计算机科学 1996 Vol. 23 №. 6

# 标本作为软件重用单元及其实现途径与策略*)

王振宇 谢 江
(武汉数字工程研究所 武汉 430074)

**摘 要** Besides software components, software architecture or frame could be reused in application software development. This paper describes the software pattern reuse techniques developed by authors in Ada-like style. Concept, approach and implementation considerations about pattern reuse are given The reuse approach of patterns is taking software components as their parameters. The key implementation treatment is a preprocessor to generate an instance of a pattern template. Whereas, a proposal of standard composition of reuse assets is put forward in this paper.

**关键词** Software reuse, Pattren, Pattern reuse, Software architecture

## 1. 引 言

软件重用可以显著提高软件生产率，降低软件成本，增强软件产品的可靠性，使软件生产真正走出"手工作坊"阶段，可以说，没有软件重用就没有软件工程。在传统的关于重用的研究中，人们只着重于功能部件的重用。而在软件的设计开发过程中，特别是对大型、复杂的软件系统而言，开发者不仅要考虑到体系结构的功能方面的需求，而且要花费很多精力考虑其非功能方面的需求。所谓非功能方面，重要的一点就是可重用性。另外也包括可改变性、可测试性、效率和可靠性等。只有能够达到较理想的非功能特性的软件才称得上是好的软件。如果软件重用只能达到功能方面的要求，可以说，这种重用的适用性是很有限的，重用的质量也受到影响。

在软件工程领域中，软件体系结构(SA)和标本(pattern)已经成为越来越重要的课题。它们是设计的技术基础，可以作为满足需求的框架，能促使软件工程的重点从功能向结构转移，以支持复杂的大型软件系统的开发和维护。[3]

## 2. 标本与软件体系结构

软件体系结构可以用三元组定义为：

软件体系结构 ::= (单元，形式，设计理由)

这就是说，一个软件体系结构就是一个具有特殊形式的结构单元的集合[4]。

- 单元 分为三个不同的级别：处理单元，数据单元和连接单元。处理单元是提供数据单元变换的部件；数据单元包含了使用和变换的信息；连接单元将结构中不同部分连接起来。
- 形式 SA 的形式包括加权特性和关系，特性用来约束对结构单元的选择，即用来定义结构所要求的每个单元的约束程度。关系用来约束结构单元的位置，也就是约束不同单元如何互相作用以及在结构中它们之间的互相关系是如何组织的。
- 设计理由 是结构的基础，在定义一个结构时，设计理由是不同的，它把握了结构式样选择的动机。

软件体系结构明显地考虑到软件系统的功能方面和非功能方面，优秀的软件人员可以在一定的时间内对不同的任务设计出好的软件，就是因为在他们的脑海中已经积累了许多问题及相关的解决方法，对设计中遇到的新问题，他可以采用曾经用过的解决类似问题的方法来解决。

标本和软件体系结构有着密切的关系。软件体系结构的一个标本描述设计过程中经常出现的一类特定设计问题，并提供已经过考验的通用解决方案。每个标本[1]：

- 提出一个特定设计问题及相应的解决办法。
- 提供如何实现软件系统的特定结构或功能原则的预定方案，具体内容是描述该系统的各个部分以

*)本课题得到武汉大学软件工程国家重点实验室的支持 王振宇 研究员，博士生导师，主要从事 Ada 软件工程、算法复杂性分析及程序复杂性度量方面的研究。谢 江 硕士，主要从事软件重用技术和软件工程方法学的研究。

及它们的协作关系和责任。

- 收集现有的、经过实践考验的设计经验。对分类和例子层上的抽象进行识别、命名和规定。
- 提供设计原理的一个公用词典和含义说明。
- 帮助缓解软件复杂性。
- 在软件开发时用作可重用的部件。
- 可针对某一特定应用领域或保持相对独立。
- 解决软件设计中的功能要求和非功能要求。

标本以不同的形式存在，有的只定义一种应用软件的基本结构，具体的实现甚至所使用的语言都由使用者去完成；有的则提供了用具体的编程语言写的程序，实现特定的设计。这所有的标本合在一起互相依存构成一系统，即标本系统。用标本系统就可以构筑软件的总体结构，支持用户系统化地开发具有确定特性的软件系统。

具体的软件体系结构往往和特定的应用领域有着密切的联系。由此，人们提出了专用领域软件体系结构(DSSA)。一个专用领域软件体系结构包括一个领域中与结构化和应用组合有关的部件、相互联系、约束条件/设计原理、可变性和通用性。它描述一种高级模型和在领域应用中表明其通用性和可变性的约束条件[5]。

DSSA 和领域分析密切相关。通过领域分析，可以建立起领域的模型，定义单元并在专业领域软件中实现继承；还可建立专门领域的软件结构，不仅提供了一个可重用的、合适的框架，而且抓住了设计原理并提供其适应度[6]。

专用领域的标本的建立与领域分析同样有着密切联系。每个标本的设计原理、变形以及不变性特征都可以通过领域分析得到答案；通过领域分析识别一个对象的可重用性，特别是可变性与不变性特征，就可以为开发新标本提供出发点，对已存在的标本进行分类。而且，领域分析为系统结构化奠定了基础，它将领域中的问题识别、抽象、分类，对系统、子系统、体系结构或模型的设计决策都会有所帮助。

## 3. 基本概念、途径及实现策略

传统的部件重用只实现了源代码级的重用。软件重用技术中，除了功能部件重用，还应当有软件体系结构或框架的重用。由于软件的设计决定了软件的结构，因此，后一种重用本质上是软件设计的重用。标本重用是软件重用技术中不可分割的一部分，它是软件设计重用的一种；标本重用的实现途径是将标本作为重用单元并参数化，通过预处理得到标本实例，从而实现重用。标本重用对面向重用的软件开发方法有重要意义，尤其是在大型、复杂软件系统的开发过程中，它可以降低开发难度，提高生产效率和软件产品的可靠性。利用标本重用，可以：

- 以系统化方法寻找一种适合待开发的软件系统的结构；
- 易于开发极特殊的结构，这种结构具有确定的功能特性和非功能特性；
- 以标本提供的设计结构为指导，实现给定的软件总体结构，以防止软件总体结构与实际之间存在差异。

### 3.1 标本作为重用单元

我们提出将标本作为重用单元用来构筑软件系统，这是实现标本重用的直接办法。标本作为重用单元丰富了重用库的内容，它是表达设计经验的新手段。利用标本重用，新开发的软件可以相对容易地达到非功能方面的需求。

在成功的重用例子中，管理支持、单一的结构、专用的部件群、固定的领域、标准和组织(organizational)支持才是关键[2]。软件单元组成的标准化，是单元形成、重用及形成系统的关键，因为一种恰当的软件单元组成标准，可以帮助用户理解重用单元，简化重用库的使用，有利于可重用单元库的发展、管理与维护，由此，本文提出了一种标本的组成标准的建议：

〈标本〉::=〈问题描述〉〈解决方案〉〈建立和使用情况〉

〈问题描述〉::=〈名字〉〈设计原理〉〈适用性〉〈类别〉〈关键词〉〈使用限制〉

〈类别〉::=〈粒度〉〈功能〉〈结构原理〉

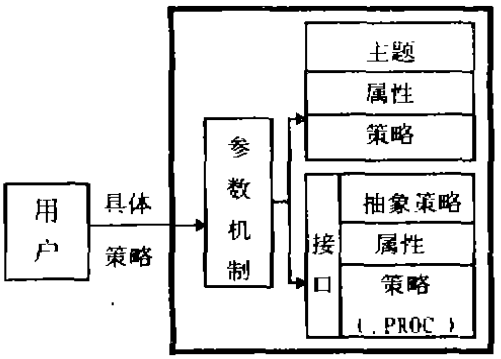
〈解决方案〉::=〈标本结构图〉〈父标本〉〈接口描述〉〈变型〉〈例子〉〈参考文献〉〈控制流图〉〈标本实现〉

〈接口描述〉::=〈参与者或协作者 1〉〈功能 1〉…〈参与者或协作者 N〉〈功能 N〉

〈建立和使用情况〉::=〈作者〉〈建立日期〉〈上次使用时间〉〈使用状态〉〈应用环境〉〈出错记录〉

以主题-策略标本(Subject-Policy Pattern)[1]为例，它的部分组成描述如下：

结构图：




**参与者或协作者**:主题

功能:描述和实现一项任务或服务,其中的算法或行为可以选择,也可以更换。

**参与者或协作者**:抽象策略

功能:描述一个接口,它具有.PROC过程。

**参与者或协作者**:具体策略

功能:以不同的方式实现抽象策略定义的那个接口。在程序运行过程中,由用户通过参数机制配置并加以封装。

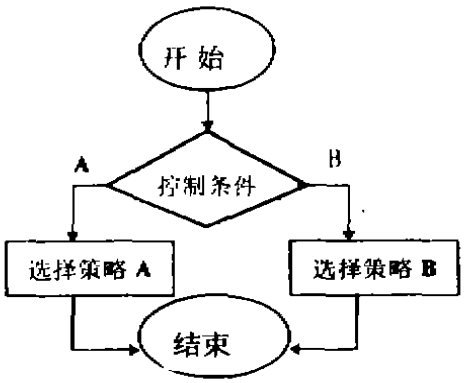
**设计原理**:有时候软件系统需要为它承担的某项工作提供多种供选择的解决办法。要求之一必须允许在软件运行期间选择不同的策略;要求之二是可以提供其它的软件系统以随时改变特定功能所依赖的算法。为了便于达到这些目标,软件结构必须有静态选择和动态配置的能力。

**适用性**:

①应用软件必须随时改变或更换它承担的任务的解决策略,以适应技术的变化和用户具体要求的变化,以便于把它集成到不同的环境中。

②应用软件必须为一项任务提供多种供选择的解决策略,而且必须可以在软件运行期间选择策略,以便在指定的情况下取得最佳结果。

**控制流图**:




### 3.2 标本重用的途径——参数化

要理解标本,一个重要的概念就在于标本是一个不变与多变的统一体。所谓不变,是指一个标本对特定问题提供的解决方案是一定的;所谓多变,则是指标本允许一组相关的对象和类之中有某些变化。例如,有的标本允许具体算法的改变,有的允许对对象访问形式的改变,还有的允许其用户的变化。可以说,这种多变性正是标本的一个重要特征,体现了标本的灵活性与抽象性,提高了其重用的价值;使标本能兼顾系统的功能特性与非功能特性;在标本重用过程中,标本本身有了一定的主动性,可以生成一个新的程序而不是简单地被组合。不过,也正是因为这种多变性,使得标本难以把握。另一方面,在软件系统的开发过程中,从一个应用软件的基本结构到具体实现特定的设计结构,都要进行处理。标本的粒度是标本分类的一个重要类别,它包括三个层次[1]:

·框架性标本　表示软件系统的基本结构组织方式。它提供一整套预先确定的子系统,规定它们的责任;该框架中还包含了如何组织这些子系统的各种规则和指导原则。

·设计性标本　提供如何精心制作软件系统的子系统或部件的基本安排,它描述反复遇到的通信部件的结构,以解决在特定的衔接环境中的一般性设计问题。

·实现性标本　描述如何实现特定的部件、它们的部分功能、或它们在给定的软件总体结构中与其它部件之间的关系。它们往往直接针对某种编程语言。实现性标本是最低的抽象层次。

每个标本都完成一个特定的、相对完整的设计细目,可以被集成到更大的软件系统中,也可以由一些子标本通过一定的规则来组成。根据标本的这些特点,标本的重用以部件作为参数,这是实现体系结构重用的一种有效的途径。由此,标本的调用参数可分为两种,一种是部件参数,另一种则是类属参数(包括变量参数、类型参数和子程序参数)。标本重用时,对部件参数需要进行预处理;类属参数则由编译系统处理。

以部件为参数比以子程序或以类型为参数又有了一次质的提高。标本以部件为参数与Ada语言中的上下文子句"将程序包拿来用"有本质的区别:"上下文子句"只是解决了可见性问题,支持Ada源程序的分离编译,但它"拿来"的程序包不是参数,不可被替换;而作为参数的部件,虽然相应的实在可能也是程序包,但参数本身并不确指某一个程序包,在标

本被重用时，可以被替换为实际的部件。

由此可见，以部件作为标本的重用参数是对Ada类属单元的一种扩充，使标本更加灵活，便于使用，又无损于类属单元的封装和信息隐藏。同时，部件参数增强了软件重用和参数化的手段，它利用各个部件都可以完成一定的功能的特点，使标本侧重于软件总体结构，而忽略系统的实现细节，具体功能的实现由不同的部件去完成。

### 3.3 标本重用的实现策略——预处理

一个子程序不仅可以实现一个具体的算法，而且也能用以表达一种设计思想或是一种体系结构。对于Ada语言应用程序，我们把标本映射为子程序或类属子程序(如果该标本需要设置类属参数的话)。标本的参数机制具有类Ada风格，并映射为Ada代码，这一工作可以作为Ada软件开发工具环境的一部分。标本由标本声明所声明，可以形式化地描述为：

〈标本声明〉::=**pattern**〈标本规格说明〉〈标本体〉

〈标本规格说明〉::=[〈部件参数声明〉][〈类属形式部分〉]〈子程序规格说明〉

〈部件参数声明〉::=〈$部件形参〉{,〈$部件形参〉}

〈部件形参〉::=标识符

〈类属形式部分〉::=**generic**{〈类属参数声明〉}

〈标本体〉::=〈子程序体〉

主题-策略标本的标本实现可以描述为：

```
pattern
    $S1, $S2;
    --S1,S2是标本的部件形参,表示供选择的策略。它们
      都具有.PROC处理过程,在预处理中被替换为相应
      实参
    generic
    procedure SELECT_PATTERN is
    begin
    if CONDITION then $S1.PROC
        else $S2.PROC
    end if;
end SELECT_PATTERN;
```

标本和Ada语言的类属单元一样是一种模板而不是一种实在的程序单元，我们可以对它取例但不能调用它。因此，对标本都要进行预处理。下图说明了标本预处理的作用：

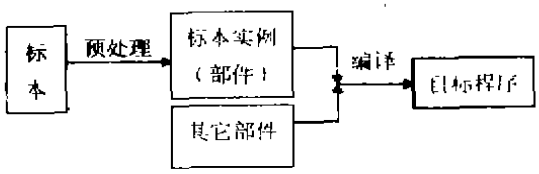



图1 标本预处理的作用

标本预处理就是对标本的部件形参进行处理，替换为对应的实参：

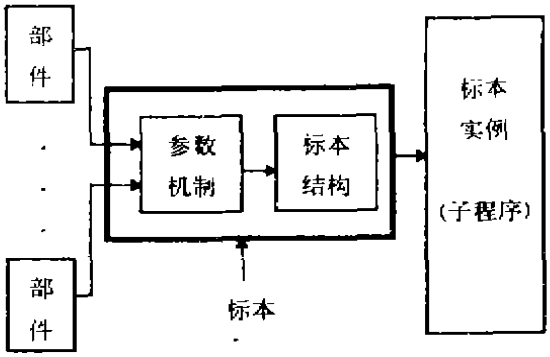



图2 标本预处理的过程

以主题--策略标本为例，它的实例化过程可以这样描述：

```
with TEXT_IO;--假定TEXT_IO总是需要的
procedure STPN is new SELECT_PATTERN
    ($S1=>S_A;$S2=>S_B);
```

系统对标本实行预处理，将标本的部件形参替换为相应实参得到标本实例：

```
with TEXT_IO;
with S_A,S_B;
procedure STPN is
begin
   if CONDITION then S_A.PROC;
                else S_B.PROC;
   end if;
end STPN;
```

其中S_A和S_B是部件实参。实际上，预处理过程本身并不检查S_A和S_B在当前程序库中是否已经存在。S_A和S_B可能是需要新构造的部件的名字(如果新部件是一个类属单元，则是它的实例的名字)，也可能是已有程序包的名字或是类属程序包实例的名字，重要的是，在标本实例交付编译之前它们应当已经产生。交付编译时，首先提交的是部件单元(新构造的部件、程序包或者类属程序包实例)，然后才编译标本实例。

通过这样的预处理，标本被实例化，成为编译单元，从而实现重用。

## 4. 可重用软件单元库的设计与实现

可重用软件单元库系统RSALS(Reusable Software Asset Library System)是在SUN工作站上的UNIX环境中使用Xview图形工具包和C语言设计和实现的。它有两个级别的用户：库管理员ABA(Asset Base Administrator)和一般用户，这是为了便于保持系统的完整性和一致性，防止一般用户对

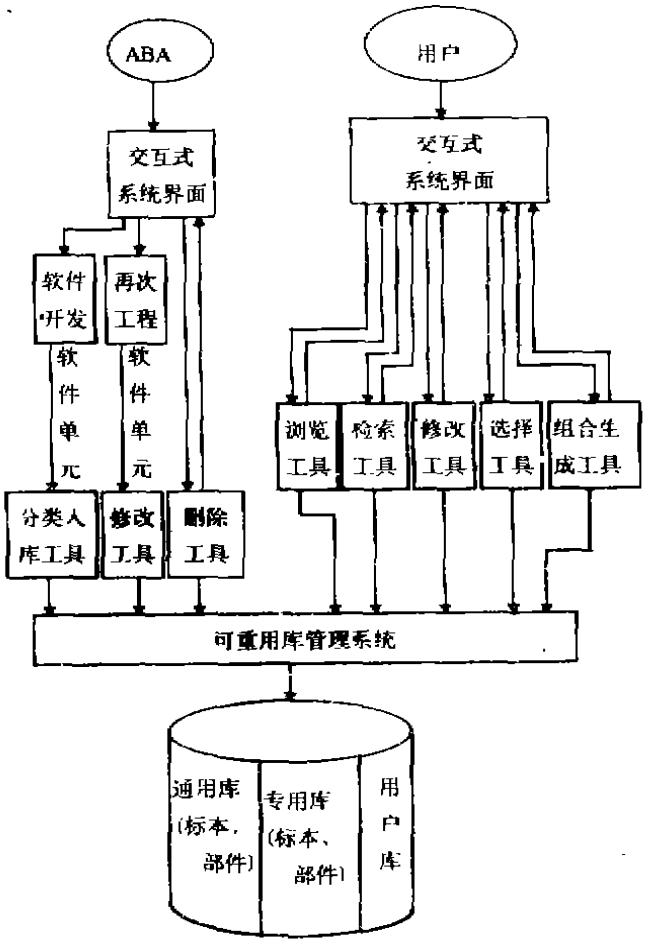



图3 RSALS的总体设计

已有单元进行非法操作；另外，它有三个库，专用库、通用库和用户库。专用库用来存放专用领域中的软部件和标本，直接面向专业领域，在该领域的软件系统中具有较高的重用率。通用库用来存放通用的软部件和标本，由于重用一般是面向特定领域的，故通用的软件单元数目较少。用户库则是提供给一般的用户使用，存放他们从该系统中收集的可重用单元及由它们构造的新的软件系统。图3给出了RSALS的总体设计。

可重用软件单元库具有良好的用户界面，所有部件、标本和类别均已可视化，形象直观，易于检索和查找。图中所示的插入、删除和修改操作只能由库管理员来进行，一般的用户没有授权完成这几个操作。

建立可重用库的目的在于：在用户插入、删除和使用库中单元时能提供一些智能化的帮助。在重用库的不断完善和其内容不断丰富的过程中，该库管理系统可自动维护库中单元之间接口的一致性。

前面阐述的关于标本重用的途径及其实现策略均已实现，且RSALS与Ada编译器之间具有良好的接口，形成了一个基于重用库的集成化软件开发环境，用户可用以选择、拼装重用单元，通过编译直接得到可执行代码。

## 参考文献

[1]Buschmann F. et al., How Patterns Build Software Systems, *Electronic Design*, 1995. 1

[2]Griss M. et al., Systematic Software Reuse Objects and Frameworks Are not Enough, *Proc. of the ACM SIGSOFT Symp. on Software Reusability(SSR'95)*, 1995. 5

[3]Helm R., Patterns, Architecture and Software, *ACM SIGPLAN Notices*, 31(1), 1996.

[4]Perry D. E. et al., Foundations for the Study of Software Architecture, *ACM SIGSOFT SOFTWARE ENG. NOTES*, 17(4), 1992

[5]Rolling W. A., A Preliminary Annotated Bibliography on Domain Engineering, *Software Engineering Notes*, 19(3), 1994

[6]Tracz W. et al., A Domain-Specific Software Architecture Engineering Process Outline, *ACM SIGSOFT SOFT. ENG. NOTES*, 18(2)