Software Prototyping with Reusable Components

SHINICHI HONIDEN,* NAOMICHI SUEDA,* AKIRA HOSHI,*
NAOSHI UCHIHIRA*, and KAZUO MIKAME*

Recently, a prototyping method has attracted attention as a software specification method. Though many methods have been proposed, no standard method has been established. This paper proposes a prototyping method with reusable components based on knowledge engineering. This method provides support to even non-expert personnel in selecting and combining individual software components to satisfy their requirements rapidly. The proposed method is realized by an expert system, which consists of a component inference part, a parameter inference part, and an execution part. The component inference part selects the appropriate components to satisfy a user's requirement, and to determine the purpose of combining components. The parameter inference part combines parameters based on logical attribute and physical attribute. The execution part activates and carries out the roles indicated in the combined components. As an application example, an image processing expert system is described.

1. Introduction

Prototyping, as a software specification method, has recently come to be highly regarded. Various prototyping methods have been proposed, such as the existing programming language method [1], a program generator method wherein the field of application is limited [2], software reuse method [3], and the executable requirement specification language method [4]. Among them, the executable requirement specification language method is more applicable to various areas than other methods. However, suitable executable requirement specification languages have not yet been fully developed, and prototyping systems based upon them are not available. In other words, in order to develop general purpose prototyping systems, such as the executable requirement specification language, many problems remain to be resolved. Therefore, the practical approach involves developing an application-oriented prototyping system.

Using a software library is one known method for application-oriented prototyping system, because a user can construct a prototype rapidly with only knowledge about the library. However, it is hard for a user who doesn't have that knowledge (such a person is called a non-expert in this paper) to use the library. To recognize this method as a prototyping method exactly, the method must include functions which allow even a non-expert to use the library easily.

This paper describes an expert system which provides support to non-experts in selecting and combining individual software components stored in the software library.

*Systems & Software Engineering Division, Toshiba Corporation, 1-1-1, Shibaura, Minato-ku, Tokyo 105, Japan.

As an application example, an image processing expert system is described. When the technology used for image processing is reviewed from the standpoint of pre-processing in two-dimensional image processing, established methods have been readily available in various areas. Many software packages for that purpose have been developed and used. Among them is SPIDER (Subroutine Package for Image Data Enhancement and Recognition) [6]. SPIDER consists of approximately 350 subroutine packages, which cover various processing areas, such as orthogonal transformation, emphasis and smoothing of image-data. They are described in FORTRAN.

Engineers engaged in image processing must choose appropriate software components among the group of subroutines (components) and develop programs to combine such components. To obviate such troublesome jobs, facilities have been requested which automatically combine, execute, and verify necessary components without programming and which generate and register the resultant combination as a new component.

In short, the requirements are:

*Each component should be used without programming.

*A combination of components should be automatically generated.

To satisfy these requirements, the image processing expert system was studied and developed.

2. Software Reuse Based on Knowledge Engineering

The proposed method involves software prototyping with reusable components. The software reusability rate has been increasing in various applications, and software reuse contributes to higher software produc-

tivity. If software can be treated as components, the components can be reused, easily modified, and easily understood with better reliability and maintainability. Many problems, however, must be solved to realize the idea. In order to stock and reuse software components, internal and external specifications for each component must be clearly defined. Also, environments where the components can be easily utilized must be provided. The authors defined the environment, in which each component can be easily used, as the one which satisfies the following criteria:

(1) Each needed component should be searched easily.

(2) Components searched for and found should be easily combined.

(3) The resultant combination should be immediately executable.

This environment provides ways to construct a prototype rapidly.

There are two software reuse methods,

(1) Customize general component to satisfy the user's requirement, that is, modify the internal structure in the component (called a white-box approach in this paper).

(2) Use components without modifying the internal structure (called a black-box approach in this paper).

The white-box approach is applicable to various areas and has high flexibility. However, the user must have knowledge about internal structure, and computerization is very difficult. The black-box approach is applied to only a limited domain, but automatic selection and combination is possible with knowledge about only external structure such as function and interface.

This paper describes a black-box approach. A typical example of the black-box approach is using a software library. That is, components stored in the library are used without modifying the internal structure. Many software libraries have been presented. However, it is hard for the library user to select and combine reusable components. It takes non-experts, who don't have knowledge about the library, much time to select and combine the components. To solve this problem, the authors developed an expert system for the education and assistance of non-expert users.

The presented expert system consists of:

(1) Component inference part

(2) Parameter inference part

(3) Execution part

The component inference part selects the appropriate components to satisfy a user's requirement, and determines the purpose of combining components. The parameter inference part combines parameters based on physical attribute, logical attribute and the purpose of combining components. The execution part activates and carries out the roles indicated in the combined components.

3. Image Processing Expert System

The image processing expert system consists of a component inference part, a parameter inference part, and an image processing execution part, as shown in Fig. 1.

3.1 Component Inference Part

This section explains component inference for user's requirements and dealing with the purpose of combining components. Its general idea is that, when a user requests processings, the system should automatically select appropriate software components.

The component inference part is realized by an expert system. An expert system consists of knowledge-base and inference engine. Knowledge representation and inference mechanism are described in the following.

3.1.1 Knowledge Representation

There are two knowledge categories, knowledge about components and knowledge about state.

The knowledge structure about the components is divided into the following categories:

(a) Meta inference (ex. for process A-process is a set of components-, processes a1, a2, a3, or a4, a5, a6 are performed).

(b) Effects on the state (ex. when process a1 is performed, state changes).

(c) Relationship between before and after processing (ex. process a1 should be performed prior to process a2).

The knowledge structure can be described using the

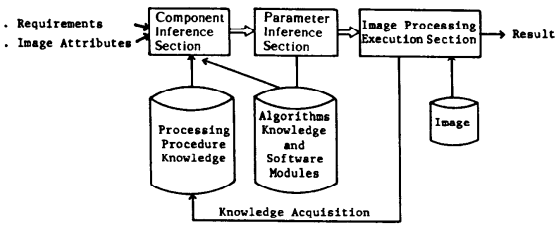

Fig. 1 Image processing expert system.

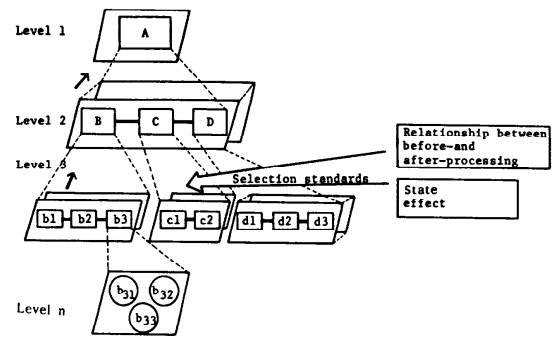

Fig. 2 Knowledge structure.

hierarchical structure shown in Fig. 2. The examples are as follow;

A: Segmentation, B: Binarization, b1: Histogram, b2: Threshold selection, b3: Binarization, b31: Single thresholding, b32: Double thresholding

This knowledge is not systematic, it is stored in fragments, and may correspond with, be inferred from, or be related to the state when the knowledge is used.

```
Rule No.: rule 21
Process request part:
Condition part: IF
        (state a and
         state b or
         state c) or
        (state d and
         state e and
         state f)
Execution part: THEN
        (component E and
         component F and
         component G)
Effect part: (state h and
         state i and
         state j and
         state k)
Parameter combination part:
        (e2 in E=f1 in F)
        (f2 in F=g1 in G)
```

Fig. 3 An example of knowledge about component.

```
Noise frame
Features
  Shape      : Circle, point, line and band
  Color      : Monochrome
  Size       : Minute and small
  Uniformity : Yes or No
Range
  Shape      : Optional
  Color      : Optional
  Size       : Optional
Effects
  Visibility : Hard to see
  Supplement : If the density is high, this density will affect optional
               processing.
```

Fig. 4 Representation example of image attribute.

Knowledge about a component is the rule which consists of rule number, process request part, condition part, execution part, effect part, and process result state. An example is shown in Fig. 3.

Knowledge about the state is closely related to the application. Here, "state" means "image attribute" in image processing. The structure and representation for the image attribute are given in Fig. 4. The specialist, to whom the image to be processed is given, should understand:

(i) Overall image
(ii) Target conditions
(iii) Relationship with the background
(iv) Noise, Strain, Diffusion

Examples of such knowledge about image attribute are as follow.

*Background is in an area other than the target area.
*Background is often a large area with the same features.
*Background is often composed of two or more textures.

3.1.2 Inference Mechanism

The inference mechanism to select the software component makes non-deterministic inferences. It adopts a forward inference method, in which the inference is made under an assumption that "if the given states are *a*, *b*, *c*, . . . , perform processes *A*, *B* and *C*." This mechanism is shown in Fig. 5. Levels 2, 3 and *n* in Fig. 5 correspond to the processing component knowledge levels 2, 3 and *n* given in Fig. 2, respectively. The mechanism unifies the fact list with the condition rule. After unifications, the fact list is updated by the changed state.

3.1.3 Rule for User's Intention Regarding Parameter Combination

As mentioned in 3.1.1, in order to select the components, the rule shown in Fig. 3 is used. For example, components E, F, G are selected in Fig. 3. Knowledge

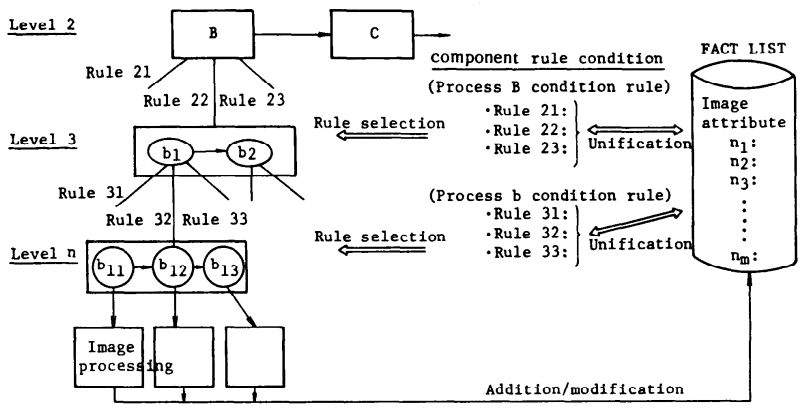

Fig. 5 Inference Process.

about the combination of parameters is also described in this rule. This rule shows that parameter e2 in component *E* is related to parameter f1 in component *F*.

3.2 Parameter Inference Part

This section explains parameter inference mechanism. Parameter inference means parameter matching among selected components in the component inference part.

It would be ideal for users if combination among components could be performed without any user assistance. To determine the relationship between components, the information shown in Fig. 6 is generally used.

Each parameter has three attributes, physical attribute, logical attribute and the purpose of combining components. An example of physical attribute is shown in Fig. 7. The logical attribute means the attribute heuristically determined, i.e., some intention, such as "Let's assign such and such a role to this parameter" is kept in mind. The authors consider that the probability of making right choices by automatic combination could be improved by introduction of the logical attribute. When the logical attribute concept is introduced, and as long as one to one correspondence between one logical attribute specified by a parameter and the other logical attribute by another parameter is retained, the character matching method is satisfactory. However, meanings of parameters do not correspond on a one to one basis. The example shown in Fig. 8 indicates this situation. In this example, the logical attribute for parameter "RHST" of the component "HIST1" is defined as "Histogram data" and that for parameter "DATA" of "THDS" is defined as "percent data". If a simple character matching is applied to this particular example, these attributes to not match and cannot be combined. To solve this problem, the inference mechanism based on the defined attribute is required.

The information regarding "purpose of combining components" is also indispensable for automatic combination. This can be shown in the following example.

CALL STAS (A, MEAN, MAX, MIN, STDEV)
CALL DIVCIR (B C)

The average value (MEAN), maximum value (MAX), minimum value (MIN), and standard deviation (STDEV) for distribution A are obtained by "CALL STAS" first. Then, value B is divided by a constant value C by "CALL DIVCIR." At this time, the value of B can be related to the value of MEAN, MAX, MIN, or STDEV. Only the user knows the right selection. Therefore, knowledge about the purpose of combining components is required to combine such components automatically. This knowledge is involved in component knowledge described in 3.1.

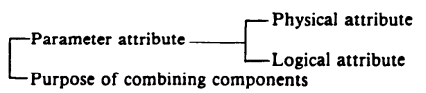

Fig. 6 Relationship between components.

* Input/output made: Input, output, input/output, . . .
* Dimension
* Element size of dimension: Constant, depending on other parameters, . . .
* Input type: Keyboard input, constant, operation between other parameters, . . .
* Input value range

Fig. 7 An example of physical attribute.

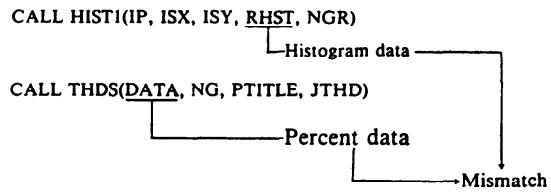

Fig. 8 An example of parameter mismatch.

3.2.1 Component Attribute Knowledge

The attribute structure for each component is organized as shown in Fig. 9. An individual component and its parameters make up an individual frame. Each logical attribute definition also takes the form of the frame. Each frame inherits necessary information from upper-level frames.

3.2.2 Rules of Combination

The component combination inference is the backward inference method and it is a non-deterministic method. The rules used for combination consists of rules for selection, rules for evaluation, and rules for determination.

(1) Rules for Selection

The selection based on the logical attribute is inferred by interpreting the meaning of each parameter using the upper-level concepts. However, if a structure of knowledge to define the logical attribute includes extremely abstract upper-level concepts, matching of some components at some level does not have any practical sense. For example, it is easily understood that a structure such as (histogram data) (percent data) . . . (numerical values) does not have any practical sense, even if components are related to each other, based on the numerical values concept. To avoid such a case, care has been taken to determine a way to furnish knowledge regarding the definition of logical attributes. Also, the concept of logical distance (the number of inheritance links) has been incorporated to quantify the discussion. Thus, the difference in logical distance can be checked, after the logical attribute between

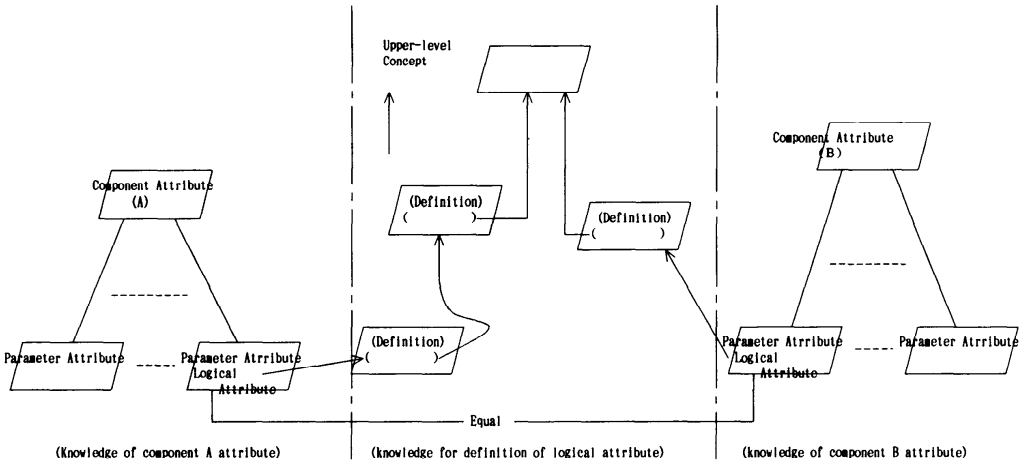

Fig. 9 Attribute structure.

parameters is matched.

(2) Rule for Evaluation and Determination

Candidates selected using the rules for selection are finally determined. For the parameters to be related, the physical distance is calculated based on the positional relationship (order of candidates in the coded program) of the candidates to be related. The index wherein parameter "*a*" will be related to parameter "*b*" is calculated, as shown below.

$f(a, b) = \alpha 1 *$(logical distance between "*a*" and "*b*")+
$\alpha 2 *$(physical distance between "*a*" and "*b*")

$\alpha 1$ and $\alpha 2$ are weighting coefficients.

The $f(a, x)$ having the smallest value is selected, using the determination rules:

Determination (a, x):- comparison $(f(a, x) < f(a, y))$.

For the parameter inference part, Fig. 10 shows an example of a Prolog fact statement which is component attribute knowledge, while Figs. 11 and 12 show examples of Prolog statements, for selection rules and evaluation rules, respectively. Figure 13 shows the system flow for the image processing expert system, which selects components needed by the user, relates to parameters for each component, generates a new component, and then registers them the library and the component attribute data.

```
class(histl).
parameter(histl, ip, isx, isy, rhst, ngr).
class(ip).
ako(ip, histl).
mode(ip, input).
dimension(ip, 2, para, isx, para, isy).
class(isx).
ako(isx, isx. d).
class(isy).
ako(isy, isy. d).
class(rhst).
ako(rhst, histl).
ako(rhst, histogram-data).
mode(rhst, output).
```

Fig. 10 An example of knowledge expressions.

```
s_rule1(*x):-mode(*x, input).
s_rule1(*x):-mode(*x, input-output).
s_rule2(*x, *y):-front(*x, *y).
s_rule3(*x):-dimension(*x, *a),
            dimension(*y, *b),
            equal(*a, *b).
s_rule25(*x, *y):-semantics(*x, *a),
              semantics(*y, *b),
              equal(*a, *b).
relation(*a, *b):-s_rule1(*a),
            s_rule2(*a, *b),
            s_rule3(*b),
            s_rule25(*a, *b).
```

```
IF
  Input/Output mode of *a is input or input-output
      AND
  *b is set before *a
      AND
  Dimension of *a is equal to dimension of *b
      AND
  Logical attribute of *a is equal to logical attribute of
*b
THEN
  *a is related to *b
```

Fig. 11 An example of a rule expression (selection rule).

```
function(*a, *b, *f):-logical_length(*a, *b, *ll),
        physical_length(*a, *b, *pl),
        multiply(*ll, alph1, *x),
        multiply(*pl, alph2, *y),
        add(*x, *y, *f).
determination(*a, *x):-function(*a, *x, *xf),
                function(*a, *y, *yf),
                less(*xf, *yf).
```

Fig. 12 An example of rules expression(evaluation and determination).

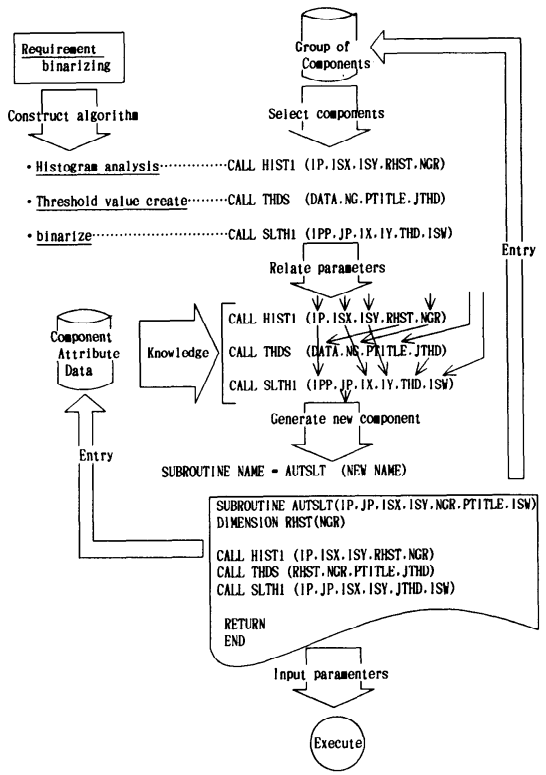

Fig. 13 Outline of component combination.

3.3 Execution Part

The execution part generates the executable format from combined components and activates and carries out the rules indicated in the components. The user's load for preparing execution environment is decreased, that is, additional codings for execution, such as parameter input routine and declaration, are generated automatically. The execution part processes the image data after obtaining necessary information through a conversation with the user.

3.4 Effects

The system has been verified with approximately 300 rules and approximately 70 software components. When processing is performed by a non-expert in image processing, about one hour is required to perform the steps from the introductory explanation to processing. System capability depends on the completeness of the knowledge base. Especially the rules for user's intention regarding parameter combination contributes to the system capability. These rules have two categories, the one beween components to be appeared in each knowledge about component and the one between components not to be appeared. Current knowledge bases do not have the latter rule. Therefore, when the combination fails, knowledge about the library is required. On the other hand, achievement of a target image depends on the component execution. Component functions are the subject of image processing technology, which should be further improved. As described previously, at present, the operator's load in the derivation of image processing procedures is markedly decreased.

5. Conclusion

A new method for use in software prototyping with reusable components is proposed. This method provides support to even non-experts, to enable them to select and combine the reusable components stored in a library rapidly. This method is realized by an expert system. As an application example, the image processing expert system is described. Currently, $\alpha 1$ and $\alpha 2$ in determination rules are set as 0.35 and 0.65, respectively, in the image processing expert system. However, to obtain better determination, $\alpha 1$ and $\alpha 2$ must be improved by practical use.

A problem to be studied in the future concerns the fact that the time required for inference will increase in proportion to the complexity of the inference procedure, when the number of components is increased and the associated quantity of the knowledge is increased.

This method can be applied to various applications. In this method, for the component inference part, there are two knowledge categories, knowledge about components and knowledge about state. The former knowledge is independent on applications, but the latter knowledge depends on the particular application. On the other hand, the knowledge used in the parameter inference part is independent from the application.

Acknowledgement

The authors would like to thank Akira Ito and Masahiko Arai, Systems & Software Engineering Division of Toshiba Corporation, for their valuable comments.

References

1. DUNCAN, A. G. Prototyping in ADA: Case Study, *ACM Sigsoft Eng.* Notes, **7**, 5 (1982), 54–60.
2. BARSTOW, D. Automatic Programming System to Support Experimental Science, *Proc. 6th ICSE* (1982), 360–366.
3. JONES, T. G. Reusability in Programming: A Survey of the state of the art, *IEEE Trans. Software Eng.* **10**, 5 (1984), 488–493.
4. BALZER, R. M. et al. Operational Specification as Basis for Rapid Prototyping, *ACM Sigsoft Software Eng. Notes*, **7**, 5 (1982), 3–16.
5. TAMURA, H., Sakaue, K. Three kinds of knowledge for building Digital-Image-Analysis expert system, Paper of Technical group AL83-49, *IECE Japan* (1983), 27–40.
6. TAMURA, H. et al. SPIDER USER'S MANUAL.
7. MIKAME, K. et al. Knowledge Engineering Application in Image Processing, *Proc. Graphics Interface* '85 (1985), 435–441.

(Received September 9, 1985; revised September 2, 1986)