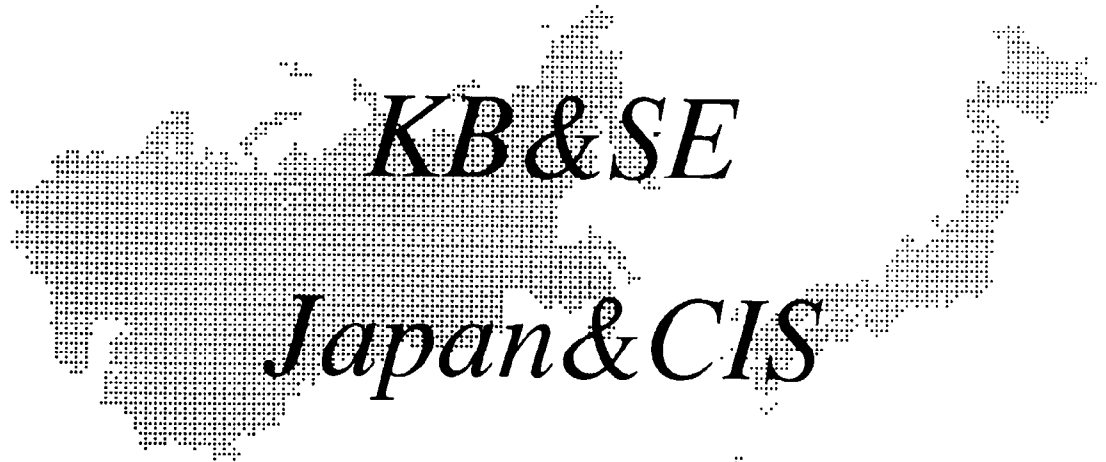


Proceedings of
JCKBSE'94

Japan-CIS Symposium on Knowledge Based Software Engineering' 94



Site

Pereslavl-Zalesskii, Russia

Date

May 10-13, 1994

Sponsored by

SIG on Knowledge Based Software Engineering,
Institute of Electronics, Information and Communication Engineers(EIC), Japan
Association for Artificial Intelligence of Russia

In cooperation with

SIG on Knowledge-Based Systems, Japanese Society of Artificial Intelligence
Research Center for Artificial Intelligence of Program Systems,
Institute of Russian Academy of Science
SIG on Software Engineering, Information Processing Society of Japan
CAI Society of Japan
IEEE Tokyo Section Computer Chapter
Research Center for the Problems of Quality of Education, Russia

Supported by

International Communications Foundation(ICF), Japan

A SYSTEMATIC APPROACH FOR IMPLEMENTATION OF HUMAN DESIGN KNOWLEDGE IN AUTOMATIC SOFTWARE DESIGN

Behrouz H. Far, Toshio Tanaka, Takeshi Sugimoto and Zenya Koono

Department of Information and Computer Sciences
Faculty of Engineering, Saitama University
255 Shimo-okubo, Urawa 338, Saitama, Japan
far@cit.ics.saitama-u.ac.jp

Keywords: AUTOMATIC SOFTWARE DESIGN, KNOWLEDGE ACQUISITION, SOFTWARE ENGINEERING, EXPERT SYSTEM

Abstract. This paper introduces a systematic approach towards implementation of software design knowledge of a team of human design experts, using cooperative expert system technology. The focus is on reuse of the design knowledge that a single expert possesses when contributing to a design shared among a number of experts. We propose a structure for expert systems that combines *function detailing* and *data detailing* for a team of human experts through interactions and negotiations among units.

1. Introduction

This paper introduces a systematic approach towards implementation of software design knowledge acquired by a systematic way described in the companion paper [7]. It uses cooperative expert system technology. The focus is on reuse of the design knowledge that human experts possess.

In *Software Creation* project, automatic software design by following design steps of human designers has been studied and a family of expert systems are developed [4-6]. We found that the Specification and Description Language (SDL) [8] is a useful tools for recording design steps in sequential software. We have been using this to derive *design*

rules and implementing them in the *Software Creator* expert systems [2, 3].

At present, a family of CREATOR expert systems have been implemented. The system is composed of an SDL CASE tool and an expert system shell, ES/KERNEL2 [1]. The latter is used for knowledge editing, browsing and interface building.

In CREATOR1 the idea of acquisition and application of design rules is elaborated. CREATOR2 has the following features: representing software design knowledge using frame technology; and integrating knowledge-based reasoning techniques with a SDL CASE tool, such as SDT [9]. CREATOR3 is an extension of the CREATOR2 that enables one with additional design schemas for splitting a process, adding events, etc., and additional representation power, such as using pictorial elements and designers' comments in the frame representation of the SDL symbols. This leads to having a uniform modeling and reasoning environment for software design that serves as a platform for studying human design.

Based on experiments on implementation of design rules in Software Creation project, we have understood that design rules are mainly detailing rules that transform a function to a group of more detailed ones. We call them rules for *function detailing*. These rules are used for replacing given symbols –widely used in GUI of commercial CASE tools –

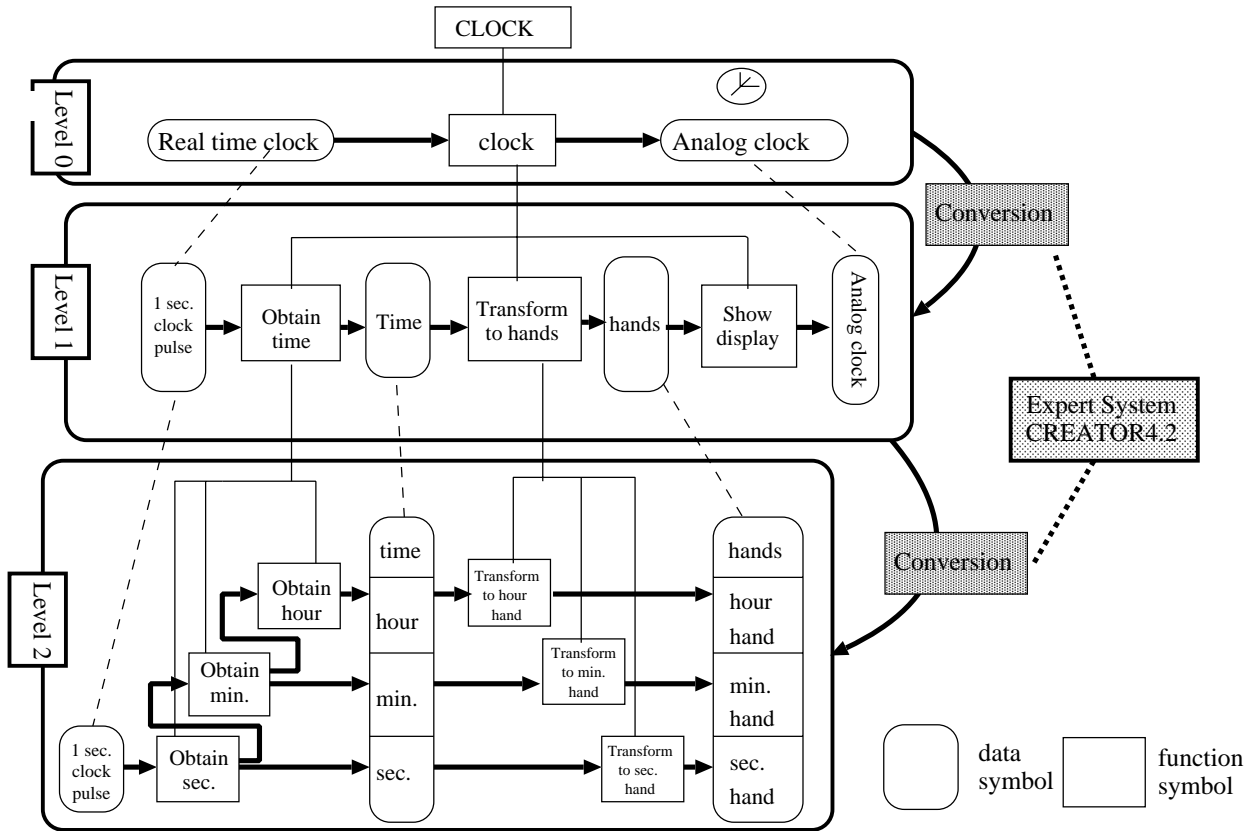


Figure 1: Design steps of a simple analog clock.

with a number of other symbols. A method for deriving design rules has already been introduced [5]. The main idea is to follow design steps of human designers. The design rules are extracted by comparing the design documents in successive design steps. These rules are implemented using frame technology. Each design rule is composed of a parent frame for the pre-transformation symbol and a number of child frames for post-transformation ones. We have also found that there are some other design rules that code a certain series of design steps. We encode such rules in a series of frames by writing methods for implementing those steps.

At the present phase of this project we are considering extending the functionality of the software CREATOR system by:

- Supporting upper design levels, such as data flow diagrams, HIPO charts and data definition tables, that are mainly different from what is conventionally considered as the domain that SDL can support.

- Supporting simulating cooperation among a team of designers in a team work design.

This paper gives various suggestions for removing the above mentioned problems. It introduces a systematic approach for implementing software design knowledge of a team of human experts, using cooperative expert system technology, with the focus on reuse of the design knowledge that a single expert possesses when contributing to a design shared among a number of experts.

2. Implementation Perspective – An Example

When moving from the flowchart level design to the data flow diagram design, we observed that the basic ontological primitives of this level are *function* and *data*. Opposite to the flowchart level design, here *functional detailing* cannot be performed solely without interactions with *data detailing*. This is the main point that our project deviates from other design techniques. Figure 1 shows data flow

diagram design of a simple analog clock system. As it is visible, the data and function detailing appear together and as shown in Figure 2 a structure for cooperation between these two is found necessary.

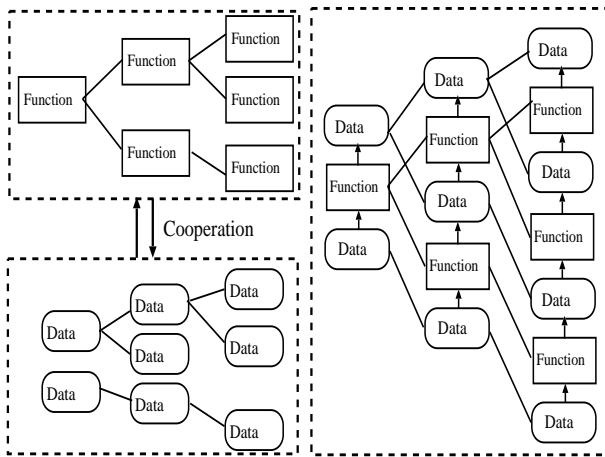


Figure 2: Cooperation between data and function detailing.

Furthermore, the requirement is usually detailed when the design progresses. This is also done by using function and data detailing. This is always done implicitly in conventional design documentation. In software reuse, recording such detailing steps and maintaining them is a crucial task. We have recorded detailing steps in the form of production rules (if ... then) and *schema* in our system. Schema is a predefined pattern that can be generated by adding situation-specific data into it. Each schema addresses a number of production rules that in turn select proper data and function objects from a candidate list. Further detailing will be performed on those selected objects.

Finally, in order to keep track of design steps, *design control* knowledge becomes necessary. The companion paper elaborates on these various knowledge categories [7].

3. Expert System Perspective

Overview of an expert system for data flow diagram design is shown in Figure 3. This system converts a given initial design sketch, in the form of a data flow diagram, to a more detailed one that can be in turn transformed to flowchart. This latter can be processed

by previously mentioned CREATOR2/3 systems.

The *conversion* function is composed of *splitting*, *matching*, *detailed function*, *detailed data*, and *merging* units, as shown in the lower part of Figure 3. Design starts with an abstract data flow diagram (see Figure 1). The design rules acquired from actual design are used for detailing. The intermediate design data, also in the form of data flow diagram is saved. This is useful for design revision and back-tracking. These units are implemented in CONVERT, SPLIT, MATCH, FUNC_DETAIL, DATA_DETAIL, and MERGE objects, respectively.

3.1. Input frame structure

The design input in the form of a data flow diagram is fed to the CREATOR4.2 system. It is converted to a frame structure, suitable for reasoning by the expert system. Each data or function symbol in the input file is represented by a frame in the CREATOR4.2 system. The relation between related data or function frames in successive design levels is encoded in those frames.

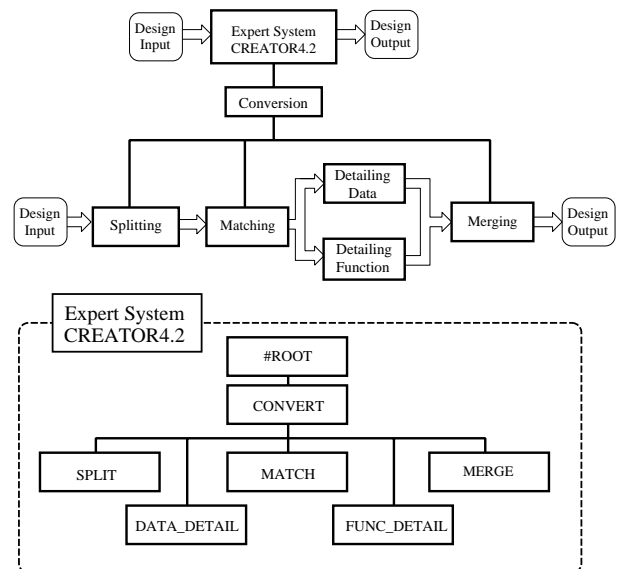


Figure 3: Overview of the CREATOR4.2 system

3.2. Design rules

Design rules are used for replacing a given symbol with a number of other symbols in detailing function or data, generating a

task from successive states, etc. In CREATOR4.2, the detailing design rules are *data-rules* and *function-rules*. Each design rule is composed of a parent frame for pre-transformation symbol and a number of child frames for post-transformation ones. Figure 4 shows an example of a design rule for detailing function. In this figure a function object from an upper level data flow diagram is converted to three subfunctions in the successive level. All the data related to each symbol and the necessary connections to other objects are encoded in the frames.

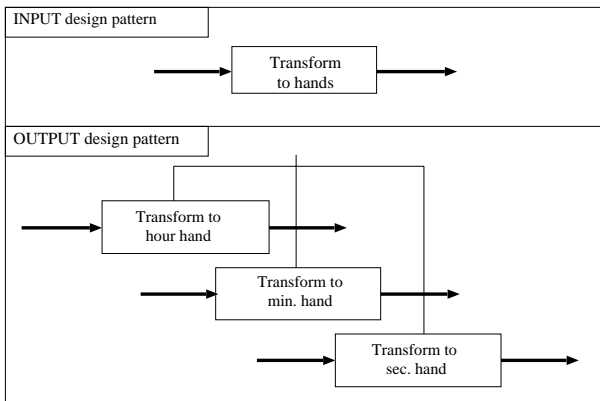


Figure 4: Frame representation of design rules

3.3. Reasoning example

Figure 5 depicts a reasoning example. The #ROOT unit has the top most authority in the system. It controls all the design steps for correctness and soundness. It sends message to the methods of various detailing and conversion units to do a particular job. In this example first the method for splitting the input, residing in the SPLIT object, is executed and then the method for combining the results in the MATCH unit is summoned. The two methods may communicate with each other to share data recorded in the slots of the objects they belong to.

4. A Systematic Structure for Cooperative Expert System

The unit expert system introduced above is a member of a group of expert systems. The companion paper elaborates on the systematic structure [7]. Figure 6 shows such struc-

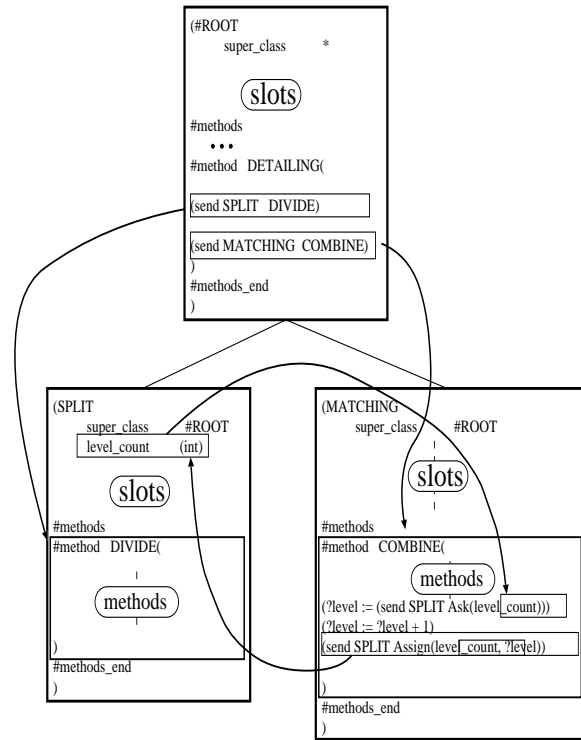


Figure 5: A simple reasoning example

ture, by taking examples from human society. As it is visible, the job assigned to each work process (left of Figure 6) can be produced by using an expert system for that particular job (right of Figure 6).

Basically, each expert unit has the domain knowledge (in this case, data flow knowledge) and control knowledge (in this case, control flow knowledge). At each level, domain knowledge is composed of design rules, mentioned earlier. Control knowledge is the knowledge for applying design rules, such as those encoded in the SPLIT and MERGE of each expert unit. It also includes knowledge of negotiation for members of the same level, as well as guiding units lower in the hierarchy, and reporting to the units upper in the hierarchy. In summary, control flow is actually a program that manages messages and executable tasks and maintains queues of commands.

The basic job of unit members of the same level is described in detail herewith. Figure 7 shows a schematic view of cooperation. In this figure, each expert system unit stands for a human designer and the whole system represents a team of designers.

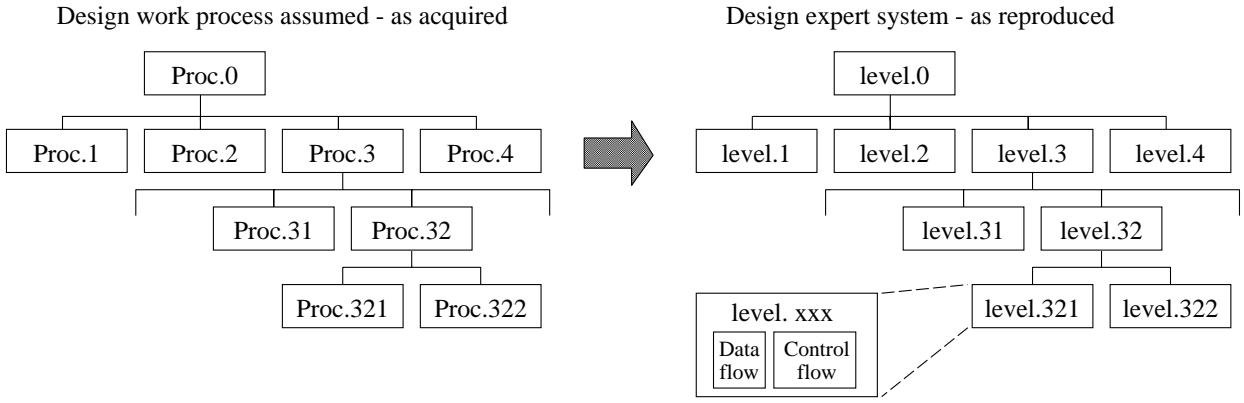


Figure 6: Hierarchical structure of cooperative expert systems

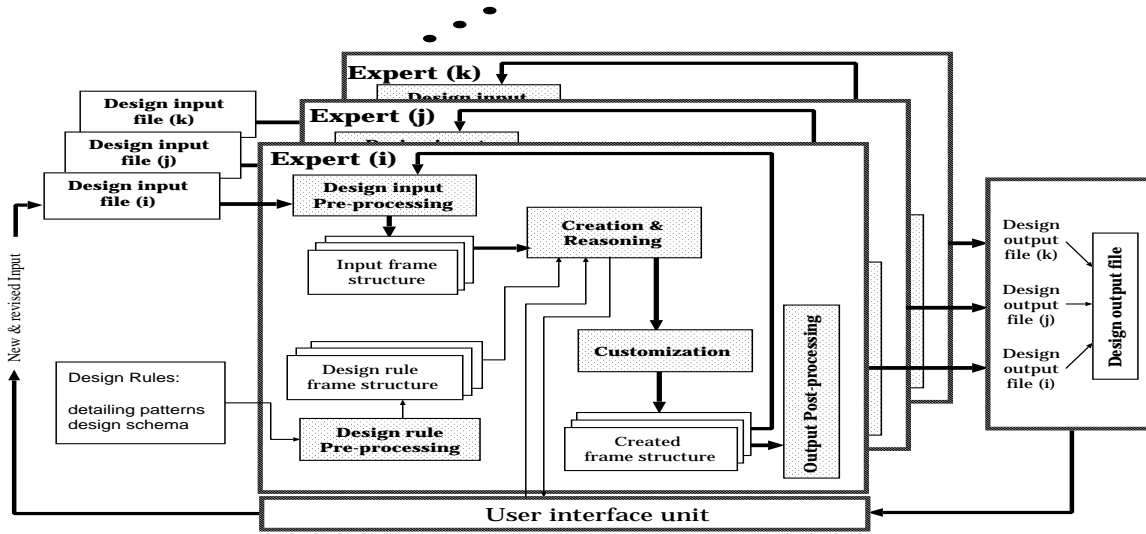


Figure 7: Overview of the system.

Each human designer, D_i , interacts with an expert system, E_i , that implements D_i 's design objectives. Expert systems E_i , E_j , etc., have identical structure with different knowledge bases, each expert in a certain area. The design sketch and commands are automatically converted to frame structure, suitable for processing by the expert system. The expert system E_i checks if this frame structure can be detailed by either a design schema or an already recorded design rule that can be customized to exhibit the required function. It fetches design rules, inserts child frames of the matched rules in the input frame structure, and copies the results in the output frame structure. It reports the results to the other E_j s. The control knowledge of this unit governs this task. Various steps of fetching and applying design rules are coded in frames

and methods of each E_i .

The results of detailing and customization are recorded in the created frame structure. The designer can check and modify the results, if it is necessary.

The design environment and commands affect the design activity. Some design steps involve little interaction between designers, such as coding, however, there are other steps emphasizing negotiation. Our system can support both. In a cooperative design environment a human designer gives some commands that are interpreted by the system. Such commands may include asking for a piece of data or a simple program code that implements the function. These commands are interpreted by the control flow of upper level expert units, that have authority over their subordinate units using their

upper level design knowledge. The upper position in the hierarchy brings the more complexity in the control flow that regulates tasks of the subordinate units as well as detecting end of a level and proceed to a next design phase. Each control function is simple and clear but they together constitute a complex behavior.

This system is useful in the sense that limited capacity of the short term memory of human designers is enhanced by system's stack, queue and array that have a larger capacity, in principle. In representing the domain concepts and organization of design input/output, frame representation is found useful. Each graphic symbol, commonly used in CASE tools, is represented by a frame and design refinement is nothing but a proper selection of such frames or their child instances and inserting message paths among them. Yet other frames are used to control data access and selection, and the whole structure can cope with the event-driven nature of the design process, ensuring high flexibility of the design and maintaining its rationale.

5. Conclusion

In this paper a cooperative expert system with uniform hierarchical structure for automatic software design is presented and basic implementation guidelines are given. It is based on the idea that design work of a team of human designers can be represented by hierarchically organized cooperative expert systems, resembling a society of human designers. The proposed system can put together separate knowledge resources, and simulates a team of human designers. The system can design simple programs in case of altered specification.

Acknowledgements

Authors thank the Information Systems Division, Hitachi, Ltd. for the support of this work.

Many of the students of the CIT Lab., Department of Information and Computer Sciences, Saitama University, have contributed to this project. Here we wish to thank them collectively.

REFERENCES

1. ES/KERNEL/2W-BS Reference Manual, Hitachi, 1991.
2. B.H. Far, T. Takizawa and Z. Koono, "Software Creation: An SDL-Based Expert System for Automatic Software Design," *SDL '93: Using Objects*, O. Færgemand and A. Sarma, eds., pp. 399-410, Elsevier Publishing Co., North-Holland, 1993.
3. B.H. Far, T. Takizawa and Z. Koono, "Software Creation: An Expert System for Reproducing Human Cognitive Processes in Automatic Software Design," in *Proc. World Congress on Expert Systems' 94*, Estoril, Lisbon, Portugal, January 1994.
4. B.H. Far, T. Takizawa, and Z. Koono, "Software Creation: An Expert System for Applying Design Process Knowledge in Software Design," in *Proc. 1993年度人工知能学会(第7回)全国大会*, Tokyo, Japan, 1993, pp. 653-656.
5. Z. Koono, B.H. Far, T. Baba, Y. Yamasaki, M. Ohmori, and K. Hatae, "Software Creation: Towards Automatic Software Design by Simulating Human Designers," in *Proc. 5th Int. Conf. on Software Eng. and Knowledge Eng., SEKE' 93*, CA, USA, June 1993, pp. 327-331.
6. Z. Koono, B.H. Far, T. Takizawa, M. Ohmori, K. Hatae, and T. Baba, "Software Creation: Implementation and Application of Design Process Knowledge in Automatic Software Design," in *Proc. 5th Int. Conf. on Software Eng. and Knowledge Eng., SEKE' 93*, CA, USA, June 1993, pp. 332-336.
7. Z. Koono, B.H. Far, T. Sugimoto, and T. Tanaka, "A Systematic Approach for Acquisition of Human Design Knowledge," this conference.
8. *CCITT Recommendation Z.100, Specification and Description Language (SDL)*, ITU, Geneva, 1992.
9. SDT CASE Tool ver. 2.2 Reference Manual, Telelogic, Sweden, 1992.