Journal of Southeast University (English Edition) Mar. 2002 Vol.18 No.1 ISSN 1003—7985

Decomposition-Based Visual Function Specification and Auto-Generation of Function*

Shen Jun** Gu Guanqun

(Department of Computer Science and Engineering, Southeast University, Nanjing 210096, China)

Abstract: On the software module, this paper proposes a visual specification language (VSL). Based on decomposition, the language imitates men's thinking procedure that decomposes a problem into smaller ones, then independently solves the results of every small problem to get the result of original problem (decomposition and synthesis). Besides, the language mixes visual with specification. With computer supporting, we can implement the software module automatically. It will greatly improve the quality of software and raise the efficiency of software development. The simple definition of VSL, the principle of auto-generation, an example and the future research are introduced.

Key words: software specification, function decomposition, data dependent, visual programming

Software model has been going through three phases: procedure-oriented, object-oriented and component-oriented[1,2]. The development of the three phases is spiral rather than throwing off and repelling each other. The granularity of software development is bigger and bigger. Therefore, the thinking focus of software developer has shifted to design stage from implement stage. The level of abstract is higher and higher. The cycle of software development is shortened, and the quality of software is improved greatly.

Even through object-oriented modeling and cooperation of components have implemented visual development at a higher level, i.e. the design level, the functions of objects and components are implemented finally based on subprogram, called functions or procedures, especially to data server objects or components, and the visual specification and the automatic generation of subprogram have not gained breakthroughs in essential. Therefore, the visual specification and the automatic generation of subprogram are the key to ensure the quality of software and to shorten the cycle of software development.

On the implement of subprogram, according to the rules of cognition[3], this paper proposes a visual specification language (VSL), based on decomposition[4]. It is used to describe the specification of software module. Based on it, we have developed a visual development platform that is used to visual design and supports the reuse and the verification of specification[5]. With the visual development platform, user only needs to know the domain knowledge, and quickly accomplishes the design of module decomposition; finally, the specification and the generation of software module will be implemented automatically. It breaks the limitations of quick visual design at level of subprogram.

The rest of the paper is organized as follows. The simple definition of VSL is described in section 1. The principle of automatic generation is presented in section 2. Section 3 gives an example. Conclusion and some future research are presented in section 4.

1 Visual Specification Language

Based on Knuth's attribute grammar[6], we propose a visual function specification language. In order to enhance the comprehension, three basic concepts are adopted in VSL: module, decomposition and package. They are defined as follows:

Module A module is a unit of processing. A module M may have an input attribute set (IN[M]) and an output attribute set (OUT[M]). It is indicated by

$$M(i_1, i_2, \cdots, i_n; o_1, o_2, \cdots, o_m)$$

where M is module name, $i_1, i_2, \cdots, i_n$ are input attributes, so $\{i_1, i_2, \cdots, i_n\}$ = IN[M]; $o_1, o_2, \cdots, o_m$ are output attributes, $\{o_1, o_2, \cdots, o_m\}$ = OUT[M].

Obviously, the handling of a module can be descried by a suitable mapping, for example

$$F(M): D_{i_1} \times D_{i_2} \times \cdots \times D_{i_n} \rightarrow D_{o_1} \times D_{o_2} \times \cdots \times D_{o_m}$$

Received 2001 - 12 - 17.

* The project supported by the project of Ministry of China about key fundamental research on application (6209003001).

** Born in 1963, male, associate professor.

where D_a indicates the value domain of input or output attribute a.

In order to increase visibility, the module may be indicated by a graph as Fig.1. The symbols ↓ and ↑ shown in the figure indicate input and output attributes.

Fig.1 Module

Decomposition When the processing of the module M_0 is simple enough, we can directly write out the mapping $F(M_0)$ between its input and output attributes. On contrary, we must decompose the module M_0 into some smaller modules: M_1, $M_2, \cdots$, M_t. Each smaller module responds to a part of the processing of the module M_0. This is called module decomposition, and it is described by $M_0 \rightarrow M_1$, M_2, $\cdots$, M_t or visual presented in Fig.2. Where M_0 is called maternal module, $M_1, M_2, \cdots, M_t$ are called

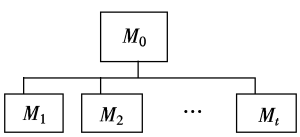

Fig.2 Non-terminal decomposition

sub-modules. When $t = 0$, this decomposition is called terminal decomposition. In this case, the processing of the module M_0 is simple enough to directly write out the mapping relation between its input and output attributes and there is no need for further decomposition. If not, it is called non-terminal decomposition. The terminal decomposition will be presented by $M_0 \rightarrow \cdot$ or by a graph shown in Fig.3.

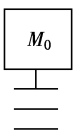

Fig.3 Terminal decomposition

When module M_0 is decomposed to M_1, $M_2, \cdots$, M_t, a set of attribute definitions must be described in order to indicate the data dependent relation of attributes in these modules, i.e., how to give the input attribute value of sub-module and how to get the output attribute value of the maternal module with the processing results of sub-module. Each attribute definition has the general form as follows:

$$a = f(b_1, \cdots, b_m)$$

where f indicates certain mapping relation; $a \in \mathrm{OUT}[M_i]$ or $a \in \mathrm{IN}[M_i]$, $1 \leqslant i \leqslant t$, and $b_1, \cdots, b_m$ are other attributes of the decomposition.

Generally, a module may have different decompositions under different cases, i.e. a module probably has many kinds of decompositions. If d is used to present module decomposition, $d_1, d_2, \cdots, d_k$ are used to describe the k kinds of decompositions of M_0. Many kinds of decompositions of M_0 are described as follows:

$$d_i : M_0 \rightarrow M_1, M_2, \cdots, M_{t_i} \qquad i = 1, \cdots, k$$

Or presented by specification. In order to indicate under what condition a kind of decomposition can be used, we must add a decomposition condition C_d to every decomposition case. C_d is usually a logical expression composed of the input attributes of the maternal module.

In a word, the decomposition of a module has a form as follows:

$$d : M_0 \rightarrow M_1, M_2, \cdots, M_t \text{ with } A_d \text{ where } C_d$$

where C_d is the decomposition condition, A_d is a set of attribute definitions.

VSL package A VSL package is a complete software specification described with VSL. It can be defined as a triad:

$$\mathrm{VSLP} = \langle M, D, M_{\mathrm{init}} \rangle$$

where M is the set of modules; D is the set of decompositions; $M_{\mathrm{init}} \in M$ is called the initial module, indicates that this module is the top module of the package, from which the execution of software module starts.

The definition indicates that a VSLP package is a group of specification of modules and decompositions. If the value V_{in} of the input attribute set $\mathrm{IN}[M_{\mathrm{init}}]$ of the initial module M_{init} is given, then $\langle \mathrm{VSLP}, V_{\mathrm{in}} \rangle$ defines a computing tree, the root of which is the initial module M_{init} and the vertices are other modules shown in the course of the decomposition. The growth of computing tree is controlled by condition of decomposition. The computing tree is ended when all vertices are terminal decomposition. Using the data dependent relation defined by the attribute definitions in the decomposition, we can figure out V_{out}, i.e., the value of the output attribute set $\mathrm{OUT}[M_{\mathrm{init}}]$ of initial module.

2 Principle of Automatic Generation

2.1 To create the data dependent relation

The characteristics of visual specification language are to describe the modules, the decompositions of

modules, the attributes of modules and the generation of definitions of attributes. The data dependent graph will be constituted corresponding to every decomposition through detailed analysis of the data dependent relation among its attributes. Based on it, many kinds of subprogram corresponding the predicted objective language can be generated automatically.

In VSL, there are four kinds of data dependent graphs defined as follows.

2.1.1 Data dependent graphs for module decomposition ($\mathbf{DG}_d$)

DG_d is a kind of directed graph which constitutes the vertexes with the input and output attributes of module d. Every directed side indicates the dependent relation of attribute value in A_d. The lined side in the figure indicates the obvious dependent relation that is corresponding to an attribute definition. The dotted lined side indicates the veiled dependent relation that will be got with computing of the corresponding module. Assume the decomposition d has a form as follows:

$d: M_0 \rightarrow M_1, M_2, \cdots, M_t$ with A_d where C_d

The definition of DG_d is described as follows:

$$DG_d = (DV_d, DE_d)$$

where $DV_d = \{a \mid a \in IN[M_i] \cup OUT[M_i], 0 \leqslant i \leqslant t\}$, $DE_d = \{\langle a, b\rangle \mid b = f(a_1, \cdots, a_m) \in A_d\}$.

2.1.2 Data dependent graphs for computing tree ($\mathbf{DG}_T$)

DG_T is a directed graph that presents attributes dependent relations in the computing tree T. According to the decompositions of module in the course of constituting T, DG_T is constituted by combining some corresponding DG_ds. It is defined as follows:

$$DG_T = (DV_T, DE_T)$$

where $DV_T = \{\omega \mid \omega$ is attributes in computing tree$\}$, $DE_T = \{ < \omega_1, \omega_2 > \mid \omega_1, \omega_2$ correspond the attributes v_1, v_2 of decomposition rule d in computing tree T, and $\langle v_1, v_2\rangle \in DE_d\}$.

2.1.3 Data dependent graphs for module ($\mathbf{DG}_M$)

DG_M is a directed graph that presents the dependent from output attributes to input attributes in module M. It is defined as follows:

$$DG_M = (DV_M, DE_M)$$

where $DV_M = IN[M] \cup OUT[M]$, $DE_M = \{\langle a, b\rangle \mid a \in IN[M], b \in OUT[M], \exists$ path from a to b in $DG_T\}$.

The definition indicates that there is a directed path from a to b in computing tree DG_T rooted with M if $\langle a, b\rangle \in DE_M$.

2.1.4 Extend data dependent graphs for module decomposition ($\mathbf{DG}_d^*$)

DG_d^* is constituted by combining DG_d with DG_{m_i}. It is defined as follows:

$$DG_d^* = (DV_d, DE_d^*)$$

where $DV_d = \{a \mid a \in IN[M_i] \cup OUT[M_i], 0 \leqslant i \leqslant t\}$, $DE_d^* = DE_d \cup \{\langle a, b\rangle \mid \langle a, b\rangle \in DG_{m_i}, 0 \leqslant i \leqslant t\}$.

Obviously, DG_d^* includes not only the data dependent of decomposition rule d but also the data dependent of computing trees that constituted root with the sub-module of d.

Otherwise, in order to enhance definition and convenience, two sets are defined as follows:

1) $PS_M(a)$: the dependent set of output attribute a in module M. It is defined as $PS_M(a) = \{b \mid \langle b, a\rangle \in DG_M\}$;

2) $PS_d(a)$: the dependent set of output attribute a of maternal module M_0 in decomposition d. It is defined as $PS_d(a) = \{b \mid b \in OUT[M_i], 0 \leqslant i \leqslant t, \exists$ path from b to a in $DG_d^*\}$.

2.2 To generate the subprogram package of predicted object language

Based on the data dependent relations, the specification of subprogram given by the user can be automatically transformed into the subprogram package of predicted object language. The user can predict the object language. At present, it can be language C, or language PASCAL.

The subprogram package of predicted object language consists of a main function and a group of declarations of function, in which every declaration of function is corresponding to a module specification of VSLP. The main function is generated according to the specification of the initial module. It includes three basic steps in sequence: reads the values of the input attributes of the initial module, calls the function corresponding to the initial module and gives the computing results[5].

Assume the module M has k kinds of decompo-

sition $d_1, d_2, \cdots, d_k$, then in language C, the corresponding declaration of function of module M is as follows:

```
void m(int x_1, ..., int x_m, int * y_1, ..., int * y_n)
{
    if (C_{d_1}) b_{d_1}; else
    if (C_{d_2}) b_{d_2}; else
        ⋮
    b_{d_k};
}
```

where $x_1, \cdots, x_m$ are the input attributes of module M; $y_1, \cdots, y_n$ are the output attributes of module M; $C_{d_1}, \cdots, C_{d_k}$ are decomposition conditions corresponding to the k kinds of decompositions of module M; and $b_{d_1}, \cdots, b_{d_k}$ are the function bodies corresponding to respective decomposition of module M.

Each function body $b_{d_1}(1 \leqslant i \leqslant k)$ is an executive statement series. Every statement is an assignment statement or a function-calling statement. The algorithm to generate the function body b_{d_i} is shown as follows with corresponding language C function syntax:

1) To independently establish an assignment statement corresponding to every output attribute of maternal module and every input attribute of each sub-module which decompose d_i. Obviously, all the assignment statements are just the A_{d_i}, the set of defined attributes in d_i. In data dependent graph, it indicates all directed lined side.

2) To independently establish a function-calling statement corresponding to each sub-module M in decomposed $d_i: M(u_1, \cdots, u_p; \&v_1, \cdots, \&v_g)$, where $u_1, \cdots, u_p$ are the input attributes names of sub-module M; $v_1, \cdots, v_g$ are output attributes names of sub-module M. In data dependent graph, the function-calling statement responds the directed dotted lined side.

3) To get a topological sorted series through sorting the data dependent graph of d_i according to the attribute dependent relations. The function body b_{d_i} is constituted with the statements established in 1) and 2) sorted according to the series.

3 An Example

As an example, how to get the xth Fibonacci number is presented.

3.1 Specification

This problem only needs a module $f(x; v)$. Module f has two kinds of decomposition: d_1 and d_2, which indicate the different computing methods of the value of v when $x \leqslant 2$ or $x > 2$, respectively. The specification is described as follows:

```
define fib
module f(int x; int v)
decompose f(x; v):: = f(x1; v1), f(x2; v2)
with x1 = x - 1; x2 = x - 2; v = v1 + v2
decompose f(x; v):: = NULL
    with v = 1      where x < 2
end
```

The visual demonstrating is presented in Fig.4. They are interchangeable. If the value of input

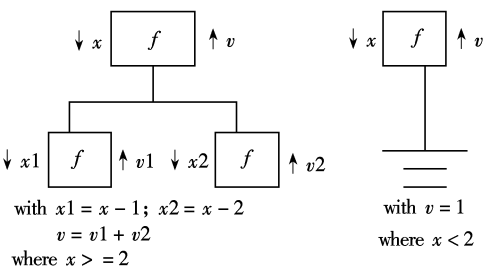

Fig.4 Visual demonstrating

attribute $x = 3$, a computing tree will be constructed as Fig.5. The dotted lines in the figure indicate the data dependent relation defined by the attributes definitions. With these attributes definitions, we can get the value of output attribute v of initial module f to be 3.

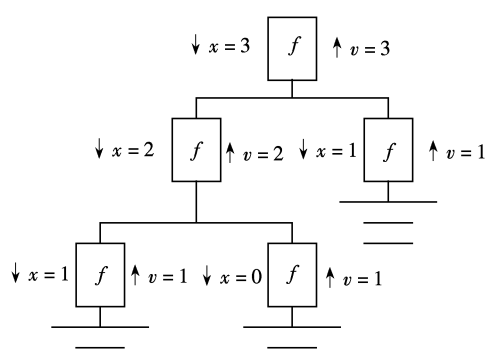

Fig.5 Computing tree of module f for $x = 3$

3.2 Data dependent relation

Fig.6 shows the data dependent relation that is corresponding to the two decompositions, specification-based and Fig.4-based.

3.3 Object subprogram package for language C

The generated function body corresponding to the non-terminal decomposition of module f in Fig.4 is

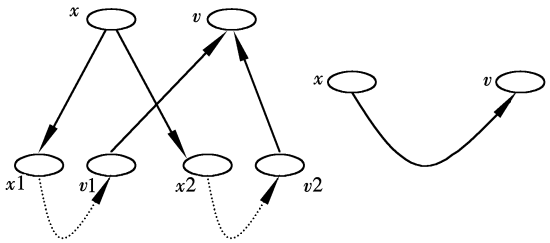

Fig.6 Data dependent graphs

described as follows:

```
x1 = x - 1;
x2 = x - 2;
f(x1, &v1);
f(x2, &v2);
*v = v1 + v2;
```

The subprogram package of language C generated automatically is given as follows:

```
void f(int x,int *v){int v2, x2, v1, x1;
if (x < 2) *v = 1;
    else {x1 = x - 1; x2 = x - 2;
        f(x1, &v1); f(x2, &v2);
    *v = v1 + v2;}
Main()
{int x, v;
 scanf("%d", &x);
 f(x,&v);
 printf("%d \n",v);
}
```

4 Conclusion

On the function of software module, this paper proposes a visual specification language, and gives the principle of generating automatically the subprogram package of the predicted object language based on it. The new ideas will break the limitations of visual describing of function of module. The efficiency of software development will be raised greatly, and the quality of software will be ensured efficiently.

The future researches include: ① The describing of parallel visual specification and the cooperating of multi-designer under network environment; ② The creating of reuse specification library; ③ The transformation from VSL package to CORBA-based or COM-based components IDL[7-9].

References

[1] Feghhi J. *Web Developer's Guide to Java Beans*[M]. America, 1997:2-11.

[2] Eddon G, Eddon H. *Inside COM + base services*[M]. America: Microsoft Press. 1999:5-8, 21-75, 385-405.

[3] Polya G. *How to solve it*[M]. Doubleday, 1957:1-3.

[4] Crimi C, Guercio A, Pacini GZ, et al. Automating Visual Language Generation [J]. *IEEE Trans Software Eng*. 1998 (16):86-94.

[5] Shen Jun, Cheng Zhengchao. A CASE environment for supporting 2-stage software model[J]. *Journal of Southeast University*, 1993, **23**(1):100-104. (in Chinese)

[6] Knuth D E. Semantics of context-free languages[J]. *Mathematical System Theory*, 1968, **2**:2.

[7] Henning M, Vinoski S. *Advanced CORBA programming with C++*[M]. America: Addison-Wesley Longman Inc, 1999:33-85.

[8] Slama D, Garbis J, Russeu P. *Enterprise CORBA*[M]. America: Prentice Hall PTR, 1999:266-271.

[9] Rofail A, Shohoud Y. *Mastering COM and COM +*[M]. America: SYBEX Inc., 1999:1-19.

基于分解的可视化功能规格说明及功能自动生成

沈 军 顾冠群

(东南大学计算机科学与工程系, 南京 210096)

摘 要 针对软件模块,本文提出了一种称为 VSL 的可视化规格说明语言.该语言模拟人的思维过程,将一个大规模复杂的问题分解为一系列较小规模的问题,然后独立解决每一个小规模问题,最后通过各小规模问题的求解综合而解决大规模问题.借助于计算机支持,VSL 能实现软件模块的可视化描述与自动化生成.从而较大地改善软件的质量,提高软件的开发效率.本文论述了 VSL 的简单定义、自动生成原理,并给出一个样例,指出了进一步的研究工作.

关键词 软件规范, 功能分解, 数据依从, 可视化程序设计

中图分类号 TP311.56