IEEE Pacific Rim Conference on Communications, Computers and Signal Processing, May 9-10, 1991

DAME: An Expert Microprocessor-Based-Systems-Designer. An Overview and Status Report†

N.J. Dimopoulos, K.F. Li, E.G. Manning,
B. Huber, M. Escalante, D. Li, and D. Caughey

Department of Electrical and Computer Engineering
University of Victoria
Victoria, B.C.

ABSTRACT

DAME (Design Automation of Microprocessor-based systems, using an Expert system approach) is an expert system capable of configuring and designing a customized microprocessor system from original specifications. We have postulated that such an expert system, can be easily constructed, since most of the interfaces used by the various microprocessors and related peripherals are standardized. Thus, once the gross structure of the design and the modules comprising it have been chosen, their interconnection is fairly straight-forward. Our investigation into the modelling of signal behavior confirmed this premise.

In this work, we present the component library and knowledge base in DAME. A notation is developed to allow the complete specification and the modelling of the static and temporal behaviors of signals found in these components. Knowledge representation using frames in a hierarchical structure is presented, together with some typical rules used for the design process. Examples of the hierarchy, frames, and rules to design interfaces between microprocessor and memory components are illustrated.

1. Introduction †

Knowledge-Based Systems (KBSs) have recently proliferated in several fields of human endeavor. These systems play the dual role of categorizing and codifying expert knowledge, and then using this knowledge in order to solve time consuming and/or challenging problems. Examples can be drawn from several diverse fields such as patient care [10,16], geological exploration [7], etc.

Computer system design and synthesis is a very complex task that involves a large search space, requires problem-dependent design strategies, and is a designer-dependent process. Since the early 1980's, several KBSs for computer systems have been developed. DAA [11] and ASP [1] are prime examples of KBS for hardware synthesis at the register transfer level. In addition, silicon compilers that use a combination of algorithmic and knowledge base approaches have become available [14,15].

At the system architectural level, R1 was developed to configure computer systems but no design knowledge was involved [12]. CMU's Micon is a system synthesis tool able to assist the designers in configuring single board system from commercially available components according to customers requirements [2]. Micron bases its design on matching the interfaces between compatible components. This design knowledge is encapsulated in family-specific templates which describe the relationships between the processor, other components, and the board level bus.

DAME (Design Automation of Microprocessor-based systems, using an Expert system approach) [3,4,5,6,8,9] is an expert system that will be capable of configuring and designing a customized microprocessor system from original specifications such as type and application, environment, communication and computational requirements, as well as economic criteria. DAME attempts to exploit the general design methodologies using generic communication and interfacing protocols, and to adjust and fine-tune the details according to the specific components' timing requirements and their interfacing properties. The objective is to configure the interfacing of different components intelligently once they are selected; in order to accomplish this, it employs rules which operate on the abstract properties of a component rather than their instantiations.

DAME organizes the design process into a hierarchy [5,8] consisting of the following phases: (1) Design Specification; (2) Configuration; (3) Behavior Description; (4) Functional Block Design; (5) Implementation and Integration.

Each hierarchical level represents an abstraction of the given design problem. As the levels are transversed, the abstraction of the design is refined, until, at the last level, the complete design is formed. Each hierarchical level manipulates objects which represent the system's concept of the design requirement at the particular abstraction level.

Our basic tenet has been that the interface signals found in various microprocessor families, follow a limited number of well defined protocols for information exchange. This information is given both descriptively and quantitatively as timing diagrams by the manufacturers. The components' descriptions include references to these basic protocols for information exchange, which are instantiated for each component.

The inclusion of the references to these basic protocols in the description of our components provides a powerful model in that we are able to carry out the design process by employing a limited number of general rules which are specific to the protocols used.

†This research has been supported in part by the Natural Sciences and Engineering Research Council of Canada under the strategic grant STR0040526 and by the Science Council of British Columbia under Science and Technology Development Fund through grant SCBC #88 243.

DAME comprises a library of available components (the knowledge base), the rule base, and the user interface. The rule base uses information from the library in order to choose the appropriate components and to eventually produce a valid design. The library of components incorporates such diverse information as names, signal protocols and timing, as well as packaging and availability. We have chosen the frame paradigm [17] to organize this diverse information. This was necessitated because of the diversity and repeatability of the information. In addition, several expert system tools provide a frame development environment. For example, Knowledge Craft™ uses schemata to structure its data.

A model is needed to encapsulate the specifications of microprocessor components. The data describing the component include information required at each design phase. The components, the capabilities of components, their signals and signal timings are some of the objects included in the representation. The notation and data structures developed for these objects are presented in Section 2. A component editor to aid the knowledge engineer in the entering of pertinent information is presented in Section 3. Abstract interface design is described in Section 4. Examples of knowledge and rule representations, and the design obtained of a bus arbitration subsystem are given in Section 5.

2. Library of Components

DAME includes a library of components. The component model is specific enough to allow the complete design to be accomplished, including the interconnection and the interfacing of incompatible modules. Components in DAME are represented as networks of schemata. Networks of schemata that represent components within a class (e.g., microprocessors, memories, peripherals, etc.) have identical structures. Such networks of schemata are generally organized hierarchically with more detail becoming evident as the hierarchy is transversed.

A major portion of such structures is devoted in describing the ways that information can be communicated to/from the components. This includes the protocols involved as well as timing and electrical characteristics of the participating signals.

The "information exchange" part of the description is considered as a collection of capabilities that describe abstract capabilities of the component (e.g., data transfer). Each capability is refined into a collection of protocols which describe the capability's functions and operations. For example, the data transfer capability of a processor component comprises the *read*, *write* and *read-write* protocols while the data-transfer capability of a Read Only Memory includes the *read* protocol only.

A protocol contains details of the fundamental operations it describes. It comprises several timing parts, typically including the control and the transfer parts. The transfer part, may be further distinguished into address, and data transfer, or a combination of both. These timing parts describe in detail timing relationships of the participating signals. The timing parts are instantiations of timing templates which are generic relationships (patterns) describing timing relations between the transitions and the interactions of abstract signals. A study of a variety of timing diagrams from microprocessor specifications [5,9] has produced a number of such templates as well as protocols and capabilities.

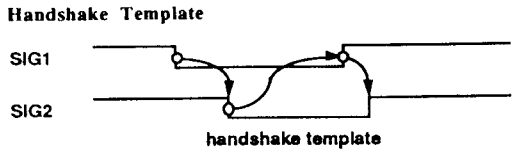
The whole concept of DAME is based on the above observation concerning "information exchange". The capability to relate the description of a component to a well-defined structure of abstract objects enables us to produce a design in the abstract space using a limited number rules, and subsequently instantiate the design with the specified components. The following describe the different levels of knowledge representation, starting at the lowest in the hierarchy.

2.1 Event Description Language

At the lowest level of the representation, signal transitions are related to events which must precede them and are considered for our purposes as their causes [8]. An event description language has been developed through which arbitrarily complex events, collections of signal transitions with imposed timing constraints and precedence, can be described. In addition, causal relations, relating events to transitions, can also be expressed. We use this language in order to encode the information in the timing diagrams provided for by the manufacturers.

2.2 Control and Transfer Protocols

At the second level of the representation, collections of limited numbers of causal relations describe basic control or transfer protocols, using the description language. An example is the two signal handshaking protocol found in many information transfer subsystems.



! ASSERTED SIG1 —> ! ASSERTED SIG2 @ T1
! ASSERTED SIG2 —> ! NEGATED SIG1 @ T2
! NEGATED SIG1 —> ! NEGATED SIG2 @ T3

2.3 Standard Behaviors

At the highest level of the representation, collections of basic protocols as they exist at the second level, constitute Standard Behaviors. In general, a Standard Behavior describing the capability is comprised of two parts: the control part which describes the behavior of the control signals participating in this behavior, and the information transfer part which describes the actual information transfer.

Examples of Standard Behaviors are data transfer behaviors such as the ones found in processors and which incorporate both a control protocol and the actual transfer protocols necessary for the delivery of addresses and data on a bus. In contrast, the data transfer behavior of static memory components are devoid of the control protocol. This absence of a control protocol illustrates the inability of a memory component to initiate activity.

Protocols as well as Standard Behaviors are depicted as semantic networks (networks of schemata in Knowledge Craft™). Templates of networks of schemata depicting generic protocols and behaviors have been constructed. These templates are customized when they are used in the description of the various components within the component data base to reflect the particular signals and timing constraints involved.

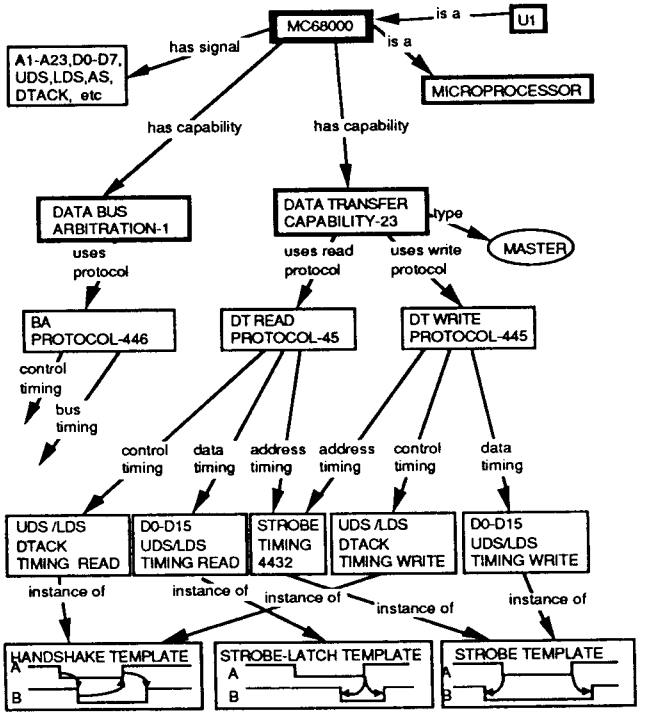


Figure 1. MC68000 Microprocessor Component Representation Hierarchy.

Fig. 1 depicts a partial network of the schemata used to describe the MC68000 component. This partial hierarchy details the data transfer capability of the component as specified by the MC68000 manual [13]. The timing parts shown in Fig. 1 are instances of timing templates which correspond exactly to the specified timings of the component.

3. The Component Editor

Many available expert system shells provide powerful editors that can be used to structure networks of schemata representing components. However, due to their generality, these editors cannot exploit the inherent regularity of our structures.

Since the number of components in the library is quite large, all of which are represented as instantiations of regularly structured networks of schemata, we are developing a customized editor which can exploit this regularity in order to guide and expedite the construction of the component library.

Because of the regularity of the structure, at each level of the hierarchy, there are but a few alternate objects (represented as networks of schemata) that can be selected for inclusions in the component. The editor therefore offers to the users the legal alternatives, and subsequently it incorporates the selected network of schemata to the component.

The editor is based on a mouse driven graphical user interface. The user is able to edit any schemata in the component hierarchy through a consistent user interface. A schemata has a name and contains several slots. To provide the user with complete information about the schemata being edited, several items are visible at all times: its name; its parent's name; its slots; contents and permitted contents of a selected slot; the component name currently being edited; and the current schemata's position in the component hierarchy. A typical window of the editor for the MC68000 displaying the above information is shown in Fig. 2. The component and the schemata currently being edited is MC68000 and its parent is the schemata MICROPROCESSOR.

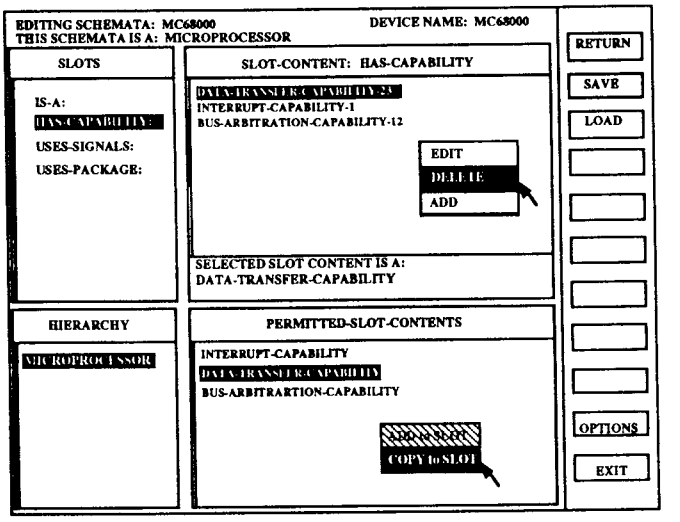


Figure 2. Schemata Edit Window for a Microprocessor Component.

Pertinent information and properties of the component are represented as the slots of the schemata: IS-A, HAS-CAPABILITY, and HAS-SIGNAL. Contents of a selected slot are also shown. The editor allows users to manipulate (EDIT, DELETE and ADD) the slot contents. The editor is also able to display and modify legal slot contents which are the legal templates available for the specific slots. To make the user aware of which is the current schemata being edited, a hierarchy window displays a trace of the path from the current schemata to the device root node.

4. Abstract Interface Design

4.1 Bus Protocols

In general, computer systems comprise several interconnected modules that include processors, memory and I/O. These modules communicate through private or shared buses that carry both information and control. In addition, well defined protocols manage the orderly access and information transfer. Buses are divided into the following general categories: *address*, *data*, *control*, *status*, *arbitration*, *interrupt*, *serial*, *ID*, and *timing*.

Modules utilizing a bus can be characterized as *masters* and/or *slaves* as well as *requestors* or *responders*. Bus masters can obtain the use of the bus and initiate data transfers, while bus slaves, merely can participate in a transaction if requested by a master. A requestor requests services, while a responder responds to such requests.

The method used by the bus for signaling its valid usage is called a *bus protocol*. Bus protocols are classified as *synchronous*, where all modules involved use a common clock; *asynchronous*, where the protocols are self-timed; and the hybrid *semisynchronous*.

4.2 Bus Arbitration Interface Design

In the subsequent, we shall address the problem of designing a bus based system comprising a number of masters sharing a common bus. Normally, a bus provides a protocol through which a single master module can be determined at any time. An arbiter adhering to the bus protocol is attached to the bus either as a dedicated module or it is distributed among the participating modules. Modules

capable of requesting the bus adhere to the requestor part of the protocol, while the arbiter adheres to the responder part of the protocol. There are but a few choices of such protocols. They are distinguished as 2- or 3-signal arbitration protocols. Arbitration structures can be centralized or distributed.

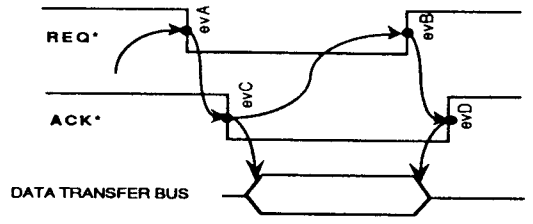


Figure 3. 2-signal Bus Arbitration Protocol (Requestor part).

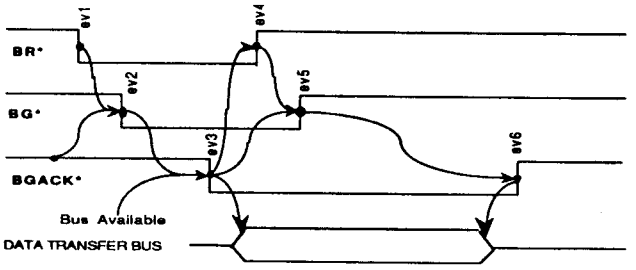


Figure 4. 3-signal Bus Arbitration Protocol (Responder part, e.g., MC68000).

Figures 3 and 4 illustrate the 2-signal and 3-signal arbitration protocols through their timing diagrams. The 2-signal requestor protocol is a fully responsive asynchronous handshake protocol between REQ* and ACK*. REQ* is an output signal indicating, when asserted, a request to use the bus, and when negated the end of the current bus use. ACK* is an input signal that when asserted informs the device that the bus is available, and when negated that the transaction is terminated.

In the corresponding 2-signal responder part, the same REQ* and ACK* signals are present, but in their dual role. REQ* is an input to the arbiter indicating, when asserted, that there is at least one requesting device, while the ACK* is an output indicating that the bus is available for the next master to use.

In the 3-signal Bus Arbitration protocol the input signal BR* indicates, when asserted, that an external device is requesting the bus. The arbiter responds to this request by asserting BG* if BGACK* is not asserted (which indicates that the previous fully responsive asynchronous transaction has ended). The external arbitration mechanism then has to decide which device is to take the bus. Once a device is selected, it must release its BR* and it has to wait until the bus is available (BAV* asserted) before starting using the bus. The device informs that it is the owner of the bus by asserting BGACK*. Finally the selected device signals the end of its transaction by negating BGACK*.

4.3 A 3-signal Multi-master System Example

As an example, we shall present the structure of a system, where several master modules share a bus employing a 3-signal arbitration protocol.

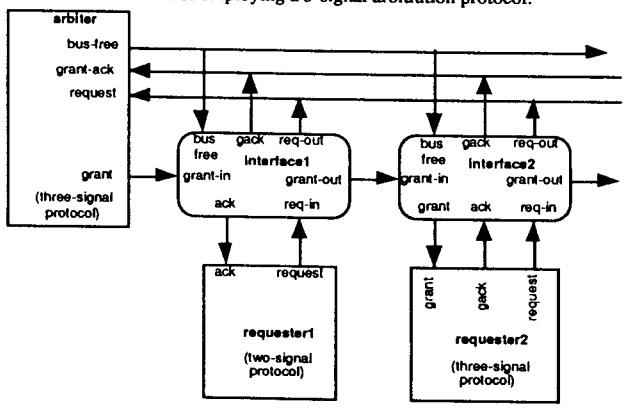


Figure 5. The Structure of a Multi-master System.

We shall assume that the bus is driven by a 3-signal arbiter, and that the modules are arranged in a daisy chain reflecting their relative priorities.This example reflects the structure of systems which may incorporate a VME bus, an MC680XX processor as the lowest priority master and bus arbiter and several DMA devices. 2- and 3-signal modules are interfaced in a daisy chained fashion to a 3-signal arbitration bus. The general structure is given in Fig. 5.

The primary responsibility of an interface module, is to propagate the grant in a daisy chain fashion and to acknowledge the grant when it is received and while the request from their corresponding master is asserted.

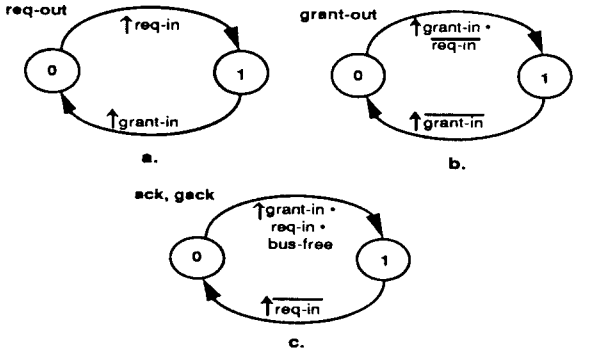


Figure 6. The sequential machines comprising Interface 1.

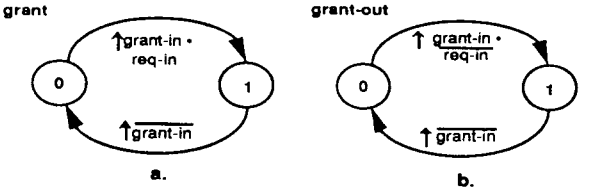


Figure 7. The sequential machines comprising Interface 2 (ACK and REQ-IN are wired directly to GACK and REQ-OUT respectively).

The sequential machines necessary for the generation of the correct signals are given in Figs. 6 and 7.

As it can be seen (Fig. 6a), the bus request (REQ-OUT) is asserted by the interface in case that the corresponding module is requesting (REQ-IN asserted), the grant is propagated if the corresponding module is not requesting (Fig. 6b) while the grant is "arrested" by the interface block and acknowledged in case that the module is requesting.

5.0 Representation

As was described in Section 2, components are represented as networks of schemata. The bus arbitration capability comprises both a requestor and a responder protocols. The design knowledge base is structured in clusters of rules pertaining to specific aspects of the design. The design process is organized in a hierarchical manner and proceeds through a continuous refinement of the structures arrived at the previous layers of the hierarchy. Clusters of rules at particular layers are activated to carry the design forward. For example, in the functional block design layer, there exist clusters of rules dealing with the design of the arbitration subsystem. Figure 8 presents such a cluster of rules.

Consistent with the design philosophy of DAME, these rules apply to the abstract design necessitated by the presence of particular types of protocols in the participating modules rather than the specific instantiations of the protocols. The knowledge base is capable of instantiating the abstract design based on the instantiations of the protocols in the modules.

This technique of abstraction, allows us to use but a limited number of powerful general rules rather than a plethora of specific rules which apply to specific instances of participating components.

Interface blocks are produced complete with the description of their function and their interconnection. Subsequent layers (implementation layer) are responsible of implementing the functionality of these blocks in a particular technology.

5.1 Arbitration Subsystem Design Example

As an example, we present the design produced by DAME for a system incorporating the VME bus, an MC68000 processor acting as the lowest priority master as well as driving the arbitration of the bus, and three DMA devices arranged as a daisy chain.

The overall structure of the system is identical to the one shown in Fig. 5 while Fig. 9 shows some of the resulting schemata describing the produced interface blocks, their functions and interconnections.

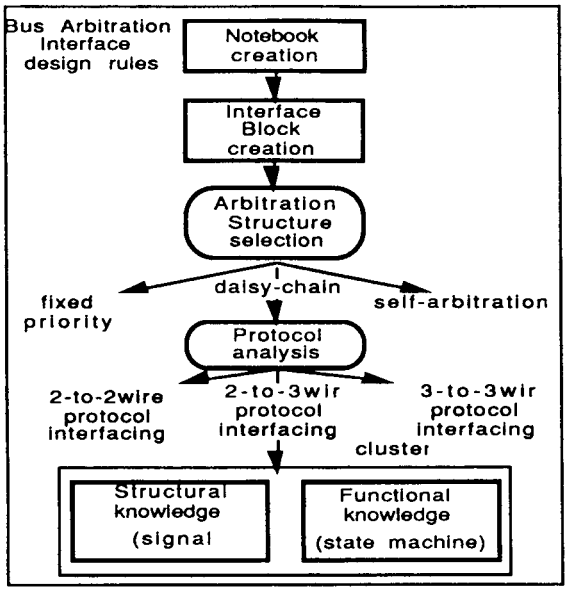


Figure 8. The Structure of the Cluster of Rules that Accomplish the Bus Arbitration Subsystem Design.

6. Conclusion

In this work, we provided the framework of DAME which is an expert system capable of configuring and designing customized microprocessor based systems from original specifications. We presented our approach in modelling the behavior of the various signals associated with microprocessor components. Notations and templates for typical timing specification are presented with examples shown for the MC68000. We used objects and relations to capture both the static and temporal behavior of these signals. We implemented these objects by using *schemata* as defined in Knowledge Craft™ and stored it in a component library.

A component editor was presented which allows the knowledge engineer to easily enter new component information into the component library or to modify existing components. Taking the advantages of the regularized structure, the editor can guide an inexperienced user and expedite the entry procedure. This is accomplished by providing the user with several valid choices of objects that can be entered at each level of the component hierarchy. We have started formulating the design rules necessary for interfacing the various modules based on a bus system.

References

[1] Baldwin, D., "A Model for Automatic Design of Digital Circuits," Technical Report 188, University of Rochester, Department of Computer Science, July 1986.

[2] Birmingham, W.P., et al., "The Micon System for Computer Design," *IEEE MICRO*, pp. 61-67, October 1989.

[3] Dimopoulos, N.J. and H.C. Lee, "Experiments in Designing with DAME: Design Automation of Microprocessor Based Systems using and Expert Systems Approach," *Proc. of Intern'l Computer Symp.*, pp. 1858-1868, Tainan, Taiwan, Dec. 1986.

[4] Dimopoulos, N.J., C.H. Lee and N. Galatis, "DAME: Automated Design of Microprocessor based Systems, an Expert Systems Approach," *Proc. of the Can. Conf. on Industrial Computer Systems*, pp. 20-1/ 20-7, Montreal, Canada, May 1986.

[5] Dimopoulos, N.J., Huber, B., K.F. Li, D. Caughey, M. Escalante, D. Li, R. Burnett, and E.Manning, "Modelling Components in DAME," *Proc. of the 3rd Intern'l Conf. on Industrial & Engg Applications of Artificial Intelligence & Expert Systems*, Vol. II, pp. 716-725, Charleston, South Carolina, July 15-18, 1990.

[6] Dimopoulos, N.J., K.F. Li, and E.G. Manning, "DAME: A Rule Based Designer of Microprocessor Based Systems," *Proc. of the 2nd Intern'l Conf. on Industrial & Engg Applications of Artificial Intelligence & Expert Systems*, vol. 1, pp.486-492, Tullahoma, Tennessee, June 6-9, 1989.

[7] Duda, R.O., P.E. Hart, K. Konolige and R. Reboh, "A computer-Based Consultant for Mineral Exploration," *Tech. Report*, SRI International, Sep. 1979.

[8] Huber, B., K.F. Li, N.J. Dimopoulos, D. Li, R. Burnett, E.Manning, "Modelling Signal Behavior in DAME," *Proc. of the 1990 Intern'l Symp. on Circuits and Systems*, New Orleans, La., Vol. 2, pp. 1497-1500, Apr. 29 - May 3,1990.

[9] Huber, B., M. Escalante, D. Caughey, N.J. Dimopoulos, K.F. Li, D. Li, and E.G. Manning, "Microprocessor Components and Signal Behaviour Modelling in DAME," *Proc.s of the 1990 Can. Conf. on Elec. and Comp. Engg.*, Vol. 1, pp. 27.1.1-27.1.4, Ottawa, Canada, Sep. 4-6, 1990.

[10] Hudson, D.L., and T. Estrin, "EMERGE - A Data-driven Medical Decision Making Aid," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-6, pp. 87-91, Jan. 1984.

[11] Kowalski, T.J., "The VLSI Design Automation Assistant: A Knowledge-Base Expert System," Ph.D. dissertation, Carnegie-Mellon University, Pittsburgh, PA, April 1984.

[12] McDermott, J., "R1: A Rule-Based Configurer of Computer Systems," *Artificial Intelligence*, vol. 19, pp. 39-88, Sept. 1982.

[13] *Motorola Microprocessors Data Manual*, Motorola Inc., Austin, Texas, 1985.

[14] Norton, S.W. and K.M. Kelly, "Learning Preference Rules for a VLSI Design Problem-Solver," *Proc. of the 4th Conf. on Artificial Intelligence Applications*, pp. 152-158 Mar. 1988.

[15] Setliff, D.E. and R.A. Rutenbar, "Knowledge-Based Synthesis of Custom VLSI Physical Design Tools: First Steps," *Proc. of the 4th Conf. on Artificial Intelligence Applications*, pp. 102-118 Mar. 1988.

[16] Shortliffe, E.H., *Computer-Based Medical Consultation: MYCIN*, Elsevier, New York, 1976.

[17] Tanimoto, S., *The Elements of Artificial Intelligence An Introduction Using LISP*, Computer Science Press (1987).

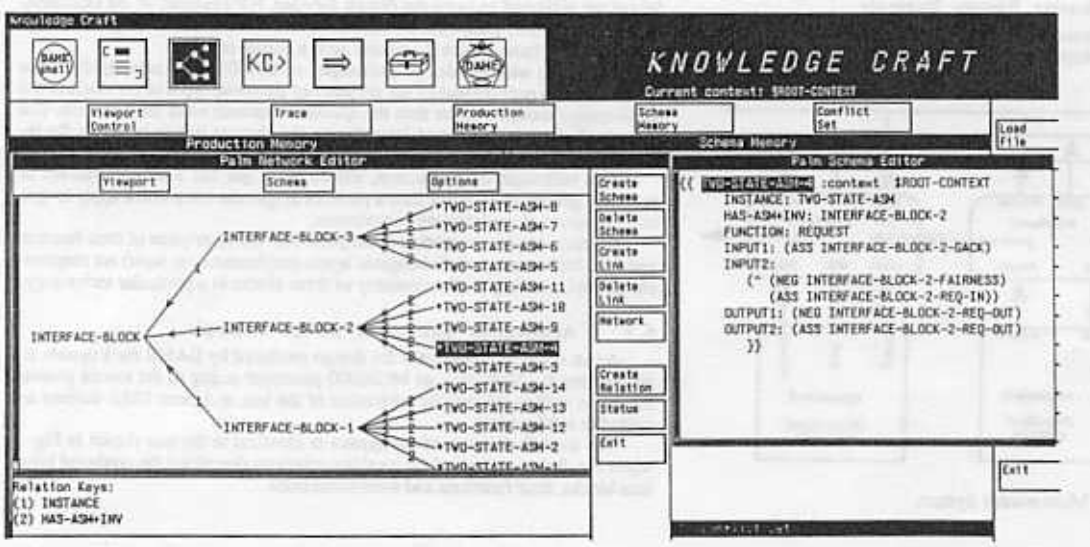


Figure 9. Schemata describing the resulting interface blocks.