Yugoslav Journal of Operations Research
3 (1993), Number 1, 43–60

REASONING WITH EXTERNAL DATA

Vladan DEVEDŽIĆ

Mihajlo Pupin Institute
Volgina 15, 11000 Belgrade, Yugoslavia

Dušan VELAŠEVIĆ

School of Electrical Engineering
University of Belgrade
Bulevar Revolucije 73, 11000 Belgrade, Yugoslavia

Zoran BOŽOVIĆ

Center for Multidisciplinary Studies
University of Belgrade
Slobodana Penezića Krcuna 35, 11000 Belgrade, Yugoslavia

Abstract. An important feature of current expert system technology is the possibility of reasoning with real–time data generated outside of expert system itself. The purpose of this paper is to describe some issues of such reasoning. The description is supported by the experience gathered through the implementation of an expert system development tool, specifically designed for building real–time expert systems capable of reasoning with external data. Therefore the paper focuses on practical implementation algorithms and techniques, not merely on design. The issues covered include knowledge base organization, models of the reasoning process, reasoning speed, communication between the inference engine and the outside world in order to exchange real–time data, the independent operation of the inference engine and the outside processes, reasoning algorithms, timeliness of decisions made by the inference engine, and potential bottlenecks in the reasoning process

Key words: Real–time expert systems, reasoning, asynchronous external information, expert system shell.

1. INTRODUCTION

Reasoning with external data is featured by including current values of signals generated by an independent process outside the reasoning mechanism itself, and by providing intelligent response to the external input in the form of output signals. An expert

---

Communicated by D.Starčević

system (ES) reasoning with external data uses three sources of information: knowledge base, internal facts from the working memory, and external data generated at run–time. As a result of its operation, two types of output information can be generated: user–oriented information and environment–oriented information (output signals and actions). An important class of problems in reasoning with external data are real–time inference problems. By "real–time inferencing with external data" we mean inferencing that can provide intelligent, knowledge–based responses to unforeseen, asynchronous external events within some predefined response time, short enough not to allow "the luxury of off–line advice or consultation" [5], [9]. The response time for real–time reasoning mechanisms is the time it takes from the occurrence of an external event until the time the reasoning mechanism produces a conclusion about the event [5].

Developing an ES capable of reasoning with external data requires special tools to be used. Such tools should satisfy several requirements not typical for ordinary ES building tools [2], [6], [8], [9], [10], [14], [17]. First, *reasoning speed* is much more critical for such ESs than for consultant type ESs, particularly in real–time reasoning. A real–time ES building tool must include an inference engine capable of making timely decisions, so that both the ES and the outside processes run normally. Second, tools for building such ESs should always include *appropriate knowledge representation techniques* that make possible for knowledge engineers to represent dynamics and changes in the outside world. External input data are often fleeting, transient signals, and must be *sensor–based*, i.e. received from the outside world over electronic channels. Next, such tools must have an *I/O module* for communication between the inference engine and the outside world. Once a conclusion is reached by the inference engine, a real–time ES capable of reasoning with external data does not stop, i.e. there is a continuous change in the external input and hence the ES should continuously produce an output. Real–time ES building tools should also have a large *bandwidth* regarding the I/O activity, since there are many applications that require intensive processing of external data, which bears the danger of contention. Finally, such tools should make possible to build ESs capable of *autonomous operation*. A real–time ES capable of reasoning with external data must perform complex actions in response to sensed environment with minimal guidance by the operator. This includes the possibility to

interrupt the current activity immediately upon receiving the external input, then execute some predefined code, and finally resume the original activity.

The purpose of this paper is to systematically present our experience gathered through the implementation of a real–time ES development tool called *ASIA*. The tool has evolved from REX, a special–purpose shell for building ESs in the domain of robotics [1], and FLEX, a similar tool for building ESs in various domains in the field of Flexible Manufacturing Systems [3]. ASIA stands for ASynchronous Information Access, and is the name originally given to the pattern–matching algorithm used in FLEX. The name refers to the fact that in real–time ES applications external processes run and generate real–time data asynchronously and independently regarding the inference engine operation. The ASIA shell is based on generalization and several modifications of the two special–purpose tools, on a study of architectures for real–time inference engines [4], [5], and on an improvement of the ASIA pattern–matching algorithm. This paper focuses on design decisions, algorithms and techniques used in the implementation of the shell prototype. Apart from our own experience with real–time ESs capable of reasoning with external data, results and achievements of other real–time ES shell developers [6], [9], [12], [14], [15], [16] have been used as well in developing the ASIA prototype.

The following Section gives an overview of the ASIA shell. Specific knowledge base organization of an ASIA–based ES and the relations between the knowledge base elements featuring reasoning with external data and those needed for symbolic processing are discussed. It is also shown how various models of the reasoning process and the external world can be included in an ASIA–based ES. Then the ASIA's reasoning mechanism is presented, from top–level design to more specific implementation issues, followed by a short description of the I/O module. The final Section briefly discusses some performance issues.

2. THE ASIA SHELL

The prototype of the ASIA shell has been developed on a 80386 personal computer, in C language. Its global design is shown in Figure 1. Note that unlike conventional ES shells ASIA has two active modules, the inference engine and the I/O module. They are implemented as independent processes. Apart from the I/O module all of

the other modules feature conventional shells as well: the inference engine, the module for knowledge acquisition and modeling, the module for representing and storing the knowledge base internally, the ES developer interface, and the module for generating the end–user interface. The interface modules and the knowledge acquisition modules are rudimentary in the prototype, and are subject of further development. The other modules and functions are fully implemented.

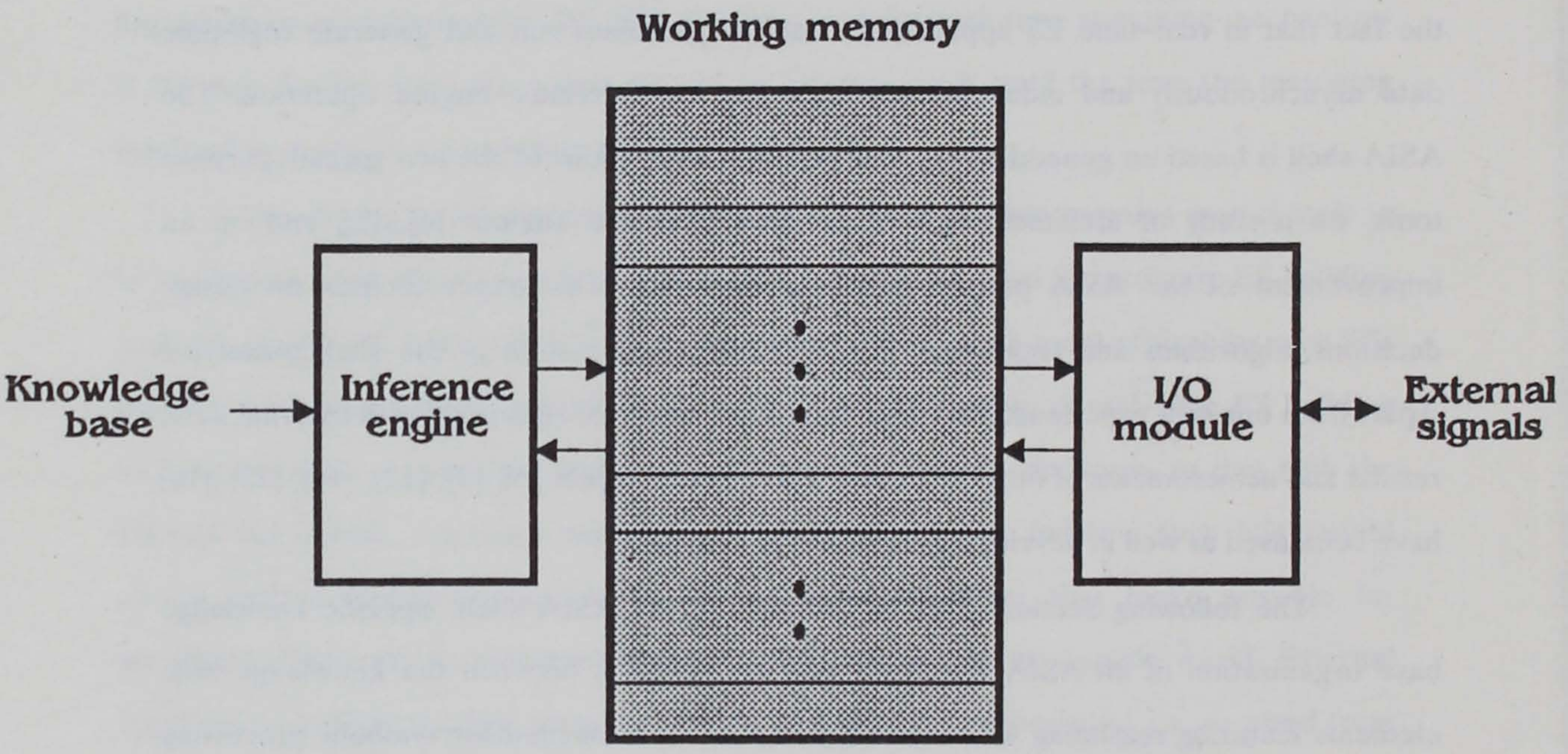

Figure 1. Modules of the ASIA shell

Figure 2 illustrates the most essential run–time activity of an ASIA–based ES regarding the process of reasoning with external real–time data. The inference engine and the external process exchange real–time information via a shared part of the working memory. The I/O module communicates with the sensor system and other sources of external real–time information, presumably present in the ES environment, by means of I/O ports. It converts input signals into *working memory elements (WMEs)*, which are readily accepted by the inference engine and used in the inference process. The I/O module also converts output WMEs, generated by the inference engine as a result of the inference

process, into output signals used by the external process. Another output activity of the I/O module can be initiating an action or calling an external procedure that performs an action in the outside world.

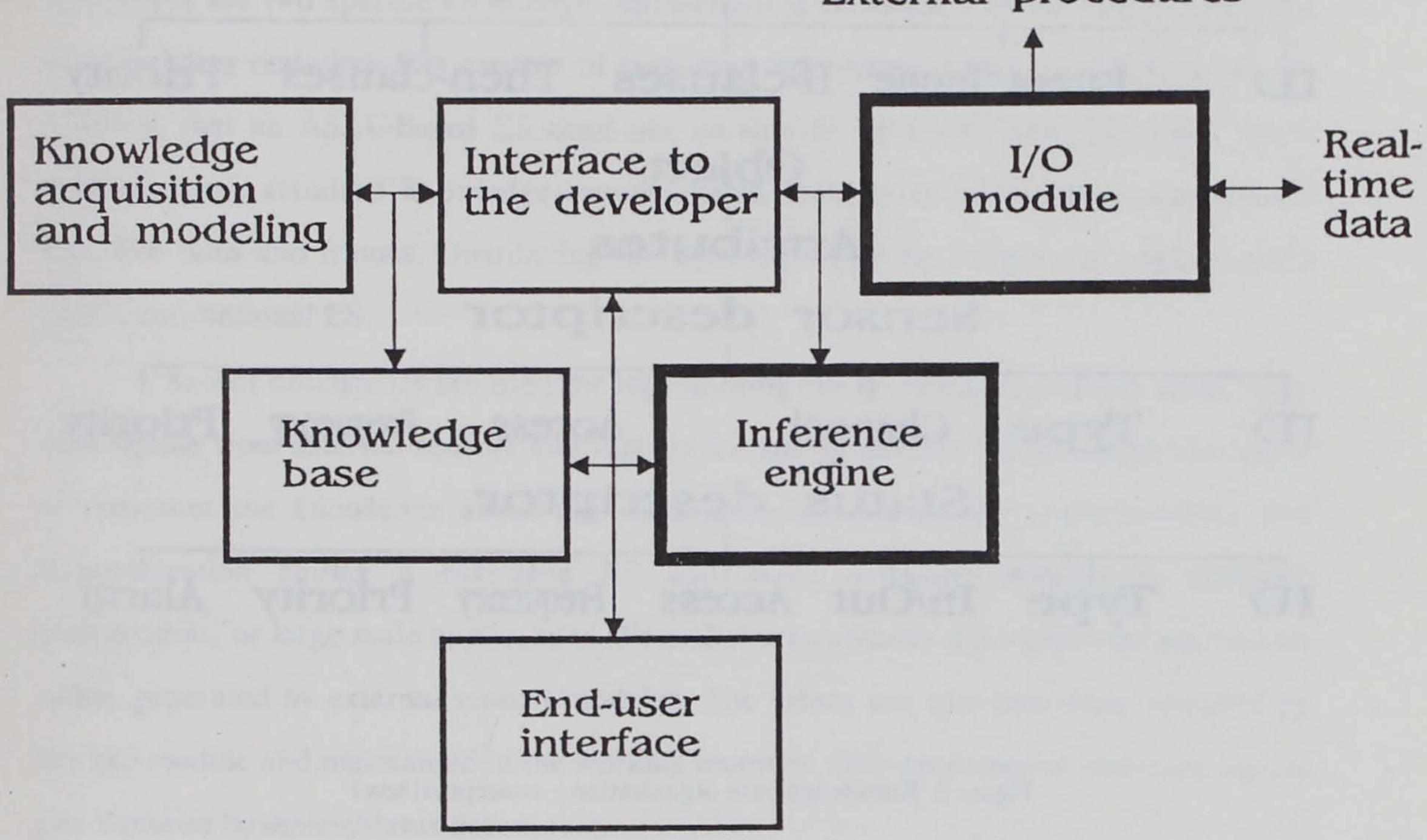

Figure 2. Real–time information exchange

3. KNOWLEDGE BASE ORGANIZATION

3.1. CONCEPTUAL LEVEL

Figure 3 shows a general case of an ASIA–based ES knowledge base organization at the conceptual level. The knowledge is distributed by four kinds of knowledge and data entities (structures): *frames, objects, sensor descriptors, and status descriptors*.

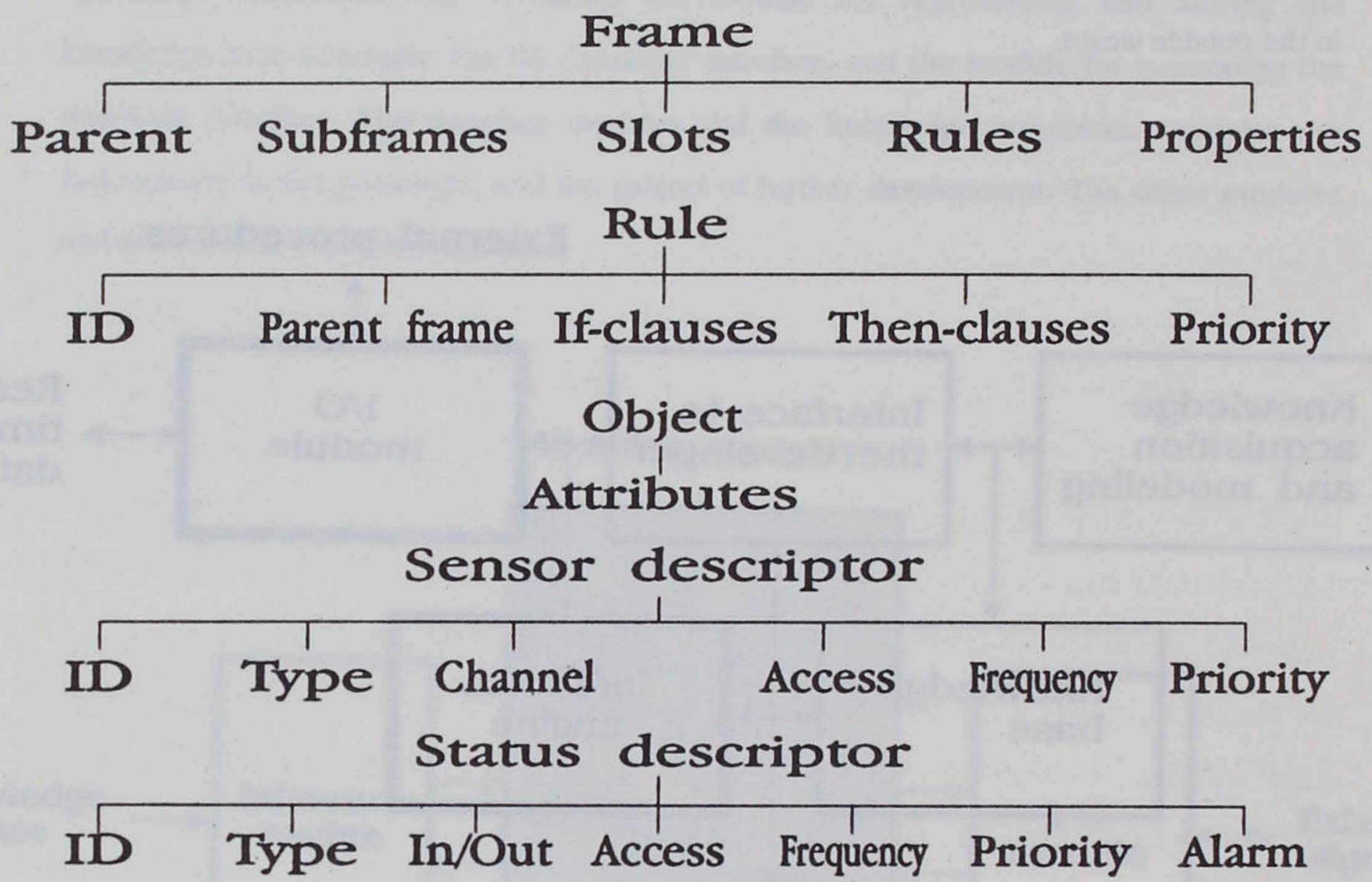

Figure 3. Knowledge base organization – conceptual level

Frames and Rules. Each frame is associated with a parent frame, a set of subframes, a set of slots, a set of rules, a set of demon functions, and several user–specified properties, like a unique name (identifier), allowed number of instantiations, etc. Production rules are distributed by frames, and each rule is featured by a set of If–clauses, a set of Then–clauses, the parent frame, and a set of properties, like a unique rule name (ID), rule priority, etc. Each ASIA–based ES must have at least one frame in its knowledge base (the *root* frame; a built–in default root frame is available and can be used as it is or modified) with at least one slot (the *goal* slot). Rules are not strictly necessary. However, the only currently supported means of reasoning with external real–time information is to refer to such information within rules. Demon functions and subframes are optional.

Objects. In our notation objects refer to unique structured data records, describing single and static physical or conceptual entities, whose attributes have constant values. Objects differ from frame instantiations, which are specific examples of frames (classes), whose slots can be filled by many different values.

Sensor Descriptors and Status Descriptors. Sensor descriptors and status descriptors are two specific knowledge representation techniques of the ASIA shell used when building real–time ESs capable of reasoning with external data. It should be noted, however, that an ASIA–based ES need not necessarily be a real–time ES, since ASIA supports some standard knowledge representation techniques for building conventional ESs, like rules and frames. Distributing the knowledge only by frames and objects would yield a conventional ES.

Sensor descriptors are used for representing the knowledge and facts about real–time signals from external sensors and various measuring devices. Status descriptors allow to represent the knowledge about real–time information used for synchronization and communication among a real–time ES and other software modules in complex, multimodule, or large scale applications. Note that sensor/status descriptors do not contain *values* generated by external sensors/modules. The values are run–time data, acquired by the I/O module and maintained in the working memory. Only *properties* of real–time signals are captured by sensor/status descriptors.

Properties of sensor signals are an identifier (ID), data type (Type), the I/O address of the electronic channel (Channel) and a pointer to the low–level I/O procedure (called Access-procedure) used for receiving values from the corresponding physical sensor device, the estimated frequency with which the sensor data value changes, and a measure of importance of the sensor data value changes in the application (Priority).

Some properties of status data correspond to the properties of sensor data (like identifier, data type and priority). The others feature status data only. It should be stressed that, unlike sensor data, status data can be both input and output (In/Out). Input status information is generated on–line, by other modules, and is used by the ES. Output status data values are generated by the ES reasoning process, and are sent to the other modules as a status information about the part of the system represented by the ES (see [2] for more details). The Access procedure of an input status data element is a user defined procedure,

which is used to transfer the actual status information generated by a particular external module to the ES. On the other hand, output status data Access procedures transform relevant feedback data generated by the ES into output status information required by other modules.

Another distinguishing property of status data are the Alarm conditions. When the Alarm conditions of input status data are established, they signalize an alarming situation in the external world, detected by another module of the entire system. Likewise, the Alarm conditions of output status data are the conditions under which an alarming output signal would be generated for the other modules. Depending on the contents of the knowledge base, an ASIA–based ES may try to handle the alarming situation, or it may interrupt or abort the current inference process. Whether it is better to handle the alarming situation or not to act at all is strongly application dependent. Sometimes the time spent for reasoning further can avert future disaster, but there are cases when that time is critical for safety and survival of the application [13]. Therefore alarming conditions are treated separately, not merely as high priority signals.

3.2. Implementation Level

The knowledge base of an ASIA–based ES at the implementation level is a set of files each one containing an ordered series of structured records. Each record describes a single knowledge base element, and has a pointer to its parent element (if any), and its children elements (if any). All elements of the same kind are contained in a unique file. The number of files in a particular knowledge base varies depending on the application, since a knowledge engineer may not need to use all possible kinds of knowledge and data elements for a certain ES. There is also a "contents" file, specifying what elements/files exactly compose the knowledge base. All files composing the knowledge base have identical names and different extensions associated with different knowledge and data elements in the files.

Figure 4 shows a general case, when all possible kinds of knowledge and data elements are used. Kinds of knowledge and data elements associated with each file are specified in the following list:

AKB – contents (ASIA Knowledge Base)
FRM – frames

SLT – slots
RUL – rules
IFC – If–clauses
THC – Then–clauses
NOD – RETE network nodes
OBT – objects
OAT – object attributes
SEN – sensor descriptors
STA – status descriptors
NAM – names (identifiers)
TRA – English translations of names (identifiers)
PPT – prompts for the user
ATM – atomic values
MOD – models of reasoning and the external world

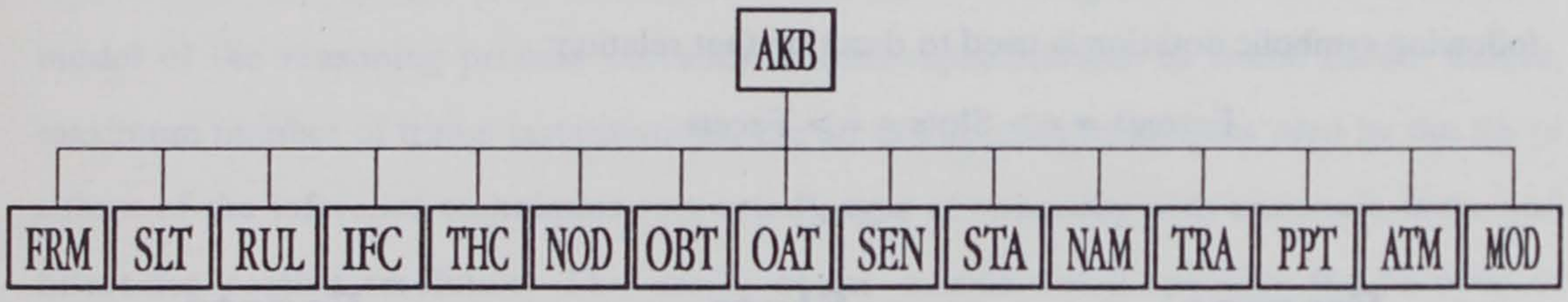

Figure 4. Knowledge base organization – implementation level

The minimum set of files that must be contained in every ASIA–based knowledge base includes the AKB, FRM, SLT, NAM, and TRA files. The other files are optional. In practice, that means that at least the description of the root frame is contained in the FRM file, and at least the goal slot is described in the SLT file. Every element defined in the knowledge base must have a unique name (identifier). The names are grouped in the NAM file. English translations of the names are correspondingly grouped in the TRA file and are used by the inference engine for interaction with the end user of the ES.

When developing a knowledge base, the user naturally defines knowledge and data elements at random order. In the end, all of these elements are automatically sorted and ordered so that they can be more easily compiled for use by the inference engine at run–time. After sorting and compiling the knowledge base, it is saved as a set of files from the

above list, and is read into memory when the ES is initialized for running. The NOD file is generated automatically as a result of compiling rules from the knowledge base. ASIA's inference engine uses an extension of the RETE algorithm for pattern–matching [7], [4], [5]. The NOD file contains the nodes of the extended RETE network.

The regular, pointer–based structure of the knowledge base in memory at run–time can be explained by Figure 5. It shows how frames, slots, and facets are interrelated. In the table of slot records, slots of each single frame occupy a continuous range of entries. The first range corresponds to the first frame in the table of frame records (the root frame), the second range corresponds to the second frame (the first subframe of the goal frame), and so on. By analogy, the values of all instantiations of all facets of a single slot occupy a continuous range of entries in the table of facet values. The first range corresponds to the first slot in the slot table, the second range corresponds to the second slot, and so on. The following symbolic notation is used to describe that relation:

Frames ==> Slots ==> Facets

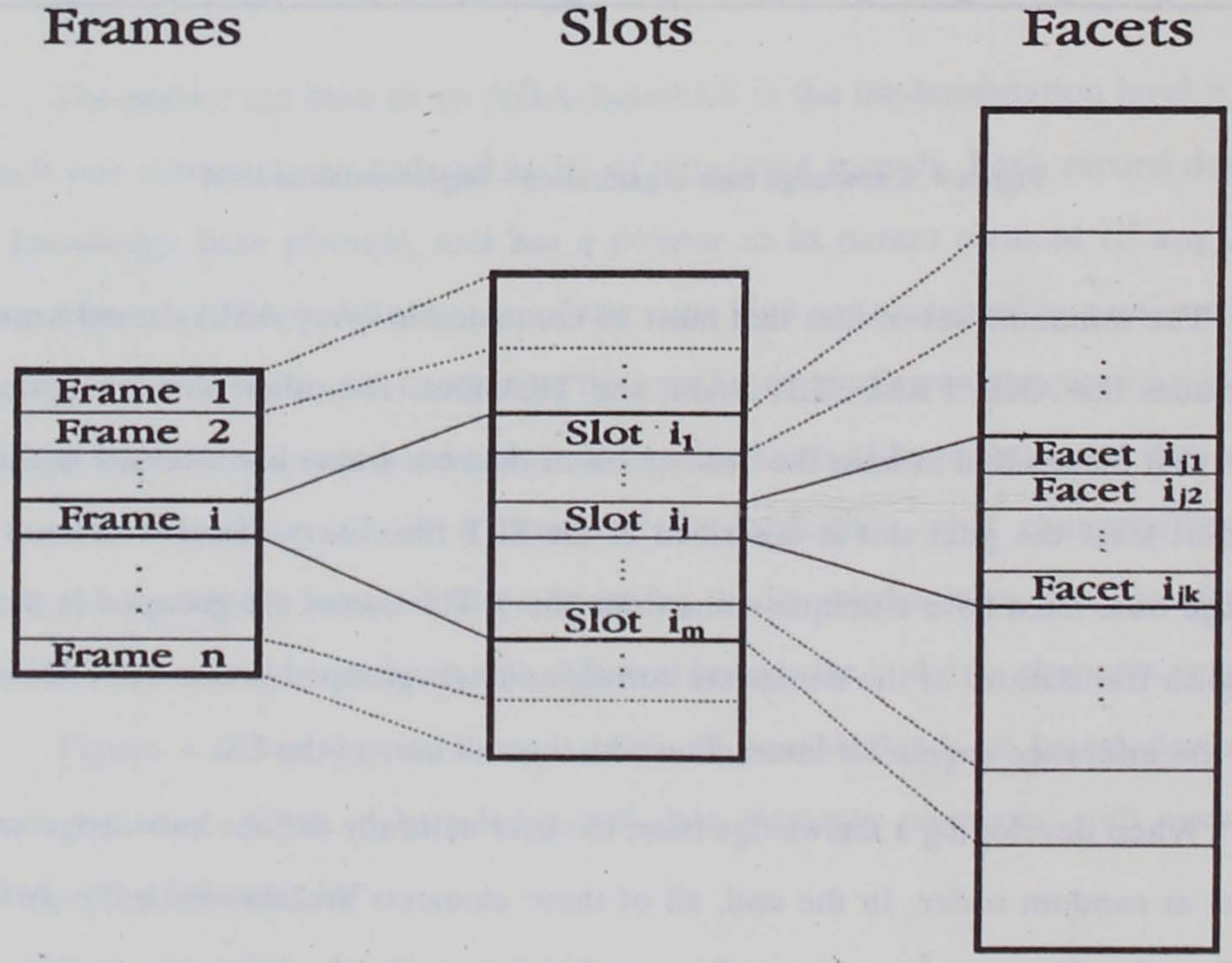

Figure 5. Relations among frames, slots, and facets at run–time

Using this notation, the following set of relations holds for sorted and compiled knowledge base of an ASIA–based ES at run–time:

Frames ==> Slots ==> Facets
Frames ==> Rules ==>
Frames ==> If–clauses of the corresponding rule sets
Frames ==> Then–clauses of the corresponding rule sets
Frames ==> RETE network nodes
Objects ==> Object attributes

All of the knowledge and data elements discussed above are actually implemented as C structures.

Depending on the application, it is possible to use various *models* of the reasoning process and the outside world in an ES capable of reasoning with external data. In an ASIA–based ES these user–defined models are stored in the MOD file, and are used drive both further knowledge base development and the reasoning process at run–time. The model of the reasoning process currently includes specifications of frame access modes, maximum number of frame instantiations, the list of inference techniques used by the ES (a subset of the inference techniques supported), type of reasoning with uncertain facts, and filters for conflict resolution. The model of the outside world specifies the number of sensors in the external sensor system, the set of discrete values that can be assigned to the Frequency field of sensor and status descriptors, the policy of handling alarming output status signals, and the list of outside modules that the ES is supposed to communicate with in a large scale application.

4. INFERENCE ENGINE

This Section describes top–level design of ASIA's inference engine and discusses some implementation details regarding inference techniques, control, and real–time reasoning.

4.1. MULTIPLE INFERENCE TECHNIQUES

A major objective of ASIA's inference engine is to provide multiple inference techniques and let the user select those that he/she believes would best fit his/her application [1]. In that sense, an important property of each frame slot is the list of inference techniques that the inference engine uses when trying to infer the slot's value. The

list is user–specified, and the inference engine tries the techniques in the order specified in the list. It is the user's choice to specify from one to as many as available techniques in any desired order. In this way maximum inference flexibility is achieved for every single slot. The inference techniques supported include rule–based inference, inheritance, default values, procedural calculations, and interaction with the user.

4.2. CONTROL

At the top control level, ASIA's inference engine always uses a frame as its operating environment. It tries to infer the value of the frame's goal slot. If there are multiple goal slots, it has to find out the value for each one. The inference process may recursively pass to subframes, if the values of their slots are needed. When inferring the value of any single slot, the inference engine uses only the techniques specified in the list for that slot, as discussed above. Figure 6 shows the pseudocode of that control structure.

4.3. REAL–TIME REASONING

A specific forward chaining strategy for rule–based inference is used in the prototype version of ASIA's inference engine for reasoning with real–time data [4], [5]. Forward chaining has been chosen because of the efficiency of several well developed forward chaining pattern–matching algorithms [7], [11], [14]. All of these algorithms are based on maintaining the run–time state of the ES in the working memory over the inference cycles and processing only changes to that state detected in the working memory from one cycle to another. Since asynchronous external real–time data actually generate changes in the working memory (see Figure 2), the inference engine can process these changes in much the same way as those changes that were generated internally.

Pattern–matching. The RETE algorithm for forward chaining pattern–matching (readers not familiar with the RETE algorithm should refer to [7]) used as the basis of the ASIA's inference strategy. The RETE algorithm is extended by defining two new node types to deal with external real–time data: one for testing values of input signals, and one for including a rule instantiation into the conflict set if firing the rule would set up the value of

```
InferGoalValue(Frame)                    /* Top-level inference control procedure */

{
        Init;                            /* Set up initial data in working memory */
        G = GetGoalList(Frame);                  /* Create list of goal slots */
        for each  g = = member(G)
            {
                GetValue(Frame, g, &Done);
                if not Done
                        ReportFailure(Frame, g);
            }
        ReportSuccess(Frame);
        return;
}

GetValue(Frame, SI, &Done)               /* Inferring value of slot SI */

{
        Done = FALSE;
        i = 1;
                                                 /* Get next allowed inference technique */
Next:
        InfTechnique = SI.InfTechnique[i];
        if InfTechnique = = NONE
                return;
        case (InfTechnique)
            {                                    /* Apply technique */
                RULES:
                        RulGetValue(Frame, SI, &Done);
                INHERITANCE:
                        InhGetValue(Frame, SI, &Done);
                PROCEDURE:
                        ProcGetValue(Frame, SI, &Done);
                DEFAULT:
                        DefGetValue(Frame, SI, &Done);
                ASK_USER:
                        AskUser(Frame, SI, &Done);
            }
        if Done
                return;
        else
            {
                i = i + 1;
                goto Next;
            }
}
```

Figure 6. Top–level inference control procedures (pseudocode)

an output signal (see [5] for details). To include real–time data in the reasoning process, appropriate sensor and status descriptors must be defined in the knowledge base and reference to them must be made within the inference rules. The extended RETE network, compiled from the inference rules, includes the nodes that correspond to these references. When the nodes corresponding to the references to external input signals are processed by

the network interpreter, current values of the external input signals are read as WMEs from the shared part of working memory. This requires a memory–lock access, in order to prevent the I/O module to access the same location simultaneously.

Conflict resolution. ASIA's conflict resolution strategy is based on the principle that high priority external signals should be processed immediately. Therefore, rule instantiations containing references to and/or inferences about high priority external signals are selected for execution prior to other rule instantiations in the conflict set.

Rule firing. Executing an action in a rule's Then–part that contains setting–up an output status value implies updating the shared part of the working memory. As with reading WMEs from the shared part of the working memory, setting–up an output status WME must be performed as a memory–lock operation.

5. I/O MODULE

The pseudocode in Figure 7 shows how ASIA's I/O module is implemented. Only a simplified version of top–level control of the continuous I/O process is shown in Figure 7. Detailed description of the implementation of lower–level I/O functions is skipped for the sake of clarity. It includes a number of I/O port access procedures.

6. PERFORMANCE

Current version of ASIA is implemented on a personal computer and targets small– to medium–scale applications (up to a few tens of external signals). Large–scale applications would require porting ASIA to more powerful hardware platforms.

Our experiments with ASIA have shown that it can be used for building ESs capable of responding to asynchronous input within 0.5 to 1 second, provided that switching from one frame to another at run–time does not come too frequently. Frequent frame switching slows down the system operation because of a relatively large amount of data that must be pushed to / popped from the stack each time two frames are switched. On the other hand, we found out that frames can be very useful for problem structuring and decomposition. Fortunately, our analysis of some domains for ES application featured by reasoning with external data has shown that most such ESs do not require large and complex knowledge bases, and usually no more than 2 or 3 frames need to be defined.

Consequently, frequent frame switching is not likely to appear in practical problems. Pointer–based structure of the knowledge base also alleviates the problem of frame switching.

```
IO( )                                   /* Top-level I/O procedure */
{
        Init;                           /* Initialize shared part of working memory */
    CreateSenList(SenL);                /* Create list of sensor descriptors */
    CreateInStatList(InStatL);      /* Create list of input status descriptors */
    CreateOutStatList(OutStatL);/* Create list of output status descriptors */
    do forever;
        {
                                        /* Read sensors */
            for each  Sen  = = member(SenL)
                {
                    GetSenValue(Sen.Port, Sen.Access, &Value);
                                        /* Convert to WME */
                    UpdateWM(&Lock, Sen, Value);
                }
                                        /* Read input status */
            for each InStat in InStatL
                {
                    GetInStatValue(InStat.Access, &Value);
                                        /* Convert to WME */
                    UpdateWM(&Lock, InStat, Value);
                }
                                        /* Send output status */
            for each OutStat in InStatL
                {
                                        /* Convert from WME */
                    ReadWM(&Lock, OutStat, &Value);
                    SendOutStatValue(OutStat.Access, Value);
                }
        }
}
```

Figure 7. Top–level procedure of the I/O module (pseudocode)

Another potential performance bottleneck is the shared memory contention problem. The I/O module is separated from the inference engine, and the necessary memory–locked access to the shared part of working memory can lead to performance degradation if the number of input and output signals is large. Fortunately, there are many real world applications where the number of external signals does not tend to be large [8]. ASIA is currently well suited only for that kind of applications.

In our experiments, performed on a 80386/25 computer without a coprocessor, the inference engine and the I/O module kept up processing up to 5 sensor signals and 4 status

signals, with an average period of changing of about 0.3sec. The experiments have been performed with a test ASIA–based ES, and all of the input signals were driven by random number generators in order to insure quasi–asynchronous conditions at the input of the I/O module. It should be stressed that comparison of these results with the results achieved using other approaches is hardly feasible, since most results published so far either hold for particular cases or have been achieved as parts of case studies using other hardware environments [8], [12], [14], [16]. An analytical performance evaluation is the subject of our current research.

7. CONCLUSIONS

Reasoning with external data requires specific knowledge representation and inference techniques. When using the ASIA shell for building real–time ESs capable of reasoning with external data, the nature and the properties of external real–time data can be described in the knowledge base by means of sensor and status descriptors, ASIA's specific techniques for representing external signals. Reference to sensor and status descriptors in rules implies acquiring input signals from the outside world at run–time and including their values into the inference process, as well as generating output status signals as inference results. Apart from sensor and status descriptors, ASIA supports multiple standard knowledge representation and reasoning techniques as well. Its run–time active modules, the inference engine and the I/O module for communication between the ES and its environment, are implemented as two independent processes that share a common part of the working memory. The two processes interchange real–time information via the shared memory.

The paper describes practical algorithms and techniques used in the implementation of ASIA's prototype. The description concentrates on the algorithms and techniques essential for real–time reasoning with external data. A brief discussion on the shell's modeling capabilities and experiments in the course of performance evaluation show that ASIA is well suited for practical applications with the required response time to the external input from 0.5 to 1 second.

REFERENCES

[1] Devedžić,V., "REX: Robotics EXpert Systems Building Tool", *Engineering Applications of Artificial Intelligence*, Vol.2, (December 1989), 286–299.

[2] Devedžić,V., "A Knowledge–Based System for the Strategic Control Level of Robots in Flexible Manufacturing Cells", *The International Journal of Flexible Manufacturing Systems*, Vol.2, No.4, (1990), 263–287.

[3] Devedžić,V., and Velašević,D., "Building Real–Time Expert Systems with FLEX", Schildt,G.H., and Retti,J., (eds.), *Dependability of Artificial Intelligence Systems*, Amsterdam: North–Holland, (1991), 321–327.

[4] Devedžić,V., and Velašević,D., "An Architecture for Real–Time Inference Engines on Personal Computers", in *Proceedings of The Hawaii International Conference on System Sciences*, Kauai, Hawaii, Vol.1, (January 7–10, 1992), 619–630.

[5] Devedžić,V., and Velašević,D., "Fast Reasoning with External Data on Personal Computers", *Engineering Applications of Artificial Intelligence*, Vol.5, No.2, (1992), 87–99.

[6] Dulieu,M.R.W., and Morgan,A.J., "COGSYS – A Real Time Expert Systems Generator", in *Proceedings of The 1990 DECUS Europe Symposium*, Cannes, France, (September 3–7, 1990), 51–56.

[7] Forgy,C.L., "RETE: A Fast Algorithm for Many Pattern / Many Object Pattern Matching Problem", *Artificial Intelligence*, Vol.19, (1982), 17–38.

[8] Laffey,T.J., Cox,P.A., Schmidt,J.L., Kao,S.M., and Read,J.Y., "Real–Time Knowledge–Based Systems", *AI Magazine*, Vol.9, No.1, (Spring 1988), 27–45.

[9] Leinweber,D., "Expert Systems in Space", *IEEE Expert*, (Spring 1987), 26–36.

[10] Mahmoud,M.S., Eid,S.Z., and Abou–Elsoud,A.A., "A Real Time Expert Control System for Dynamical Processes", *IEEE Transactions on Systems, Man and Cybernetics*, Vol.19, No.5, (September/October 1989), 1101–1105.

[11] Miranker,D.P., "TREAT: A Better Match Algorithm for AI Production Systems", in *Proceedings of 1987 National Conference on Artificial Intelligence, AAAI*, (August 1987), 42–47.

[12] ParsonD.E., and Blank,G.D., "Constant–Time Pattern Matching for Real–Time Production Systems", in *Proceedings of SPIE Applications of Artificial Intelligence VII*, Part 2, Orlando, Florida, (March 28–30, 1989), 971–982.

[13] Payton,D.W., and Bihari,T.E., "Intelligent Real–Time Control of Robotic Vehicles", *Communications of The ACM*, Vol.34, No.8, (August 1991), 48–63.

[14] Sabharwal,A.S., Iyengar,S.S., de Saussure,G., Weisbin,C.R., and Pin,F.P., "Asynchronous Production Systems for Real–Time Expert Systems", in *Proceedings of 8th Int. Workshop on Expert Systems and Their Applications*, Avignon, France, May 30. – June 3., Vol. Specialized Conferences, (1988), 113–146.

[15] Schor,M.I., Daly,T.P., Lee,H.S., and Tibbitts,B.R., "Advances in RETE Pattern Matching", in *Proceedings of 1986 National Conference on Artificial Intelligence, AAAI*, (August 1986), 226–232.

[16] Wang,G., and Dubant,O., "BUOSHI: A Tool for Developing Active Expert Systems", in *Proceedings of SPIE Applications of Artificial Intelligence IX*, Part 1, Orlando, Florida, (April 2–4, 1991), 11–15.

[17] Wolfe,A., "An Easier Way to Build a Real–Time Expert System", *Electronics*, March 4., (1987), 71–73.