基于GenVoca模型的软件构架研究*

喻 勇 徐 锦 赵文耘 许幼鸣 钱乐秋

(复旦大学计算机科学系 上海 200433)

E-mail: wyzhao@fudan.edu.cn

摘要 GenVoca是一个领域独立模型,用于设计和构造基于大规模软件重用的层次软件系统.该模型为在多个领域中实现软件构架技术提供了一套有效的方法.文章讨论了GenVoca模型的主要特征,提出了基于该模型构造的一个可复用的图形编辑器(graphical editor,简称GE)软件构架,并对该构架中的主要域、构件以及构件间的相互关系进行了分析,给出了将这些构件复合成一些不同的图形编辑器的示例.

关键词 GenVoca模型,构件,域,软件构架,图形编辑器.

中图法分类号 TP311

随着软件系统规模和复杂度的日益升级,人们逐渐意识到,系统总体结构的设计和规格说明比对算法和数据结构的选择重要得多[1].在这种背景下,人们认识到了软件构架(software architecture)的重要性,并认为对软件构架的深入研究将会成为提高软件重用度的新的最有希望的途径,甚至有人认为,90年代是软件构架的年代[1,2].

软件构架是抽象的系统规格说明,是对软件系统的组成、系统结构及系统如何工作较为宏观的描述.简单地讲,软件构架描述了一个系统中的构件(component)及构件间的互连关系.基于构架的软件开发将开发的重心从代码行转移到了更大粒度的构架元素(例如构件等)以及这些元素间的总体关联结构上,其开发的重点是构造可重用的软件构架.构架的重用是基于系统整体结构的更高层次的重用,它强调的是分析和设计的可重用性,而不仅仅局限于代码重用,其最终目的是能高效地开发相关的多个应用[3].

为软件构架建立形式化的模型是实现基于构架的软件开发的重要内容,因而构架的建模近来已成为一个重要的研究主题.其中,如何为构架提供形式化模型,以及形式化地描述模型中构件及构件间的相互关系等问题是研究的热点[1~3].针对这些问题,一种途径是利用构架描述语言(architecture description languages)来实现构架的建模(文献[4]对当前的主要构架描述语言进行了分类和比较).由于构架描述语言或者与特定领域相关,或者比较复杂,实现代价较大.在我们的研究中,需要一种便于实现的、对构架较高层次的描述手段,以求从较高的层次上实现构架的建模.因此,我们使用了一种领域无关的元模型来实现构架建模,这种元模型就是GenVoca模型.

GenVoca是一个领域独立模型,用于设计和构造基于大规模软件重用的层次软件系统,适合描述由参数化的可重用构件组织起来的层次软件系统.我们的工作是将GenVoca模型方法引入到构架的建模和描述中,构造所需的软件构架,并在相关方面进行一些探索.

在国家"九五"攻关项目"需求分析工具和设计工具构件的研究开发"中,我们使用GenVoca模型方法实现了构架的建模,构造了基于GenVoca模型的一个图形编辑器(graphical editor,简称GE)软件构架.

---

* 本文研究得到国家"九五"攻关项目基金资助.作者**喻勇**,1973年生,硕士生,主要研究领域为软件工程.**徐锦**,女,1974年生,硕士生,主要研究领域为软件工程.**赵文耘**,1964年生,教授,主要研究领域为软件工程,电子商务.**许幼鸣**,1972年生,硕士生,主要研究领域为软件工程.**钱乐秋**,1942年生,教授,博士生导师,主要研究领域为软件工程,软件生产自动化.

本文通讯联系人:赵文耘,上海 200433,复旦大学计算机科学系软件工程实验室

本文1998-05-12收到原稿,1998-08-24收到修改稿

1 GenVoca 模型

GenVoca 模型由 Don Batory 等人提出并加以完善[5~7],它是从两个相互独立的软件系统(涉及数据库领域的 Genesis 和关于网络通信系统的 Avoca)的设计方法中提炼出来的产物. GenVoca 模型实际上是一个元模型,可以用来定义不同领域的开放构架的模型[5]. 该模型的主要概念包括构件、域(realm)、对称构件(symmetric component)和类型等式(type equation)等.

1.1 构件和域

在 GenVoca 模型中,层次软件系统由一系列逐步抽象的虚拟机来定义. 其中,构件(即层)是一个虚拟机的实现(与一般意义上的层次概念不同,GenVoca 模型中的层与层之间是可以复合的),是构造大规模的软件系统的基本元素;实现同一个虚拟机的构件集合称为域(realm),域的成员可以通过枚举的方式表示,例如,有两个域 R 和 S:

$$R=\{a,b,c\},$$

$$S=\{d[R],e[R],f[R]\}.$$

上面的表达式表明,域 R 有 3 个成员,即构件 a,b 和 c,它们是 R 接口(即虚拟机)的不同实现,因而我们称构件 a,b 和 c 的类型为 R;域 S 也有 3 个成员,每个成员代表 S 接口的一种不同的且可选的实现.

构件可以被参数化. 例如,前述域 S 中的所有构件都有唯一一个属于域 R 的参数,域 S 中的每一个构件(例如 d)都输入 R 虚拟机接口而输出 S 虚拟机接口,从而构件 d 实现了从 S 虚拟机到 R 虚拟机的一个转换. 其中,构件 d 所作的转换并不依赖于 R 的接口是如何实现的,因为构件 d 封装了接口 R 与 S 间的复杂映射.

1.2 构件复合与对称构件

构件通过复合组成系统,通过构件间输入、输出接口的关联建立起构件的复合关系. 当一个构件输出的接口与其输入接口相同时,该构件是对称的(即域 W 中的对称构件至少有一个 W 类型的参数). 对称构件可以任意方式复合. 在下面的域中,构件 n 和 m 是对称的,而 p 却不是对称构件.

$$W=\{n[W],m[W],p,\ldots\}.$$

由于 n 和 m 是对称的,复合 $n[m[p]]$,$m[n[p]]$,$n[n[p]]$和 $m[m[p]]$都是可能的,其中后两种复合表明构件可以与自身复合. 一般来说,构件的复合顺序会对最终系统的语义和性能产生很大的影响.

1.3 类型等式和软件系统

GenVoca 把软件系统模型化为类型等式. 例如,下面两个关于前述域 R 和 S 的系统:

$$System\_1=d[b];$$

$$System\_2=f[a].$$

System_1 是构件 d 和 b 的复合,System_2 是构件 f 和 a 的复合;而这两个系统都是类型为 S 的等式,这就意味着两个系统都实现了同一个虚拟机. 由此可知,System_1 和 System_2 是接口 S 的两个可互换的实现.

构件重用是软件重用的一种重要形式,在 GenVoca 模型中,当在两个或更多的表达式中使用了同一个构件时,构件重用得以实现. 这就是说,如果 $a[b[c]]$和 $d[b[q]]$是两个表达式(软件系统),则实现了构件 b 的重用.

2 GE(图形编辑器)软件构架

2.1 背 景

在过去的研究中,我们已经开发了多个 CASE 工具,例如,结构化分析工具(SAT)和结构化设计工具(SDT)等. 这些工具的核心部分是一个图形编辑器. 我们对这些工具中的图形编辑器进行分析后,发现它们在体系结构上存在极大的相似性. 然而,由于过去几乎是相互独立地开发这些工具,各个工具的相同或相似部分在设计和实现上并没有遵循统一的规范,这就导致了这些工具中的图形编辑器很难为新的应用所重用. 因而在我们开发一个新的 CASE 工具的过程中,大量的工作依然要集中在其图形编辑器的开发上. 鉴于这种状况,我们希望通过使用软件构架技术来实现这些应用间较大规模的软件重用,以便提高软件的质量、缩短开发周期. 这是我们研制图

形编辑器构架的出发点.

一般来说,CASE工具中的图形编辑器的基本功能应包括:

〈1〉图元的创建;

〈2〉图元的选择:单个选择、逐次添加选择、多个选择和全部选择;

〈3〉图元的移动及改变大小;

〈4〉图元的拷贝/剪贴/删除;

〈5〉取消与重复以前的操作;

〈6〉创建子图;

〈7〉浏览图的层次结构;

〈8〉图形打印.

我们的目标是,基于图形编辑器的上述基本功能开发一组通用的图形编辑构件,构造一个可重用的图形编辑器构架.基于该图形编辑器构架,在开发一个新的CASE工具时,我们便可以通过重用整个图形编辑器构架来实现该工具的图形编辑器部分,并且可以在开发时重用构架中的大量构件,只需把重心集中到新应用中特殊的构件的开发上,从而大大减轻了新系统中图形编辑器部分分析设计和编码的负担,因此可以更加快捷地开发出新的应用.

2.2 GE层次化构架

基于上述目标,我们按照GenVoca模型方法,制作了所需的各个构件,构造了一个GE(图形编辑器)层次构架,如图1所示.

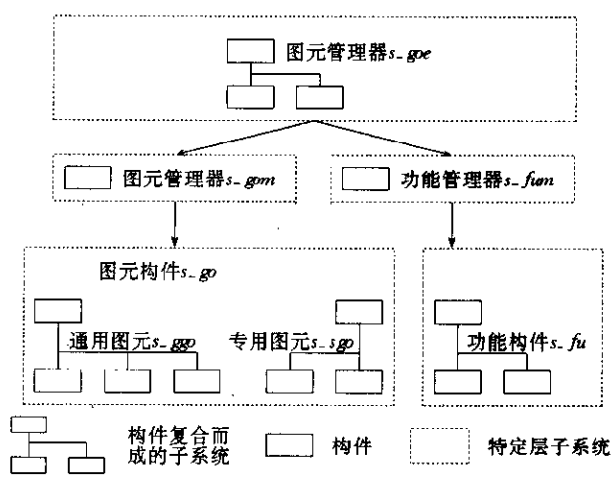

图1　GE的层次化构架

该构架在总体上由5个子系统(也可看作构件)组成:图形对象编辑器子系统s_goe,图元管理器子系统s_gom,功能管理器子系统s_fum,图元构件子系统s_go和功能构件子系统s_fu.相应的域为R_GOE,R_GOM,R_FUM,R_GO和R_FU.其中,图形对象编辑器s_goe集成了常见的图形编辑功能;图元构件子系统s_go和功能构件子系统s_fu分别实现了各种基本的图元构件和图形编辑所需的各种基本功能构件(本文将图形编辑器中的基本图形对象也称作图元);图元管理器子系统s_gom负责图元实例的创建及管理,实现图元对象与图形对象编辑器间的连接;而功能管理器子系统s_fum则负责对功能构件的管理,实现功能构件与图形对象编辑器的通信.因此,GE层次化构架的域可以表示为:

$$R\_GOE=\{s\_goe[x:R\_GOM,y:R\_FUM]\},$$

$$R\_GOM=\{s\_gom[x:R\_GO]\},$$

$$R\_FUM=\{s\_fum[x:R\_FU]\},$$

$$R\_GO=\{s\_go\},$$

$$R\_FU=\{s\_fu\}.$$

从而一个GE层次系统(h_ge)可用类型等式表示为

$$h\_ge=s\_goe\,[s\_gom\,[s\_go],\ s\_fum\,[s\_fu]].$$

在上述5个子系统中,s_goe,s_gom和s_fum都是构架中相对固定的部分,新的图形编辑器应用一般可以完全重用这些成分;而图元对象子系统s_go和功能构件子系统s_fu中的构件则是可装卸的,这就意味着在新的图形编辑器应用之中,可以根据具体需要选取和添加(移去)相应构件.可见,构架配置的这种灵活性不仅有利

于新的应用的快速开发,还为构架适用于更多的应用提供了可能.下面我们详细讨论GE构架中各子系统的构成情况.

2.3 GE构架中的子系统

2.3.1 图元构件子系统

图元构件子系统 *s_go* 由数十个基本图元构件构成.图元构件主要完成两个功能:一个是维护自己特有的属性,另一个是画出自己的形状.为了使构架的配置尽可能灵活,我们没有将这些构件放在一个虚拟机中实现,而是将图元构件划分成通用图元构件和专用图元构件两类,并分别为这两类构件提供了标准的虚拟机接口,形成两个图元构件域,即通用图元构件域 *R_GGO* 和专用图元构件域 *R_SGO*.专用图元构件是指具体图形编辑器应用中所特有的图元构件,而通用图元构件是各图形编辑器应用中通用的基本图元构件,这两类构件对应的子系统是 *s_ggo* 和 *s_sgo*.此外,专用图元构件往往可以通过重用相应的通用图元构件来实现.

通用图元构件域包括实体类图元构件子域 *ENTITY*、关系类图元构件子域 *RELATION* 和图元组子域 *GROUP*.其中,子域 *ENTITY* 中的构件表达实体类的图元,例如圆(circle)、菱形(diamond)和椭圆(elipse)等;子域 *RELATION* 中的构件表达关系类的图元,例如一对一(OneOne)、一对多(OneMany)等;子域 *GROUP* 中的构件则用来表达成组的图元.总的来看,通用图元构件子系统所对应的域 *R_GGO* 是子域 *ENTITY*, *RELATION* 和 *GROUP* 的并集,即

$$R\_GGO = ENTITY \cup RELATION \cup GROUP,$$

如图2所示.

```
ENTITY={                                  RELATION={
  Circle,      /* 圆 */                      OneOne,      /* 一对一 */
  Diamond,     /* 菱形 */                    OneMany,     /* 一对多 */
  Elipse,      /* 椭圆 */                    OneOneHalf,  /* 单折一对一 */
  Rect,        /* 矩形 */                    OneOneMuti,  /* 多折一对一 */
  RoundRect,   /* 圆角矩形 */                OneOneDir,   /* 有向一对一 */
  Triangle,    /* 三角形 */                   ...
  Text,        /* 文字 */                    ReGroup[x:RELATION]   /* 关系图元组 */
  ...                                       ...}
  EnGroup[x:ENTITY]   /* 实体图元组 */    GROUP={
  ...}                                      GeGroup[x:R_GGO]   /* 图元组 */

R_GO=ENTITY∪RELATION∪GROUP
```

图2 通用图元构件域

在上述域中,构件 *EnGroup* [*x*:*ENTITY*], *ReGroup* [*x*:*RELATION*]和构件 *GeGroup* [*x*:*R_GGO*]的输入/输出接口是相同的,因此,它们都是对称构件.利用这些构件,可以方便地实现构件间的组合,形成图元组,满足图形编辑器应用中对成组的图元实例进行编辑的需要.

专用图元构件域由具体图形编辑器应用中的各专用构件子域组成,这部分的构件要根据具体的应用来配置.制作这些构件是开发新CASE工具时图形编辑器部分的主要工作.例如,对于需求分析工具和设计工具中的图形编辑器,可以制作相应专用构件,分别放到专用构件子域 *DFD_GO* 和 *MSD_GO* 中,可见专用图元构件子系统的所对应的子域 *R_SGO* 是域 *DFD_GO*、域 *MSD_GO* 和其他专用构件子域的并集,即 $R\_SGO = DFD\_GO \cup MSD\_GO \cup \ldots$

2.3.2 功能构件子系统

图形编辑器不仅可以交互地编辑单个或部分选中的(即成组的)图元实例的位置、大小、形状等属性,有时候需要对整个页面或整张图进行某种功能性的操作(例如,需对整张图进行重名检查,一旦查出有两个图元实例重名,则需更改相应名称).构架必须能为这些需求提供方便实用的支持,这种支持以功能构件的方式实现,我们把这些功能构件放到子域 *FUNCTION* 中.另一方面,对具体图形编辑器应用而言,有一些功能是他们特有的,例如词典管理、关键字管理等,为了支持这些特殊功能,我们也制作了相应的专用构件,设计了相应专用子域.例如,下面两个子域 *DICTIONARY* 和 *KEY* 分别用于词典管理、关键字管理.因此,上述域的并集组成了功能构件

域R_FU,即 $R\_FU=FUNCTION\cup DICTIONARY\cup KEY\cup\ldots$,如图 3 所示.

```
FUNCTION={                          DICTIONARY=
  Dname,      /* 重名检查 */        { DicEntry,   /* 词条 */
  Unify,      /* 一致性检查 */        Dic,        /* 词典,存储图元描述 */}
  Test,       /* 测试 */           KEY={
  Search,     /* 查找 */             Key,        /* 关键字 */
  ...}                               KeySet      /* 关键字集 */
...                                  ...}
```

$R\_FU=FUNCTION\cup DICTIONARY\cup KEY\cup\ldots$

图3　功能构件域

2.3.3　其他子系统

在图元管理器子系统 *s_gom* 和功能管理器子系统 *s_fum* 中,我们分别实现了一个构件,即图元管理器 *gom* 和功能管理器 *fum*. 其中,*gom* 负责交互式地可视化创建图元实例,同时,通过该管理器,不同种类(包括未来图形编辑器应用新建)的图元构件可以方便地加入/移出图形编辑器;*fum* 则负责完成对功能构件的管理,并且该管理器应与图元管理器相似,允许以后新的功能性构件的插入与移去.

图形对象编辑器子系统 *s_goe* 集成了常见的图形编辑功能,它支持对图的分页存储管理和按页编辑方式,因而它操作的每张图应由一个或多个页面组成,每个页面管理具有层次关系的图元实例. 为此,我们用 4 个构件来实现图形对象编辑器子系统 *s_goe*,它们是:管理图形数据的文档管理构件 *doc*、管理页面上图元实例的页面管理构件 *pag*、对各个页面进行协调管理的图管理构件 *gra* 以及协调对图形及图形数据的管理的编辑器构件 *edt*,它们之间的关系如图 4 所示. 因此,图形对象编辑器子系统 *s_goe* 可以用类型等式表示为

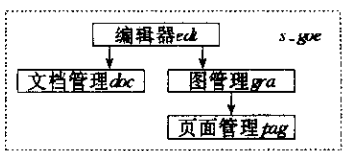

图4　图形对象编辑器子系统

$$s\_goe=edt[doc,gra[pag]].$$

图形对象编辑器子系统是图形编辑器中最核心的部分,其主要功能由构件 *doc*,*gra* 和 *pag* 完成. 其中,文档管理构件 *doc* 提供文件存储功能,即将图存入文件,或从文件中读出图后,恢复原先的结构;图管理构件 *gra* 用于显示图中各页面间的组织结构关系,并帮助用户在不同页面间导航和快速切换;页面编辑构件 *pag* 用于编辑页面上的图元实例,它提供以下功能:图元实例的拖放式创建;对(多个)图元实例的标准编辑操作:选择/全选、剪切/复制/粘贴、可视化移动、形状更改/属性更改、页面显示缩放以及页面打印/打印预览等.

2.4　构架的实例化

从 GE 构架构造新的图形编辑器的过程是:〈1〉制作所需图元构件与功能构件. 在该过程中,可以从构件库中选择合适的构件进行复用;或者直接复用,或者通过一定的方法定制适合于特定应用的新构件. 定制方法 1 是:新构件继承所选构件,并对所需更改的部分进行重新定义;定制方法 2 是复制所选择构件的代码,并对其代码进行相应修改,形成新的构件. 方法 1 主要适合于新构件外部接口不变,但内部处理方式有所改动的情况,而方法 2 则适用于新构件与所选构件性质相似,但内部实现相差较大的情况. 〈2〉根据新系统的实际需求,装卸构架中的其他构件,配置生成新的图形编辑器,进而构造新的应用系统.

按照上述过程,我们实现了两个 CASE 工具:用于绘制模块结构图的 MSD(module structure diagram) Editor 和用于绘制数据流图的 DFD(dataflow diagram) Editor. MSD Editor 主要有两种特定于应用的图元构件:模块(Module)和调用(Call),它们可以直接从构件库中选取相符的构件实现复用:对 Module 构件,我们选择 ENTITY 域中的"矩形"构件(Rect);由于调用需要用带箭头的直线表示,而构件库中的"有向一对一"构件(OneOneDir)与之相符,因而可以直接复用得到 Call 构件. DFD Editor 的构造方法为:〈1〉应用 GE 构架,采用方法 1 构造图元构件,利用库中图元 Elipse 来构造应用所需的"加工"图元(Process),利用图元 Rect 构造"源宿"图元(SourceSink)等;〈2〉通过方法 2 来构造两个特殊构件:DFD 字典管理功能构件(DFDDic)和 DFD 一致性检查功能构件(DFDUnify). 〈3〉将新的构件装载到构架中,同时卸去那些与系统不相关的构件,完成应用配置.

3 结 语

软件构架技术是实现更大规模的软件重用的新的、最有希望的途径，它使得软件重用进入了一个新的时代. GenVoca 模型为在多个领域中实现软件构架技术提供了一套有效的方法，基于该模型，构架的建模、构架的描述和构件间的复合等都较易实现，该模型可以广泛应用到各种构件之间可以复合的层次软件系统中. 本文利用 GenVoca 模型，成功地实现了一个图形编辑器的建模，构造了一个图形编辑器构架，展示了一个复杂的系统是怎样被一些由基本的构件构成的域所组成的元模型描述的. 在 GE 构架的实现过程中，我们发现：实现一个良好定义的、可扩充的、易于复用的构架，其工作量是实现单个应用系统的两倍以上(一个有趣的现象是 GE 构架的总代码量与单个应用系统相近，但接口部分的代码量却相差较大，前者是后者的两倍左右)，而在实例化已有构架，生成新的应用系统时，就可以大大提高效率，缩短开发周期. 我们利用 GE 构架来实现 MSD Editor 和 DFD Editor 时，工作量分别为 0 人/月(约 2 天)和 0.5 人/月，而过去开发类似系统所需的工作量则均为 6 人/月.

参考文献

1 Garlan David *et al*. Summary of the dagstuhl workshop on software architecture. ACM SIGSOFT Software Engineering Notes, 1995,20(3):63～83

2 Perry D E, Wolf A L. Foundations for the study of software architecture. ACM SIGSOFT Software Engineering Notes, 1992,17(4):40～52

3 Garlan David, Perry D E. Introduction to the special issue on software architecture. IEEE Transactions on Software Engineering, 1995,21(4):269～274

4 Medvidovic Nenad, Talor Richard Nc. A framework for classifying and comparing architecture description languages. ACM SIGSOFT Software Engineering Notes, 1997,22(6):60～76

5 Batory Don, O'Malley Sean. The design and implementation of hierarchical software system with reusable components. ACM Transactions on Software Engineering and Methodology, 1992,1(4):355～398

6 Batory Don, Singhal Vivek *et al*. The GenVoca model of software-system generators. IEEE Software, 1994,9(6):89～94

7 Batory Don, Coglianese Lou. Creating reference architecture: an example from avionics. ACM Software Engineering Notes, 1995,8(proceedings of the SSR'95):27～37

Research of Software Architecture Based on GenVoca Model

YU Yong XU Jin ZHAO Wen-yun XU You-ming QIAN Le-qiu

(*Department of Computer Science Fudan University Shanghai* 200433)

Abstract GenVoca is a domain-independent model of hierarchical software system design and construction that is based on large-scale reuse. This model is a blueprint for implementing software component technologies in many domains. In this paper, the authors discuss the main features of GenVoca model and present a reusable graphical editor (GE) software architecture based on this model. They explain the realms and their components of the architecture, examine the relations between these components, and show how combining these components into some graphical editors is achieved.

Key words GenVoca model, component, realm, software architecture, graphical editor.