软件领域中的模式研究

梁义芝[1,2] 王延章[1] 赵晓哲[3] 缪旭东[2]

(大连理工大学信息与决策技术研究所 大连116024)[1]

(海军大连舰艇学院作战软件研究中心 大连116018)[2] (海军大连舰艇学院科研部 大连116018)[3]

The Pattern Research in the Software Domain

LIANG Yi-Zhi[1,2] WANG Yan-Zhang[1] ZHAO Xiao-Zhe[3] MIU Xu-Dong[2]

(Institute of Information and Decision Technology, Dalian University of Technology, Dalian 116024, China)[1]

(Combat Softare Research Center, Dalian Naval Academy, Dalian 116018, China)[2]

(Science and Research Department, Dalian Naval Academy, Dalian 116018, China)[3]

Abstract Software patterns will provide mature soluitons to common software problems. They are abstraction of software domain knowledge and their use realizes the reuse and sharing of software domain knowledge. Based on the introduction and analysis of the primary situation of pattern research in the software domain, this paper reveals the necessity and importance of pattern reseach.

Keywords Patterns, Software patterns, Pattern system

软件模式是目前软件界尤其是面向对象的软件研究的热门话题。它试图对一类通用软件问题提供相对成熟的解决方案，软件模式反映了问题的公共概念结构，包括可用模型、模型的基本原理和使用模型的条件和限制，其使用实现了知识的重用和共享。

软件模式是软件设计与管理知识和经验的提炼、概括与总结，是一种专家知识，不是人工发明和创造的新东西。

1 软件模式的起源与发展

软件模式首先起源于设计模式。设计模式最早是一位叫做 Christopher Alexander 的建筑师提出来的，他着眼于研究建筑物的通用结构，试图找到一种结构化、可复用的方法，以在图纸上捕捉到伟大的建筑物的基本要素。

第一篇关于软件模式的文章是 Ward Cunninghan 和 Kent Beck 合写的“Using Pattern Languages for Object-Oriented Programs”，发表于在 Orlando 举行的 OOPSLA'87 (Object-Oriented Programming Systems, Languages and Applications 1987)会议。当时他们正在使用 Smalltalk 设计用户界面，他们决定借鉴 Alexander 的思想开发一个包含5个模式的很小的模式语言，用于指导初学 Smalltalk 语言的程序员，他们最终将这一有意义的工作总结并发表。

1988年底，Jim Coplien 开始编写 C++ 习语(设计模式的一种)目录并于1991年出版了“Advanced C++ Programming Styles and Idioms”一书。

OOPSLA'91会议上，有关设计模式的讨论成为热点，并决定用 Alexander 著作中描述建筑模式的方式来描述软件设计模式。

1991年，Erich Gamma 完成了他的博士论文：“Object-Oriented Software Development based on ET++: Design Patterns, Class Library, Tools”。

1992年和1993年的 OOPSLA 成立了讨论面向对象设计模式的专题小组。

1993年8月，Kent Beck 等发起了著名的山坡小组(Hillside group)讨论会，1994年4月 Hillside group 再次聚集到一起，策划第一届 PLOP (Pattern Languages of Program Design)会议。

软件模式被广泛应用归功于1995年出版的由 Erich Gamma, Richar Help, Ralph Johnson 和 John Vlissides(通常被叫做 GOF，即 Gang of Four)合著的一本名为“Design Patterns: Elements of Reusable Object-Oriented Software”的书，这本书总结了面向对象的软件设计中的23个设计模式，设计模式从此成为面向对象领域的一个重要分支，软件模式的研究成为软件领域的研究热点。

后来的研究表明，模式存在于软件开发的各个阶段，如：面向对象的软件分析[2]、定义软件体系结构[3]以及工程组织和整个软件开发过程[4,5]。软件模式从最开始的设计模式发展到多种软件模式，包括分析模式、组织模式、过程模式、体系结构模式等。

许多研究者注意到，模式不是孤立存在的，相互之间有关联，应将其作为一个完整的系统即模式系统[6]进行研究，这导致了多种模式系统的开发。最近，模式和模式系统甚至被用于创建软件文档[7]以及对软件产品家族的可变性建模[8]，Agent 设计模式被用于描述移动 Agent 设计中公共问题的解[9]，有些 CASE 工具嵌入了设计模式或具有管理设计模式的功能[10]。

设计模式已经用于实际的开发项目中，文[11]中 Kent Beck 等介绍了在一些不同的工业集团中(包括 Hewitt Associates, Orient Overseas Container Limited, AT&T, Motorola, BNR, Siemens, and IBM)应用设计模式的经验与教训。文[12]中作者介绍了设计模式应用于通信软件领域的几个项目中的情况。

2 模式的定义

Alexander 对模式的定义是：每个模式是一个三元组的规则，它表达了给定上下文、问题以及解之间的关系。其中上下文(Context)是指产生某一问题的情形；问题(Problem)是

指在上述上下文情形下反复出现的待解问题，它包含一系列设计要素(Forces)，即解决该问题需考虑的各个方面，如解的部分必须满足的要求、必须考虑的约束条件以及解应该具备的特性等；解是指对这一类问题反复验证的成功的解决方案。后来提出的多种模式定义都没有脱离 Alexander 提出的三元组的基本概念。如以下模式定义：

模式是在特定上下文情形下对某一特定问题的经反复验证的解，其中的解包括解决策略、依据、组成部件、部件的职能及关系、部件间的协同方式等。

总之，模式提出问题，描述可能引发问题的上下文，分析问题，提供并说明问题的解。

3 模式的描述

有多种模式形式用于描述模式，最常见的模式形式有：Alexanderian Form，The GOF Form，The Portland Form，The Coplien Form。

Alexanderian Form 是由 Christopher Alexander 提出的最早的模式形式。后来的模式形式基本上是在 Alexanderian Form 的基础上，或是更清楚地划分模式描述内容的组成部分，或是增加一些描述内容形成的。

The GOF Form 是 GOF 在描述设计模式中使用的模式形式，用于描述面向对象的软件设计模式，是描述内容最丰富的模式形式，其具体内容可见文[1]。

The Portland Form 是由 Ward Cunningham 等三位作者在 PLOP 94提出的一种模式形式，因三位作者均来自 Portland 而得名。The Portland Form 实际上是直接使用了 Alexanderian Form 只是在编排格式上有些简化。

The Coplien Form 由 Coplien 提出，反映了 Alexanderian Form 的基本内容，将模式描述的内容划分为相应的几个部分(Section)，每一部分由名字及其包含的内容组成。

无论哪种模式形式，其描述的基本内容大致相同，将其总结为如下的模式应描述的基本内容：

名称(Name)：名称是模式的标识。

模式名称应能代表模式的核心内容，查找模式时，首先看到的是模式的名称，一个能反映模式所解决问题及解决方法的模式名称将非常便于模式的查找与使用，尤其当人们不熟悉该模式系统时更是如此；模式名称将很快成为一个工作组的"语言词汇"，因此一个简短的有意义的名称将有利于开发小组成员的交流，甚至成为软件领域的通用词汇，如，在讨论体系结构模式时，关于"管道/过滤器"即"客户/服务器"模式，人们只需提及模式名称而不必进行详细解释，因为该名称已经代表了大家公认的通用的模式。

有一些模式名称隐含了模式的历史意义或文化背景，对于不了解其历史渊源的人难于理解和记忆，可以使用别名(Alias)代表其核心内容，便于理解和使用。

问题(Problem)：描述模式要解决的问题。

一个言简意赅的问题描述可以帮助模式查询者决定是否需要理解模式的其它内容再决定是否采用此模式，因此问题描述经常可以作为模式选择的主索引。

上下文(Context)：描述问题和问题的解重复出现的前提，说明模式的适用性，为模式施用之前的系统状态。

上下文是单个模式组成模式系统的关键因素。模式应用于当前的系统状态，即上下文，通过问题的解决改变系统状态，产生新的上下文，该上下文将适于另一个模式的施用，因此可以说是上下文将相关模式"编织"成为一个模式系统。

设计要素(Forces)：或称为设计要求，描述解决问题需考虑的各个方面，如解的部分必须满足的要求、必须考虑的约束条件以及解应该具备的特性等，设计要素很可能相互关联或冲突，需要权衡。

设计要素是一个模式的关键部分，揭示了问题的复杂性，对设计要素的理解可以加深对问题的理解和对问题的解的认识，从而合理有效地选择和使用模式。

解(Solution)：描述静态结构和如何获得期望输出的动态行为或规则。可用图形、图表及文字描述解的结构、成员及成员之间的合作，以说明问题如何解决。问题的解可以描述静态结构也可描述动态行为，静态结构说明解的形式和组织，动态行为描述获得解的步骤与过程。

解可以描述问题的全部解决方案，也可以只描述部分方案，而给出能描述另一部分解决方案的其它模式。

结果上下文(Resulting Context)：模式施用后的系统状态，包括模式施用的结果、新的上下文可能引发的问题、下一步可能使用哪些相关模式。

相关模式(Related Patterns)：与该模式相关的模式，如该模式的前任模式(Predecessor Patterns)-其应用导致该模式的使用；该模式的接任模式(Successor Patterns)-继该模式之后使用的模式；该模式的伴同模式(Codependent Patterns)-可能或必须与该模式同时使用的模式；该模式的备选模式(Alternative Patterns)-与该模式有相同的问题但受不同的设计要素与约束(Forces)限制因而给出与该模式不同解的模式。

一些特殊模式可能需要描述其它内容，如设计模式的描述内容还包括模式的分类、意图、动机、代码示例等。

4 软件模式的分类

自从设计模式被软件界普遍接受以来，人们逐渐认识到软件领域中模式的重要性，软件模式的研究深入到软件工程的各个方面，软件模式分为设计模式、分析模式、体系结构模式、组织模式、过程模式等。

4.1 设计模式

设计模式是最早提出并且在目前仍是应用最广泛的一类软件模式，产生并应用于软件生命周期过程中的软件设计阶段。

GOF 所著的"Design Patterns：Elements of Reusable Object-Oriented Software"一书是关于软件设计模式的经典之作(该书的中文版已由机械工业出版社出版)。作者总结了面向对象软件开发中的软件设计经验，文档化为设计模式，阐述了模式在建造复杂系统过程中所处的角色，为如何引用一组精心设计的模式提供了一个实用方法，可以帮助开发者针对特定应用问题使用适当的模式进行设计。

设计模式描述了面向对象软件设计过程中针对特定问题的简洁而优雅的设计方案，抽象并确定了一个通用设计结构的主要方面，包括其所包含的类和实例，它们的角色、协作方式以及职责分配，该设计结构可复用于面向对象的设计。

GOF 总结了23种设计模式并按目的准则和范围准则对这23种模式进行分类，形成如表1所示的二维表。

模式按目的分为创建型、结构型、行为型三种，创建型模式与对象的创建有关，结构型模式处理类或对象的组合，行为型模式描述类或对象怎样进行交互和怎样分配职责；模式按

范围分为是用于类还是用于对象，类模式处理类和子类之间的关系，这些关系通过继承建立，是在编译时即可确定的静态关系，对象模式处理对象间的关系，这些关系是动态的，运行时可以改变。

表1 设计模式空间

		模式目的		
		创建型	结构型	行为型
模式范围	类	Factory Method	Adapter (Class)	Interpreter Template Mtehod
	对象	Abstract Factory Builder Prototype Singleton	Adapter (Object) Bridge Composite Decorator Facade Flyweight Proxy	Chan of Responsibility Command Iterator Mediator Memento Observer State Strategy Visitor

表1中的23种设计模式在面向对象软件系统中有广泛的应用，是面向对象设计专家多年经验的总结，它们将帮助设计者将新的设计建立在以往工作的基础上，复用成功的设计方案，更快更好地完成系统设计。

4.2 分析模式(Analysis Patterns)

分析模式产生并应用于软件生命周期中的需求分析阶段，描述该阶段产生的行业业务过程模型。与过程模式不同，分析模式不反映实际的软件实现，而是体现行业业务过程的概念结构。分析模式是多种行业的业务过程的抽象，可用于多种行业的需求分析。

人们发现，在针对一个新的项目进行需求分析时会遇到许多以前对其它项目进行需求分析时遇到的同样问题，可以用已有的经验与方法进行解决，应用的同时改进了已有的经验与方法，同时积累了更多的经验与方法，这些经验与方法按照模式的概念抽象为分析模式，可以指导以后的需求分析并实现复用。

分析模式按用途划分为以下类型：Accountability；Observations and Measurements；Observations for Corporate Financ；Referring to Objects；Inventory and Accounting；Planning；Trading；Application Facades。

参与者(Party)模式[2]是 Accountability 类型中的基本模式，用于为所有的层次组织结构建模，是人、公司、组织等特定概念的一般性概念，这些特定概念可从参与者模式直接获取其对应的概念层次。

例如，当我们考虑为公司的联系方式建模时，其基本属性一般包括地址、电话、E_mail 地址、传真。同样，组织的联系方式也应该包括以上属性，用 UML 描述模型如图1所示。

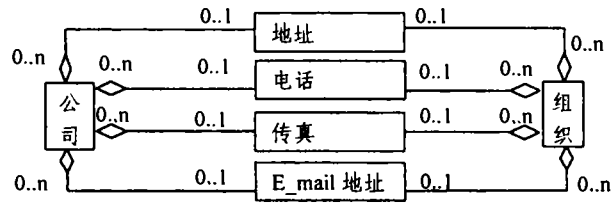

图1 公司和组织的联系方式模型

将公司、组织、人等特殊性概念一般化为参与者概念，即引进参与者类，建成图2所示模型。

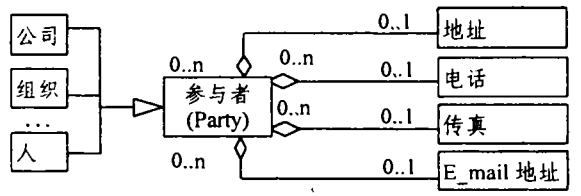

图2 使用参与者模式的联系方式模型

由参与者模式的内容我们可以看出，参与者模式与具体行业无关，可用于不同行业的需求分析，这是所有分析模式的共同属性。

4.3 体系结构模式

软件系统由软件部件和连接软件部件的各种类型的连接器组成；软件系统的整体结构由软件体系结构描述，软件体系结构描述了一组软件部件、软件部件的外部可见特性及其相互关系。

软件部件相互作用的方式不同，需要不同的连接器，形成不同的软件体系结构风格，适于解决不同的领域问题。目前已有多种软件系统结构风格，有多种类型的软件部件和连接器(如管道、过滤器、客户、服务器等)以及它们的组合规则，用于指导软件系统的设计并通过重用降低系统实现成本。

软件体系结构模式是对软件体系结构风格的抽象及在抽象基础之上的形式化。体系结构模式描述了关于软件系统结构的信息，有助于建立和维护系统的内部一致性以及实现面向体系结构风格的分析、设计、检查和复用。

体系结构模式包括管道/过滤器模式、客户/服务器模式、分层结构模式、数据抽象模式等，有些模式是对部件组织的整个体系结构风格的抽象，有些模式只是对部件连接方式的抽象。实际使用时，一般采用自上而下的方法首先用一种或几种体系结构模式确定系统的总体设计，进一步向下描述时，可用其它模式描述具体部件，逐步细化，直至描述整个体系结构。

以下以管道/过滤器模式[3]为例说明一个具体的体系结构模式。

当系统需对一个数据流进行一系列计算时，应采用管道/过滤器模式，其具体描述为：

名称：管道/过滤器模式

问题：对一组有序数据进行一系列计算，有时计算可以同时进行

上下文：可将问题分解为一组转换，每一个转换是一个分离的过程，该过程处理输入数据流产生输出数据流，其输入数据流从其它一个或多个过程获得，其输出数据流输出给其它一个或多个过程

解：该模式的系统模型是被叫做过滤器的部件之间的数据流动

图示：见图3。

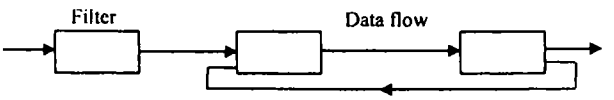

图3 管道/过滤器模式

4.4 组织模式

组织模式是软件开发组织所拥有的关于软件开发组织管

理的知识，覆盖整个软件生命周期过程。与前面所述的其它模式不同，组织模式描述了软件开发组织的管理技术和组织结构，以管理方式影响组织中每个成员能力的发挥，因此，组织模式不具体实现软件开发而是以组织管理的形式支持软件开发。

组织模式受软件开发方法的制约，随着软件开发方法的改进而变化，90年代中期以来，软件生命周期过程逐步脱离了传统的瀑布模型，而采用迭代和增量式的软件过程，因此，目前大部分的组织模式支持软件开发的迭代和增量式过程。

James Coplien 定义的 Domain Expertise in Roles 模式[4]如下所示：

问题：如何为组织中的成员分配角色

上下文：已知关键过程需要的角色及角色应具备的知识与能力

设计要素：所有的角色必须由合格的组织成员担任，担任不同角色的人越多（如不同的角色由不同的人担任），组织中人员的交流越复杂

解：雇用经验丰富的领域专家，每个专家可以承担多个角色

结果上下文：角色可以被成功地分配

高效和成功的软件组织拥有许多重复使用的成功的管理经验，正是这些经验的概括与抽象形成了软件模式中的组织模式。

4.5 过程模式

过程模式是软件开发组织所拥有的关于软件过程管理的知识，覆盖整个软件生命周期过程，过程模式不具体实现软件开发而是以过程管理的形式支持软件开发。软件过程模式与软件开发方法密切相关，不同的软件开发方法需要不同的过程模式支持。

软件过程是指人们用于开发和维护软件及其相关产品的一系列活动、方法、实践和革新。软件过程模型是对实际软件过程的抽象描述，即软件过程元素（一般包括活动、产品、角色）及其关系的抽象描述，是对实际的软件过程的再工程。软件过程模式描述了软件过程中经过验证的成功的方法或一系列活动[5]，是软件过程模型中成熟的过程步（或称为活动）按模式概念体系的抽象。过程模式是软件过程知识的概括与总结，支持过程知识的重用，是构建软件过程的可重用的构造部件，可用于软件组织建立成熟的软件过程。

下面以“技术审查”过程模式[5]为例说明一个具体的过程模式。

名称：技术审查（Technical Review）

问题：软件开发过程中产生的制品需要确认其满足规格要求和组织的质量标准

动机：①软件开发过程中产生的制品（如模型、原型、文档、源代码等）用于生成最终交付用户的软件产品，因此在使用之前必须确认其满足要求；②应尽早发现并修复缺陷，发现越晚，修复成本越高；③没有参加制品生成的人员进行审查有利于发现缺陷；④技术审查是进行工作交流的方式之一。

上下文：有一个或多个制品需要审查，制品已生成并可以进行审查，开发小组已作好审查准备。

解：技术审查过程模式包含6个基本步骤：①开发小组准备审查：准备审查内容以便提交。②开发小组请示审查：向审查主管提交审查内容并说明已准备好被审查。③审查主管进行粗略审查：确定审查内容已经准备好，确认审查可以开始。④审查主管计划并组织审查：准备审查设施，组织审查人员，发放相关素材。⑤开始审查：制品开发人员必须参加以便回答提问并解释相关工作，必须保证所有内容被审查。⑥产生审查结论：审查结论以文档的形式说明被审查制品的优点和缺陷，对每一个缺陷要说明称其为缺陷的理由并给出修正建议。该文档作为开发小组下一步的工作指南，并作为审查主管在下一轮审查时检查其缺陷是否修正的依据之一。

结果上下文：高级管理人员确认开发小组生成的制品满足用户要求与质量标准。

技术审查过程模式可用于软件过程中的模型审查、文档审查、原型审查、需求审查、代码检查等。

5 模式语言、模式系统、模式目录

模式语言最早应用于建筑领域，由 Alexander 提出并推广。Alexander 对模式语言的定义是：模式语言是由一组模式形成的词汇，用于理解和交流思想。这样一组模式可以组合成一个可揭示其成员内在结构和关系的紧密“整体”以实现一个共同目标。Cope 对模式语言的定义是：模式语言定义了一组模式及将该组模式组合成一定体系结构风格的规则。

总之，模式语言由模式及模式组合的规则组成，模式语言中的模式可以是基本模式也可以是复合模式，复合模式可以由基本模式组成也可以由复合模式再复合而成；模式组合的规则定义了生成复合模式应满足的规定。

任何一个复合模式既与其所包含的下层模式也与其被包含的上层模式相关。模式语言为模式提供了灵活的组合方式，可解决不同层次、不同规模的复杂问题。

模式语言不是通用语言，是领域相关的。如设计模式语言不能用于理解、交流和解决软件过程问题。一个领域，其积累的经验与知识越丰富，其模式语言功能越强。

模式系统是由相互作用和相互依赖的若干模式结合而成的具有特定功能的系统。模式系统可由模式语言生成，模式语言描述了生成完整体系结构的所有可能的模式组合和变化，模式系统只是其中的一种。一方面模式语言描述和生成了模式系统，另一方面随着模式系统的增加，模式语言将不断丰富。

模式目录是一组相关模式的列表，这组模式只是内容相关，各自解决不同的问题，不存在模式组合问题，如 Gamma 的23种设计模式。

6 软件模式研究的意义

软件模式抽象并描述了软件生命周期中从需求分析、软件设计、软件测试、软件维护到软件组织管理和软件工程管理的知识，一方面为人们总结专家经验，表达、交流、改进和完善软件活动知识提供了技术方法，为初学者提供了专家知识，另一方面为软件复用提供了新的复用机制。更为重要的是软件模式是当前软件活动知识的高层次的抽象。

计算机科学的历史已经表明，软件领域的任何进步都来自于更好的抽象。当数字计算机在20世纪50年代出现的时候，软件是用机器语言编写的，软件工程师交流的语言基础是机器语言语句；汇编语言、助记符和宏替换可以说是最早的软件抽象技术；到50年代后期，进一步的抽象导致了高级语言的出现和数据类型的使用，高级语言是比机器语言和汇编语言表达能力更强、语义更丰富、更易于交流的语言，使得非计算机

（下转第156页）

持平台为支撑。图3给出基于CORBA软总线的构件组装模型。具体说明如下：

1）即插即用软总线技术　CORBA规范提供了即插即用软总线(ORB)技术，利用有关管理工具将有关领域构件/构架、业务构件及热点业务构件注册到ORB总线的接口库中，供访问和调用。

2）基于CCM的构件管理与装配　利用CCM构件模型的有关机制(如抽象模型、构件配置及打包模型及容器模型相互协作)完成构件的连接装配，实现构件的动态集成。

3）问题说明　关于构架的设计及构架与构件的连接是一个复杂的问题，将另有文章专门讨论。

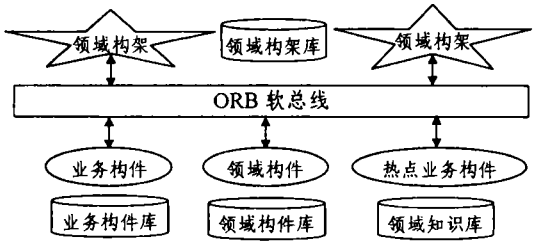

图3　基于软总线的构件组装模型

结束语　RRIS是业界的研究热点，针对可重构实现技术方面存在的不足，本文提出基于业务构件的方法来实现信息系统的快速可重构；将系统的可重构范围界定在功能重构及业务过程重构上，以简化问题的复杂性，并提出有关重构策略；重点研究了BCRRIS的关键技术：构件模型、体系结构、业务构件及框架，限于篇幅有关方面的阐述不能十分详尽，旨在提供某种思路供同行探讨。

参考文献

1　可重构制造系统研究室. http://sy.863cims.net/lab2
2　李群明，张士廉，王成恩，宋国宁. 可重构的企业管理信息系统. http://sy.863cims.net/lab2
3　朱建江，戴勇，王宁生. 基于组件装配的分布式可重构企业信息系统. 南京航空航天大学学报，2001.8
4　沈延森. 快速可重构信息系统及其关键技术研究：[南京航空航天大学博士学位论文]. 2001.5
5　万麟瑞，李绪蓉. 系统集成方法学研究. 计算机学报，1999，22(10)
6　于卫，杨卫东，蔡西尧. 软件体系结构的可视化描述与实现. 计算机科学，1999，26(10)：81～83，31
7　郑明春，王岩冰，等. 一种基于事务构件的ERP系统集成方法. 小型微型计算机系统，2001
8　Aoyama M. New Age of Software Development：How Component-Based Software Engineering Changes the Way of Software Development. 1998 international Workshop on Component-Based Software
9　常煜芬. CORBA构件模型的研究与实现：[南京航空航天大学硕士论文]. 2001.2
10　Herzum P，Sims O. The Business Component Approach. OOPSLA'98 Business Object Component Workshop IV
11　Kozaczynski W. Architecture framework for business components. [C] In：5th Intl. Conf. on Software Reuse，computer socity press，1998. 300～307
12　Bronsard F，Bryan D，Kozaczynski W. Toward Software Plug-and-play. In：Proc. of the Symposium on Software Reusability，Boston，May 1997
13　符进强，汪洋，钱乐秋. 基于动态构件框架的构件演化. 计算机科学，2001，28(1)
14　Baster G，Konana P，Scott J E. Business Components：A Case Study of Bankers Trust. Communications of the ACM，2001，44(5)
15　李景峰，刘西洋，陈平. 一种可重用构件模型——类属构件. 计算机科学，1999，26(9)：83～86，80

(上接第152页)

专业人员使用计算机解决其领域问题成为可能，极大地推动了计算机的普及与发展；面向对象技术的引进，使得类、对象、继承、重用等术语加入到软件语言中，其内涵更丰富，抽象层次更高；软件模式中的设计模式和体系结构模式是高级语言和面向对象语言的使用知识的抽象，是目前内涵最丰富、抽象层次最高的交流语言，如前面所述的23种面向对象的设计模式及多种体系结构模式比高级语言语句和面向对象语言语句包含有更丰富的内容；软件模式中的分析模式是需求分析经验与知识的抽象与总结，为需求分析提供了高层次的交流语言，我们说人类语言极大地推动了人类的文明进步与发展，分析模式为需求分析提供专业术语与语言，将促进需求分析的科学化与规范化；软件模式中的过程模式是软件过程中基本过程步的抽象，为软件过程的分析与改进提供交流语言，是提高软件过程可视性与可控性的基本保障；总之，软件模式是软件生命周期中各项软件活动经验与知识的抽象、概括和总结，使软件活动在更高的层次上进行，将推动软件领域的进步。

参考文献

1　Gamma E，Helm R，Johnson R，Vllissides J. Design Patterns：Elements of Reusable Object-Oriented Software. Addison-Wesley，1995
2　Owler M. Analysis Patterns：Resuable Object Models. Addison-Wesley，1997
3　Haw M. Patterns for Software Architectures. In：Coplien J，Schmidt D. Pattern Languages of Program Design. Addison-Wesley，1995. 453～462
4　Coplien J. A Generative Development-Process Pattern Language. In：Coplien J，Schmidt D. Pattern Languages of Program Design. Addison-Wesley，1995
5　Ambler S. Process Patterns：Building Large-Scale Systems Using Object Technology. Cambridge University Press，1998
6　Coplien J，Vlissides J，Kerth N. Pattern Languages of Program Design-Vol. 2. Addison-Wesley，1995
7　Kotula J. Using Patterns to Create Component Documentation. IEEE Software，1998，15(3)：84～92
8　Keepence B，Mannion M. Using Pattern to Model Variability in Product Families. IEEE Software，1999，16(7)：102～108
9　Aridor Y，Lange D B. Agent Dedign Patterns：Elements of Agent Application Design. In：Proc. of Autonomous Agents '98，Minneapolis，MN，1998. 108～115
10　Florijn G，Meijers M，Van Winsen P. Tool Support for Object-Oriented Patterns. In：Proc. of ECOOP'97，Finland，1997
11　Beck K，et al. Industrial Experience with Design Patterns. In：Proc. of 18th Intl. Conf. on Software Engineering，Berlin，Germany，February 1996. 103～114
12　Schmidt D. Using Design Patterns to Develop Reusable Object-Oriented Communication Software. Communications of the ACM 38，Oct. 1995. 65～74