no environmental model explicitly present in the MASK platform, the instantiated and used environmental model is always present in the resulting multi-agent system but entirely conceived by the designer of the system. The resulting and instantiated model of the environment, in the current set of our applications, spatialised (or spatially grounded in a referential system) and provided with a (most often Euclidean) metrics.

2.2. The Interactions Toolbox

2.2.1. Interactions between Cognitive Agents

2.2.1.1 The IL Interaction Language

As pointed out in [3], if communication is solely described in terms of sending and receiving messages, each agent must be able to infer what the sender intended when uttering a message. If the messages are not structured, this inference could be inefficient. Thus, formal restrictions bind messages that should be structured for the ease of interpretation. As an example, one may employ message types in order that the intention of the sender could be immediately recognised from the message itself. This naturally leads to distinguish between the *Communication Language* and the *Knowledge Representation Language*. The former mainly translates the message from the point of view of distributed systems. The latter surely carries at least the multi-agent domain knowledge (*Multi-agent Language*) which encompasses, as much as possible, the intention of the sender, but also

includes the *Application Domain Language* (e.g., an Application Language for Computer Vision [4]). Our common *Interaction Language* introduced in [1] defines the common vocabulary and its semantics within the system. Its purpose is to support the information exchanges, called *interactions* between the agents. The interactions are exchanges of actions, plans, goals or hypotheses corresponding to the type of information handled within the individual control model. Interactions comply with the following syntax:

<interaction> ::= <communication> <multi-agent> <application>

The Communication Language. The Communication Language consists of different fields used by the communication layer of the system.

<communication> ::= <from> <to> <id> <via> <mode>

The fields are the receiver (from) and the sender (to – the content of this field is an agent's identity or the keyword broadcast) of the message, the identity of the message (id), the type of communication channel to use (via – in some communication systems, several kinds of communication channels are available: one for the emission of a large amount of data, and another one for message passing), and the mode of communication (mode – that is synchronous or asynchronous). This layer of the system is usual. It enables to specify the communication medium.

The Multi-agent Language. It appears quite clearly that, at a given level of meaning, the complexity degree of the interaction protocol – and thus, the complexity of the Multi-agent Language – is inversely dependent on the development degree of the Application Language that is used by the agents to interact within the Multi-Agent System. As we want to adopt a simple Application Language, like a first-order one, that will be adequate for particular domain like the robot cooperation one. We will focus here on a quite complex Multi-agent Language that will lead to complex interaction protocols. Our multi-agent language, gathers all the information related to the multi-agent system. It is used by the dialog functions of the agents. There are three kinds of information : the type, the strength, and the nature of the interaction.

<multi-agent> ::= <type> <nature> <force>

As for the type of interaction, we have adopted the primitives proposed by G. Gaspar [5] to define the message. They consist of the four possibilities: (present, request, answer or inform, the first one enabling an agent to enter a society and to present to the others), which ensures the openness of the system. The last three ones have a quite trivial meaning and will not be detailed here. The nature defines on which agent control layer the content of the interaction has to be taken in account by the receiver, and forecasts three alternatives: decision layer (e.g. goals), command layer (e.g. actions) and observation layer (e.g. facts or hypotheses) [4]. According to us, such a nature field has to be defined independently from the agent model as much as possible. However, it is obviously necessary to know the structure of all agents in order to adequately fill out this field. The strength defines the priority of this information. We borrow its possible values from the Speech Act Theory [6] : going from commanding (highest priority) to mere informing (lowest priority). This grading can be refined, and in fact, we have adopted the sets of labels described by

Campbell and d'Inverno [7] as a basis. From their set of tones, we have extracted a subset that is suited for the different kinds of exchanges we are considering. They allow expressing the following intentions: information seeking, informing, warning, advising, bargaining, persuading, commanding and expressing. This information enables the agent to be informed of the intention the other agent had while sending the message. In this way, one can associate a confidence factor to the received information. These tones also allow explicit control of the information exchanges. This is illustrated for example with the bargaining tone, forcing an agent to answer and to close the communication. Let us point out that one can guarantee the receiving and handling of any message sent by restricting the communication language possibilities, which is very important. As a matter of fact, it amounts to position the strength of a message to its maximal priority, depending on the relative agents' roles within the society. If the sending agent dominates the receiving one, the interaction will be interpreted as an order to obey by the receiver. In order for a message to get interpreted as an order it is necessary that the sending agent be in power to give such order, from a social standpoint. In other words, the strength associated to a message ought to present no antinomy with the role of the agent assembling the message.

The Interaction Protocols. The usage of the multi-agent language enables each agent to extract explicitly from the meaning of the message some information that is useful for the control of the information exchange and for the control of the whole society. Decoupling the intention of the sender from the message itself is a first step but is not enough, since the agents also ought to know how to react to a message, or what to expect after sending a message. All these requirements should be met collectively by a unique framework. We call such a framework for structuring interactions among agents "Interaction Protocols". Each type of message in a multi-agent language is in fact associated with a distinct "basic protocol" that could be chosen by the receiver from the type of message, as presented in Burmeister's paper [8]. When talking about "Interaction Protocols", we mean that we add a set of protocols that comes to *restrict* the different interactions an agent can link with other agents with regards to some problem to solve.

An Interaction Protocol Example. As an illustration let us exemplify this with a learning protocol taken from [9] where tentative hypotheses are inferred by individual agents through the use of some induction on examples of events occurring in their environment. These tentative hypotheses are treated as opinions of individual agents, and interactions among them aim at finding the most consistent form of the hypotheses.

Details of the principle components of Sian's model are left out for brevity. We only rapidly detail the cooperation part that leads to implementing an original and simple negotiation formalisation. Sian defines nine operators available to agents and limits their use by imposing a possible sequencing among the message types

An interaction either begins with the proposition of a new hypothesis to all other relevant agent ("propose") or a non-modifiable assertion ("assert") imposing the agent's viewpoint. In this latter case there's no other choice for the concerned agents than to accept the current hypothesis ("accept"). To the "propose" operator the responding agents can enter in a "modify"-"propose" loop or either "confirm", "disagree" or have "noopinion" about the current hypothesis. A confidence factor is computed for each use of an operator. Depending on this factor the hypothesis is "agreed" on or "withdrawn" from the agents'

databases. To become fully accepted such a hypothesis should be accepted by any single agent.

The corresponding interaction protocols could be written in the following manner:

```
Protocol proposal {
  state init {
    [inform(broadcast) (Information_seeking) (matter=propose, rule) ->
state opinion]; }
  state opinion {
    [inform(broadcast) (Information_checking) (matter=modify_into, rule,
new_rule) -> state init] |
    [inform(broadcast) (Expressing) (matter=confirm,rule) -> state
decision] |
    [inform(broadcast) (Expressing) (matter=disagree,rule) -> state
decision] |
    [inform(broadcast) (Informing) (matter=noopinion,rule) -> state
decision]; }
  state decision {
    [inform(broadcast) (Warning) (matter=withdraw, rule) -> state end] |
    [inform(broadcast) (Expressing) (matter=agree, rule) -> state
agreement]; }
  state agreement {
    [inform(broadcast) (Expressing) (matter=accept, rule) -> state
agreement]; }
  state end {
    [end]; }
  }

Protocol assertion {
  state init {
    [inform(broadcast) (Commanding) (matter=assert, rule) -> protocol
proposal state agreement];
    }
  }
```

2.2.1.2 The IL Interaction Toolbox

The previous section has showed how communication among agents can be structured. The present section presents an application-programming interface for IL called ILAPI. ILAPI's first goal is to contribute in the development of the multi-agent system design platform MASK by providing its interaction toolbox that will then be used in the implementation of numerous applications. The next section will show a concrete application with the contract net protocol. ILAPI satisfies various criteria such as portability, extensibility, generic and simplicity. First, it allows for the development of heterogeneous agents without the taking into account of this communication medium. One needs to be able to change the communication medium without having to change the agent's code systematically. Second, since the IL message structure is not fully stable, ILAPI must be adaptable to a future evolution. Third, ILAPI must allow for the observation of how interactions are handled in any multi-agent system making use of this toolbox. Fourth, it stay as simple as possible. One does not thrive for performance whether it is on the communication speed level of the robustness in case of hardware or software failures. ILAPI is geared at multi-agent system developers. It offers a standard interface for operating interactions among agents according

to the IL specifications. Functions this interface offer may be gathered into two categories:

- Communication functions: enter a society of agents, send or receive messages, exit the society of agents.
- Dialog functions: create an IL formatted message, structure message passing through interaction protocols.

A third category of functions is forecasted in order to trace the interactions of the system, which would enable to analyse the interaction protocols as well as study the dependencies between the agents. In order to implement these various functions categories, ILAPI is structured into three layers. The ILAPI communication layer handles communication functions. The ILAPI dialog layer handles dialog functions. The third category is handled by ILAPI-demonstration. Depending on their needs, agents make use of any of ILAPI's layers. In case of porting a multi-agent application to another communication medium such a structure has the advantage of having to rewrite only the communication layer. This complies with ILAPI's first criteria.

The ILAPI-communication Layer. An agent needs four basic primitives in order to communicate:

- Connect: declare its presence to the other agents.
- Disconnect: declare its absence to the other agents.
- Send: send a message to a given agent.
- Receive: receive a message from an agent.

The communication layer consists in the implementation of these four primitives along with the services offered by the layer supporting the communication . In the present case, it is based on TCP/IP, whose services can be accessed via the socket interface. We chose to use Java sockets to support the communication with applets to send and receive messages. Sockets allow the exchange of information assuming that the various processes addresses are known. However, agents in a multi-agent system can enter and exit the system at any time. Therefore, a name server is necessary in order to act as an entry point and centralise the addresses of all the agents present in the system. In order to enter the system an agent will have to know this name server's address.

The ILAPI-dialog Layer. The ILAPI-dialog layer corresponds to the set of functions allowing an agent to format messages and conversations according to the IL specifications. The extensibility criterion has consequences on the message structure, since such a structure may vary depending on the application. Handling and operating interaction protocols has been based on a *protocol description language* introduced in [10] and extended in [11]. Such language expounds a universal language that can be used to describe a set of protocols. It has been shown it is generic enough to describe a number of protocols that have been presented in the literature. PDL allows for the coexistence of different versions of the same protocol. This introduces dynamics to the protocols used in the society, since they can be improved. This language provides a textual representation of the various transitions accessible from a state of a protocol. Each transition determines the values to be given to the fields' type, strength and application of the associated message. Let us note that this language cannot convey which behaviour to adopt in case of an

absence of answers, or at what time one should consider there is no answer. This behaviour depends on the agent. Agents refer to files holding the protocols' textual descriptions in order to adapt their behaviour. In order to handle these protocols an agent needs the following primitives: getting a protocol, getting protocol states, getting state transitions.

The ILAPI-demonstrator Layer. The demonstrator layer consists in providing a tool for graphically handling interaction protocols. Such a tool can also be used for developing functions such as displaying used interaction protocols as well as dependencies among agents. This graphical tool enables a user to choose a protocol and to get its diagram displayed (see section "Monitoring the Information Exchange"). Pieces of information attached to a transition such as its condition of application, the speech act type, the applied strength, the content and the ending state.

2.2.2. Interactions between Reactive Agents

Instead of considering the search for a solution for a given problem as some kind of optimisation – minimisation – of a global energy function, the PACO paradigm [12] proposes to model this search as the co-evolution of a finite set of agents. Each agent represents an entity that takes part to a partial solution of the global problem to be solved, although none of them knows when this global solution has been reached as only an external observer is able to detect this overall solution – or stability state – at the MAS level. Usually, interactions between PACO agents are modelled as forces determining the displacements of the agents in the environment, the schema for combining these forces being defined *a priori* by the conceiver of the multi-agent system. Within the PACO paradigm, the agents do not hold a representation of themselves nor from the other agents or the environment, although the agents are able to distinguish other agents form objects within the environment. A simple modification of the input data (the local environment of the agents) provokes an immediate reactivation of the agents searching for a new stable position, first at the local agent level and by propagation through the interactions at the society level. This means that the search for an equilibrium state can be considered as being adaptive. The behaviour of the agents, compliant with the PACO paradigm, is characterised by a combination of elementary interactions defined a priori. Each type of interaction is therefore linked to the agent's capability of perceiving a type of agent or a given object within the environment. Based on the notion of potential fields, the PACO paradigm introduces three types of scopes: a perception scope, a communication scope and an action scope. The perception scope is a reference to what is visible within the environment for the agent (his local environment). The communication scope determines with which other agents the agent may eventually start interacting, while the action scope defines the space in which the agent may move. The explicit and local control of the different scopes by the agent itself, allows it to constrain the set of possible interactions with the other agents (communication scope), the environment (the perception scope) or his displacement actions within the environment (actions scope). The intensity of the perceived information together with the desire to interact with an other agent or object within the environment (which is translated by the explicit control of the perception and/or communication scope), fires off, if permitted by the action scope of the agent, either an interaction with the other agents of the society or an action in the environment. Typically, the force models instantiated within the PACO paradigm correspond to models inherited and instantiated from the physics domain, such as spring forces or attraction/repulsion forces [13].

2.3. The Organisation Toolbox

The organisation toolbox is currently limited to the RESO (Representations of Structures for Organisations) model [14]. RESO is a generic model and design framework allowing expressing organisational structures based on the notions of groups and the corresponding relationships between the entities participating in the organisation. This model integrates the structural aspects of an organisation, the "organic structure", linked to the notion of recursion as expressed in [15], and the relational aspects, expressed in the "interactional structure". We define the organic structure of an organisation as the set of relationships that allow describing the decomposition of an agent (or group) into more elementary agents (at a lower level of granularity). In order to do so, we have to take into account the recursive aspect of the structure of the multi-agent system. We recall that for a given level of granularity, every single entity (agent, group of agents or the multi-agent system itself) of the multi-agent system can be considered as being a single entity or as a complete multi-agent system. When creating the organic structure for such a kind of static organisation, the conceiver of the multi-agent system must have a complete *a priori* knowledge of the kinds of agents that are present in the system, as well as the relationships that bind them (recursion and decomposition mechanisms between the levels). The interactional structure then corresponds to the set of relationships that enables the designer to set out the framework within which the agents may interact. This interactional structure is defined by three kinds of relationships: the acquaintance relation (who do I know), the communication relation (whom can I communicate with), and the subordination relation (who am I controlling). In order to link both structures expressing an instantiated organisation, we combine the interactional and organic structure through the different agent types (or kinds / roles to be fulfilled within the system). This leads to a two-step process for the designer of an organisational model using the RESO model and toolbox: first he has to specify the organic structure (which are the elementary agents, how are they combined into groups), and then he has to complete his organisational model by instantiating the interactional structure (who knows who, who can communicate with who, who is controlling who). This process then leads to an instantiated organisational structure (populated with the actual agents), as normally called "an organisation". The RESO toolbox proposes a graphical interface to specify relationships, and primitives for the exploitation of the organisation knowledge.

2.4. The Agent Toolbox

2.4.1. Integrating Deliberative and Reactive Capabilities

2.4.1.1. The ASTRO Model

The integration of deliberative and reactive capabilities is possible using parallelism in the structure of the agents. Separating Reasoning/Adaptation and Perception/Communication tasks allows a continuous supervision of the evolution of the environment. The reasoning model of our agent is based on the Perception / Decision / Reasoning / Action paradigm. The cognitive reasoning is thus preserved, and predicted events contribute to the normal progress of the reasoning process. ASTRO can be presented as a disintegrated agent [16], where a functional decomposition in terms of capabilities provides a modular approach to the model. Decision modules evaluate the importance of the unpredicted events and have

the obligation to place new actions or new goals in the internal state of the agent's reasoning. New goals imply the activation of the reasoning modules in order to partially or totally re-plan according to the importance of the event. New actions are placed on the agenda of actions directly in order to be executed in the specified delay. We now describe the different modules needed by such a deliberative/reactive agent .

Representation of the World. The central part of the agent is its world model. This model comprises its knowledge about the environment, the internal states of other agents, and its own internal state. In particular, the proper internal state includes the plans to be executed or which the agent takes into consideration. An interpretation process of the sensory data maintains the model.

Perception and Communication Modules. Evolving in a real world, each agent has to integrate perception capabilities realised through sensor devices. The perception modules assemble the knowledge about the environment. Other agents are perceived through communication modules. Agents can send information about their knowledge of the environment, their plans, their goals or their current state. Communication modules are probe loop events waiting for messages from agents. Emitters are considered as actions.

Control Modules. To ensure the reactivity of the agent, an evaluator continuously examines the world model. Agent control modules detect situations to which the agent needs to react, evaluate them, and decide to take the appropriate actions which may be to create, suspend, or kill goals, i.e. to change the context of the planning and executing process. The continuous supervision of the agent's situation ensures that the agent can react to unpredicted events at any time. The role of the perception evaluator is rather similar to the Perceptual Schema Controllers of the AuRa architecture of Arkin [17]. But additionally, we introduce similar mechanisms to take into account the interactions with other agents using the interaction protocols proposed by [1]. Triggers and guards can be control loops observing the world representation or an evaluation function launched by the occurrence of perception or communication events.

Reasoning Modules. The reasoning process consists of planning, scheduling and sequencing modules. Whenever a goal is created (or modified) a plan is searched that realises the goal, this task is realised by planning modules.

We first describe the structure of the agent plans. Each plan has an identifier and an associated deadline. Execution of the plan requires the execution of several local goals. Each local goal has an identifier and an associated priority. The execution of each local goal may be accomplished by executing one of the several alternative actions. Each action has a duration (the execution time) and a satisfaction value associated to it. Plans related to a given goal are stored in the part of the architecture concerning the internal state of the agent. The planner details the action in the order according to which they will be executed. This process may be guided by hierarchical planning trying to infer the sequence of actions in a top-down fashion. In simple applications, we may assume that the agent has plans for every possibly encountered goal and possesses all the necessary actions; the planning process is in this case reduced to a fast pattern-matching algorithm. For complex applications, involving more social organisations, agents can negotiate with other agents about the actions they are not capable of, according to [18]. The purpose of the scheduling

algorithm is to schedule the actions in the first place, to meet the deadline and obtain a maximal utility value and a maximal satisfaction value. To achieve this, we have established an algorithm informally described below.

(1) For each plan, make a schedule of all the local goals taking minimal duration. If a deadline is violated at any stage, abort the plan and exit.
(2) For a local goal with maximal priority from the remaining local goals:
    (a) if (action has already started) then go to step (b) else find the action with maximal satisfaction from the remaining actions
    (b) if the deadline is violated due to some new action go to step (a) else replace action with minimal duration by action with maximal satisfaction

Assuming that actions with higher satisfaction values require more execution time, this algorithm ensures that a schedule (if it exists) for meeting the deadline with a maximal satisfaction is made. The actions start executing according to the schedule. Then the highest priority local goal is taken and a higher satisfaction action is put in the schedule provided that the action has not already started and that it does not violate the deadline. This process is continued for the remaining local goals. Therefore, we provide to the agent an immediate schedule to start actions. Then try to improve it by further iterations, allocating more resources to actions having more value. At anytime the algorithm has a current valid schedule. We thus have used a hybrid any time / design to time approach to the scheduling mechanism. Actions are placed in an agenda. Furthermore, the scheduler has the possibility to include internal actions, such as replanning or the setting of guard and trigger modules, in order to be more adaptive at run time. Guards and triggers supply information about the current situation during the execution phase of the actions. Due to this technique, if the agent has to act fast, the scheduling and the execution of an incomplete plan can start before the planning process is completely finished. This ensures adaptation of the agent to the evolution speed of his environment and is necessary if the agent pursues several goals at the same time. The committed actions are performed at the scheduled time by the action modules triggered by the sequencer module (executor).

2.4.1.2. The ASTRO Editor

The ASTRO agent model has been implemented using a real time blackboard architecture. A parallel blackboard aims at expressing the inherent parallelism of the conceptual blackboard model [19] as modules react to modifications of the blackboard, for their activation and inhibition. They work on a local context that is a part of the blackboard data. A domain blackboard contains domain data (used for the problem solving process). A control mechanism is in charge of the communication between modules and of the control of the management of the modules' activity. Control data (summary of the state of the solution) are stored in a Control Blackboard managed by the control unit. Modules communicate with the control mechanism through event streams. The control unit sends a control stream to the modules. The controller manages all the communications. This blackboard control unit ensures stimulation and inhibition of the modules following their specifications.

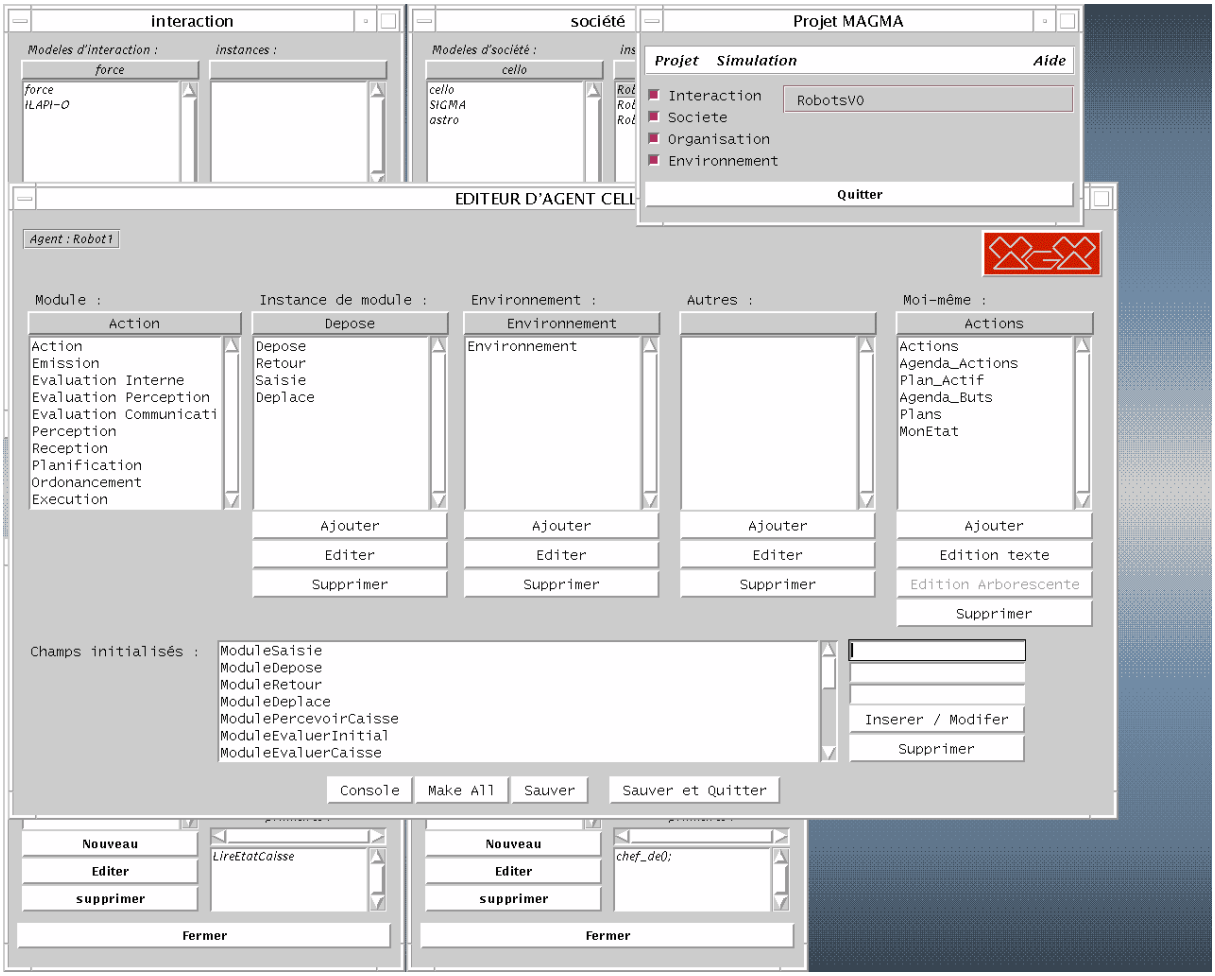

Fig. 2. The ASTRO editor

The behaviour of a module is described through its interactions with the control data, i.e. a representation of this behaviour is given. This behaviour is managed by message exchanges with the control unit. A module is integrated in the system by the specification of its behaviour when faced to the blackboard data. An external specification of the behaviour of the module can be expressed by an objective, preconditions of activation and interruption conditions. The control unit receives events from modules and emits control signals to them. Modules that have all their conditions validated are activated by an activation flow. Inhibition signals trigger exception processing in the modules. The control unit is application independent. We detailed and formalised this system in [20]. The different modules of the agent are organised according to the blackboard model described above. The model of the world constitutes the domain blackboard. The control unit manages all modules. This multi-module approach allows a modular and independent description of each of the action and perception tasks in separate modules. The communication primitives can be of two types : internal communication (between the control unit and each module) and inter-agent communication (managed by the model of interaction). The blackboard kernel has been written in C++ using UNIX communication libraries (cf. figure 2). A generic tool has been developed in Tcl/Tk. The designer can specify through editors declarative (using given languages) aspects:

- The description of actions and plans ,
- The description of knowledge on the environment (using the corresponding Environment Toolbox Editor), on the other agents,

as well as operative aspects (using C++ or an other language supplying UNIX executable code) :

- The perception and action modules (integrating primitives of the Environment Toolbox Libraries)
- The interaction modules (integrating primitives of the Interaction Toolbox Libraries as protocols operating primitives or message exchanges primitives).

2.4.2. Reactive Agents within Organisations

A multi-agent system based on the PACO paradigm complies with the VOWELS approach as set out in [2]; we can therefore specify this model through its four elementary axes. The core of the agent model is largely based on the PACO model (for further details we refer to [12]) and the RESO model as described before. Every agent is denoted by an identifier, his mass (representing its relative weight with respect to the overall solution), his position, velocity and acceleration. Every agent controls autonomously and locally a set of scopes for perception, communication and action. We constrain the set of scopes of the agents by introducing organisational knowledge (both on the structural aspects and the interactional aspects, as provided by the RESO model). We use on the one hand this organisational knowledge to control the communication and perception scopes of the agents through the use of the acquaintance and communication relationships, and on the other hand, we use the subordination relationships between the agents to constrain the action of the agents. We can therefore clearly see that the organisational knowledge within the multi-agent system (distributed among the society of reactive agents) takes into account the input side (constraining perception and communication scopes) as well as the output side (constraining the action scopes). Combining PACO and RESO allows us to model, conceive and build reactive multi-agent systems with static organisational structure that are conceived in advance. Using the organisational knowledge is then done through the structural aspect of the instantiated organisation by using it as a legal framework within which the interactions between the agents take place.

3. Combination of Models and Dynamics of the System

This section describes the approach we apply to build the MAS through a practical example. The main point of interest is the way we integrate the interactional and organisational knowledge in the reasoning process and so realise the dynamics of the MAS. The mineral extraction collective robotics experiment concerns a simulation of robots whose mission is to find and extract minerals from a given zone. Robots are of three types according to their capabilities :

- Exploration robots in charge of the detection of a site containing minerals,
- Drilling robots able to dig the ground to extract minerals,
- Carrier robots moving minerals from the extraction site to the base camp.

None of the robots are able to complete alone the entire task. Co-operation is needed between different kinds of robots in order to achieve correctly the extraction process. We can solve the problem by exploiting organisational and interactional knowledge. We explain how we build the multi-agent system in the following sections.

3.1. Combination of the models

3.1.1. An agent centred phase

3.1.1.1 Scheduling of tasks

The internal aspects of the agent's reasoning process, i.e. everything that the agent would have to do if he was alone, must be first analysed. It is an agent centred phase and consists of elaborating a set of plans. The set of actions needed to accomplish the mission has to be defined. These actions are scheduled without taking external influences related to the work of other agents into account. This plan expresses the normal progress of the work the robot has to do for a given goal:

- A plan can include actions aiming to modify the environment or to acquire information of the environment,
- A plan can include actions for information exchange between agents. It is possible to define actions the agent will delegate during execution [18],
- A plan can include initialisation of interaction processes with other agents.

In the case of our experiment, we have simple plans to realise goals.

Exploration robots. Goals are Exploration and Request of Extraction. For exploration, the plan consists of moving the robot until it detects some minerals. The <Perception> action refreshes a set of indicators on the characteristics of the site. The <Analysis> action evaluates these indicators and if necessary can create a goal called Detected, which will be realised by the action <Ask-for-Extraction> initiating an interaction to find an extractor.

- Exploration : <Moving>// <Perception> // <Analysis>

Drilling robots. Goals are Extraction and Request of Evacuation. For Extraction, the plan consists of moving the robot to the site and to extract the minerals. The <Drilling> can create a goal called Extraction Failed or a goal called Extraction OK. This last case will trigger the action <Ask-for-Evacuation> initiating an interaction to find a conveyor.

- Extraction : <Moving>; <Drilling>

Carrier robots. Goals are Evacuation. The plan consists of moving the robot to the site, loading the mineral and moving back to the base camp.

- Evacuation : <Moving>; <Loading>; <Moving>

Representation of the environment and of the others. In most cases, actions work with data available in the representation of the environment perceived by the agent. A definition

of perception capabilities is then needed in order to maintain the state of the perceived environment. In our application, the environment is constituted by the zone map and by site characteristics (list of indicators used by the <Analysis> action and refreshed by the <Perception> action for exploration robots, co-ordinates of sites for the other types). In the case of the delegation of tasks, agents that are able to execute these tasks have to be specified. A representation of the capabilities of the other agents has therefore to be built. For each known robot, we can store its type and current localisation.

3.1.2. A society oriented phase

3.1.2.1 Expression of the organisation.

Our approach considers an external representation of the organisation of the MAS. We use the RESO model [14]. In our application, we instantiate a hierarchical organisation for the robots and we can adopt two approaches:

- Grouping robots into fixed teams composed of an explorer, an extractor and several conveyors (cf. figure 3),
- Grouping dynamically robots according to their types, their current localisation and tasks, and the current needs of the explorers.

As the organisation is external, this choice has no impact on the programming of the <ask-for-extraction> action, only the set of candidate robots will be different.

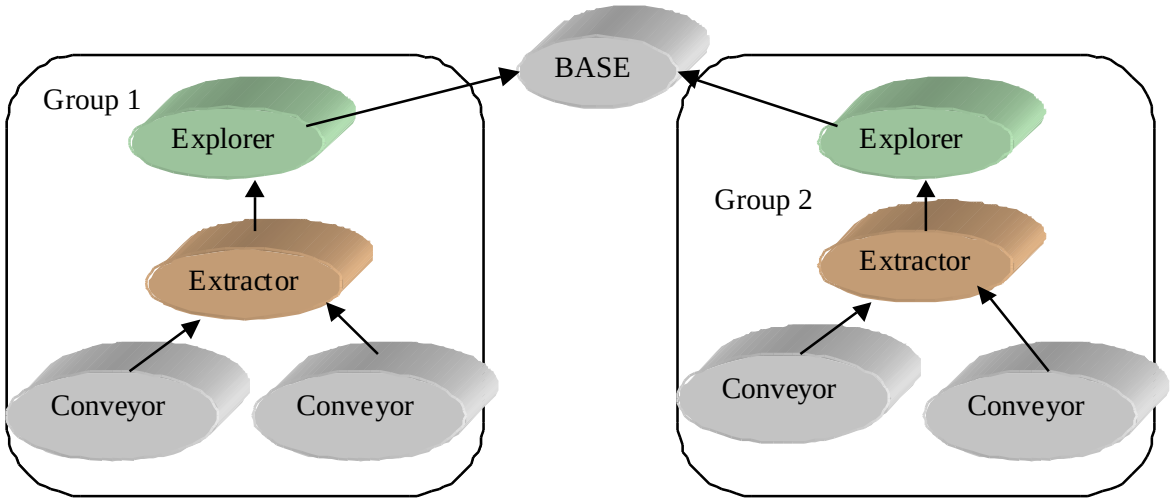

Fig. 3. The organisation for the mineral extraction example

3.1.2.2 Expression of the interaction.

The interaction mechanisms using interaction languages with protocols is without any doubt the best adapted approach for our kind of agent model, even if other mechanisms can be used. ASTRO agents use the IL language [1] based on speech acts as IL uses protocols. We need two kinds of interaction processes in our application: the exchange of data (co-ordinates of the sites) (Request/Answer Protocol), the negotiation about task allocation (Contract Net Protocol).

Implementing the Contract Net Protocol in IL. The use of the contract net protocol [21] in our application domain authorises the extraction system to be configured dynamically, when taking into account factors as the number of exploration, the mining robot and the carrier robots that are available, their locations, and the ease with which communication can be established. The contract net is an adequate mechanism for task allocation. It is founded on the contract procedure used in public markets. A contract is established by a process of local mutual selection based on a two-way transfer of information between a *manager* and a *contractor*. A manager is responsible for monitoring the execution of a task and processing the results of its execution. A contractor is responsible for the actual execution of the task. Agent interaction is viewed as an agreement between one agent with a task to be performed and one agent capable of performing that task. The overall process is realised through four phases:

- Task announcement: managers make task announcements to all contractors. With respect to the IL language, such (broadcast) message is of type request.
- Task bid: available contractors evaluate the task announcements received and submit bids on tasks for which they are suited. This message is of type answer.
- Task award: the managers evaluate the bids and award contracts to the agents they determine to be most appropriate. This message is of type inform.
- Task commitment: the awarded agent becomes the contractor and commits himself to fulfil the contract. This message is of type answer.

A negotiation process may then recur. A contractor may further partition a task and award contracts to other nodes. It is then the manager for those contracts. That is exactly what takes place in our application domain where the ultimate goal is to transport to the base camp as much minerals as possible. The contract net partitions this general task into two subtasks. The first subtask is the *extraction* task. It involves gathering of information about the site and mining of ore. The managers for this task are exploration robots that do not have mining capabilities, but do have extensive sensing capabilities. They attempt to find a set of mining robots to extract the ore. The mining robots, on the other hand, have limited sensing capabilities and attempt to find managers that can further find mining sites. The second subtask is the *transportation* task. It deals with carrying the ore from the mining site to the storage location. Mining robots dynamically take on a manager role during the course of problem solving, and carriers robots contract out such tasks. Let us examine the negotiation for the *extraction* task. For example, from the perspective of the *extraction* task managers (i.e., exploration robots), the best set of contractors has an adequate spatial distribution about the surrounding area. From the point of view of the *extraction* task contractors (i.e., mining robots), on the other hand, the best managers are those closest to them in order to minimise potential communication problems and travelling time. The ability to express and deal with such disparate viewpoints is one advantage of the contract net protocol. To see how the appropriate resolution is accomplished, consider the messages exchanged between the *extraction* managers and potential *extraction* contractors. Each *extraction* manager announces its own *extraction* task, using a message of the sort shown here.

Message: task announcement
To: * // indicates a broadcast message
From: Explorer-10

Contract-id: Mining-Explorer-10-1
Task Abstraction:
Task type: Mining
Eligibility Specification:
Must-Have: mining capabilities
Must-Have: position area A
Bid Specification:
Position: latitude, longitude
Mining Features: type of soil, performance
Expiration Time:
1999-09-30 14:15:50

Each message in the contract net protocol has a set of slots for the task-specific information in the message. The information that fills the four slots of the task is encoded in a simple language common to all nodes. It corresponds to the application part of IL messages (see section 2.2.1.1). The task abstraction is the type of task and the position of the manager making the announcement. The position enables a potential contractor to determine the manager to which it should respond. The eligibility specification indicates that the only robots that should bid on this task are those which (1) have mining capabilities and (2) are located in the same area as the manager that announced the task. This helps to reduce extraneous message traffic and bid processing. The bid specification indicates the information that a manager needs in order to select a suitable set of mining robots — the position of the bidder and the performance and type of soil to be dealt with. Finally, the expiration time is a deadline for receiving bids. Each potential contractor listens to the task announcements made by *extraction* managers. It ranks each announcement relative to the others thus far received, according to the distance to the manager. Just before the deadline for the task announcement associated with the perceived nearest manager, the node submits a bid.

Message: task bid
To: Explorer-10
From: Mining-5
Contract-id: Mining-Explorer-10-1
Bid Abstraction:
Position: 62N, 9W
Mining Features: sand .5, clay .3, schist .8

The bid message supplies the position of the bidder and a description of its mining capabilities. A manager uses this information to select a set of bidders that covers its area of responsibility with a suitable variety of mining capabilities, and then awards an *extraction* contract on this basis.

Monitoring the Information Exchange. This interaction protocol can be modelled by a state-transition graph representing agents going from one conversational state to another. Such representation shows the possible message types an agent is allowed to send in any given situation. Edges (transitions) exclusively represent sent messages. On the other hand, nodes represent states of the agent which sends the message or messages indicated by outgoing edges. Thus, for instance, in a two-agent interaction, the nodes alternatively represent one agent and then the other on any path going from an initial node to a final one. By the way, such a path represents a sequence of messages corresponding to a complete interaction between these agents. The main difference with Petri nets is that the next state cannot be determined by an external decision. That is, such decision is not inferred from

the agent's current state and the general network state, but the agent evaluates by itself and makes its own decision. The contract net protocol just described can be modelled as in figure 4.

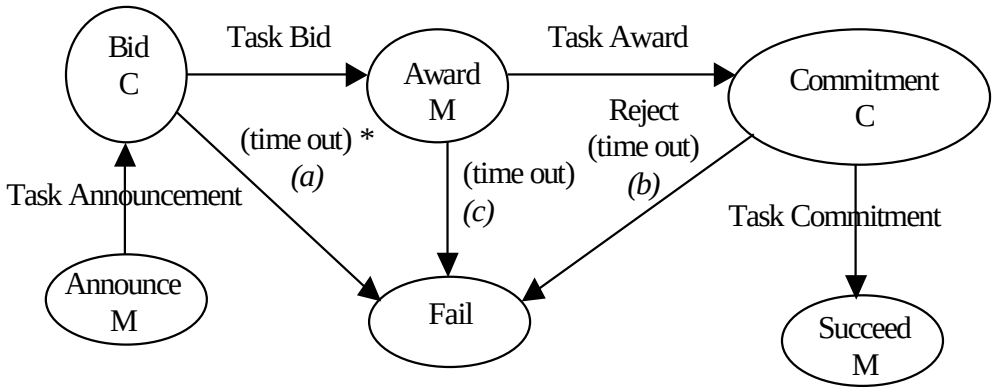

Figure 4.: The contract Net Protocol.

The horizontal sequence of messages corresponds to the normal course of the message exchange, in case of a successful task award for a contractor. Transition labelled *(a)* corresponds to a contractor not answering in time (before expiration) the task announcement. Transition *(b)* corresponds to a contractor rejecting the task award or not committing to the task in time. Transition *(c)* is traversed when a manager does not award the task a contractor who bids.

3.2. Dynamics of the System

The evolution of the environment, the interactions between agents and the organisation of the MAS have an effect on the progress of the mono-agent plan built at the beginning of the design process. The influences on the initial plan have now to be analysed. These influences are integrated in the agent decision making process by creating the capabilities of evaluation of the protocols' messages and of the perception detailed in the agent model as reactive elements. Consequences of these influences are diverse: reaction of adaptation, reaction on the plan progress (suspend action, suppress action, etc), and internal reaction: plan, new goal, etc. In our experimental context, the progress of the protocol law will be constrained by the social status of the participant expressed in the organisation. Before passing a transition, the reasoning process has to consider the relation between the agents involved. The dynamics of the system is so realised. The ASTRO Toolbox allows creating each capability of the agents by including primitives from ILAPI and RESO libraries. The external description of protocol laws and of the organisation are built with the aid of the associated graphical tool and exploited by the agent modules using corresponding primitives. Figure 5 shows all the relations between the elements.

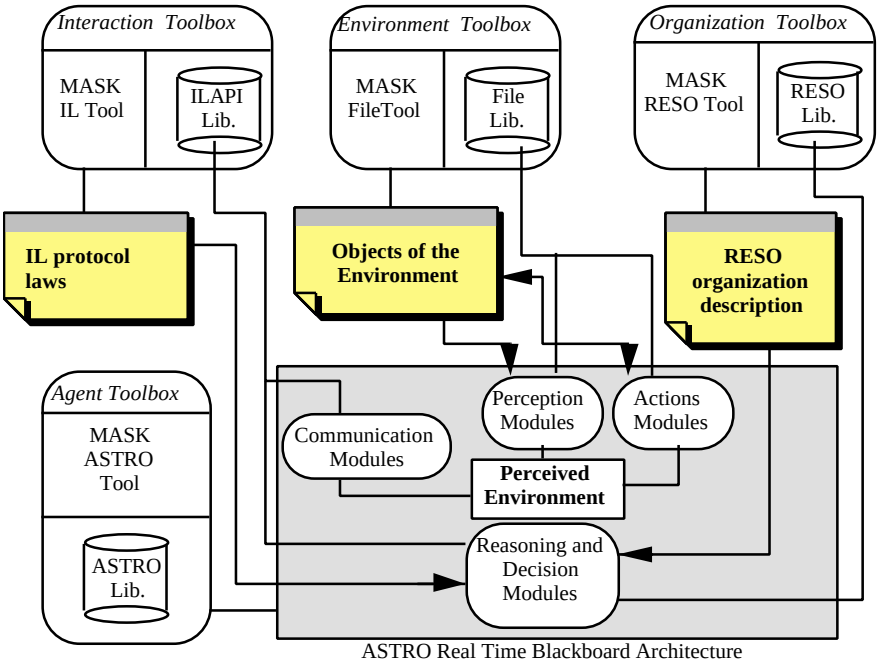

Figure 5.: Using MASK to build ASTRO agents.

4. Comparison, Discussion, and Perspectives

More and more academic and industrial MAS tools are nowadays developed. Until now, these tools proposed generic customisable architectures often supplied as program libraries. Recently, the emergence of agent and multi-agent oriented programming methods has led to consider at the same time analysis and programming aspects of a MAS. Although the notion of MAS life cycle is not yet totally defined, the tools now evolve towards real multi-agent software engineering environments. Their performance can be estimated not only in terms of their technical architecture and capabilities but also with respect to their coverage of the life cycle. We can evaluate MASK features by comparing this toolbox with related work according to these characteristics. MASK has been built as a first software support for the VOWELS method, to design and to build multi-agent systems. Some tools, indeed, forget one or the other of these two life cycle phases. The Alaadin method [22], for example, privileges the design phase but the associated Madkit tool makes the user responsible for the entire implementation work, only supplying the user with organisational primitives. On the other hand, tools as JACK (based on the dMars model [23]) emphasise the programming aspects without addressing design phases. Different characteristics are required for MAS components in order to build the appropriate MAS that will be well suited to problem and domain specifications. One of the main features of MASK is to adopt a multi-model approach. The environment allows choosing among different kinds of models of Agent, Interaction or Organisation in order to instantiate an operational MAS, whereas most of the other existing platforms are based on a unique model. Some of them attempt to solve this problem by supplying multi-paradigm agent models able to satisfy all requirements [24]. One of the drawbacks of MASK is its lack of embedded simulation and validation tools, even if some work is done currently in this direction to complete the platform, especially in the domain of executable specifications or interaction protocol validation. At the tool level, a main advantage is the flexibility of the overall architecture guaranteeing the possible evolution of the tool. MASK allows the user to add user-defined

models of AEIO. Even if everything is not agent as in MACE [25], the toolbox for each entity can be seen as an assisting process coupled to a component library. Considering AEIO as components is a rather recent and original approach, as components are used generally to build agents only [26]. Compared to industrial products, MASK proposes a quite poor man-machine interface, but a visual programming approach as in ABE can be envisioned to make the tool more attractive. At the level of the execution platform, MASK supports real distribution among workstations., but MASK does not propose any deployment tools. In this domain, the more complete platforms are those dedicated to mobile agents such as Voyager or Zeus, even if their objectives are rather different and do not care much about methodological preoccupations. The MAGMA group is currently working in revisiting the MASK platform, to include such deployment tools.

5. References

[1] Y. Demazeau, “From cognitive interactions to collective behaviour in agent-based systems”, In Proceedings of 1st European Conference on Cognitive Science, Saint Malo, France, 1995.

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

[3] B. Burmeister and K. Sundermeyer, “Cooperative problem solving guided by intentions and perception”, In Werner, E. and Demazeau, Y., editors, Decentralized AI, volume III, Amsterdam, The Netherlands. Elsevier Science Publishers B.V., 1992.

[4] O. Boissier and Y. Demazeau, “A Multi-Agent Control Architecture for Studying the Control of an Integrated Vision System”, IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, MFI '94, Las Vegas, Nevada, 1994.

[5] G. Gaspar, “Communication and belief changes in a society of agents: Towards a formal model of autonomous agent”, In Demazeau, Y. and Müller, J.-P., editors, Decentralized AI, volume II, Amsterdam, The Netherlands. Elsevier Science Publishers B.V., 1991.

[6] J. Searle, “Speech Acts: An Essay in the Philosophy of Language”, Cambridge University Press, Cambridge, 1969.

[7] J. Campbell and M. d'Inverno, “Knowledge interchange protocol”, In Demazeau, Y. and Müller, J-P., editors, Decentralized AI, volume I, pages 63--80, Amsterdam, The Netherlands. Elsevier Science Publishers B.V., 1990.

[8] B. Burmeister, A. Haddadi and K. Sundermeyer, “Generic, configurable, cooperation protocols for multi-agent systems”, In Castelfranchi, C. and Müller, J-P., editors, From Reaction to Cognition, volume 957 of Lecture notes in AI, pages 157--171, Berlin, Germany. Springer Verlag, 1995 (Appeared also in MAAMAW-93, Neûchatel)

[9] S. Sian, “Adaptation based on cooperative learning in multi-agent systems”, Decentralized AI 2, Y. Demazeau and J.P. Muller eds, Elsevier Science Publishers, 1991

[10] P. Populaire, Y. Demazeau, O. Boissier and J. Sichman, “Description et implémentation de protocoles de communication en univers multi-agents”, In 1ères Journées Francophones sur l'Intelligence Artificielle Distribuée et les Systèmes Multi-Agents (JFIADSMA-93), Toulouse. Afcet &Afia, 1993.

[11] R.H. Bordini, “Contributions to an Anthropological Approach to the Cultural Adaptation of Migrant Agents”, University College London, Department of Computer Science, 1999.

[12] Y. Demazeau, “La plate-forme PACO et ses applications”, In Proceedings 2ième Journée du GDR-PRC IA, Montpellier 1993.

[13] J.W. Perram and Y. Demazeau, “A Multi-Agent Architecture for Distributed Constrained Optimization and Control”, In Proceedings of SCAI’97, Scandinavian Conference on Artificial Intelligence, G. Grahme, Ed., IOS Press, pp. 162-175, 1997.

[14] C. Baeijs, “Fonctionnalité émergente dans une société d'agents autonomes: étude des aspects organisationnels dans les systèmes multi-agents réactifs”, PhD Thesis, Institut National Polytechnique de Grenoble, 1998.

[15] M. Occello and Y. Demazeau, “Modelling decision making systems using agents satisfying real time constraints”, In Proceedings 3rd IFAC Symposium on Intelligent Autonomous Vehicles, Madrid, Spain, march 1998.

[16] D. Moffat and N.H. Frijda, “Where there's Will there's an agent”, In M. Woolridge and N. Jennings,

editors, Proceedings of ECAI-94 ATAL Workshop on Agent Theories, Architectures, and Languages, volume LNAI 890, pages 245-260, Amsterdam, The Netherlands, Springer-Verlag, August 1995.
[17] R.C. Arkin and D. MacKenzie, “AuRA: Principles and practice in review”, Journal of Experimental and Theoretical Artificial Intelligence, 9(2), 1997.
[18] J.S. Sichman, R. Conte, and Y. Demazeau, “A social reasoning mechanism based on dependence networks”, In Proceedings of ECAI'94 - European Conference on Artificial Intelligence, Amsterdam, The Netherlands, August 1994.
[19] D.D. Corkill, “Advanced Architectures: Concurrency and Parallelism”, In V. Jagannathan, R. Dodhiawala, and L.S. Baum, editors, Blackboard Architectures and Applications, chapter II, pp. 77-83, Academic Press, 1989.
[20] M. Occello, “Distributed and parallel blackboards: application to dynamic systems control in robotics and computer music”, PhD Thesis, University of Nice – Sophia Antipolis, 1993 (In French).
[21] R.G. Smith, “The contract net protocol: High-level communication and control in a distributed problem solver”, IEEE Transactions on Computers, C-29 (12): 1104—1113, 1980.
[22] J. Ferber and O. Gutknecht, A meta-model for the analysis and design of organizations in multi-agent systems, Proceedings of 3rd International Conference on Multi-Agent Systems- ICMAS 98, pp. 128-135, Paris, France, july, 1998.
[23] M. D'inverno, D. Kinny, M. Luck and M. Wooldridge, “A formal specification of dMars”, In M. Singh, A. Rao and Jennings N., editors, Proceedings of ATAL Workshop on Agent Theories, Architectures, and Languages, volume LNCS/LNAI 1365, pages 155-176, Providence, USA, August 1997. Springer-Verlag.
[24] Z. Guessoum and J.P. Briot, “From active objects to autonomous agents”, IEEE concurrency, 7(3):68-76, 1999.
[25] L. Gasser, C. Braganza and N. Herman. “MACE: a flexible testbed for DAI”, Distributed Artificial Intelligence, Pitman, 1987.
[26] E. Kendall and M. Malkoun, “Design patterns for the development of multi-agent systems”, 2nd australian workshop on DAI, Cairns, Australia, LNCS/LNAI 1286, Springer-Verlag, 1996.