Translation from Transactions of IPSJ

The Software Development Environment: dmCASE

SAEKO MATSUURA* and MASAHARU OHBAYASHI*

In this paper, we propose a new software development environment: dmCASE on a new paradigm in the 1990's which Dr. Balzer and others proposed.

This environment is composed of three major concepts. First thing is a design method: DMC (Design Method based on Concepts) which is a kind of object oriented programing methodology. Secondly, we use the standard ML which is a functional language, as an executable formal specification language which can verify formal description. In addition, tools and technologies which support the formal description using DMC are important. A flexible user interface like the process of human thinking plays an important part of our environment.

Our goal is to construct an integrated environment for software development such that we can be concerned in more intelligent work.

1. Introduction

At present, with rapid progress of hardware technology there are many tools which support the programming activities with a flexible user interface. Nevertheless, an essential problem of software development in the 1990's is not yet solved. Several new software technologies such as prototyping, object-oriented programming, formal specification description, AI, automatic programming, etc. [2–4] have emerged. However, these are no established paradigm for the design phase in a software development.

Accordingly it is our most important objective to build an integrated software development environment based on many useful tools and technologies. It also should support a life-cycle model of software development. It is expected that such an environment will improve productivity, portability and reliability. We propose a software development environment named dmCASE which is based on a paradigm proposed by R. Balzer and others. To verify and maintain specifications (but not programs) in an early stage of development, we must write specification using a design methodology and an executable formal specification language. Moreover, a flexible user interface which assists human thinking process will play an important role. Our aim is to construct a new integrated software development environment named dmCASE with a design methodology, a formal specification language and a tool which supports our design process. In addition we propose a total design method with two view points. First is a Space Design to analyze and design a system object with a three dimensional view. Second is a Plane Design that divide this space object into several planes so that we can design them as a two dimensional structure.

In this paper, section 2 discusses our concept on software development environment. In section 3 we introduce our design methodology named "Design Method based on Concepts" and its features. Section 4 is on our formal specification language and section 5 discusses tools which support our design process. In section 6 we explain the organization of dmCASE and its features. Finally section 7 describes a few experiments on specifications and its evaluation.

2. Software Environment

We propose a software development environment which has a paradigm [1] proposed by R. Balzer, et al. To verify and maintain specifications (not programs) in an early stage of software development will make us to engage in more intellectual works. Following four factors are important:

- A methodology which analyzes incomplete requirements and constructs a model of an object from it (identify its hidden or imprecise requirements).
- An executable, formal and verifiable specification language.
- A technology with tools that supports our design method.
- A translation of a specification into an executable program.

Conforming to a design method we specify a requirement formally. Then its formalism enables to verify its static justification and executing the specification enables to verify its justification dynamically. Namely,

This is a translation of the paper that appeared originally in Japanese in Transactions of IPSJ, Vol. 31, No. 7 (1990), pp. 1091–1103.

*Kanri Kogaku Kenkyusyo, Ltd. 1-9-6 Ebisuminami, Shibuya-ku, Tokyo 150, Japan.

to create a clear specification by a strongly formalized language probably improves reliability of software. Moreover, this increases productivity by decreasing a process of back tracking. The difference of this process from programming is that in the former we construct a design structure with specifying objects at more abstract level. In other words, we specify objects without considering an environment of implementation, that is, several features of hardware and a construction of windows, etc. And implementation program will be create to instantiate an abstract specification.

We realize the above four factors to our software environment dmCASE as shown under.

- An object-oriented method named "Design Method based on Concepts" as a design methodology.
- A functional language ML as a formal specification language.

And our basic policy to support above paradigm using tools are following.

- Support a flexible user interface and a design process which assists human thinking process.
- An environment in which we will be engaged in essential works on our design process.
- Constraints which never disturb our thinking.
- Verification and validation

We can design an extensible model which is easy to maintain using the above "Design Method based on Concepts". This point is very important to control collectively a software development process. In above four factors, we have not implemented the last factor.

3. Design Methodology

In this section, we introduce the "Design Method based on Concepts" (abbreviation, DMC) [11] adapted as a modeling method. And we compare it with other several methods and mention its features. In the following, we express our method DMC. And in the following explanations, we write proper words of DMC are marked with [ ].

3.1 Basic Concepts of DMC

DMC is one of the design methods such that we construct a structure of computer system in our way of object-oriented thinking. And it is based on ideas of abstract data types. Bases in our design process are ⟨objects⟩ which represent concrete objects and ⟨concepts⟩ which represent more abstract objects in requirements. We call them [chips]. A requirement is regarded as a set of these [chips]. Then we will create a system model with describing the interaction of [chips] by a two dimensional network diagram. We call this network a [conceptual structure chart] and wired [chips] on this chart [instances]. (see Fig. 1) Each [chip] is a set of several data and processes (=functions) which have a relation to an ⟨object⟩ or a ⟨concept⟩ that appears in requirements. Such words which represents an ⟨object⟩ or a ⟨concept⟩ is also a [chip] in a wide sense. In an early stage of our design process, extracting such words or selecting new words not being in the requirements, we create a set of [chips]. We call this set a [word table]. A [word table] will perform a part to arrange many informations. And we register a lot of design fragments on our design process in it.

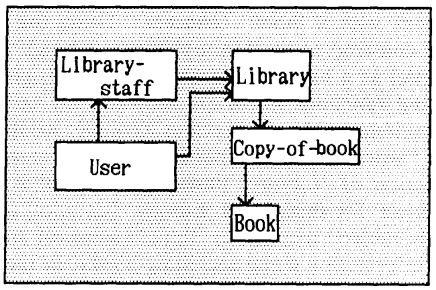

Fig. 1 An example of "The conceptual structure chart".

For example, we suppose there is a requirement to design a library system of "The Library of Nishiazabu". In the words appearing on this requirement, a word "The Library of Nishiazabu" is a concrete ⟨object⟩ and a word 'library' is a ⟨concept⟩ as an abstraction of the previous word. At first step, analyzing the requirement we get a gist of this problem as the following sentence. "In this library, we deal with many and various books and operate a registration and search of books." As a [chip] in a wide sense, we can extract some words such as 'library', 'book', 'register' and 'search', etc. from this sentence. We can recognize that the sentence "In this library, we deal with many various books." means a concept 'library' is explained by the works of a ⟨concept⟩ 'book'. We suppose there is a relationship between a ⟨concept⟩ 'library' and a ⟨concept⟩ 'book'. We describe this relation as an oriented segment on a ⟨conceptual structure chart⟩. Because of this a [chip] 'library' and a [chip] 'book' become concrete [instance] in this "The Library of Nishiazabu" model.

The frame of our designing a structure of [instance] is based on a notion of an abstract data types which is a similar notion of an object in object-oriented programming. This means that we can realize an information hiding. However, we notice that a [instance] can include not only its data and some mechanism to access them, but also some processes which have a connection with a ⟨concept⟩ of [instance]. In this mean, a concept of [instance] is near to the object. Namely an [instance] represents a composite abstract data types which includes not only its own data but also several data in connection with its ⟨concept⟩. In the previous example, the 'Book' [instance] includes 'book' as a data. A data 'book' consists of some data, that is, 'book_name', 'author' and 'subject' which are regarded as attributes of a data 'book'. It also includes some functions to access them. In this way, we will design a 'Book' [instance] as a set which consists of four data and three

functions with a 'book' ⟨concept⟩. Using the above model constructed by 'Library' [instance] and 'Book' [instance] as a basis, in the next step we design some data and functions which are necessary for satisfying the requirement. In the previous example, we thought about "search books in a particular subject area" as a transaction of 'Library' [instance]. We call it 'search_subject' for short. And 'Library' [instance] sends a 'search_subject' message to 'Book' [instance] and gets some information about searching books. In the other word, 'Library' [instance] refers a function 'search_book' which is a transaction of 'Book' [instance]. This enables us to realize a function 'search_subject' which is a transaction of 'Library' [instance]. Each data belonging to an [instance] can refer some data in a lower direction within a [conceptual structure chart]. As shown above, [instances] are [chips] wired by some oriented segments within a [conceptual structure chart]. Moreover, to concrete a [chip] means that we add some signatures of its connected [instance] to a set of signatures belonging to the [chip].

As mentioned above, we construct an initial network model for requirements and refinement this model by arranging some data and processes with their attendance [instance] representing its ⟨concept⟩.

An [instance] specification describes a ⟨concept⟩ of [instance] in formal and it consists of two collections which are signatures and declarations. Signatures are constructed by several type signatures belonging to an [instance]. In other words, they are a set of words which represent data and processes and their types. On the other side, declarations are a module and implementations of signatures. So signatures are the only interface of an [instance] to which the other [instance] can refer. A module hides own information from other modules. There are two ways to instantiate [chips]. One is a 'reference' to the other informations and the other is an 'inheritance'. 'Reference' is a relation between two [instances] which they can call another process in own process. 'Inheritance' is a relation which they can inherit another informations and request to call processes as a 'superclass' in the object-oriented programming. We designed a [conceptual structure chart] and some [instances] like this. It follows that we can represent a concept of data transparency and information hiding.

In the above step, while designing a [conceptual structure chart] and [instances] on it, we describe [instances] specification as a formal specification by a formal language. In other word, we express the structure of data and the contents of messages related to the [instance] by representing their concept using a formal description language. One reason of separating signatures from declarations in an [instance] specification is to turn a module which has same signatures into a software part. Since declarations are an implementation of signatures, we can realize another implementations and create another [instances].

We show design steps on DMC and some foundations of decision for each step in the following Table 1. In This explanation we adopt the Library Problem [10] for an example.

In the process to construct a rough model from an incomplete requirement and design each [instance] which is an element constructing the model, it is a basic concept on DMC to understand and arrange the problem as a collection of objects with common concepts. We consider that arranging with some common ⟨concepts⟩ is very useful to improve readability and enables to extend and maintain a specification.

What is the difference between DMC and other methods? We mention it in the following. In the data flow model, we are recommended to take note of data transferred between processes and to construct a system model by them. In the E-R model, we consider 'entity', that is, a concept of existence and its relations. An 'entity' means an existence which has a notion about a creation and an extinguish, but doesn't mean a complex collection of data and processes such as ⟨concepts⟩ on DMC. On DMC, we regard data and processes as one collection and consider its behavior as a whole structure and describe them. On the other hand, above methods propose to analyze a system in a viewpoint of data and from the different standpoint analyze processes. Nevertheless, I wonder whether it is more natural for human thinking that we regard several objects as a ⟨concept⟩ with data and processes as one collection than regarding them as only data or processes. To speak more precisely, a model on DMC may be similar to a mental model which means that we simulate an object in our brain for the sake of understanding it. In the design step, it may be easy to understand but is insufficient to see its whole structure which enables us to analyze a problem on the particular viewpoint. DMC proposes that we construct a natural model similar to human thinking and give a detail design on the previous model as a basic model. Moreover, analyzing on several viewpoints enables us to understand each element and the whole structure well. In the case of DMC, we realize this concept by understanding the whole structure of problem using a [conceptual structure chart] and supporting to analyze them on various viewpoints.

Then, what is the difference between DMC and object-oriented programming (for example, Small talk)? In object-oriented programming there are concepts which represent 'instance', 'class', 'message passing' and 'inheritance'. The last concept represents a relation of two classes which inherit own informations to another. On the other hand, on DMC, there are concepts of [chip] and [instance]. In this design step, only related [instances] on the structure are effective. In object-oriented programming, a class definition is the first decision and its instance definition is the next, but on DMC we design concrete [instances] at first. It may be difficult to consider an abstract concepts with hierarchy on the early stage of design phase and so we should consider concrete objects at first. We will mention [chips] later as

Table 1 Design step on DMC.

Step	Working items	Viewpoints and bases of judgment to design works	Results
(1)	Get a gist of the problem and denote it in some short sentences	You can explain to others what this system wants to do. If this explanation is not sufficient, you should add further explanations of words which appear in the sentence	Several sentences written by natural language
(2)	Extract words which represent [chips] as candidates for [instances] from the above sentence (1). Register these words to a [word table]	According to a basis of judgment like (1), you should consider whether you can construct an explainable structure with these words. Mainly words are selected from the subjects and objects in the sentences	A [word table]
(3)	Extract several [chips] which will perform parts as data and processes from the problem sentences. Register these words to a [word table]	You should select process words by extracting words which appear in the sentence as system functions. And as data, you should select words which are objects of processes. This means that these words are selected from the subjects and objects in the sentences	A [word table]
(4)	Construct a [conceptual structure chart] with these above words (2)	According to a basis of judgment like (1), you should consider whether you can construct an explainable structure. To speak correctly, you should arrange these above words (2) by regarding the relation between the subjects and the predicates in the context of the above sentence (1) as a relation between upper and lower concepts on the [conceptual structure chart]	A [conceptual structure chart] (see Fig. 1)
(5)	Arrange the above (4) words according to their usage	You should arrange the words by deciding their usage, that is, process, data, exception handling	A [word table]
(6)	Assign these above (5) words to each appropriate [instance definition table]	You should assign the process word to its subject [instance]. And the data word should be assigned [instance] with its own name, or, if it is a component of the word which has the same name with an [instance], it will be assigned this [instance]	An [instance definition table]
(7)	Describe the details of data and process definitions using a specification description language	You should describe each process by sharing own work in other processes. And data structure is decided by the context of the problem and the description of processes	The definition of data and processes with a specification description language
(8)	Back Tracking: Register the words turned to clear their need at the phase (7) to the [word table]. And assign them to appropriate [instance], reconstruct the [conceptual structure chart]. And the other work is to repeat above work from (3) to (7)	You should assign registered words to the [instance] or extract new words according to their need. When an [instance] has too many words for one [instance] to posses, you should divide this [instance] concept to several concepts and reconstruct the [conceptual structure chart] by adding new [instances]	A [word table], a [conceptual structure chart], an [instance definition table] and [instance] specifications (see Table 2)

a problem of dividing parts and reusability. Another character of DMC is to describe a transference of messages on a [conceptual structure chart] using a graphic user interface. There are 'refer' and 'inherit' relations in the upper and lower relations between [instances]. A 'refer' relation plays a role to transfer each message. An 'inherit' defines a relation which is, what is called a 'superclass' relation on object-oriented programming. Nevertheless, on DMC, it is a means of understanding [instances] related by 'inherit' as one object and these [instances] exist on the two dimensional structure. We can control 'information hiding' by changing our selection according to the situation.

3.2 Extension DMC

In section 3.1 we have explained basic concepts of DMC. However, it may be impossible to design a large scale system by describing with one [conceptual structure chart]. So to take measures on this matter, we propose to extend the concept of DMC.

We call a basic unit of description on DMC a [board]. More precisely it means a unit of a model represented by a [conceptual structure chart]. We design a [board] as a two dimensional plane. At the previous phase we considerd to design a system in three dimensional space. Our aim is modeling objects existing in a real world as a system model in space. At first we recognize our objects in three dimensional space and next divide this space at different viewpoints and design each plane so that we can understand its structure easily. And we consider how each plane performs in the space and generate a system model from a three dimensional viewpoint. In this way there are two design steps on DMC, that is, (1) space design and (2) plane design which supplement each other part. We will explain them as follows.

(1) Space design

First step is to divide a system object into several subsystems which are independent of each other. More pre-

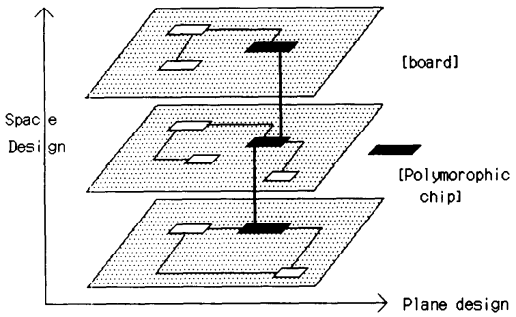

Fig. 2 The multi-layer "board".

cisely, we select some keywords which can construct an object system from given requirements considering their independence and take up them as names of subsystems. This subsystem and a [board] has one-to-one correspondence. And these keywords construct a three dimensional system model. Namely a system model is built by several [board] layers.

In this step, we extract some [chips] which express each subsystem from requirements. These [chips] are respectively instantiated within a [board] and turn into concrete [instances]. If a [chip] is used in some [boards], it will turn into original [instance] in each [board]. This means that we extend a concept of [chip] to [polymorphic chip] which has several sides as a [chip]. An [instance] is defined uniquely in a system model, while [instances] which are turned from [polymorphic chips] and appear on more than one [boards] have a common concept represented by the [polymorphic chip].

In the step of plane design about every subsystem, we design two dimensional structure on a [conceptual structure chart] using these [polymorphic chips]. Some [chips] added during our design step are also regarded as [polymorphic chips] and instantiated in each [board]. And the system possesses these [chips] in common. In this process, several [board] layers which construct a system model are wired vertically along [polymorphic chips]. Then three dimensional system model is formed (see Fig. 2).

(2) Plane design

A design of [board] is equivalent to a plane design. The scope of words used in this step is restricted within a [board] except [polymorphic chips]. So it is not necessary for us to mind the other [boards] design. In a viewpoint of system model, we regard words on a [board] as words with a tag which represents a subsystem's name. This enables us to guarantee the uniqueness of each word on the [board].

One of the famous design methods is a hierarchical system decomposition. In this method we can understand their vertical relations between decomposed modules, but it is hard to find their horizontal relation. Then many similar modules may appear on a whole system structure. On DMC, we consider whole structure as a multi-layers system. This meaning is to decompose a system to layers which are independent [boards] and construct hierarchical structure on each [conceptual structure chart]. We can catch vertical relations between similar modules (=⟨concepts⟩) using [polymorphic chips] as connective concepts. For example Figure 3 shows a system model of dmCASE in which some [polymorphic chips] such as [word table], [conceptual structure chart], etc. appear on it as a common concept.

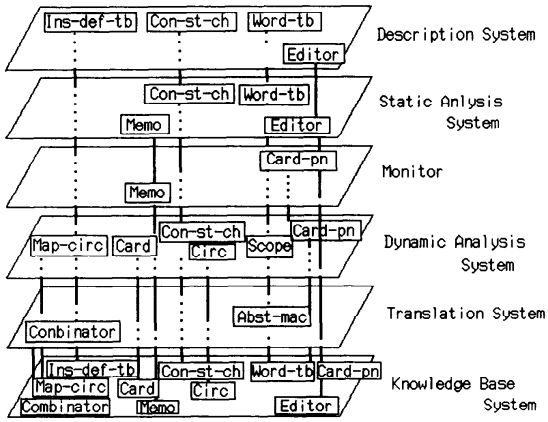

Fig. 3 The system model of dmCASE.

3.3 Software Parts

An [instance] is a specification which expresses a ⟨concept⟩ of original [chip] by deciding relations between the other [instances] on a [conceptual structure chart]. By turning some [instances] connected below into parameters, we can create a [part chip]. In the previous example, we can apply a matter of "In this library, we deal with many various books". to a case of a rental video shop. That is, abstracting the signatures of [instances] to which 'Library' [instance] refers, we can reuse its algorithm as a part. In general, most of [part chips] may be [chips] which have tight condition of connections. Because it depends on the way to design the [instance]. Anyway we have a possibility to reuse [chips] as [part chips].

4. Specification Description Language

4.1 ML as A Specification Description Language

As mentioned above, it is very important to describe rigorous and correct specifications with a formal specification language. Because, we must verify it at the stage of requirement of definition or design phase in the software development process. For this purpose, we think that a specification language has to possess the following factors.

(1) Clear model for calculation ⇒
Improve readability of a specification.

(2) Formalism to be able to analyze, verify and

transform ⇒

Static verification and generate executable codes.

(3) Executable ⇒

Dynamic verification like prototyping.

(4) Abstract description ⇒

Separate specifying from an implementation.

Viewed from these points, we have adopted a functional language ML (Standard ML) [5, 6] as our specification language. ML was developed at Edinburgh University as a language which describes a logical proof system and its features are the following.

1. Higher order function

To be able to treat functions as data, so we can describe specification compactly by subdivision of processes.

2. Pattern matching

We can describe variable patterns as arguments of a function.

3. Exception handling

ML has many expressions for handling exceptions and we can describe complex operations using an exception identifier.

4. Type inference mechanism

ML is a strongly typed language and has a mechanism to infer types of their expressions [8].

5. Polymorphism

We can define a function with any type represented by type variables. Such a function whose types are type variables performs a general-purpose function and we can reuse them in various cases.

Correspondence with above factors, we will mention the reasons for adopting ML as our specification language.

(1) The calculation model of ML is a functional model. A function has two aspects. First is to express a clear correspondence as a mapping from a set to another set. This performs a role of a message between [instances]. Second is to be regarded as a process to compute output values from input values. When we specified requirements, whether to describe "what is a requirement" or "how to satisfy a requirement" is often discussed. In this case, above two aspects it seems that the former express 'what' and the later express 'how'. We think that is important for a specification language is to have semantics which do not depend on their process to compute. Moreover, this enables the improvement of the readability and portability of a specification. The concise description using higher order functions enables us to easily determine errors of function's behavior. The above nature may allow us to say that functions have what it takes to be a specification language.

(2) ML is based on the theory of lambda calculus and we can make use of the various results of lambda calculus. For example, we can translate a source program in ML into its categorical combinator code according to categorical combinatory logic and execute it on an abstract machine.

ML is a strongly typed language. In general, it is a troublesome task for programmers to write types explicitly in their programs. However, in case of writing a specification, to define data types of a function which means defining its input and output values as data types will be useful for us to clear our purpose in describing its functional property. Moreover, its type inference mechanism enable us to verify a specification.

(3) Functions are rules of calculation and we can execute them by reducing expressions. It is possible to execute them symbolically. Accordingly we can execute a specification with some undefined symbols.

(4) When we describe a specification, we must describe it at the independent level to implementation environments (that is, characters of machine and a construction of windows etc.). It is desirable that we describe what is required essentially and translate them into implementation programs automatically or interactively. And we can generate a program from a specification by its instantiation. This means that one language plays several roles which correspond to its abstraction level. We can make such an abstract description using ML and it may be possible to translate them into implementation programs.

For the reasons mentioned above, we adopted ML as a formal specification language. Nevertheless, there are some problems to use it as a specification language. For one reason, as what is generally said, it is difficult to understand a formal language and hard to get used to its formality. In spite of that, probably supporting tools enable to cope with this difficulty. And we think that we should overlook some difficulty for improvement of the quality.

For another, ML does not have a mechanism to describe parallel processes with independent actions. A specification language PAISLey [4] based on an operational approach concept is also a functional language. And besides this, we can simulate the actions of parallel processes by describing some functions named exchange functions in which communicate each other. To simulate the actions of parallel processes, it may be necessary to extend ML itself.

4.2 Adaptation ML to DMC

We adopted ML (Standard ML) as a formal specification language. To use it in the design step on DMC, we added a mechanism of modularization [7] to it. An [instance] specification consists of the signatures and declarations of《type declaration》,《value declaration》,《exception declaration》and its lower [instances] (Table 2). Each declaration corresponds to the definition of a data structure, a process and error handling respectively.

4.3 Translation ML into Categorical Combinator Code

The way to implement a functional language using the theory of categorical combinator logic was studied

recently [9]. Cousineau and others proposed an abstract machine CAM (Categorical Abstract Machine) based on this theory and designed a processor of ML (this is not Standard ML) using this idea. We propose to extend this idea to the whole Standard ML without altering functions of CAM.

5. Supporting Tools

dmCASE is based on the method and the specification language which we have explained in section 3 and 4. In this section, we mention how to support our method and language and how to make use of these features as tools.

A design phase is a process on which we will repeat trial and error and so we should support these process smoothly. Our basic policy is the following. First is to construct user interface which matches human thinking. Second is to realize an environment where we devote ourselves to essential works of design by giving moderate restrictions which never disturb our thinking. Accordingly we notice that the following viewpoints are important for an environment.

•Verification: A formal specification language enables us to verify statically at the early stage of the design phase. We make use of the rules of our method and language as knowledge to verify. There are two levels of verification. The first is to guarantee a consistency in the model using a [conceptual structure chart] and that there is no inconsistency in the process of description. Second is a verification using types of function which form [instance].

•Visualization: This is an indispensable aspect for ease to understand a design process. Subject to visualize is for example, a model of a system, data structure and algorithm of process. Such a visualization aims at improving the understanding of a specification and understanding its actions.

•Information arrangement: At the stage of design with trial and error, it is a very important problem how to arrange numerous and various information. A system should provide a plain mechanism to arrange them. We suppose the following mechanisms or functions for example. A mechanism that users can select and keep some available data from much information offered by a system. Moreover, a mechanism to judge its efficiency. A function to support to make notes what come to mind, for instance an algorithm memo which means to arrange several chips of algorithm.

•Generation of testing data: Test is one of the ways to verify the specification dynamically. However, there is no way to do a complete test for the present. Nevertheles, there are several ways to support a test phase. For example, to derive the restrictions of data, that is, creating a template of input data using types of functions. Moreover, generating test data automatically to some extent is considered.

•Guidance: We should use various information which occurs at the design phase effectively. In order to guide users effectively, it is necessary to propose a mechanism of hyper text and interactive user interfaces and represent various information visually.

•Navigation: There are several rules on method. So a system enables to lead what a user can do or what a user should do at the next step. If the system controls us completely, our work will not get along well. So we need to consider a moderate lead.

•Monitoring: We hope an environment where we devote ourselves to essential works of design. Accordingly the system should monitor design steps and provide some information as the need arises.

•Documentation: At first the requirement is written by informal natural language. After we translate these into the formal descriptions and verify them, we need documents written by natural language which reflect intentions of the formal specification. It follows that they are easy to understand. This documents aim is not to maintain them but to assist in understanding the system.

•Flexibility: Human thinking is not always systematic such that it is sometime top-down as well as bottom-up. Particularly, the describing step is a process of trial and error, so that the system should keep up with various situations flexibly.

6. Outline of dmCASE

6.1 Design Process on dmCASE

The design process on dmCASE that we recommend as a paradigm of a new software development is the following.

(1) Process of constructing a system model

Requirements given at first stage are written by natural language, that is, an informal specification. From this informal specification we construct a system model on DMC and describe specification of [instances] using ML. We notice that at the present we don't support the space design mentioned at section 3.2.

(2) Simulation Process

By giving data to the described specification and executing it, we will verify the specification dynamically. As this results, we must return the previous process and reconstruct the model of specification.

(3) Translation Process

After a confirmation based on simulation of specification, we will translate this specification into efficient categorical combinator codes and execute them on an abstract machine. By the way, we have not realized a translation mechanism as that mentioned in section 2, so that we omit its explanations.

6.2 Compositions of dmCASE

We speak of the system design method on dmCASE using a notion of space design and plane design mentioned at section 3.2. First, as the space design, we con-

sider the following six subsystems. This idea means that we divide the system's whole works into several parts at the static viewpoint. Next we extract various [polymorphic chips] which represent each ⟨concept⟩ appearing on dmCASE. Using them each charge of layer of the system model designs own subsystem=a [board] as the plane design. This result has been shown in the previous Fig. 3 as the system model of dmCASE. In the Fig. 3, each layer represents a [board] and each word encircled by a square on it expresses a [chip]. When we look at the system from a static viewpoint, each [board] plays the following role.

•Description System [13]: This system realizes various functions to support describing a specification using the [word table], [conceptual structure chart], [instance definition table] and [syntax oriented editor].

•Static Analysis System [14]: This system realizes to verify a specification statically and gets various information using the rules of DMC and ML as the following. To check the usage of words arranged in the [word table]. To guarantee the consistency between the relationship among several [instances] and the specification composed of such [instances] on the [conceptual structure chart]. To verify the process specification (the description of value declarations) by the type inference mechanism.

•Monitor: To protect users from a cntradiction in the process of description. To show the need to execute the specification again with the change of description, etc. . . . Monitor realizes to watch the process of constructing a system model and simulation process like this.

•Dynamic Analysis System [14]: To analyze a specification dynamically, this system realizes the following functions. To execute a specification with a [card]. To generate its execution environment. Moreover, tracing, breaking, understanding an execution path and displaying the process algorithm, etc. . . .

•Translation System: This system realizes to translate the specification into its categorical combinator codes and execute them with an abstract machine.

•Knowledge Base System: This system realizes to collect and manage various design knowledge such as a [word table], a [conceptual structure chart], [instance] specifications, [cards] etc. . . .

On the other hand, the design process mentioned at section 6.1 expresses dynamic parts of the system on the system model shown in the Fig. 3. The relationship between each [boards] as above mentioned and this proc-

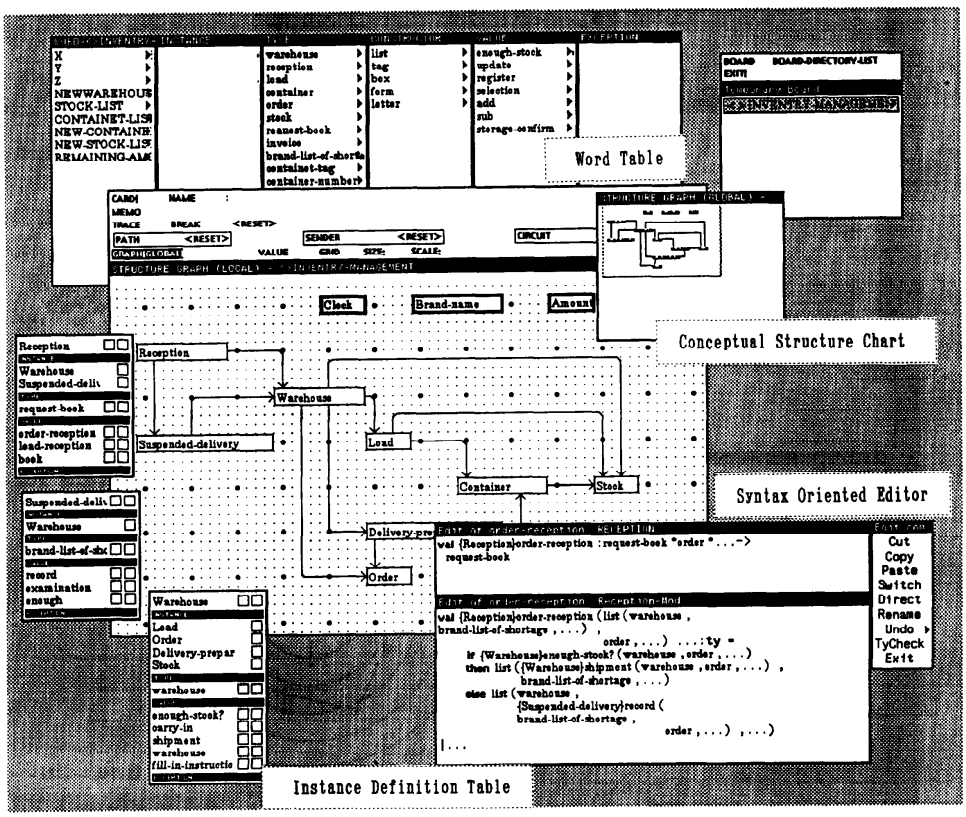

Fig. 4 The word table·The conceptual structure chart·The instance definition table·The syntax oriented editor.

ess is the following.

(1) In the process of a constructing system model, while writing our specification with the description system, we verify it with the static analysis system. And at the time the monitor system watches this process.

(2) In the simulation process, based on various accumulated knowledge the dynamic analysis system analyzes and verifies the specification dynamically and at the time the monitor system watches its analysis information.

(3) In the translation process, the translation system translate our specification into its combinator codes and executes them.

And various information accumulated at each process is controlled by the knowledge based system.

Moreover, among several [chips] appearing in figure 3 which consist of dmCASE, there is something to perform a part of user interface. That is, the word table, the conceptual structure chart, the instance definition table, the syntax oriented editor, the memo note, the card, the card panel, the circuit chart, the scope and the map of circuit chart. These [chips] are instantiated in each subsystem. And they appear on the various processes as an [instance] and form a concept of the [board] as an interface with users. The following explains these ⟨concepts⟩ and in the explanations we show the part as supporting tools which are mentioned in section 5 as [[ ]]. And the figure 4 and figure 5, show an example which describes the problem of the inventory management system [12]. Then we use this example with the following explanation.

(1) Word Table

This is a table which aims to register various words which represent [chips] extracted from requirements respectively. And its other aim is to arrange them to get much information to describe a specification (see Fig. 4). The word table consists of six attribute columns, for example instance, data type, process, etc. . . . To arrange them with their attributes enables us to decide the usage of extracted [chips]. For example, we use 'Warehouse' and 'Reception' as an instance with which form a problem structure and 'order-reception' as a process name which represents a task to receive orders. To arrange words with their attribute is useful for us to simulate the system model as a mental model. Moreover, it gives some constraints which decrease to occur many contradictions such that the usage of words are incorrect in the specification. And the word table enables us to give some information as to how to use these words. The main works of the word table are functions such as in-

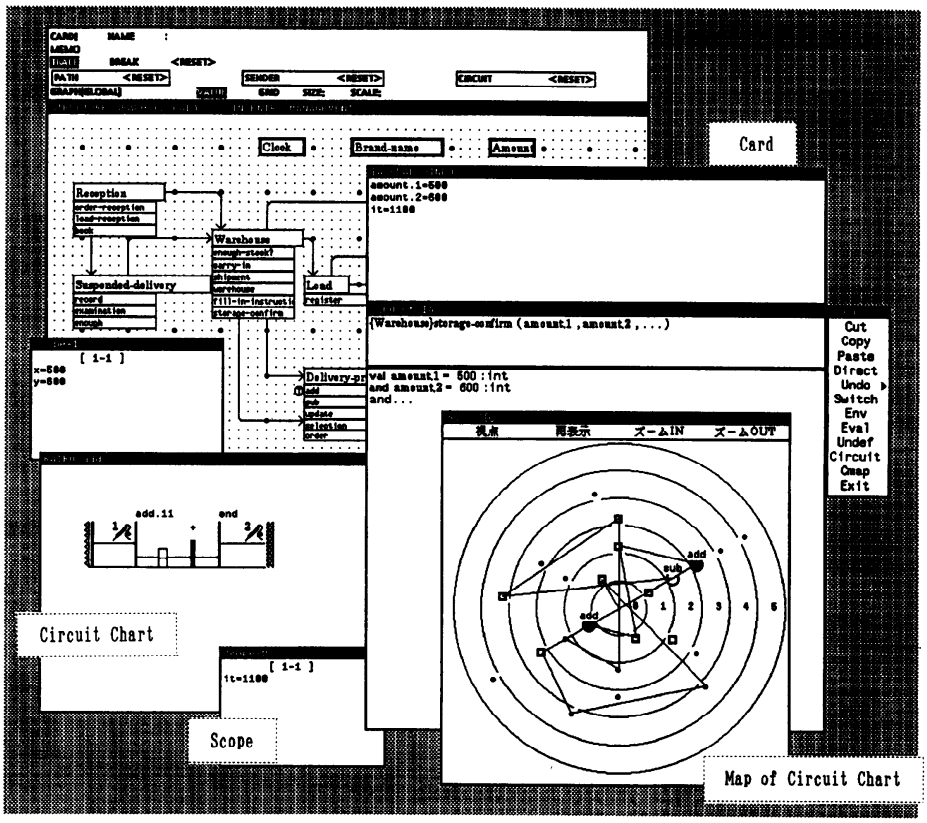

Fig. 5 The card · The circuit chart · The scope · The map of circuit chart.

put, move, copy, delete, search, rename. [[Information Arrangement]] [[Verification]] [[Flexibility]]

(2) Conceptual Structure Chart

The conceptual structure chart is a chart which expresses the relations between several [instances]. It explains the problem structure using several [chips] working a central role of explanation (see Fig. 4). This aim is to create a natural model as repeating a trial and error in human thinking. Figure 4 shows that the instance 'Delivery-preparation' refers to two instances, that is, 'Container' and 'Order'. While it is the way to catch the entire structure of a problem at the describing phase, it enables us to observe this model from different viewpoints by confirming process actions and searching data and processes on it. The main works of the conceptual structure chart are functions such as editing chart, display a process path, display a circuit chart, tracing, breaking, etc. . . . [[Visualization]] [[Verification]] [[Flexibility]] [[Guidance]]

(3) Instance Definition Table

The instance definition table is a table on which we register various [words] which form each [instance] specification (see Fig. 4). This means that we list what we want to do in the [instance] and decide the outline of behavior of it. Figure 4 shows that the instance 'Reception' consists of one data such as 'request-book' and three processes such as 'order-reception', 'load-reception' and 'book'. We register the appropriate word to a destination instance by selecting from the word table and indicating the position of the destination instance definition table or instance on the conceptual structure chart. There is some functions such as move, copy, delete of words and starting an editor, display a list of [instance] specification, display a description degree of specification. Displaying a list of [instance] specification is a textual expression with a form like Table 2. On dmCASE, we can write a specification in Japanese and so appropriate usage of words will allow us to make clear our intention of [instance] to the readers. A process consists of some lower processes by referring them. So a display of a process specification is a function to express such a reference relation as a textual form. This function correspondences to another textual representation of a function to display a process path on the conceptual structure chart. [[Information Arrangement]] [[Documentation]] [[Monitoring]] [[Flexibility]]

(4) Syntax oriented Editor

The syntax oriented editor is an editor of ML. And we write an [instance] specification with it (see Fig. 4). We edit them using template of ML syntax elements by following two ways. First is selecting a syntax element from the menu of syntax components of ML. Second is selecting words from either the word table or the instance definition table. Then we can refine them step by step. Accordingly its syntactic correctness is always guaranteed and we will be released from syntactical mistakes. The calling form of process is derived from its signature. For example, the pattern 'record (brand-list-of-shortage, order)' is derived from the definition of a process 'record' signature and embedded in the description of the process 'order-reception' by selecting a process 'record' from the instance 'Suspended-delivery'. The [instance] specification consists of its signatures and declarations. The signature of a process definition is a description of the types of its input and output values. These types can be decided using a type inference mechanism, but the other side, users own description of its types enables to verify its module definition and confirm their intention. The main works of the syntax oriented editor are functions such as editing, type inference, hyper textual interface, etc. . . . Moreover above ⟨concepts⟩ from (1) to (4) are very important in the description process and we proceed a design by traveling these four components. So we designed these components to be used without them being influenced by their operation's order. [[Navigation]] [[Verification]] [[Guidance]] [[Flexibility]]

(5) Memo Note

The memo note has a role to offer some information. With it we can confirm various contradictions caused in the description process repeating a trial and error and keep its correctness. If some trouble of correctness is caused, it will perform its purpose to preserve its informations for confirming and allow users to be able to

Table 2 An example of [Instance] specification.

① signature	LIBRARY	SIGNATURE	① signature	② Instance to inherit
② instance			③ type constructor	④ type of value declaration
③ type	library		⑤ exception identifier	⑥ end of SIGNATURE
④ val	book-add	MODULE	⑦ module name	⑧ Instance to refer
⑤ exception			⑨ type declaration	⑩ value declaration
⑥ end			⑪ exception declaration	⑫ end of MODULE
⑦ module	Library-Mod			
⑧ instance	Copy-of-book			
⑨ type	library is copy-of-book-list			
⑩ val	book-add(copy-of-book-list, book)=			
	let val numberofcopy=maxnumber(copy-of-book-list, book)			
	in store-bookshelf(library, copy-of-book-list			
	copy-of-book-list@copy(book, numberofcopy, −, −) end			
⑪ exception				
⑫ end				

revise it correctly using this information. More precisely, it watches the influence of delete operation of words. When we delete a function or data which is referred from other definitions, it will inform users of its influence and confirm whether the word may be deleted or not. After confirmation by the user, it keeps the information amended as a memo note. For example, in Fig. 4, if we delete the process 'enough-stock?' referred in the definition of the process 'order-reception' from the instance 'Warehouse', we should revise the definition of the process 'order-reception'. In such a case, a memo note with this informations is created as above mentioned. If we have revised it, it may be destroyed. Moreover, depending on the situation, the revival of its definition may be possible. The work of a memo note is the following. To watch the description process always and renew its information. Moreover, to propose them to users as the need arises. [[verification]] [Monitoring]] [[Information Arrangement]]

(6) Card

The card is an environment in which the description is executed (see Fig. 5). We can execute ML description by evaluating its《expressions》. We indicate an objective process name and give its input appropriate values. In the Fig. 5, the《expression》storage-confirm (amount. 1, amount. 2) was evaluated as the situation amount. 1=500 and amount. 2=600. We can generate several environments of input values and execute them in various situations. The environment of input values is written by the way to fill a template. The input types or patterns indicated process are derived its process definition and the system generates its input patterns. Moreover, a pattern of type in the derived patterns are also derived its definition. The environment of input values is derived from the definitions, so that we can get considerable test data from the definitions. Moreover, as we can execute《expressions》including undefined symbols, so we can examine its execution in the halfway of derivations. Now we can only derive types of test data. However, analyzing process definitions will enable us to give some constraint of input values. The main works of the card are functions such as editing (derivation, delete, substitution), display an undefined symbol, evaluation, generate more than one environment, starting a circuit chart, etc. . . . [[Verification]] [[Generation of testing data]]

(7) Card Panel

On dmCASE we can analyze a specification dynamically by executing each process using the card. The card panel is a mechanism to manage the whole of these cards. Cards are managed at each instance to which a process belongs. And the main work of the card panel are functions such as starting a card, card delete, translation to categorical combinator code and execution and display card information. Card information is composed of the need of reexecution and retranslation with changing the descriptions and the state of each execution (its input and output value) and a user note about an execution result. [[Information arrangement]] [[Guidance]] [[Monitoring]]

(8) Circuit Chart

The circuit chart is a graphic structure which represents the described process specification. That is, this correspondences to another graphic representation of a function to display a process path on the conceptual structure chart and a function to display a process specification on the instance definition table. As ML is based on the theory of lambda calculus, we represent the state of valuables binding and the structure of expressions as various graphic patterns (see Fig. 5). The circuit chart aims to play a role of understanding and confirming the described process algorithm and provides some information about valuables binding and branches at the execution by the card. The circuit chart has a hierarchical structure which consists of several [blocks] being composed of [vertical line], [horizontal line] and [rectangle]. A [block] expresses a process and a [vertical line], a [rectangle] and a [horizontal line] represents a wall of lambda binding,《expression》and a flow of data respectively. If an《expression》has some components, it will expand its structure in the lower direction. The circuit chart in Fig. 5 shows a structure of the process 'add' graphically. [[Visualization]]

(9) Scope

The scope plays a role to look through the binding situation of some variables when process specification is executed. And it starts from the circuit chart (see Fig. 5). Figure 5 shows the input values (the upper scope) and the output values (the lower scope) of 'add' process to which 'storage-confirm' process evaluated in the card refers at run time. Looking at these values, we will confirm whether expected values are bound. Necessary information, what we want to know for confirming the actions of processes is to find out where errors have occurred. As debugging method, to execute the process specification partially on the circuit chart is a question under consideration. So we would like to use this scope as a setting value interface of the occasion. [[Information Arrangement]]

(10) Map of Circuit Chart

The map of circuit chart shows the state of process specification at run time by representing its hierarchical level of process structure as a concentric circle. It shows the whole structure of process in the form to put together information of the circuit chart (see Fig. 5). More precisely, it is a concentric image to which its hierarchical level of process structure maps. Moreover, a process, a structural expression and a constant or valuable are illustrated on it by symbols [○], [□] and [•] respectively. In the case that we set a trace or break point to the process, the symbols become [●] or [◎] respectively. At the present, it is a medium which enables us to understand the execution state by displaying the path at run time and to observe the valuable state of the process set trace or break points by opening the correspondence circuit chart. Figure 5 shows the map of cir-

cuit chart of 'storage-confirm' process executed by its card. This map shows there are two processes such as 'add' and 'sub' which are referred from the 'storage-confirm' process. And it also shows its trace of execution by several lines. When a trace point set to 'add' process, if the symbol [●] on the map is selected, the circuit shown in Fig. 5 will be opened and we will get some debugging information. We would like to find the instruction and branch coverage of processes by analyzing these execution paths and display such quantitative results to the map. [[Verification]] [[Visualization]]

Finally we mention its operational features. This design process is a process of trial and error, so that a consistent operation should be executed to avoid disturbing our thinking. The issue of operation using a keyboard and a mouse is discussed and mingled operations may be a troublesome task. The operation using only a mouse is probably suitable for a trial and error process. We then decided that inputting only words by keyboard and other operations executed by selecting and using a mouse.

7. Evaluation by Some Experiments and Further Problems

To actually use the software development environment that we proposed we must make clear what the problem is and whether there is something lacking. Accordingly we tried some experimental descriptions for the problems such as the inventory management system [12], the lift control system [10] and the library system [10]. We discuss evaluations of dmCASE and some of its problems through these experiments. We noticed that these experiments placed emphasis on the process of constructing a system model. And we evaluated it from the viewpoints such as the description ability, the readability of specification and the user interface.

(1) Description Ability

In these cases, we created each model as one [board], because the scale of each problem is small. We described all processes which realize the requested functions, but it turned out that the present mechanism can't deal with the modeling of parallel processes with the synchronization between processes such as the lift control problem. Nevertheless, for the lift control problem, we treat it as a model of processes acted with states, that is, a state transition model. The structure represented by a [conceptual structure chart] is static structure. In order to simulate parallel processes with independent actions from each other, we should expand the kernel language or add a mechanism describing event sequences to it.

(2) Readability

The formal specification of dmCASE consists of a [conceptual structure chart] and several [instance] specifications. The [conceptual structure chart] is useful to understand the whole structure of the problem. This experiment is made by one examinee. It is expected that another examinee will create another model. However, a [conceptual structure chart] is a moderate material to discuss the system model at the halfway stage of design. It is not easy to discuss using very detailed materials, but explaining with a [conceptual structure chart] which has an extensive view may be easy to understand. Moreover, an [instance] specification is a description of each [instance] behavior, so that we would like to understand their intention by reading them. On dmCASE, we can write them in Japanese. So it seems that effective usage of words enables us to replace declarations written by ML syntax with sentences written by a natural language. And we use them as a document. Accordingly selecting and using words are key points in the above matter and it is desirable to select them effectively so that we take the intention of a requirement into consideration and use appropriately. Moreover dmCASE offers a mechanism of changing the name of a word to more suitable word after selecting it. This mechanism is the function of renaming words. So this leads to improvement being easy to understand.

(3) User Interface

It may be suitable for human thinking to give some conditions to support a describing process and execute most operations using only a mouse. Moreover, we think that a mechanism supporting information arrangement should add to the frame of dmCASE. This mechanism means the following. As the former stage of describing by ML, we make some notes of process algorithms and data structures that came to mind within the framework of ML with guidance information. We think that this allows us easily to understand ML language.

8. Conclusion

It is not enough to keep up with improving software productivity and reliability that we only systematize an old traditional way of design. We must construct a new environment which has a clearer framework of design, so that we develop reliable software. We think that such an environment enables one to improve software productivity. In this paper we proposed a new environment dmCASE based on three pillars such as the design method, the formal specification description language and supporting tools. We have not realized the following yet. A space design using multi layered [board]. Translation a described formal specification to an informal document. Translation a description using ML at abstract level into implementation programs. Considering the above facts including those mentioned in section 7 as future directions, we would like to see growth in dmCASE.

Acknowledgement

A part of this study is the "Formal Approach to Software Environment Technology" project produced by

Joint System Development Corp. supported by the Information-technology Promotion Agency (IPA).

References

1. BALZER, R., CHEATHM, T. and GREEN, K. Software Technology in the 1990's: Using a New Paradigm, *IEEE Comput.*, **16**, 11 (Nov. 1983), 39–45.
2. BURSTAL, R. M. and GOGUEN, J. A. Putting theories together to make specifications, *Proc. of 5th IJCAI* (Jul. 1977), 1045–1058.
3. FTATSUGI, K., GOGUEN, J. A., JOUANNAUD, J.-P. and MESEGUER, J. principles of OBJ2, *In Proc. Symposium on Principles of Programming Languages, ACM* (1985), 52–66.
4. ZAVE, P. An Overview of the PAISLey Project-1984, **SIGSOFT SEN**, **9**, 4 (1984), 12–19.
5. WIKSTOM, A. Functional Programing using Standard ML, Prentice Hall International, Hemel Hempstead, 1986.
6. MILLNER, R. A proposal for standard ML, *Conference Record of 1984 ACM Symposium on LISP and Fun-ctional Programming*, ACM (Aug. 1984), 184–197.
7. MACQUEEN, D. Modules for standard ML, *Conference Record of 1984 ACM Symposium on LISP and Fun-ctional Programming, ACM* (Aug. 1984), 198–207.
8. YOKOTA, Type inference and ML, *bit*, **20**, 3 (in Japanese), 74–83.
9. COUSINEAU, G., CURIEN, P-L. and MAUNY, M. THE CATEGORICAL ABSTRACT MACHINE, *Lecture Notes in Computer Science 201*, Springer-Verlag (1985), 50–64.
10. Problem Set for the 4th International Workshop on Software Specification and Design, *Proc. of 4th international workshop on Software Specification and Design* (1987), ix-x.
11. SEKINE and OHBAYASHI, A new programing method—Design Method based on Concepts (DMC), *bit*, **14**, 7 (1982) (in Japanese), 102–114.
12. OHBAYASHI, Declarative Specification of an Inventory Control System by the Design Method based on Concepts, *Softw. Eng.* **58**–5 (1988) (in Japanese), 33–40.
13. MATSUURA, NAKAZATO and OHBAYASHI, Implementation of a Program Development Environment with the Design Method based on Concepts, *Softw. Eng.* **58**–6 (1988) (in Japanese), 41–48.
14. MATSUURA and OHBAYASHI, An Analysis Support environment in the Program Development System dmCASE, *Symposium of CASE Environment* (1989) (in Japanese), 117–124.