LUND UNIVERSITY

A Comparison Between Three Development Tools for Real-Time Expert Systems: CHRONOS, G2 and MUSE

Årzén, Karl-Erik; Sallé, Stéphane

1989

Document Version:
Publisher's PDF, also known as Version of record

Link to publication

Citation for published version (APA):
Årzén, K.-E., & Sallé, S. (1989). *A Comparison Between Three Development Tools for Real-Time Expert Systems: CHRONOS, G2 and MUSE*. (Technical Reports TFRT-7436). Department of Automatic Control, Lund Institute of Technology (LTH).

Total number of authors:
2

General rights
Unless other specific re-use rights are stated the following general rights apply:
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.
• Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
• You may not further distribute the material or use it for any profit-making activity or commercial gain
• You may freely distribute the URL identifying the publication in the public portal

Read more about Creative commons licenses: https://creativecommons.org/licenses/

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

LUND UNIVERSITY

PO Box 117
221 00 Lund
+46 46-222 00 00

Download date: 20. Dec. 2025

CODEN: LUTFD2/(TFRT-7436)/1–10/(1989)

A Comparison Between Three Development Tools for Real-Time Expert Systems: CHRONOS, G2 and MUSE

Stéphane Sallé
Karl-Erik Årzén

Department of Automatic Control
Lund Institute of Technology
November 1989

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118
S-221 00 Lund Sweden

Document name
INTERNAL REPORT

Date of issue
November 1989

Document Number
CODEN: LUTFD2/(TFRT-7436)/1–10/(1989)

Author(s)
Stéphane Sallé
Karl-Erik Årzén

Supervisor

Sponsoring organisation

Title and subtitle
A Comparison Between Three Development Tools for Real-Time Expert Systems: CHRONOS, G2 and MUSE

Abstract

A comparison, based on a study of how time is treated, between three development tools for real time expert system is exposed. After a presentation of the tools, some of their advantages and disadvantages are pinpointed. Since these tools are only a partial answer to the problem of real time within the expert system's area, a standpoint concerning a good use of each tool is given.

Key words
Expert systems; Real time; Chronos; G2; Muse

Classification system and/or index terms (if any)

Supplementary bibliographical information

ISSN and key title

ISBN

Language
English

Number of pages
10

Recipient's notes

Security classification

The report may be ordered from the Department of Automatic Control or borrowed through the University Library 2, Box 1010, S-221 03 Lund, Sweden, Telex: 33248 lubbis lund.

To be presented at the 1989 IEEE Control Systems Society Workshop on Computer-Aided Control System Design held in Tampa, Florida (U.S.A.) on December 16, 1989.

A Comparison Between Three Development Tools for Real-Time Expert Systems: CHRONOS, G2 AND MUSE

Stéphane E. Sallé and Karl-Erik Årzén

Department of Automatic Control
Lund Institute of Technology
P.O. Box 118, S-221 00 LUND, SWEDEN
Tel: (46) 46.10.87.80

Abstract **- A comparison, based on a study of how time is treated, between three development tools for real time expert system is exposed. After a presentation of the tools, some of their advantages and disadvantages are pinpointed. Since these tools are only a partial answer to the problem of real time within the expert system's area, a standpoint concerning a good use of each tool is given.**

Keywords **- Expert systems; Real time; Chronos; G2; Muse.**

1. INTRODUCTION

There is presently a strong interest in knowledge-based systems within the process industry. Many applications, mainly considering on-line monitoring, diagnosis and alarm analysis (e.g. *Kramer and Finch*, 1989), have been proposed and tested in pilot projects. The technique is also used for closed-loop control. In fuzzy control (e.g. *Mamdani and Assilian*, 1975) expert system rules are used to mimic the human operator's manual control strategy. Expert control (*Åström et al*, 1986; *Årzén*, 1987), seeks to extend the range of conventional control algorithms by encoding general control knowledge and heuristics regarding auto-tuning, adaptation and control loop supervision in an expert system included in the controller.

Knowledge-based systems in regular, day-to-day operation are however still not common. One reason for this has been the lack of appropriate expert system tool suited for on-line, real-time operation. The purpose of this paper is to compare three commercial development tools for real-time expert systems. This is based on the authors experience as users. This is not a study of all the products dealing with this problem.

After a brief description of the problems related to the use of time in an expert system, the different tools are described. We discuss how facts and rules are represented, the possibilities of the inference engine and how time is treated. The computer requirements and the user interfaces are also presented. In the last section, the systems are compared. Some advantages and drawbacks of each tool are pointed out.

2. REAL-TIME ASPECTS OF KNOWLEDGE-BASED SYSTEMS

Real-time, on line applications of knowledge-based systems (KBSs) contains a set of special problems that differ substantially from the conventional off-line consultation applications that KBSs originally were developed for. A short overview of these problems is made in the next section. A more detailed discussion can be found in *Laffey et al* (1988) or in *Chantler* (1988).

Non-monotonicity:

A real-time expert system operates in a dynamic, changing environment. Incoming sensor data, as well as inferred facts, do not remain static during the execution of the program. Data are either not durable and decay in validity with time, or they cease to be valid because events have changed the state of the system. In order to maintain a consistent view of the environment, the reasoning system must be able to automatically retract inferred facts.

Reasoning under time constraints:

Reasoning under time constraints covers the problem where a reasoning system must be able to come up with a solution in time when the solution is needed. Furthermore, the best possible solution within a given deadline is desired. To do this, the system must be able to estimate the time needed to build a solution. A measure of goodness on the solution expressed in terms of completeness, precision, and certainty is also needed.

Asynchronous events and focus of attention:

When a significant, asynchronous event occurs, it is important that the real time system can be interrupted by this event and focus its resources on the currently most important issues.

Temporal reasoning:

Time is an important variable in real-time systems. A real-time system must be able to represent time and reason about past, present, and future events as well as the sequence in which the events occur.

Missing and uncertain data:

Missing data is a problem that must be taken into account since, for example, temporary sensor failures must not lead to a stop of the expert system. Data can also lose validity or have questionable validity because of degradation in sensor performance.

Continuous operation:

Real-time expert system must be capable of continuous operation and, for instance, the reasoning must not be interrupted due to garbage collection.

Several of these issues (non-monotonicity, reasoning under time constraints, etc...) are deep theoretical, philosophical problems

which probably never will be solved. Practical, ad-hoc methods are however emerging for several of the problems.

3. CHRONOS

Chronos is a rule-based, forward chaining system written in ADA, developed jointly by the two French companies EURISTIC Systems and SAGEM.

3.1 The Chronos Knowledge-Base

The knowledge base consists of facts and rules. Chronos is not presently object-oriented and the facts are represented by triplets of the type Attribute (Object) = Value. To handle time, four dates can be associated with a fact: creation date (the time when the fact is entered in the facts base), starting date (the time when the fact becomes valid), ending date (the time beyond which the fact is invalid) and obsolescence time (the duration beyond which an invalid fact is removed from the facts base). These dates can be given in absolute form (terms of day and time), or be related either to the current time or to an arbitrary reference time T0 defined by the user. Any element of a triplet in the rules can be replaced by a variable (included all three elements at the same time).

Rules are used to encapsulate an expert's knowledge of what to conclude from conditions and how to respond to them. All rules have an antecedent that lists the conditions and a consequent that tells what to conclude and how to respond. A rule in Chronos is expressed as: "**as soon as** conditions **then** actions" or "**as long as** conditions **then** actions". The first kind of rule is equivalent to the classical structure "**if** conditions **then** actions". In the second kind of rule, a link is created between the time stamps of the facts of the condition part and the time stamps of the inferred facts. Any modification of the time stamps of the facts in the condition part is spread to the time stamps of the inferred facts. This leads to an efficient non-monotonic reasoning. In the condition part, logic operators (and, no, exits, for any), arithmetic and mathematics operators can be combined without any limits. Tests on the current time, Clock, can also be performed.

The actions are written in a structured procedural language which allows conditional branching and loops. Within the actions, facts can be manipulated (created, modified or deleted), rules can be fired and external procedures written in other languages can be called (the arguments of these procedures must be ASCII strings). All actions can be postponed for a defined period.

Different priority levels are associated with the rules and a rule can be declared as not interruptible. If this is not the case, the execution of a rule can be interrupted by another rule with a higher priority.Parts of rule actions can also be declared as not interruptible.

An example of a rule is given below. This rule is issued from a set of rules which monitors several connected reactors (*Chronos*, 1988). As soon as the reactor temperature is greater or equal to one hundred degrees for the last thirty seconds and none of the valves of the reactor was opened within the five following seconds, then inform the operator and execute the procedure action_1.

In a Chronos rule, the variables are preceded by an exclamation mark and the comments by two hyphens. The end of a statement is marked by a semicolon.

```
rule name_1:
priority :=4;
uninterruptible;

as soon as
   !x:=100;
   temperature(!reactor) >= !x [!t1, !t2];   -- look for a reactor
                                             --with a too high temperature
   clock >= !t1 + 30;
   no(exists associated_valve(!reactor) =!valve  [!t3, !t4] such
      that state(!valve)=opened, !t3 =< !t1+5.0, !t4 > clock);
then
   put_line("Warning, problem with reactor "); put(!reactor);
   call "action_1.exe" (!reactor:in, !t1:in);
   -- call of the external procedure action_1.exe
end rule;
```

3.2 The Real-Time Inference Engine

Chronos's inference engine is purely forward chaining and uses a modification of the RETE algorithm (*Forgy*, 1982). It consists of tasks with different priorities. During execution, three groups can be distinguished. The first one deals with acquisition of external information and data. Its priority is very high in order to realise continuous, asynchronous and/or periodic acquisition. The second group checks if some rules are satisfied by the new facts and if so fires them depending on their priorities. The last one takes care of the removal of no longer valid facts.

During a periodic acquisition, a new fact is created only if the acquired value has changed compared to the previous value. Chronos monitors the associated time stamps.

The rules can be fired either when a change in the database occurs or when a test on clock becomes true. The test on the current time is not treated as the other tests. If all the tests, except the test on clock, of the condition part of a rule R1 are satisfied by a given set of facts, this set of fact is removed from the test network and put in a scheduler. As soon as the test on clock is satisfied, Chronos checks if the set of facts previously defined still satisfies the condition part of R1 and, if this is the case, fires the rule R1.

Three execution modes are available: real time (the expert system uses the computer clock), virtual time (the expert system uses time defined by the supervised process) and step by step.

3.3 The User Interface

The development interface is based on a multi-window interface with menu and mouse interaction. This allows rule editing and display of the rule flow charts with possibilities for zooming and scrolling.

During rule execution, four windows are used which allow Chronos to display acquired and deduced facts, justification of deduced facts and execution trace (messages sent to the operator from rules). The command window allows an interactive dialogue during the execution. The dialogue may be a display of the rule base, a display of parts of the fact base (defined e.g. by: "show state(*)=*"), on-line modifications of the fact base (insertion of a new fact, deletion of a fact). The content of the windows may be stored in files. The number of facts present in the database, the number of rules to be fired (rules present in the agenda), the number of rules in the scheduler are also displayed. A possible saturation of the inference engine can thus be detected. A typical Chronos display screen is shown in Figure 1.

Chronos runs on IBM PC and compatibles, VAX and UNIX workstations. Chronos can be used either as a stand-alone system or

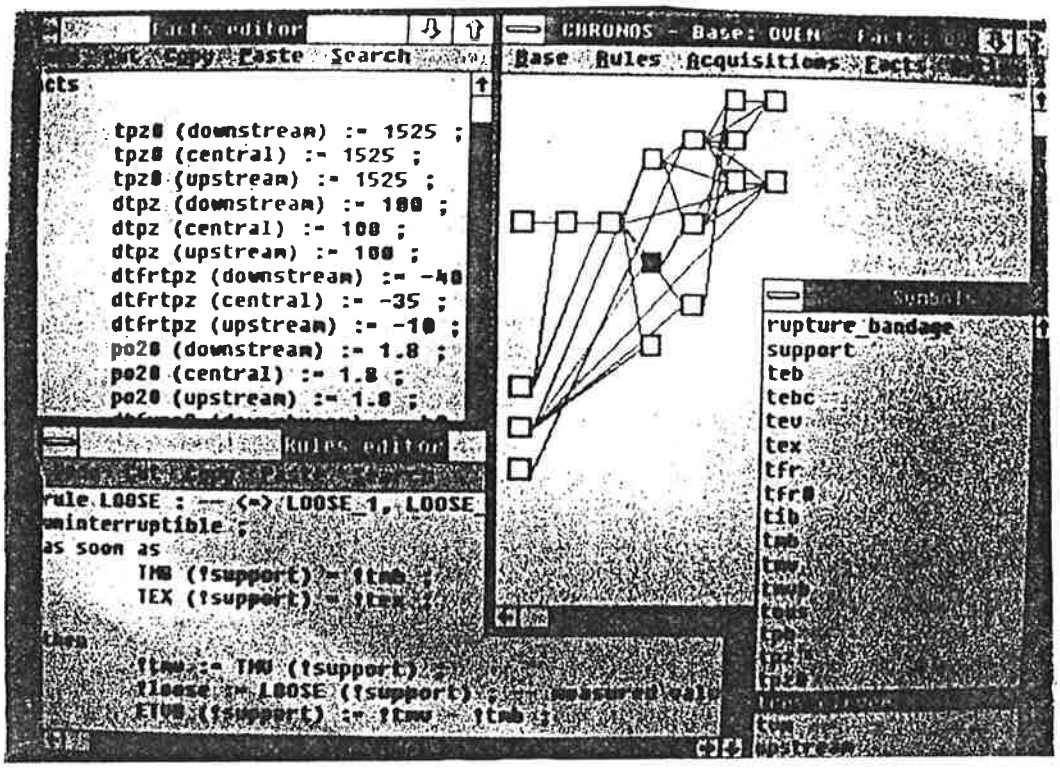

Fig. 1. A typical CHRONOS display screen on PC.

as an ADA package integrated with other ADA programs. On an IBM PC, the required size is at least 1 Mb of free hard disk memory and a 3 Mb RAM memory extension. In January 1989, the price for the IBM PC development licence was 60 000 FF and 120 000 FF for the development licence for UNIX workstations and VAX/VMS. The system was installed on fifteen different sites with application in computer vision, system diagnosis, etc ...

4. G2

G2 from Gensym Corporation is an expert system development tool aimed at real-time, process industry applications (*Moore et al*, 1988). Gensym Corp. was founded by the group from Lisp Machines Inc. who previously had developed the PICON system. G2 is intended to be used on top of a conventional process control system as an operator assistant.

The main part of G2 are: the knowledge-base, a real-time inference engine, a simulator, the development environment, the operator interface, and optional interfaces to external on-line data servers.

4.1 The G2 Knowledge-Base

The knowledge-base consists of three different forms of knowledge: objects, rules, and dynamic models. Objects are used to represent the different concepts of the application. Attributes describe the properties of a certain object. The attributes values may be constants, variables, or other objects. The objects are organised into a class hierarchy with single inheritance. Objects are represented by graphical icons as shown in Figure 2. Relations between objects are represented by connection objects. Usually, objects are used to represent the physical components in an application with the connections representing physical connections such as pipes or wires. It is, however, also possible to have objects that represent abstract concepts and connections that represent general relations among objects. Variables are a special type of objects for representing parameters whose value vary over time. Variables are either quantitative, symbolical, logical, or textual and have validity intervals indicating the length of time the value remains valid after having been updated. It is also possible to indicate that the validity should be computed from the validity intervals of the variables from which the value is inferred. Other variable attributes determine from where the variable receives its value (e.g. the simulator, the inference engine, or a data server), and whether a history should be kept for the variable or not. G2 contains built-in functions for referencing past variable values and for the usual statistical operations on time series such as mean, standard deviation, rate of change, maximum, minimum, etc...

G2 rules are used to encapsulate an expert's heuristic knowledge of what to conclude from conditions and how to respond to them. Five different types of rules exist. Of them, four are different forms of "**if** conditions - **then** actions" rules.The last type is the "**whenever**" rules which allows asynchronous rule firing as soon as a variable receives a new value or fails to receive a within a specified time-out interval. The rule conditions contain references to objects and their attributes in a natural language style syntax. Objects can also be referenced through connections with other objects. G2 supports generic rules that apply to all instances of a class. The G2 rule actions makes it possible to conclude new values for variables, send alert messages, hide and show workspaces, move, rotate, and change colour of icons, etc... G2 rules can be grouped together and associated with a specific object, a class of objects, or a user-defined category. This gives a flexible way of partitioning the rule-base. The following is an example of a G2 rule:

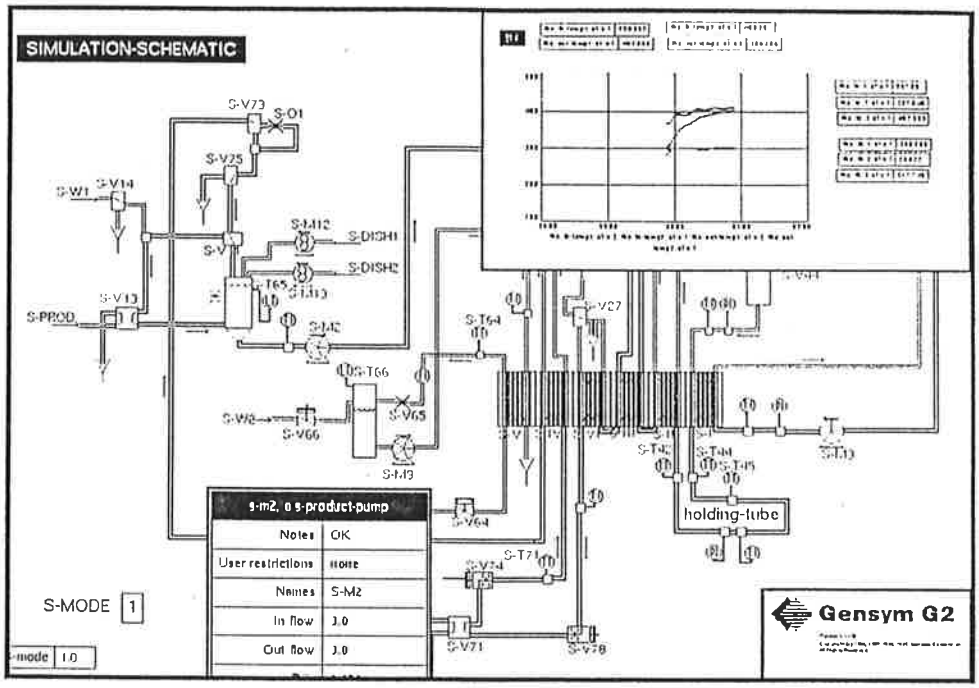

Fig. 2. A typical G2 display screen on Symbolics.

for any water-tank
  if the level of the water-tank < 5 feet **and**
  the level-sensor connected to the water-tank is working
then conclude that the water-tank is empty
  and inform the operator that
  "[the name of the water-tank] is empty"

Dynamic models are used to simulate the values of variables. The models are in the form of first-order difference and differential equations. The models can be specific to a certain variable or apply to all instances of the variable class.

4.3 The G2 Real-time Inference Engine

The real-time inference engine initiates activity based on the knowledge contained in the knowledge base, simulated values, and values received from sensors or other external sources. Apart from the usual backward and forward chaining rule invocation, rules can also be invoked explicitly in several ways. First, a rule can be scanned regularly. Second, by a focus statement all rules associated with a certain focal object or focal class can be invoked. Third, by an invoke statement all rules belonging to a user defined category, like safety or startup, can be invoked. The scanning of a few vital rules in combination with focusing of attention is meant to represent the way human operators monitor a plant. It is also an important way to reduce the computational burden on the system. Regular scanning of rules and thus updating of information in combination with variables with time-limited validity gives a partial solution to the problem of non-monotonic, time-dependent reasoning. The inference engine automatically sends out request for sensor variables that have become non-valid and waits for new values without halting the system. Priorities can be associated with rules.

G2 has a built-in simulator which can provide simulated values for variables. The simulator is intended to be used both during development for testing the knowledge base, and in parallel during on-line operation. In the latter case, the simulator could be used for estimation of signals that are not measured. The current simulator has however limitations. Each first-order differential equation is integrated individually with individual and user-defined step-sizes. This may cause problems. The numeric integration algorithm used is a simple Euler method with constant step-size. Further, the simulator interprets the simulation equations which slows down the system. GSPAN, an interface between G2 and external simulators is available as a separate product.

4.4 The G2 Environment

G2 has a nice graphics-based development environment with windows (called workspaces), popup menus, and mouse interaction. Input of rules and other textual information is performed through a structured grammar editor. Facilities for browsing through the knowledge-base exist. The operator interface contains variable displays such as graphs, meters, readout tables, etc..., and operator controls in the form of different types of buttons and sliders for changing variables and executing rule actions.

The data servers are the interfaces to either conventional control systems or other signal sources such as e.g. databases. The signal sources either run in another process on the same computer as G2 or in a separate computer system. In the latter case the communication is done over Ethernet. Gensym sells a generic interface, GSI, which the user can modify to implement custom interfaces. A goal of Gensym is to provide standard, off-the-shelf interfaces to the major manufacturers' control systems. Interfaces to Fisher Controls, Yohogawa, and Siemens exist and interfaces to Honeywell and Allen Bradley are being developed.

G2 is implemented in Common Lisp and runs on Sun, HP, Dec Vaxstation, TI Explorer and MicroExplorer, Symbolics, Compaq 386 and Mac II. For most machines, 16 Mb RAM memory is required. The price ranges from $ 18.000 to $ 36.000 depending on computer.

For portability reasons, G2 uses their own window system and object-oriented system. To avoid garbage collection, care is taken for G2 not to generate any garbage. Version 2.0 of G2 has been announced for release in September 1989 and will among other

things include improved operator interface facilities and procedures.

Gensym has currently sold around 200 G2 licenses with applications in such areas as process control, robotics, manufacturing, and simulation prototyping. Gensym has also sold around 50 on-line GSI licenses.

5. MUSE

MUSE from Cambridge Consultants in U.K. is a toolkit for embedded, real-time Artificial Intelligence. Muse consists of an integrated package of languages for knowledge representation which all share the same set of database and object structures. The central component of the package is the Pop-talk language. Pop-talk which is implemented in C is derived from the Pop series of languages and has been extended to support object-oriented programming. It is also a stack-based language that combines strong list-processing elements with a block-structured syntax. It is an imperative programming language such as C or Pascal. On the top of the basic object language, a frame (or schema) system is built that includes multiple inheritance, methods, relations and demons.

A Muse application ranges from a simple expert system with just a single database and a single rule-set, to a complete blackboard system with many knowledge sources and databases which co-operate to solve the problem.

5.1 The Muse Knowledge-Base

A major part of Muse is a set of architectural support facilities that allows a complex application to be split-up into modules. The modules include knowledge sources and notice boards. A knowledge-source contains one or more rule sets and a local storage to hold the data it is reasoning with. A notice board is a special case of knowledge-source that is only used for storing data. The databases in a Muse application can be attached to rule-sets, knowledge-sources or notice boards and will normally contain objects. For real-time applications written with Muse, it is important to consider at the design stage how the application can be divided into separate knowledge sources (different rule-sets and databases) as the efficiency of the implementation is derived from ensuring that searches for the rule matches are performed in databases which contain only relevant objects.

The facts are objects which can have several slots, methods, demons and relations. A creation date can be associated to a fact. The time is measured from the beginning of the session in milliseconds. Tests can be based on this time. Ending date, obsolescence time, past values of a variable are not explicitly available but can be built by the user with the use of demons and methods.

Two different rule languages are available. In the Forward Production System (FPS), rules are expressed as "**if** conditions **then** actions". The Backward Chaining System (BCS) uses the format "**rule** name_of_rule(arguments) **provided** conditions". Variables can be used in order to match slots in the condition part of a rule. It is not possible to replace a schema by a variable in a rule. The matching of the rule with the facts in a database is expressed with the operators "**if there is a**" or "**if there is no**". Poptalk functions and expressions (such as conditional branching , loops, arithmetic and mathematics operators, function calls, etc...) can be used in both sides of the rules. Such code can also be included in the schema as methods or in the knowledge sources as demons. In the action part, manipulation of facts (creation, modification and deletion) is allowed.

A typical rule of the FPS is shown below. The goal of the rule-set is to do a simple signal classification. The rule is in charge of identifying parts of the treated signal which are said to be spikes. Segment is a schema which has the slots type, i_start (which contains the instant of beginning of the segment), i_end (which contains the instant of end of the segment). Variables begin with a capital letter and comments are marked by two vertical signs.

```
|| name of the rule:
|| id_spike_spike.
if
   there is a segment S -type spike_unknown,
      -i_start Istart, -i_end Iend where (abs(Iend - Istart) =< 10)
then
   assert {segment S: -type spike}
```

A typical rule of the BCS is shown below. Its goal is to search a way from one point A to another point B. Such a way exists either if B is equal to A or if there is another point Y such that there is a road which starts in A and finishes in Y and there is a way from Y to B. In this example, road is a schema with a slot "starting point" and a slot "ending point".

```
fact way_from(A,A)

rule way_from(A,B) provided
   there is a road -start A, -end Y and
   way_from(Y,B)
```

Files written in C can also be included as linked files. External procedures can be called by creating a new pop-talk function and building the interface between pop-talk, a C program of the linked library and the external procedure. The arguments are passed in a stack.

5.2 The Muse Real-Time Inference Engine

The control of the knowledge-sources is handled by the agenda. This is an ordered list of things to execute, typically but not necessarily knowledge-sources. When the agenda is empty, a scavenger function spreads notifications of changes around to the knowledge sources. This may then cause knowledge-sources to be scheduled to run anew. Ten levels of priority can be associated to a knowledge source or to schedulable events and the execution of such events can be interrupted by another events with higher priority.

The FPS inference engine is based on a modification of the RETE algorithm. The BCS is Prolog-type backward chaining rule system that support depth-first backtracking, unification of logical variables on standard Muse objects, and flow control via 'cut' and 'fail'.The rules for the backward chaining must explicitly be written.

Incremental garbage collection is essential for real-time applications allowing garbage collection to be handled as a background task while Muse is running, rather than temporarily completely halting the application.

Muse is interfaced to the external world through data channels which allow periodic acquisitions of external information and data.The physical implementation of the data channels depends on the underlying hardware and software. For example, on a UNIX system the data channels are UNIX sockets. The data channels provide filter functions that only allow through the particular pieces

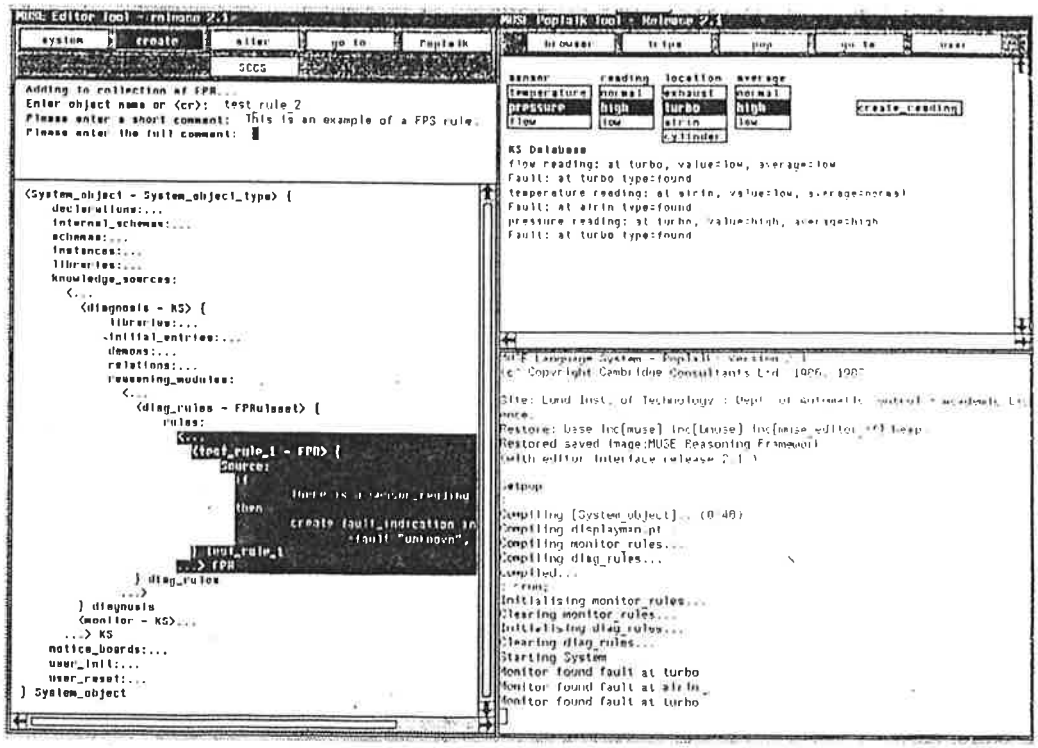

Fig. 3. A typical MUSE display screen on SUN3.

of information that the applications decide are needed. These filters are user-definable and on the lowest level implemented in C.

5.3 The User Interface

The interface consists of four windows. A typical Muse screen display can be seen in Figure 3. The first window contains in a structured editor which allows the user to create, edit and modify Muse source code. As all Muse applications can be viewed as collections of objects (including the rules) and the structured editor is aware of the types of structures available in Muse, the editor is able to guide developers in the production of correctly-formed Muse objects. The second window is the run-time browser. Here, the user can examine, within a running application, the Muse internal data structures and also, with the help of a menu interface, use some important Muse functions such as the debugging system. These two run time tools allow the display of the contents of the database and also the modification of Muse objects. The third window is a pseudo-terminal interface on to the Muse language and the last window is the Emacs text editor from which the user also can edit the application code.

Muse is currently running on Sun 3/4 workstations. On this machine, it requires at least 12 Mb of free memory. Applications developed in Muse are compiled to a compact intermediate code which can be packaged with a run-time support kernel and downloaded to a dedicated target machine for testing or final installation. At the beginning of the year 1989, the cost ranged from £ 15 000 for a single machine license to £ 25 000 for a network license. Around 30 licences have been sold mainly within the U.K.

6. COMPARISON

6.1 Management of the Real-time Problems of KBSs

All the problems implied by using an expert system in a real-time application are not resolved by these tools. However, they are a first step toward this use. Indeed, the three tools allow:

* acquisition of asynchronous events and external information and data,
* automatic update of sensor values and facts,
* reasoning interruption.

The tools use different approaches to resolve the different problems concerning real-time expert systems described in the first section.

Non-monotonicity:

The approach used in G2 is to attach validity intervals and time stamps to facts and to inferred facts. This technique can only represent that a fact has become invalid due to passed time. It cannot represent that a fact becomes invalid due to an occurred event.

In Muse, this problem is neither considered nor treated. The user has some tools to resolve it - such as the creation date of a fact, demons, methods, etc ... - but, he must build his own solution.

The approach proposed by Chronos is a possible answer to this problem. Validity intervals and time stamps are attached to all acquired and inferred facts. This represents that a fact has become invalid due to passed time. The rule *as long as - then -* represents that a fact has become invalid due to an occurred event because all the time stamps joined to a fact can be modified and this modification is immediately considered by the expert system.

Reasoning under time constraints and high performance:

In these tools, the problem of high performance is only treated, on the constructor viewpoint, by saying that their tools are very efficient and that the management of tests with regard to current time is optimised. In fact, the existing real-time shells have no means

to cope with a guaranteed response time.

One approach taken in Chronos is the possibility to associate a deadline to the expert system. If this time is overpassed, the reasoning is stopped. We can notice that a similar approach can be built with Muse.

This problem is far from being resolved but the user can not expect that a tool will ever be able to manage alone with the time constraints. The great evolution of the computer performances will allow faster and faster reasoning. A solution which can be built temporarily by the user in order to deal with these time constraints consists, for instance, in the building of several knowledge-sources or special rules which have to cope with special cases as emergency cases. Different hierarchical abstraction levels of knowledge can be also defined in order to give a very fast answer when it is needed with a high level of abstraction or a very accurate answer with a deeper level (*Krijgsman, Verbruggen, Bruijn, 1988*). Another point of view is to say that the user can deal with this problem if its on-line expert system is closer to a decision table (one or two rules are fired when the system is in a given situation) than to a real reasoning system with the chaining of many rules.

Missing and uncertain data:

Missing data can be treated in G2 with the use of the built-in simulator provided that a simulation model has been defined for the variable. Chronos and Muse can also resolve this with a predetermined rule which calls a simulator included in an external procedure.

Uncertain data is a problem which is quite difficult to cope with even in an off-line expert-system. Different methods exist for computing the certainty of the inferred facts from the certainty of the conditions and the rule certainty. The result of different methods may vary substantially. Another problem with certainty-based expert systems is how to correctly determine the initial certainties of the measurements.

None of the tools use certainty factors.

Temporal reasoning:

The time stamps associated with facts in Chronos and G2 allow a first step in temporal reasoning.

Both Chronos and G2 allow reasoning about the past since variable histories may be kept. The built-in history functions for numerical values in G2 are useful.

With Chronos, the possibility to enter facts that will become true in the future allows the user to build a reasoning about the future. The built-in simulator of G2 allows another kind of reasoning about the future.

Focus of attention:

Only G2 includes such a treatment. A similar thing can be built in Chronos and Muse with the creation of specific rules and facts' attributes or slots.

Continuous operation:

Both Chronos and Muse perform the garbage collection as a background task that is evenly spread out over the execution. However, when the computing load is close to the maximum level, the systems do not have time to perform this task. After a certain time this leads to suspension of the reasoning.

G2 handles the memory allocation and reallocation internally during run-time to avoid generating any garbage at all.

6.2 Summary

The flexible pattern-matching facilities of Chronos allows rules which treat many different cases. This leads to rather small KS which can also time efficient. The manipulation of time within this tool is easy to use and powerful. The tool is very easy to use at the beginning. The user becomes familiar with its possibilities quite rapidly.

However, Chronos is a purely rule-based system restricted to forward chaining. The system provide no means for partitioning the knowledge base except by explicitly associating a special context attribute to the facts and the rule. The system is not object oriented, although this has been foreseen to be available for the beginning of 1990.

G2 is the most widely spread system of the three. The graphical development and end-user interfaces are very pleasant to use and also quite powerful. The built-in simulations possibilities are useful both in the development phase and for dynamic on-line simulation. The interest among control system vendors to provide off-the-shelf interfaces to G2 is interesting.

The real-time inference engine and the natural language style rule syntax which do not distinguish between forward and backward chaining rules are powerful. The object-orientation which allows inheritance and rules and simulation equations that apply to classes of objects are also powerful although multiple inheritance and methods and demons are not allowed.

G2 is, however, also a quite closed system. Due to the method for avoiding generating any memory garbage, the user is not allowed to add any LISP code to the system. Calls to user-written C and Fortran code can be made but not on all machines that G2 runs on. More advanced communication with other programs must be made through the GSI interface. As in the case of Chronos, version 1 of G2 is a rule-based system. For more advanced applications procedures are extremely important.

The blackboard architecture of Muse is powerful especially for large applications. The modularised facilities makes it possible to develop the different knowledge sources independently. It also makes it easier to modify, read, and understand the system than if a single knowledge base was used. Muse is closer to a general programming environment that an expert system shell with all the advantages and disadvantages this implies. Muse is flexible and powerful. Object-oriented programming can be combined with procedures, and rules. Methods and demons are allowed and external C procedures can be called.

The disadvantage is that it takes some time to learn the system. Moreover, the call of external procedures is not so easy to realise: a C interface must be built and a stack is used to manipulate the subroutine arguments. Another drawback due to the use of a blackboard architecture is that all modifications of facts are not compulsory immediately notified to the external knowledge sources (special instructions must be used and they are much less time efficient). These modifications will be spread in all the databases by the scavenger which has the lowest priority of the system.

The real-time facilities in Muse are less developed than in G2 and Chronos. The two inference engines used have no special

real-time features. Instead this is taken care of by the agenda which decides which knowledge source to run. The possibility to download onto a smaller target machine is important for embedded systems where compact and inexpensive hardware is important.

6.3 Table

The features of the different tools are summarised in table 1. Six different domains are distinguished: the inference engine, the rules, the fact representation, the functions related with time, the development interface and the operator interface. The speed of the different tools is difficult to compare since it depends a lot on computer configuration.

A simple croos means that this feature will be available soon. A double cross means that this feature is not available for all the computers.The expression "uniform rule syntax" means that the same rules are used for the forward and the backward chaining. "General procedures" means that a knowledge source may be replaced by procedural code which reacts as an ordinary knowledge-source would do. A multiple-valued fact is used in expressions as e.g.: "associated_valve(reactor_1)". The high level programming constructs include for instance arrays, lists, etc... The time statistical functions include for instance the computation of the mean of a variable, its slope, etc... The operators controls are parts of the run-time browser such as buttons, sliders, etc...; they allow an on-line interaction between the expert system and the operator. The auto-explanation generation contains a chronogram of acquired and deduced facts, a display of the rule firing, a justification of deduced facts. "Interface to external systems" means that an interface with other systems (as e.g. control systems) is provided.

7. Conclusion

Commercial expert system tools for real-time, on-line applications are now emerging. In this paper we have tried to compare three systems based on how the specific real-time aspects are handled. Each of the systems have their advantages and disadvantages. Several problems also remains to be solved.

All three tools are intended for general real-time applications such as monitoring, diagnosis, and planning. It must, however, be remembered that the tools in themselves do not give explicit support for any kind of application. What they provide are different knowledge representation and inferring facilities. They do not provide any higher-level generic problem solving agents for, e.g. model-based process diagnosis.

TABLE I
MAIN FEATURES OF EACH TOOL

		Chronos	G2	Muse
ENGINE	Reasoning interruption	*	*	*
	Asynchronous event or rule invocation	*	*	*
	Periodic rule testing	*	*	
	Periodic data acquisition	*	*	*
	Forward chaining	*	*	*
	Backward chaining		*	*
	Guaranteed response	*		
	Incremental garbage collection	*		*
	Guaranteed continuous operation		*	
	Focus of attention		*	
	Blackboard architecture			*
	General procedures		+	*
RULES	Uniform rule syntax		*	
	Natural language rule syntax		*	
	Rule priorities	*	*	*
	Call of ext. procedure in condition part		++	*
	In the action part:			
	Procedural rule actions	*		*
	Call of external procedures	*		*
	Fact creation and deletion	*	+	*
	Fact modification	*	*	*
	Rule invocation	*	*	*
FACTS	Attibutes or objects as variables	*		
	Multiple-valued facts	*		*
	"Attribut(Object)=Value" representation	*		
	Object oriented	+	*	*
	Message passing			*
	Multiple inheritance			*
	Demons,methods,relations			*
	High level programming constructs			*
	Certainty factors			
TIME	Time stamps	*	*	
	Validity intervals	*	*	
	Reasoning about future events	*		
	Variable histories	*	*	
	Time statistical functions		*	
	Current time manipulation	*	*	*
	As long as operator	*		
	Delayed rule actions	*		*
DEVELOPMENT	Structured syntax editor	*	*	*
	Debugger			*
	Break points		*	*
	Tracing	*	*	*
	Step-by-step execution	*		
	Time metering		*	
	Operator controls		*	*
	Built-in simulator		*	
	Multiple windows	*	*	*
	Menus	*	*	*
OPERATOR	Detection of engine saturation	*		
	Graphical Icon-based interface		*	
	Information browser		*	*
	Animation and color		*	
	Variable displays		*	
	Interaction during execution	*	*	*
	Autoexplanation generation	*		
	Run time version	*		*
	Interface to external systems		*	

\+ : Available soon
++ : Not for all the computers

References

Chronos: Chronos documentation set: Installation guide, presentation guide.

Chronos 1988: Example of monitoring a process, presented at the commercial exposition of the congress "Expert Systems and Their Applications" Avignon 88.

G2: G2 documentation set.

Muse: Muse documentation set: concepts, tutorial, system, reference.

M.J. Chantler (1988): "Real-Time Aspects of Expert Systems in Process Control", Computing and Control division, presented at the Colloquium in "Expert System in Process Control",March 28, 1988.

C.L. Forgy (1982): "RETE: A fast algorithm for the many pattern / many object pattern match problem.", Artificial Intelligence Vol 19, nº1, pp 17-37, 1982.

M.A. Kramer and F.E. Finch (1989): Fault Diagnosis of Chemical Processes Knowledge-Based System: Diagnosis, supervision and control, Ed.Spyns G. Tzafeslas, Plenom Press N-Y.

Krijgsman et al (1988): A.J. Krijgsman, H.B. Verbruggen, and P.B. Bruijn: "Knowledge-based real-time control", presented at the IFAC Workshop on A.I. in real-time control, Swansea (U.K.).

Laffey et al (1988): T. Laffey, P. Cox, J. Schmidt, S. Kao, and J. Read: "Real-Time Knowledge-Based Systems", A.I. Magazine, Spring 1988, pp. 27-45.

E.H. Mamdani and S. Assilian (1975): "A fuzzy logic controller for a dynamic plant", Int. journal of Man-Machine Studies, vol. 7, pp. 1-13, 1975.

Miles et al (1989): J. Miles, J. Daniel, D. Mulvaney: "Real-Time Performance Comparison of a Knowledge-Based Data Fusion System using Muse, Art and ADA", presented at Avignon 1989.

Moore et al (1987): R.L. Moore, L.B. Hawkinson, M. Levin, A.G. Hoffmann, B.L. Matthews and M.H. David: "Expert system methodology for real-time process control", Proceedings of the 10th IFAC Workshop on A.I. in real-time control, Swansea, Sep. 21-23.

Årzen K.E (1987): Realisation of Expert System Based Feedback Control, Ph.D. thesis CODEN: LUTFD2/TFRT-1029, Dpt of Automatic Control, Lund Institute of Technology, Sweden.

Åström et al (1986): K-J. Åström, J.J. Arton and K-E. Årzén, "Expert control", Automatica, vol. 22, nº 3, pp. 277-286, 1986.