Deliberative Server for Real-Time Agents

Carrascosa, C.; Rebollo, M.; Julián, V.; Botti, V.

Dept. Sistemas Informáticos y Computación
Universidad Politécnica de Valencia
Camino de Vera s/n
46022 - Valencia
Spain
Phone Number: 96 3877352
{carrasco,mrebollo,vinglada,vbotti}@dsic.upv.es

Abstract. Over the last few years more complex and flexible techniques have been needed to develop hard real-time systems. The agent paradigm seems to be an appropriate approach to be applied in this area. In this paper, an approach to incorporate agency characteristics in hard real-time software systems is presented. This is done through a hard, real-time, intelligent agent architecture, called ARTIS agent architecture. The ARTIS agent architecture incorporates a control module which manages the agent behavior and controls the execution of the different agent components. This paper is focused on an improvement of the control module.

Keywords: Intelligent Agents, control & real-time, meta-reasoning
Topics: AI in real-time
To be considered for Paper Track section

Deliberative Server for Real-Time Agents

Abstract. Over the last few years more complex and flexible techniques have been needed to develop hard real-time systems. The agent paradigm seems to be an appropriate approach to be applied in this area. In this paper, an approach to incorporate agency characteristics in hard real-time software systems is presented. This is done through a hard, real-time, intelligent agent architecture, called ARTIS agent architecture. The ARTIS agent architecture incorporates a control module which manages the agent behavior and controls the execution of the different agent components. This paper is focused on an improvement of the control module.

1 Introduction

Over the last few years the use of the agent paradigm has increased sharply as an important field of research within the Artificial Intelligence area. This paradigm has been applied to different fields, such as control processes, mobile robots, commercial applications, etc. Concurrently, a new discipline has emerged, called Real-Time Artificial Intelligence, which provides useful techniques for solving complex problems which require intelligence and real-time response times.

This paper takes the application of agents to hard real-time environments as a starting point. In order to situate the topics, different definitions are presented. First, a *Real-Time Environment* is defined. This is an environment with temporal restrictions that may have different features which affect its control. Two of the main features of a real-time environment are non-deterministic and dynamic. A Real-Time Environment is controlled by a *Real-Time System (RTS)*, which is defined as a system in which the correctness of the system depends not only on the logical result of computation, but also on the time at which the results are produced [16]. In a RTS, some tasks have deadlines. A *deadline* defines the greatest time interval in which the system can provide a response. If the response is obtained after this time, it will probably not be useful. Occasionally, some people confuse the term "real-time" with "on-line" or "fast". The main feature of a RTS is not to always be interconnected or to be the fastest system. A RTS should guarantee its temporal restrictions and, concurrently, it should try to accomplish its goals. Researchers differentiate between two types of RTS. First, a *Hard Real-Time System* is a RTS where the execution of a task after its deadline is completely useless. Systems of this kind are critical systems, and severe consequences will result if the timing responses are not satisfied. On the other hand, a *Soft Real-Time System* is characterized by the fact that the execution of a task after its deadline only decreases the quality of the task result [16]. Different techniques are needed for hard and soft RTS. In this paper, only hard real-time systems are considered.

Once a RTS is defined, it is possible to define a *Real-Time Artificial Intelligence System (RTAIS)* as a system that must accomplish complex and critical processes under a probably dynamic environment with temporal restrictions by using AI techniques. Previous approaches to RTAIS can be found in the literature. Anytime algorithms [4] and approximate processing [6] are the most promising. One line of research in RTAIS has been to build large applications or architectures that embody real-time concerns in many components [6], such as Guardian [8], Phoenix [9] and CIRCA [11]. All these applications are soft RTS because they don't assure the fulfillment of their temporal restrictions.

Within RTAIS research area, a *Real-Time Agent (RTA)* can be defined as an agent with temporal restrictions. In the same way as RTS, it is possible to talk about "hard" or "soft" RTA. Almost all the existing approaches, like PRS [10], the CELLO agent model [13] and, more recently, DECAF [7], are designed as soft RTA (without critical temporal restrictions).

According to what has been stated, a hypothetical agent designed for hard real-time environments must accomplish its goals, responsibilities and tasks with the added difficulty of hard temporal restrictions. One of the main problems is how to achieve *agency* taking into account the execution of critical tasks that the agent should guarantee.

In this paper, an improvement to the control module (the incorporation of a deliberative component) of an architecture for hard, real-time, intelligent agents is presented. This architecture was previously presented in [1].

2 Is a Hard, Real-Time, Intelligent Agent possible?

The basic architecture of a hard, real-time, intelligent agent should consist of three components: a set of sensors, a set of effectors, and a cognitive capability which can compute actions on the environment from sensor perceptions in a bounded time. More specifically, there must be a module that estimates the current state of the environment (perception), a module of cognition which is in charge of computing the set of actions allowing the agent to reach its goals, and a module of action which acts on the environment. However, it is necessary for all of these modules to have a bounded worst-case execution time (wcet), in order to determine whether the system reacts according to its temporal restrictions. It is usually easy to obtain the wcet of the methods employed in perception and action modules. There exist classical analysis techniques in RTS that provide a solution for this problem.

The main problem in this architecture is with the cognition module. This module uses AI techniques as problem-solving methods to compute more *intelligent* actions. In this case, it is difficult to extract the time required by this module because it can either be unbounded or if bounded, its variability is very high. When using AI methods, it is necessary to provide techniques that allow their response times to be bounded. These techniques are based on RTAIS techniques [6].

With regards to the concept of agent, an agent may have a set of features associated to it. These features add specific differences not available in more classic software systems. When researchers talk about concepts like autonomy, sociability, reactivity, proactivity, etc. they want to provide an agent its own identity. It is not the goal of this paper to determine whether or not the fulfillment of any of these features is obligatory. Nor does this paper attempt to determine the degree of fulfillment necessary which is still an open discussion. Nevertheless, it is true that adding some of this features to an agent will affect its architecture.

Some of the most important features of agency are the capacities: to work autonomously, to adapt to the environment, to reason, to learn, to predict the future effect of the performed actions and to predict the future behavior of the environment. It is obvious that, if a specific software achieves any or all of these features, it is due to an extra effort in its development process. Therefore, even minimal fulfillment significantly complicates the implementation and functionality of an agent. If the agent must operate in a hard real-time environment, the agent construction complexity is increased enormously. Evidently, different environments require different software structures. Therefore, in an agent context, it is necessary to define an appropriate structure in order to use agent features in hard real-time environments.

RTS tasks have temporal restrictions which must be previously guaranteed. This limitation in the system functionality mentioned above affects the features of an agent that tries to be modeled as a RTS. For example, the time unbounded problem-solving methods are a serious problem because their execution cannot be temporal restricted. The real existence of a hard, real-time, intelligent agent depends on the ability to overcome this problem and to be able to incorporate the typical features of agency while maintaining the real-time behavior of the system.

The hard real-time ARTIS agent architecture [1] incorporates all the necessary aspects that the agency features provide to a software system, but adapted to hard real-time environments. This architecture includes techniques of RTAIS, which overcomes the problem. This approach starts from the basic real-time architecture presented above, and it guarantees reacting on the environment in a dynamic and flexible way. The ARTIS agent architecture guarantees an agent response that satisfies all the critical temporal restrictions of the system and it tries to always obtain the best answer for the current environment status. This is due to its capacities for problem-solving, adaptability and proactivity which have been added to the architecture.

3 ARTIS Agent: A Hard, Real-Time, Intelligent Agent

This point presents an agent architecture, called ARTIS Agent (AA) architecture, for hard real-time environments. In accordance with existing agent architectures [18], the AA architecture could be labelled as a vertical-layered, hybrid architecture with added extensions to work in a hard real-time environment [1].

One of the main features of the AA architecture is its hard real-time behavior. It guarantees the execution of the entire system's specification by means of an off-line analysis of the specification. This analysis is based on well-known predictability analysis techniques in the RTS community [5].

The off-line analysis only ensures the schedulability of real-time tasks. However, it does not force the task sequence execution. The AA decides the next task to be executed at run-time, allowing it to adapt itself to environment changes, and to take advantage of the tasks using less time than their wcet.

The AA reasoning process can be divided into two stages. The first one is a mandatory time-bounded phase. It obtains an initial result of acceptable quality. After that, if there is available time left (also called *slack time*), the AA may use this time for the second reasoning stage. This is an optional stage and it does not guarantee a response. It usually produces a higher quality result through intelligent, utility-based, problem-solving methods.

3.1 ARTIS Agent Architecture

The architecture of an AA can be viewed from two different perspectives: the user model (high-level model) and the system model (low-level model) [1]. The user model offers the developer's view of the architecture, while the system model is the execution framework used to construct the final executable version of the agent.

From the **user model** point of view, the AA architecture is an extension of the blackboard model [12] which is adapted to work in hard real-time environments. It is formed from the following elements:

- A set of **sensors** and **effectors** to be able to interact with the environment. Perception and action processes are time-bounded.
- A set of **in-agents** that models the AA behavior. The main reason to split the whole problem-solving method into smaller entities is to provide an abstraction which organizes the problem-solving knowledge in a modular and gradual way.
  Each in-agent periodically performs a specific task. An in-agent is also an agent according to the Russell's agent definition [14]. Each in-agent has to solve a particular subproblem, but all the in-agents of a particular AA cooperate to control the entire problem, and an in-agent may use information provided by other in-agents.
  Depending on the temporal restrictions and the "intelligence" used in its problem-solving method, in-agents can be classified into critics and acritics. A **critic** in-agent is characterized by a period and a deadline. The available time for the in-agent to obtain a valid response is bounded. It must guarantee a basic response to the current environment situation. It is formed by two layers (see Figure 1): the reflex layer and the real-time deliberative layer. The first one assures a minimal quality response and the second one tries to improve this response. The reflex layer of all the in-agents make up the AA mandatory phase. On the other hand, the real-time deliberative layers form

the optional phase. An **acritic** in-agent only has the real-time deliberative layer.
- A set of **beliefs** comprising a world model (with all the domain knowledge which is relevant to the agent) and the internal state. This set is stored in a frame-based blackboard.
- A **control module** that is responsible for the real-time execution of the in-agents that belong to the AA. The temporal requirements of the two in-agent layers (reflex and deliberative) are different. Thus, the control module must employ different execution criteria for each one.

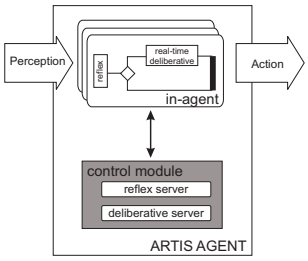

Fig. 1. ARTIS Agent architecture

The **system model** provides a software architecture for the AA. The main features of this model are [5]:

- Off-line schedulability analysis.
- Task Model that guarantees the critical temporal restrictions of the environment.
- Slack extraction method to on-line calculate the available time for executing the real-time deliberative layer.
- Set of extensions to the Real-Time Operating System incorporating features for managing real-time capabilities.

There is a toolkit, called InSiDE [15], which automatically converts the user model to the system model [5]. The result is an executable AA.

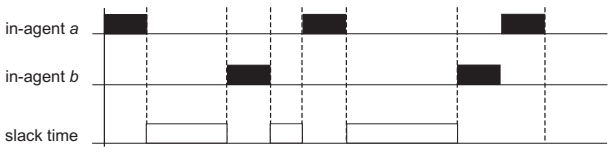

Fig. 2. AA execution timing diagram

Figure 2 shows a possible execution stage of an AA by a timing diagram. In this example, the AA is comprised by two in-agents (a and b). Black boxes

represent the processor time intervals assigned to the in-agent reflex layer execution. Between these executions there exists available time (white boxes). This time can be used by the AA in order to improve its responses. One of the tasks of the control module is the management of this slack time. It will allocate this time among the real-time deliberative layers of the in-agents.

The integration of intelligence in an AA lies in the effective management of the slack time by the control module. So, the rest of the paper explains this module in detail.

3.2 Control Module: intelligence for RTA

The control module of an AA is the component in charge of controlling how and when the different components of the AA are executed. According to the classical structure of a traditional blackboard system, it corresponds to the control module of systems of this kind.

The utility of the AA control module is to guarantee that the agent always reacts to the environment and that this response is the best possible result according to the slack time and the knowledge that the agent has available. Therefore, an AA is an agent with a reactive level which is always guaranteed. This level provides a first time-bounded response, but this response probably is not the best one. Moreover, the AA has a second level, with a deliberative behavior. The main goal of this second level is to provide a better response than the reactive one. The execution of this last level is conditioned by the available time that the agent has to run this level. Another condition for executing the second level is that the agent considers it appropriate depending on the real situation.

Therefore, the control module of an AA is divided in two submodules (that communicate through events): the reflex server and the deliberative server.

- **Reflex server (RS)** This module is in charge of controlling the execution of reactive components, that is, the components with critical temporal restrictions. Due to these restrictions, it is integrated within a Real-Time Operating System (RTOS)[1] [17]. It includes the First Level Scheduler (FLS) that must schedule the execution of all these components, in order to guarantee their temporal restrictions. This scheduler is implemented according to a common RTS scheduling policy, a Fixed-Priority, Pre-emptive Scheduling Policy.
  Once the execution of the critical parts is assured, there may be free intervals between the execution of these critical parts. These slack times (calculated using an algorithm based on the Dynamic Slack Stealing algorithm [3]) can be employed by the second submodule of the control module in order to do different functions, the goal of which is to refine the reactive response and to improve its quality.
  This module carries out the following functions to accomplish its purpose:
  - To schedule the execution of all in-agents with critical temporal restrictions. This process must guarantee the fulfillment of these restrictions.

[1] The current version of the AA architecture uses RT-Linux as its RTOS

- To cede the agent control to the DS during the system idle time.
- To inform the deliberative server of the execution state of the in-agent reflex part and the time it has available to use. This slack time is calculated just before informing the DS to take into account the tasks using less time than their wcet.

– **Deliberative server (DS)** This module is in charge of controlling the execution of the deliberative components. Therefore, this server is the intelligent element of the control module, but with temporal restrictions.

4 Deliberative Server

As it has been declared in the previous point, this is the AA control module part in charge of managing the intelligence of this agent.

Its main purposes are: to improve the quality of the agent responses, to adapt the agent to important environment changes and to pursue its own objectives. In order to achieve its goals, some functionalities, that are described in section 4.2, have been implemented.

4.1 Internal Structure

It can be said that the DS is implemented following a control blackboard architecture [8]. This architecture is an event-driven module. More specifically, it implements a variation of the so-called "satisfacing cycle" [8] as described in figure 3:

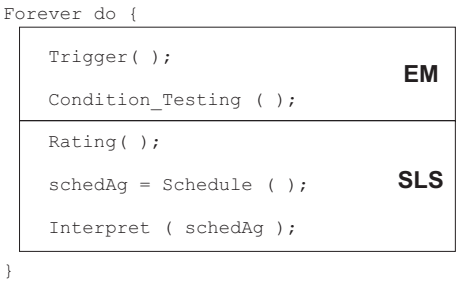

Fig. 3. Relation between control cycle and DS modules

The DS cycle has been adapted to be used with temporal restrictions and bounded execution time. This is due to the DS is executed in slack time (see Figure 2).

Moreover, the DS control cycle has been implemented through two modules, the Event Manager (EM) and the Second Level Scheduler (SLS). The first one receives significant events and reacts to them. It comprises the trigger and condition testing phases of the "satisfacing cycle". The second one is in charge

of scheduling the use of the slack time and comprises the rating, schedule and interpret phases.

In this way, when the DS begins to work, the Event Manager looks for new events and takes the appropriate measures to react to them (trigger phase). The events are sorted by their contribution to system quality to be able to respond to the most appropriate event at each moment.

Next, it checks what is available to be scheduled in its available time (condition testing phase). When the Event Manager ends, it is time for the SLS to work. First of all, it rates all the deliberative in-agent parts that it has available for scheduling (rating phase). After this, it makes a plan for the slack time that it has available (schedule phase). Lastly, it makes the execution of this plan possible (interpret phase).

At this point, an overview of the control schema can be made. It follows a schema with two schedulers (FLS and SLS) as can be seen in Figure 4a. The FLS schedules the reflex parts of the in-agents assuring the fulfillment of hard temporal restrictions. In the slack time, it cedes the control to the DS, where the SLS schedules the Real-Time Deliberative (RTD) parts of the in-agents.

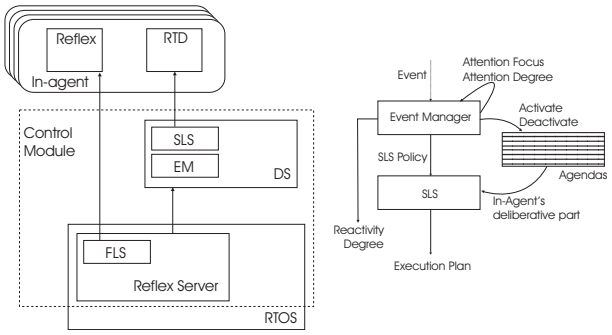

Fig. 4. a) Two Schedulers Control Schema (left). b) Deliberative Server Interactions (right)

The main purpose of the SLS is to decide what to execute to improve the agent's response. The decision may be to execute the optional part of a critic in-agent, or to execute an acritic in-agent previously activated by the event manager. To make this decision, the SLS has a set of scheduling policies with different features which are appropriated for different situations. The available scheduling policies [2] can be classified into two types: **Greedy Policies** (they rate all the items in the agenda and choose one for execution) and **Deliberative Policies** (they construct a plan with the sequence of items to execute). Both types of scheduling policies has bounded planning time, that is controlled by the RTOS.

The SLS fixes the maximum execution time available for each item it sends to execution. This time will be the minimum between the item's wcet and the

slack time still available to the SLS. If the item doesn't end its execution before this time, it is suspended and inserted back at the agenda so that it may be resumed in the future.

4.2 Describing EM Functionalities

The two deliberative server modules and their interactions can be seen in Figure 4b. The main functionalities which are implemented in the Event Manager to achieve the main goals of the DS are shown in this figure:

- To activate and to deactivate acritic in-agents: The EM may activate / deactivate deliberative processes that are independent of the in-agents whose reflex part has just been processed.
- To manage the Attention Degree: The DS must divide its execution time between its two modules. This time assignment does not have to be equitable. In fact, one of the ways to make the agent adaptive is to control the percentage of execution time provided to event management. This percentage is the Attention Degree and may be modified by the event manager. The time assigned to the event manager allows it to receive all the events or only some of them. This variation is gradual, when the extreme situations are the following: all the events are attended (the deliberative server is totally focused on the environment) and no event is attended (the deliberative server is totally focused on its own objectives).
- To manage the Attention Focus: The Event Manager may change the agent attention focus. That is, the belief subset which updates will direct the deliberative process. Changes in this subset will produce the events that the EM is interested in. This feature has two extreme situations: the EM is interested in the whole belief set, controlling all the modifications, and the EM is interested in variations of only one belief (any other belief data modification will be transparent for the DS).
- To change the SLS Policy: The EM will control the SLS policy to be used by this scheduler. It will make a change when the environment state and / or schedule quality make it necessary. For instance, if the SLS policy is a deliberative one, and there is very little execution time available, then it seems appropriate to adopt a greedy SLS policy.
- To manage the Reactivity Degree: The Reactivity Degree is the percentage of slack time dedicated to cognition processes. This parameter allows the EM to control the slack time that it can use for its execution.
  The usual working of an AA delays the execution of its actions as late as possible. This behavior leaves the maximum time for the cognition process. Nevertheless, sometimes it might be interesting to execute an action even if there is slack time left. This feature has two extreme situations: all the slack time may be used to deliberate (this is the default mode and the agent has the lowest Reactivity Degree), and there is no cognition process (the agent behaves as a reflex agent and it has the highest Reactivity Degree).

4.3 DS Goal Achievement

Through the use of this set of functionalities, the DS is responsible for the fulfillment of almost all the AA agency features, as can be seen in the following points.

- Improvement of the agent answer quality: The Reflex Server is in charge of obtaining a low-quality first response from the AA (which is implemented as the mandatory time-bounded phase). The DS is responsible for using the available time to improve the quality of this response. This process is carried out by controlling the AA reasoning capability at two levels:
  - It decides at each moment what subject to reason about in the slack time. Thus, it decides what to run from the active items stored in the agenda (real-time deliberative layers of the critic in-agents and acritic in-agents). These items are what really reason about the problem.
  - Meta-reasoning: it decides when to change the SLS scheduling algorithm, the reasoning process focus or the Attention Degree. This improves the reasoning process and, thus, the agent response quality obtained through this process.
- Adapting to important environment changes: The adaptability of the agent is expressed in its ability to change dynamically. Actually, any functionality change that the Event Manager triggers as a reaction to an event helps to adapt the AA to new situations. For instance, if the agent is very focused on the changes in the environment (the Attention Degree is very high) and the event rate is also very high, a greedy scheduling policy should work better than a deliberative one. If the SLS is working with a deliberative policy, the EM would change it.
- Pursuing its own objectives: The Attention Focus allows the AA to focus its reasoning process on different belief sets. Therefore, in some situations, the AA is able to focus its deliberation on its own initiatives and to "avoid" the environment changes. This effect can be augmented by tuning the attention degree to dedicate all the available time to the SLS.

5 Coclusions

The work presented in this paper shows how the AA architecture provides the execution of intelligent components incorporating a control module. It is thanks to this control module that agents developed with this architecture will add agency features.

This architecture has been successfully applied to mobile robot control [15], by means of a prototype running over RT-Linux. Further investigation is centered on social aspects of the AA architecture, by extending the architecture in order to develop real-time, multi-agent systems. This new feature will allow the AA to offer services and to make them available to other agents through the control module. New examples are considered in order to study this new behavior. More specifically, a train traffic control system is being developed, where each train is modeled as an AA.

References

1. V. Botti, C. Carrascosa, V. Julian, and J. Soler. Modelling agents in hard real-time environments. *Proc. of the MAAMAW'99. LNCS, vol. 1647*, pages 63–76, 1999.
2. V. Botti and L. Hernandez. Control in real-time multiagent systems. In *Proceedings of the Second Iberoamerican Workshop on DAI and MAS*, pages 137–148, Toledo, Spain, 1998. Garijo, F., Lemaitre, C. (Eds.).
3. R.I. Davis, K.W. Tindell, and A. Burns. Scheduling slack time in fixed priority preemptive systems. In *Proc. R-T Systems Symposium, North Carolina*, pages 222–231. IEEE Comp. Society Press, 1993.
4. T. Dean and M. Boddy. An analysis of time-dependent planning. *Proceedings of the seventh National Conference on Artificial Intelligence. St. Paul, Minessota,*, pages 49–54, 1988.
5. A. Garcia-Fornes, A. Terrasa, V. Botti, and A. Crespo. Analyzing the schedulability of hard real-time artificial intelligence systems. *EAAI*, pages 369–377, 1997.
6. A. Garvey and V. Lesser. A survey of research in deliberative real-time artificial intelligence. *The Journal of Real-Time Systems*, 6:317–347, 1994.
7. John R. Graham. *Real-Time Scheduling in Distributed Multi-agent Systems.* PhD thesis, Dept. of computer and Information Science. University of Delaware, 2001.
8. B. Hayes-Roth, R. Washington, D. Ash, A. Collinot, A. Vina, and A. Seiver. Guardian: A prototype intensive-care monitoring agent. *AI in Medicine*, 4:165–185, 1992.
9. A. E. Howe, D. M. Hart, , and P. R. Cohen. Addressing real-time constraints in the design of autonomous agents. *The Journal of Real-Time Systems*, 2:81–97, 1990.
10. F. Ingrand, M. P. Georgeff, and A. Rao. An architecture for real-time reasoning and system control. *IEEE Expert*, pages 34–44, December 1992.
11. D. Musliner, E. Durfee, and K. Shin. Circa: a cooperative intelligent real-time control architecture. *IEEE Transactions on Systems, Man and Cybernetics*, 23(6), 1993.
12. P. Nii. Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architectures. *The AI Magazine*, pages 38–53, Summer 1986.
13. M. Occello and Y. Demazeau. Modelling decision making systems using agents satisfying real time constraints. *IFAC Proceedings of 3rd IFAC Symposium on Intelligent Autonomous Vehicles*, 1:51–56, 1998.
14. S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach.* Prentice Hall International Editions, 1995.
15. J. Soler, V. Julian, C. Carrascosa, and V. Botti. Applying the artis agent architecture to mobile robot control. In *Proceedings of IBERAMIA'2000. Atibaia, Sao Paulo, Brasil*, volume I, pages 359– 368. Springer Verlag, 2000.
16. J.A. Stankovic. Misconceptions about real-time computing. *IEEE Computer*, 12(10):10–19, 1988.
17. A. Terrasa, A. Garca-Fornes, and V. Botti. Flexible real-time linux. *Real-Time Systems Journal*, 2:149–170, 2002.
18. M. Wooldridge and N.R. Jennings. Intelligent agents: Theory and practice. *The Knowledge Engineering Review*, 10(2):115–152, 1995.