



**The 5th International
Conference on
Software Engineering
and
Knowledge Engineering**

Co-Sponsored by

Knowledge Systems Institute
The Johns Hopkins University Applied Physics laboratory

In Cooperation with

IEEE Computer Society and ACM SIFSOFT

Tutorials, June 14-15, 1993

Technical Program, June 16-18, 1993

Hotel Sofitel, San Francisco Bay, USA

Software Creation: Implementation and Application of Design Process Knowledge in Automatic Software Design

Zenya KOONO Behrouz H. FAR Takeshi TAKIZAWA
Mari OHMORI Ken-ichiroh HATAE Takeshi BABA
Department of Information and Computer Sciences
Saitama University
255 Shimo-okubo, Urawa 338, Saitama, Japan

ABSTRACT

The goal of this project is to automate software design by accumulating knowledge and experience of human designers. This paper presents the CREATOR2, an experimental expert system for automatic software design, that simulates the behavior of human expert designers by applying the design process knowledge. In CREATOR2 the focus is on reproducing human cognitive processes and automating software design by shifting the modifications to the higher design levels rather than the source code. Some novel points are (1) implementing the design process knowledge, including transformation patterns for detailing and control knowledge, along with the design product knowledge together in an expert system named CREATOR2, and (2) using multiple strategy in applying the design process knowledge. This leads to having a uniform modeling and advanced reasoning environment for software design.

1 Introduction

In this paper we report on an expert system for supporting knowledge intensive tasks of software design. The focus is on implementing a system that can support the *design process knowledge*, in order to capture and reason with the sort of knowledge that human experts use in design.

(1) Conventional CASE tools and automatic programming techniques facilitate the requirement analysis (i.e. structured analysis tools), detailed design (i.e. structured design tools) and code generation (code generating tools) [17]. Their ultimate goal is to improve the quality and productivity of the design by assisting the designer throughout the different levels of the development process. Conventional CASE tools cannot support the higher level *knowledge-intensive* activities of design due to the lack of deep representations and sophisticated reasoning methods [14]. Presently, higher level design activities, such as those related to selecting proper pieces of knowledge, decision making and

evaluation, are to be handled by the designer. The knowledge of the software development process can be captured and partially implemented in the CASE tools using Artificial Intelligence (AI) techniques. This paper presents the CREATOR2, an experimental expert system for automatic software design, focusing on reproducing human cognitive processes, and automating the software design by shifting the modifications to the higher design levels rather than the source code.

(2) There are two inter-related knowledge categories involved in design, addressing either the *design process* or the *design product*. During the intermediate design steps, human experts rely on their design process knowledge that is accumulated over years of developing similar software products and is useful for decision making and design evaluation. The design process knowledge includes certain transformation patterns for detailing and control knowledge for applying them. The latter is hardly documented, maintained and applied to novel cases. The design product knowledge, on the other hand, consists of the domain-specific concepts and constraints of the task.

In some empirical studies of software design it is observed that both expert and novice designers decompose their design in a top-down manner at different levels of detail until reaching a point that a portion of the decomposed structure can be converted to the code [8]. Opposite to novice designers, experts can manage to develop many sub-goals at the same level of abstraction before further proceeding to the next level [4]. However, in designing complex systems, as the number of sub-goals increases, it becomes increasingly difficult to keep track of them. A main problem is that even expert designers show rather poor performance in keeping track of all intermediate pieces of information and subgoals simultaneously. This is called *tunneling* effect of the *cognitive overload* [6]. Efficient encoding the design process knowledge, i.e. keeping record of decision steps and applied transformation patterns, design evaluations and modifications, customization, etc., and applying such knowledge in novel cases can remove the tunneling effect and help reduce software development costs.

(3) The idea of automating software design is the main theme of the Software Creation Project and some preliminary implementations have already been reported [1, 10, 11, 12]. The companion paper describes the project [13] and this paper introduces a viewpoints on capturing and applying the design process knowledge, using multiple strategy in applying control knowledge and an O-O implementation of it in the CREATOR2 system.

2 Software design knowledge

Here we concentrate on representation and support of software design knowledge, i.e. *design product knowledge* and the *design process knowledge*, in reusable form, particularly by introducing an structure for integrating these two knowledge categories.

2.1 Design product knowledge

The design product knowledge relies on the perspective that the design system is viewed. It includes domain-specific concepts, constraints and qualitative or quantitative models of the system. We have used the Specification and Description Language (CCITT-SDL) [3] for representing the design product knowledge. CCITT-SDL has both graphic (SDL-GR) and text-based (SDL-PR) versions. Fig. 1 shows an example of SDL-GR and SDL-PR.

In CCITT-SDL the system is viewed as a collection of ‘blocks’ embodying concurrent ‘processes’. Processes are represented by an *Extended Finite State Machines* (EFSM) that communicate with discrete signals [3]. An advantage of using CCITT-SDL is its ability to concentrate on the ‘process’ as the basic module of design and exchange messages between processes that corresponds with our O-O view of the system.

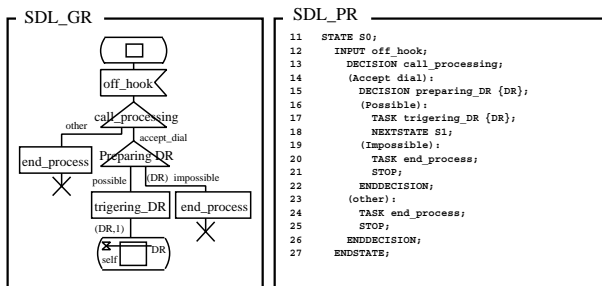


Figure 1: Example of SDL-GR and SDL-PR.

2.2 Design process knowledge

The design process is viewed as a progression towards a certain goal (objective) by employing predefined

detailing patterns. The design process knowledge involves (a) ‘transformation patterns’ acquired from human design, and (b) ‘control knowledge’ to make such patterns operational. Transformation patterns are declarative in nature and are used to transform a given pattern to a more detailed one. Such patterns are called ‘*design rules*’ and an approach for deriving them has already been introduced [1, 10, 11, 12].

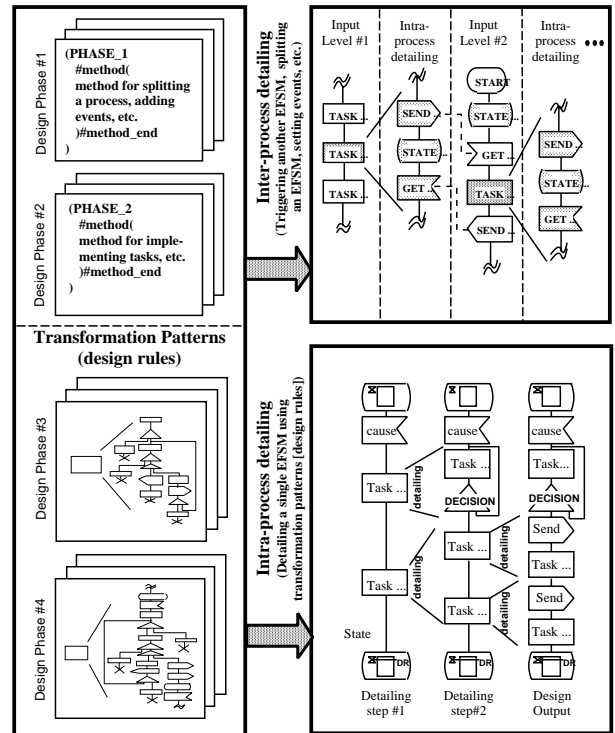


Figure 2: Intra- and inter-process detailing.

A main activity in human design involves selection and application of such patterns using the control knowledge. Control knowledge is neither declarative as in the transformation patterns, nor sequential as in the design product knowledge. Here each step may be triggered based on a certain symptom. Some recent empirical studies show that this knowledge may be applied in an opportunistic way that deviates from the top-down stepwise refinement of design [23]. The need for a scheme that can integrate the pure top-down approach with the data driven strategy has already been mentioned [7, 4].

In the control knowledge, we have distinguished between two strategies for applying transformation patterns, i.e. *intra-process detailing* and *inter-process detailing* (see Fig. 2). In the intra-process detailing, the control knowledge is used for applying transformation patterns within a given sequential machine, that is a SDL process. This is shown in the lower portion of Fig. 2. For instance, in this case a single graphical SDL-GR symbol is replaced by a collection of other symbols that

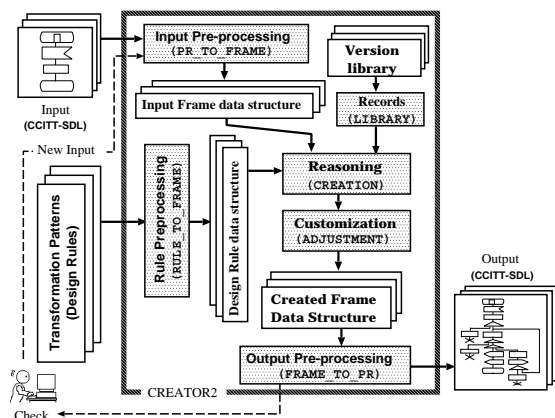


Figure 3: Prototype CREATOR2 system.

exhibit the same function in more detail within a single sequential machine. This type of transformation is quite common in lower level design in which a major process is already selected to realize a goal, but some elaboration or fine tuning is required to fit it into the specific situation. In SDL terms, this is synonym to replace a DECISION or a TASK symbol with a collection of other symbols that embody the detailed activities of the DECISION or TASK. On the other hand, in the inter-process detailing, control knowledge is used for applying patterns that trigger other sequential machines, i.e. CCITT-SDL processes. For instance, an instance of a new process is created. This is shown in the upper portion of Fig. 2.

The whole detailing procedure consists of 4 phases:

- Phase #1: Splitting the processes;
- Phase #2: Implementing tasks;
- Phase #3: Detailing functions (upper level);
- Phase #4: Detailing functions (lower level).

In intra-process strategy, a local and limited search is sufficient to find the proper pattern. However, in the inter-process strategy a look-ahead (e.g. finding the succeeding state and the events between them) or look-back (e.g. finding the preceding state and the events leading to the present state) search strategy are applied.

3 The CREATOR2 system

We have developed the CREATOR2 system based on the idea of integrating different experts to achieve a common goal. The system is composed of 6 experts that are specialized in a certain area. Each expert is composed of a number of objects and each object can store, process and create information as well as communicate with the other objects (see Fig. 3).

(1) The human designer may decompose the initial goal to a number of subgoals until getting to the

level that the subgoal can be realized by an elementary combination of simple graphic symbols used in CASE tools (in this project the CCITT-SDL symbols), such as TASK (calculation, etc.), DECISION (branching, loop, case, etc.) or OUTPUT (send message, data-base access, etc.). The system checks if this can be detailed by an already recorded transformation pattern (design rule) that can be customized to exhibit the required function. If a proper pattern exists, it can be involved in the design. Customization is necessary to adjust all the interfaces to the other program units. Already used versions are recorded and applied in similar situations. On each step, the given information, such as next link, next symbol or next slot value selects the next design step. This ensures high flexibility of the design while maintaining its rationale.

(2) The CREATOR2 system is implemented on Hitachi 3050 Workstation using ES/KERNEL2 expert system shell [5] that works together with the SDT CASE Tool [20]. SDT is used for graphical editing, translation between text based SDL-PR and graph based SDL-GR, and simulation of the results. All the other design tasks, conversion, knowledge based reasoning and detailing are performed by the CREATOR2 system.

(3) The CREATOR2 system is composed of 6 experts (plus a dedicated #ROOT program) each performing a limited but complex and specialized task (see Table 1). The PR_TO_FRAME expert is used for preprocessing and converting the text based SDL-PR to the frame structure suitable for processing by the CREATOR2 system. The *class objects* in CREATOR2 are defined as the counterparts of the elementary CCITT-SDL symbols. Every graphical symbol in CCITT-SDL code is associated with a frame in the CREATOR2 that includes all the data related to that symbol, including its function and connections to other symbols. Therefore each design input file has a unique frame structure counterpart in the CREATOR2 system. The lower part of Fig. 4 shows the frame structure for an input file.

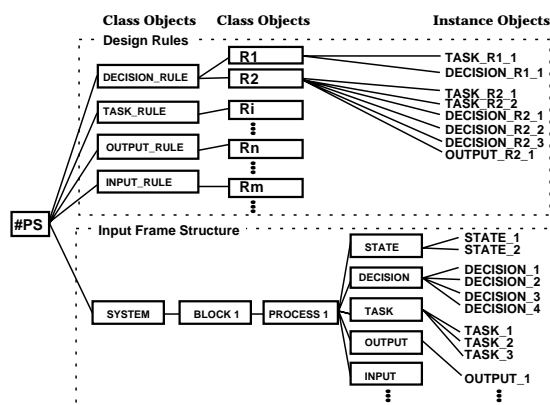


Figure 4: Example of frame structure for an input file and transformation patterns.

The extracted transformation patterns (design rules)

are also converted to the frame structure using the `RULE_TO_FRAME` expert. Again class objects are defined to distinguish between the transformation patterns. At present, there are 4 classes of transformation patterns called `DECISION-RULE`, `TASK-RULE`, `OUTPUT-RULE` and `INPUT-RULE`. These are shown in the upper part of Fig. 4. Every pattern is associated with a parent frame representing the pre-transformation symbol and a number of child frames for post-transformation ones. Each transformation pattern also has a unique frame structure counterpart in the `CREATOR2` system. Fig. 5 shows a typical transformation pattern.

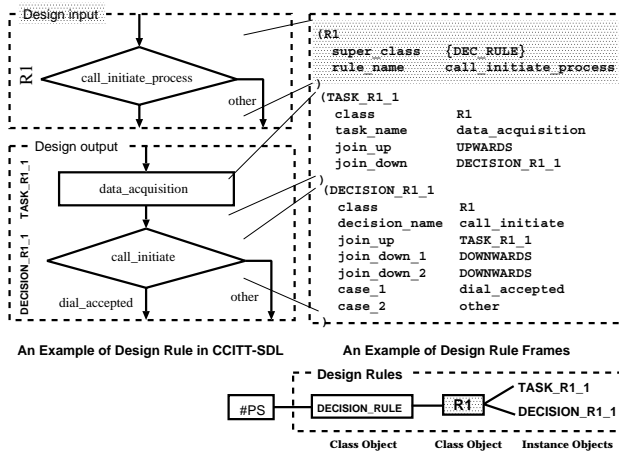


Figure 5: An example of frame structure for transformation patterns.

The `CREATION` expert stays as the core of the `CREATOR2` system. In the knowledge base, there are already two group of frames for input file and transformation patterns (see Fig. 3). The `CREATION` expert is responsible for checking the input frame structure, fetching transformation patterns and checking the present active and pending goals of the designer. Design proceeds by applying transformation patterns at various levels during inter- and intra-process detailing. All design steps are recorded according to their order of appearance and the system can explain each step if asked to. The results of creation are delivered to the `ADJUSTMENT` expert which is responsible for customizing the candidate frames and adjusting the links. This is the hardest task of automatic design because every single slot of a candidate frame must be checked and all newly created frames should be accounted for. The `LIBRARY` expert keeps record of the customized patterns and the pending ones. This is necessary for saving time in similar design cases and when a pattern is applied repeatedly. The `ADJUSTMENT` and `LIBRARY` experts together realize the learning function of the `CREATOR2` system. Finally, the `FRAME_TO_PR` expert converts the final frame structure to text based `SDL-PR` that can be used by the `SDT CASE` tool. The number of objects and control program size for each expert is given in Table 1.

(4) Presently, the `CREATOR2` system is tested for creating software for switching services at the lower levels of design. Some experiments are reported in the companion paper [13]. It takes about 12 minutes to design a typical switching control program (such as `POTS`) whose input file is composed of about 50 `CCITT-SDL` symbols.

(5) In `CREATOR2` a set of frequently used `CCITT-SDL` symbols are accounted for. This is sufficient for most of the design cases. However, the `PR_TO_FRAME`, `RULE_TO_FRAME` and `FRAME_TO_PR` experts should be extended to cover other `CCITT-SDL` symbols, too.

The most important limitation is that as the number of features increase the control program of the `ADJUSTMENT` expert becomes increasingly complicated. This also dominates the total run time of the system. We are presently investigating the guided search method and the `LIBRARY` expert to select the best option and minimize the run time.

4 Related works

Research in automatic software design is inspired by top-down approach of the generic task-oriented methodologies, such as [18, 15, 19, 9], in which software design is viewed as incremental editing and refinement of a text-based generic object. In many systems the design can only proceed if the user is familiar with the given generic object and the available library [16]. Also some knowledge-based design systems employ heuristic and structural programming rules [22]. Our approach is different in the sense that we have applied graphical symbol-based detailing rather than the text-based one. This allows integration of the presently available `CASE` tools with the knowledge based reasoning techniques.

In some works the need for distinguishing between the design product knowledge and design process knowledge is mentioned [2, 21]. We have integrated and implemented the design process knowledge and design product knowledge together. Furthermore, we have distinguished between the transformation pat-

Table 1: Program size and no. of objects of the `CREATOR2` experts.

Expert	Function	program size	no. of objects
<code>ROOT</code>	Controlling and coordination of the other experts	(45 lines)	1
<code>PR_TO_FRAME</code>	Converting <code>SDL-PR</code> to frame structure	(770 lines)	7
<code>RULE_TO_FRAME</code>	Converting transformation patterns [design rules] to frame structure	(400 lines)	5
<code>CREATION</code>	Selecting and applying design rules	(2090 lines)	12
<code>ADJUSTMENT</code>	Adjusting value of the slots of the frames	(3100 lines)	12
<code>LIBRARY</code>	Recording newly generated patterns	(1200 lines)	5
<code>FRAME_TO_PR</code>	Converting frame structure to <code>SDL-PR</code>	(840 lines)	7

terns and the control knowledge of the design process. Many systems have considered a single strategy in design [19, 9]. The CREATOR2 system is unique in the sense that it adopts various strategies at different design phases. The top-down decomposition is used for proceeding towards a goal and the data-driven strategy is used within a given design phase to select proper detailing pattern.

5 Conclusion

In this paper a framework for integrating representational models of software design with the design problem solving knowledge of human experts was introduced and an experimental CREATOR2 system for automatic design of switching software was reported. The key feature of our approach is using a unified representation scheme for modeling the design process, decision schemas, transformation patterns and the design product and also using multiple strategy in applying the knowledge. In long term, the CREATOR2 system is supposed to be integrated with the on going project Software Creation System, aiming at meeting the basic requirements of automatic software design, speeding up the software development, and leading to reduction of development costs that is currently a critical factor in software design.

ACKNOWLEDGEMENT

The authors thank The Telecommunications Advancement Foundation for supporting the switching control area, Information and Telecommunications Division, Hitachi, Ltd. for supporting the switching administration area, and Information Systems Division, Hitachi, Ltd. for supporting the expert system area. We also wish to express our gratitude to students of the Department of Information and Computer Sciences, Saitama University who contributed to this project by implementing various parts of the system.

References

- [1] T. Baba, K. Miya, T. Yabuuchi, Y. Shigemori, Y. Naito and Z. Koono, "Software Creation: The First Results," in *Proc. IEICE Fall Conf.*, Tokyo, pp. 6.420-421, 1992.
- [2] S. Bhansali and H.P. Nii, "KASE: An Integrated Environment for Software Design," in *Artificial Intelligence in Design' 92*, J.S. Gero, ed., Kluwer Academic Publishers, 1992, pp. 371-389.
- [3] *CCITT Recommendation Z.100, Specification and Description Language (CCITT-SDL)*, ITU, Geneva, 1992.
- [4] S.P. Davies and F. Simplicio-Filho, "Opportunistic and Goal Oriented Behavior in Software Design," in *Artificial Intelligence in Design' 92*, J.S. Gero, ed., Kluwer Academic Publishers, 1992, pp. 839-860.
- [5] ES/KERNEL/2-W Reference Manual, Hitachi Ltd., 1991.
- [6] B.H. Far, M. Nakamichi and M. Sampei, "Qualitative Supervisory Control Featuring Cognitive Capabilities of The Domain Practitioners," in *Proc. 2nd Makuhari Int. Conf. on High Technology (MIGHT' 91)*, Chiba, Japan, Feb. 1991, pp. 135-138.
- [7] B. Hayes-Roth and F. Hayes-Roth, "A Cognitive Model of Planning," *Cognitive Science*, vol. 3, pp. 275-310, 1979.
- [8] R. Jeffries, A. A. Turner, P.G. Polson and M.E. Atwood, "The Processes Involved in Designing Software," in *Cognitive Skills and Their acquisition*, J.R. Anderson, ed., Erlbaum, NJ, pp. 255-283.
- [9] M. Klein, "DRCS: An Integrated System for Capture of Designs and Their Rationale," in *Artificial Intelligence in Design' 92*, J.S. Gero, ed., Kluwer Academic Publishers, 1992, pp. 393-412.
- [10] Z. Koono, T. Baba, K. Miya, T. Yabuuchi, Y. Shigemori and Y. Naito, "Software Creation: A Systematic Approach," in *Proc. IEICE Fall Conf.*, Tokyo, pp. 6.420-421, 1992.
- [11] Z. Koono, T. Baba, T. Yabuuchi, Y. Naito, Y. Shigemori and K. Miya, "Software Creation: Systematic Approach," *Technical Report of IEICE*, KBSE92-28, pp. 33-40, 1992.
- [12] Z. Koono, T. Baba and T. Yabuuchi, "Software Creation: A Trial for Switching software," in *Proceedings 5th JC-CNSS, 1992 Joint Conf. on Communications, Networks, Switching Systems and Satellite Communications*, Kyungju, Korea, 1992, pp. 261-265.
- [13] 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 *Proceedings 5th Int. Conf. on Software Engineering and Knowledge Engineering, SEKE' 93*, this conference.
- [14] M.R. Lowry, "Software Engineering in the Twenty-First Century," in *Automating Software Design*, M.R. Lowry and R. D. McCartney, eds., AAAI Press, 1991, pp. 627-654.
- [15] M.D. Lubars and M.T. Harandi, "Intelligent Support for Software Specification and Design," *IEEE Expert*, vol. 1, no. 4, pp. 33-41, Winter 1986.
- [16] Luqi, "Knowledge-Based Support for Rapid Software Prototyping," *IEEE Expert*, pp. 9-18, Winter 1988.
- [17] R. McCartney, "Knowledge-Based Software Engineering: Where We Are and Where We Are Going," in *Automating Software Design*, M.R. Lowry and R. D. McCartney, eds., AAAI Press, 1991, pp. 17-31.
- [18] T.M. Mitchell, L.I. Steinberg and J.S. Shulman "A Knowledge-Based Approach to Design," *IEEE Trans. Pattern Anal. and Machine Intell.*, vol. PAMI-7, no. 5, pp. 502-510, September 1985.
- [19] C. Rich and R.C. Waters, "A Research Overview: The Programmer's Apprentice," *IEEE Computer*, pp. 11-25, November 1988.
- [20] SDT CASE Tool ver. 2.2 Reference Manual, Telelogic, Sweden, 1992.
- [21] J. Treur and P.J. Veerkamp, "Explicit Representation of Design Process Knowledge," in *Artificial Intelligence in Design' 92*, J.S. Gero, ed., Kluwer Academic Publishers, 1992, pp. 677-696.
- [22] J.B. Tsai and J.C. Ridge, "Intelligent Support for Specification Transformation," *IEEE Software*, pp. 28-35, November 1988.
- [23] W. Visser, "More or Less Following A Plan During Design: Opportunistic Deviations in Specification," *Int. J. of Man-Machine studies*, vol. 33, no. 3, pp. 247-278, 1990.