Physical Agents

Vicente Julian, Carlos Carrascosa, Juan Manuel Corchado, Javier Bajo, Esteve Acebo, Bianca Innocenti and Vicente Botti

Abstract. The aim of this section is to provide a current view of different approaches which try to develop new models, architectures or real applications for physical agents.

1. Introduction

Nowadays, many of the developments in the research field are obtained through simulation, but once they are fully tested in simulation, there may exist problems to take them to the real world. It is due, in part, to the peculiarities belonging to the real world environment, that may not be simulated. So, when talking about agents, it may not be the same to work with a simulated robot, than to work with a real robot.

Moreover, only through working in a real physical environment may be carried out one of the main purposes of any research, the transference of technology.

In the rest of this section, some comments about agents working in real physical environments are written, along with some experiences in test examples or even transference of technology carried out by AgentCities.es nodes.

1.1. What are physical agents?

Physical agents can be defined as agents situated in a physical real world. Some examples of this kind of agents are, for instance, robots as the *aibot* ones in the *Robocup* competition.

From a very abstract perspective, the basic architecture of a physical agent should consist of three components: a set of sensors, a set of effectors, and a cognitive capability which can compute actions on the physical environment from sensor perceptions probably 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. This

basic architecture proposed for a physical agent is shown in Figure 1. However, it is necessary for all of these modules to have a bounded worst case execution time, in order to determine whether the system reacts according to its temporal restrictions.

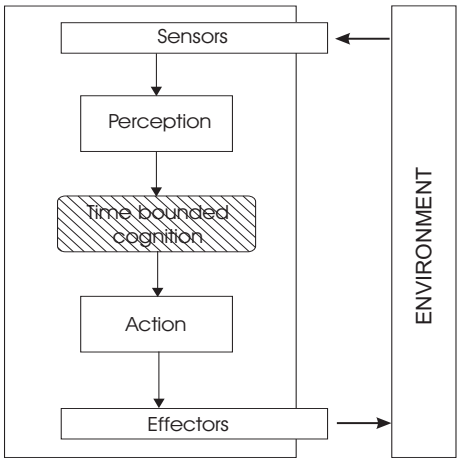

FIGURE 1. General structure for a Physical Intelligent Agent

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 [19].

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 with its own identity. Some of the most important features of agency are the following 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 physical 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 physical environments.

1.2. Main Problems

The term *physical* applied in the agent or multi-agent area is still in a premature state of development which is why we don't have clear and consolidated definitions. Nevertheless, we can perform a characterization of some requirements of this kind of systems. The requirements that should be met are:

Access to continuous data. A physical agent must access to a physical real environment where the information is produced in a continuous way, not in a discrete one.

Time representation in the communication process. To be able to reason over the temporal instant at which an event is produced in a physical environment, it is necessary to integrate the time concept inside the information transmitted among the different entities that make up a distributed system. The agent communication language FIPA ACL was not developed with these features in mind.

Global time management. In a physical distributed system it is absolutely necessary to have a common global time for all the elements that make up the system. There are several strategies that manage and synchronize the clocks of each computer. Different levels of precision are obtained with each of them. The Network Time Protocol (NTP) [31] for Internet, or Cristian's algorithm for intranet [14] are some of the most important algorithms. Platform Agents must provide this global time service, so that the agents can be synchronized. Platform agents should also be adapted to have a global time for the agents of diverse platforms that want to interact.

Real-time Communication. In physical distributed systems, it is necessary be able to assure the communication. This supposes communication protocols with a low and restricted latency[1], as well as fault detection. One example is the CAN protocol [40]. The applications where it is feasible to use the MAS paradigm does not need such strict restrictions for communication. However, there is no doubt about the need for efficient protocols to assure a maximal delivery message time to the developed applications.

Hard resource management. The execution of tasks in this kind of systems is assured, by exercising a strict control on the available resources. There is a planning algorithm that assures the task execution and the coherent use of resources. The agent platforms must implement hard resource management. This is very important, for instance, when facilitating agent mobility among platforms.

Fault tolerant execution. This kind of systems are considered to be predictable. However, it is also indispensable for them to develop fault tolerant systems. This is fundamental since the systems that are controlled are usually critical and a system fall would be catastrophic. A relevant work in this area is [13]. The fault tolerance in MAS must be twofold. First, the execution of the agents must be assured after an internal failure, as well as after a failure in the communication

[1]Latency: interval between the sending of a message and its reception

process. Platform agents must be capable of assuring certain requirements in order to offer communication mechanisms and strategies that permit agent execution recovery.

1.3. Related Work

Over the last years, a number of researchers have used agent technology in attempts to resolve the above presented problems, whereas only a few testbeds and real applications have been developed and reported.

Some members of the Agentcities.es have developed applications of this kind, as will be explained in the next sections. These applications may be classified into two big groups:

- Application of generic architectures to specific problems, as the classical subsumption architecture [7] which is the most widely known robot architecture, the ARCHON platform created for industrial multi-agent systems [23], or the applications of ARTIS agent architecture to mail robots [37, 11], or the application of CIRCA agent architecture to Unmanned Aerial Vehicles (UAVs) under development by the military and deep space probes being developed by NASA [12].
- Ad-hoc developments to solve problems regarding robotics, industrial processes or any physical environment related. In these cases, the developed solutions bear in mind the concrete problems and their extrapolation is very difficult to other problems or domains. With respect to robotics we can find a lot of works or proposals developed to control a robot, and more specifically a mobile robot. It is difficult to select a little set of works. Maybe we can highlight the work of Mackenzie [30] where an agent-based method for designing controllers is presented, or the work of Van Breemen [41] which describes a method for modeling complex control problems. On the other hand, in industrial applications, it can be emphasized the control of a line of production developed by the industrial consortium of the DaimlerChrysler [8] where it was provided a flexible and robust system for the control of a production line in the factory. The electricity transport management developed by the Spanish company Iberdrola [24] where there was realized a process for monitoring and controlling the generation, transport and distribution of the electrical energy. Other interesting works can be the control of the air supply in painting cabins of the General Motors [16] or the air traffic control by [27]. At Spanish level we can emphasize the control of a container Terminal [33] where a global management of a terminal is realized including the control of physical devices as the cranes or transtainers. In next sections this latter work will be explained in a more detailed way.

2. Deliberating while the physical world changes

One of the main problems that needs to be overcame when applying generic agent architectures to physical environments is the efficient integration of high-level, multi-agent planning processes within this kind of architectures. These complex deliberative processes, which allow the agent to adapt and learn, are unbounded and it is difficult to integrate them in physical and restricted systems. Typically, in the multi-agent area these processes are carried out by so-called deliberative agents, which decide what to do and how to do it according to their mental attitudes. In a deliberative agent, it is relatively simple to identify decision processes and how to perform them. However, its main drawback lies in finding a mechanism that permits its efficient and bounded execution. Therefore, it would be interesting to integrate complex deliberative processes for decision-making in physical systems in a simple and efficient way.

Intelligent agents may use a lot of reasoning mechanisms. One of them is based on planning techniques [1]. Planning-based agents decide the course of an action before it is realized. Thus, a plan represents the structure of such action. A planning-based agent will execute plans allowing it to reach its goals. To do this, the agent goes from an initial state to try to get to a final state or set of states. The mechanism used to reach the goals is to apply a set of operators over the objects composing the agent's environment.

2.1. A bounded deliberative technique

In this section, the objective of integration of new bounded deliberative techniques into an agent architecture is pursued. More specifically, this technique is applied to the ARTIS agent architecture [6]. This kind of agent will be able to incorporate a new planning proposal known as CBP-BDI (Case Base Planning - Beliefs Desires Intentions) in order to carry out deliberative planning tasks at the moments where the timing restrictions will not be considered critical.

This proposal has been applied to the specific problem of a mail robot whose work is to collect and to deliver mail to people working at a company department. The robot working is developed in a restricted and well-known test environment. Moreover, the example has been tested in order to proof the proposal. This case study consists in solving the automatization of the internal mail management of a department that is physically distributed in a single floor of a building plant. At the department, there is a mail robot in charge of attending sending requests, carried out by a user from a department office through a PDA to send a letter or packet to other office of such department. In this way, the robot will be in charge of picking up and delivering the external mail received by the department or the mail that is going to be sent to the outside.

The robot is going to be controlled by an ARTIS agent. Each ARTIS agent has a reflex server able to plan tasks at real-time and a second level deliberative server in charge of non-critical timing restrictions. The deliberative server will plan the execution of CBR-BDI (Case Base Reasoning - Beliefs Desires Intentions)

techniques [28], and it will be in charge of generating optimal plans to pick up and deliver mail at slack time (the spare time once critical time restrictions are satisfied).

A CBR-BDI agent uses as reasoning mechanism the case-based reasoning allowing it to learn from an initial knowledge; to autonomously interact with the environment, the users and the rest of the system's agents; and to have a great capability of adapting to the environment's needs. So, case-based reasoning is a suitable technique to implement a planner for the mail robot problem. The CBP-BDI agent generates plans, where a plan is a sequence of pick up and delivery points. In the same way, the CBP-BDI agent will be available to replan in situations where the robot is unable to fulfil the assigned plans, such as finding obstacles, closed doors, low battery level, or receiving new requests of picking up or sending mail while the robot is executing a plan.

The study case has been implemented in order to evaluate the proposal. To do this, different experiments have been carried out investigating, basically, the performance of the system and the planning/replanning behaviour. The results have shown the benefits obtained with the integration of the CBP-BDI deliberative behaviour into the ARTIS agent while maintaining the fulfilment of the critical time restrictions. A detailed version of this proposal can be found in [9].

2.1.1. ARTIS Agent: A Hard, Real-Time, Intelligent Agent. This point provides a short description of the ARTIS Agent (AA) architecture, for hard real-time environments (a more detailed description can be found in [6] [37]). The AA architecture could be labelled as a vertical-layered, hybrid architecture with added extensions to work in a hard real-time environment [6].

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, and it is defined in [18].

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 satisfactory quality. After that, if there is available time left (also called *slack time* in the RTS literature), 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. This split reasoning process is described in detail in [6].

ARTIS Agent Architecture The architecture of an AA can be viewed from two different perspectives: the user model (high-level model) [10] and the system model (low-level model) [39]. The user model offers the developer's view of the

architecture, while the system model is the execution framework used to construct the final version of the agent.

From the **user model** point of view, the AA architecture is an extension of the blackboard model 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. Due to the environment features, the perception and action processes are time-bounded.
- A set of **beliefs** comprising a world model (with all the domain knowledge which is relevant to the agent) and the internal state, that is the mental states of the agent. This set is stored in a frame-based blackboard [4].
- A set of **behaviors** that models the answer of the AA to different situations. It can be said that an *state* (internal along with an environment representation) defines a *situation* (represented by the current beliefs and goals) which actives a behavior or allows it to go on being active. This behavior determines the agent current set of *goals and restrictions*, along with the *knowledge* needed to control the situation.

  Each one of these behaviors are formed by a set of **in-agents**. The main reason to split the whole problem-solving method is to provide an abstraction which organizes the problem-solving knowledge in a modular and gradual way. (see figure 2).

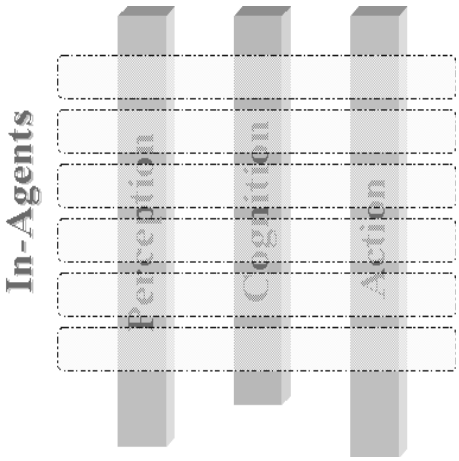

FIGURE 2. Modular division of an AA into in-agents

Each in-agent periodically performs an specific task. An in-agent is also an agent according to the Russell's agent definition [35]. 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.

In-agents can be classified into critics and acritics. The first ones are in charge of solving essential problems of the AA, so its execution is assured at least for calculating a low-quality answer. The last ones are in charge of solving non-essential problems of the AA to improve its performance quality. 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. From a functional point of view, an in-agent consists of two layers: the reflex layer and the real-time deliberative layer. The reflex layer assures a minimal quality response (an off-line schedulability analysis of the AA, considering all the in-agents in the AA, guarantees that this reflex layer will be fully executed). On the other hand, the real-time deliberative layer tries to improve this response (this level will be executed in slack time). 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 **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.
  - **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 part of a Real-Time Operating System (RTOS)[2] [39]. It includes the First Level Scheduler (FLS) that must schedule the execution of all the reactive components, in order to guarantee their temporal restrictions. This scheduler is implemented according to a common RTS scheduling policy, a Fixed-Priority, Preemptive Scheduling Policy [3].
    Once the execution of the critical parts is assured, there are slack time intervals between the execution of these critical parts. These slack times (calculated using an algorithm based on the Dynamic Slack Stealing algorithm [15]) 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.
    * 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

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

FIGURE 3. ARTIS Agent architecture

The **system model** provides a software architecture for the AA that supports all the high level features expressed in the user model. The main features of this model are [18]:

- 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.

2.1.2. Integration of a CBP-based planner in ARTIS. This section shows how has been integrated a new bounded CBP-based planner techniques inside of the ARTIS agent architecture. This planner allows a more efficient execution time management, according to the agent's goals. It has to be taken into account that this planner activates tasks to fulfil agent's goals that will be deal by the real-time task schedulers in order to be executed guaranteeing the real-time constraints. CBR-based planner (or CBP) has been included as a sporadic in-agent that will be activated when a new plan needs to be generated for a new goal. Moreover the in-agent will be also activated when replanning because the environment evolution

makes it impossible to finish the current plan. The in-agent's initial part reads the planning or replanning event that activated it. According to this event, it checks if the existing current plan is still feasible. If such plan is not yet applicable, it builds a new plan or modifies the existing one. In an optional way, it tries to improve the new plan. Lastly, the action part of this in-agent begins the plan.

The CBR-based planner provides planning based on previous experiences. CBR systems use memories (past experiences) to solve new problems. The main concept when working with CBR systems is the concept of case. A case is a past experience that can be represented as a 3-tuple $< P, S(P), R >$. In this way a case is composed of a problem description (initial state), the solution applied to solve the problem (in CBP the solution is a plan or a set of plans, in other words, the sequences of actions executed in order to achieve the objectives) and the result obtained after applying the solution (the final state an the evaluation of the plan executed). The planner needs to maintain a case memory that will be used to solve new problems. When a new problem is presented the planner executes a CBR cycle to solve it. The CBR cycle is composed of four sequential stages: Retrieve, where those cases with the most similar problem description to the current problem are recovered from the cases memory; Reuse, in which the plans (solutions) corresponding to the similar cases retrieved in the previous stage are reused to construct a new plan; Revise, where the proposed plan is evaluated; and Retain, where the planner learns from the new experience. One of the key points in the CBR-based planning is the notation used to represent the solution (the plans). A solution can be seen as a sequence of intermediate states transited to go from an initial state to the final state. States are usually represented as propositional logic sets. The set of actions can be represented as a set of operators together with an order relationship. Furthermore Carbonell (Carbonell 1986) indicates that additional information is needed on the decisions taken during the plan execution.

A deliberative CBP-BDI agent is specialized in generating plans and incorporates a case-based planning (CBP) reasoning mechanism. The purpose of a CBR agents is to solve new problems by adapting solutions that have been used to solve similar problems in the past [**?**], and the CBP agents are a variation of the CBR agents, based on the plans generated from each case. An innovative technique that allows replanning in execution time has been incorporated in order to construct an efficient planner. Next, the CBP planner is presented. Let $E = \{e_0, ..., e_n\}$ the set of the possible tasks that have to be completed.

$$a_j : \underset{e_i}{E} \rightarrow \underset{a_j(e_i)=e_j}{E} \tag{1}$$

An Agent plan is the name given to a sequence of actions (1) that, from a current state e_0, defines the path of states through which the agent passes in order to offer the better path according to the initial problem description. Below, in (2), the dynamic relationship between the behaviour of the agent and the changes in the environment is modelled. The behaviour of agent A can be represented by its

action function $a_A(t)\ \forall t$, defined as a correspondence between one moment in time t and the action selected by the agent,

$$Agent\ A = \{a_A(t)\}_{t \in T \subseteq N} \tag{2}$$

From the definition of the action function $a_A(t)$ a new relationship that collects the idea of an agent's action plan (3) can be defined,

$$p_A(t_n) = \int_{t_0}^{t_n} a_A(t)dt \tag{3}$$

The variation of the agent plan $p_A(t)$ will be provoked essentially by: the changes that occur in the environment and that force the initial plan to be modified, and the knowledge from the success and failure of the plans that were used in the past, and which are favoured or punished via learning. O indicates the objectives of the agent and O' are the results achieved by the plan. R represents the total resources and R' are the resources consumed by the agent. The efficiency of the plan (4) is the relationship between the objectives attained and the resources consumed

$$E_{ff} = \frac{\#(O' \bigcap O)}{\#R'} \tag{4}$$

Where # means cardinal of a set. The objective is to introduce an architecture for a planning agent that behaves – and selects its actions – by considering the possibility that the changes in the environment block the plans in progress. This agent is called MRPI (most re-plan-able Intention agent) because it continually searches for the plan that can most easily be re-planned in the event of interruption. Given an initial point *e0*, the term planning problem is used to describe the search for a way of reaching a final point $e_i = e* \in E$ that meets a series of requirements. Given a problem E and a plan *p(t)* the functions *Ob* and *Rc* accumulated are constructed from the objectives and costs of the plan (5). For all time points *ti* two variables are associated:

$$Ob(t_i) = \int_a^{t_i} O(t)dt\ \ Rc(t_i) = \int_a^{t_i} R(t)dt \tag{5}$$

This allows us to construct a space representing the environment for planning problems as a vectorial hyper dimensional space where each axis represents the accumulative variable associated with each objective and resource. In the planning space, defined in this way, conform to the following properties:

1. Property 1: The representations of the plans within the planning space are always monotonously growing functions. Given that *Ob(t)* and *Rc(t)* are functions defined as positive, function *p(t)* expressed at these coordinates is constant or growing.

2. Property 2: In the planning space, the straight lines represent plans of constant efficiency. If the representations of the plans are straight lines, the slope of the function is constant, and coincides with the definition of the efficiency of the plan. $\frac{d}{dt}p(t) = cte \Leftrightarrow \lim_{\Delta \to 0} \frac{\Delta O(t)}{\Delta R(t)} = cte$

In an n-dimensional space, the extension of the straight concept line is called a geodesic curve. In this sense, the notion of geodesic plans can be introduced, defined as those that maintain efficiency at a constant throughout their development. This way, only the plans of constant efficiency (geodesic plans) are considered, due to the fact that they are the ones of minimum risk. In an environment that changes unpredictably, to consider any plan that is different from the geodesic plan means to accept a certain risk. The agent must search for the plan that determines a solution with a series of restrictions *F(O;R)=0*. In the plans base the plans sought are those that are initially compatible with the problem faced by the agent, with the requirements imposed on the solution according to the desires, and in the current state [**?**]. If all the possible plans $\{p_1, ..., p_n\}$ are represented within the planning space, a subset of states that the agent has already attained in the past will be obtained in order to resolve similar problems. With the mesh of points obtained (generally irregular) within the planning space and using interpolation techniques, we can obtain the working hyperplan *h(x)* (that encapsulates the information on the set of restrictions from restored experiences, by definition leading to a hyperplan since it verifies $h(x_j) = p_j j = 1, \ldots, n$ and the planning space is the dimension n). From this, geodesic plans can be calculated and and the variation calculation is applied. Suppose, for simplicity's sake, a planning space of dimension 3 with coordinates $\{$ *O,R1,R2* $\}$. Between point *e0* and objective points $f_s f = \{e1, ..., em\}$ and over the interpolation surface *h(x)*, the Euler Theorem [29] [25] guarantees that the expression of the geodesic plans will be obtained by resolving the system of equations in (6), where R_i is the function accumulated R, O is the function of accumulated O and L is the distance function on the hyperplan *h(x)*, $L = \int_h dl$..

In order to obtain all the geodesic plans that, on the surface *h(x)* and beginning at *e0*, allows us to reach any of the points $e* \in (f_s f)$, a condition of the surrounding must be imposed: the initial point will be $e_0 = (O_0, R_0)$. Once an efficient plan is developed, the plan around it (along its trajectory) are used to create a denser distribution of geodesic plans. The tool that allows us to determine this is called the minimum Jacobi field associated with the solution set [26]. $g_0 : [0, 1] \to S$ be a geodesic over a surface S. Let $h : [0, 1]x[-\varepsilon, \varepsilon] \to S$ be a variation of *g0* so that for each $t \in (-\varepsilon, \varepsilon)$, the set $\{ht(s)\}t \in (-\varepsilon, \varepsilon)$: $ht(s)$ for all $t \in (-\varepsilon, \varepsilon)$ are geodesic in S and they begin at $g_0(0)$, in other words, they conform to $h_t(0) = g_0(0)$ for all$t \in (-\varepsilon, \varepsilon)$. In these conditions, taking the variations to a differential limit (7).

$$\begin{cases} \frac{\partial L}{\partial R_1} - \frac{d}{dO}\frac{\partial L}{\partial R_1'} = 0 \\ \frac{\partial L}{\partial R_2} - \frac{d}{dO}\frac{\partial L}{\partial R_2'} = 0 \end{cases} . \quad (6)$$

$$\lim_{t\to 0}\{h_t(s) = g_0(s+t)\} = \lim_{t\to 0}\{h(s,t)\} = \quad (7)$$
$$\frac{\partial g_0}{\partial t}\Big|_{(s,0)} = \frac{dg_0}{ds} \equiv J_{g_0}(s)$$

The term $Jg_0(s)$ is given to the Jacobi Field of the geodesic g_0 for the set $\{gn(x)\}n \in N$, and in the same way that the definition has been constructed, it is possible to give a measurement for the distribution of the other geodesics of $\{gn(x)\}n \in N$ around g_0 throughout the trajectory. Given a set of geodesics, some of them are always g* that, in their environment, have a greater distribution than other geodesics in a neighbouring environment. This is equivalent to saying that it presents a variation in the distribution of geodesics lower than the others and therefore the Jacobi Field associated with *{ gn(x) }* $n^a N$ reaches its lowest value at J_g*. Let's return to the MRPI agent problem that, following the recuperation and variation calculation phase, contains a set of geodesic plans *{ p1,...,pn }* . If the p^* is selected with a minimum Jacobi Field value, it can be guaranteed that in the event of interruption it will have around it a greater number of geodesic plans in order to continue. This suggests that given a problem with certain restrictions *F(O;R)=0*, the geodesic plan p^* with minimum associated Jacobi field associated with the set $\{gn(x)\}n \in N$ is called the most re-plan-able solution. The behaviour model G for the MRPI agent is (8).

$$G(e_0, p_1, \cdots, p_n) = p^* \Leftrightarrow \exists n \in N / J_{g_n} \equiv J_{g*} = \quad (8)$$
$$\underset{n\in N}{Min}\, J_{g_n}$$

If the plan p^* is not interrupted, the agent will reach a desired state $ej = e* \in fsf, j \in \{1, \ldots, m\}$. In the learning phase, a weighting *wf(p)* is stored. With the updating of weighting *wf(p*)*, the planning cycle of the CBP motor is completed. In Figure 4, it is possible to see what happens if p^* is interrupted. Let's suppose that the agent has initiated a plan p^* but at a moment $t > t0$, the plan is interrupted due to a change in the environment. The geodesic planning meets the conditions of the Bellman Principle of Optimality [5], in other words, each one of the plan's parts is partially geodesic between the selected points. This guarantees that if *g0* is geodesic for interrupted *e0* in *t1*, because *e0* changes to *e1*, and *g1* is geodesic to *e1* that is begun in the state where *g0* has been interrupted, it follows that: *g = g0+ g1* is geodesic to *e= e0 (t1 - t0)+e1 (t2 – t1)*

The dynamic process follows the CBP cycle recurrently: each time a plan finds itself interrupted, it generates from the state reached so far, the surroundings of the plans from the case base and adjusts them to the new problem. With this it calculates the geodesic plans and selects the one which meets the minimum conditions of the associated Jacobi field. A minimum global Jacobi field J(t) also meets Bellman's conditions of optimality [5], in other words, a minimum global

FIGURE 4. Model for behaviour G(t)

If on the one hand, successive Jacobi fields generate one Jacobi field, and on the other hand, minimum Jacobi fields generate a minimum Jacobi field, the MRPI agent that follows a strategy of replanning *G(t)* as indicated to survive a dynamic environment, generates a global plan *p*(t)* that, faced with all possible global plans $\{pn(t)\}n \in N$, presents a minimum value in its Jacobi field $J_g * (t) = J_p * (t)$. An agent has been formally defined that in a dynamic environment seeks plans that lend it greater capacity for replanning.

3. Applications

In this section will be presented examples of current or past experiences of the Spanish research groups. Specifically, three real applications are shown. The next section presents two examples made by the Universitat de Girona. The last example is an industrial application and it has been developed by the Universidad Politécnica de Valencia.

3.1. The Grill robot. A multi-agent control architecture

One of the current challenges of robotics is to make completely autonomous robots capable of modifying their performance in complex and changing environments. So, distributed control systems should be used to develop the robot control architecture, in order to provide mechanisms to distribute, coordinate, adapt and extend the control system of the robots. On the other hand, robots require high-level cognitive capacities, and multi-agent architectures provide the appropriate way to define them. Merging both research lines, distributed control and multi-agent systems, a multi-agent architecture to control a single robot, an ActivMedia

FIGURE 5. MAS Architecture.

Pioneer 2DX mobile robot has been developed by the EXIT Research Group at the Universitat de Girona [20].

In order to implement the multi-agent control architecture, an ad-hoc multi-agent platform has been built to deal with real-time issues. This platform follows the FIPA (the Foundation for Intelligent Physical Agents) standards [17] but avoid some bottleneck on communication, a key issue in robotics when a real time response is expected.

In Fig. 5 the agents of the MAS architecture is shown, as well as some platform specific agents and the information flow among them. Agents can be grouped in perception, behavioral, actuator and deliberative agents. Solid lines among agents means that there are no restriction in message passing while dotted lines indicate that only one agent at a time can send the message (after coordination). White solid lines denote conflicting agents.

Perception agents obtain information about the environment and about the internal conditions of the robot, as they collect data from the sensors and transform it to provide the suitable information to other agents. There are as many perception agents as there are sensors or group of them in the robot. Particularly for the robot perception agents are the following:

The encoder agent is in charge of obtaining the position and heading of the robot with reference to a fixed frame.

- **The sonar agent** which collects all the sonar readings and creates a local map of obstacles.
- **The battery sensor agent** which monitors the battery charging in order to prevent that it could be permanently damaged.

Behavioral agents carry out specific actions, such as avoiding obstacles. There are as many agents as necessary to describe the behavior of the robot. Based on the information received from perception agents, they react to changes in the environment and in the robot itself. Particularly, behavioral agents are the following:

- **The goto agent** in charge of driving the robot to the target position, based on the information provided by the encoder agent.
- **The avoid agent** that must go around obstacles found in the path of the robot.
- **The battery charger agent** that asks for replanning when the battery is going under a threshold in order to guide the robot to the recharging area.
- **The gothrough agent** is in charge of going through narrow places like doors, based on the information received from the sonar and the localization agents.

Deliberative agents implement high-level complex tasks as for example, planning. These agents are the following:

- **The localization agent** that must localize the robot in the global map.
- **The path planning agent** that calculates the trajectory to the goal, free of non-moving obstacles.
- **The task planning agent** that plans the sequence of tasks to perform in order to reach the goal.
- **The interface agent** which interacts with the user.

Actuator agents are in charge of controlling the linear and angular speed of the robot interacting directly with motors. There is an actuator agent per each possible actuator. Particularly only one agent is needed because of the limitations of the Pioneer 2DX operating system. This agent is:

- **The robot agent** which communicates, each 100 ms, with the robot microcontroller and gets the actual position and sonar readings and sends the desired linear and angular speeds to the onboard controllers. The role of the actuator agent has been reduced to a merely interface between the robot and the whole architecture, due to the robot constraints mentioned above.

Platform Agents: They implement the basic services that have to be in the platform in order to guarantee the correct functioning of the community of agents. Particularly, there is only one agent that provides several basic services. This agent is:

- **The directory facilitator agent (DFA):** that knows which agents are active in the community, their location in the net, the services they provide and the resources they need. It also informs the agents when a new one joins the community, the resources it use and the services it provides.

As can be seen in Fig. 5, there are several agents trying to use the same resource at a given time, so some coordination is necessary. For exemple, conflicts

can arise among the avoid, the goto and the gothrough agents when trying to send conflicting actions to the robot agent and between the battery charger and the task planning agents when demanding a trajectory to the path planning agent.

One solution to this problem is to define a central coordinator agent which, having knowledge of the agents in conflict imposes one decision. However, we believe that such centralized coordination mechanism can be a bottleneck when dealing with architectures with a lot of agents. Conversely, we think that conflicts are local and a distributed coordination approach can be more appropriated. Particularly, a peer-to-peer coordination mechanism among the agents involved in one conflict is proposed. Coordination process is carried out locally based on utilities values computed by the agents in conflict.

All the agents in the architecture have their own utility function, known only by themselves, but all normalized between [0,1] (they are comparable). Agents that can have conflicting decisions, exchange their utility value. In case of conflict, the agent who has a higher value of utility wins the decision. This agent is getting the control regarding to the conflicting activity. For example, suppose that the goto agent has a utility value of 0.5, the gothrough agent of 0.3 and the avoid agent of 0.7; being 0.7 the higher value, the avoid agent takes the control of the situation, and it is the only one that sends messages to the robot agent (see [21, 22]).

In order to reduce communication among agents, the last agent who has had the control, broadcasts it utility value. If there is no response, meaning that it has the higher value, the agent uses the resource. On the other hand, if there is an agent with a higher utility value, then it informs all the agents with its utility value, indicating that the agent is going to use the resource. In this way, communication process is reduced and centralization of coordination is avoided.

3.2. The $ICTINEU^{AUV}$ submarine robot

From 1990, the Association for Unmanned Vehicle System International (AUVSI) has promoted the design and development skills of Autonomous Underwater Vehicles (AUV) by means of an annual competition. Inspired by this competition, the Defence Science and Technology Lab (DSTL), the Heriot Watt University and the National Oceanographic Centre of Southampton organized the first Student Autonomous Underwater Challenge Europe (SAUC-E) [36]. In January 2006, a team of students collaborating with the Underwater Robotics Lab of the University of Girona decided to form the VICOROB-UdG Team [38] to face the challenge by designing its own submarine robot, $ICTINEU^{AUV}$.

3.2.1. Design. The SAUC-E [36] mission takes place in a small volume in which a high maneuverability is required. In this situation a hover-type vehicle propelled and steered by thrusters is the most desirable configuration. A classical open frame design, together with a modular design of the components conveniently housed in pressure vessels, has been considered [34] the simplest and most reliable approach for the physical design of the $ICTINEU^{AUV}$ robot. The robot is propelled by four thrusters that made it a fully actuated vehicle in four degrees of freedom:

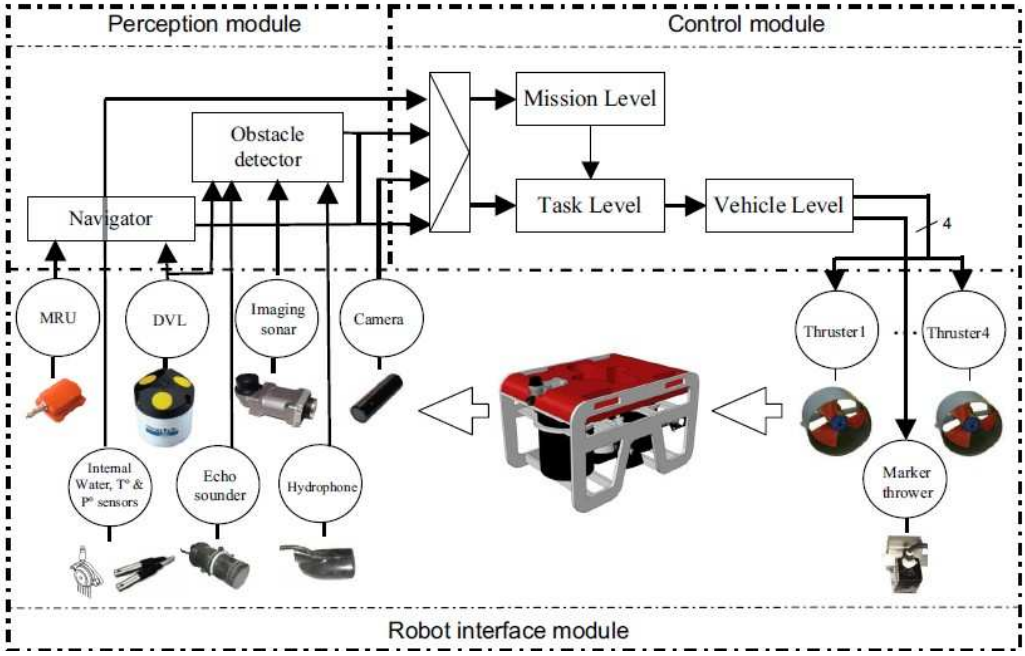

FIGURE 6. $ICTINEU^{AUV}$ software architecture.

surge, sway, heave and yaw, while being passively stable in roll and pitch as its meta-center is above the center of gravity. The robot chassis is made of Delrin material. Three pressure vessels are used for holding the electronics. One of them houses the computers, another the thruster controllers and the batteries, and the last encapsulates the Motion Reference Unit (MRU).

The robot uses two PCs, one for control and one for image and sonar processing. It is also equipped with a complete sensor suite composed of a forward-looking color camera, a downward looking b&w camera, an imaging sonar, an echo sounder, a transducer for acoustic device detection and an Argonaut Doppler Velocity Log which also includes a compass/tilt sensor.

The software architecture of the robot can be seen in figure 6, there are three main modules: robot interface module, perception module and control module. Following the principles of hybrid control architectures, the control module is organized in three layers: vehicle level, task level and mission level. The vehicle level controls the speed of the robot, the task level is a conventional behavioral layer [2] including a library of behaviors that can run alone or in parallel. During the execution of a mission, more than one behavior can be enabled simultaneously; hence, a coordinator module is used to fuse all the responses corresponding to the enabled behaviors into a single response to be sent to the velocity controller (vehicle level). Finally, the upper layer (mission level) is responsible for the sequencing

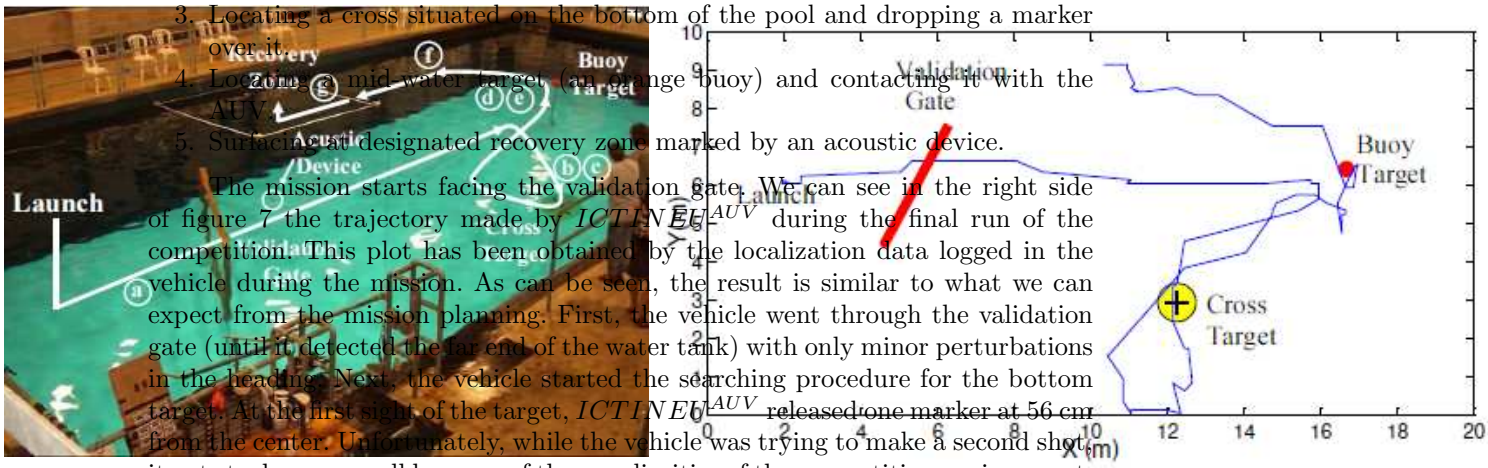

FIGURE 7. LEFT: The mission for the final run. RIGHT: Plot of $ICTINEU^{AUV}$'s trajectory

of the mission tasks, selecting for each mission phase the set of behaviors that must be enabled as well as their parameters.

3.2.2. The SAUC-E Competition. The SAUC-E competition takes place in a water tank environment of 20 meters by 10 meters and a depth of 6 meters. The mission consists of (see the left side of figure 7 for a graphical representation):

1. Moving from a launch/release point and submerging.
2. Passing through a 3x4 meter validation gate.
3. Locating a cross situated on the bottom of the pool and dropping a marker over it.
4. Locating a mid water target (an orange buoy) and contacting it with the AUV.
5. Surfacing at designated recovery zone marked by an acoustic device.

The mission starts facing the validation gate. We can see in the right side of figure 7 the trajectory made by $ICTINEU^{AUV}$ during the final run of the competition. This plot has been obtained by the localization data logged in the vehicle during the mission. As can be seen, the result is similar to what we can expect from the mission planning. First, the vehicle went through the validation gate (until it detected the far end of the water tank) with only minor perturbations in the heading. Next, the vehicle started the searching procedure for the bottom target. At the first sight of the target, $ICTINEU^{AUV}$ released one marker at 56 cm from the center. Unfortunately, while the vehicle was trying to make a second shot, it got stuck near a wall because of the peculiarities of the competition environment. The zone boundary between the black walls and the white bottom of the tank caused the vision algorithm to get confused. After the timeout expired, the vehicle proceeded with the mission going to the next waypoint. When $ICTINEU^{AUV}$ found the buoy, it was too close. This made it harder to aim the target. As a result, the vehicle missed the target by a few millimeters. Finally, the vehicle moved to the recovery zone to end the mission. $ICTINEU^{AUV}$ probed its capability to undertake a preprogrammed mission. It did two tasks and almost completed the other two, being the only entry of the competition able to link all the tasks. This performance gave the final victory to the VICOROB-UdG team.

3.3. An industrial application: A Port Container Terminal

A multi-agent system for solving the automatic allocation problem in a container terminal is presented in this section. This proposal has been developed by the Universidad Politécnica de Valencia. This section only present a brief explanation of the proposal. A more detailed explanation can be found at [33] and [32]. The operations carried out in this terminal are included in the most complex tasks of the transport industry. This is due to:

- The great diversity of entities acting in the container import and export processes.
- Interaction with a dynamic environment.
- The distributed nature of the problem which is formed by a set of independent systems, but whose individual decisions directly affect the performance of the others.

The traditional centralised and sequential applications for Container Terminal Management are being found to be insufficiently flexible to respond to changing management styles and highly dynamic variations in loading/unloading requirements. With the traditional centralised approaches to management and control, the entire terminal is generally controlled by central software, which limits the expandability and reconfiguration capabilities of the systems. Using hierarchical organization forces the grouping of resources into permanent, tightly coupled subgroups, where information is processed sequentially by a centralised software supervisor. This may result in much of the system being shut down by a single point of failure, as well as plan fragility and increased response overheads. The multi-agent system model seems to be an adequate framework to overcome such problems and for dealing with the design and development of an application which is flexible, adaptable to the environment, versatile and robust enough for the efficient management of a container terminal. It is very important for the turn-around time of a cargo ship which is in port container terminals to be as short as possible. An average cargo liner spends 60% of its time in port and has a cost on the order of U.S. $1000 for each hour it spends in port. The whole container allocation process must be directed towards minimise the containership stowage time. This is the main objective of the optimisation of the global performance allocation process.

3.3.1. Problem Description. The traditional solutions to Container terminal management are addressed by means of a modular decomposition of the problem into several sub-problems each one represents a specialized aspect of it. The set of operations to be conducted in the terminal is very extensive, but the existing approaches share some common systems:

- Marine Side Interface. This system focuses on loading/unloading containers to/from ships. Normally two or three gantry cranes (GC) are used to move containers for each ship.
- Transfer System. It transfers containers from/to the apron to/from the container storage yard. The method used in the terminal is to employ yard trucks

FIGURE 9. Transtainer view

3.3.2. System Architecture. Figure 10 shows the system architecture, the agents are mainly characterised by their independence from the rest of the system elements. They are able to coordinate and to communicate some decisions to the rest of the system. The communication between agents is done by means of asynchronous messages, which are based upon the FIPA-ACL standard. The proposed distributed approach enhances flexibility, efficiency and robustness. Five agent classes can be found in this system:

- The Ship agents: they control the ships load and unload sequence scheduling process

FIGURE 10. System Architecture

- The Stevedore agents: they manage the loading and unloading of all the ships docking in the port.
- The Service agents: they distribute the containers in the port terminal.
- The Transtainer agents: they optimise the use of these machines.
- The Gate agents: they interact with the land transport (I/O of containers by land).

3.3.3. Agent Description. This section presents the above commented agents which forms the multi-agent approach:

The Ship Agent: In response to the arrival of a ship the system will create a new Ship agent instance for this ship and its load profile. Its goals are: to minimize the gantry crane idle time, to maximize its utilization, to minimize the ships load/unload time, and to minimize the derived costs from the stowage process. This work is closely related to the Stevedore agents involved, which the Ship agent will have to co-ordinate with. The different Ship agents active at any given moment must co-ordinate with each other as a whole to minimise the possible blockages between the assigned cranes. The goal of this minimisation is to maximise the active time of all the cranes and to reduce the load/unload time of each ship.

The Stevedor Agent: When a gantry crane is active loading or unloading containers from a specific ship, the Stevedore agent will try to obtain the most appropiate scheduling to manage the container stowage in the ships load/unload

sequences. To develop these goals, the agent is co-ordinated with the rest of the active Ship agents and the suitable Service agents.

The Service Agent: The Terminal has been divided into services. Each service has assigned some specific stacking ranges. The main goal of a service agent is to determine the appropriate allocation for the arriving containers in the Terminal from a specific service (allocation problem) and the suitable configuration of the portion of the yard the agent controls. The agent has to coordinate with the other service agents in order to resolve any conflicts. The goal of the service agents is to maximise the stacking density in its yard portion The service agent launches this process automatically, when the agent considers it to be necessary (pro-activity).

The Transtainer Agent: Each transtainer is modelled as an autonomous agent whose goal is to efficiently perform the stacking operations of the containers in the yard. The transtainer agent has to minimize its empty movements. Each one of these agents is waiting for stacking requests from the different service agents, who facilitate the transtainer agent with:

- The containers to be moved from the stack and where they are located: this is done for vessel or external truck loading.
- The containers to be moved to the stack and where they must be placed: this is done for vessel or external truck unloading.

The Gate Agent: it controls the containers input and output by land. The agent has to manage the terminal gate assigned, informing the corresponding service agent when necessary. It will have to inform the corresponding service agent of the new containers' arrival (to store them) and of the trucks' arrival (to retire containers from the yard).

This section has presented a multi-agent system architecture for the automatic allocation problem in a port container terminal . Apart from the benefits obtained from a multi-agent approach, the independence which is obtained in all of the presented subsystems must be pointed out. This architecture provides a maintenance of the necessary co-operation in order to minimise the time the ships are in the container terminal. A first version of the system is currently implemented, which models the container terminal function of a real port. This prototype has been integrated integrated with a yard simulator developed at the same time.

References

[1] Martens A. and Uhrmacher A.M. Adaptative tutoring processes and mental plans. *In Proceedings of Intelligent Tutoring Systems-ITS 2002, Cerri S.A., Gouardères G. and Paraguaçu F. (Eds). Springer*, pages 71–80, 2002.

[2] R. C. Arkin. *Behavior-Based Robotics*. MIT Press, 1998.

[3] N.C. Audsley, A. Burns, Davis R.I, Tindell K.W., and A.J. Wellings. Fixed priority pre-emptive scheduling: An historical perspective. *Real-Time Systems*, 8:173–198, 1995.

[4] F. Barber, V. Botti, E. Onaindía, and A. Crespo. Temporal reasoning in reakt: An environment for real-time knowledge-based systems. *AICOMM*, 7(3):175–202, 1994.

[5] R.E. Bellman. Dynamic programming. *Princeton University Press*, 1957.

[6] 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.

[7] R.A. Brooks. A roboust layered control system for a mobile robot. In *IEEE Journal of Robotics and Automation*, 1986.

[8] S. Bussmann and K. Schild. Self-organising manufacturing control: an industrial application of agent technology. *Proc. 4th international Conference Multi-Agent Systems, Boston*, pages 87–94, 2000.

[9] C. Carrascosa, J. Bajo, V. Julian, J.M. Corchado, and V. Botti. Hybrid multi-agent architecture as a real-time problem-solving model. *Expert Systems With Applications*, 34, 2007.

[10] C. Carrascosa, A. Terrasa, A. García-Fornes, A. Espinosa, and V. Botti. A meta-reasoning model for hard real-time agents. *Selected Papers from the 11th Conference of the Spanish Association for Artificial Intelligence (CAEPIA 2005)*, 4177:42–51, 2006.

[11] Javier Palanca Cámara, Gustavo Aranda Bada, Carlos Carrascosa Casamayor, and Luis Hernández López. Hybrid mas to solve problems with different temporal constraints. In *New Trends in Real-Time Artificial Intelligence (NTeRTAIn 2006 / ECAI 2006)*, pages 19–28, 2006.

[12] Darren Cofer, Eric Engstrom, Robert Goldman, David Musliner, and Steve Vestal. Applications of model checking at honeywell laboratories. In *Lecture Notes in Computer Science, Vol. 2057*, pages 296–303. Springer-Verlag, 2001.

[13] F. Cristian. Understanding fault-tolerant distributed systems. In *Comm. ACM*, pages 57–78, 1991.

[14] F. Cristian and C. Fetzer. Probabilistic internal clock synchronization. In *Proc. of 12th Symposium on Reliable Distributed Systems. IEEE Computer Society Press*, pages 22–31, 1994.

[15] 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.

[16] G. Ekberg. Benefits of autonomous agent approach to manufacturing systems control. *Proc. of Third Annual Chaos East Technical Conference*, 1997.

[17] FIPA Foundation for Intelligent Physical Agents. http://www.fipa.org. *FIPA Abstract Architecture Specification*, 2002.

[18] 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.

[19] 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.

[20] B. Innocenti, López. B., and J. Salvi. How MAS support distributed robot control. *37th International Symposium on Robotics, ISR'06 ISBN.3-18-091956-6.*, 2006.

[21] B. Innocenti, B. López, and J. Salvi. A multi-agent collaborative control architecture with fuzzy adjustment for a mobile robot. *Proceedings of Third International Conference on Informatics in Control, Automation and Robotics. ISBN.972-8865-60-0.*, pages 523–526, 2006.

[22] B. Innocenti, B. López, and J. Salvi. A multi-agent system with distributed coordination for controlling a single robot. *7th Portuguese Conference on Automatic Control (CONTROLO), CDROM: Article 126*, 2006.

[23] N.R. Jennings, J.M. Corera, and I. Laresgoiti. Developing industrial multi-agent systems. In *First International Conference on Multi-agent Systems, ICMAS-95*, pages 423–430, 1995.

[24] N.R. Jennings, E. Mamdani, J. Corera, I. Laregoiti, F. Perriolat, P. Skarek, and L.Z. Varga. Using archon to develop real-world dai applications. *IEEE Expert*, 11:64–70, 1996.

[25] J. Jost and X. Li-Jost. Calculus of variations. *Cambridge University Press*, 1998.

[26] J.M. Lee. Riemannian manifolds. an introduction to curvature. *Springer-Verlag, New Tork, Inc*, 1997.

[27] A. Lucas. Decision support systems for arrivals flow management. *ATCA International Technical Conference, Prague*, 1997.

[28] Corchado J. M. and Laza R. Constructing deliberative agents with case-based reasoning technology. *International Journal of Intelligent Systems.*, 18:1227–1241, 2003.

[29] Glez-Bedia M. and Corchado J. M. A planning strategy based on variational calculus for deliberative agents. *Computing and Information Systems Journal*, 10:2–14, 2002.

[30] D. Mackenzie. A design methodology for the configuration of behavior-based mobile robotics. In *PhD Dissertation. University of Georgia Institute of Technology*, 1996.

[31] D. Mills. Improved algorithms for synchronizing computer network clocks. In *IEEE Transactions Networks*, pages 245–254, 1995.

[32] M. Rebollo, V. Julian, C. Carrascosa, and V. Botti. A mas approach for port container terminal management. In *Proceedings of the 3rd IBEROAMERICAN WORKSHOP ON DAI - MAS*, pages 83–94, 2000.

[33] M. Rebollo, V. Julián, C. Carrascosa, and V. Botti. A multi-agent system for the automation of a port container terminal. *Proc. of Agents in Industry Workshop, Autonomous Agents, Barcelona*, 2000.

[34] D. Ribas, N. Palomeras, E. Hernandez, P. Ridao, and M. Carreras. $ICTINEU^{AUV}$ wins the first SAUC-E competition. Scheduled for presentation during the Regular Sessions "Field Robotics: Systems and Applications" (WeA5). 2007 IEEE International Conference on Robotics and Automation, 10-14 April, 2007, Rome, Italy.

[35] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall International Editions, 1995.

[36] SAUC-E. Available at http://www.dstl.gov.uk/news events/competitions/sauce/.

[37] 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.

[38] VICOROB-UdG Team. Available at http://eia.udg.es/sauce.

[39] A. Terrasa, A. García-Fornes, and V. Botti. Flexible real-time linux. *Real-Time Systems Journal*, 2:149–170, 2002.

[40] K. Tindell and A. Burns. Guaranteeing message latencies on control area network (can). In *Proc of the 1st International CAN Conference*, 1994.

[41] A. Van Breemen. Agent-based multi-controller systems – a design framework for complex control problems. In *PhD Dissertation. University of Twente*, 2001.

Vicente Julian
Departamento de Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia

e-mail: `vinglada@dsic.upv.es`

Carlos Carrascosa
Departamento de Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia

e-mail: `carrasco@dsic.upv.es`

Juan Manuel Corchado
Departamento de Informática y Automática
Universidad de Salamanca

e-mail: `corchado@usal.es`

Javier Bajo
Universidad Pontificia de Salamanca

e-mail: `jbajope@upsa.es`

Esteve Acebo
Agent Research Group
Universitat de Girona

e-mail: `acebo@ima.udg.es`

Bianca Innocenti
EXIT group
Universitat de Girona
Lluís Santaló s/n
Girona 17071. Spain
e-mail: `bianca.innocenti@udg.edu`

Vicente Botti
Departamento de Sistemas Informaticos y Computacion
Universidad Politecnica de Valencia

e-mail: `vbotti@dsic.upv.es`