Incorporating Temporal-Bounded CBR techniques in Real-Time Agents

M. Navarro, S. Heras, V. Julián[1], V. Botti

Departamento de Sistemas Informáticos y Computación,
Universidad Politécnica de Valencia,
C\ Camino de Vera s/n, 46220 Valencia, Spain

Abstract

Nowadays, MAS paradigm tries to move Computation to a new level of abstraction: Computation as interaction, where large complex systems are seen in terms of the services they offer, and consequently in terms of the entities or agents providing or consuming services. However, MAS technology is found to be lacking in some critical environments as real-time environments. An interaction-based vision of a real-time system involves the purchase of a responsibility by any entity or agent for the accomplishment of a required service under possibly hard or soft temporal conditions. This vision notably increases the complexity of these kinds of systems. The main problem in the architecture development of agents in real-time environments is with the deliberation process where it is difficult to integrate complex bounded deliberative processes for decision-making in a simple and efficient way. According to this, this work presents a temporal bounded deliberative case-based behaviour as an anytime solution. More specifically, the work proposes a new temporal-bounded CBR algorithm which facilitates deliberative processes for agents in real-time environments, which need both real-time and deliberative capabilities. The paper presents too an application example for the automated management simulation of internal and external mail in a department plant. This example has allowed to evaluate the proposal investigating the performance of the system and the temporal bounded deliberative case-based behavior.

Key words: Real-Time Systems, Multi-Agent Systems, Case-Based Reasoning

[1]Corresponding Author: Dept. Sistemas Informaticos y Computacion Univ. Politecnica de Valencia. 46022, Valencia, Spain
Tel. + 34 96 387 70 07 + 73583 Fax. + 34 96 387 73 59
vinglada@dsic.upv.es

1. Motivation

In multi-agent systems (MAS) one of the most important goals is to build systems capable of making decisions in an autonomous and flexible way, where different agents interact forming coalitions or organizations among themselves to achieve their current goals. Based on this idea, the MAS paradigm tries to move Computation to a new level of abstraction: Computation as interaction [1]. Given this approximation, it seems natural to view large complex systems in terms of the services they offer, and consequently in terms of the entities or agents providing or consuming services [2]. This means making a drastic change in the development of large complex systems, where the technology of multi-agent systems has some characteristics that show its potential to support this new paradigm of computation as interaction.

Nevertheless, MAS technology is found to be lacking in some environments. Specifically, the applicability of multi-agent systems to real-time environments needs specific functionalities such as appropriate time management in the communicative processes or time-bounded deliberative processes. Moreover, an interaction-based vision of a system tries to model the system behavior through the invocation of services among the entities of that system. So, the responsibility acquired by any entity or agent for the accomplishment of a required service under possibly hard or soft temporal conditions increases the complexity of these kinds of systems.

These functionalities are not available at the moment. Agents which work in the above-mentioned environments, must fulfil specific restrictions, which implies the need for specific agent architectures. A hypothetical and appropriate agent designed for real-time environments must accomplish its goals, invoking services and related tasks with the added difficulty of temporal restrictions. In accordance with this, a *Real-Time Agent (RTA)* can be defined as an agent with temporal restrictions in, at least, one of its responsibilities (goals, services, tasks, ...) [3] [4]. The RTA may have its interactions bounded, and this modification will affect all communication processes in the multi-agent system where the RTA is located.

The main problem in the architecture of a RTA is with the deliberation process. This process probably uses AI techniques as problem-solving methods to compute more *intelligent* actions. If this is the case, it is difficult to extract the time required, because it can either be unbounded or its variability is very high. If the agent has to operate in a real-time environment, the agent complexity in order to achieve any or all of these features is greatly increased. Thus, in a RTA an efficient integration of high-level, deliberative processes within reactive processes is necessary. When using AI methods, it is necessary to provide techniques that allow their response times to be bounded. These techniques are mainly based on well-known Real-Time Artificial Intelligence System (RTAIS) techniques [5] [6].

Therefore, it would be interesting to integrate complex deliberative processes for decision-making in real-time agents in a simple and efficient way. Some of the most important features of agents are their capabilities: 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. Intelligent agents may use a lot of reasoning mechanisms to achieve these capabilities. For example, planning techniques [7] or Case-Based Reasoning (CBR) techniques [8]. The applications of CBR to control some aspects of the deliberative process of agents in MAS developed for specific purposes are many [8]. The main assumption in CBR is that similar problems have similar solutions. Therefore, when a CBR system has to solve a new problem, it retrieves precedents from its case-base and adapts their solutions to fit the current situation. This reasoning methodology greatly resembles the way people reason about their experiences. CBR can thus be very suitable applied in agent reasoning, where similar problems should have similar solutions. However, few of the existing approaches cope with the problem of applying CBR as a deliberative engine for agents in MAS with real-time constraints. With this in mind, this work presents a temporal bounded deliberative case-based behaviour as an *anytime* solution. This approach facilitates deliberative capabilities in a real-time agent, which facilitates the development of hybrid agent architectures with both real-time and deliberative capabilities.

Moreover, a case of study has been implemented in a simulated environment in order to evaluate the proposal. Different experiments have been implemented, basically, investigating the performance of the system and the temporal bounded deliberative case-based behavior. The results show the benefits obtained with the integration of this deliberative behaviour into a real-time agent while maintaining the fulfilment of the critical time restrictions.

The rest of the paper is structured as follows: section 2 presents a study of related work; section 3 focuses on how to incorporate a temporal bounded CBR into a real-time agent; a simulated application example and the analysis of the results obtained is shown in section 4; finally, conclusions are described in section 5.

2. Related Works

This work tries to apply CBR techniques, as a deliberative engine, for agents in MAS with real-time constraints. In accordance with this idea, this section analyses previous works of CBR applications in MAS. Then, previous approaches of agent technology for real-time environments are also studied.

2.1. CBR applications in Agent technology

In AI research, the combination of several AI techniques to cope with specific functionalities in hybrid systems has a long history of successful applications. A CBR system provides agent-based systems with the ability to reason and learn autonomously from the experience of agents. These systems propose solutions for solving a current problem by reusing or adapting other solutions that were applied in similar previous problems. With this aim, the system has a case-base that stores its knowledge about past problems together with the solution applied in each case. The most common architecture of a CBR system is shown in Figure 1. It consists of 4 phases (commonly named as the 4 R's) [8]: the first one is the *Retrieval* phase, where the most similar cases are retrieved from the case-base; then, in the *Reuse* phase, those cases are reused to try to solve the new problem at hand; after this, in the *Revise* phase the solution achieved is revised and adapted to fit the current problem and; finally, in the *Retain* phase, the new case is stored in the case-base (if this is considered necessary). Thus, the case-base is updated and the system learns from new experiences.

The integration of CBR systems and MAS has been studied taking many different approaches. Therefore, the literature of this scientific area reports research on systems that integrate a CBR engine as a part of the system itself [9], other MAS that provide some or all of their agents with CBR capabilities, or even the development of BDI agents following a CBR methodology [10]. In this section we have focused the review on the second approach, CBR applied to MAS, since it fits better the scope of our paper.

The distributed nature of the data sources in the real world makes it necessary to use decentralised access and control systems. MAS technology fits these requirements perfectly. As pointed out before, CBR provides deliberative agents with reasoning and learning capabilities. Since the 90's, the synergies between MAS and CBR are many, although the approaches differ. One early approach was the development of multi-agent CBR systems, which are MAS with cooperative agents characterised by the distribution of their case-bases and/or certain phases of the CBR cycle between them.

Some examples of multi-agent CBR systems are: the technique called *Negotiated Case Retrieval* [11], which allows a set of agents to respond to a global request made to the group by looking for the solution in their distributed case-bases; the *Federated Peer Learning* framework [12][13], where the possible cooperation modes between homogeneous agents with learning capabilities were investigated; the *Collaborative CBR (CCBR)* [14] which allows agents to share problem solving experiences and responsibilities to generate travel route plans that fit the preferences of the user in a territory with an unknown map; the

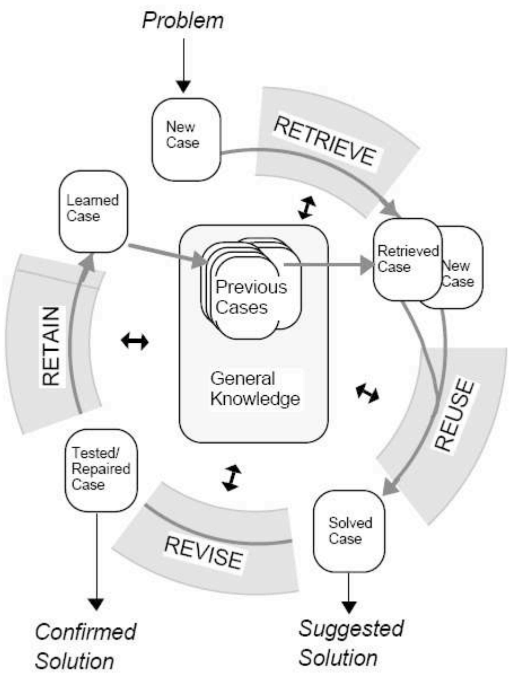

Figure 1: CBR reasoning cycle [8].

Multi-CBR (MCBR) [15][16], where several case-bases for solving the same task or different tasks in different environments can co-exist; and finally, a distributed learning methodology that combines individual and cooperative learning in a MAS framework [17]. The latter work was the basis of later research in the area of argumentation in MAS.

The application of CBR to manage argumentation in MAS is an innovative approach that has recently made a significant contribution to the fields of AI and argumentation theory.[18]. In this field, a case-based negotiation model for reflective agents (i.e. agents aware of their temporal and situational context) was designed [19]. In their framework, a set of situated agents that control certain sensors try to track several mobile targets. The aim of the agents is to coordinate their activities and collaborate to track the path to as many targets as possible. Another piece of research in this area is the new case-based selection model called *ProCLAIM* [20], which extended the architecture of the decision support MAS for the organ donation *CARREL+* and the *Argumentation Based Multi-Agent Learning (AMAL)* framework [21] which features a set of agents that try to solve a classification problem by aggregating their expert knowledge by means of a collaborative deliberation dialogue.

Furthermore, a research area where the integration between CBR and agent techniques has produced a huge amount of successful applications is the robot navigation domain. An important contribution here was a case-based model for managing the *ROBOCATS* system [22], playing in the *Robocup* league. This framework considers different types of cases, which are used to manage the goal-keeper position, to select the appropriate team formations and to recognise the different game states. In addition, a case-based approach to model and predict the opponent's movements was also studied. Also, the *RUPART* system [23] features a hybrid planner for a mobile robot that delivers mail in real-time. This system combines reactive planning (based on behaviours) with deliberative planning (based on cases) to decide the best path to follow (or at least, to specify a generic action to take when there is no similar case to retrieve). Another application of CBR to manage autonomous navigation tasks was proposed in a system for the automatic selection and modification of assemblage parameters [24]. This system replaces the typical manual configuration of the robot navigation parameters and increases the efficiency of the navigation by automatically selecting and adapting the parameters that best fit the robot task-environment in real-time. Finally, some important research that applies CBR to a MAS with mobile robots was started with a model that integrated a CBR agent in a multi-agent navigation system to determine if a current situation in the robot environment is similar to a previous one [25]. Therefore, problematic situations can be identified and overcome by taking the proper action to avoid them. Later, this framework gave rise to a system for modeling team playing behavior in the robot soccer domain by using CBR [26].

The case cited above are outstanding examples of systems that join the research efforts and results of both CBR and MAS. In addition, the applications of CBR to control some aspects of the deliberative process of agents in MAS developed for specific purposes are many. Most are not intended to cope with the problem of applying CBR as a deliberative engine for agents in MAS with real-time constraints. In fact, in the systems reviewed, the concept of real time is relative with *doing things quickly*. However, in real-time multi-agent systems this concept implies meeting deadlines [4].Therefore, if CBR is applied, the reasoning cycle must observe temporal restrictions.

2.2. Real-Time Agents

The incorporation of artificial intelligence techniques, and more specifically the MAS paradigm, in a real-time environment is justified due to the necessity to provide real-time systems with 'intelligent' capabilities for solving complex problems. In order to do so, Artificial Intelligence (AI) methods are moving towards more realistic domains requiring real-time responses, and real-time systems are moving towards more complex applications requiring intelligent behaviour. To cover this field of research, the *Real-Time Artificial Intelligence System (RTAIS)* area was created. The RTAIS is a system that must accomplish complex and critical processes in what is probably a dynamic environment with temporal restrictions by using AI techniques. A broader definition of RTAIS can be found in [5], [27] and [28].

Different approaches have been proposed for adapting AI techniques to real-time requirements. The main works are aimed at dealing with:

- The modification of traditional AI algorithms to improve their predictability.
- AI software architectures suitable for real time operations.
- The modification of traditional scheduling policies in real-time systems to adapt to the use of less predictable algorithms.

Previous approaches to RTAIS can be found in the literature. Anytime algorithms [29] and approximate processing [5] are the most promising algorithms. One line of research in RTAI has been to build large applications or architectures that embody real-time concerns in many components [5], such as Guardian [30] and Phoenix [31].

Another example is SA-CIRCA (Self-Adaptive Cooperative Intelligent Real-Time Control Architecture), proposed by Musliner et al. [32]. This architecture is an evolution of the classic CIRCA architecture. CIRCA architecture is composed of two independent systems: a planner in charge of creating plans that fulfill the temporal constraints, and a real-time system that executes them. A new component called Adaptative Mission Planner(AMP) is added in SA-CIRCA. This component gives the reasoning and auto-adaption capacity. One of the problems of this architecture is its limited reactivity. The system is only capable of recognizing a limited number of simultaneous threats and if the limit is exceeded, system integrity is compromised.

In the RTAIS research area, a Real-Time Agent (RTA) can be defined as an agent with temporal restrictions [33]. In these agents, it is necessary to take temporal correctness into account, which is expressed by means of a set of temporal restrictions that are imposed by the environment. The RTA must, therefore, ensure the fulfilment of such temporal restrictions. The RTAIS literature reports works that are aimed at developing mechanisms to support real-time agents. Several architectures have been proposed for real-time agents, as well as research in scheduling agent tasks within the architecture. Much of the real-time agent scheduling work relies on the assumption that in order to perform a task, an agent or set of agents may have multiple ways of solving the same problem, each with varying time requirements to compute the result, and with a variation in the quality of the results produced. Typically, the more time available to solve the problem, the higher the quality of the result. This becomes very useful in real-time agent scheduling because it allows for a trade-off between the quality of the result and the amount of time required in order to meet specified time constraints. Garvey et al. in [34] presents a design-to-time scheduling algorithm for incremental decision-making that provides a hierarchical abstraction of the problem solving processes, which describe alternative methods for solving a specific goal depending on the amount of time available. This algorithm is extended in [35] to develop a more general model that can take into account any scheduling criteria, such as time, cost, and quality, and it can use uncertainty as part of the decision-making process. An example of the use of the design-to-criteria model is the DECAF architecture [36] which incorporates scheduling algorithms based on this model.

The ObjectAgent Architecture is another example of real-time agents. This architecture, developed by Pinceton Satellites in 2001 [37], is used to control little mono-function satellites launched. These satellites work together as a unique satellite with multiple functions. Each mini-satellite is identified by an agent with its temporal restrictions. This architecture supports real-time communication, while the net topology is known and predictable. To do so, ObjectAgent has a special agent, called Postoffice, which is in charge of distribute messages among the agents. If the Postoffice is statically located, the time needed to send a message through the network is predictable. Unfortunately, this assumption is only true for very specific networks (CAN networks, inter-satellite laser links, etc. ...). Thus, if this platform were extrapolated to common network media (Ethernet, serial, wifi, etc ...), this feature would be lost.

DiPippo et al. [38] [39] presents a Real-Time Multi-Agent system (RT-MAS) based in RT-Corba [40]. The operation of this system is based on CORBA, but here the client and server have real-time features. For example, a server can register in the *Scheduling Service* a task whose parameters are a deadline and service quality. Then when a client requests the task, the *Scheduling Service* and the real-time component select a server using the scheduler algorithm -more specifically EDF algorithm- and provide the client with the information about which server is the most appropriate to fulfil the task. This RT-MAS provides a Middleware that works as a multi-agent platform taking into account the system temporal constraints. The communication among agents is implemented using KQML with an extension where the temporal restrictions and the service quality parameter can be reflected. However, this approach to the real-time multi-agent system has some problems. On one hand, the time that the Scheduling Service needs to deliberate is unknown and, on the other hand, the communication process is temporally unbounded, and therefore, unpredictable.

Another example of Real-Time Agent is presented by Prouskas et al. in [41]. They define time-aware agents as agents capable of operating in two temporal dimensions: agent-agent and human-agent, seamlessly combining the predictability and reliability of small-scale real-time exchanges with the fuzzy temporal requirements of large-scale human interactions. Time-aware agent systems deal with an amalgam of hard, soft, human and non-real-time interactions, reason about the temporal constraints placed on the system by each type of interaction, make transformations between themselves and co-ordinate (schedule) activities seamlessly irrespective of their constituent constraints. An architecture where these time-aware agents can exist and work towards achieving their goals has been developed and tested by using a prototype implementation named TARA, implemented in the Agent Process Interaction Language (April) [42].

In addition, the ARTIS agent specifically designed to develop Real-Time Systems was also presented [6]. An ARTIS agent is an agent able to operate in distributed real-time domains. The ARTIS architecture is an extension of the blackboard model [43], which has been adapted to work in hard real-time environments. This architecture includes the use of well-known RTAIS techniques. This approach guarantees reacting on the environment in a dynamic and flexible way. It incorporates all of the necessary aspects that the agency features provide a software system, including social aspects, but adapted to hard real-time environments. The ARTIS agent incorporates a control module that is responsible for the real-time execution of the tasks of the agent. This module is the component in charge of controlling how and when the different components of the ARTIS agent are executed. In order to manage the communication processes within other agents, the ARTIS agent has been extended with a communication module (CoMo) [3]. On the other hand, the execution of the reflex components (the components with critical temporal restrictions) is controlled by a sub-module of the control integrated within a Real-Time Operating System. According to these features, the ARTIS agent architecture guarantees an agent response that satisfies all of the critical temporal restrictions of the system. Its capacities for problem-solving, for adaptability and for proactivity help to provide a good enough response for the current environment status. Its critical timing requirements are 100% guaranteed by means of an off-line schedulability analysis. The main problems of this proposal are the lack of complex reasoning capabilities in ARTIS agents and the complexity of the design and implementation processes, which makes the use of this proposal very difficult.

Finally, SIMBA architecture allows the development of multi-agent systems which work properly in social real-time domains [3]. The SIMBA architecture allows the development of different related agents for hard real-time environments. The SIMBA system is mainly formed of a set of ARTIS agents with probably critical temporal restrictions. This set of agents controls the subsystem of the real-time environment with hard critical constraints. Additionally, the system may integrate different types of agents, which cover other non-critical activities in the system. For this reason, SIMBA must be able to incorporate heterogeneous agents using standard agent-interaction processes. The main problem of this proposal is the integration of non-critical agents with critical agents and the employment of specific, and sometimes complicated, communication and model languages.

Studied approaches have shown the utility of using CBR techniques as a deliberative engine for agents in MAS. Moreover, real-time agents have been shown to be an interesting research area where different approaches have tried to apply the MAS paradigm in real-time environments. Approaches in this area can be found in [44] and [45]. Nevertheless, the employment of a temporal bounded CBR in the agent deliberation process is still an open topic.

3. Temporal bounding agent reasoning

Problems to be solved by a real-time agent must be complex problems which need real-time responses. According to Knowledge Engineering,these kinds of problems require the use of knowledge-intensive (KI) tasks but with an added complexity due to their hard or soft temporal constraints. KI tasks have been traditionally studied and classified, mainly according to Newell's Taxonomy [46]. There are different works that try to classify KI tasks to facilitate their development [47] [48] [49]. The latter is one of the most widely employed, since it covers the different types of tasks traditionally addressed by the AI community. This task model identifies two main classes of KI tasks: analysis tasks and synthesis tasks. The main difference between them is that the first works with an existing system, while the goal of the second one is to build the system itself. That is, the system does not exist and it is the result of the execution of the synthesis task.

The Analysis tasks can be classified into five sub-classes: Classification, Valuation, Monitoring, Diagnostic and Prediction. On the other hand, Synthesis tasks are divided into: Design, Planning, Assignation and Scheduling.

Different AI techniques has been employed trying to implement these tasks with better or worse outcomes. In problem solving terms, the world is often a repetitive and regular place. Similar problems tend to recur and require similar solutions. Case-based reasoning techniques attempt to exploit this repetitiveness and regularity by leveraging past problem solving experience - in the form of concrete problem solving cases - when it comes to solving new problems. In short, a case-based reasoner solves new problems by adapting solutions that were used to solve old problems [50]. Taking this idea into account, CBR techniques have been used to solve both Analysis and Synthesis tasks.

- **Analysis Tasks**: Where an object or event must be matched with another object in a library from which an answer can be inferred. These tasks are easy to implement because they are in accordance with the CBR cycle, with cases that tend to be easier to perform and recover.

- **Synthesis Tasks**: These tasks are used in the domains of design or planning to try to simplify the creative process. The final plan or design is produced by the adjustment of plans or designs stored in the case-base. Design, planning an configuration tasks are some examples of the synthesis tasks.

In both analytical and synthetical approaches, the literature of CBR reports many successful applications. Thus, CBR has coped with analytic tasks in systems devoted to prediction [51][52], fault [53][54] and medical diagnosis [55], legal reasoning [56], mediation [57], negotiation [58][19], tutoring [59] and customer service support [60][61]. In addition, CBR systems have also tackled synthetic tasks in systems that work in the domains of design [62][63], planning [64] and configuration [65].

However, if we want to use CBR techniques as a reasoning mechanism in real-time agents, it is necessary to adapt these techniques to be executed guaranteeing real-time constraints. The following section explains how to develop temporal bounded CBR-based techniques to be integrated into a real-time agent architecture. This approach will allow for a more efficient execution time management, according to the agent's goals.

3.1. Designing a Temporal Bounded CBR

CBR systems are highly dependent on their application domain. Therefore, designing a general CBR model that might be suitable for any type of real-time domain (hard or soft) is, to date, unattainable. In real-time environments, the CBR phases must be temporally bounded to ensure that solutions are produced on time. In this section, we present some guidelines with the minimum requirements to be taken into account to implement a CBR method in real-time environments.

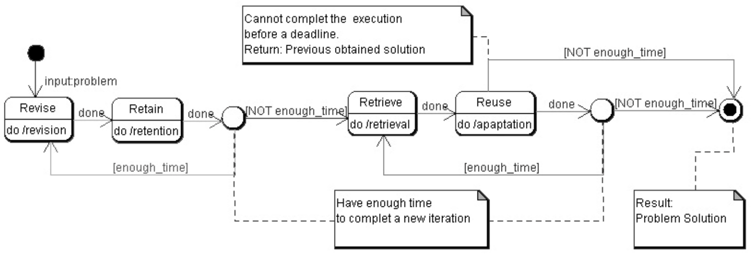

Figure 2: Temporal Bounded CBR cycle.

As a first step, we propose a modification of the classic CBR cycle in order to adapt it to be applied in real-time domains. Figure 2 shows a graphical representation of our approach. Firstly, we group the four reasoning phases that implement the *cognitive task* of the real-time agent into two stages defined as: the *learning stage*, which consists of the revise and retain phases and the *deliberative stage*, which includes the retrieve and reuse phases. Both phases will have their own execution time scheduled. Therefore, the designer can choose to either assign more time to the deliberative stage or keep more time for the learning stage (and thus, design agents that are more sensitive to updates). These new CBR stages must be designed as an anytime algorithm [29], where the process is iterative and each iteration is time-bounded and may improve the final response.

In accordance with this, the operation of our Time Bounded CBR cycle (TB-CBR) is the following. Firstly, the main difference that can be observed between the classic CBR cycle and the TB-CBR cycle is the starting phase. Our real-time application domain and the restricted size of the case-base (as explained in the following sections) gives rise to the need to keep the case-base as up to date as possible. Commonly, recent changes in the case-base will affect the potential solution that the CBR cycle is able to provide for a current problem. Therefore, the TB-CBR cycle starts at the learning stage, checking if there are previous cases waiting to be revised and possibly stored in the case-base. In our model, the solutions provided at the end of the deliberative stage will be stored in a solution list while a feedback about their utility is received. When each new CBR cycle begins, this list is accessed and while there is enough time, the learning stage of those cases whose solution feedback has been recently received is executed. If the list is empty, this process is omitted.

After this, the deliberative stage is executed. Thus, the retrieval algorithm is used to search the case-base and retrieve a case that is similar to the current case (i.e. the one that characterizes the problem to be solved). Each time a similar case is found, it is sent to the reuse phase where it is transformed into a suitable solution for the current problem by using a *reuse* algorithm. Therefore, at the end of each iteration of the deliberative stage, the TB-CBR method is able to provide a solution for the problem at hand, although this solution can be improved in following iterations if the deliberative stage has enough time to perform them.

Hence, the temporal cost of executing the *cognitive task* is greater than or equal to the sum of the execution times of the learning and deliberative stages (as shown in equation 1):

$$\begin{aligned} t_{cognitiveTask} &\geq t_{learning} + t_{deliberative} \\ t_{learning} &\geq (t_{revise} + t_{retain}) * n \\ t_{deliberative} &\geq (t_{retrieve} + t_{reuse}) * m \end{aligned} \tag{1}$$

where $t_{learning}$ and $t_{deliberative}$ are the total execution time of the learning and deliberative stages; t_x is the execution time of the phase x and n and m are the number of iterations of the learning and deliberative stages respectively.

According to this temporal restriction, a first view of the TB-CBR algorithm can be seen in Algorithm 1. This algorithm can be launched when the real-time agent considers it appropriate and there is enough time for it to be executed. The real-time agent indicates to the TB-CBR the maximum time (t_{max}, where $t_{max} >= t_{cognitiveTask}$) that it has available to complete its execution cycle. The time t_{max} must be divided between the learning and the deliberative stages to guarantee the execution of each stage. The $timeManager(t_{max})$ function is in charge of completing this task. Using this function the designer must specify how the real-time agent acts in the environment. The designer can assign more time to the learning stage if it desires a real-time agent with greater capacity to learn. On the contrary, the function can allocate more time to the deliberation stage. Regardless of the type of agent, the $timeManager$ function should allow sufficient time for the deliberative stage to ensure a minimal answer.

The first phase of the algorithm executes the learning stage. This stage is executed only if the real-time agent has the solutions of previous executions stored in the *solutionQueue*. The solutions are stored just after the end of the deliberative stage. The deliberative stage is only launched if the real-time agent has a problem to solve in the *problemQueue*. This configuration allows the agent to launch the TB-CBR in order to only learn (no solution is needed and the agent has enough time to reason about previous decisions), only deliberate (there are no previous solutions to consider and there is a new problem to solve) or both.

```
   Input: $t_{max}$
1  $(t_{learning}, t_{deliberative}) \longleftarrow$ timeManager($t_{max}$)
2  if solutionQueue $\neq \emptyset$ then
3      {Learning stage}$^{\leq t_{learning}}$
4  end
5  if problemQueue $\neq \emptyset$ then
6      {Deliberative stage}$^{\leq t_{deliberative}}$
7  end
```

Algorithm 1: Abstract TB-CBR algorithm

This section explains how to adapt each CBR phase to meet the real-time constraints of a RTAIS.

3.1.1. Data Format

The design decision about the data structure of the case-base and the different algorithms that implement each CBR phase are important factors for determining the execution time of the CBR cycle. The number of cases in the case-base is another parameter that affects the temporal cost of the retrieval and retain phases. Thus, a maximum number of cases in the case-base must be defined by the designer. Note that, usually, the temporal cost of the algorithms that implement these phases depends on this number.

For instance, let us assume that the designer chooses a *hash table* as a data structure for the case-base. This table is a data structure that associates keys to specific values. Search is the main operation that it supports in an efficient way: it allows access to elements (e.g. phone and address) by using a *hash function* to transform a generated key (e.g. owner name or account) to a *hash number* that is used to locate the desired value. The average time to make searches in hash tables is constant and defined as O(n) in the worst case. Therefore, if the cases are stored as entries in a hash table the maximum time to look for a case depends on the number of cases in the table (i.e. O(♯cases)). Similarly, if the case-base is structured as an *auto-balanced binary tree* the search time in the case-base in the worst case would be O(log n).

In any case, the retrieval and retention time can be reduced by using an indexing algorithm. These algorithms organize the case-base by selecting a specific feature (or set of features) from the cases, grouping together those cases that share the same values for these features. This reduces the cost of the search for similar cases (for retrieval or previous to the introduction of new cases in the case-base) to a specific set of cases with the same index as the current case [66] [67] [68].

3.1.2. Retrieve Phase

In the retrieve phase the retrieval algorithm is executed to find a case that is similar to the current problem in the case-base. In order to bound the temporal cost of the algorithm and to ensure the adequate temporal control of this phase, the execution time of the algorithm is approximated to its worst-case execution time (WCET). Thus, the temporal cost of the execution of this algorithm will never exceed the WCET. Since WCET depends on the structure of the case-base and its number of cases, the designer must calculate this WCET and use this time to estimate the time needed to execute an iteration of the retrieval algorithm.

$$t_{retrieve} = WCET(f_{retrieval}(currentCase, case - base)) \tag{2}$$

The execution of the retrieval algorithm will provide a unique case similar to the current problem (if it exists in the case-base). This result is used as input for the reuse phase. However, in the following iterations of the deliberative stage more similar cases can be retrieved with the intention of providing a more accurate solution for the problem. This functionality must be done in the proposed TB-CBR algorithm by means of the *adaptProblem* function, where the problem is adapted to a correct format, and by the *search* function, which searches similar cases in the case-base.

3.1.3. Reuse Phase

In this phase, the selected case obtained from the retrieve phase is adapted to be used as a potential solution for the current problem. Thus, this case is stored in a list of selected cases. Each time the reuse phase is launched, the adaptation algorithm searches this list and produces a solution by adapting a single case or a set of cases to fit the context of the current problem to be solved. Therefore, the execution time of this algorithm depends on the number of cases that the algorithm is working with.

$$t_{reuse} = \begin{cases} WCET(f_{adaptation}(firstCase)) \\ f_{adaptation}(listOfCases) \end{cases} \tag{3}$$

As shown in equation 3, to guarantee that the RTA assigns enough time to execute the cognitive task, the designer must know the WCET to execute the adaptation algorithm in the first iteration (with one case). Thus, the RTA can estimate if the deliberative stage can be completed and provide at least one solution. In order to control the execution time of the adaptation algorithm in subsequent iterations, the RTA must be able to stop the execution of the algorithm if it realises that the time assigned to complete the deliberative stage will be exceeded. Then, the RTA provides the best solution among the solutions completed in previous iterations. This solution is stored in a list of solutions to be verified in the learning stage. This phase is implemented in the TB-CBR algorithm by means of the *adaptSolution* function.

3.1.4. Revise Phase

During this phase, the accuracy of the final solutions obtained in previous executions of the TB-CBR cycle is checked. The algorithm only checks one solution per iteration, fixing the potential problems that it had in cases of erroneous results. The outcome of this phase is used to update the case-base. Thus, the maximum temporal cost of this phase is bounded by the WCET of the revision algorithm:

$$t_{revise} = WCET(f_{revision}(solution)) \tag{4}$$

Note that, in order to guarantee a known maximum execution time, this check must be performed automatically by the computer without human interference. This WCET does not depend on the number of stored solutions or the number of cases in the case-base and again, is fixed by the selected algorithm. In our TB-CBR proposal, this phase is implemented by an *analysesResult* function which determines if the solution is correct or not, and if it has to be included in the case-base.

3.1.5. Retain phase

One of the most important phases in the TB-CBR cycle is the retain phase. In this phase it is decided whether a checked solution must be added as a new case in the case-base. Here, keeping the maximum size of the case-base is crucial, since the temporal cost of most retention algorithms depends on this size. If there is a case in the case-base that is similar enough to the current case, this case (its problem description and solution) is updated if necessary. On the contrary, if there is not a case that represents the problem solved, a new case is created and added to the case-base. Maintaining the maximum size of the case-base could entail removing an old case from it. This decision should be taken by the retention algorithm. Nevertheless, the maximum temporal cost that the retain phase needs to execute one iteration is the retention algorithm WCET.

$$t_{retain} = WCET(f_{retention}(solution, case - base)) \tag{5}$$

This phase is built by the *retainResult* function which includes the solution in the case-base if it is sufficiently significant. To do so, the function determines if the solution entails adding a new case in the case-base or if a similar case exists. If the solution has a similar case in the base-case, the similar case is update with the new data that the TB-CBR can extract from the solution. If the solution entails adding a new case and the case-base is full, the algorithm extracts a case from the case-base that it considers outdated or useless to make space for a new case. Otherwise, the new case is added to the case-base.

According to the temporal analysis of each phase of the CBR cycle, an extended version of the previously presented TB-CBR algorithm is shown in Algorithm 2. The anytime behavior of the TB-CBR is achieved through the use of two loop control sequences. The loop condition is built using the *enoughTime* function, which determines if a new iteration is possible according to the total time that the TB-CBR has to complete each stage.

```
Input: $t_{max}$
($t_{learning}$,$t_{deliberative}$) ⟵ timeManager($t_{max}$)
if solutionQueue $\neq \emptyset$ then
    while enoughTime($t_{now}$,$t_{revise}$,$t_{retain}$,$t_{learning}$) and
    solutionQueue $\neq \emptyset$ do
        $r$ ⟵ pop(solutionQueue)
        {adequate ⟵ analysesResult($r$)}$^{\leq t_{revise}}$
        if adequate then
            {retainResult($r$)}$^{\leq t_{retain}}$
        end
    end
end
if problemQueue $\neq \emptyset$ then
    problem ⟵ pop(problemQueue)
    repeat
        {cases ⟵ push(search(adaptProblem(problem)))}$^{\leq t_{retrieve}}$
        {solution ⟵ adaptSolution(cases)}$^{\leq t_{reuse}}$
        bestSolution ⟵ bestSolution(solution,bestSolution)
    until ¬enoughTime($t_{now}$,$t_{retrieve}$,$t_{reuse}$,$t_{deliberative}$) ;
    solutionQueue ⟵ push(bestSolution)
    return bestSolution
end
```

Algorithm 2: TB-CBR Extended

4. Example: Postman Service Problem

The previous section described some guidelines for applying a temporally bounded CBR as a deliberative technique in a real-time agent, proposing a general algorithm. According to these guidelines, this section presents an application example of a multi-agent system including agents with real-time constraints which incorporates a temporally bounded CBR deliberative process.

The problem to be solved consists of the automated management of the internal and external mail (post mail, non-electronic) in a department plant. The system created by this automation must be able to request the shipment of a letter or package from an office on one floor to another office on the same floor, as well as the reception of external mail at a collection point for subsequent distribution. Once this service has been requested, a set of mobile robots must gather the shipment and address it to the destination. Note that each mail or package distribution must be ended before a maximum time, specified in the shipment request. This example is a complex problem and is clearly distributed, which makes the multi-agent system paradigm suitable for its resolution. Moreover, the robot agents must incorporate temporal bounded reasoning techniques to estimate the appropriate paths to the different target positions they need to achieve.

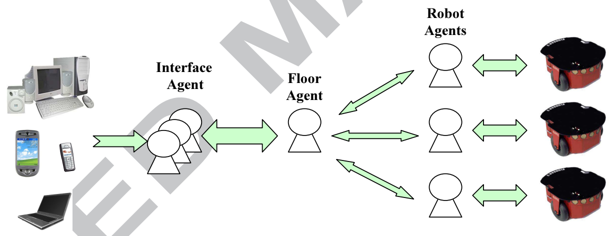

Figure 3: Diagram of the developed multi-agent system.

Figure 3 shows a diagram with a general view of the proposed example. There are three types of agents in the figure:

- **Interface Agent**: this agent is in charge of gathering user requests. A request will be transmitted by the Interface Agent to the mail service provided by the department plant (this service is provided by the Floor Agent). The user can employ a mobile device to communicate with an Interface Agent.
- **Floor Agent**: the mission of this agent is to gather the delivery/reception of mail sent by the Interface Agent and to distribute the work among the available Robot Agents around the plant. The Floor Agent interacts with the Robot Agents by means of the invocation of their mail delivery service.
- **Robot Agent**: the Robot Agent is in charge of controlling a physical robot and managing the mail list that this robot must deliver. In the

proposed example, three Robot Agents are employed. Each Robot Agent controls a *Pioneer 2* mobile Robot.

The Robot Agent must satisfy critical time restriction since the tasks that control the robot sensors and effectors have temporal constraints. Moreover, this agent periodically sends information about its situation and state to the Floor Agent. This information is used by the Floor Agent to select the most appropriate agent to send a new delivery/reception request to. The next section shows the Robot Agent architecture in detail.

4.1. Robot Agent Architecture

The Robot Agent (Figure 4) ensures that the physical robot completes its goals. To perform this function the Robot agent is composed of three modules:

- **Communication module**: this module is in charge of coding/decoding and sending/receiving the messages. The message is coded at this level using FIPA ACL as the communication language. This codification converts the message into a useful load for the transport message.
- **Navigation module**: this module is used for the robot to navigate around the physical environment and go towards the next objective. The navigation through the environment is carried out using the information obtained by the sensors and the execution of specific time-bounded tasks which are necessary to achieve a specific goal.
- **Temporal Constraint Analysis module**: by means of this module, the agent can decide if it can perform a specific service before a deadline is met and hence, commit itself to the execution of this service.

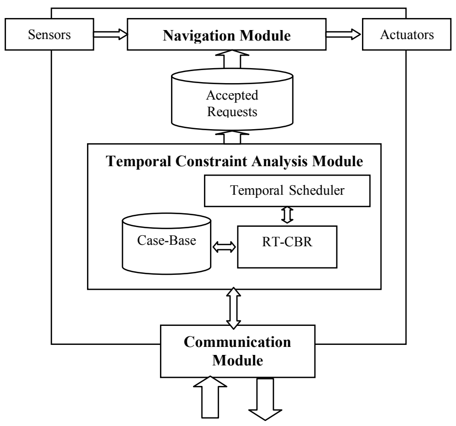

Figure 4: Internal architecture of the Robot Agent.

The main goal of the robot Agent is to collect a letter or package and move it to a location specified by the client. This goal is complex since, to complete the goal, the robot must navigate, dodge obstacles, and pick up/deliver the letter or package. In the robot agent architecture proposed here, the robot agent uses the navigation module to fulfill this main goal. Internally, this module executes the following tasks:

- **The Collect task**, which picks up/leaves the mail in the corresponding location.
- **The Navigate task**, which is in charge of managing the route across the environment until the final position where the mail is delivered.
- **The Sensor task**, which controls the robot's physical integrity of preventing the robot from crashing into obstacles. The execution of all of these tasks makes the navigation and mail delivery possible.

The problem arises when it is necessary to know the time needed to complete the goal exposed above. The number of times that you run any of the above tasks to deliver mail from point A to point B is unknown. The robot travels through a dynamic environment where the emergence of mobile obstacles or changes in delivery points, walls, etc. is more than probable. In this case, to determine the number of times that the robot agent finds an obstacle -and then dodges it- or to know the 'exact' and 'real' path that the robot agent will follow to achieve the goal is very complicated. Hence the difficulty in determining the necessary time to complete the requested goal lies in knowing the exact number of tasks -and their type- to complete the target. Since this information cannot be known a priori, it is necessary to make a rough estimation of the time to complete the assigned goal. Therefore, the Temporal Constraint Analysis module, which incorporates a Temporal-Bounded CBR following the proposed guidelines was implemented. This module is explained in the next section.

4.2. Temporal Constraint Analysis Module

The Temporal Constraint Analysis (TCA) module must decide if a Robot agent has enough time to perform a specific service. One possible way to perform such decision-making functionality is to use the knowledge that the robot agent has gained from previous analyses, undertaken in the past. It is assumed that a robot agent has enough time to perform a service if it has already succeeded in doing so in a similar situation.

To carry out the decision-making process regarding whether or not to contract a commitment to perform the service, the TCA module has been enhanced with a TB-CBR module. According to the KI task classification mentioned previously, this new TB-CBR module copes with a typical planning problem, where a path must be built as a temporal ordering of a set of activities. So, it is necessary to address the problem as a planning process where the final plan (solution) is produced by the adjustment of plans stored in the case-base.

The cases of the TB-CBR module are structured as follows:

$$C =< I, F, N_t, N_s, T > \tag{6}$$

where I and F represent the coordinates of a path from the initial position I to the final position F that the robot travelled (one or several N_t times) straight ahead in the past, N_s stands for the number of times that the robot successfully completed the path within the case-based estimated time and T shows the series of time values that the robot spent to cover that route. Note that only straight routes are stored as cases, since we assume that they are the quickest way between two points. This design decision should minimise the time needed to travel an unvisited route that the TB-CBR module would try to compose by reusing several known routes (old cases).

Therefore, the TB-CBR module estimates the duration of new paths by means of a function $t : T \rightarrow f(T)$ computed over the temporal values that last similar previous paths. The expected time T_s to perform a path that consists of a collection of known sub-paths is the aggregation of the estimated time for each one of these sub-paths:

$$T_s = \sum_{i=0}^{I} t_i \tag{7}$$

Finally, the series of observed execution times could also allow the TB-CBR module to estimate a *success probability* $P(T_s)$ for a request to be performed within a specified time. This is interesting data for agents, which could use this probability to make strategic decisions about their potential commitments. Setting a *confidence factor* (CF) that represents a minimum threshold for the success probability, agents would commit themselves to fulfilling a delivery/reception mail request if:

$$\exists T_s / P(T_s) \geq CF \wedge Ts \leq deadline \tag{8}$$

Thus, agents with riskier strategies could undertake commitments with lower confidence values than more cautious agents.

The TB-CBR reasoning cycle starts when the TCA module has to decide if an agent can fulfil a shipment service within the time assigned to do it, following the cycle described in section 3. Then, the selected case-base format and the operation of each reasoning phase of the module are shown.

4.2.1. Data format

A Hash table has been chosen as a case-base structure as mentioned in section 3.1.1. The *WHAT* tool [69] has been used in order to obtain the worst-case execution time of the different functions of the algorithm which involve access to the case-base. This tool has been adapted to be used in Real-Time Java language over SUSE Linux Enterprise Real Time 10 as a real-time operating system. The results of this analysis can be seen in Table 1.

The table shows the temporal behaviour of the different functions according to the number of cases stored in the case-base. The times obtained have been used in the temporal analysis of the different parts of the algorithm (in the *enoughTime* function, for example).

	analysisResult()		retainResult()		search()		adaptSolution()	
Asymptotic cost	O(1)		O(n)		O(n)		O(1)	
case-base size	$\bar{x}$	wcet	$\bar{x}$	wcet	$\bar{x}$	wcet	$\bar{x}$	wcet
5	<1	1	198.05	304.97	64.37	79.47	<1	1
10	<1	1	198.93	340.04	64.53	116.39	<1	1
15	<1	1	198.99	367.25	64.57	143.60	<1	1
20	<1	1	198.90	428.90	64.63	254.06	<1	1
25	<1	1	199.00	455.46	64.75	289.23	<1	1
30	<1	1	199.94	484.69	64.54	314.46	<1	1
35	<1	1	200.15	513.04	64.70	340.62	<1	1
40	<1	1	198.64	557.02	64.28	396.79	<1	1
45	<1	1	198.78	581.39	64.12	421.66	<1	1
50	<1	1	197.82	609.20	64.30	448.83	<1	1
55	<1	1	197.97	635.60	64.24	474.77	<1	1
60	<1	1	198.13	665.39	64.27	511.27	<1	1
65	<1	1	198.14	698.58	64.34	482.79	<1	1
70	<1	1	198.87	731.59	64.49	525.36	<1	1
75	<1	1	199.19	817.61	65.25	650.46	<1	1
80	<1	1	198.77	864.64	64.40	687.76	<1	1
85	<1	1	198.62	898.58	64.62	733.02	<1	1
90	<1	1	198.77	929.12	64.66	753.20	<1	1
95	<1	1	198.58	974.06	64.49	804.34	<1	1
100	<1	1	198.87	1025.61	64.50	847.98	<1	1

Table 1: Asymptotic and temporal costs analysis in the case of study (all times in nanoseconds)

4.2.2. Revision phase

As shown in the previous section, the TB-CBR starts with the Revision phase. Once the Robot Agent has finished the shipment service, it sends a report to the TCA module with the coordinates of each path that it finally travelled straight ahead on, and the time that it took to do so. The TCA stores this information until the revision phase is launched. Thus, the manager can check the performance of the TB-CBR module by comparing the time estimated by the module and the time that the robot finally took to complete the journey. In order to do this, the *analysisResult* function, defined in section 3, has been implemented.

Note that if we were in a static domain, the agent could try to perform the shipment by following the same route that ended successfully in the past. However, due to the fact that some new obstacles could be found along the route, the design decision to report the specific paths that the Robot Agent has travelled along has been taken.

Once the revision is completed, the next step is to decide if the case should be stored in the case-base.

4.2.3. Retention phase

The second step of the reasoning cycle considers the addition of new knowledge in the case-base of the TB-CBR module. As pointed out before, the size of the case-base must be controlled and therefore, only useful cases should be added (and correspondingly, out-of-date cases must be eliminated). Therefore, decisions regarding the addition of a new case in our model is crucial. In this example, the *retainResult* has been implemented to include a simple but effective procedure by defining a threshold α below which two points must be considered to be nearby in our shipment domain. Let us consider a new case c with coordinates (x_i^c, y_i^c) (initial point) and (x_f^c, y_f^c) (final point) to be added in the case-base. Following an *Euclidean* approach, the distance (dissimilarity) between case c and each case z of the case-base can be computed with:

$$dist(c,z) = max(\sqrt{(x_i^c - x_i^z)^2 + (y_i^c - y_i^z)^2}, \sqrt{(x_f^c - x_f^z)^2 + (y_f^c - y_f^z)^2}) \quad (9)$$

Therefore, the new case will be included in the case-base iff:

$$\forall z \in caseBase \;/\; dist(c,z) < \alpha \quad (10)$$

In this case, the new case $< (x_i^c, y_i^c),\ (x_f^c, y_f^c),\ 1,\ 1,\ time >$ will be added to the case-base ('1' values stand for this first time that the path has been successfully travelled along). Note that the addition of new cases is always conditioned to the existence of 'free space' in the case-base. Otherwise, a maintenance cycle will be triggered, deleting, for instance, those old cases that have low usage. If a similar case in the case base has been identified, the number of times that the agent has travelled the path that represents the case (N) will be increased by 1 and the time taken to travel the current path will be added to the time series of that case. Once the case-base is updated the delivery stage starts. Thus, the retrieval and reuses phases are executed.

4.2.4. Retrieval and Reuse phases

Due to the temporal constraints that the CBR process has, we have followed an anytime approach [29] in the design of the algorithm that implements the retrieval and reuse phases of the TB-CBR module. In our design, both phases are coupled in the algorithm, reusing the time estimations for several paths in order to retrieve the most suitable case(s) and routes along which to travel (the composition of cases that minimises the travelling time). At the end of each iteration, the algorithm provides the manager with the probability of it being able to perform the shipment service on time. If more time is still available, the algorithm computes better estimations on subsequent iterations.

Firstly, the TB-CBR module must adapt the problem to the case-base structure, implementing the *adaptProblem* function proposed in the general algorithm. After this, the TB-CBR module, by means of the *search* function, searches its case-base to retrieve a case that represents a similar path along which the Robot Agent travelled in the past. Then, for each retrieved case, the algorithm uses a *confidence function* to compute the probability of being able to travel from an initial point to a final point in an area without diverting the agent's direction. It is assumed that the probability of the shortest paths being affected by unpredictable circumstances which could deviate the agent from its route is lower and hence, they are preferred to longer ones. In the best case, there will be a case in the case-base that covers exactly, or very approximately, the same path along which the agent has to travel. Then, the time needed to perform the shipment can be estimated by using the time taken in the previous case. Otherwise, the route could be covered by aggregating a set of cases and estimating the global time by adding the time estimation for each sub-case. If the route can somehow be composed with the cases of the case-base, the following confidence function will be used:

$$f_{trust}(i,j) = 1 - \frac{dist_{ij}}{maxDist} * \frac{N_s}{N_t} \text{ where } dist_{ij} \leq maxDist \tag{11}$$

where $dist_{ij}$ is the distance travelled N_t times between the points $< i, j >$, N_s represents the number of times that the robot has travelled along the path within the case-based estimated time and *maxDist* specifies the maximum distance above which the agent is unlikely to reach its objective without finding obstacles.

In the worst case, the agent would never have travelled along a similar path and hence, cannot be composed with the cases stored in the case-base. If this is the case, a confidence function that takes into account the distance that separates both points will be used:

$$f_{trust}(i,j) = \begin{cases} 1 - \frac{dist_{ij}}{const_1} & \text{if } 0 \leq dist \leq dist_1 \\ 1 - const_2 * dist_{ij} & \text{if } dist_1 < dist \leq dist_2 \\ \frac{dist_{ij}}{dist_{ij}^2} & \text{if } dist_2 < dist \end{cases} \tag{12}$$

where *const1* and *const2* are normalisation parameters defined by the user, $dist_{ij}$ is the Euclidean distance between the initial and final points of the path $< i, j >$ and $dist_1$ and $dist_2$ are distance bounds that represent the thresholds that delimit near, medium and far distances from the initial point. This function computes a smoothed probability of the robot being able to travel along its path straight ahead. As the distance between the initial and the final point increases, the confidence in travelling without obstacles decreases.

Once the probability of reaching the robot's objective is computed for each case, the complete route with the maximum probability of success from the starting point to the final position must be selected. This route is composed using a selection function $F(n)$ (13), which follows an *A** heuristic search approach [70]. The function consists of two sub-functions: $g(n)$ (14) which computes the case-based confidence of travelling from the initial point to a certain point n and $h(n)$ (15) which computes the estimated confidence level of travelling from the point n to the final point (always better than the real confidence level). Finally, the function $T(n)$ (16) checks if the Robot Agent has enough time to complete the shipment service by travelling along this specific route. Otherwise, the algorithm prunes the route. The function consists of two sub-functions: *time(n)* (17) which computes the case-based time of travelling from the initial point to a certain point n and *E(n)* (18) which computes the estimated travel time from point n to the final point. In (17) $dist_{mn}$ represents the distance between the last point m visited by the algorithm and the current point n, V_{robot} is the speed of the robot, $f_{trust}(m,n)$ corresponds to (11) or (12) (depending on the possibility of composing the route by using the cases in the case-base) and the constant $const_{trust} \in [0, 10]$ shows the degree of caution of the robot agent. Bigger values of this constant stand for more cautious agents.

Finally, if the TB-CBR algorithm is able to compose the entire route with the information stored in the case-base, it returns the case-based probability of performing the shipment service on time. Otherwise, it returns the product of the probability accumulated to that moment and a pessimistic probability of travelling from the last point that could be reached, by using the cases in the case-base, to the final point of the route. Finally, in the event of all possible solutions computed by the algorithm exceeding the time assigned to fulfil the service, it returns a null probability of performing the service successfully. The *bestSolution* function is in charge of returning the solution obtained.

$$F(n) = g(n) * h(n) \tag{13}$$

$$g(n) = g(m) * f_{trust}(m,n) \tag{14}$$

$$h(n) = 1 - \frac{dist_{nf}}{maxDist} \text{ where } dist \leq maxDist \tag{15}$$

$$T(n) = time(n) + E(n) \tag{16}$$

$$time(n) = time(m) + \frac{dist_{mn}}{V_{robot}} + \frac{const_{trust}}{f_{trust}(m,n)} \tag{17}$$

$$E(n) = \frac{dist_{nf}}{V_{robot}} \tag{18}$$

The probability returned by the TB-CBR algorithm will be used to determine whether the agent can commit itself to performing the service, or whether it should reject the service. Each agent has a *confidence value*. If the returned probability is greater than or equal to the confidence value, then the service will be accepted for execution. This confidence value is different according to the behaviour of the agent. A cautious agent will have a high confidence value and thus, will only accept those services with a high probability of fulfilling the goal. On the other hand, a fearless agent will have a low confidence value.

4.3. Tests and Results

To develop and test the postman multi-agent system, we have used the jART platform [71] (which is specially designed for real-time multi-agent systems) and RT-Java [72] as the programming language. Once the example was implemented over the jART platform, several simulation experiments were conducted to evaluate different parameters in order to verify the use of the proposed module. A simulation prototype was implemented using a Pioneer 2 mobile robot simulation software (specifically, the Webots simulator [73]). The simulation experiments were conducted to evaluate different aspects and to try to show the benefits and correct behavior of the temporal-bounded CBR integrated on the TCA module in a robot agent. the following section describes the results of the experiments. The different experiments were tested on a system without the TCA module and on one that included the module.

In our prototype, the Floor Agent is informed of the arrival of new mail through an Interface Agent running on a PDA or mobile phone. The Floor Agent selects the most appropriate Robot Agent to perform the request. Later, the selected Robot Agent is informed of new mail orders. At this point, the Robot Agent must decide whether it accepts the mail order. If it accepts, the Robot Agent is committed to carrying out the request within the agreed time. If it does not accept, the Floor agent must find another Robot Agent that can carry out the request.

It is important to note that, in the case of the agent without the TCA module, the received mail orders are stored in a request queue. This queue only stores up to five pending requests. When the Robot Agent receives a request, if it has space in the queue, the Robot Agent accepts the commitment associated to the request. Otherwise the request is rejected. In each case, the mail or package distribution must be ended before a maximum time, and the robot control behaviour must guarantee the robot's integrity, which implies hard real-time constraints. In the tests carried out, all requests have the same priority. Therefore, using the number of requests managed by the Robot Agent is an adequate metric to verify the improvement made by the use of the TCA module. Thus, the greater the number of requests satisfied on time by the Robot Agent, the better the system performs. If the requests have different priorities, this metric is not correct. In this case, fulfilling tasks with high priority is more important than fulfilling a greater number of low priority tasks.

The first set of experiments investigates the request acceptance of the system according to the frequency of packages or mail reaching an agent without the temporal constraints analysis (TCA) module and other agents with the TCA module (for these agents, three tests modifying the confidence values from 70%, 80% and 90% are executed). The simulation prototype was tested by increasing this frequency incrementally and testing the number of unaccepted requests. The tests consisted of groups of 10 simulations with a duration of five minutes. The Floor Agent received between 5 and 30 requests during these five minutes. Each experiment was repeated one hundred times and the results show the average value obtained. The results are shown in Figure 5. This figure shows that, at a low request frequency, the Robot Agent accepts all requests and is committed to fulfilling them, independently of whether the Robot Agent incorporates the TCA module. Nevertheless, when the request frequency is increased, the agent with the module works better (independently of the confidence value), even with a high frequency rate. These results must be contrasted with the percentage of successfully completed requests. With respect to the agent without the module, it can be observed that, in the event of low average rates, the behaviour is slightly better at the beginning. The reason behind this is that the agent accepts all requests as long as its request queue is not full, without taking into account whether it will be able to successfully process these requests. The

Robot Agent that has the TCA module rejects requests sooner that the other one because it only accepts requests if it is able to successfully complete the work associated with that commitment.

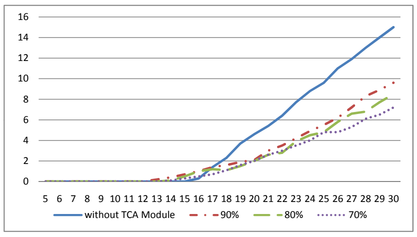

Figure 5: Analysis of the number of unaccepted requests

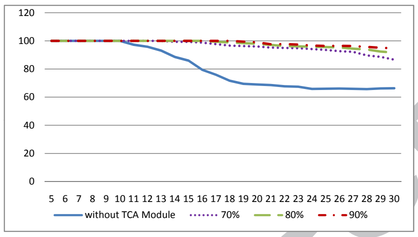

Figure 6: Percentage of successfully ended requests

The second set of experiments investigates the success rate of accepted commitments according to package or mail arrival frequency (Figure 6). This figure shows that using the TCA module with a confidence value of 90 % is very efficient in order to maintain the success rate close to 100%. In contrast, if the Robot Agent does not use the module, the success rate decreases as the number of requests increases. Even when the saturation of requests in the system was very high, the agent with the TCA module still had a success rate of approximately 90% independently of the confidence value. When the confidence value decreases the success rate is worse. The reason for this loss is the time consumed by the TCA module to analyse the requests and control the execution of the accepted request. At the beginning, this execution time is bounded, but when the request arrival frequency is very high, this time is overrun, which affects the execution of the rest of the agent's tasks. This will be improved in the next implementations of the TCA module.

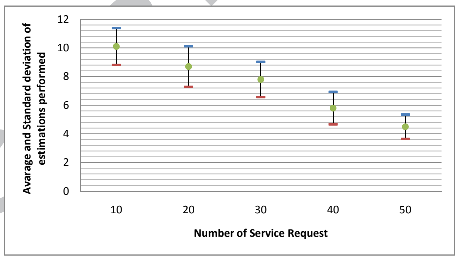

Figure 7: Average and standard deviation of the number of estimations performed vs number of service requests.

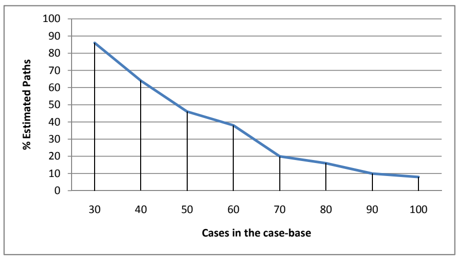

Figure 8: Percentage of estimated paths in a complete route vs number of cases in the case-base.

The next test analyses the behaviour of the TCA as it receives new requests by increasing the number of queries. As shown in Figure 7, the number of estimations that the TCA performs decreases as new requests are queried. This demonstrates that as the number of requests increases, the case-base learns

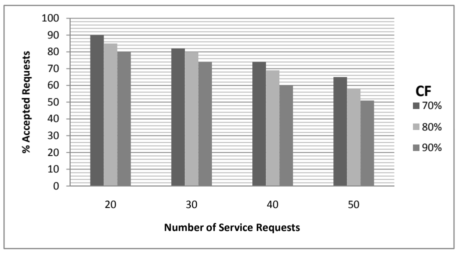

Figure 9: Percentage of accepted requests vs number of service requests.

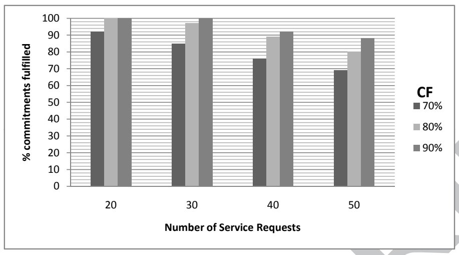

Figure 10: Percentage of commitments fulfilled vs number of service requests.

the new information properly and hence, the number of routes that can be composed with the cases increases (and an estimation is not necessary). Figure 8, which shows the relation between the number of cases in the case-base and the percentage of estimated routes, also supports this conclusion. Finally, the percentage of distrust from which an agent can commit itself to performing a service was also checked (modifying the confidence factor values from 70%, 80% and 90%). As expected, bigger confidence percentages resulted in agents committing themselves to performing more tasks (Figure 9). However, in such cases the percentage of services accepted and completed on time decreases, since the agent committed itself to the performance of a large amount of services (Figure 10). Logically, when the confidence factor increases the acceptation percentage is lower.

The results obtained in Figure 9 and Figure 10 have been merged in Figure 11, which shows the robot agent behavior for each confidence factor comparing accepted requests fulfilled versus those not fulfilled. From this figure, we can extract that if the agent selects a high CF, it obviously accepts fewer proposals but it can fulfill all of the proposals even with high request frequencies. On the other hand with a low CF the agent behavior results are completely opposed.

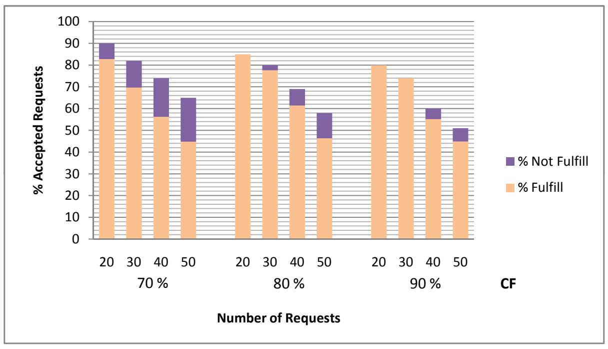

Figure 11: Analysis of the robot agent behavior for each confidence factor

In order to reaffirm this fact, we can see in Table 2 the average acceptance

	70%	80%	90%
accepts	77,75	73	65,25
errors	14,3525	5,3975	2,73
fulfill	63,3975	67,6025	62,52

Table 2: Average of the different obtained rates

rate of the different tests made and how much of this percentage has not been completed successfully. Analyzing the table, we can conclude that if we want to have a high percentage of requests successfully completed without taking the errors that were committed into account, we can see that when a Confidence factor of 80% is obtained nearly 67,6% of the requests were completed while only 5,39% of requests are served after their deadlines. With a confidence value of 90% the percentage of requests successfully completed is only 62%. Taking these results into account, the agent must dynamically vary its confidence factor. How it can be adapted depends on the context, mainly in the current load of the agent. With a high load the agent must increase its CF in order to avoid possible failures in its commitments. On the contrary, the agent can decrease its CF, and probably accept more requests, while the system load is not excessive.

5. Conclusions

This paper has shown the flexible and efficient integration of a high-level deliberation processes with real-time behaviours in complex and dynamic environments. The agent paradigm has changed the design and development processes of complex software systems. Nevertheless, the technologies employed in multi-agent systems must be adapted for their correct use in real-time environments. In accordance with this idea, this paper has proposed the integration of a deliberative capacity, based on bounded case-base reasoning techniques, into a real-time agent. More specifically, the work has proposed a new temporal-bounded CBR algorithm to be integrated as a deliberative capability inside a real-time agent architecture. This new approach takes into account the time available and consumed during the deliberative process. This temporal control allows an efficient execution time management.

A multi-agent scenario with temporal bounded deliberative and reactive processes has been implemented using the TB-CBR approach. Specifically, this approach has been tested in the automated management simulation of internal and external mail in a department, allowing it to decide if an agent can commit itself to performing delivery/collection services without exceeding the maximum time assigned for performing those services. The results are promising for deployment within a real scenario in the near future. The characteristics of the architecture presented in this paper make it very suitable for application in dynamic environments, in which learning and adaptation to constant changes is required and temporal boundaries exist. In this sense our future research work will consist of the application of the proposed approach in other environments, such as e-commerce, e-learning, entertainment, manufacturing processes, scheduling and control, etc.

Acknowledgment

This work is supported by TIN2006-14630-C03-01 projects of the Spanish government, GVPRE/2008/070 project, FEDER funds and CONSOLIDER-INGENIO 2010 under grant CSD2007-00022.

References

[1] M. Luck, P. McBurney, O. Shehory, S. Willmott, Agent Technology: Computing as Interaction (A Roadmap for Agent Based Computing), AgentLink, 2005.

[2] M. Luck, P. McBurney, Computing as interaction: Agent and agreement technologies, in: IEEE SMC Conference on Distributed Human-Machine Systems, 2008, pp. 1–6.

[3] J. Soler, V. Julian, M. Rebollo, C. Carrascosa, V. Botti, Towards a real-time multi-agent system architecture, in: Workshop: Challenges in Open Agent Systems, AgentCities, 2002, pp. 1–11.

[4] V. Julian, V. Botti, Developing real-time multi-agent systems, INTEGRATED COMPUTER-AIDED ENGINEERING 11 (2004) 135–149.

[5] A. Garvey, V. Lesser, A survey of research in deliberative real-time artificial intelligence, The Journal of Real-Time Systems 6 (1994) 317347.

[6] V. Botti, C. Carrascosa, V. Julian, J. Soler, Modelling agents in hard real-time environments, in: MAAMAW'99 Proceedings, Vol. 1647 of LNAI, Springer-Verlag, 1999, pp. 63–76.

[7] A. Martens, A. Uhrmacher, Adaptative tutoring processes and mental plans, Intelligent Tutoring Systems (2002) 71–80.

[8] A. Aamodt, E. Plaza, Case-based reasoning: foundational issues, methodological variations and system approaches, AI Communications 7 (1) (1994) 39–59.

[9] N. Karacapilidis, D. Papadias, Computer supported argumentation and collaborative decision-making: the hermes system, Information Systems 26 (4) (2001) 259–277.

[10] J. Corchado, A. Pellicer, Development of cbr-bdi agents, International Journal of Computer Science and Applications 2 (1) (2005) 25–32.

[11] M. V. N. Prasad, V. R. Lesser, S. Lander, Retrieval and reasoning in distributed case bases, Tech. rep., UMass Computer Science Report 95-27. CIIR Technical Report IC-5 (1995).

[12] R. Plaza, J. L. Arcos, F. Martín, Cooperative case-based reasoning, Distributed Artificial Intelligence meets Machine Learning LNAI 1221 (1997) 180–201.

[13] F. Martín, E. Plaza, J. L. Arcos, Knowledge and experience reuse through communication among competent (peer) agents, International Journal of Software Engineering and Knowledge Engineering 9 (3) (1998) 319–341.

[14] B. S. L. McGinty, Collaborative case-based reasoning: Applications in personalised route planing, in: 4th International Conference on Case-Based Reasoning: Case-Based Reasoning Research and Development, 2001, pp. 362–376.

[15] D. B. Leake, R. Sooriamurthi, When two case bases are better than one: Exploiting multiple case bases, case-based reasoning research and development, in: Fourth International Conference on Case-Based Reasoning (ICCBR-01), 2001.

[16] D. B. Leake, R. Sooriamurthi, Automatically selecting strategies for multi-case-base reasoning, in: International Florida Artificial Intelligence Research Society Conference (FLAIRS-02), 2002, pp. 106–110.

[17] L.-K. Soh, C. Tsatsoulis, Reflective negotiating agents for real-time multisensor target tracking, in: International Joint Conference on Artificial Intelligence (IJCAI-01), 2001, pp. 1121–1127.

[18] T. Bench-Capon, P. Dunne, Argumentation in artificial intelligence, Artificial Intelligence 171 (10-15) (2007) 619–938.

[19] L.-K. Soh, C. Tsatsoulis, A real-time negotiation model and a multi-agent sensor network implementation, Autonomous Agents and Multi-Agent Systems 11 (3) (2005) 215–271.

[20] P. Tolchinsky, S. Modgil, U. Cortés, M. Sánchez-Marré, Cbr and argument schemes for collaborative decision making, in: Conference on Computational Models of Argument, COMMA-06, Vol. 144, 2006, pp. 71–82.

[21] S. O. nón, E. Plaza, Learning and joint deliberation through argumentation in multi-agent systems, in: International Conference on Autonomous Agents and Multiagent Systems, 2007.

[22] C. Marling, M. Tomko, M. Gillen, D. Alexander, D. Chelberg, Case-based reasoning for planning and world modeling in the robocup small sized league, in: IJCAI Workshop on Issues in Designing Physical Agents for Dynamic Real-Time Environments, 2003.

[23] S. E. Fox, Behavior retrieval for robot control in a unified cbr hybrid planner, in: International Florida Artificial Intelligence Research Society Conference, 2001, pp. 98–102.

[24] M. Likhachev, M. Kaess, Z. Kira, R. Arkin, Spatio-temporal case-based reasoning for efficient reactive robot navigation, Tech. rep., Mobile Robot Laboratory, College of computing, Georgia Institute of Technology (2005).

[25] C. S. R. Ros, R. López de Mántaras, J. Arcos, A cbr system for autonomous robot navigation, in: Proceedings of CCIA-05, Frontiers in Artificial Intelligence and Applications, Vol. 131, 2005, pp. 299–306.

[26] R. Ros, R. L. de Mántaras, J. Arcos, M. Veloso, A case-based approach for coordinated action selection in robot soccer, Artificial Intelligence 173 (9-10) (2009) 11014–1039.

[27] D. J. Musliner, J. A. Hendler, A. K. Agrawala, E. H. Durfee, J. K. Strosnider, C. J. Paul, The challenges of real-time ai, IEEE Computer 28 (1995) 58–66.

[28] L. Hernndez, V. Botti, A. Garca-Fornes, A deliberative scheduling technique for a real-time agent architecture, Engineering Applications of Artificial Intelligence 19 (2006) 521–534.

[29] T. Dean, M. Boddy, An analysis of time-dependent planning, in: Proc. of the 7th National Conference on Artificial Intelligence, 1988, pp. 49–54.

[30] B. Hayes-Roth, R. Washington, D. Ash, A. Collinot, A. Vina, A. Seiver, Guardian: A prototype intensive-care monitoring agent., Artificial Intelligence in Medicine 4 (1992) 165185.

[31] A. E. Howe, D. M. Hart, P. R. Cohen, Addressing real-time constraints in the design of autonomous agents, The Journal of Real-Time Systems 2 (1990) 8197.

[32] D. Musliner, E. Durfee, K. Shin, Circa: a cooperative intelligent real-time control architecture, IEEE Transactions on Systems, Man and Cybernetics 6 (23).

[33] V. Julián, V. Botti, Developing real-time multi-agent systems, in: Proc. of the 4th Iberoamerican Workshop on Distributed Artificial Intelligence and Multi-Agent Systems, 2002, pp. 150–165.

[34] A. Garvey, V. Lesser, Design-to-time real-time scheduling, IEEE Transactions on Systems, Man and Cybernetics- Special Issue on Planning, Scheduling and Control 23 (6).

[35] T. Wagner, V. Lesser, Design-to-criteria scheduling: Real-time agent control, in: AAAI Spring Symposium on Real-Time Systems, Springer-Verlag, 2000, pp. 89–96.

[36] J. Graham, K. Decker, Towards a distributed, environment-centered agent framework, in: Intl. Workshop on Agent Theories, Architectures, and Languages (ATAL-99), 1999.

[37] D. M. Surka, M. C. Brito, C. G. Harvey, The real-time objectagent software architecture for distributed satellite systems, in: I. Proceedings (Ed.), Aerospace Conference, Vol. 6, 2001, pp. 2731–2741.

[38] L. C. DiPippo, E. Hodys, B. Thuraisingham, Towards a real-time agent architecture - a whitepaper, in: WORDS '99: Proceedings of the Fifth International Workshop on Object-Oriented Real-Time Dependable Systems, IEEE Computer Society, 1999, p. 59.

[39] L. C. DiPippo, V. Fay-Wolfe, L. Nair, E. Hodys, O. Uvarov, A real-time multi-agent system architecture for e-commerce applications, International Symposium on Autonomous Decentralized Systems 0 (2001) 357.

[40] OMG, Realtime corba, http://www.omg.org/docs/orbos/98-10-05.pdf.

[41] K. Prouskas, J. Pitt, Towards a real-time architecture for time-aware agents, in: AAMAS '02: Proceedings of the first international joint conference on Autonomous agents and multiagent systems, ACM, 2002, pp. 92–93.

[42] F. McCabe, K. Clark, April: Agent process interaction language, Intelligence Agents LNCS,890.

[43] P. Nii, Blackboard systems: The blackboard model of problem solving and the evolution of blackboard architecture, The AI Magazine (1986) 38–56.

[44] J. Bajo, V. Julian, J. Corchado, C. Carrascosa, Y. de Paz, V. Botti, J. de Paz, An execution time planner for the artis agent architecture, Engineering Applications of Artificial Intelligence In press (2008) 1–1.

[45] C. Carrascosa, J. Bajo, V. Julian, J. M. Corchado, V. Botti, Hybrid multi-agent architecture as a real-time problem-solving model, Expert Systems With Applications 34 (2008) 2–17.

[46] A. Newell, The knowledge level, Artificial Intelligence 18 (1982) 87–127.

[47] F. Hayes-Roth, D. Waterman, D. Lenat, Building Expert Systems, Addison-Wesley, New York, 1983.

[48] W. Clancey, Model constructions operators, Artificial Intelligence 53 (1) (1992) 1–115.

[49] A. Schreiber, J. Akkemans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. V. de Velde, B. Wielinga, Knowledge Engineering and Management: The CommonKADS Methodology, MIT Press, 2000.

[50] C. K. Riesbeck, R. C. Schank, Inside Case-Based Reasoning, Lawrence Erlbaum Associates, Cambridge, MA., 1989.

[51] F. Fdez-Riverola, J. M. Corchado, Fsfrt: Forecasting system for red tides, Applied Intelligence. Special Issue on Soft Computing in Case-Based Reasoning 21 (3) (2004) 251–264.

[52] H. Jo, I. Han, H. Lee, Bankruptcy prediction using case-based reasoning, neural networks, and discriminant analysis, Expert Systems with Applications 2 (3) (1997) 97–108.

[53] M. M. Richter, S. Wess, Similarity, uncertainty and case-based reasoning in patdex, Automated Reasoning - Essays in Honor of Woody Bledsoe.

[54] P. Cunningham, B. Smyth, A. Bonzano, An incremental retrieval mechanism for case-based electronic fault diagnosis, Knowledge-Based Systems 11 (1998) 239–248.

[55] A. Aamodt, Knowledge-intensive case-based reasoning in creek, in: Spinger (Ed.), 7th European Conference, ECCBR 2004, LNAI 3155, 2004.

[56] K. Ashley, Reasoning with cases and hypotheticals in hypo, International Journal of Man-Machine Studies 34 (1991) 753–796.

[57] R. L. S. J. L. Kolodner, The mediator: Analysis of an early case-based reasoner, Cognitive Science 13 (4) (1989) 507–549.

[58] K. Sycara, Resolving adversarial conflicts: An approach integrating case-based and analytic methods, Ph.D. thesis, School of Information and Computer Science (1987).

[59] V. Aleven, K. Ashley, Teaching case-based argumentation through a model and examples, empirical evaluation of an intelligent learning environment, in: 8th World Conference of the Artificial Intelligence in Education Society, 1997, pp. 87–94.

[60] T. Acorn, S. Walden, Smart: Support management automated reasoning technology for compaq customer service, in: A. Press (Ed.), Proc. ITS'92, Vol. 4, 1992, pp. 3–18.

[61] S. Heras, J. A. GarcíaPardo, R. Ramos-Garijo, A. Palomares, V. Botti, M. Rebollo, V. Julián, Multi-domain case-based module for customer support, Expert Systems with Applications 36 (3) (2009) 6866–6873.

[62] K. Sycara, R. Guttal, J. Koning, S. Narasimhan, D. Navinchandra, Cadet: a case-based synthesis tool for engineering design, International Journal of Expert Systems 4 (2).

[63] M. B. B. M. L. Maher, D. M. Zhang, Case-based reasoning in design, Lawrence Erlbaum, 1995.

[64] H. M.-A. M. T. Cox, R. Bergmann, Case-based planning, Knowledge Engineering Review 20 (3) (2006) 283–287.

[65] H. E. Tseng, C. C. Chang, S. Chang, Applying case-based reasoning for product configuration in mass customization environments, Expert Systems With Applications 19 (4) (2005) 913–925.

[66] D. W. Patterson, M. Galushka, N. Rooney, Characterisation of a novel indexing technique for case-based reasoning, Artificial Intelligence Review 23 (2005) 359–393.

[67] J. R. Quinlin, C4.5 Programs for Machine Learning, Morgan Kauffman, 1993.

[68] S. Wess, K. D. Althoff, M. Richter, Using k-d trees to improve the retrieval step in case-based reasoning, in: European Workshop, Topics in Case-based Reasoning, 1993, pp. 67–81.

[69] J. Palanca, A. García-Fornes, Uso de técnicas híbridas en el cálculo del wcet, in: VIII Jornadas de Tiempo Real, Universidad del Pais Vasco, 2005, pp. 243–249.

[70] P. E. Hart, N. Nilsson, B. Raphael, A formal basic for the heuristic determination of minimum cost paths, IEEE Transactions on SSC 4 (1968) 100–107.

[71] M. Navarro, V. Julián, J. Soler, V. Botti, jart: A real-time multi-agent platform with rt-java, in: 3rd IWPAAMS, 2004, pp. 73–82.

[72] T. R.-T. for Java Expert Group, Real-time specification for java, http://www.rtsj.org.

[73] Cyberbotics, Webots, http://www.cyberbotics.com/.