第4卷第1期
1993年2月

软 件 学 报
JOURNAL OF SOFTWARE

Vol. 4, No. 1
Feb., 1993

基于面向对象概念的结构编辑器及其生成系统

费翔林 王和珍 汪承藻 廖 雷

(南京大学计算机科学系,南京 210008)

STRUCTURED EDITOR BASED ON THE OBJECT-ORIENTED CONCEPTS AND ITS GENARATOR

Fei Xianglin, Wang Hezhen, Wang Chengzao and Liao Lei

(*Department of Computer Science, Nanjing University, Nanjing* 210008)

Abstract Object — oriented software construction is a kind of promising software methodology, and leads to a whole new way of solving problems. In the research on rapid construction of Structured Editor (SE) that support detailed design and coding in software development, a generator that can generate the SE has been applied as a metatool. The kernal of SE is a syntax—directed editor based on the object— oriented concepts. One of the key issue in the design of SE is how to represent the elements of a target language by means of the class concepts internally. In this paper key concepts, design of the SE and its generator as well as implementation of a prototype are to be discussed.

摘要 面向对象软件开发是一种极有生命力的软件方法学,它导致了全新的求解问题的方法.为了快速构造支撑软件详细设计和编码的结构编辑器SE,我们已研制了一个能够生成SE,作为元工具使用的生成系统.SE的核心是基于面向对象概念的语法制导编辑程序.设计SE的关键问题之一是在系统内部如何用类的概念来表达靶语言的各种语言成分.本文简要介绍了设计思想、SE和生成系统的设计及其实现.

§1. 引 言

软件方法是指导软件开发和研制的标准规程.迄今为止,已有许多不同风格和不同适用范围的软件方法被实际使用,如SD、JSD等.这些传统软件设计方法的固有缺点是数据抽象能力差,导致所开发的软件易修改性、易维护性差.面向对象软件设计(Object — Oriented Software)是基于对象概念进行软件开发的一种极有生命力的软件设计方法.与面向对象软件

本文1991年2月定稿.作者费翔林,副教授,主要研究领域为软件工程,包括面向对象计算和Re—3技术.王和珍,副教授,主要研究领域为Re—3技术和情报检索.汪承藻,高级工程师,主要研究领域为Re—3技术.廖雷,助教,1990年硕士毕业于南京大学,主要研究领域为面向对象计算.

设计相关的面向对象程序设计集数据抽象、抽象数据类型、类型继承为一体，使软件工程公认的模块化、信息隐蔽、抽象、局部化、重用性等原则在面向对象语言机制下得以充分实现，有力地支持了模块化程序设计和软件重用，由此生成的软件系统结构清晰、易于理解、维护方便.总之，采用面向对象方法来开发软件可以提高软件生产率，改进软件质量[1][2][3].

我们采用面向对象程序设计方法研制了结构编辑器(Structured Editor)及其生成系统.SE是一个同时支持软件详细设计和编码的结构编辑程序，对于软件详细设计使用PDL语言表达，通过自顶向下方法对详细设计逐步求精，就可得到用程序设计语言书写的源程序.由于PDL和程序设计语言的编辑都依照语法制导进行，并作必要的语义检查，因而，它能保证所生成的源程序在语法和部分语义上是正确的.针对特定语言的结构编辑器SE是由用户交互式地提供某种程序设计语言的语言成分，再由SE生成系统自动生成.SE的核心是基于面向对象概念的语法制导编辑程序.设计SE的关键问题之一是在系统内部如何使用类的概念来表达靶语言的各种语言成分.本文简要介绍了设计思想、SE和生成系统的设计及其实现.

§2. SE的设计

2.1 设计原则

系统设计中着重考虑下列原则：首先SE应能同时支持软件的详细设计和编码；其次，生成系统的独立性和开发环境的相关性，一方面，生成系统能根据用户给定的不同语言描述自动生成不同的SE、即生成系统独立于具体的程序设计语言.另一方面，由生成系统生成的SE与特定的程序设计语言相关，这样能方便用户使用，提高开发效率；最后SE应具有良好的可扩充性、易于增加新功能和新设施，支持增量式软件开发，具有一致性内部表达和友善的用户接口.

2.2 设计思想

SE的核心是一个基于面向对象概念的语法制导编辑程序，从用户使用的观点来看，一个程序仅由两种成分组成：模板和短语.模板用来定义程序构造类型，即一种模板定义语言的一种语言成分，并作为表达该语言成分的固定框架，它说明了该类程序构造的组成部分及组成方式，模板中含有被称为位标的区域，起着语法提示和限制语法成分的作用，一方面位标指明程序的哪些部分必须细化；另一方面又指明该位标可以细化成哪些语言成分.在SE中，短语定义为简单的程序单位，如表达式和变量说明.位标可以被模板替换，也可能被填入一个短语，通过不断地把位标展开成新的模板或填入短语，最终可得到所需功能的程序.例如，在PASCAL语言中，WHILE语句的模板是

```
WHILE <expression> DO
      <statement>
```

其中，<expression>和<statement>均为位标，前一个位标处允许插入表达式(一种短语)，后一个位标处允许展开各种语句或另外的模板.

在SE内部，每个被编辑的程序唯一地表示成一棵抽象语法树，所有的编辑操作都在语法树上进行.从系统内部表示的观点来看，一棵抽象语法树由许多结点和指针组成.一个结点对应于一个模板，结点可以看作树的分叉，分叉的根节点相当于模板名，用以标识语言成分；分叉的叶节点相当于模板中的位标，描述了语言一种成分的语法结构.指针则将一个结点中的某

些叶节点与另一个结点的根节点连结起来.程序的编辑操作和上下文无关文法的推导极其相似,其中,模板名相当于产生式的名字,模板相当于产生式的右部,位标相当于非终结符,模板的插入相当于推导,被编辑的程序文件相当于推导树,光标位置相当于推导树的结点.

采用面向对象概念设计SE要解决的关键问题之一是如何用类及层次结构来描述程序设计语言的各种成分.一个程序设计语言是在某个特定字母表上定义的,按一组规则构成的符号串的集合.这样一组规则可以抽象的定义文法,从而确定了语言的语法结构,常用扩充的BNF范式来表达规则,其中涉及到AND和OR规则.PASCAL语言AND规则的一个例子是:

<IF statement>::=IF <expression> THEN <statement>
[ELSE <statement>]

它刻划了如果语句IF的这一语言成分的语法结构,即如果语句IF是由一个表达式expression,一个THEN语句,以及0个或1个ELSE语句组成.通过类似的AND规则可以表达一个程序设计语言的所有语句的语法结构.

用OR规则描述PASCAL语言成分的一个例子是:

<statement>::=<IF statement>|<WHILE statement>
|<FOR statement>|……

其含义是产生式右边的任一语言成分均可以取代产生式左部.对于形如B::=A|…的OR规则,A既能替换B,表明A拥有B的性质.这种关系还具有传递性,若A能替换B,B能替换C,则A就能替换C.显然,对每条AND规则都定义一个相应的结点类,那么,OR规则很容易用面向对象概念中的继承性来刻划.具有替代关系的产生式左右部则相应于基类(父类)和导出类(子类)的关系.因此,从系统实现的观点来看,语言的各种成分可用结点类来实现,一个结点类描述了靶语言的一种语言成分和可对此语言成分实施的各种操作.通过定义一系列类及其继承关系来表达程序设计语言BNF范式描述中的AND规则和OR规则,在此基础上,一个被编辑的程序表示为一棵抽象语法树,树上的每个结点属于预定义的某个类,即它是某个类的实例(对象),编辑操作通过对象间的消息传递实现,程序的编辑相当于生成和撤消对象,从而,增长或删去语法树的结点.

采用面向对象程序设计方法开发软件的关键步骤是:首先,分析和识别应用问题中的不同对象及其操作;其次,根据对象的性质和功能归纳入不同类,建立类的层次结构;最后,确定对象之间的消息发送联系.下面具体来讨论SE系统中对象的识别,类的划分,以及如何用类及其层次结构来表达程序设计语言的各种成分.

通过分析一个程序设计语言的成分可以识别出若干不同的对象,仍以PASCAL为例,对语句来说,可以区分IF语句、FOR语句、WHILE语句等不同对象;对定义/说明来说,可以区分过程说明、函数说明、变量说明、类型定义等不同对象等等.面向对象软件设计主要是类的设计,而不是对象的设计,因为,对象是在程序执行时动态产生的.我们采用的规则是:将具有相同性质、包括相同外部性质和内部处理能力的对象划为一类作为最低类,然后,采用自底向上逐步抽象方法,把具有共性的类的公共数据结构和公共操作再并入一个相对于被抽取共性的类的基类中.被抽取共性的类称为导出类(子类),抽取形成的类称为基类(父类).如上进行,不断产生更高的基类,直到某些基类无共性可抽取为止,至此,建立了表达程序设计语言成分的类及其层次结构.图1是根据上述规则生成的表达PASCAL语言成分的类及其层次结构,

其中最低层的每个类相应于BNF范式的一个AND规则，而类的层次结构中不同分叉相当于BNF范式的不同OR规则.

图2是模板，结点类和抽象语法树之间关系的示意.

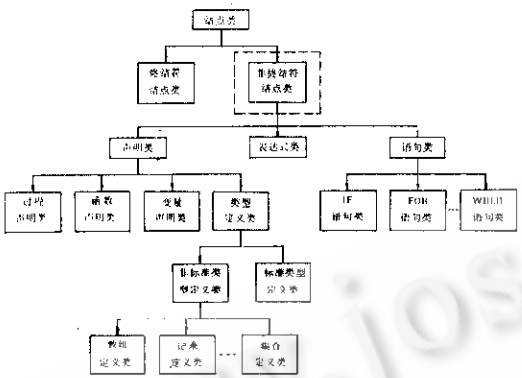

图1　PASCAL－Oriented SE类的层次结构

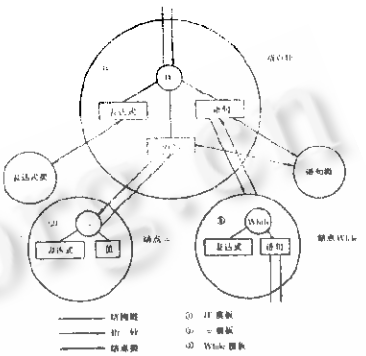

图2　模板，结点和结点类之间的关系

2.3 结构编辑功能

SE提供支持用户操纵程序建立、删除、修改、显示等各种结构编辑操作，以及光标移动、程序文件存取等辅助操作.

(1) 光标移动

由于编辑操作都在语法树上进行，因而，可控制光标按顺序、有层次地遍历抽象语法树结点，支持用户寻找编辑目标，执行编辑操作.提供的光标移动操作有：移动到树根结点；移至同层前(后)一个节点；移至上层(下层)头一个结点；以及用于短语编辑的左(右)移一个字符.

(2) 程序建立

程序是通过自顶向下或层次地在现有的模板或位标处插入新模板和短语来建立和生长的.模板经选择菜单或键入命令推出；模板的插入都在光标当前位置进行，但光标位置可由用户操纵，因而，模板插入的次序不必固定.利用模板展开方式建立和生长程序能避免某些语法错误，减少击键次数.SE采用模板和正文混合方式编辑，对于短语则采用正文方式输入，减少了展开的复杂性，方便用户使用.此外，提供各种插入操作，允许用户在一些程序构造的前后插入合法的模板，加快编辑速度，提高编程效率.

(3) 程序修改

程序修改通过操纵模板或位标完成，这种修改方式保持了程序语法语义的正确性.光标指向的位置总是一个模板或位标，而不是正文行.系统提供一组灵活、方便的修改操作，包括：

. 删除　删去标记子树，使当前程序结构重新成为一未展开的程序结构.

. 移动　将标记的子树移动到语法树上当前光标处.

. 复制　将标记的子树复制到语法树上当前光标处.

. 标记/去标记　对光标所指向的子树设置或去除标记，在屏幕上则以反转色显示相应正文.

(4) 程序显示

每当一个编辑操作处理完毕后，通过语法树反扫描程序，根据各结点所属类提供的反扫描信息，可在屏幕上显示三种不同格式的正文程序.

.美化格式正文程序　显示完整的正文，呈锯齿状排列，操纵控制键可浏览程序正文的不同部分.

.隐蔽格式正文程序　可将结构化语句的细节隐蔽起来，让更多程序信息显示在屏幕上，操纵控制键可在美化格式和隐蔽格式正文程序间作转换.

.嵌套格式正文程序　类似于隐蔽格式正文程序，但把具有指定嵌套深度的内嵌结构化语句全部隐蔽起来，达到更有效地利用屏幕区域，呈现程序结构的目的.

此外，可通过特殊窗口观察并浏览程序的抽象语法树.

(5) 程序文件存取

提供两种格式文件保存被编辑的程序，一是抽象语法树文件，二是正文文件. 前者便于恢复成系统内部表示形式；后者便于和其他软件工具(如编译系统)交互.

§3. SE 的实现

系统是在 IBM PC/286，MS－DOS 操作系统下，用面向对象程序设计语言 C＋＋实现的[5]. SE 以基于面向对象概念的窗口系统 OOWS(A Windows Software Based on Object－Oriented programming method) 为人机界面，通过多窗口支撑、菜单选择、鼠标驱动、信息提示形成十分友善的用户接口. 目前，由生成系统生成的 PASCAL－SE 和 C－SE 已能正常运行. 每当调用编译系统之前，总将抽象语法树文件和符号表调出并释放相应空间，以保证内存开销较小；每当编辑命令执行后，通过反扫描语法树获得的正文程序直接送显示缓存并显示在屏幕上，响应速度较快. 针对存储空间和响应速度采取的措施进一步增强了 SE 的实用性.

3.1 系统结构

SE 及其生成系统的结构如图 3 所示. SE 由六部分组成：

.用户接口　提供多窗口、菜单选择、鼠标驱动以及完成与用户的交互功能.

.命令解释程序　将命令翻译成消息，并把消息发送给抽象语法树上当前光标所在结点对应的对象.

.结点类　支持建立抽象语法树，树上每个结点必是某一结点类的实例，对象接收到一条消息后，激活并执行相应结点类中有关的动作子程序，以完成某一编辑操作.

.反扫描程序　每当处理完一条编辑命令，反扫描程序遍历抽象语法树结点，并根据结点类中的反扫描信息得到程序的正文表示，并显示在屏幕上.

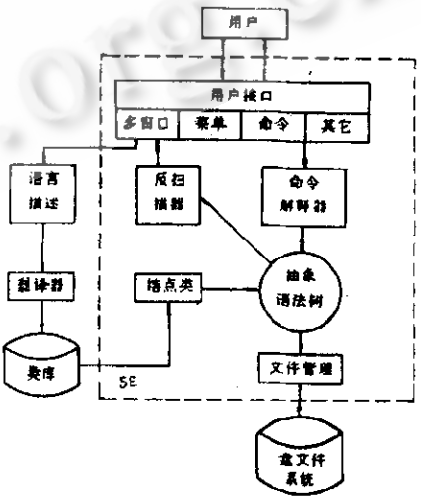

图 3　系统组织和生成的 SE

. 抽象语法树　被编辑程序的内部表示.

. 文件管理　用于和操作系统的文件管理进行交互.

此外，SE 中的结点类是通过自行设计的翻译器，转换用户给出的靶语言描述而获得的. 结点类被存放于结点类库中，用以支持建立和生成抽象语法树.

3.2 结点类组成和实现

一个结点类由以下信息组成：

(1) 结点类名　用以标识某个结点类.

(2) 基类名　导出类可以从其基类处继承实例变量和操作.

(3) 实例变量　用来刻划类所拥有的性质. 在 SE 中，结点类的实例变量包括：语法结构链、反扫描格式信息和语义属性指针.

语法结构链决定了语言某一成分的语法结构，即刻划了模板的结构. 每个非终结符结点类具有固定个数的语法结构链，而终结符结点类却没有. 语法结构链由结点类名和模式组成. 模式共有三种情形：固定的、选择的和重复的. 其含义分别是此语言成分必须出现一次、最多出现一次和可出现任意多次. 由于语言的每一成分都有一个结点类进行描述，所以，上述三种情形下，与该语言成分相应的结点类就必须出现一次、最多出现一次和可出现任意多次. 例如：PASCAL 语言的 IF 语句中，表达式部分必须出现一次，ELSE 部分最多出现一次；而复合句中的语句序列最多可出现任意多次.

反扫描信息用于描述语言成分的正文显示格式，语义属性指针指向符号表.

(4) 动作子程序　实现各种编辑功能和有关输出操作. 一个动作子程序对应着一个编辑或其他操作，当结点接受到一条消息时，结点类中的动作子程序被按名调用，从而完成某一功能.

下面讨论如何用 C++来实现结点类，用 C++语言描述结点类具有以下形式.

```
struct structlink {
                    char *name;
                    int kind;/* 固定的,可选择的,可重复的 */
                    struct structlink *next;
                    };/* 语法结构链 */
class node {
                    public:
                    char *name;
                    node *father, *son, *elder, *younger; /* 指针 */
                    int Newline,Ident,Row,Colum; /* 反扫描信息 */
                    struct structlink *head; /* 语法结构链头 */
                    node(); /* 类建立器 */
                    virtual void Insert(); /* 动作子程序 */
                    virtual void Delete(); /* 动作子程序 */
                    virtual void Move(); /* 动作子程序 */
                            :
                    virtual void Copy();/* 动作子程序 */
                    }
```

接着对抽象级较低的类进行描述，在 C++中，class A :public B 表示 A 是 B 的子类，IF

语句类是 statement 类的子类，因此，它具有下列形式：

```
class ifstatement : public statement {
                    public :
                    ifstatement();/ * 建立器 * /
                    virtual void Insert();
                        ⋮
                    virtual void Copy();
                    };
```

为了与 Terminal 类相对应，求得概念上的一致性，我们引入了全盘继承 node 类，未做任何扩充的 Nonterminal 类(图 1 中用虚线框出)，但在实现上，node 类就是 Statement 类的父类. 所以 statement 类被描述为：

```
class statement : public node {
                  public:
                  statement();/ * 建立器 * /
                  virtual void Insert();
                      ⋮
                  virtual void Copy();
                  };
```

从上述类定义中可以看出，类中有一个建立器，它被用来对类中的私有数据进行初始化. node 类，statement 类对应的类建立器分别为：

```
node :: node / * node class 建立器 * /
            { name="";
            father=son=elder=younger=NULL;
            newline=Ident=row=colum=0;
            head=NULL;
            };
statement :: statements / * statement 建立器 * /
             { name="STATEMENT";}
```

注意，node 类的建立器对 head，newline，Ident 等进行了初始化，statement 类中的私有量初始值除 name 是自身给出的外，其余的均从 node 类处继承，从而实现了代码重用.

3.3 语法、语义检查

为了保证被编辑程序在语法和语义上的正确性，必须提供相应的语法检查和语义检查功能. 语法正确性从两方面来考虑，一是采用对应于 BNF 语法描述的模板能自动实现上下文无关语法的正确性；二是每当在位标处填充一条短语后，立即做短语的上下文无关语法检查，(如表达式错)，接着从根结点处遍历语法树做全程语法检查(如变量未说明先引用).

标识符及有关的信息放在符号表中，符号表包括：标识符表和类型表等. 编译涉及某些标识符时，符号表中的信息需作相应改变. 语义与程序在动态运行时刻所处的状态相关，因此，利用编辑器来静态地进行语义分析是不现实的. 尽管如此，SE 能尽量把静态时发现的语义错指出来. 在系统定义的符号表基础上，检测诸如：表达式中下标变量是否越界、变量使用前是否有初值、变量只定义而不使用等等. 可以看出，在 SE 中，所有语义检查都附属于全程语法检查.

3.4 生成系统

每个特定的SE是根据用户提供的靶语言描述信息由SE生成系统自动生成.为了描述靶语言,用户必须提供:靶语言名、靶语言保留字、模板格式和符号表描述.

模板格式采用交互式输入,下面是PASCAL的IF语句模板格式输入的例子,下划线部分由用户回答:

```
Classname:IFstatement
Superclass :statement
Format :IF<expression>
        THEN <statement>
        [ELSE <statement>]
```

其中,[ ]部分表示任选项.用户并不需要知道有关类的知识,翻译器分析用户的回答,从对靶语言BNF的格式输入获得格式和参数,获得类的继承关系,并填入C++类描述的适当位置,自动将各种模板格式转换成不同结点类.上述IF语句模板相应结点类的C++程序为:

```
IFstatement :: IFstatement() /* IFstatement 建立器 */
        { struct structlink *cur, *p;
          p=malloc(sizeof(structlink)); /* 分配内存 */
          p->name="if"; /* 语法结构链中元素的结点类名 */
          p->kind=0; /* 结构链中元素的种类,0表示是固定的 */
          p->newline=0; p->ident=0; /* 反扫描信息初始化 */
          p->next=NULL; cur=p; head=p; /* 链头初始化 */
          p=malloc(sizeof(structlink));
          p->name="expression";
          p->kind=0; p->newline=0; p->ident=0;
          cur->next=p; p->next=NULL;cur=p;
          p=malloc(sizeof(structlink));
          p->name="then";
          p->kind=0; p->newline=1; p->ident=3; /* 编排缩进格式 */
          cur->next=p; p->next=NULL; cur=p;
          p=malloc(sizeof(structlink));
          p->name="statement";
          p->kind=0; p->newline=0; p->ident=0;
          cur->next=p; p->next=NULL; cur=p;
          p=malloc(sizeof(structlink));
          p->name="else";
          p->kind=1; p->newline=1; p->ident=0;
          cur->next=NULL; cur=p;
          p=malloc(sizeof(structlink));
          p->name="statement";
          p->kind=1; p->newline=0; p->ident=0;
          cur->next=p; p->next=NULL; cur=p;
          };
```

通过上述方法,不仅刻划了IF语句的语法结构,而且还描述了它在屏幕上的反扫描显示格式.由于不同的靶语言,对同一语法成分有不同的缩进格式,在SE中,这些控制信息放在与靶语言有关的格式表中.每当生成语句结点类时选取语言相应的格式信息赋给对应类变量.如

果用户要求修改缩进格式，只需修改格式表.

最后，对结点类、语法分析程序、语义分析程序、动作子程序、公共处理子程序以及有关支撑软件进行编译和连接便可生成针对特定语言的 SE.

§4. 结束语

语法制导编辑程序的典型代表有：CPS[6]，GANDALF[7]，MENTOR[8]，PECAN[9]，A System for Generating LOE[10]等. 它们有以下共同特性：一个程序被表示成一棵抽象语法树；抽象语法树由结点和指针组成，结点表达靶语言成分，指针将结点连接起来；基于模板的编辑操作；提供一组结构编辑操作.

SE 除借鉴上述系统的有关思想外，主要区别在于：采用面向对象程序设计方法开发系统，用类及其层次结构表达靶语言的各种成分，提出了一种简便易行的模板格式描述方法，通过生成系统转换模板格式成结点类并自动生成特定语言的 SE.

总结 SE，得到以下结论：(1) 程序被表示成一棵抽象语法树，树由结点和指针组成，每个结点由标识语言成分的根节点及描述语言成分、语法结构的叶节点组成. 指针将一个结点的某些叶节点与另一个结点的根节点连接起来. (2) 模板定义程序构造类型，由结点类实现，结点类中的语法结构链决定了模板的语法结构. (3) 抽象语法树上的每个结点都是一个对象，是一个结点类的实例，编辑操作由结点类的动作子程序来实现，对象之间发送消息激活结点类中的动作子程序，执行结果会引起抽象语法树结点的增减、相当于对象的建立和撤消. (4) 增加公共设施和修改共性功能只需更改类层次结构中的某些基类，所以，功能易于扩充，系统便于修改. 进一步要研究的问题包括：一般正文程序到 SE 抽象语法树转换工具；语义分析器生成系统. 致谢：衷心感谢王晓虎、陈皓华同志在本系统设计和实现中所做的工作.

参考文献

[1]A. Goldbery and D. Robson, Smalltalk—80 : The Language and its Implementation, Addison Wesley, 1983.

[2]Stetik, M and Borow, D. G. , Object—Oriented Programming: Themes and Variations, AI Magazing, Vol. 6, 4, 1986.

[3]G. Booth, Object—Oriented Development, IEEE Trans. , On S. E Feb. , 1986.

[4]Gerald E. Peterson, Object—Oriented Computing, Volume1: Concept, Published by Computer Sodiety Press of the IEEE, Washington D. C. , 1987.

[5]B. Stroustrup, The C++ Programming Language, Addison—Wesley, 1986.

[6]Tim Teitelbaum and Thomas Reps, "The Cornell Program Systhesizer: A Syntax—Directed Programming Environment ", Commun. ACM, Vol. 24, No. 9, Sept. 1981.

[7]A. N. Habermann and D. Notkin, "Gandalf: Software Development Environments," IEEE Trans. Software Engineering, Vol. Se—12, No. 12, Dec. 1986.

[8]V. DonzeauGouge, G. Huet, GKahn, and B. Lang, "Programming Environment Based on Structured Editors: The MENTOR Experience", INRIA Tech. 26, May, 1980.

[9]S. Preiss, Graphical Program Development with PECAN Program Development System, Proc. ACM SIGSOFT/SIGPLAN Software Engineering Symp. , Practical Software Development Environments, Apr. 1984.

[10]Takao Tenma et al. , A System For Generating Language—Oriented Editors, IEEE Trans. on Software Engineering, Vol. 14, No. 8, August, 1988.