MECHANISMS FOR TEMPORAL LOGIC IMPLEMENTATION IN RULE-BASED SYSTEMS

Josef Hahn, Karl-Heinz Krempels and Christoph Terwelp
Informatik 5, Information Systems and Databases
RWTH Aachen University, Aachen, Germany

Keywords: Temporal logic, Rule-based system.

Abstract: Rule-based systems are more and more important in middleware architectures and distributed applications. Although support for temporal constructs would be very conveniant for many domains, implementations are not yet widespread. This paper is about several methods to expand rule-based systems and the commonly used RETE algorithm in order to gain basic support for temporal logic constructs. A few promising approaches are discussed and compared with respect to efficiency, memory usage, and implemetation details. The paper is limited on the discussion of temporal logics in rule-based systems and does not take temporal logic in other contexts into account.

1 INTRODUCTION

A rule-based system (RBS) can be used by an user to automatically reason about an existing knowledge base with the help of rules in order to deduce new knowledge. A rule based system consists in general of:

Working Memory which stores the facts that are asserted by the system. Each fact is a data structure that represents the information about one entity or relation in the domain of consideration.

Rule Base which stores the rules that are used to deduce new knowledge. Each rule consists of a condition and an action-list that is processed every time when the condition becomes true.

Inference Machine which applies all rules from the rule base to the working memory to deduce new knowledge. This is called the reasoning process. Whenever the assignment of a new fact satisfies the condition of a rule in the rule base, the action-list of the rule will be executed. The action-list can change the working memory and the rule base to produce new knowledge.

For many domains inside and outside the field of artificial intelligence it is desirable to have access to temporal constructs in order to model the temporal dimension of the domain of discourse in a natural way (Baader et al., 2003) (Ross, 2003). Existing rule-based systems provide none or only a very limited support for temporal modelling constructs (Lin and Krempels, 2008) (Krempels et al., 2009) (Hill, 2003). The objective of this paper is to introduce a few practical ways for upgrading an existing RETE-based system.

Therefore, the first step is to achieve support for temporal facts and temporal rules. A temporal fact is a fact with a finite life time that can be limited by a predefined activation and expiration time. Outside of this time interval the inference machine assumes that the fact does not exists. The same capabilities have to be provided for rules to model temporal rules.

2 THE RETE ALGORITHM

The RETE algorithm (Forgy, 1990) is inspired by the idea of dynamic programming and is one of the most common approaches for discrimination networks. The algorithm creates a network (a directed, acyclic graph) to process the rules in an efficient way. In this network each test used in the condition of a rule is represented by a node. The whole condition of the rule is represented by a subgraph in the network. The network has one root node as source and for each rule one terminal node as sink. Each node has at least one input, except for the root node. Furthermore, each node has one output, except for the terminal nodes.

The objects that are processed in the network are facts and tuples of facts. A node applies a predefined set of filters to its input facts and forwards the match-

Hahn J., Krempels K. and Terwelp C..
MECHANISMS FOR TEMPORAL LOGIC IMPLEMENTATION IN RULE-BASED SYSTEMS.
DOI: 10.5220/0003310605080513
In *Proceedings of the 3rd International Conference on Agents and Artificial Intelligence* (ICAART-2011), pages 508-513
ISBN: 978-989-8425-40-9

ing ones to the output. Every time, when a fact or a tuple arrives at a terminal node of the network, the rule which this node belongs to is activated. Every node in the network has its own memory that stores all the facts that fulfill the test condition of the node. In this way the network saves intermediate processing results and in this meets the dynamic programming requirements.

The filtering inside the network reflects the filtering in the condition-part of the rules inside the rule base. This is enabled by different types of nodes:

Object Type Node selects all facts arriving on its input, which are derived from a defined template. For example, it can forward all facts based on the template “customer”.

Alpha Node selects all facts arriving on its input, which have a defined value in a defined slot. For example, it can forward all facts with the value “red” in the slot “color”.

Join Node has an alpha input and a beta input. The alpha input handles facts whereas the beta input handles tuples of facts.

A Join Node expands the tuples by the facts in the way of the cartesian product. So, for n-tuples on the beta input, it will output $(n+1)$-tuples. This node type can apply additional filters to the results in the same way as the Alpha Node.

3 THE APPROACH

To extend rule-based systems by temporal reasoning the following steps are required:

1. A time and date data type for facts and rules, which is used by the engine to represent the activation and expiration time for this elements.
2. Extension of the RBS programming language to support the time and date data types for the optional specification of validity intervals for facts and rules.
3. Definition and implementation of functional and relational operators for time and date data types.
4. Enhancement of the RBS engine to process the temporal rules and facts.

Since the first three items can be realized very easy we will focus a little more on the enhancement of the RBS engine to process temporal rules and facts. The approaches presented here were implemented and evaluated with the help of the RBS *Jamocha* that uses CLIPS (C Language Integrated Production System) as rule and fact definition language (programming language).

```
TemporalConstraint = "(temporal-validity "
       StartAttribute Duration ")";
StartAttribute = [MilliSecond] [Second] [Minute]
       [Hour] [Day] [Month] [Year] [Weekday];
Duration = "(duration " DurationValue ")";
MilliSecond = "(msecond " MilliSecondValue ")";
Second = "(s " SecondValue ")";
Minute = "(minute " SecondValue ")";
Hour = "(hour " HourValue ")";
Day = "(day " DayValue ")";
Month = "(month " MonthValue ")";
Year = "(year " YearValue ")";
Weekday = "(weekday " WeekdayValue ")";
Value = "*" | "*/" Number | NumberSequence;
NumberSequence = Number { "," Number };
Number = Digit { Digit };
```

Figure 1: Extensions of the CLIPS language for the definition of temporal constraints.

3.1 Extension of the RBS Programming Language

An extension of the RBS programming language is required in order to support the data types for the optional specification of validity intervals for facts and rules. With the help of the syntactical productions given in Fig. 1 the RBS programming language CLIPS is extended by a *TemporalConstraint*.

With the help of the production of a *TemporalConstraint* it is possible to specify both a fixed time interval, consisting of an activation time and the duration of the rules activation, or a sequence of time intervals. The fixed temporal constraint shown in Fig. 2 describes the time interval starting at the 7th of April 1971 at 0:00:00s.000ms with a duration of 24 hours.

```
(temporal-validity (msecond 0)
     (second 0) (minute 0)(hour 0)
     (day 7)(month 4)(year 1971)
     (duration 1d)
)
```

Figure 2: Example for a fixed temporal constraint.

Fig. 3 shows a periodic temporal constraint that describes the first 12 hours of every Wednesday in April in every year.

```
(temporal-validity (msecond 0)
     (second 0) (minute 0)(hour 0)
     (month 4)(weekday 3)
     (duration 12h)
)
```

Figure 3: Example for a periodic temporal constraint.

The duration can be provided for a fixed as well as for a periodic time constraint either as an integer value in milliseconds (without a suffix) or as an integer value seconds, minutes, hours, or even days (with the corresponding suffix `s`, `m`, `h`, or `d`). The value ranges for milliseconds, seconds, minutes, hours, days, months, years and weekdays are given in Fig. 4.

```
MilliSecondValue : {0, 1, . . . , 999}
SecondValue : {0, 1, . . . , 59}
MinuteValue : {0, 1, . . . , 59}
HourValue : {0, 1, . . . , 23}
DayValue : {1, . . . , 31}
MonthValue : {1, . . . , 12}
Year : {0000,1981,1982, . . . ,3000}
Weekday : {0,1, . . . ,6}
```

Figure 4: Value range restrictions for the definition of temporal constraints.

The *TemporalConstraint* described above was added to the syntactical production rules of the RBS for rules and facts to enable one to define temporal rules and facts. The modified syntactical production rules for the RBS programming language CLIPS are shown in Fig. 5. In both defintions for rules and facts the *TemporalConstraint* was added as an optional attribute that is processed by the temporal reasoning part of the RBS to activate and deactivate the temporal elements at the corresponding points of time.

```
Rule = "(defrule " Name Comment
     Attribute {Condition}
     "=>" {Action} ")";
Attribute = [Priority] [AutoFocus]
     [Version] [TemporalConstraint];

Fact = "(" TemplateName
     [TemporalConstraint] {Slot} ")";
```

Figure 5: Syntactical production rules for rules and facts.

3.2 Temporal Rules and Facts

A simple way to take temporal constraints into account is to provide time dependent rules. We will not discuss temporal facts in detail because they can be emulated easily by temporal rules. We just create a temporal rule for each temporal fact which adds and removes the fact as required.

The implementations of Rete networks suggest three approaches for the realization of temporal rules.

3.2.1 Common Time Fact

The first approach consists in a special fact that is updated periodically in well defined time intervals. Every time when this special fact is updated the activation conditions of all the temporal rules have to be evaluated. Thus, this approach is not very efficient since the computation complexity for the well defined time interval for the update of the time fact will increase with the number of temporal rules.

A sketch of this approach is given in Fig. 6. The special fact *point-in-time* is updated periodically (controlled by a timer). All the temporal rules from the rule base are checked after every update of the special fact.

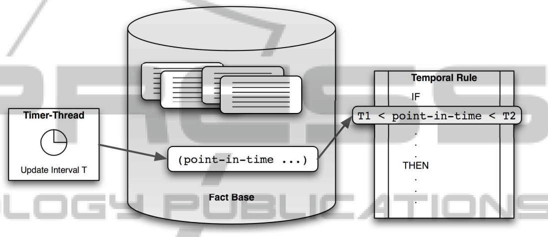


Figure 6: Temporal rule implementation by single time fact.

3.2.2 Temporal Trigger Facts

Another approach is to introduce a trigger fact for each temporal rule as shown in Fig. 7. This fact is used by the rule and controlled by a central timer thread. This timer thread can manage all the existing trigger facts and adds or removes them when the corresponding rule should become valid or invalid. This method should be much more efficient as the first approach because the timer thread is only active to add and remove trigger facts. The remaining time it sleeps and uses a minimum of the systems ressources. Maybe the performance could be further improved by introducing trigger fact sharing, so the amount of required trigger facts and timer thread activities is reduced.

3.2.3 Temporal Nodes

The last approach we discuss is to implement the temporal rules directly in the discrimination network. Therefore, we introduce the node type *temporal node* in the network that will activate and deactivate a temporal rule with respect to its temporal conditions. The implementation of temporal rules can be done either by providing an own thread that is watching the temporal constraints of exactly one rule, or by a common thread that will watch the temporal constraints for all the rules. In the latter way the temporal constraints of rules have to be added to a chronological

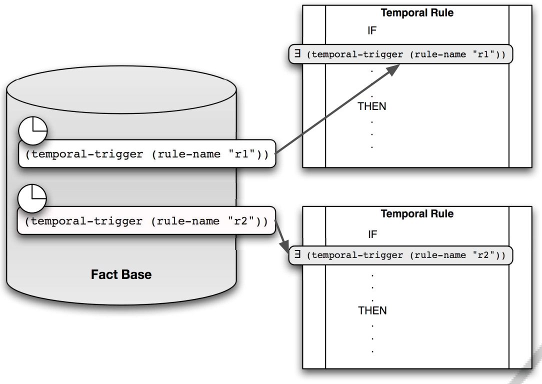


Figure 7: Temporal rule implementation by unique trigger fact.

list of tuples (consisting of the rule-id and the corresponding activation or deactivation time of the rule) for further processing by the common thread. Since for each temporal rule at least one tuple has to be inserted into the list, the compilation time for rules and the processing time of the tuples will depend on the list's length. Therefore, it seems to be recommendable only for applications with a small part of temporal rules and facts.

Temporal facts can also be filtered directly in the discrimination network. The filter for temporal facts can be applied in the alpha network, in the beta network, or at the terminal node of a rule. Applying a filter for a temporal fact in the alpha network takes the advantages of the RETE network in processing facts. A drawback of this method is that facts with a high temporal activation / deactivation rate can overload the test nodes of the alpha network. This is also the case if the filters are applied in the beta network. So, it seems that the right place to check the temporal validity of a fact is the terminal node of a consuming rule. The time point when a fact arrives at the terminal node of a rule is very close to the execution time of the rule, so we have to disregard the dispatching time of a rule in the agenda of the RBS.

4 EVALUATION

All the approaches described above have been evaluated after implementation for three different configurations of the RBS:

Temporal Facts Configuration is used for the determination of the processing time for temporal facts. The set of facts of this configuration contains 30 temporal facts with two slots. All the temporal facts are activated every 10*ms* for 5*ms*. The set of rules of this configuration contains exactly one rule that consumes all the facts comparing the values of its two slots.

Temporal Rules Configuration is used for the determination of the processing time for temporal rules. The set of facts of this configuration contains only one fact. The set of facts of this configuration contains 30 temporal rules that are activated all together every 10*ms* for 5*ms*.

Temporal Facts and Rules Configuration is used for the determination of the processing time for a mix of temporal elements (facts and rules). The set of facts of this configuration contains 30 temporal facts and the set of rules contains 30 temporal rules. The temporal facts and rules are activated every 10*ms* for 5*ms*.

The evaluation of the measurement results is discussed separately to allow a rating of each approach based on all three configurations.

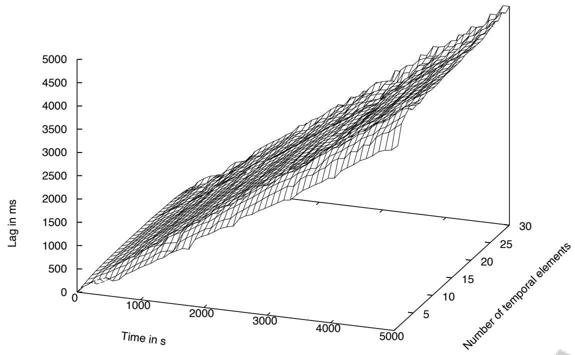
The evaluation of the time fact approach is based on the measurements for the three configurations shown in Fig. 8. The measurements for al three configurations shown in Fig. 8(a), Fig. 8(b), and Fig. 8(c) reflect that the time lag is nearly identical to the passed processing time, showing that nearly no processing of temporal elements happens. The system is clearly totally overloaded.

The evaluation of the trigger fact approach is based on the measurements for the three configurations shown in Fig. 9. The measurements for all three configurations shown in Fig. 9(a), Fig. 9(b), and Fig. 9(c) lead to the conclusion that this approach performs pretty good as long as only temporal facts *or* temporal rules are used but we see a huge change to the worse if they are combined.

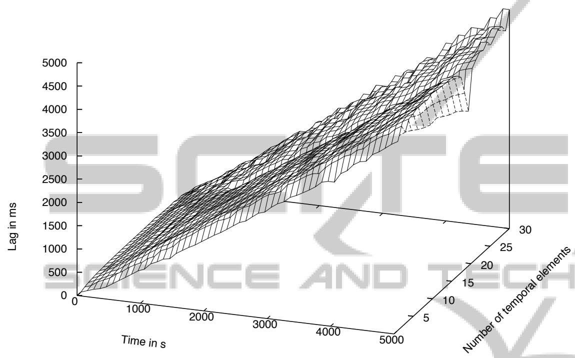
The temporal nodes approach was evaluated based on the measurements for the three configurations shown in Fig. 10. The measurements for alle three configurations shown in Fig. 10(a), Fig. 10(b), and Fig. 10(c) reflect that it performs a little bit better than the trigger fact approach but as worse in the case of using temporal facts and rules combined.

5 CONCLUSIONS AND OUTLOOK

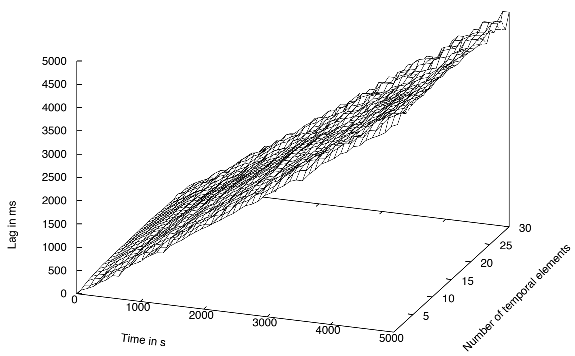
The analysis of the implemented approaches shows that the current implementation of the RBS Jamocha can process among 2500 temporal elements per second without producing a processing time lag. From this value we can derive how many temporal elements can be processed in a rule-based application within



(a) Temporal facts

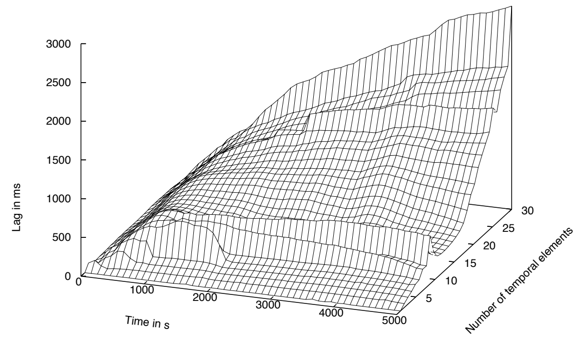


(b) Temporal rules

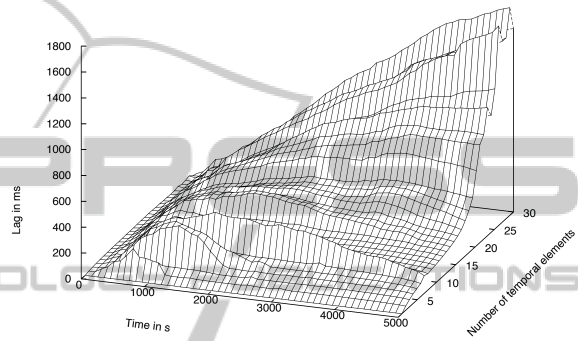


(c) Temporal facts and rules

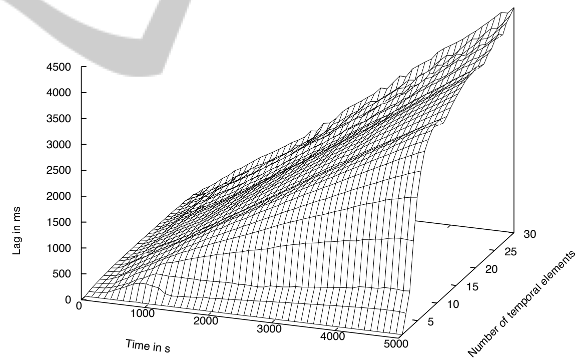
Figure 8: Processing lag for temporal facts and rules using the time fact approach.



(a) Temporal facts



(b) Temporal rules

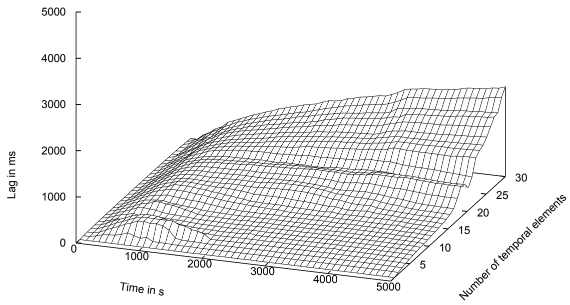


(c) Temporal facts and rules

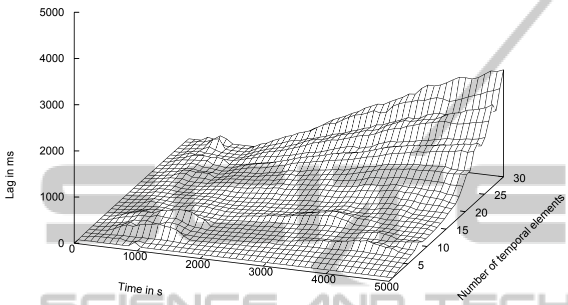
Figure 9: Processing lag for temporal facts and rules using the temporal trigger facts approach.

the given processing interval of one second. Furthermore, we can derive the expected processing lateness or system imprecision if a rule-based application requires a higher number of temporal elements than the estimated value.

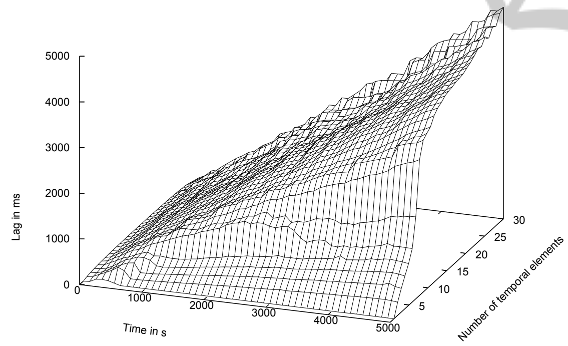
However, with the help of this system parameter a rule-based application must be designed in advanced and the design considerations with respect to time lags for temporal elements have to remain still valid, even if the RBS is overloaded. To overcome this drawback it seems suitable to introduce a special system fact, similar to the time fact discussed above. This fact should provide the respective actual time lag of the RBS and the expected time lag for the next fire cycle (based on the number of activated and nearly activated rules). Further investigations will focus on the prediction of the time lag produced by processing temporal elements in RBS to allow the design of adaptive rule-based applications with variable or differentiated requirements for processing lateness.



(a) Temporal facts



(b) Temporal rules



(c) Temporal facts and rules

Figure 10: Processing lag for temporal facts and rules using the temporal nodes approach.

REFERENCES

Baader, F., Calvanese, D., McGuinness, D., Nardi, D., and Patel-Schneider, P., editors (2003). *The Description Logic Handbook*. Cambridge University Press.

Forgy, C. L. (1990). Rete: a fast algorithm for the many pattern/many object pattern match problem. In *Expert systems: a software methodology for modern applications*, pages 324–341, Los Alamitos, CA, USA. IEEE Computer Society Press.

Hill, E. F. (2003). *Jess in Action: Java Rule-Based Systems*. Manning Publications Co., Greenwich, CT, USA.

Krempels, K.-H., Christoph, U., and Wilden, A. (2009). Jamocha – a Rule-based Programmable Agent. In *International Conference on Agents and Agent Technology 2009 (ICAART'09)*, INSTICC Setubal, Porto.

Lin, P. and Krempels, K.-H. (2008). A temporal logic extension for the rete algorithm. Technical report, Jamocha.org.

Ross, R. G. (2003). *Principles of the Business Rules Approach*. Addison Wesley.