Time Annotation in COMMONKADS Rules

Jose A. Maestro, César Llamas, Carlos J. Alonso
Grupo de Sistemas Inteligentes
Departamento de Informática
University of Valladolid
{jose, cllamas, calonso}@infor.uva.es

Abstract

Real Time expert systems design presents problems concerning both, knowledge and time constraint representation. COMMONKADS has become one of the most successful approach to knowledge based systems construction. So much effort has been devoted to employ this methodology for the domain knowledge description of monitoring and diagnosis in a dynamic domain. In a time-dependent domain, the time is an important variable itself which has to be taken into account. Therefore, this kind of system requires an intelligible representation of time in order to be able to write rules in which time is an important issue.

In this paper we focus on how to include the time into part of the the domain system knowledge description. The necessary constructors for our domain problem have been included and made some simple enhancements to represent time in the usual rules with the COMMONKADS notation, CML2. We present some examples to illustrate their use.

1 Motivation

Real-Time domains represent a challenging environment using the knowledge-based approach. Nevertheless, it is well known that classical AI techniques are not suitable for addressing environments with time constraints. There exists some situations where it makes sense to use this classical techniques. Specially when all other techniques are not suitable or as has been stated by *Laffey et al.* [8] "The principal reason for using a Real-Time expert system is to reduce the cognitive load on users or to enable them to increase their productivity without the cognitive load on them increasing". In [8] it is also proposed the features which would be expected in a Real-Time expert system. These features include a "temporal reasoning facility" which allows the "representation of temporal relationships" and the capability for "maintaining, accessing and statistical evaluating historical data".

Perhaps, one of the most important contributions to knowledge engineering is the COMMONKADS methodology by *Schreiber et al.* [11]. In this methodology, the knowledge involved in a reasoning process is represented by different ways, including procedures, rules, object oriented techniques, and so on. The methodology split off the knowledge involved in a system in domain knowledge, inference knowledge and task knowledge. Each one of these is referred to a different level of knowledge existing in the environment.

For some years ago, it has been developed several software applications based on knowledge for monitoring and diagnosing industrial on-line processes at the University of Valladolid. The knowledge approach has been applied to a medium-size factory about 400 control loops. Controlling the processes was not desired, but monitoring the plant processes in order to assist failure detection and identification.

Such kind of systems, which have a time-dependent behaviour and can include several models of normal and abnormal behaviour, could fall better within the class of *Monitoring & Diagnosis Assistant* as it is called by *Bredeweg* in [4, p. 123], rather than *Monitoring & Control System*, and therefore the knowledge approach fits well on them.

Despite in COMMONKADS time is not considered, it is still a suitable methodology for knowledge representation and knowledge engineering. The knowledge engineering is a young discipline which is still evolving. Several attempts for modifying and/or extending KADS have appeared in the literature, COMMONKADS-RT [7] is a valid example, and we would like to contribute with a slightly, partial change in the COMMONKADS notation to deal with the problem of time representation in the rule specification stage.

First we present the necessity of including time in the knowledge representation, in this case: the rules. In the next section a time representation model is argued. The time model selected is applied to two case studies that presents a relevant importance: rules with an explicit reference to past and then rules which reference future. In the last section some additions to the standard CML2 notation are proposed, enough, in our opinion, to tackle with the problems we have found in modeling knowledge related with time.

2 Statement of the Problem

We have applied the knowledge based approach to the supervision of a beet-sugar factory (*Alonso et al.* in [1] and [2]). This kind of system is characterized by its complexity and relays heavily on human supervision mainly as a consequence of being a continuous system, not a batch system. The plant behaves in several ways depending on some production optimization constraints. Apart from this, the plant supervisor and operator posses an implicit model of the plant in relation to the possible failures and malfunction candidates, in the case of troubles appear.

Finally, the dynamic nature of the knowledge involved in the control and supervision of the plant adds a temporal dimension to the usual static knowledge approach.

In our opinion, in this problem, the complete solution representation leads to the explicitly inclusion of time in the knowledge description involved. Despite the time is not usually desirable to be included in the representation, in some problems is almost unavoidable, as it is closely related to both the environment and the solution expertise model.

Our principal issue, in this paper, is to present an easy and comprehensive time representation in the rules that represents the applicable knowledge in the domain.

3 Time Representation

The sort of monitoring and diagnosis approach chosen needs some kind system model to work properly, as a model-based approach to monitoring and diagnosis is taken. In a time-dependent domain like this, it could be interesting to represent the time explicitly inside the model specification, since the involved knowledge is time-dependent. In this way, the analysis would reflect same essential temporal aspects. For example, a typical (pattern) rule might look like this:

> *"the signal is going wrong if it is over the threshold during the last three minutes."*

In this example, the data is time-dependent. Even more, the signal data must be looked at for the last three minutes at least. This kind of time reference requires two types of concepts; the concept of instant of time and the concept of interval of time. The former appears in this rule as "three minutes" which states a fixed point of time. The last is the lapse from three minutes ago to *now*. It is due that "the last three minutes" is referred to now implicitly.

Then, both intervals and instants have to be included in order to manage the time consequently. Even more, we need jointly, the concept of "now" and the notions of bounded "past" and "future". Also, a time model is needed, which states the type of time that we ought to represent.

3.1 Time Model

The computational treatment of the time is an attractive problem for mathematical and computer science communities for many years. Some important efforts has been done for modeling the time itself and its properties using logic, resulting in powerful formal systems. The COMMONKADS rules are a sort of simply logic rules describing the knowledge needed to resolve the problem. So, using logic is not a choice but almost a kind of imposition. In this way, it would be a good idea to maintain the notation as simpler as possible.

The time model must allow us to refer to the past, the present and the future. As the previous rule shows, it is necessary at least make references to past and present. This is a rule pattern for monitoring. A simple sort of diagnosis is also possible to be modeled with this template. However, a complex diagnosis process should convey references to future.

It is also necessary both relative and static time references. It could be possible to refer a fix instant of time in the past or to the future and, of course, it must be exists the concept of the present instant of time (now).

Furthermore, references to intervals must be possible. Even more, references to all the points of an

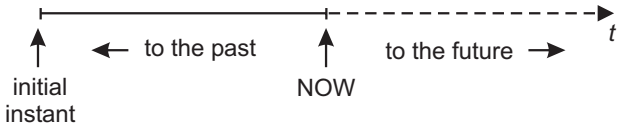

Figure 1: A linear time model.

interval and to a specific point in an interval must be desirable. And, maybe, it is also desirable, to state relationships between intervals, and between an interval and an instant. Then, the definition of a relationship set is almost unavoidable.

In recent years, a mathematical basis for a computational logic treatment of time has been proposed by several authors. *Torsun* in [12] proposes a linear, discrete, bounded past and future and interval based time model for the computational tractability of USF logic. This kind of time model fits very well our necessities. It has bounded past and future, and it is linear too.

Barber et al. in [3] present an environment, REAKT, that allows past, present and future time references. This environment is designed to construct knowledge based control systems. Predictive control is supported, so that the environment can manage multiple predictions at one instant in the future. This kind of system is based on a branching time model. In this model, one past and present and several futures are possible, so that the system can deal with uncertainty in future events.

Our software applications are not going to control, but only diagnosing the industrial plant. It means that no predictive control is needed and, therefore, no actions are going to be taken in advance, not even judgements about which component(s) is malfunctioning are going to be done beforehand. So that a time model that allows prediction representations, is not needed in our models. The model of time we are using allows the sense of a unique timeline as is shown in the figure 1.

4 A Case Study

First of all, a common first order logic alike notation has been chosen to write out the knowledge involved in the problem. This representation is not a strict logical formalism, but mostly a natural and intuitive representation of the system logic as it is stated in [11, ch. 13]

The diagnosis system exploits rules that make reference to future events, whereas the monitoring subsystem usually employs facts that rely on the past. Then we decided first to focus on this type of rules, trying to preserve this simplicity on the rules that refers to future.

4.1 Past Events

The implemented monitoring (and diagnosis) task is the model-based approach. The monitoring task only refers to data and events in the past and present. A logic rule template for monitoring is shown below:

$$\forall\, t \in (\text{NOW} - \text{period}, \text{NOW}):\ \text{value}(t) > \text{threshold} \rightarrow \text{state} = \text{BAD}$$

This rule is refereed to an observed signal. It assigns a state to the signal. This state depends on a threshold. If the rule is over the threshold during at least certain *period* of time, the state is consigned BAD. This means that something could be going wrong. The value of the signal at a precise instant of time is denoted by value (t) where t is a dummy variable that takes values in the interval stated, at it is shown in figure 2.

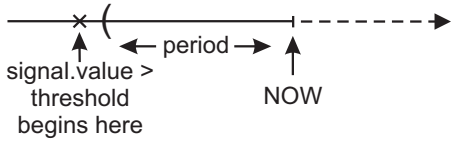

Figure 2: Sample timeline related to the monitoring rule.

As it has been noted, the diagnosis may involve past, present and future data and events. The designed diagnosis is divided into two types named *fast* diagnosis and *delayed* diagnosis. The former takes into account the recent past (just now and a short period of time after) and intends to find out what component would be malfunctioning, and what is the reason (which in COMMONKADS is called, a causal-dependency based diagnosis [4]). This sort of diagnosis is the *fast* diagnosis, which tries to identify the faults at the moment, so that the rules are similar to the monitoring rules. However, there exists a

significant difference between monitoring and diagnosis, the diagnosis rules may, usually have to refer more than one signal. A pattern for a rule for fast diagnosis could be like this one:

$\forall$ t $\in$ (NOW – signal1.beforePeriod, NOW):
signal2.value (t) $>$ threshold $\wedge$
signal3.state $\neq$ BAD
$\rightarrow$ fault = signal1.component + "is going wrong"

The usual dotted notation has been used to tackle with more than one signal. It is possible to have more than one interval (in the example above there is only one interval) and it is possible to have relationships between intervals and instants.

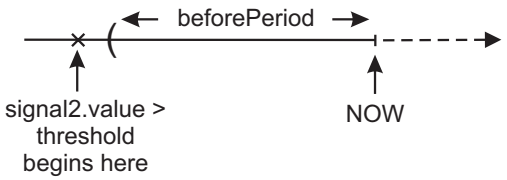

Figure 3: Sample timeline related to the diagnosis rule.

This rule means the state of signal1 depends on the value of signal2 over an interval and the state of signal3. If the antecedent is evaluated to be true then the consequent is a string that points to the component that is malfunctioning as it is shown in figure 3. Of course, an output string is not obligatory.

4.2 Future Events

When it is known a signal is getting wrong, sometimes remains a strong uncertainty about what component is failing, doing necessary to wait for enough symptoms to appear and be able to diagnose the problem. In other words, there would appear other kind of rules that refers to a future instant. These rules are embraced by the *delayed* diagnosis. Now here arises the problem of tackling with the future in the rule.

We would like to maintain the declarative form of the rules about knowledge used in COMMONKADS, so we have extended the standard set of constructors described in CML2.

The knowledge involved in this kind of diagnosis adopts the following aspect:

> *"If the signal2 has been over the threshold in any instant 3 minutes before now and the signal3 state is not BAD during 2 minutes after now then the component malfunctioning is the one related to signal1."*

A possible template for a time-dependent rule of this kind of diagnosis may be the one shown below:

$\exists$ t $\in$ (NOW – signal1.beforePeriod, NOW):
signal2.value (t) $>$ signal2.threshold $\wedge$
$\forall$ t $\in$ (NOW, NOW + signal1.afterPeriod):
signal3.state (t) $\neq$ BAD
$\rightarrow$ fault = signal1.component + "is going wrong"

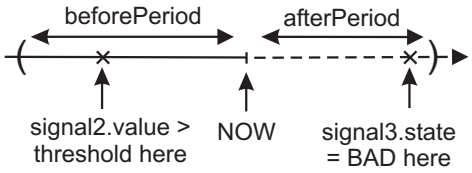

Figure 4: Sample timeline related to the delayed diagnosis.

In this rule the time is relative to NOW and hence the interval (NOW – signal1.beforePeriod, NOW) refers to the past (every period is supposed to be positive) as NOW – signal1.beforePeriod is before NOW. Likewise, the interval (NOW, NOW + signal1.afterPeriod) refers to a future instant since NOW + signal1.afterPeriod is after NOW. In the figure 4 is shown a time line for a possible scenario for this kind of rule.

It is possible, of course, to rewrite this type of rules to be similar to other "usual" rules in the knowledge base. If the point NOW is moved to the instant in which the rule is completed then a rule only focused on the past is obtained (similar in the form that they are written, since both forms are going to be conceptually equivalent).

This way of express knowledge with time involved, allow us to preserve a declarative representation of the system knowledge. Otherwise it must be necessary to take an operational point of view. *Rao et al.* [5] and *Myers* [10], for example, take the operational approach. In [10], even is stated a set of constructors that includes primitives such as WAIT-FOR. However, COMMONKADS enforces a declarative point of view in rules. COMMONKADS states a procedural description for the task method description, but it could be seen that including *all* the time related knowledge at this level has some drawbacks: it could complicate the description (*Alonso et al.* [2]); could force the use of another new primitives like FORK, and so on.

5 Some Design Issues

This kind of rule is though to be evaluated by parts. Just right now, it could be evaluated the past part of the rule, and at some instant of the future, the rest of the rule will (eventually) be evaluated, and maybe completed.

TURBOLID [2] uses this approach to supervising and diagnosing a industrial plant. The figure 5 shows the sequential steps in the diagnosing process in TURBOLID.

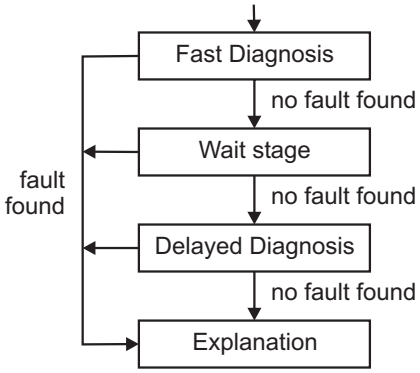

Figure 5: Sequential step chaining in Turbolid.

Including this kind of rules in the real knowledge base is straightforward as it can be split up in the past part and in the future part. At least two rules have to be written down, maybe more, in the expert system and must be properly chained in the normal way it is done in an expert system.

The last rule could be rewritten as it is shown here:

$\exists$ t $\in$ (NOW – signal1.beforePeriod, NOW):
signal2.value(t) $>$ signal2.threshold
$\rightarrow$ (τ = NOW) **[**
$\forall$ t $\in$ (τ, τ + signal1.afterPeriod):
signal3.state (t) $\neq$ BAD
$\rightarrow$ fault = signal1.component +
"is going wrong" **]**

Note that, the rule has been split off into two parts. The first part is referred to the past and/or the present instant. This part is included, as a rule in its own in the *fast diagnosis stage* in figure 5. This stage contains those rules that depends only on past and present time.

If the fault is not found out, the system goes to a *wait stage*. This wait stage is looking for new the data to appear. The system waits until any rule, containing future references, can be completed. Those rules that can not be completed at time are dropped out the working memory. Whenever a rule is completed the explanation stage is reached. If no rule is completed a default explanation is provided.

The part of the rule that relies on the future, a rule in its own, concerns a new environment (a closure) in that the rule is evaluated referring to a future and static interval. This part is included as a rule in its own in the delayed diagnosis stage and a maximum wait lapse is programmed into the wait stage.

In our study cases, finding out the component that is malfunctioning is concerned with the rules described up to here. However, there are other rules which explains why a component would be malfunctioning. These rules are scanned when the fault is identified. They have to associate a explanation to the fault found.

6 Notation

Intervals, instants and relationships between them have been studied by *Allen & Koomen* and *Vilain*. They have defined relations that appears in [6, p. 205] and we have adopted them as these are enough to express every relation useful to the proposals we have.

A COMMONKADS alike notation [11, appendix A] is defined so that this kind of rules can be included in the expert system documentation. The complete set of primitives are shown in the table 1.

Two new range types are defined, INSTANT and INTERVAL which are defined to refer time. The interval can be defined by two instants but it is a new type in its own. The two logical operators FORALL and EXISTS are redefined to be the interval operators ALWAYS and SOMETIME respectively and also it is included a new set of relationships between intervals and between a interval and a instant. The last logic pattern rule is now again translated into the new CML2 notation in the following example.

SOMETIME t IN (NOW−signal1.beforePeriod,NOW):
signal2.value(t) $>$ signal2.treshold AND
ALWAYS t IN (NOW, NOW + signal1.afterPeriod):
signal3.state (t) ! = BAD

IMPLIES

fault = signal1.component + "is going wrong";

The usual rule definition notation is not modified.

Syntax	Description
INSTANT	concept of “time instant”
INTERVAL	concept of “convex set of instants”
(‘[’ — ‘(’) i_1 ‘,’ i_2 (‘]’ — ‘)’)	interval, made up of two instants i_1 and i_2 with $i_1 \neq i_2$
NOW	constant instant that identify the actual instant of time
ALWAYS *var* IN *interval expression*	temporal universal quantifier; the dummy variable *var* allows references to each value in the *interval* and the *expression* must be verified for each of those values.
SOMETIME *var* IN *interval expression*	temporal exists quantifier; the dummy variable *var* allows references to each value in the *interval* and the *expression* must be verified for at least one of those values.
interval predicates (*A*, *B*)	
A BEFORE *B*	*A* occurs strictly before than *B*
A MEETS *B*	*A* finishes in the same instant that *B* begins
A OVERLAPS *B*	*A* shares its final instans with *B*, and one is not included in the other
A BEGINS *B*	*A* and *B* begin at the same instant and *A* is included in *B*
A FALLS-WITHIN *B*	*A* is strictly included in *B*, both begin in the same instant and *A* is included in *B*
A FINISHES *B*	*A* and *B* finishes at the same instant, and *A* is included in *B*
A EQUALS *B*	*A* and *B* both has the same initial and final instants
instant predicates (*i*) and interval (*I*)	
i PRECEDES *I*	*i* occurs strictly before than *I*
i STARTS *I*	*i* is the initial instant of *I*
i DIVIDES *I*	*i* is after the *I* initial instant and is before the *I* final instant
i ENDS *I*	*i* is the final instant of *I*
i FOLLOWS *I*	*i* occurs strictly after of *I*

Table 1: New primitives for the temporal rules.

7 Conclusion

It has been shown that CML2 is easily extensible, and the time can be naturally integrated into rules, with slightly notation changes.

Also, some discussion has been made about how to include direct references to past, present and future in rules. It is included a useful notation to describe the temporal relations in the domain knowledge. It has been made some consideration about those rules that involves the constant NOW, which refers to the present instant, and how to improve the manageability of those rules including past and future instants and intervals.

References

[1] C. Alonso González, G. Acosta, J. Mira, and C. de Prada. Knoledge based process control supervision and diagnosis: the AEROLID approach. *Expert Systems with Applications*, 14:371–383, 1998.

[2] C. Alonso González, B. Pulido Junquera, G. Acosta Lazo, and C. Llamas Bello. On-line Industrial supervision and diagnosis, knowledge level description and experimental results. *Expert Systems with Applications*, 20(2):117–132, February 2001.

[3] Federico Barber, Vicente Botti, Eva Onaindia, and Alfons Crespo. Temporal Reasoning in REAKT (An Environment for Real-Time Knowledge-Based Systems). *AiCommunications*, 7(3):175–202, September 1994.

[4] J. Breuker and W. Van de Velde, editors. *CommonKADS Library for Expertise Modelling. Reusable problem solving components*, volume 21 of *Frontiers in Artificial Intelligence and Applications*. IOS Press, Amsterdam, 1994.

[5] Anand S. Rao François Félix Ingrand, Michael P. Georgeff. An Architecture for Real-Time Reasoning and System Control. Report 92521, LAAS, 1992.

[6] Dov M. Gabbay, C.J. Hogger, and J.A. Robinson, editors. *Epistemic and Temporal Reasoning*, volume 4 of *Handbook of Logic in Artificial Intelligence and Logic Programming*. Oxford Science Publications, 1995. Antony Galton (coordinador del volumen).

[7] M. Henao, J. Soler, and V. Botti. Developing a Mobile Robot Control Application with CommonKADS-RT. In Moonis Ali László Monostori, József Váncza, editor, *Engineering of Intelligent Systems*, volume 2070, pages 651–660. Springer, June 2001. 14th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, IEAAIE 2001.

[8] Thomas J. Laffey, Preston A. Cox, James L. Schmidt, Simon M. Kao, , and Jackson Y. Read. Real-time knowledge-based systems. *AI Magazine*, 9(1):27–45, 1988.

[9] Jose A. Maestro Prieto, César Llamas Bello, and Carlos J. Alonso González. Anotación de aspectos temporales en la especificación de la tarea en CommonKADS: Aplicación a la monitorización. In *IX Conferencia de la Asociación Española para la Inteligencia Artificial (CAEPIA-TTIA'01)*, pages 449–458, Gijón, España, 2001.

[10] Karen L. Myers. A Procedural Knowledge Approach to Task-Level Control. In B. Drabble, editor, *Proceedings of the Third International Conference on AI Planning Systems (AIPS-96)*, pages 158–165. AAAI Press, 1996.

[11] G. Schreiber, H. Akkermans, A. Anjewierden, R. de Hoog, N. Shadbolt, W. Van de Velde, and B. Wielinga. *Knowledge Engineering and Management, The CommonKADS Methodology.* The MIT Press, 1999.

[12] Imad S. Torsun. *Foundations of Intelligent Knowledge-Based Systems*. Academic Press, 1995.