Reprinted from

DATA &
KNOWLEDGE
ENGINEERING

Data & Knowledge Engineering 15 (1995) 189–211

A temporal blackboard for a multi-agent environment[1]

V. Botti[a,*], F. Barber[a], A. Crespo[b], E. Onaindia[a], A. Garcia-Fornes[a], I. Ripoll[b], D. Gallardo[b], L. Hernández[a]

[a]*Dpto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, E-46022 Valencia, Spain*
[b]*Dpto. de Ingeniería de Sistemas, Computadores y Automática, Universidad Politécnica de Valencia, E-46022 Valencia, Spain*

Received 28 January 1994; revised manuscript received 9 January 1995; accepted 21 February 1995

N·H

ELSEVIER

DATA & KNOWLEDGE ENGINEERING

Aims & scope

Data & Knowledge Engineering (DKE) serves designers, managers, and users of database systems, expert systems, and knowledge-based systems. The major aim of the journal is to identify, investigate, and analyze the underlying principles in the design and effective use of these systems.

The DKE journal will be devoted to cross-fertilization of ideas and to stimulating interactions between workers in the database, knowledge engineering, and expert system areas. To achieve this aim, the journal will collect and disseminate original research results, technical advances, and news items on data engineering, knowledge engineering, or the intersection of these two fields.

The DKE journal welcomes original research papers in the areas of design, implementation, and applications of data/knowledge-based systems. The journal will emphasize the following topics:

1. *Representation and manipulation of data or knowledge:* Conceptual data models, knowledge representation techniques, Data/knowledge manipulation languages and techniques.
2. *Architecture of database, expert, or knowledge-based systems:* New architectures for database/knowledge base/expert systems, design and implementation techniques, languages and user interfaces, distributed architectures.
3. *Construction of data/knowledge bases:* Data/knowledge base design methodologies and tools, data/knowledge acquisition methods, integrity/security/maintenance issues.
4. *Applications, case studies, and management issues:* Data administration issues, knowledge engineering practice, office and engineering applications.

ⓧ The paper used in this publication meets the requirements of ANSI/NISO 239.48-1992 (Permanence of Paper).

Published 9 times a year 0169-023X/95/$09.50 Printed in The Netherlands

ELSEVIER
Data & Knowledge Engineering 15 (1995) 189–211
DATA & KNOWLEDGE ENGINEERING

A temporal blackboard for a multi-agent environment[1]

V. Botti[a,*], F. Barber[a], A. Crespo[b], E. Onaindia[a], A. Garcia-Fornes[a], I. Ripoll[b], D. Gallardo[b], L. Hernández[a]

[a]*Dpto. de Sistemas Informáticos y Computación, Universidad Politécnica de Valencia, E-46022 Valencia, Spain*
[b]*Dpto. de Ingeniería de Sistemas, Computadores y Automática, Universidad Politécnica de Valencia, E-46022 Valencia, Spain*

Received 28 January 1994; revised manuscript received 9 January 1995; accepted 21 February 1995

Abstract

The multi-agent system paradigm emerges as an interesting approach in the Knowledge Based System (KBS) field, when distributed problem-solving techniques are required for solving problems that can be represented as a collection of groups of cooperating intelligent individuals. A key concept in the multi-agent systems is the interaction between agents. On the other hand time plays a crucial role in a wide range of KBS applications. Temporal reasoning and representations consists of formalizing the notion of time and providing means to represent and reason about the temporal aspects of knowledge. This paper presents a framework for agent communication based on the blackboard paradigm which is able to manage temporal information, and it provides its multiple access and coherence management protocols.

Keywords: Blackboard; Knowledge-Based Systems; Real-time; Multi-agent; Temporal reasoning; Temporal Representation

---

1. Introduction

The multi-agent system paradigm emerges as an interesting approach in the Knowledge Based System (KBS) field, when distributed problem-solving techniques are required for solving problems that can be represented as a collection of groups of cooperating intelligent individuals. On the other hand, temporal representation and reasoning problems arise in a wide range of KBS application areas, where time plays a crucial role, such as in process control and monitoring, fault detection, diagnosis and causal explanation, resource manage-

---

* Corresponding author. Email: vbotti@dsic.upv.es

[1] This work has been funded by the ESPRIT-II Project of the European Community No. 5146 and 7805 (REAKT), by grant No. TAP-0511-C02 of the Spanish Government and by grant No. GV-1112/93 of the Generalitat Valenciana (Spain).

SSDI: 0169-023X(95)00007-0

ment and planning, etc. In these cases, temporal data representation and coherence management are needed in order to obtain conclusions about the problem.

A multi-agent world can be defined as a system in which several agents interact [9]. An agent is considered as a physical or abstract entity which is able to act on itself and on its environment in order to manipulate a partial representation of its environment and to communicate with other agents. The agent's behavior is a consequence of its perception, of its knowledge and of its interactions with other agents.

Two main directions have emerged in the research field related to the study of multi-agent worlds [8]:

- *Multi-expert system* paradigm or Distributed Artificial Intelligence (DAI): Agents, represented by a knowledge base or a specialized procedure, can be considered as specialized modules interacting together according to a specific architecture.
- *Robot acting in a multi-robot environment* paradigm: Agents can be considered as autonomous entities which have perception and communication capabilities, as well as decision abilities. They can act on their environment in an autonomous way.

A key concept in the multi-agent paradigm is the interaction between agents. This interaction cannot only be considered by single message exchange. Usually, communication occurs through sophisticated protocols for informing, requesting or convincing [7]. There are different levels of interaction between agents [7]:

- strong interaction between decision capabilities,
- medium interaction between reasoning capabilities, and
- weak interactions between perceiving capabilities.

This paper deals with the last kind of interaction between agents. The weakest interaction which can be found among several agents is to use the external world to exchange information. In fact, behavior can be described as deriving directly or not, from the knowledge an agent has about its environment. Sending information to an agent (communication case) or modifying the environment (world used as a blackboard) are two means by which cooperative behavior can be achieved among loosely coupled autonomous entities [21].

From the previous considerations, the need for computational models for providing a framework to communicate agents is inferred. In this sense, we can find two paradigms [7]:

- the actor paradigm, which is based on an object-oriented language where each agent has an independent life and communicates with others by sending messages,
- the blackboard paradigm, in which agents communicate by writing on a shared structure called a blackboard [4,5,10].

The work presented here is based on the second paradigm, and it provides a framework for sharing coherent information between agents. The blackboard is structured for organizing communications at various levels of abstraction, and an agent communicates with another one by writing on the blackboard. Those agents are activated (by the control component) when given patterns of information are present on the blackboard.

In this paper, a Temporal Blackboard framework for a multi-agent world, including concurrence control protocol, is described. The blackboard is a component in a real time expert system shell developed in the framework of the REAKT Esprit project [16].

2. Temporal information

Temporal information (Fig. 1) in dynamic systems is relevant in order to know the evolution of the system behavior and the time-stamped deduced values and to be able to reason with them (temporal constraints or dates for each data, trends, threshold crossing dates, average values in an interval, etc.) [1,15]. With a *reified* approach, problem temporal data are projected on a time line. It is assumed that a temporal value holds in a temporal interval defined by its extreme time points (I^-, I^+) [12]. From a temporal point of view, a temporal value can be:

- **exactly known**: the time at which the value has been taken and its duration are known (the ALARM was ON **from** 10:20:03 **till** 10:28:45).
- **partially known**: the value is the result of a sampling and the end time of the value is undefined (the TEMPERATURE **at** 10:23:22 was 120 degrees)
- **known with imprecision**: a variable value starts or finishes at some point within an interval (the KILN-STATUS will be HIGH **before** 20 minutes).

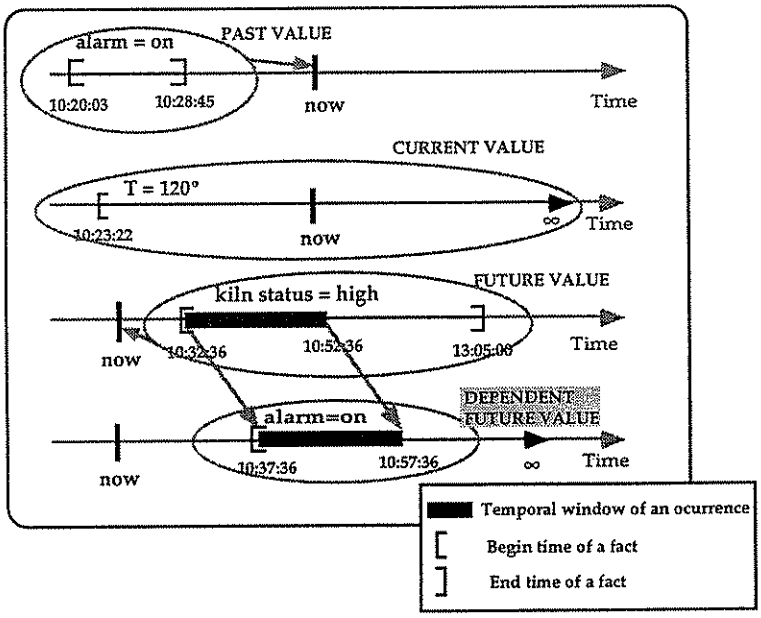

Fig. 1. Temporal information.

- **dependent**: the starting or ending time of a temporal value depends on other temporal facts (the ALARM will turn ON 5 minutes **after** the KILN-STATUS is HIGH)

Additionally, it is necessary to make predictions about process variables and reason with them to be able to take actions in advance to avoid future problems (predictive control). Consequently, the need for temporal information management requires handling the following kinds of information:

- **Past values** will usually be '*exactly known*' values related to the instant at which they were produced, so, a date can be used to store them.
- **Current values** are the values that hold at the present time and can be assumed as '*partially known*' and '*dependent*' values. They were asserted at known instants (i.e.: sampling period, its begin time depends on the begin time of other values), and, as no more information has been added, the same value could be maintained. A *persistence* attribute can be defined for this type of value. If the persistence time is exceeded, a lack of information is produced and no value is assumed. If there is no defined persistence, the last value is always maintained until the next incoming value.
- **Future values** are used to represent expected or predicted values. Their management presents more difficulties due to the lack of knowledge about the exact instant they will be produced. Thus, as a *temporally imprecise* value, it can be predicted that a KILN-STATUS will be HIGH at some instant **before** 20 minutes, but the concrete instant it will start or finish, or its duration is not known. In *advanced-future* reasoning, predictions can deduce other predictions. Thus, as a result of a causal prediction, *temporally dependent* values can also be predicted for instance, 'The ALARM will turn ON 5 minutes after the KILN-STATUS is HIGH'. This temporal dependency must be represented in order to reason about the future. When the causal prediction that originates the dependent predictions is fulfilled, these dependent facts becomes *temporally imprecise* facts. Thus, when reasoning in advance on present and future facts, rules can be fired with current or future facts, and actions on the right-hand side of a rule have to be executed at the instant (present or future) the values are deduced.

3. System architecture

In this section, the system architecture will be described. The system is built around a Temporal Blackboard for a Real-Time Knowledge Based Architecture, named **Knowledge Data Manager** (KDM). The KDM is integrated in the REAKT toolkit [16] a high-level environment for the development and production of Real-Time Knowledge Based Systems. The architecture of this temporal blackboard provides storage and management of application objects used by a set of *knowledge sources* (KS) running concurrently (Fig. 2).

The main components in the architecture are:

- ICM: The Intelligent Communication Manager is a task that provides the interface with

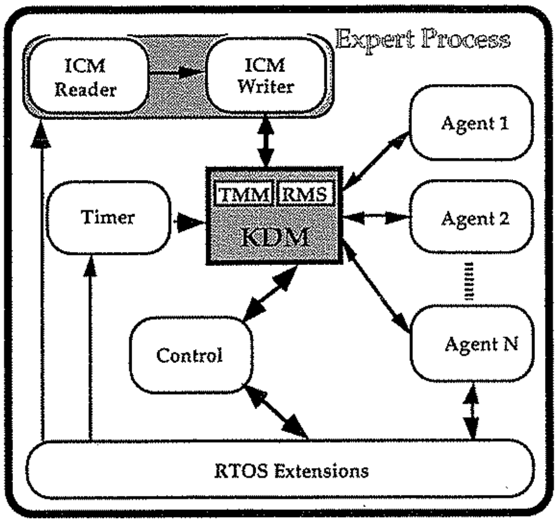

Fig. 2. REAKT global architecture.

external processes for sending and receiving data. ICM performs an intelligent filtering and stores values in the KDM.

- KDM: The Knowledge/Data Manager implements a Blackboard (shared memory) for temporal data. Temporal data are managed by means of a **Time Map Manager** (TMM) module that is in charge of maintaining temporal relations between temporal information. These temporal constraints are kept in a temporal graph: *Time-Map* (TM). The **Reason Maintenance System** (RMS) module is in charge of maintaining the logical coherence of temporal data. It deletes the deduced value when some premise is deleted or not fulfilled, and it modifies the temporal status (past, current, predicted) of deduced values according to the temporal status of premises [3].
- Agents are executable instances of Knowledge Sources (KS) which are the static compilation entities holding the domain knowledge. A KS in the REAKT architecture can be procedural or rule-based. In order to apply this approach to real time systems, each agent is structured as an independent task (integrated by the KS, the inference engine and the local memory or context, in the case of rule-based KS). Each agent is able to cooperate in a concrete reasoning process to obtain a global solution. Sequential execution of agents provides the global answer to a goal. The system can be seen as a collection of agents (reasoning modules) which share a common database (blackboard) and communicate with each other by signaling events.
- Control: this module receives significant events in the system and, in consequence, creates agents or sequences of cooperating agents (*intentions*) for solving a situation [13,14]. Several agents can run concurrently as a consequence of external or internal asynchronous event occurrences.
- Real-Time Operating System (RTOS) Extensions: this module provides system calls to create internal processes and to communicate and to synchronize them. Internal processes can be scheduled using user defined policies [6].

- Timer: It provides timing signals and message manipulation to execute actions at determined instants with respect to a real time clock.

All these components are organized as a process in order to concentrate all intelligent activities and to apply more complex scheduling policies to guarantee the best quality answer in the specified response time. This process is called **Expert Server** task and is implemented as a separate process on top of a UNIX operating system with real time extensions [2].

4. Blackboard structure

In this section, we will describe the Temporal Global Memory of a Real-Time Knowledge Based Architecture, named Knowledge Data Manager (KDM). The architecture of this temporal blackboard provides storage and management of application objects used by a set of agents reasoning concurrently.

The KDM data structure has an **object-oriented** approach (Fig. 3). Each object in the blackboard has a set of static and temporal slots. The structure of a temporal slot is formed by three set of pointers [past, current, future] to **Temporal-slot-value** structures.

The Temporal-Slot-Value class is defined by means of a *value* and its *begin* and *end* points are defined as the dates between where the value was, is, or will be taken. When the occurrence time of a temporal value is not precisely known, these fields contain a *tmnode* (objects that implement time points managed by the TMM); otherwise these slots will keep a real number as they stand for dates or absolute times. Each *tmnode* represents a time point and it has a set of temporal constraints with other ones.

There is a one-to-one mapping between temporal values and RMS nodes, the latter being used to compute the status of the associated temporal value. The value of the dependencies slot will be an instance of the rms-node class. This class is composed of two basic attributes: the *depends-on* slot will contain all the support-pointers that denote different derivations of the temporal value associated with this rms-node. Each support is basically constituted by a set of rms-nodes denoting that the belief of this rms-node depends on the belief of each rms-node in the support. The *support-to* slot of a rms-node will contain all the pointers to those rms-nodes whose belief is subject to the belief of such rms-node.

In the KDM architecture (Fig. 4), the **Time Map Manager (TMM) module** is in charge of maintaining temporal relations between temporal information: creation, deletion, and temporal relation management between temporal values. The **Reason Maintenance System (RMS) module** is in charge of handling the logical dependencies between temporal values, deleting each consequent of a temporal value when this one is deleted, it is also in charge of modifying the temporal status (past, current, predicted) of deduced values according to the temporal status of premises (this change in the temporal status of deduced values can also be caused by the incoming of new data or by the deletion of an outdated prediction in the TMM). Both TMM and RMS components allow for managing logical dependencies and the coherence of temporal values.

TMM only keeps future values, considering an end point of a current value as future data

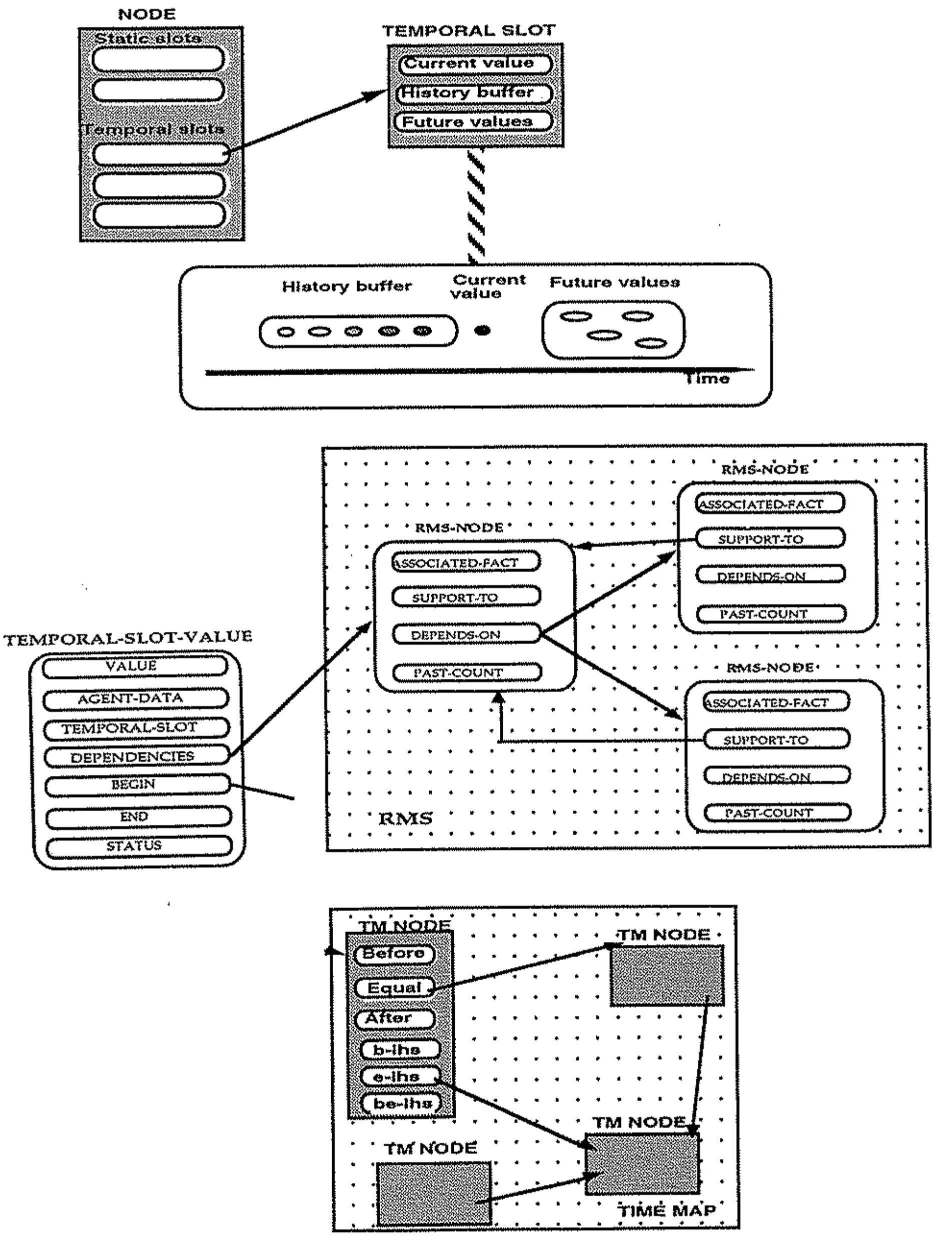

Fig. 3. KDM internal data structure.

until the current value becomes a past value. Hence, a time point with a precise time can be deleted from the TMM after propagating its consequences.

Initial data for each agent creates its initial **local-kdm**, by a reference-link from **global-kdm**. A local-kdm is a set of local dates which are independent from other local-kdm's. The local-kdm structure is the same as for a global-kdm and all KDM operations have the same behavior when they are to be applied on a different temporal data structure. Each agent works

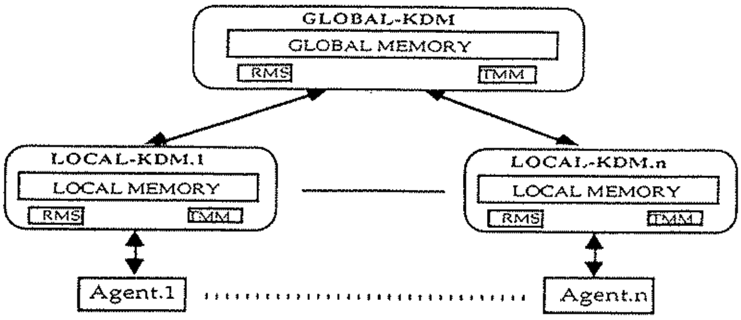

Fig. 4. Temporal blackboard.

on its local-kdm, where it writes its intermediate data. At the end of the agent execution, global data in the local-kdm are committed to the global-kdm. Thus, this global-kdm stores the shared data for all agents.

4.1. Blackboard interface

Blackboard interface provides a set of functions that can be used by a user language to express actions, such as class definition, object instances, and temporal updates and queries. Some of these functions (query functions) can be used as rule predicates in the *left hand side* (LHS) of rules (queries in procedural agents), and others can be called from the *right hand side* (RHS) of rules (actions in procedural agents) updating the internal information.

With respect to the **Left-Hand Side** temporal functions provided by the blackboard, they can be stated as:

- **when (object slot value begin end)**: This predicate is satisfied for those temporal values (current or predicted) which, belonging to the temporal slot, match the value. The begin and end of the temporal value may be instanced in variables. They can be either a date value or a time point.
- **past-value (object slot value begin end)**: This predicate is satisfied when a current value in the defined slot is outdated and is moved from current to past.
- **temporal-test (any temporal expression)**: this function is used for asking about temporal relations between: i) dates, beginnings or endings of temporal values or ii) dates or time points obtained from the begin or end of a temporal value and the current time (NOW).The temporal-test management is achieved by using the TMM internal functions to inform at the moment that the temporal-test becomes true, by changing their temporal windows.

Temporal intersection is a concept related to the validity for rule application. When a future value created in the KDM depends on other predictions, it is necessary to primarily ensure the simultaneous completion of these temporal dependencies in order to infer the prediction in the conclusion. The temporal window defined by the temporal-intersection between the values matching the LHS premises is called 'LHS-TIME' and it is bounded by two time-points:

[begin-lhs-time (lhs-BT), end-lhs-time (lhs-ET)]. These time-points are managed by the TMM module.

Blackboard functions on the **Right-Hand Side** of the rule modify the internal state by adding values and predictions about slot values of nodes:

- **predict-value (object slot value begin end dependencies)**: Adds a predicted value to a temporal slot. The user specifies the begin (lhs-BT by default) and end date for the value, by writing temporal restrictions expressing 'before', 'after' or 'at' a time point, and the dependencies of the predicted values (called premises in what follows), which, in the case of rule-based agents, are, by default, the values matching the lhs of the rule.

The status of the created value is always predicted. It may change according to the following specification:

- to **current**: a predicted value created with this function is changed to current when the prediction is fulfilled. This can only occur when a new value (that matches the prediction) arrives by means of a put-value (see function definition below).
- **deleted**: a predicted value is deleted in two cases: i) when the upper time f its begin temporal window arrives and the prediction has not been fulfilled and ii) when one (or more) of the predicted values in **dependencies** is deleted.

The following meaning for the predicted values is assumed: a prediction is **matched** by a new (predicted) temporal value when it has the same value and the begin and the end of the new temporal value are respectively within the temporal windows of the begin and end points of the prediction. Then no new prediction is created and the temporal window of the existing matched prediction will be constrained in accordance with the new prediction and will also update its prediction dependencies. In this case, pending queries can be updated. A prediction is **fulfilled** by a new (current) temporal value when it has the same value and the begin and the end of the new temporal value (current) are respectively within the temporal windows of the begin and end of the prediction. If there is some prediction about the slot which is fulfilled by the put-value (current value), then a new current temporal value is not created. Then the status of the fulfilled prediction value is changed to current.

- **put-value (object slot value begin end dependencies)**: This function sets a value to a slot. The user specifies the begin (lhs-BT by default) and end date for the value, by writing temporal restrictions expressing 'before', 'after' or 'at' a time point, and the dependencies of the predicted values, which, in the case of rule-based agents, are, by default, the values matching the lhs of the rule. If the slot is a temporal slot, then the function is used to create a current or a predicted temporal value (dependent-prediction) according to the values in **dependencies** (premises):

  - if there is at least one predicted temporal value in **dependencies**, then the asserted value will be **predicted**.

- if all the values in **dependencies** are current or past temporal values (at least one current) then the asserted value will be **current**.
- if all the values in **dependencies** are past temporal values then the asserted value will be past (it is possible to decide not to assert past values).

In the former case, when all the predicted values supporting the asserted value (premises) change to current, then the created value also changes to current. And if some predicted premise is not fulfilled, then the asserted predicted value will be deleted.

When a current value is asserted in a slot, if there is a previous current value in it and no contradiction is detected, then this current value is moved to its historical buffer of past values. If a contradiction is detected, a message for the control is sent. The end time of this previous current value is modified with the date corresponding to the begin time of the new created current value.

The KDM informs other components in the system when new current, past or prediction values are created or predictions are fulfilled or deleted. Each created prediction is a local one until it becomes global in the commit-step. Predictions can be created starting from local or global values (either current or predictions). When only local predictions are involved in the creation of a new one, all links between rule LHS and RHS are local, and thus stored in the same data structure. But if any global prediction appears in the premises of a predict-slot-value or put-slot-value functions, then it is necessary to hold a special link between the global and local prediction. By means of this special link between local and global predictions, global predictions can be matched locally. This avoids having to redo the reasoning process with the new value, which matches the prediction, but it may create an inconsistency between the global and local memory. A global value can match the global prediction as well as the local one. A local value can match local predictions, but such a value would not be communicated to the KDM until the commit-step.

5. Multiple access and coherence management

In this section the multiple access and coherence management protocol implemented in the REAKT architecture is described. The proposed protocol must take into account the expected behavior of agents:

1. Agents execute a Knowledge Source
2. For an agent to be executed, CONTROL must do the following sequential operations
   a) create the agent (or KS instance)
   b) read data (resources) from the KDM
   c) reason about them using a local memory
   d) write results into the KDM

3. When a deadline for an agent is reached, CONTROL kills the agent. All results which have not yet been written in the KDM are lost.

4. Several agents are executed simultaneously.

One problem related to the information coherence arises when several agents must be executed **concurrently** [11]. The KDM has to serve queries (transactions) from agents to read or write data. In the REAKT architecture, these transactions have timing and consistency constraints that must be accomplished. In order to provide response for queries and updates in the available time while maintaining the consistency of data, real-time concurrence control should involve efficient integration of ideas from both database concurrence control and real-time scheduling [18–20].

Several concurrence control protocols have been proposed using either pessimistic or optimistic approaches. In the pessimistic approaches, based on the two-phase locking protocol [17], problems are anticipated tending to delay transactions with data conflicts in order to avoid them later. These protocols allow for multiple reads and writes of different objects, but several locks in the same data object are not possible.

Optimistic approaches are based on the assumption that *nothing will go wrong*. When the transaction is ready to commit, conflicts are checked for and a resolution scheme is then applied to solve the conflicts. To resolve the conflicts, this approach uses a drastic solution: it aborts the transaction. A general approach is to utilize existing concurrence control protocols, such as the two-phase locking, and to apply time-critical transaction scheduling methods that favor more urgent transactions. However, some problems can be detected when using these protocols as a result of two operations: blocking and roll-backs of transactions. Both are barriers to meeting time-critical schedules.

5.1. Multiple access conflict resolution

The assumed environment is based on the KDM with agents sending queries (transactions). Each transaction has an initial **priority** and a **start-time-stamp**. As the result of work with temporal values, the time-stamp of deduced facts is part of the data. Each data value has its begin and end time associated.

The execution of each transaction is based on the Two-Phase Locking protocol with three different phases: the **read phase**, the **wait phase** and the **write phase**. During the **read phase**, a transaction can read from the database and write to its own local memory. This write operation will be referred as a **prewrite operation**. Before the transaction may perform a reading, it must obtain the **read-lock** on the data object it is interested in. To perform a write (prewrite) operation in its local memory, the transaction also has to obtain the **write-lock** on the data object. The read and prewrite operations can be done at any moment during the read phase.

After the transaction is completed, previous to writing the results in the database, it has to wait in the wait phase. When its turn has been granted, the process shifts to the write phase and downloads all the prewrite objects into the database.

The protocol with priorities is based on the principle that higher priority transactions should be completed before lower ones. If two transactions conflict, the higher priority transaction should precede the lower priority transaction.

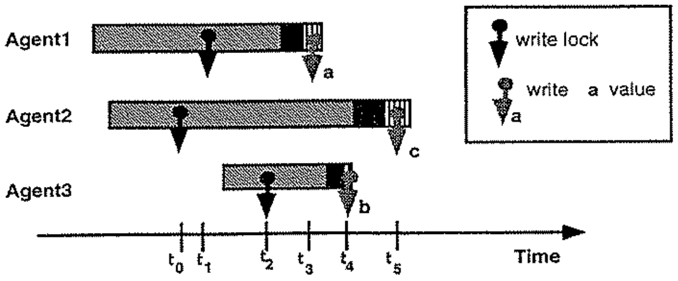

Fig. 5. Write lock example.

Depending on the priority and the order of the transactions, several possible conflicts can be found when different agents access to the same data (slot).

- Several read operations: all of them are granted.
- Several write operations: all of them are granted. The deduction time of each agent determines the final temporal ordering. Fig. 5 shows how three different tasks (agents) want to write on the same data and ask for a write lock that is granted by the KDM. Assuming t0, t1 and t2, as the begin time of the value, each write is performed in the local memory. In the write phase, values are written into the global memory. Using temporal attributes of data, the KDM maintains the coherence of deductions. The value evolution is shown in Fig. 6.
- A read operation before a write operation of a higher priority agent: in this case the less important task is aborted if it is in the execution (read phase). If it is in the wait phase, the write lock is delayed until it finishes.
- A write operation before a read operation of a higher priority agent: in this case the less important task is adjusted (wait phase) to ensure the coherence during the higher priority agent execution.
- Other cases in the serialization (first higher agents and then lower ones) are not considered due to the scheduling policy and the assumption about non input/output blocking of agents.

5.2. KDM-agent communication

Rule based agents and real-time tasks work with important differences with respect to resource management. In this case, REAKT agents work with first order rules, so they require reading all instances they are interested in. A large amount of data should be locked in the

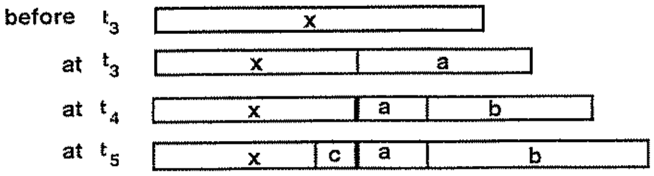

Fig. 6. Evolution of slot values.

agent initialization. But not all these data will satisfy rules, only a small number of them (optimistic approach) will do. Indeed, only current values that match LHS of a rule should be locked. Therefore, it is convenient to provide two kinds of read operations:

1. *Without lock*: it is used in the agent initialization; KDM sends all required slot values to the interested agents. When a value is updated in the KDM, a message should be sent to all interested agents which have not applied a locking on this slot.
2. *With lock*: the concrete slot value is involved in a rule that has already been accomplished. Before firing the rule, the agent must inform the KDM of its interest in a read lock on that slot.

The system reactivity with respect to an agent decreases as long as the agent is granted locking operations on the slots. Resources should be locked when they are needed. A rule requires that the system lock a value in either of these two cases:

1. a single pattern in the LHS of a rule is satisfied. When this occurs it means there exists a matching of the pattern to a temporal value (either acquired from the external world or deduced by the application). In this case the temporal value matching the pattern should be locked.
2. all the patterns in the LHS of a rule are satisfied, thus applying one lock of all instanced slots.

The second choice seems more adequate because only required slots are locked in a minimum of time. Since all nodes to be modified are already known in a rule, it would be possible to perform a write lock at the same time as a read lock. A description of the relation and queries between an agent and the KDM is shown in Fig. 7.

At the initialization, each agent receives the values of all the slots it is interested in. As soon as a slot value is updated into the KDM and no read lock has been requested by the agent, it is sent to all agents in the same condition. When a LHS of a rule is satisfied, slot values in the

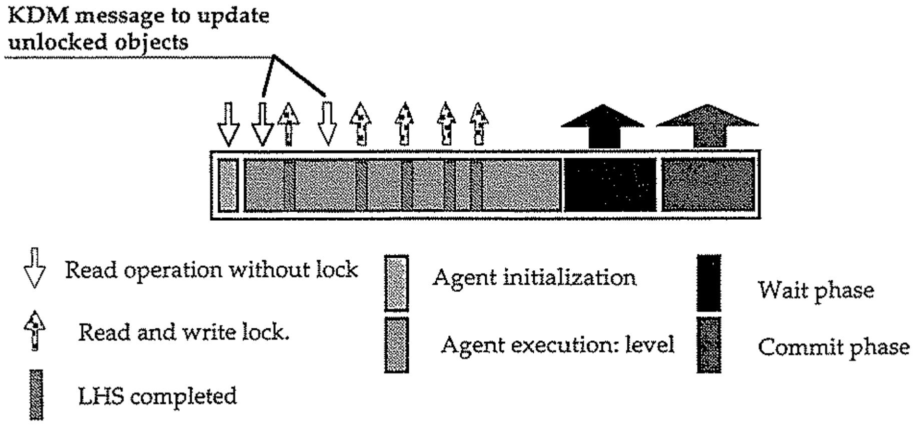

Fig. 7. KDM/agent communication.

LHS are read-locked while slots in the right hand side are write-locked. At the end of the commit phase all locks are released.

6. Example application: The tank scenario

The proposed scenario (Fig. 8) is an academic example application to illustrate the functionalities of the REAKT toolkit. Many of the physical aspects have been obviated in order to get a comprehensible scenario and focus on the relevant issues explained in the paper.

- The scenario is composed of two tanks connected through a valve. Initially both tanks are empty. Process starts by filling Tank1, warming up the water and filtering the liquid to Tank2 which in turn will supply the treated water to a chemical process. A kiln associated to Tank1 is in charge of warming up the water.

  Water is provided to Tank1 whose maximum capacity is 500 liters. The kiln is permanently working at a constant temperature but water temperature varies as long as new input of water is poured in Tank1. There is a sensor attached to the connect-valve to find out the time at which 250 liters of water at approximately 100° can be found. Notice that the temperature of the water in the tank gets colder when new streams are entered in Tank1 through the inp-valve.

  The connect-valve is programmed to open during two time units (from here on, time unit = sec) in order to let the 250 liters pass to Tank2. Once the water is in Tank2 the out-valve will flow 200 liters to the chemical process; volume in Tank2 must never drop below 50 liters.

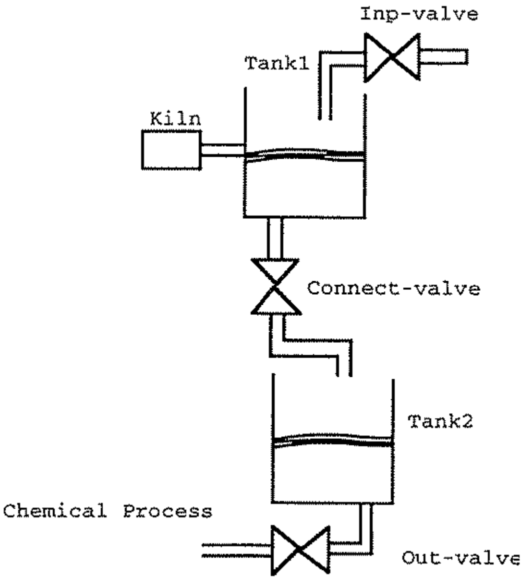

Fig. 8. The tank scenario.

The kiln takes around 2 time units to warm up the first 100 liters poured in Tank1. Since the following inputs of water will be mixed with warm water, the kiln will take different times to warm up the whole contents of the tank. This will depend on the amount and the temperature of the water contained in the tank, the flow of water falling in Tank1 etc.

- The normal behavior of the process is as follows:
  - Initially both tanks are empty
  - Open inp-valve to fill Tank1 with water
  - Tank1 begins filling with 500 liters (4 sec)
  - At the same time, the kiln begins warming up the water at ≈100° (6 sec: this duration comprises the 4 sec of filling the Tank)
  - Close inp-valve when water flow has stopped
  - Let the kiln warm up the whole volume of water (up to 6 sec for 500 liters).
  - Open connect-valve when there are 250 l. at ≈100°. In case water is still being poured in Tank1, close inp-valve before opening connect-valve
  - Emptying Tank1 and Filling Tank2 (250 l. approx. in 2 sec)
  - Close connect-valve
  - Open inp-valve to fill Tank1 again with water (Tank1 fills up to 500 liters: input flow ≈250 l. ≈2 sec)
  - At the same time open out-valve (supply water from Tank2 to the chemical process)
  - Kiln warms up the water. (6 sec for 500 l.)
  - Open connect-valve when there are 250 l. at ≈100°
  - . . .
- For the sake of simplicity the example application has been simplified and some details have been omitted. In a real process we can also find control loops for the overflow in Tank1 and Tank2 as well as a control procedure to ensure the kiln temperature always ranges over the correct values.
- We are now given the following situation:
  - Tank1 was initially filled with 500 liters and everything proceed as stated above.
  - Second time, volume in Tank1 did not reach 500 liters but only 400 liters (the water flow input was not the expected one). The kiln took 4 seconds to warm up the water. The connect-valve worked properly and again 250 liters were poured in Tank2 in 2 sec.
  - Next time, input flow in Tank1 has also decreased in 100 liters. The KDM registers the different values the water-level in Tank1 has taken on at different times. From this knowledge we can infer how the volume in Tank1 will evolve. The goal is to predict and advance the next and following states in the process in order to propose actions in case a fault is detected in the process (either volume in Tank1 does not reach 250 l. or volume in Tank2 drops below 50 l.).
- Input variables in the process are read through sensors and they stand for the data acquired from the external world. These values are then communicated to the KDM. In our example these variables are:

- **water-level in Tank1**. The KDM is informed about the level in Tank1 at each time the inp-valve is closed, that is:
  - at the time Tank1 has reached 500 l. if everything evolves normally or
  - when the water flow has stopped or

  Under normal conditions, the KDM will receive this information within 4 sec the first time and every 6 sec for the successive times.
- **water-level in Tank2**. The KDM is informed about the level in Tank2 at each time the connect-valve is closed. This means that the expected volume in Tank2 will be 250 l. every 8 sec under normal conditions.
- **water-temperature in Tank1**. The water temperature varies as long as water is introduced in the tank. The KDM is informed about the temperature at the same time the level is read and when the required quantity of water has reached 100° in order to open the connect-valve.

- Notice the difference between the observable values (water-level and temperature) and non-observable values (connect-valve opening, closing ...). The latter are directly governed by the controlling system (i.e. the REAKT application) whereas the behavior of the observable values is given by the controlled system (e.g. the industrial plant; in our case the scenario composed of the two tanks, the valves, the kiln etc.)
- Following we show the most relevant slots in each of the objects in the application. For the sake of simplicity the kiln is not represented as an object. Our choice is to state the kiln task (i.e. warming up the water) as a slot named 'water-temperature' in the class TANK.

CLASS TANK:
Tank1, Tank2:
- water-level {temporal slot which takes on a numeric value}
- water-temperature {temporal slot which takes on a numeric value}
- associated-alarm {all tanks have an associated alarm. This static slot keeps the alarm identifier}
- name {static slot for the name of the tank}

CLASS ALARM:
Alarm-Tank2:
- status: {Values for this temporal slot are 'ON', 'OFF'}

CLASS VALVE:
Connect-valve:
- status {temporal slot with values 'OPEN', 'CLOSED'}
- input-tank {static slot to store the input tank identifier #Tank1}
- output tank {static slot to store the output tank identifier #Tank2}

There exists two agents: the first one performs the control of TANK1 (water-level, kiln temperature, process of warming up the water) and second's agent task is to ensure the optimal water level in TANK2. First agent is only interested in data about Tank1 and Agent2 is interested in both Tank1 and Tank2 as many of the deductions for Tank2 depend on knowledge about Tank1.

Rules for the first agent are:

R1: R1 shows the process behavior under normal conditions. Once Tank1 contains at least 250 l. at 100° the connect-valve is open and a prediction on the volume of Tank2 is done. The prediction is used to confirm that everything worked properly: the connect-valve was open for two seconds and 250 l. of water were poured in Tank2.

```
(((in-class ?Tk TANK
     (when (?Tk water-level ?wat-lev))    ;; temporal slot
     (>=?wat-level 250)                   ;; condition on the slot value
     (when (?Tk temperature ?temp))       ;; temporal slot
     (>=?temp 100°))                      ;; condition on the slot value
  (in-class ?Vl VALVE
     (input-tank ?Tk-inp)
     (output-tank ?Tk-out)
     (=?Tk ?Tk-inp)))
→
(put-value ?V1 status OPEN)
(predict-value ?Tk-out water-level 250 (begin 1 3))
```

R2: R2 verifies a possible malfunction in the system. To do that, R2 makes a comparison between the two last past temporal values and the current value of the water-level in Tank1. If the level has been decreasing at each time (i.e. there exists a total order relation between the three values), the KDM performs a prediction of the volume in Tank1 at the next instant of time. Function 'evolution' returns the estimated value. The temporal interval is set by the user taking into account that Tank1 will empty 250 l. within a short period of time.

```
((in-class ?Tk TANK
     (name TANK1)
     (past-value (?Tk water-level ?wat-lev1 ?beg1 ?end1))
     (past-value (?Tk water-level ?wat-lev2 ?beg2 ?end2))
     (when (?Tk water-level ?wat-lev3))
     (>=?wat-lev3 250)
     (temporal-test (?beg2 after ?beg1))
     (<?wat-lev1 ?wat-lev2 ?wat-lev3))
→
(predict-value ?Tk water-level (evolution ?wat-lev1 ?wat-lev2 ?wat-lev3)
        (begin 4 6)(end 10))
```

R3: R3 makes the valve to close when a volume lower than 250 liters is detected in Tank1. Since the new level for Tank1 is still a prediction, the action of firing the rule is a dependent prediction, i.e. as soon as the prediction is confirmed the valve will be closed. The valve may happen to be already closed when the LHS prediction is fulfilled. But in case the valve were open it would be closed immediately. R3 determines the closing of the connect-valve under abnormal conditions.

```
(((in-class ?Tk TANK
     (when (?Tk water-level ?wat-lev))
     (<?wat-lev 250)
     (name TANK1))
  (in-class ?V1 VALVE
     (input-tank ?Tk-inp)
     (=?Tk ?Tk-inp)))
→
(put-value ?V1 status CLOSED)
```

Second agent is in charge of following the evolution of Tank2. Since the behavior of Tank2 is linked to the evolution of Tank1, Agent2 will be interested in data for both tanks.

R4: The same as for R1, R4 states the facts that should happen in the application under normal conditions. When Tank2 is filled with 250 l. the connect-valve is closed. This implies that both the inp-valve and out-valve will open to start the process again (the latter statements are omitted; only the connect-valve is managed in the example). R4 stands for the normal closing of the connect-valve.

```
(((in-class ?Tk TANK
     (name TANK2)
     (when (?Tk water-level ?wat-lev))
     (>=?wat-lev 250))
  (in-class ?V1 VALVE
     (output-tank ?Tk-out)
     (=?Tk ?Tk-out)))
→
(put-value ?V1 status CLOSED)
```

R5: R5 checks the water-level in TANK2. In order to control the level never drops below 50°C. in Tank2, Agent2 performs a checking on those predictions that point out a possible fall in the volume of Tank1. If the volume in Tank1 is predicted not to reach the necessary 250 l. at any point and that value is estimated to last 5 or more sec, Tank2 will run out of water very rapidly. Consequently, the alarm associated to Tank2 is planned to be fired as soon as the system is informed about such a possibility.

```
(in-class ?Tk2 TANK
     (name ?Tk2 TANK2)
     (associated-alarm ?A1)
     (in- class ?Tk1 TANK
```

ıe valve were
t-valve under

```
      (name ?Tk1 TANK1)
      (when (?Tk1 water-level ?wat-lev ?beg ?end))
      (<?wat-lev 250)
      (temporal-test (?end after ?begin 5)))
→
   (put-value ?A1 status ON)
```

- Fig. 9 shows the process behavior. The starting point is the normal expected evolution in the system. Tank1 fills with 500 liters and the inp-valve is closed at that moment. The KDM is informed about the current level and temperature in Tank1. After 6 sec have elapsed, the connect-valve is open as the sensor attached to the valve detects the water has reached the correct temperature. At this time the KDM is informed about the new water temperature.

ıvior of Tank2
ı tanks.

lication under
l. This implies
:ter statements
'or the normal

ɜr drops below
t a possible fall
ıecessary 250 l.
ıt of water very
as soon as the

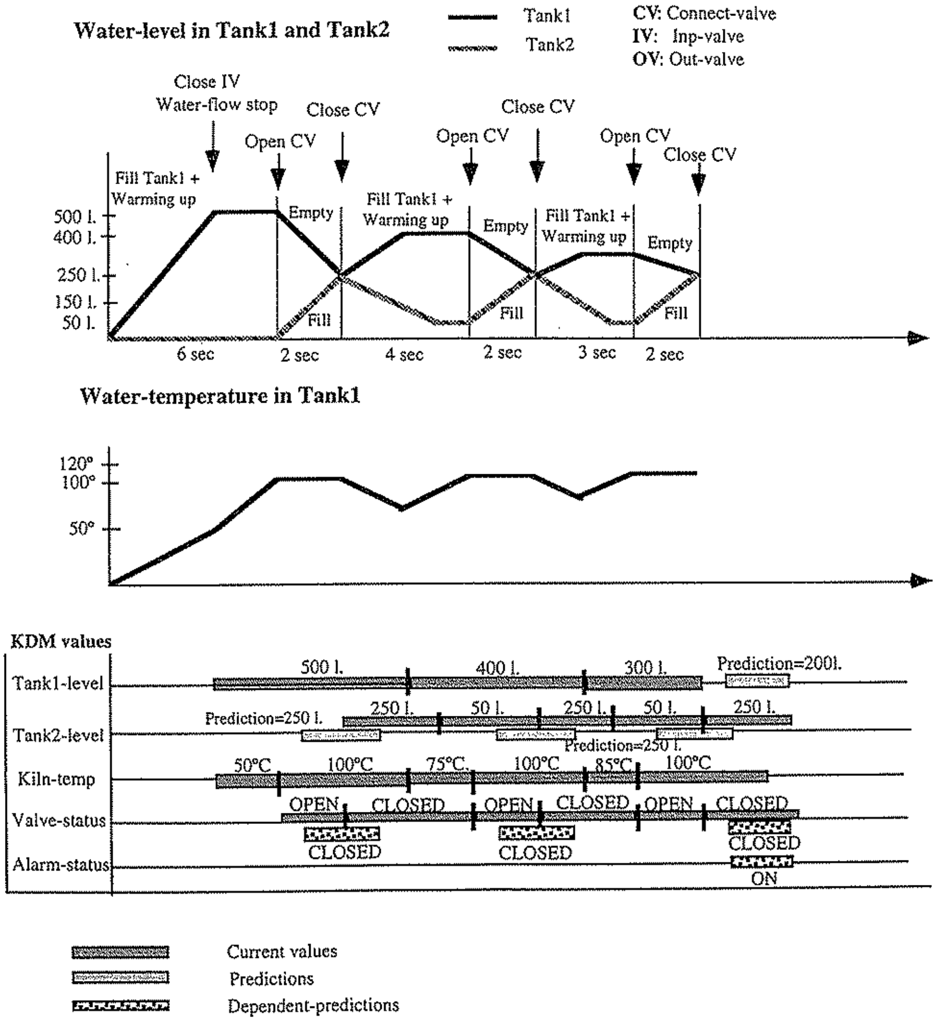

Fig. 9. Process behavior.

R1 is fired and volume in Tank2 is predicted to be 250 liters. The KDM is informed when Tank2 contains 250 l. (prediction fulfilled) and the connect-valve is closed at that time (*R4*).

- Next time the KDM is informed about 400 l. in Tank1 and a temperature of 75°C yet. When the water is enough warm, the connect-valve is open and liquid is poured in Tank2. The connect-valve is closed again. The process goes on and at the third input of water in Tank1 the system checks the slope in the water-level of Tank1 is decreasing (**R2** is fired). Since the new volume is estimated not to reach 250 l. the valve is planned to close at the same time (**R3**).
- Agent1 then commits its conclusions and Agent2 is then aware of a prediction on the level of Tank1. **R5** is then fired taking into account that the level in Tank1 is estimated to be below 250 l. for a period of at least 5 sec. An alarm is planned to ring at the same time that the new value for the level of Tank1 is calculated.
- At this time the system is ready to assist a possible malfunction in the system. In case the prediction on the water level of Tank1 is not satisfied (i.e. the flow of water increases and Tank1 is filled again with 500 liters) all derivations from the old fact are deleted.

Two important features in the system are remarkable:

- Rules are also fired on future values. This functionality allows to anticipate changes in the world and plan properly actions in accordance. In this way we can dispose of a picture of the expected future and new inputs are only used to confirm the predicted changes (rules are not fired again since the reasoning process was already done at an earlier time).
- The multiple access protocol permits agents to work with more reliable information. Notice in the example that Agent2 does not execute until Agent1 downloads all the changes in the KDM. Let's suppose for instance that the prediction on the water-level in Tank1 is deleted while Agent1 is still in execution. This prediction would not be then communicated to the KDM and Agent2 would not reason about; otherwise Agent2 would be reasoning with a false prediction.

7. Conclusions

In this paper, a temporal blackboard for a multi-agent world has been presented. Temporal information is associated to the temporal attributes of stored objects using a model based on time points. The proposed model permits us to represent a great number of situations including temporal imprecision. In order to have efficient methods to be applied under time constraints, past facts are stored with respect to an absolute clock and future facts are stored using a graph when it is not otherwise possible due to the fact dependencies. Moreover, a proposal for multiple access and coherence management has been described in this paper. The following issues can be considered in the protocols: the lowest granularity level is at the slot level, concurrence control and coherence are managed. The method proposed is appropriate for working with agents as the timing constraints of agents are met as much as possible, and the protocol allows working with agents when no computation time or resources are known in advance.

The proposed architecture is the basis of the REAKT toolkit, a software environment for the development and production of real-time knowledge-based systems, which is being used in particular to develop an on-line alarm management application installed in a large oil refinery in Italy. The main goal of the present work is to illustrate the adaptation of a temporal blackboard to meet the requirements of real-time knowledge-based systems: in particular it provides functionalities allowing continuous operation and data input handling, efficient and powerful knowledge-based reasoning and temporal reasoning.

Acknowledgments

The authors acknowledge T. Chehire, A. Mensch, J.Y. Quemeneur, D. Kersual, F. Charpillet, J.J. Galán, I. Rodríguez, E. Pezzi, C. Luparia, A. Lupi, M. Shenton and R. Fjellheim for their comments and their participation in the global architecture described in this paper.

References

[1] J.F. Allen, Towards a general theory of action and time, *Artificial Intelligence* 23 (1984).
[2] A. Crespo, V. Botti, F. Barber, D. Gallardo and E.Onaindía, A temporal blackboard for real-time process control, *Engineering Applications of Artificial Intelligence* 7 (3) (Pergamon Press, 1994) 225-266.
[3] F. Barber, V. Botti, E. Onaindía and A. Crespo, Temporal reasoning in REAKT: An environment for real-time knowledge-based systems, *AICOMM.* 7 (3) (1994) 175–202.
[4] R. Bisiani and A. Forin, Parallelization of blackboard architectures and the Agora system, in: *Blackboard Architectures and Applications* (Academic Press, 1992).
[5] D. Corkill, Design alternatives for parallel and distributed blackboard systems, in: *Blackboard Architectures and Applications* (Academic Press, 1992).
[6] A. Crespo, J.L. Navarro, R. Vivó. A. Espinosa and A. García, RIGAS: An expert server task in real-time environments, *IFAC/IFIP Symp. on Artificial Intelligence in Real Time Control*, Delft (1992).
[7] Y. Demazeau and J.P. Müller, Decentralized artificial intelligence, in *Decentralized A.I.* (North-Holland, 1990) 3–13.
[8] J. Ferber and M. Ghallab, Problématique des univers multi-agents intelligents, in: *Actes des Journées Nationales Intelligence Artificielle* (Teknea, Toulouse, 1985).
[9] M.P. Georgeff and A.L. Lansky, eds., *Reasoning about Actions and Plans* (Morgan Kaufman, 1987).
[10] B. Hayes-Roth, Architectural foundations for real-time performance in intelligent agents, *Real Time Syst.* 2 (1,2).
[11] V. Jagannathan, Realizing the concurrent blackboard model, in: *Blackboard Architectures and Applications* (Academic Press, 1992).
[12] T. Dean, Using temporal hierarchies to efficiently maintain large temporal databases, *JACM* 36 (4) (Oct. 1989) 687-718.
[13] V.R. Lesser, J. Pavlin and E. Durfee, Approximate processing in real-time problem solving, *AI Mag.* 9 (1) (1988).
[14] A. Mouabdid, F. Charpillet and J.P. Haton, Approximation and progressive reasoning, *Proc. AAAI Workshop on Approximation and Abstraction of Computational Theories*, San Jose (July 1992).
[15] W.Perkins and A.Austin, Adding temporal reasoning to expert-systems building environments, *IEEE-Expert* (Feb. 1990).

[16] Thomson, Syseca, Crin, GMV, UPV, Marconi and Etnoteam, REAKT: Environment and methodology for real-time knowledge based systems, *ESPRIT II 4651* (1990–93).

[17] L. Sha, R. Rajkumar, S. Son and Ch. Chang, A real-time locking protocol, *IEEE Trans. Comput.* 40 (7) (July 1991).

[18] A. van Tilborg and G. Koob, Foundations of real-time computing: Scheduling and resource management, in: *Concurrency Control in Real-Time Database Systems* (Kluwer Academic, 1991).

[19] S. Son and J. Lee, A new approach to real-time transaction scheduling, in: *Proc. Fourth EUROMICRO Workshop on Real-Time Systems*, Athens, Greece (June 3-5, 1992).

[20] A. van Tilborg and G. Koob, Foundations of real-time computing: Scheduling and resource management, in: *Scheduling in Real-Time Transaction Systems* (Kluwer Academic, 1991).

[21] L. Steels, Cooperation between distributed agents through self-organisation, in: *Decentralized A.I.* (North-Holland, 1990) 175–196.

Vicente J. Botti received the B.S. degree in Electrical Engineering and Ph.D. degree in Computer Science from the Polytechnical University of Valencia, Spain in 1982 and 1990 respectively. He is currently the head of the Department of Computer Science, Polytechnical University of Valencia, Spain. Since 1985, he has worked on the development of knowledge-based systems. His research interests include distributed Artificial Intelligence, real-time knowledge based systems and non-monotonic reasoning.

Federico A. Barber received the B.S. degree in Electronic Engineering from the Polytechnical University of Madrid, Spain in 1981, and the Ph.D. degree in Computer Science from the Polytechnical University of Valencia, Spain in 1990. He is currently an Associate Professor in the Department of Computer Science, Polytechnical University of Valencia, Spain. Since 1985 he has worked on the development of knowledge-based systems. His main research areas include knowledge representation, temporal reasoning and planning.

Alfons Crespo received the B.S. and Ph.D. degree in Electrical Engineering from the Polytechnic University, Valencia, Spain in 1979 and 1984 respectively. He is currently a Professor of Computer Engineering and Science at the Polytechnic University of Valencia, Spain. Since 1988, he has been the head of the Knowledge Technology group, leading several national and European research projects. His areas of technical interests are real-time systems, automation process control and real-time operating systems.

Eva Onaindia received the B.S. degree in Computer Science from the Polytechnical University of Valencia, Spain in 1990. She is currently pursuing the Ph.D. degree in Computer Science at the Polytechnic University of Valencia, Spain, where she is a Teaching Assistant with the Department of Computer Science. Her research interests are in knowledge representation, temporal reasoning and Artificial Intelligence in real-time.

Ana Garcia-Fornes received the B.S. degree in Computer Science from the Polytechnic University of Cataluña, Spain in 1986. She is currently pursuing the Ph.D. degree in Computer Science at the Polytechnic University of Valencia, Spain, where she is an Assistant Professor with the Department of Computer Science. Her research interests focus on real-time systems scheduling, real-time operating systems, and real-time knowledge-based systems.

Ismael Ripoll received the B.S. degree in Computer Science from the Polytechnical University of Valencia, Spain in 1992. He is currently pursuing the Ph.D. degree in Computer Science at the Polytechnic University of Valencia, Spain, where he is an Assistant Professor with the Department of Computer Engineering and Science. His current research interests are real-time systems scheduling and concurrency control.

methodology for

ıput. 40 (7) (July

management, in:

ı EUROMICRO

management, in:

zed A.I. (North-

ːeived the B.S. de-
r Science from the
versity of Valencia,
ː is currently pursu-
ːgree in Computer
lytechnic University
in, where she is a
ıt with the Depart-
ːr Science. Her re-
are in knowledge
ːemporal reasoning
ıtelligence in real-

ːs received the B.S.
ter Science from the
ːersity of Cataluña,
e is currently pursu-
.egree in Computer
ılytechnic University
in, where she is an
.or with the Depart-
ːer Science. Her re-
focus on real-time
ng, real-time operat-
ıd real-time know-
ːms.

ːceived the B.S. de-
ter Science from the
niversity of Valencia,
He is currently pursu-
degree in Computer
'olytechnic University
ıain, where he is an
ssor with the Depart-
uter Engineering and
ırrent research inter-
ıe systems scheduling
y control.

Domingo Gallardo received the B.S. degree in Computer Science from the Polytechnical University of Valencia, Spain in 1991. He is Assistant Professor with the Department of Computer Science, University of Alicante, Spain. He is pursuing the Ph.D. degree in Computer Science at the Polytechnical University of Valencia, Spain. Current research interests include robot control architectures, real-time systems and vision.

Luis Hernandez received the B.S. degree in Computer Science from the Polytechnical University of Valencia, Spain in 1992. He is a Ph.D. candidate in Computer Science at the Polytechnic University of Valencia, Spain, where he is a Teaching Assistant with the Department of Computer Science. His research interests are in knowledge representation, non-monotonic reasoning and distributed Artificial Intelligence.

DATA & KNOWLEDGE ENGINEERING

Publication information

DATA & KNOWLEDGE ENGINEERING (ISSN 0169-023X). For 1995 volumes 15–17 are scheduled for publication. Subscription prices are available upon request from the publisher. Subscriptions are accepted on a prepaid basis only and are entered on a calendar year basis. Issues are sent by surface mail except to the following countries where air delivery via SAL is ensured: Argentina, Australia, Brazil, Canada, Hong Kong, India, Israel, Japan, Malaysia, Mexico, New Zealand, Pakistan, PR China, Singapore, South Africa, South Korea, Taiwan, Thailand, USA. For all other countries airmail rates are available upon request. Claims for missing issues must be made within six months of our publication (mailing) date. Please address all your requests regarding orders and subscription queries to: Elsevier Science B.V., Journal Department, P.O. Box 211, 1000 AE Amsterdam, The Netherlands. Tel.: 31-20-4853642, fax: 31-20-4853598.

Instructions to authors

All contributions should be written in English with an abstract and a list of keywords, and should be sent in five copies to the Editor-in-chief, the Editor for Europe, or one of the Editorial board members. The authors are requested to put their mailing address on the manuscript. Upon acceptance of an article, the author(s) will be asked to transfer copyright of the article to the publisher. This transfer will ensure the widest possible dissemination of information. No page charge is made.

Please make sure that the paper is submitted in its final form. Corrections in the proof stage, other than printer's errors, should be avoided; costs arising from such extra corrections may be charged to the authors.

The manuscript should be prepared for publication in accordance with instructions given in the "Instructions to Authors" (available from the Publisher), details of which are condensed below:

1. The manuscript must be typed on one side of the paper in double spacing with wide margins. A duplicate copy should be retained by the author. Electronic submissions (accompanied by a paper printout and the original figures) are welcome.
2. All mathematical symbols which are not typewritten should be listed separately.
3. Footnotes, which should be kept to a minimum and should be brief, must be numbered consecutively and typed on a separate sheet in the same format as the main text.
4. Special care should be given to the preparation of the drawings for figures and diagrams. Except for a reduction in size, they will appear in the final printing in exactly the same form as they were submitted by the author; normally they will not be redrawn by the printer. In order to make a photographic reproduction possible, all drawings should be on separate sheets, with wide margins, drawn large size, in India ink, and carefully lettered. Exceptions are diagrams only containing formulae and a small number of single straight lines (or arrows); these can be typeset by the printer.
5. References must be numbered alphabetically. In the text they should be referred to by bracketed numbers. The list of references must be typed on separate sheets, in the same format as the main text, and ordered consecutively, according to the following models:

   For a paper in a contributed volume:

   [1] R. Elmasri and G. Wiederhold, GORDAS: A formal high-level query language for the entity-relationship model, in: P.P. Chen, ed., *Entity-Relationship Approach to Information Modeling and Analysis* (North-Holland, Amsterdam, 1983) 49–72.

   For a paper in a journal:

   [2] J. Mackinlay and R. Genesereth, Expressiveness and language choice, *Data & Knowledge Engrg.* 1 (1985) 17–29.

   For a book:

   [3] H.F. Korth and A. Silberschatz, *Database System Concepts* (McGraw-Hill, New York, 1986).

   For an unpublished paper:

   [4] S.E. Fahlman, A system for representing and using real-world knowledge, MIT Technical Report AI-TR-450, Cambridge, MA, 1977.
6. When accepted the final manuscript should be sent to the corresponding editor, together with a photo (passport-size) and a short biography of each author.

Note to authors: Please mention your email address and/or fax number on the first page of your manuscript.