数理解析研究所講究録
第618巻 1987年 76-96

REUSABLE DESIGN METHODOLOGY FOR SWITCHING SOFTWARE

Yoshinori HORI　　Shigekatsu KIMURA　　Hiroyuki MATSUURA
堀　好徳　　木村　重勝　　松浦　洋征

NTT Electrical Communications Laboratories

Reusability offers tremendous potential for significantly improving switching software productivity. Of the two roughly divisible reusability technology approaches, source program code and design information, this paper focuses on the latter. Specifically, it describes a reusable design methodology and a design support system which facilitate the reuse of design information being used during the design process. The paper also presents a design example produced through this reusable design methodology.

1. Introduction

In a number of countries where telecommunications technologies are particularly advanced, integrated services digital networks (ISDNs) of one form or another are being constructed to usher in the information-intensive society of the future.[1] ISDN implementation requires increasingly capable and expansive switching software, which, in turn, necessitates that switching software development productivity receive considerably greater

attention.

It is clear that reusability is a central key to the significant improvement of switching software development productivity. The presently used methods as well as those being researched for facilitating software reuse can be roughly divided into two approaches depending upon the nature of the objects being reused: the source program code, which represents the final output of the design process, and the design information used during the design process. The reusable design methodology presented here aims at facilitating a broader reuse of all types of design information.

This reusable design methodology is based on three concepts. First is the division of design information necessary for designing switching software into four knowledge specification types: service, signaling system, hardware structure, and hardware detail specifications. The former two constitute switching system architecture-independent knowledge, while the latter two involve switching system architecture-dependent knowledge.

Second is the performance of program design by successively applying these four types of knowledge specifications to the design process, whereby four types of programs are derived. Of these, an abstract program and two types of intermediate programs are the intermediate phase outputs of the design process, while a concrete program for a target switching system is the final output.

Third is the derivation of reusable components, which include

various kinds of design information, from the four program types. For example, reusable components derived from the abstract program include only the design information of service specifications. This is because the abstract program is derived using only service specifications. On the other hand, reusable components derived from the concrete program include all types of design information.

Additionally, the classification of the design knowledge in conjunction with the stepwise application of this knowledge to the design process enable the natural introduction of knowledge-based processing technologies into this reusable design methodology. This should, in turn, facilitate computer-aided design.

This paper first outlines the trends in software reusability technologies. Next, it describes a reusable design methodology and a design support system which facilitate the reuse of design information. Finally, it presents a specific example describing how programs are derived using this design methodology.

In this paper, we broadly define "design" to also include the requirements definition as well as the manufacturing in the conventional software life cycle. We also use information and knowledge interchangeably.

2. Trends in software reusability technologies

(1) Outline

Considerable attention has recently been paid to reusability as a principal method for improving software development productivity.[2] Along these lines, various approaches have been

studied and attempted. These have included one for providing reusable source program code components, and one which enables programs to be derived semi-automatically or automatically from specifications.[3],[4]

Up to the present, however, no single agreed-upon approach to reusability exists. The existing approaches, however, are classified into the reuse of the source program code, which represents the final output of the design process, and the reuse of the design information used during the design process itself. It should be remembered that program code reuse is subdivided into the reuse of program code modules and that of similar program systems in the same application area. In line with this, Table 1 shows the principles of reuse, the technological levels, and typical methods applicable according to this classification.

(2) Switching Software

Since switching software depends directly on the target switching system hardware architecture, many data structures and procedures are resultantly dependent on the actual hardware structure and details. Thus , the chance is relatively small that a program code module in one type of switching software will coincide with another program code module in a different type of switching software. Program modularization as well as standardization of both switching system architecture and switching software architecture is essential for facilitating the reuse of program code modules. On the other hand, new program design methodologies need to be established to facilitate the reuse of design information. In general, switching architecture is determined with a view toward upcoming technological advances.

Facilitating design information reuse is considered to be the most promising approach because it is uninfluenced by switching system architecture.

3. Reusable design methodology through program transformation

To ensure the complete design information reuse facilitation necessary for designing switching software, a design methodology placing greater emphasis on reusability must initially be established. Following this , three additional processes are important. First is the design of switching software using this design methodology. Second is deriving reusable components from the design information being used during this design process, and from the programs output. Third is the design of other switching software types using these reusable components.

3.1 Model of switching software design

The model underlying our reusable design methodology is based on five specific principles.

(1) Switching knowledge necessary for designing switching software can be divided into switching system architecture-independent knowledge and -dependent knowledge. The former can be subdivided into service specifications and signaling system specifications. The latter can be subdivided into hardware structure specifications and hardware detail specifications. Here, hardware structure specifications are concerned both with the functions of such hardware components as the trunk and switching network, of which a switching system is composed, and with the physical connective relationship between them. On the

other hand, hardware detail specifications involve the control of each hardware component toward the realization of a certain function.

(2) Switching software can be considered to be the design result in which the two knowledge types are reflected.

(3) The design of switching software can be performed through stepwise program transformations by successive applications of these knowledge types.

(4) The result of program transformations in each step can be considered to be the program derived by applying the knowledge necessary in this step to the result of program transformations in the immediately preceding step.

(5) Reusable components can be derived from programs derived in each step.

3.2 Outline of reusable design methodology

The reusable design methodology based on the above model can be outlined based on five key points as indicated in Fig. 1 and Table 2. Essentially, then:

(1) An abstract program (level 1), which is independent of switching system architecture, is designed using only the design knowledge derived from service specifications.

(2) An intermediate program (level 2), which is also independent of switching system architecture, is derived from the abstract program of level 1 through program transformations in which the design knowledge derived from signaling system specifications is applied to the abstract program of level 1.

(3) An intermediate program (level 3) for a target switching

system is obtained from the intermediate program of level 2 through program transformations in which the design knowledge of its hardware structure specifications is applied to the intermediate program of level 2.

(4) A concrete program (level 4) for the target switching system is derived from the intermediate program of level 3 through program transformations in which the design knowledge of its hardware detail specifications is applied to the intermediate program of level 3.

(5) Reusable components, which are independent of switching system architecture, can be gotten from the abstract program of level 1 and the intermediate program of level 2 , while reusable components which are dependent on a target switching system can be obtained from the intermediate program of level 3 and the concrete program of level 4.

This reusable design methodology can also be considered to be a kind of operational approach rather than to be a conventional software life cycle approach to software development.[5] A comparison is made in Table 3 and Fig. 2 between this reusable design methodology and the conventional design methodology.

3.3 Program reusability of each level

The reuse of design information is essentially equivalent to the reuse of the programs derived. That is , an abstract program of level 1 is one which reflects only service specifications. An intermediate program of level 2 is one which reflects signaling system specifications in addition to service specifications. An intermediate program of level 3 is one which reflects hardware

structure specifications in addition to service and signaling system specifications. Finally, a concrete program of level 4 is a final design result which reflects all the design information including hardware detail specifications.

Essentially, the lower the level number of the programs is, the lower is its dependence on switching system architecture. Accordingly, the reuse of levels 1 and 2 programs is generally possible between different switching system architectures. Such is the case, for example, in the design of switching software, which satisfies the same service and signaling system specifications, for a different switching system architecture. On the other hand, the reuse of levels 3 and 4 programs is generally possible within the same switching software, particularly in the case of functional additions. The reuse of the level 4 program is no other than that of the source program code.

3.4 Reusable design support system

Figure 3 outlines a reusable design support system example, which aims to facilitate computer-aided design by introducing knowledge-based processing technologies, based on the proposed reusable design methodology. The design results database in the system accommodates such design process outputs as the programs of each level as well as design history information. The components database is an assembly of reusable components derived from the programs of each level. The design knowledge database contains the switching knowledge mentioned previously as well as the programming knowledge used for designing the programs of

each level.

3.5 Effect of reusable design methodology

Four important effects will result from the application of this reusable design methodology.

(1) Enhancing the reusability of all types of design information is possible, especially that of design information independent of switching system architecture.

(2) Facilitating computer-aided design is possible through the positive utilization of knowledge-based processing technologies, which, in turn, improves design efficiency and quality.

(3) Design history information such as design knowledge applied during the design process can be documented, which serves to enhance program understandability.

(4) Maintenance is possible on such programs as the abstract program in which only high level specifications are reflected, rather than on the concrete program, by automating the program transformation process in the future.

4. Program design example

This section concentrates on the design of state transition diagrams and the generation of tasks from the diagram necessary to perform state transitions through the proposed reusable design methodology . It also takes a look at intra-office connection as a target service feature, and at a function-distributed digital switching system as a target switching system.

4.1 State transition diagram design and task generation

In state transition diagram design and task generation, a

state transition diagram is first designed which indicates service specifications and which is independent of both the signaling system type and the switching system architecture. The state transition diagram is then refined stepwise incorporating such design knowledge as signaling system and hardware structure specifications. The state transition diagram is finally partitioned using the design knowledge of functional distribution into several state transition diagrams each of which corresponds to each subsystem of the function-distributed digital switching system. From each of these diagrams, tasks can be generated using such design knowledge as programming specifications.

4.2 State transition diagram design

At this point a closer look at each of the above steps beginning with the actual state transition diagram design is important. The call state of a state transition diagram is specified by a set of resources, each of which performs a particular function. These resources are considered to be abstract resources in the state transition diagrams of levels 1 and 2. On the other hand, they consist of such hardware resources as trunk, speech path and tone sender, and such software resources as timing supervision and metering in the state transition diagram of level 3. In a simpler sense, these resources can be considered to be "objects". This allows us to define a call state as a set of these objects along with their own states.

The CCITT Specification and Description Language (SDL/GR) is used to represent the state transition diagram. Also adopted is a

textual expression like PROLOG, which is suitable for state transition diagram refinement and task generation.

In terms of state transition diagram refinement, stepwise refinement is accomplished through the transformation processes, consisting of locating the part to be refined using the design knowledge, and of modifying or replacing it with more detailed information. The knowledge of signaling system specifications is used to refine the state transition diagram of level 1 into that of level 2. An example of the knowledge used in this case is that of the digit receiver (DR) in the digit-receiving state being classified into a PB receiver (PBR) and a DP receiver (DPR). The knowledge of hardware structure specifications is used to refine the state transition diagram of level 2 into that of level 3. Examples of the knowledge used in this case are the switching network consisting of concentration and distribution stages, and a subscriber line connected to the distribution stage via a line concentration trunk (LCT).

With respect to partitioning of the state transition diagram, three subsystems consisting of call control, subscriber line control, and trunk control, cooperatively perform call processing based on their own state transition diagrams in this type of function-distributed digital switching system. Thus, the level 3 state transition diagram must be partitioned into the state transition diagrams of these three subsystems. This is achieved by first clarifying the objects with which each subsystem deals based on the knowledge of function distribution. The state transition diagram of each subsystem is then derived by extracting all objects and their state descriptions associated

with the subsystem.

4.3 Task generation

The tasks for performing state transitions can be generated from the state transition diagram of each subsystem through three main procedures utilizing the knowledge of programming and of hardware specifications particularly related to task generation. In terms of these procedures, it is assumed that a task is generated for a state transition from a certain present state to the next state.

(1) The state change of each object between the present state and the next state is extracted. Such changes include the appearance of a new object, the disappearance of an unused object and so on.

(2) The task macro-instruction which performs the state transition corresponding to the state change of each object is selected from a set of previously prepared task macro-instructions. When a new object appears in the next state, for example, a task macro-instruction is necessary to initiate the search for a new object from a pool of idle objects of the same type.

(3) The correct execution sequence among all selected task macro-instructions is determined.

Figure 4 is a design example of a state transition diagram consisting of both the digit-receiving state and the ringing state of the intra-office connection, and the generation of its level 3 task based on the above procedure.

5. Conclusion

To meet the growing demand for the switching software essential to efficient and effective ISDN implementation, reusability has surfaced as a potential key to improving switching software development productivity. Along these lines, this paper specifically described a reusable design methodology for facilitating the reuse of the design information used during the design process. This is accomplished by dividing the design information into switching system architecture-independent knowledge and -dependent knowledge.

The method was shown to have two central features. First, it permits the derivation of two types of switching system architecture-independent programs, such as an abstract program, and two types of switching system architecture-dependent programs, such as a concrete program. Second, it enables derivation of reusable components, which are independent of switching system architecture, from the former programs, while allowing derivation of reusable components, which are dependent on a target switching system, from the latter programs. The paper also presented an example designed employing the present reusable design methodology.

Experimental results confirmed that the proposed reusable design methodology is effective in facilitating switching software reuse.

To firmly establish this reusable design methodology, three chief problems remain:

(1) Designing an abstract program and developing program transformation methods of refining a program of a certain level

into a program of the next more concrete level.

(2) Expressing design knowledge and organizing a design knowledge database for facilitating the application of knowledge-based processing technologies to program transformations.

(3) Establishing a criterion for dividing a program of each level into reusable components.

Acknowledgment

The authors would like to thank Nobuo Araki and Katsumi Maruyama for their helpful suggestions and discussions.

References

(1) Y.Kitahara, "Telecommunications for the advanced information society - INS (Information Network Systems)", Telecom 83 Forum, Part 1, VI.2, Geneva, 1983.

(2) E.Horwitz and J.B.Munson, "An expansive view of reusable software", IEEE Transactions on Software Engineering, Vol.SE-10, No.5, pp.477-487, 1984.

(3) R.G.Lanergan and C.A.Grasso, "Software engineering with reusable designs and code", IEEE Transactions on Software Engineering, Vol.SE-10, No.5, pp.498-501, 1984.

(4) T.E.Cheatham, "Reusability through program transformations", IEEE Transactions on Software Engineering", Vol.SE-10, No.5, pp.589-594, 1984.

(5) P.Zave, "The operational versus the conventional approach to software development", Communications of the ACM, Vol.27, No.2, pp.104-118, 1984.

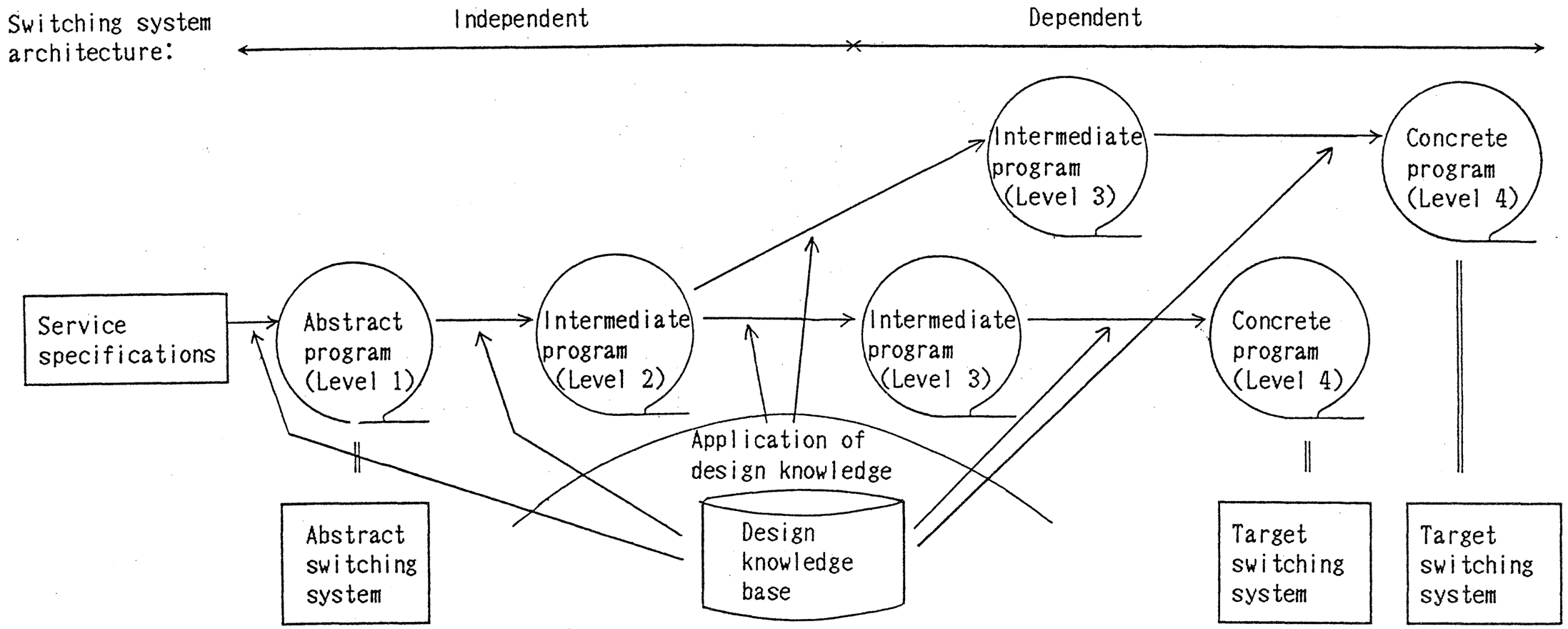
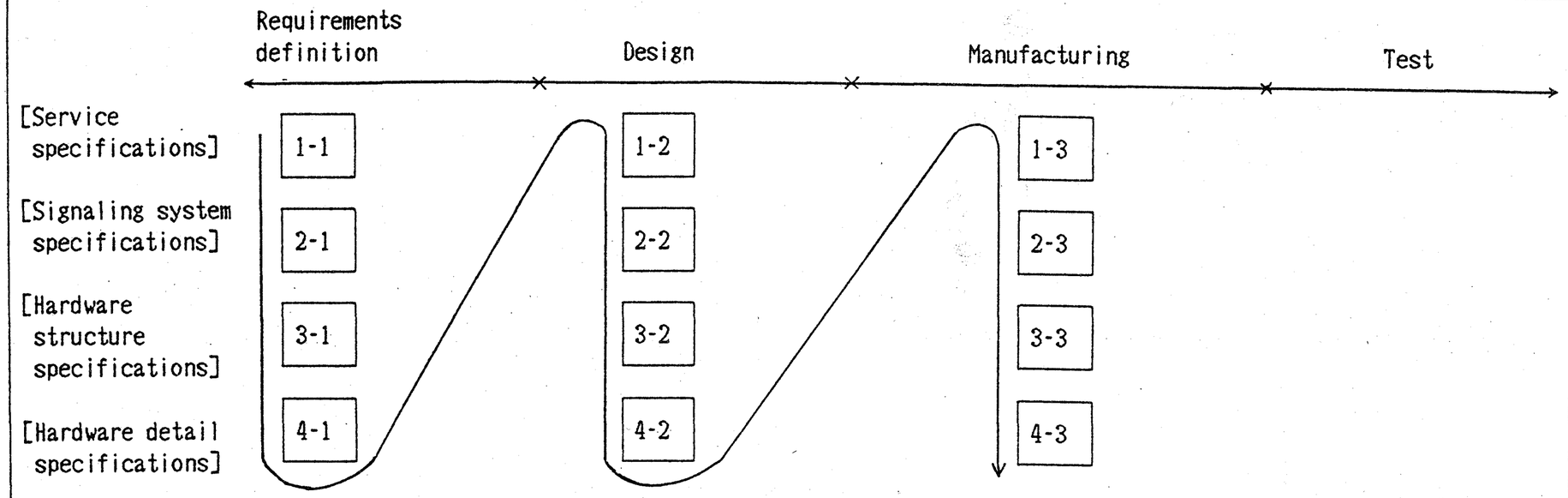


Fig.1 Outline of reusable design methodology through program transformation

Conventional design methodology



Reusable design methodology

Conventional requirements definition, design and manufacturing are distributed for the respective design of each level program, and are repeated during the design process of each level program.

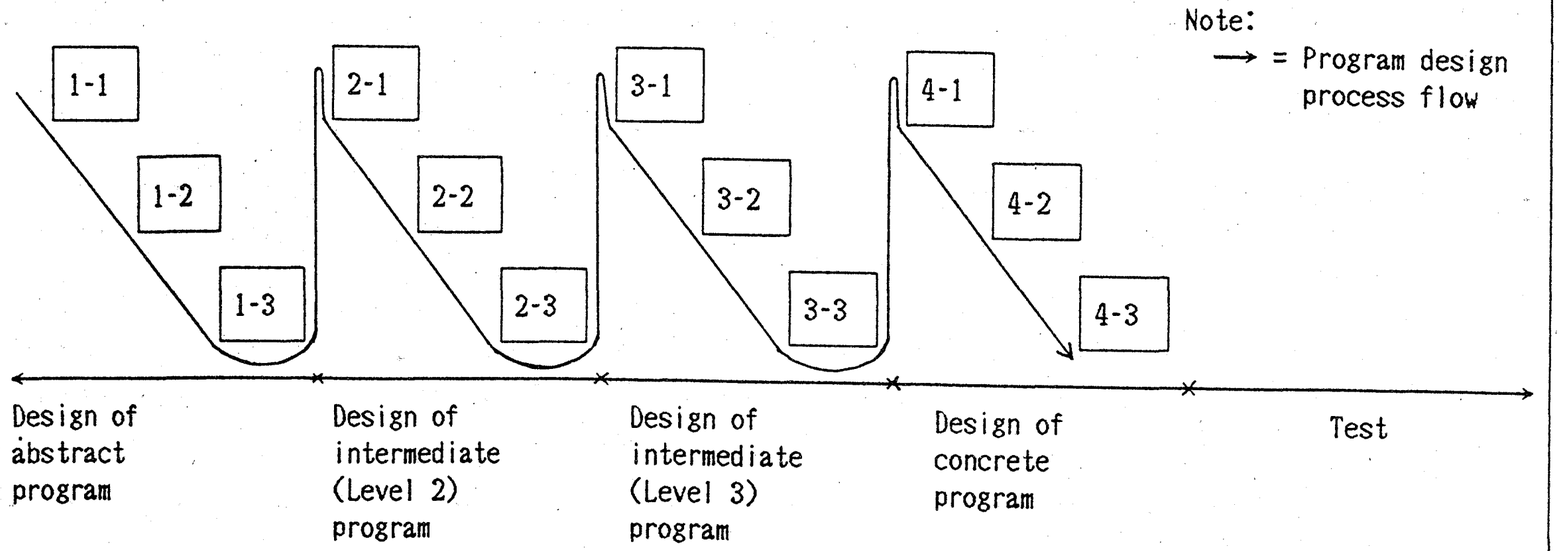


Fig. 2 Software life cycle based on reusable design methodology

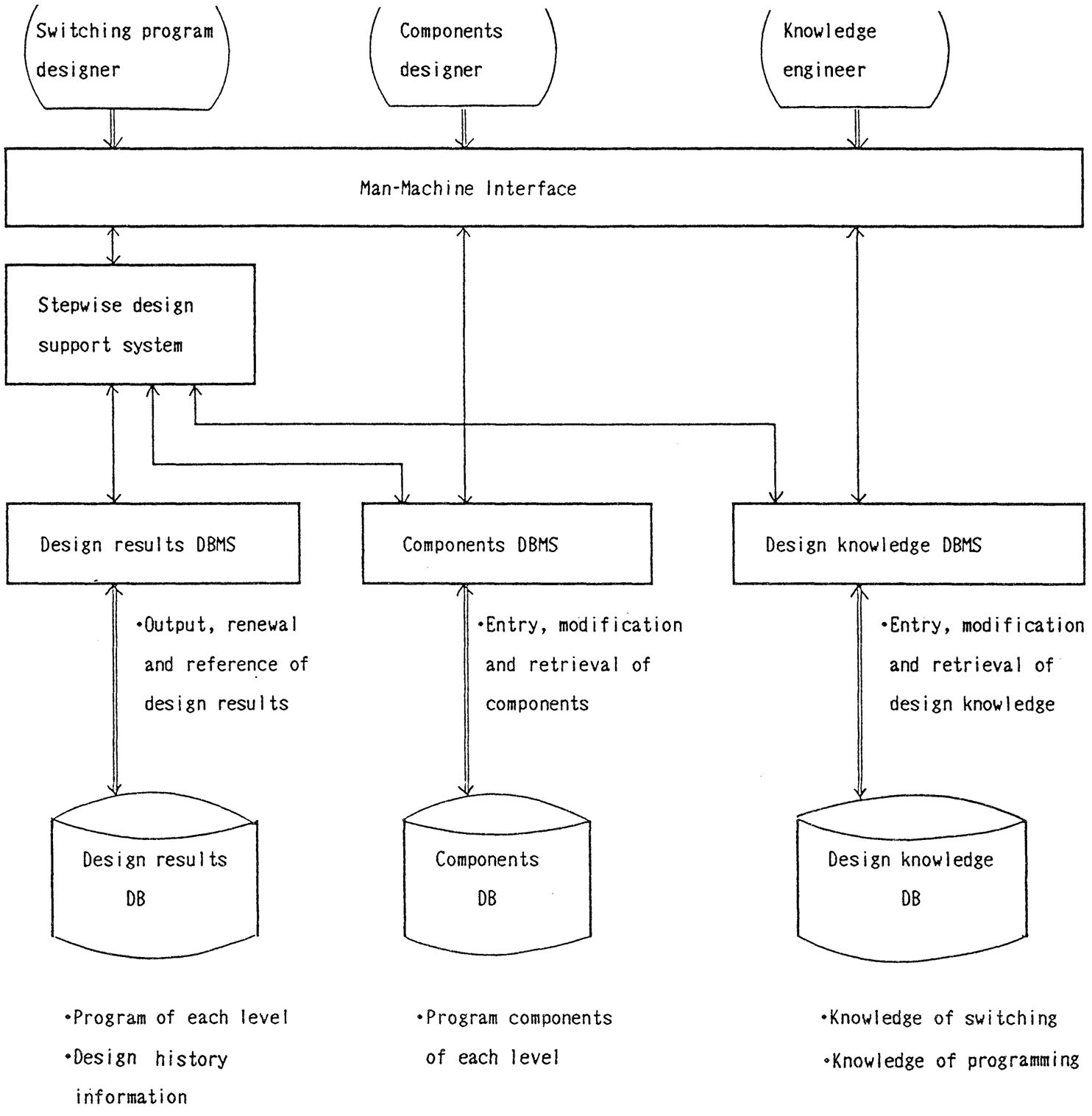
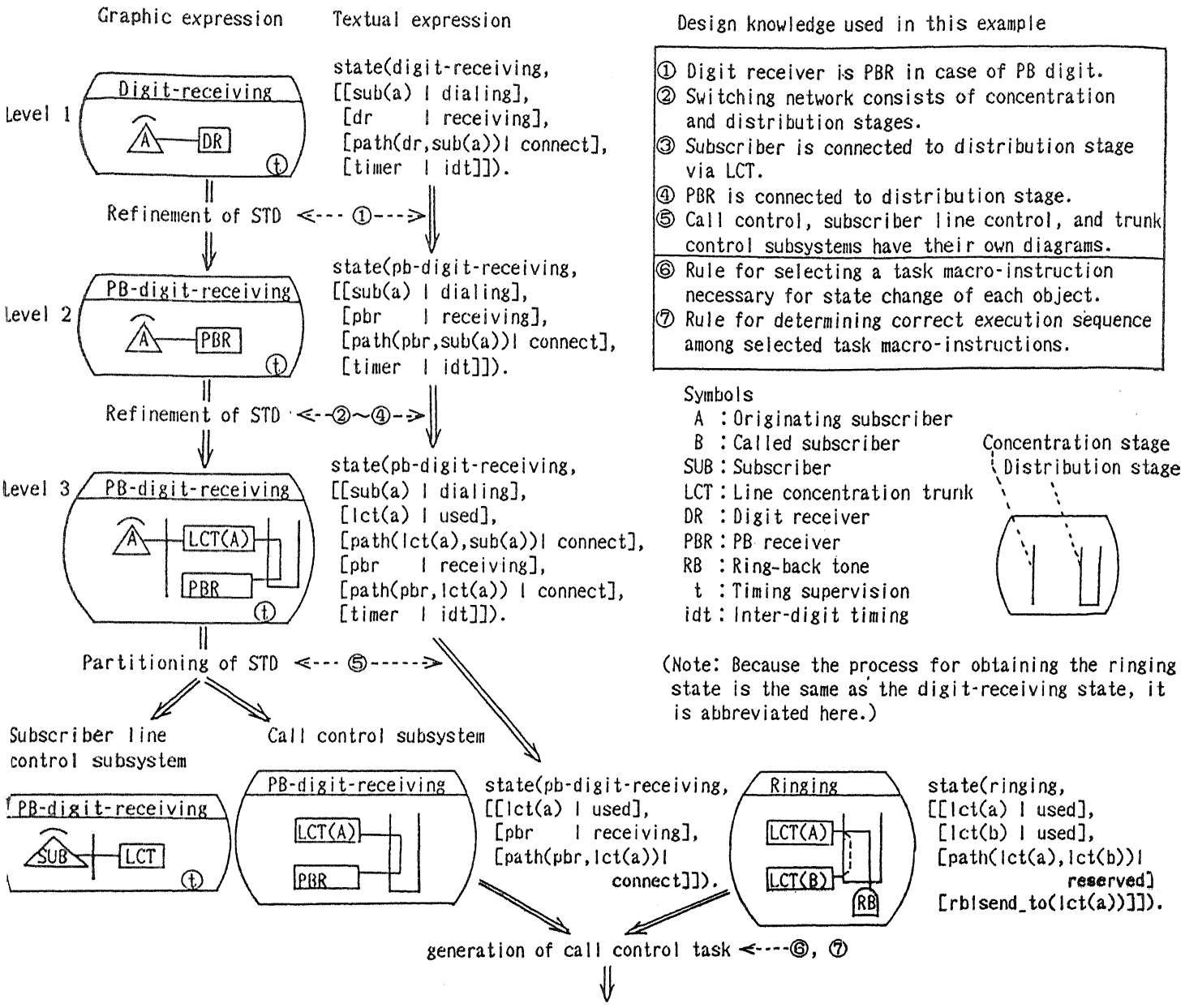


Fig. 3 Configuration of reusable design support system example



Level 3

Task	Function of task macro instruction
hunt(lct(b))	hunt LCT trunk resource
hunt(path(lct(a),lct(b)))	hunt path resource
disconn(path(pbr,lct(a)))	disconnect path
tone_send(rb,lct(a))	send ring-back tone
free(path(pbr,lct(a)))	release path resource
free(pbr)	release PBR resource

Fig.4 Example of state transition diagram(STD) design and task generation

Table 1 Trends in reusability technologies

Reusable object	Source program code		Design information
	Similar program systems	Source program code modules	
Principle of reuse	Development of a concrete program by modification of a similar program	Composition of a concrete program from reusable components	Generation of a concrete program from reusable components
Technological level	Level of practical use	Level of partiall practical use	In general, level of research
Typical methods	Modifications	•Application component libraries •Organization and composition principles	•Very high-level languages •Application generators •Program transformations

Table 2 Program level model

Program level				Program model
Dependency	Program	Level	Specifications	
Switching system architecture-independent	Abstract program	Level 1	Service specifications	Description of behavior of switching system which realizes service specifications in the most abstract level independent of switching system architecture and signaling system type.
	Intermediate program	Level 2	Signaling system specifications	Description of above behavior in the level dependent on signaling system type and independent of switching system architecture.
Switching system architecture-dependent		Level 3	Hardware structure specifications	Description of above behavior in the level dependent on the target switching system architecture which considers hardware structure specifications.
	Concrete program	Level 4	Hardware detail specifications	Description of above behavior in the most concrete level fully dependent on the target switching system architecture which considers hardware detail specifications.

Table 3 Comparison of reusable design and conventional design methodologies

	Conventional design methodology	Reusable design methodology	Aims of reusable design methodology
1.Reusable object	Source code	Design information	•Improvement of reusability
2.Development method	Based on conventional life cycle model	Based on operational approach	•Confirmation of user requirements at an early period •Automation of program transformation from specifications
3.Management of design knowledge	Individual management by each designer	Common management using knowledge base	•Joint ownership of design knowledge
4.Application of design knowledge to design process	No standard	Stepwise application of design knowledge from service specifications to hardware detail specifications	•Improvement of reusability •Standardization of design
5. Design efforts	Person-based design	Computer-aided design	•Improvement in design work efficiency •Improvement in design quality
6.Documentation of design process	Insufficient	Documentation of design knowledge used in design processes	•Improvement in program understandability