A Control Structure for Agent Interaction

Luís M. Botelho

Department of Information Sciences and Technologies of ISCTE
Group of Intelligent Systems and Telecommunications of ADETTI
Lisbon, Portugal
Phone: 351-21-7903906 **Fax:** 351-21-7903099

Luis.Botelho@iscte.pt

Abstract

This paper describes the interaction control structure of the agents of a traffic-monitoring multi-agent system. The goals of the agent are acquired by three mechanisms: agent innate goals (pre programmed in the agent); the reception of requests in inter-agent communication; and sub-goaling. In contrast with the mainstream view, goals are conditional structures represented by condition-action pairs. We show that condition-action pairs are suitable for representing persistent conditioned goals, non-persistent conditioned goals, persistent non-conditioned goals, and non-persistent non-conditioned goals.

Knowledge is represented in ACL/SL, the same language used for inter-agent communication. This option eases the process by which request messages generate conditioned goals in the procedural memory of the receiving agent.

The conditions of the goals stored in the agent procedural memory are evaluated and the goals whose precondition is satisfied are scheduled for satisfaction. Scheduled goals become the intentions of the agent.

Agents are implemented as C programs, but goals are explicitly represented in procedural memory. The evaluation of the conditions of the goals relies on procedural attachment. Each predicate, function and action is attached to specific handlers, which form the interface between the knowledge level of the agent and its internal procedures and data structures.

Keywords: Traffic Monitoring and Control, System Architectures

1. Introduction

This paper describes the internal architecture of the agents that belong to a multi-agent system for video-based traffic monitoring called Monitorix [1][8]. Monitorix has been developed by the Modest Project, which is a European project of the ACTS Programme.

Proceedings of the 2000 Intelligent Vehicles Conference, the Ritz-Carlton Hotel, Dearborn, MI, USA, October 4-5 2000

Monitorix is a multi-agent multi-camera system for video-based traffic monitoring. The system comprises several cameras placed along a highway. Each camera is coupled to a set of algorithms for image segmentation and indexing, called the Video Kernel (VK). The VK algorithms send image descriptions to a Proxy agent that delivers them by a set of agents that analyse them and produce higher level interpretations and decisions. Besides the Proxy, the multi-agent system associated to each camera comprises the LocalSite, the Classifier, the Behaviour, the Tracker, the Statistics and the UserAgent.

The LocalSite maintains mostly static information about road-configuration surrounding the site of the camera. The Classifier produces classifications of the vehicles observed in its camera. The Behaviour determines the typical trajectories of each class of vehicles and provides a quantitative description of the behaviour of each vehicle observed in its camera. The Tracker identifies vehicle descriptions in one camera with vehicle descriptions in the next camera. The Statistics computes the frequencies of vehicles of each class and computes a pollution index in its camera-site. Finally the UserAgent accepts information requests from the user and consults the other agents to obtain the desired information.

Aside from the application agents, the system also has some other agents that provide application independent services. The DF (directory facilitator) provides a yellow pages service. The AMS (Agent Management System) provides a white pages service.

Agents communicate with each other using FIPA ACL communication language and FIPA SL content language. ACL is a language for agent communication based on the speech act theory [7]. SL extends first order logic with action operators, the usual set of modal operators for beliefs, goals and intentions, an uncertainty operator, and a referential operator [2].

Agent interaction follows a protocol adapted from [6] called the information-subscription protocol [1].

In terms of internal agent architecture, we have taken a hybrid systems approach that integrates a knowledge based approach with an algorithmic approach. Each agent is composed of two main layers plus an interface layer between the other two. The internal layer (agent kernel) consists of a set of efficient specialised procedures that manipulate specialised data structures. The agent kernel is responsible for the agent tasks (e.g., determining typical trajectories). The external or social layer governs the interaction of the agent with other agents. It interprets and processes received messages and controls the agent's behaviour in terms of its interaction goals. The social layer of the agent is a particular implementation of a BDI-like architecture [3][5] using a production system. The middle layer is an interface between the social layer (knowledge layer) of the agent and the agent kernel. This interface relies on a technique called procedural attachment.

Section 2 describes the control apparatus responsible for the interaction of the agent with other agents. Section 4 presents our view of goals as motivational conditional structures. We show how several kinds of goals may be represented by conditional expressions. In section 4, we show how ACL/SL is used as a knowledge representation language for representing the agent's conditioned goals by means of action-condition pairs. Section 4 also shows what goals are generated when the agent receives a message of the request family. Section 5 describes an implementation architecture that supports the three layers of the agent described above. Finally, section 6 presents some conclusions, emphasises the main contributions of the paper and presents some future developments.

2. Interaction control

At each point in time, agent interaction is controlled by its private innate goals plus the goals acquired as a result of previous interactions with other agents. In the framework described in this paper, both of these are conditioned goals represented by production rules in the agent procedural memory. Rules R1 to R3 represent examples of conditioned goals of the Tracker agent in the Monitorix multi-agent system.

R1. If <Agent> provides a yellow-pages service and I've not registered myself with <Agent>, then register myself with <Agent>

R2. If <Agent> provides a vehicle-classification service and I've not subscribed the classes of the vehicles with <Agent>r, then subscribe the classes of newly observed vehicles with <Agent>

R3. If I have identified vehicle V in camera number 2 with vehicle U in camera number 1 and I have not sent this identification to the Classifier yet, then tell the Classifier V and U are the same vehicle

The first two goals (R1 and R2) are innate goals of the agent. The third goal (R3) resulted of a request sent by the Classifier agent asking the Tracker to identify vehicles observed in camera 1 with vehicles observed in camera 2. Goals may also appear as a result of the means-ends reasoning of the agent applied to its previous goals and beliefs.

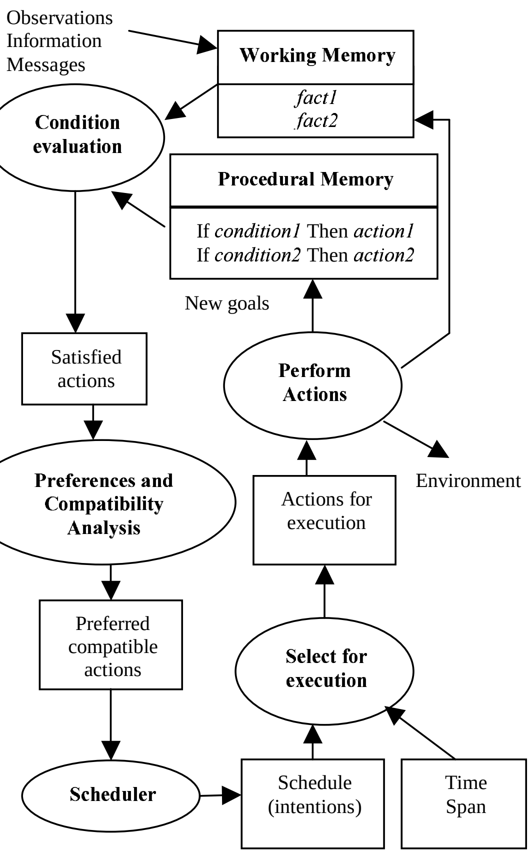

Figure 1 - Agent-interaction control

Since the goals of the agent are represented in its procedural memory as production rules, the behaviour of the agent is naturally determined by its procedural memory. The agent control loop repeatedly checks and processes new messages possibly updating its procedural memory. Information messages update the contents of the agent's working memory. Accepted requests are stored as new conditioned goals (i.e., production rules) in the agent's procedural memory.

After having processed new messages, the agent evaluates the conditions of all conditioned goals stored in its procedural memory and selects those actions whose conditions are satisfied. Then, it chooses a subset of non-conflicting actions from the set of actions with satisfied conditions and schedules the chosen set of actions for execution. The actions scheduled for execution become the intentions of the agent.

In the last step of the agent control loop, it executes all actions scheduled in a given time-span (Figure 1).

The described control process assumes the goals of the agent can be represented as condition-action pairs. The next section analyses the representation of several kinds of goals as condition-action pairs.

3. Representing conditioned goals

Goals may be classified according to their persistence and according to whether or not they are conditioned by any event or state.

Non-persistent goals disappear after being satisfied, whereas persistent goals don't. Certain kinds of requests generate non-persistent goals. For instance, when an agent receives a query from another agent (sender), it generates the non-persistent goal of sending the desired reply to the sender. The goal is non-persistent because, after the reply has made is way through, the goal of sending the reply disappears.

As an example of a persistent goal, consider the goal generated when the agent receives a request of sending an alarm message whenever it detects an accident. If an accident is detected, the agent must send the alarm message, but the goal does not disappear. Another alarm message will be sent, shell another accident occur.

Conditioned goals are goals dependent on some condition. *request-whenever* messages generate conditioned goals. Non persistent goals don't depend on anything. A query generates a non-conditioned goal.

In spite of this apparent diversity, all these kinds of goals may be represented by means of condition-action pairs. As can be seen in Figure 2, a non-conditioned goal may be represented by a condition-action pair in which the condition is true; whereas a non-persistent goal may be represented by a condition-action pair in which the last action in the action part destroys the goal.

Goal	Representation
Persistent goal of doing action *A* whenever condition *C* is true.	(A C)
Persistent goal of doing action *A*.	(A **true**)
Non-persistent goal of doing action *A* if condition *C* is true.	((**sequence** *A* **rmGoal**) *C*)
Non-persistent goal of doing action *A*.	((**sequence** *A* **rmGoal**) **true**)

Figure 2 - Representation of conditioned goals

For instance, if the agent has a non-persistent goal of performing a certain action A in any conditions, then the condition-action pair that represents the goal has a condition equal to **true** and an action equal to the sequence of A followed by the special action **rmGoal**. When the agent evaluates the conditions of the rules in its procedural memory, the action component of this goal is selected (since its condition is **true**). When the action part of the goal is performed, action A gets executed and then **rmGoal** is also executed removing the goal from the procedural memory. Therefore, the goal will not be considered again.

4. Knowledge representation language

The agents of the Monitorix system communicate with each other in ACL (FIPA Agent Communication Language [4]) with SL contents (FIPA Semantic Language [4]). SL is an extension of the language of the first order predicate logic with the usual modal operators of the BDI-like logic [3][5].

Since some of the goals of the agent are generated as a result of its interaction with other agents, we have decided to use ACL/SL both for inter-agent communication and for knowledge representation.

Sections 2 and explain how the goals of the agent may be represented as condition-action rules. In this section we show how an action condition pair may be represented in ACL/SL. We also show that persistent conditioned goals, non-persistent conditioned goals, persistent non-conditioned goals, and non-persistent non-conditioned goals may be generated when the agent receives messages of the *request* family.

One kind of SL expression is an action condition pair. We use this expression to represent the production rules that capture the goals of the agent. In a SL action-condition pair, the action may be any agent specific action, any communicative act (i.e., sending a message), or a compound action. Compound actions may be sequences of actions or action alternatives. The condition part of an action-condition pair may be arbitrarily complex propositions of the SL language.

We start with an example. Let's see how the conditioned goal R1 (section 2) is represented in ACL/SL. *"If <Agent> provides a yellow-pages service and I've not registered myself with <Agent>, then register myself with <Agent>"*.

The action part of the rule must be changed a little so that, after the Tracker registers itself with the agent that provides the yellow-pages service, it must learn that it is registered already. Therefore, the action part of the rule must be a sequence of two actions. The first action is the communicative act through which the Tracker asks the yellow pages agent to register its services. This communicative act is a request sent to the yellow pages agent asking it to register the Tracker. The second action asserts a fact in the agent working memory saying that it has been registered (we assume, for simplification, the communicative act is a successful act).

The condition part of the rule is a conjunction with two atomic propositions: the first of these states the name of the agent that provides a yellow pages service; the second atomic proposition states the Tracker agent is already registered with the yellow pages agent.

```
// Action sequence
((squence
  (request
    :sender Tracker identifier
    :receiver <agent>
    :content (action <agent>
      (register (df-agent-description
        :name Tracker identifier
        :protocols (set
          fipa-request
          information-subscription)
        :ontology (set
          traffic-surveillance)
        :language (set sl)
        :services (set
          (service-description
            :name vid1
            :type vehicle-identification
            :ontology traffic-surveillance
          (service-description
            :name vp1
            :type vehicle-prediction
            :ontology traffic-surveillance))
        :ownership (set adetti-iscte))))
    :language sl0
    :ontology fipa-management
    :protocol fipa-request)
  (assert
    (registered Tracker identifier)))

 (and // Condition
  (service-type <agent> yellow-pages)
  (not
    (registered Tracker identifier))))
```

Figure 3 – A conditioned goal in ACL/SL

In the conditioned goal represented in Figure 3, we use a special device that is not part of the original syntax of the ACL/SL language: a name between angle brackets, e.g., <agent>. This is used as a variable to be instantiated by pattern matching.

The request family of ACL messages supports all kinds of goals discussed in section **3**. In the remaining of this section, we describe the power of this message family to create several kinds of goals in the receiver (see Figure 4).

When an agent receives *request-whenever* or *subscribe* messages it creates persistent goals. *request-whenever* gives rise to conditioned goals, whereas *subscribe* generates non-conditioned goals. *subscribe* messages create the persistent goal of sending an inform message with the objects that satisfy a given condition (represented by *Proposition*).

request-whenever and *request-when* messages generate conditioned goals but whilst *request-whenever* generates a persistent conditioned goal, *request-when* generates a non-persistent conditioned goal.

request, *query-ref* and *query-if* messages all create non-persistent, non-conditioned goals. The goals created by *query-ref* and *query-if* messages are a special kind of action: sending *inform* messages.

Message	request-whenever (A C)
Goal	(A C)
Message	request-when (A C)
Goal	((sequence A rmGoal) C
Message	subscribe E
Goal	((inform-ref E) true)
Message	request A
Goal	((sequence A rmGoal) true)
Message	query-ref E
Goal	((sequence (inform-ref E) rmGoal) true)
Message	query-if P
Goal	((sequence (inform-if P) rmGoal) true)

Figure 4 – Goals created by ACL requests

It is worth comparing the table of Figure 4 with the table represented in Figure 2.

5. Internal agent architecture

Section 2 describes the interaction control structure of the agents of the Monitorix system. The main components of that control structure are the Working Memory, the Procedural Memory and the Action Schedule (see Figure 1). This section describes the interaction between the procedural memory and the working memory and describes the underlying implementation architecture. The Action Schedule and the scheduling algorithm lay beyond the scope of this paper.

In the first step of the agent control loop, (part of) the messages received by the agent are processed. In the scope of the present paper, the most important message families are information messages and requests. Requests give rise to conditioned goals in the agent procedural memory. Information messages update the contents of the agent's Working Memory. Hence, we need a way to assert new facts to Working Memory.

The second step of the agent control loop evaluates the conditions of all conditioned goals stored in the agent procedural memory. The conditions of the production rules are compared with the contents of the agent's Working Memory to check whether or not they are satisfied. Therefore, we need a way to query the contents of working memory.

The remaining of this section explains how new facts are asserted to working memory (as a result of incoming information messages) and how the contents of working memory are queried during the process of evaluating the conditions of the rules stored in procedural memory.

Conceptually, working memory is a repository of facts (represented as atomic formulas) but there is really no explicit representation of predicates, functions and actions. Internally, an agent is a C program that manipulates data structures that may bear no resemblance whatsoever with predicates, functions and actions. Therefore, we used procedural attachment to carry out the evaluation of the conditions of rules in procedural memory. Each predicate can be queried or asserted, therefore each predicate name is attached with two predicate handlers: one for consulting (evaluating) the predicate and another one to assert new facts with the same predicate[1].

The arguments of an atomic formula may be functional expressions therefore we also need function handlers to be used to evaluate functional expressions.

Similarly, the actions specified in the action component of conditioned goals are not explicitly defined in the agent knowledge base. Actions too, are performed through the execution of associated action handlers.

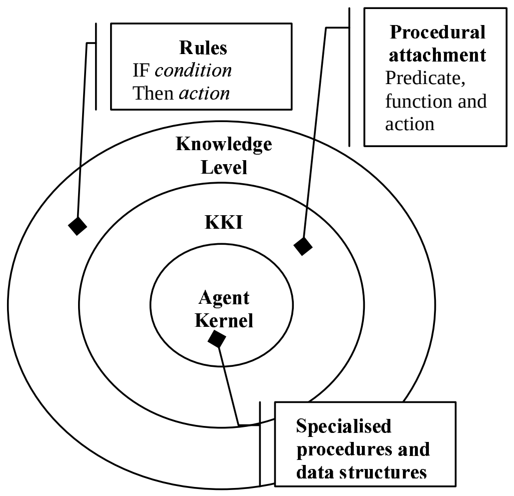

Figure 5 – An agent from outside in

As Figure 5 shows, an agent is seen from the outside as a knowledge-based system whose interaction is controlled by a set of production rules whose conditions are matched against the contents of the agent's working memory. However, from the inside, an agent is a regular program whose main task is carried out by a special purpose procedure. Between the social layer (knowledge level) of the agent and its internal routines and data structures, there is an interface (KKI, "Knowledge to kernel interface") that implements the communication between them using procedural attachment.

KKI is composed of three tables that represent the predicate handlers, the function handlers and the action handlers respectively (Figure 6).

Predicate assertion handler
int assert_pred(AgentData *ad, List args)
Predicate evaluation handler
int eval_pred(AgentData *ad, ArgsTuple *args, InstSet *iset)
Function handler
int eval_fun(AgentData *ad, ArgsTuple *args, Term **result)
Action handler
int perform(AgentData *ad, ActionParameters *params)

Figure 6 – handler prototypes

All predicate evaluation handlers have the same set of parameters and return the same return status information. The arguments of a predicate evaluation handler are the agent internal data structures, the list of the arguments of the predicate as specified at the knowledge level, and the set of alternative variable instantiations. A variable instantiation is a set of variable/value pairs. If all variables passed as arguments of a predicate take the values specified in the variable instantiation, the proposition becomes true.

Predicate assertion handlers all have the same arguments and return the same constants in similar circumstances. The arguments of a predicate assertion handler are the agent internal data structures, and the list of the arguments specified for the predicate in the particular formula considered.

The arguments of function handlers are the agent data structures, the set of arguments of the considered functional expression and a result parameter to receive the result of the evaluation.

Finally, the arguments of action handlers are a set of attribute value pairs. Each attribute represents the name of the argument and each value is the parameter itself. Figure 6 represents the prototypes of the predicate, function and action handlers.

6. Conclusions

This paper discusses some of the approaches followed in the development of a video-based traffic surveillance multi-agent system. In spite of being a particular application domain, some general lessons were learnt about agent architecture and knowledge representation.

The procedural attachment approach allowed us to build an efficient agent using specialised procedures and data structures while preserving the flexibility and the possibility to explain the behaviour of the system provided by the knowledge

[1] Actually, each predicate is attached to four predicate handlers. One is used to assert a new fact with that predicate. The second is used to create an uncertain instance of the predicate. The third is used to evaluate atomic formulas with that predicate. Finally, the fourth is used to evaluate uncertain propositions with the predicate. We won't talk about uncertainty here to avoid missing the main point.

based approach used at the social level. From the outside, the agent interaction is governed by a rule base and a working memory. From the inside, the agent tasks are performed by efficient specialised procedures and data structures. The procedural attachment approach also allowed us to completely de-couple the design of the agent interaction from the design of the problem solver.

An important contribution of the project is the view of goals as motivational conditional structures. Agents don't have the same goals irrespective of their current contexts. The use of conditioned goals allowed us to build agents with context-dependent goals.

In spite of the different way we view goals, the general idea is still a BDI-like architecture since goals become intentions and intentions become actions.

We also found ACL/SL to be suitable both for inter-agent communication and for knowledge representation. Using the same language for communication and representation provides easy and natural ways to generate new goals from the interaction and also to interpret received messages in terms of the meaningful internal structures.

Finally, the *request* family of ACL messages (request, request-when, request-whenever, query-ref, query-if, and subscribe) is powerful enough to generate the whole variety of goal types in the receiving agent: information and actions goals, persistent and non-persistent goals, and conditioned and non-conditioned goals.

Two main developments will deserve our attention in the near future. One of these problems relates to the representation and reasoning about action dependencies. Not all actions with satisfied pre conditions may be scheduled for execution since the effects of executing one of them may impair the pre conditions of the other ones. Besides, the actions once scheduled may have to be reconsidered because, if they are not immediately executed the dynamics of the environment may render them inappropriate. Maybe one possible way to solve this problem is to reschedule all actions that were scheduled but not executed during a certain time interval.

The other development will be the development of a declarative memory for representing more general and complex knowledge than is currently possible through the procedural attachment approach taken so far.

Acknowledgements

This work was partially supported by the ACTS project AC 304 and by UNIDE/ISCTE.

7. References

[1] Botelho, L.M.; Lopes, R.; Sequeira, M.M.; Almeida, P.; and Martins, S. 1999. "Inter-agent communication in a FIPA compliant intelligent distributed dynamic-information system". In Callaos, N; Nada, N; Cherif, A; and Aveledo, M. (eds) *Proceedings of the 5th International Conference on Information Systems Analysis and Synthesis (ISAS99).* International Institute of Informatics and Systemics (IIIS).

[2] Bretier, P.; and Sadek, D. "A rational agent as the kernel of a cooperative spoken dialogue system: implementing a logical theory of interaction". In Müller, J.P; Wooldridge, M.J.; and Jennings, N.R. (eds) I*ntelligent Agents III - Proceedings of the Third ATAL Workshop*. Springer Verlag.

[3] Cohen, P.R.; and Levesque, H. 1990. Intention is choice with commitment. *Artificial Intelligence*. 42:213-261

[4] Foundation for Intelligent Physical Agents. 1998 FIPA 97 Specification, Version 2.0 Part 2 "Agent Communication Language", http://www.fipa.org/spec/fipa98.html

[5] Georgeff, M.P. and Rao, A.S. (1995) "The semantics of intention maintenance for rational agents", *IJCAI'95,* p704-710

[6] D'Inverno, M.;. Kinny, D.; and Luck, M. 1998. Interaction protocols in Agentis. In Proceedings of the ICMAS98, p112-119

[7] Searle, J.R. 1969. Speech Acts. Cambridge University Press, 1969

[8] Trigueiros, M.J.; Botelho, L.M.; Lopes, R.J.; Nunes, L.M.; Sequeira, M.M.; David, N.; Almeida, A.P.; Almeida, P.; Vieira, S.; Marques, G.; Martins, S.; Afonso, R.; Teles, R.; Figueiredo, P.; and Abreu, B. 2000. "Description and software implementation of the Camera Assistant". Deliverable of the Modest Project (in preparation)