软件学院推荐教材

软件体系结构

张友生　编著

清华大学出版社

软件学院推荐教材

软件体系结构

张友生　编著

清 华 大 学 出 版 社
北　京

内 容 简 介

本书系统地介绍软件体系结构的基本原理、方法和实践，全面反映软件体系结构研究和应用的最新进展。既讨论软件体系结构的基本理论知识，又介绍软件体系结构的设计和工业界应用实例，强调理论与实践相结合。

全书共七章，第一章简单地介绍软件体系结构的概念、发展和应用现状；第二章讨论软件体系结构建模，包括视图模型、核心模型、生命周期模型和抽象模型；第三章介绍软件体系结构的风格和特定领域软件体系结构；第四章讨论软件体系结构的描述方法，重点介绍软件体系结构描述语言；第五章讨论基于体系结构的软件开发方法，介绍基于体系结构的软件过程；第六章讨论软件体系评估方法，重点介绍ATAM和SAAM方法；第七章介绍软件产品线的原理和方法、框架技术，重点讨论产品线体系结构的设计和演化。

本书可作为计算机软件专业本科生、研究生和软件工程硕士的软件体系结构教材，也可作为软件工程高级培训、系统分析员培训、系统构架设计师培训教材，以及软件开发人员的参考书。

图书在版编目（CIP）数据

软件体系结构/张友生编著. —北京：清华大学出版社，2004.1
ISBN 7-302-07810-6

Ⅰ. 软… Ⅱ. 张… Ⅲ. 软件—系统结构 Ⅳ. TP311.5

中国版本图书馆CIP数据核字（2003）第116065号

出 版 者：清华大学出版社
http://www.tup.com.cn
社 总 机：010-62770175
地　　址：北京清华大学学研大厦
邮　　编：100084
客户服务：010-62776969
组稿编辑：丁　岭
文稿编辑：陶萃渊
封面设计：付剑飞
印 刷 者：北京市清华园胶印厂
装 订 者：三河市金元装订厂
发 行 者：新华书店总店北京发行所
开　　本：185×260　**印张**：16　**字数**：384千字
版　　次：2004年1月第1版　2004年6月第2次印刷
书　　号：ISBN 7-302-07810-6/TP·5694
印　　数：5001～7000
定　　价：29.00元

前　言

随着软件系统规模越来越大、越来越复杂，整个系统的结构和规格说明显得越来越重要。对于大规模的复杂软件系统，其总体的系统结构设计和规格说明比起对计算的算法和数据结构的选择已经变得明显重要得多。在此种背景下，人们认识到软件体系结构的重要性，并认为对软件体系结构的系统、深入的研究将会成为提高软件生产率和解决软件维护问题的新的最有希望的途径。

自从软件系统首次被分成许多模块，模块之间有相互作用，组合起来有整体的属性，软件就具有了体系结构。好的开发者常常会使用一些体系结构模式作为软件系统结构设计策略，但他们并没有规范地、明确地表达出来，这样就无法将他们的知识与别人交流。软件体系结构是设计抽象的进一步发展，满足了更好地理解软件系统，更方便地开发更大、更复杂的软件系统的需要。

事实上，软件总是有体系结构的，不存在没有体系结构的软件。体系结构一词在英文里就是“建筑”的意思。把软件比作一座楼房，从整体上讲，是因为它有基础、主体和装饰，即操作系统之上的基础设施软件、实现计算逻辑的主体应用程序、方便使用的用户界面程序。从细节上来看每一个程序也是有结构的。早期的结构化程序就是以语句组成模块，模块的聚集和嵌套形成层层调用的程序结构，也就是体系结构。结构化程序的程序（表达）结构和（计算的）逻辑结构的一致性及自顶向下开发方法自然而然地形成了体系结构。由于结构化程序设计时代程序规模不大，通过强调结构化程序设计方法学，自顶向下、逐步求精，并注意模块的耦合性就可以得到相对良好的结构，所以，并未特别研究软件体系结构。

对于软件项目的开发，一个清晰的软件体系结构是首要的。传统的软件开发过程可以划分为从概念直到实现的若干个阶段，包括问题定义、需求分析、软件设计、软件实现及软件测试等。软件体系结构的建立应位于需求分析之后，软件设计之前。但在传统的软件工程方法中，需求和设计之间存在一条很难逾越的鸿沟，从而很难有效地将需求转换为相应的设计。而软件体系结构就是试图在软件需求与软件设计之间架起一座桥梁，着重解决软件系统的结构和需求向实现平坦地过渡的问题。

体系结构在软件开发中为不同的人员提供了共同交流的语言，体现并尝试了系统早期的设计决策，并作为系统设计的抽象，为实现框架和构件的共享和重用、基于体系结构的软件开发提供了有力的支持。鉴于体系结构的重要性，Dewayne Perry 将软件体系结构视为软件开发中第一类重要的设计对象，Barry Boehm 也明确指出：“在没有设计出体系结构及其规则时，整个项目不能继续下去，而且体系结构应该看做是软件开发中可交付的中间产品”。

软件体系结构是根植于软件工程发展起来的一门新兴学科，目前已经成为软件工程研究和实践的主要领域。专门和广泛地研究软件体系结构是从 20 世纪 90 年代才开始的，1993~1995 年之间，卡耐基梅隆大学的 Mary Shaw 与 David Garlan，贝尔实验室的 Perry，南加州大学的 Barry Boehm，斯坦福大学的 David Luckham 等人开始将注意力投向软件体系

结构的研究和学科建设。

目前，软件体系结构领域研究非常活跃，如南加州大学专门成立了软件体系结构研究组，曼彻斯特大学专门成立了软件体系结构研究所。同时，业界许多著名企业的研究中心也将软件体系结构作为重要的研究内容。如由 IBM、Nokia 和 ABB 等企业联合一些大学研究嵌入式系统的体系结构项目。国内也有不少的机构在从事软件体系结构方面的研究，如北京大学软件工程研究所一直从事基于体系结构软件组装的工业化生产方法与平台的研究，北京邮电大学则研究了电信软件的体系结构，国防科学技术大学推出的 CORBA 规范实现平台为体系结构研究提供了基础设施所需的中间件技术。许多大学为计算机软件专业硕士研究生和软件工程硕士研究生都开设了软件体系结构课程。

本书共分七章，第一章简单地介绍软件体系结构的概念、发展和应用现状；第二章讨论软件体系结构建模，包括视图模型、核心模型、生命周期模型和抽象模型；第三章介绍软件体系结构的风格和特定领域软件体系结构；第四章讨论软件体系结构的描述方法，重点介绍软件体系结构描述语言；第五章讨论基于体系结构的软件开发方法，介绍基于体系结构的软件过程；第六章讨论软件体系评估方法，重点介绍 ATAM 和 SAAM 方法；第七章介绍软件产品线的原理和方法、框架技术，重点讨论产品线体系结构的设计和演化。

在本书出版之际，我要特别感谢国内外软件工程和软件体系结构专著、教材和许多高水平论文、报告的作者们（恕不一一列举，名单详见各章中的主要参考文献），他们的作品为本书提供了丰富的营养，使我受益匪浅。我在本书中引用了他们的部分材料，使本书能够尽量反映软件体系结构研究和实践领域的最新进展。

我还要感谢我的妻子何玉云女士，她为本书的稿件整理和校对花费了大量的时间。感谢中南大学信息学院陈松乔教授对我的指导，感谢湖南师范大学物理与信息科学学院陈祖福、匡东满、方茂发等教授对写作工作的支持和帮助。感谢湖南大学软件学院王益民博士，他为我的工作和写作提供了许多方便。

由于作者水平有限，时间紧迫，加上软件体系结构是一门新兴的学科，本身发展很快，对有些新领域作者尚不熟悉。因此，书中难免有不妥和错误之处，我诚恳地期望各位专家和读者不吝指教和帮助。对此，我将深为感激。

张友生

2003 年 12 月

目　录

第一章　软件体系结构概论

1.1　从软件危机谈起

软件危机（software crisis）是指在计算机软件的开发（development）和维护（maintenance）过程中所遇到的一系列严重问题。20 世纪 60 年代末至 70 年代初，“软件危机”一词在计算机界广为流传。事实上，几乎从计算机诞生的那一天起，就出现了软件危机，只不过到了 1968 年在原西德加密施（Garmish）召开的国际软件工程会议上才被人们普遍认识到。

1.1.1　软件危机的表现

1．软件成本日益增长

在计算机发展的早期，大型计算机系统主要是被设计（design）应用于非常狭窄的军事领域。在这个时期，研制计算机的费用主要由国家财政提供，研制者很少考虑到研制代价问题。随着计算机市场化和民用化的发展，代价和成本就成为投资者考虑的最重要的问题之一。20 世纪 50 年代，软件成本（cost）在整个计算机系统成本中所占的比例为 10%~20%。但随着软件产业的发展，软件成本日益增长。相反，计算机硬件随着技术的进步、生产规模的扩大，价格却在不断下降。这样一来，软件成本在计算机系统中所占的比例越来越大。到 20 世纪 60 年代中期，软件成本在计算机系统中所占的比例已经增长到 50%左右。

而且，该数字还在不断地递增，下面是一组来自美国空军计算机系统的数据：1955 年，软件费用约占总费用的 18%，1970 年达到 60%，1975 年达到 72%，1980 年达到 80%，1985 年达到 85%左右。

2．开发进度难以控制

由于软件是逻辑、智力产品，软件的开发需建立庞大的逻辑体系，这是与其他产品的生产不一样的。例如：工厂里要生产某种机器，在时间紧的情况下可以安排工人加班或者实行“三班倒”，而这些方法都不能用在软件开发上。

在软件开发过程中，用户需求（requirement）变化等各种意想不到的情况层出不穷，令软件开发过程很难保证按预定的计划实现，给项目计划和论证工作带来了很大的困难。

Brook 曾经提出：“在已拖延的软件项目上，增加人力只会使其更难按期完成”。事实上，软件系统的结构很复杂，各部分附加联系极大，盲目增加软件开发人员并不能成比例地提高软件开发能力。相反，随着人员数量的增加，人员的组织、协调、通信、培训和管理等方面的问题将更为严重。

许多重要的大型软件开发项目，如 IBM OS/360 和世界范围的军事命令控制系统（WWMCCS），在耗费了大量的人力和财力之后，由于离预定目标相差甚远不得不宣布失败。

3. 软件质量差

软件项目即使能按预定日期完成，结果却不尽人意。1965 年至 1970 年，美国范登堡基地发射火箭多次失败，绝大部分故障是由于应用程序错误造成的。程序的一些微小错误可以造成灾难性的后果，例如，有一次，在美国肯尼迪发射一枚阿脱拉斯火箭，火箭飞离地面几十英里高空开始翻转，地面控制中心被迫下令炸毁。后经检查发现是飞行计划程序里漏掉了一个连字符。就是这样一个小小的疏漏造成了这支价值 1850 万美元的火箭试验失败。

在“软件作坊”里，由于缺乏工程化思想的指导，程序员几乎总是习惯性地以自己的想法去代替用户对软件的需求，软件设计带有随意性，很多功能只是程序员的“一厢情愿”而已，这是造成软件不能令人满意的重要因素。

4. 软件维护困难

正式投入使用的软件，总是存在着一定数量的错误，在不同的运行条件下，软件就会出现故障，因此需要维护。但是，由于在软件设计和开发过程中，没有严格遵循软件开发标准，随意性很大，没有完整的、真实反映系统状况的记录文档，给软件维护造成了巨大的困难。特别是在软件使用过程中，原来的开发人员可能因各种原因已经离开原来的开发组织，使得软件几乎不可维护。

另外，软件修改是一项很“危险”的工作，对一个复杂的逻辑过程，哪怕做一项微小的改动，都可能引入潜在的错误，常常会发生“纠正一个错误带来更多新错误”的问题，从而产生副作用。

有资料表明，工业界为维护软件支付的费用占全部硬件和软件费用的 40%~75%。

1.1.2 软件危机的原因

从软件危机的种种表现和软件作为逻辑产品的特殊性可以发现软件危机的原因。

1. 用户需求不明确

在软件开发过程中，用户需求不明确，其问题主要体现在四个方面。

- 在软件开发出来之前，用户自己也不清楚软件的具体需求；
- 用户对软件需求的描述不精确，可能有遗漏、有二义性，甚至有错误；
- 在软件开发过程中，用户还提出修改软件功能（function）、界面（interface）、支撑环境（environment）等方面的要求；
- 软件开发人员对用户需求的理解与用户本来愿望有差异。

2. 缺乏正确的理论指导

缺乏有力的方法学和工具方面的支持。由于软件不同于大多数其他工业产品，其开发过程是复杂的逻辑思维过程，其产品极大程度地依赖于开发人员高度的智力投入。由于过分地依靠程序设计人员在软件开发过程中的技巧和创造性，加剧软件产品的个性化，也是发生软件危机的一个重要原因。

3．软件规模越来越大

随着软件应用范围的扩大，软件规模愈来愈大。大型软件项目需要组织一定的人力共同完成，而多数管理人员缺乏开发大型软件系统的经验，而多数软件开发人员又缺乏管理方面的经验。各类人员的信息交流不及时、不准确，有时还会产生误解。软件项目开发人员不能有效地、独立自主地处理大型软件的全部关系和各个分支，因此容易产生疏漏和错误。

4．软件复杂度越来越高

软件不仅仅是在规模上快速地发展扩大，而且其复杂性（complexity）也在急剧地增加。软件产品的特殊性和人类智力的局限性，导致人们无力处理“复杂问题”。所谓“复杂问题”的概念是相对的，一旦人们采用先进的组织形式、开发方法和工具提高了软件开发效率和能力，新的、更大的、更复杂的问题又摆在人们的面前。

1.1.3　如何克服软件危机

人们在认真地研究和分析了软件危机背后的真正原因之后，得出了“人们面临的不光是技术问题，更重要的是管理问题。管理不善必然导致失败。”的结论，便开始探索用工程的方法进行软件生产的可能性，即用现代工程的概念、原理、技术和方法进行计算机软件的开发、管理和维护。于是，计算机科学技术的一个新领域——软件工程（software engineering）诞生了。

软件工程是用工程、科学和数学的原则与方法研制、维护计算机软件的有关技术及管理方法。软件工程包括三个要素：方法、工具和过程。

- 软件工程方法为软件开发提供了“如何做”的技术，是完成软件工程项目的技术手段；
- 软件工具是人类在开发软件的活动中智力和体力的扩展和延伸，为软件工程方法提供了自动的或半自动的软件支撑环境；
- 软件工程的过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。

迄今为止，软件工程的研究与应用已经取得很大成就，它在软件开发方法、工具、管理等方面的应用大大缓解了软件危机造成的被动局面。

1.2　构件与软件重用

尽管当前社会的信息化过程对软件需求的增长非常迅速，但目前软件的开发与生产能力却相对不足，这不仅造成许多急需的软件迟迟不能被开发出来，而且形成了软件脱节现象。自 20 世纪 60 年代人们认识到软件危机、并提出软件工程以来，已经对软件开发问题进行了不懈的研究。近年来人们认识到，要提高软件开发效率，提高软件产品质量，必须

采用工程化的开发方法与工业化的生产技术。这包括技术与管理两方面的问题：在技术上，应该采用基于重用（reuse，有些文献翻译为“复用”）的软件生产技术；在管理上，应该采用多维的工程管理模式。

近年来人们认识到，要真正解决软件危机，实现软件的工业化生产是惟一可行的途径。分析传统工业及计算机硬件产业成功的模式可以发现，这些工业的发展模式均是符合标准的零部件/构件（component，有些文献翻译为“组件”或“部件”）生产以及基于标准构件的产品生产，其中，构件是核心和基础，重用是必需的手段。实践表明，这种模式是产业工程化、工业化的成功之路，也将是软件产业发展的必经之路。

软件重用是指在两次或多次不同的软件开发过程中重复使用相同或相近软件元素的过程。软件元素包括程序代码、测试用例、设计文档、设计过程、需求分析文档甚至领域（domain）知识。通常，把这种可重用的元素称做软构件（software component，通常简称为构件），可重用的软件元素越大，我们就说重用的粒度（granularity）越大。

使用软件重用技术可以减少软件开发活动中大量的重复性工作，这样就能提高软件生产率，降低开发成本，缩短开发周期。同时，由于软构件大都经过严格的质量认证，并在实际运行环境中得到检验，因此，重用软构件有助于改善软件质量。此外，大量使用软构件，软件的灵活性和标准化程度也能得到提高。

1.2.1 构件模型及实现

一般认为，构件是指语义完整、语法正确和有可重用价值的单位软件，是软件重用过程中可以明确辨识的系统；结构上，它是语义描述、通信接口和实现代码的复合体。简单地说，构件是具有一定的功能，能够独立工作或能同其他构件装配起来协调工作的程序体，构件的使用同它的开发、生产无关。从抽象程度来看，面向对象（Object Orientation）技术已达到了类级重用（代码重用），它以类为封装的单位。这样的重用粒度还太小，不足以解决异构互操作和效率更高的重用。构件将抽象的程度提到一个更高的层次，它是对一组类的组合进行封装，并代表完成一个或多个功能的特定服务，也为用户提供了多个接口。整个构件隐藏了具体的实现，只用接口对外提供服务。

构件模型（model）是对构件本质特征的抽象描述。目前，国际上已经形成了许多构件模型，这些模型的目标和作用各不相同，其中部分模型属于参考模型（例如 3C 模型），部分模型属于描述模型（例如 RESOLVE 模型和 REBOOT 模型）。还有一部分模型属于实现模型。近年来，已形成三个主要流派，分别是 OMG（Object Management Group，对象管理集团）的 CORBA（Common Object Request Broker Architecture，通用对象请求代理结构）、Sun 的 EJB（Enterprise Java Bean）和 Microsoft 的 DCOM（Distributed Component Object Model，分布式构件对象模型）。这些实现模型将构件的接口与实现进行了有效的分离，提供了构件交互（interaction）的能力，从而增加了重用的机会，并适应了目前网络环境下大型软件系统的需要。

国内许多学者在构件模型的研究方面做了不少的工作，取得了一定的成绩，其中较为突出的是北京大学杨芙清院士等人提出的“青鸟构件模型”，下面，我们就以这个模型为例。

青鸟构件模型充分吸收了上述模型的优点，并与它们相容。青鸟构件模型由外部接口

（interface）与内部结构两部分组成，如图 1-1 所示。

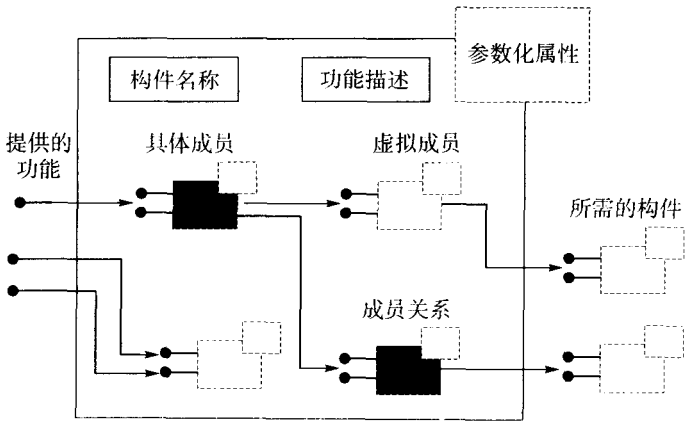

图 1-1　青鸟构件模型

1．外部接口

构件的外部接口是指构件向其重用者提供的基本信息，包括：构件名称、功能描述、对外功能接口、所需的构件、参数化属性等。外部接口是构件与外部世界的一组交互点，说明了构件所提供的那些服务（消息、操作、变量）。

2．内部结构

构件的内部结构包括两方面内容：内部成员以及内部成员之间的关系。其中内部成员包括具体成员与虚拟成员，而成员关系包括内部成员之间的互联，以及内部成员与外部接口之间的互联。

构件实现是指具体实现构件功能的逻辑系统，通常也称为代码构件。构件实现由构件生产者完成，构件重用者则不必关心构件的实现细节。重用者在重用构件时，可以对其定制，也可以对其特例化。

1.2.2　构件获取

存在大量的可重用的构件是有效地使用重用技术的前提。通过对可重用信息与领域的分析，可以得到：

- 可重用信息具有领域特定性，即可重用性不是信息的一种孤立的属性，它依赖于特定的问题和特定的问题解决方法。为此，在识别（identify）、获取（capture）和表示（represent）可重用信息时，应采用面向领域的策略。
- 领域具有内聚性（cohesion）和稳定性（stability），即关于领域的解决方法是充分内聚和充分稳定的。一个领域的规约和实现知识的内聚性，使得可以通过一组有限的、相对较少的可重用信息来解决大量问题。领域的稳定性使得获取的信息可以在较长的时间内多次重用。

领域是一组具有相似或相近软件需求的应用系统所覆盖的功能区域，领域工程（domain engineering）是一组相似或相近系统的应用工程（application engineering）建立基本能力和必备基础的过程。领域工程过程可划分为领域分析、领域设计和领域实现等多个活动，其中的活动与结果如图 1-2 所示。

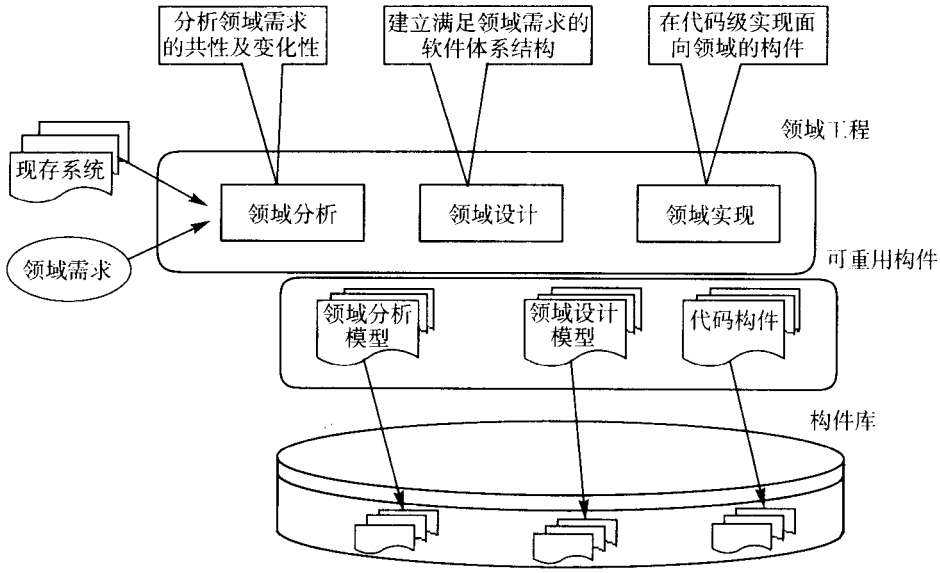

图 1-2 领域工程中的活动与结果

在建立基于构件的软件开发（Component-Based Software Development，简称 CBSD）中，构件获取可以有多种不同的途径：

- 从现有构件中获得符合要求的构件，直接使用或做适应性（flexibility）修改，得到可重用的构件。
- 通过遗留工程（legacy engineering），将具有潜在重用价值的构件提取出来，得到可重用的构件。
- 从市场上购买现成的商业构件，即 COTS（Commercial Off-The-Shell）构件。
- 开发新的符合要求的构件。

一个组织在进行以上决策时，必须考虑到不同方式获取构件的一次性成本和以后的维护成本，然后做出最优的选择。

1.2.3 构件管理

对大量的构件进行有效的管理，以方便构件的存储、检索和提取，是成功地重用构件的必要保证。构件管理的内容包括构件描述、构件分类、构件库组织、人员及权限管理和用户意见反馈等。

1．构件描述

构件模型是对构件本质的抽象描述，主要是为构件的制作与构件的重用提供依据；从管理角度出发，也需要对构件进行描述，例如：实现方式、实现体、注释、生产者、生产日期、大小、价格、版本和关联构件等信息，它们与构件模型共同组成了对构件的完整描述。

2．构件分类与组织

为了给使用者在查询构件时提供方便，同时也为了更好地重用构件，我们必须对收集和开发的构件进行分类（classify）并置于构件库的适当位置。构件的分类方法及相应的库结构对构件的检索和理解有极为深刻的影响。因此，构件库的组织应方便构件的存储和检索。

可重用技术对构件库组织方法的要求是：

- 支持构件库的各种维护动作，如增加、删除以及修改构件，尽量不要影响构件库的结构。
- 不仅要支持精确匹配，还要支持相似构件的查找。
- 不仅能进行简单的语法匹配，而且能够查找在功能或行为方面等价或相似的构件。
- 对应用领域具有较强的描述能力和较好的描述精度。
- 库管理员和用户容易使用。

目前，已有的构件分类方法可以归纳为三大类，分别是关键字分类法、刻面分类法和超文本组织法。

（1）关键字分类法

关键字分类法（keyword classification）是一种最简单的构件库组织方法，其基本思想是：根据领域分析的结果将应用领域的概念按照从抽象到具体的顺序逐次分解为树状或有向无回路图结构。每个概念用一个描述性的关键字表示。不可分解的原子级关键字包含隶属于它的某些构件。图 1-3 给出了构件库的关键字分类结构示例，它支持图形用户界面设计。

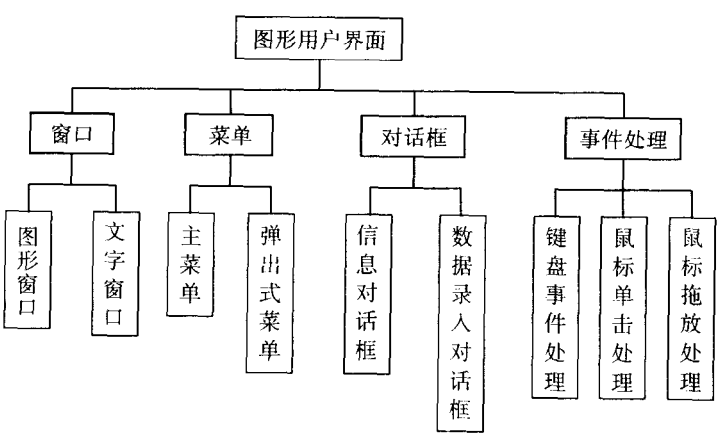

图 1-3　关键字分类结构示例

当加入构件时，库管理员必须对构件的功能或行为进行分析，在浏览上述关键字分类结构的同时将构件置于最合适的原子级关键字之下。如果无法找到构件的属主关键字，可