模式驱动的软件架构设计研究综述

张英杰 朱雪峰

(中国石油大学(北京)石油数据挖掘北京市重点实验室 北京 102249)

(中国石油大学(北京)地球物理与信息工程学院 北京 102249)

摘 要 在目前的软件开发理论和实践过程中,软件生产从需求获取到代码完成都需要人工完成。从需求分析到体系结构的对应与转换依然依赖于软件设计者的技能、经验和创造力;大多数软件代码的生产仍然需要依靠程序员来人工完成。这种传统的软件生产方式为软件产业带来了许多问题。随着软件工程理论和 case 工具的发展,突破传统软件开发方式的方法论逐步被提出。基于模式的软件自动化生产方式能够在从软件抽象模型到软件代码自动生成的过程中节省大量人力,提高软件开发效率,增加软件的自适应性。通过介绍基于模式的软件自动化生产方式来重点研究软件架构的设计。

关键词 体系结构,设计模式,自动化生产,开发效率,自适应

中图法分类号 TP31 **文献标识码** A

Review of Pattern Driven Software Architecture Design

ZHANG Ying-jie ZHU Xue-feng

(Beijing Key Lab of Petroleum Data Mining, China University of Petroleum, Beijing 102249, China)

(College of Geophysics and Information Engineering, China University of Petroleum, Beijing 102249, China)

Abstract In the current software development theory and practice, software production needs to be done manually from aquistion of requirement to code completion. The mapping from software requirements analysis to software architectures still needs designer's skills, experience and creativity. Most software code production still depends on the programmer to do it manually. This traditional way of software production poses many problems for the software industry. With the development of software engineering theory and case tools, the methodology of breaking through traditional way of software development has been put forward gradually. Software automation production methods based on pattern can save a lot of manpower in the process of the software abstract model to the automatic generation of software code. This approach improves the efficiency of software development and increases the adaptability of the software. This paper studied the design of model-driven software architecture by introducing mode-based software automation production methods.

Keywords Architecture, Design pattern, Automated production, Development efficiency, Adaptation

1 引言

在高需求、高投入、高竞争的环境下,软件生产的规模和效率成为软件企业最关注的问题之一。在传统软件开发过程中,大多数软件代码的生产要依靠程序员人工完成,为软件开发带来了大量问题。首先,软件生产的效率低下,项目延期率高;其次,软件质量难以保证;再次,技术更新换代快,软件可移植性低;最后,软件产品难于修改和维护。

随着软件开发理论的发展和完善,传统软件开发方法已不能满足社会和企业的需求,新的软件开发方法被提出。OMG 在 UML 的基础上提出了模型驱动构架(MDA)[1],为从软件抽象模型到软件代码自动生成指明了一个很好的方向。在 Microsoft 发布的 VS. Net 2005 中加入了支持软件工厂[2]的工具 Team System。以软件自动化生产为特点的软件生产方式成为新的关注点。

事实上,各种新型 CASE 工具正在悄悄地改变着软件开发模式。然而,自动化的软件生产时代并不会马上到来。软件生产自动化并不仅仅是几个软件工具,更是一种新的软件开发的方法学。它需要管理人员、开发人员转变传统的开发观念,使技术积累、过程管理等方面都适应新的开发模式。

在软件开发各个阶段独立的自动化方法已经存在。在需求阶段,为易于理解和沟通,基于资源、组织与业务的需求模型(ROB)[3]用递归分解的方法分别从资源、组织与业务抽取需求。ROB 全局统一的树形结构便于形式化定义需求对象,以实现在计算机中存储和管理;在规范需求变更管理组织、流程的基础上,建立数据模型,设计数据操纵语言,定义操纵接口,通过编辑修改对象模拟需求发生的变更,在完整性、一致性规则约束下,自动完成由需求变更引发的完整操作模块[14]。

设计模式阶段,检验前置条件可以被认为是一种设计模

张英杰(1992—),男,硕士生,主要研究方向为软件开发;**朱雪峰**(1973—),男,博士,硕士生导师,主要研究方向为软件可靠性,E-mail:xuefeng.zhu@cup.edu.cn。

式，因为它是对一个不断发生的问题的可重复的解决方案。Microsoft 的代码契约就是设计模式自动化的一个完美的例子，它基于原生 C# 或 Visual Basic，提供了一组 API 以表达检验规则，规则的具体形式包括前置条件、后置条件和对象不变式。

像代码契约这样的编译期扩展固然很好，但官方推出的扩展要花费数年的时间进行开发，直至成熟与稳定。由于存在着很多不同的领域，每个领域又有着它自身的问题，官方的扩展是不可能覆盖所有问题的。

2　模型驱动架构(MDA)

2.1　MDA 概述

模型驱动架构(Model Driven Architecture，MDA)(见图 1)的核心思想是首先抽象出与实现技术无关、完整描述业务功能的核心平台无关模型(Platform Independent Model，PIM)；然后针对不同实现技术制定多个转换规则，通过这些转换规则及辅助工具将 PIM 转换成与具体实现技术相关的平台相关模型(Platform Specific Model，PSM)；最后将 PSM 转换成代码。与传统软件开发方法相比，MDA 关注模型，PIM 开发轻松，提高了软件的生产效率；从 PIM 转换为 PSM 的过程中，可以一对多转换，增加了软件开发的可移植性[1]。软件工厂可被认为是包含 MDA 并在此基础上进行了扩展，比基于 PIM 和 PSM 的 MDA 的定义范围更广泛[2]。

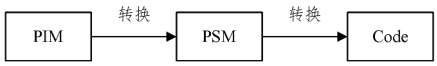

图 1　模型驱动架构

2.2　基于模式的转换方法

基于模式的转换方法是由模型驱动架构产生的方法：模型使用具有精确定义形式和含义的语言对系统做出描述，以适合计算机自动解释。模型之间的变换需要定义一组变换规则，用于描述源语言中的元素如何变换为目标语言中的元素。

基于模式转化的 MDA(见图 2)中，变换由变换规则和变换操作两步构成。变换规则定义变换之前和变换之后的条件，当满足变换规则时，执行变换操作；变换操作执行从模式库中查找模式，然后调用该模式实现模型的转换，完成由元模型元素向对应的目标模型元素的转换。这一过程中会出现模式库模式不足的情形，此时需要定义转换规则，完成模型转换，再将其定义为新的模式存储在模式库中[4]。

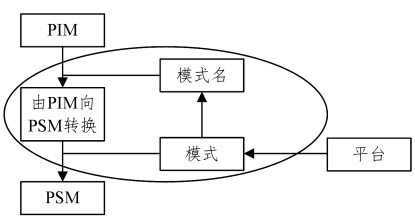

图 2　基于模式转换的 MDA

3　模式

后续不再关心软件转换的具体语言和使用的工具，着重研究基于模式的软件自动化开发方法中的模式。

将模式驱动的软件自动化生成方法中的模式分为需求阶段、设计阶段和实现阶段，主要研究从需求到软件架构的实现；第 4 节引入软件架构模式系统[5]和问题框架[6]等辅助工具到方法中，使得从需求阶段更好地过渡到软件架构设计阶段。

3.1　模式概述

模式(Pattern)[7]是指从生产经验和生活经验中抽象和升华提炼出来的核心知识体系。模式其实就是解决某一类问题的方法论，把解决某类问题的方法总结归纳到理论高度，即为模式。模式是一种参照性指导方略，在一个良好的指导下，初学者能利用训练有素的软件工程师的集体经验，它们记录了软件开发领域已得到充分证明的既有经验，可帮助推广良好的设计实践。

模式一词的指涉范围甚广，它标志了物件之间隐藏的规律关系，而这些物件并不必然是图像、图案，也可以是数字、抽象的关系，甚至可以是思维的方式。只要是一再重复出现的事物，就可能存在某种模式[7]。Alexander 给出了经典定义，即每个模式都描述了一个在我们的环境中不断出现的问题，然后描述了该问题的解决方案的核心[8]。

一般而言，模式＝模式名称＋问题＋解决方案＋效果。模式名称用两个词来描述模式问题、解决方案和效果；问题描述应该在何时使用模式；解决方案描述设计的组成部分、它们之间的相互关系及各自的职责和协作方式；效果描述了模式的使用效果及使用模式时应权衡的问题。

3.2　基于模式的 MDA 方法中的模式

在基于模式的 MDA 方法中，需求分析师获取需求后采用需求模式来描述需求，平台无关模型的建立发生在传统软件开发方法中的需求分析到底层设计阶段，这一阶段采用架构模式和设计模式。平台相关模型的建立发生在传统软件开发方法中的底层设计到编码阶段，这一阶段采用成例模式。

3.2.1　需求分析阶段

除了无关紧要的系统，所有系统的需求本质上彼此相似，例如每个系统都有很多查询功能，每个功能都有自己特有的需求。定义一个业务系统时，相当大比例的需求归属相对少量的类型。因此，以一致的方式定义同样类型的所有需求是必要的。

需求模式：描述定义一种特定类型需求的方法[9]。需求模式应用于单个需求，一次帮助定义一个单一需求。需求人员编写完需求，模式的任务就结束了，但是开发人员和测试人员可以根据需求获得工作提示和测试方法。

需求模式包括如下元素：基本细节、适用性、讨论、内容、模板、实例、额外需求、开发考虑、测试考虑[9]。

使用需求模式的好处：1)需求模式提供指导，如建议包含哪些信息，提出忠告，提醒常见缺陷，以及指出其他应该考虑的问题；2)需求模式节省时间，不需要从头开始写一个需求，模式给出了合适的出发点和开发的基础；3)需求模式促进同种类型需求的一致性。

需求模式使开发者有机会确定一种类型的需求所依赖的基础架构，不必为某一个需求考虑。单个需求处于食物链下游；设计处于食物链上游，以需求为食。Martin Fowler 的分析模式[10]处于需求模式的另一边，在食物链上分析模式比需求模式更高一级，而设计模式轮流以需求模式和分析模式为食。

3.2.2　设计阶段模式

每个模式都是由 3 部分组成的规则，诠释了特定背景、问

题和解决方案之间的关系。作为现实世界的一个元素,模式阐述了特定背景、该背景下反复出现的一系列作用力以及消解这些作用力的空间配置。作为一个语言元素,模式提供了指南,指导如何在相关背景下反复利用这种空间配置,以消解一系列给定的作用力。简而言之,模式既是流程又是作品,既描述了一件具有生命力的作品,又阐述了该作品的创作流程[11]。

从理论上说,模式分类方式有助于挑选可能对解决给定设计问题有用的模式。着手确定粗粒度设计时,可使用架构模式;在整个设计阶段都可使用设计模式;在实现阶段可使用成例模式。

软件架构模式[5]描述了在特定设计情形下反复出现的设计问题,并提供了已得到充分证明的通用解决方案的摘要。解决方案摘要描述模式的组件、组件的职责和关系,以及这些组件协作的方式。架构模式描绘基本的软件系统组织纲要,提供了一组预定义的子系统,指出了这些子系统的职责,包含用于对子系统间关系进行组织的规则和指南。

设计模式[5]提供了对软件系统的子系统、组件或它们之间的关系进行改进的纲要,描绘了对彼此通信的组件进行组织的常见结构,可解决特定背景下的一般性设计问题。

软件架构的子系统及它们之间的关系通常由多个更小的架构单元组成(用设计模式来描述它们);设计模式是中型模式,比架构模式小,但独立于编程语言和编程范式。

3.2.3 实现阶段模式

成例[5]是一种低层模式,针对的是特定的编程语言。成例阐述如何使用给定语言的功能来实现组件或组件间关系的特定方面,方便程序员之间交流,提升软件开发和维护的速度。

与设计模式阐述通用的结构性原则不同,成例描述如何在特定的编程语言中解决具体的实现问题,还可阐述设计模式的具体实现,因此它们之间的界线并不清晰。成例能够直接解决与语言使用相关的低级问题。

在代码实现层次,为任意给定模式提供一个完全通用的实现方式是不可行的。与完全通用的实现相比,定制的、特定应用和领域的模式实现,以及在框架和产品线架构中可见的实现,是更简单、更高效的选择。

3.3 模式详解

无论是单个架构模式还是多个架构模式组合,都不是完备的软件架构,只是给软件系统提供了架构框架,必须进一步规范和细化,包括集成框架与应用程序功能,细化软件系统的组件及它们之间的关系。这些任务可能需要借助设计模式和成例完成。选择一个或多个架构模式只是软件系统架构设计的第一步。

模式并不是孤立的,每种模式一般都与其他模式相关联。GoF 在《设计模式》中用一幅图画出了书中所述的 23 个模式之间的依赖关系,并在介绍每个模式时用"相关模式"一节来详细介绍其依赖关系[12]。通过对"好"的软件架构进行分析得出,在使用模式方面常常倾向于高密度、紧密集成的方式[5]。模式之间互相补充和完善,并以多种多样的方式组合成一个更大的结构。

文献[5,13]认为,有些模式的应用推动了设计,提供了新特性和新的可能性。另一些模式的应用可认为是完善了设计,这种完善不是指把设计推向全新方向并获得全新能力的情况,而是使现有的设计变得圆满,协作原设计达到它在逻辑上的终点。相互竞争的模式提供了一种可替代的选择来激发一场设计对话,而相互合作的模式依靠另一种设计的互补性来完善设计。

3.3.1 模式分类

模式分类有助于挑选可能对解决给定设计问题有用的模式。针对具体的模式,不同著作的分类方式不同,甚至包含的模式种类也不同。

GoF 未将 MVC 作为一种模式[12],但是在模式分解方面达成了类似的结论:Observer 用于通知,Composite 用于嵌套 View,Strategy 模式用于参数化 Controller 的行为,可选的 Factory Method 用于给系统指定一个默认的 Controller 类,还可用 Decorator 来给 View 增加特性。

Frank Buschmann 等将具体的模式分为 14 组[5],帮助用户大致识别模式,为选择模式提供引导。GoF 将模式分为创建型模式、结构型模式和行为模式三大类[12]。

4 辅助工具

4.1 软件架构模式系统

为了帮助开发高品质的软件系统,软件架构模式系统[5]应运而生。模式系统将模式联系起来,描绘了模式是如何相互关联、相互补充的。模式系统必须符合如下规则[5]:包含的模式足够多;以统一的方式描述所有模式;揭示模式之间的各种关系;对模式进行组织;能够帮助打造软件系统;能够不断发展。

为了方便学习和使用,将模式系统中的模式进行分类。从两个维度进行分类,一是模式类别,如架构模式、设计模式、成例;二是问题类别,如从混沌到有序、分布式系统、交互式系统、可适应系统、结构分解、工作组织等。

模式系统为管理数量庞大的模式提供了方便的途经:以统一的方式描述所有模式;通过分类使用户对其中的模式有大致认识;提供合理的查找策略帮助用户选择模式;提供使用模式开发软件系统的指南,充分发挥模式的威力[5]。

4.2 问题框架

为任意模式提供一个完全通用的实现方式是不可行的。因此,对于适用于一个或一类应用模式的实现,在上下文发生变化时可能就不再适用。为了提高设计和代码的可重用性,可以找到这种限制下的可重用技术,如框架、产品线架构和生成工具。

框架通常定义了应用体系的整体结构类和对象的关系等设计参数,以便具体应用实现者能集中精力于应用本身的特定细节。框架主要记录软件应用中共同的设计决策,框架强调设计复用,因此框架设计中必然要使用设计模式。框架实现可以提供绑定了特定模式的机制。

为了方便、正确地理解和使用模式,引入了 Michael Jackson 的问题框架[6]。问题框架的概念绕过"分析与设计合一"的方法进行深入的分析:理解问题域本身,而无需接受编程世界的隐喻和结构。问题框架是对一类问题的普通化,给出了问题的清晰描述以及可能采取的解决办法。

问题框架和模式具有很好的兼容性,两者都是力求具体。问题框架可以为某些关键的模式提供适用的上下文。从分析

模式的角度看待问题框架，只不过在它的上下文中涉及的解决方案较少。问题框架提出了界定问题世界的课题，而不涉及解决方案方面的内容。基于模式的开发方法试图通过理解模式要解决的问题所在的上下文中的驱动力来确定问题的边界[13]。另外，设计模式有助于对框架结构的理解，掌握设计模式是剖析系统的一把利器。

问题框架描述了增强的、详细的意图，抓住了问题领域及其对解决方案的约束的本质，为模式的组合提供了一种方法；界定了一个合适的边界，开发人员可以以此为界专注于设计。

4.3　软件自适应性与自动化生产方式

软件自适应是指为了保证持续、高质量地提供服务，软件在运行时检测环境变化和自身状态，并据此对自身行为进行主动调整的活动。自适应软件的基本周期为“感知－决策－执行”[14]。

基于模式的软件自动化生产方式中软件体系结构模式和设计模式贯穿自适应软件的各个周期，为自适应软件构造提供了技术支持。软件体系结构包含开发阶段的静态实体和运行时的软件体系结构，它至少可以在感知和执行两个阶段为软件自适应提供支持：在感知阶段获取体系结构信息；在执行阶段使得体系结构可动态调整[15]。

针对自适应软件的构造与实现，Ramirez 等总结了包括传感器工厂（sensor factory）、基于用例的推理（case-based reasoning）、服务器重配置（server reconfiguration）等在内的 12 种软件设计模式[16]；Schmidt 等总结出了一系列可用于自适应中间件的设计模式，并在中间件 TAO 和 ZEN 等中得到应用[17]；Gomma 等提出了运行时对软件进行重配置的 Master/Slave、Client/Server、非集中控制等模式[18]。

软件自动化生产方式和软件自适应相辅相成。软件具备自适应性，在上下文等影响环境发生相近变化时，一种类型的应用软件的实现可以比较方便地转换为另一种类型的应用软件。软件自动化生产方式的实现，使软件在自适应方面发展得更远。

5　思考与总结

5.1　疑问与思考

软件自动化生产会遇到 4 方面固有的困难：1）如何清楚、准确地说明软件的需求、设计和功能并便于开发人员理解；2）如何使开发团队中的成员能够对软件具有统一的正确认识；3）如何在需求变更后快速做出指导和应对；4）软件开发过程中抽象出的模型如何清楚地被理解。这些困难在软件环境发生变化时，会使得自动开发更加困难。解决以上困难需要有统一的沟通方式和适应变化的能力。

软件生产是否可以自动化？文献[19]中描述了软件产品和一般实物产品的区别：软件产品在某些方面与传统工程学科中的有形产品存在相似之处，但也存在某些重要的区别，使得软件开发与众不同。由于软件是逻辑概念而非实物，因此其成本集中在开发过程中而不是生产过程中。又因为软件不会磨损，因此其可靠性取决于逻辑质量（如正确性和稳健性）而非物理质量（如硬度和韧性）。

理解清楚生产和开发之间的不同，有利于思考软件自动化生产的实现。当需求确定、生产过程规格统一时，软件可以自动化批量生产。产品自动化批量生产的前提是：需求确定和规格确定。若完全确定，则只需要拷贝。若功能需求确定但是组合不同，则需要修改配置。

5.2　问题的解决过程

恰当地解决某个特定问题的方式如下：首先，需求分析师获取需求并用需求模式描述需求；其次，根据需求模式的描述获取问题，提出可能会影响问题解决方案的驱动力；然后，给出各种可能的解决方案；最后，讨论这些解决方案的利弊。在定义系统时，查询所有需求模式领域决定系统的相关领域，只关注选择领域的模式。

典型的电子商务应用对需求进行分类，处理大量的数据，但是在数据上只进行相对简单的计算，主要是查询功能。对于这类特定领域，模式模型的转换规则是可预定的，因此模式驱动的软件自动化开发方法实现是可行的。对需求模式化，可以将模型转换过程定义为数据表格处理模式、查询处理模式和其他处理模式；并定义每种处理模式的程序模板和实体对象等，可以实现从电子商务应用模型到平台相关模型的转换。

模式引导开发者去思考面对问题的本质、解决问题时需要考虑的问题等。模式起一个指导的作用，所有的活动都在开发人员自己的控制之下。模式能够帮助开发人员去设计推导，但是其设计的合理性还需要验证。模式虽然限制了一些自由，但是不妨碍开发人员做出选择，并且通过明确设计的效果和代价来使这种选择过程更加清晰。

在模式著作里有很多模式是成对的，它们实质是解决相同的问题。如果多个模式都解决相同的问题，该选择哪个？开发人员需要综合更多的信息来获得合理的选择，结合自身对开发软件的经验，理解解决问题所处的上下文和其驱动力。开发人员对信息的获取量和理解，模式的上下文、驱动力、优势和额外成本，都是影响选择的条件。

5.3　分析总结

在基于模式转换的 MDA 的基础上引入软件模式系统，使基于模式的软件自动化开发方法有别于传统基于模式驱动的软件开发方法：模式从需求开始就作用于系统，需求到体系结构的映射自然过渡，特定系统中模式转换可预定，基于模式的软件自动化开发方法中自动化成为可能。

基于模式的转换方法需要先建立一个包含设计模式比较全的模式库。模式系统很好地解决了基于模式转换过程中模式库建立的问题，而且为模式样式的新增提供了指导。

软件架构模式系统的引入为开发人员提供了如何在软件开发中实现、组合和使用这些模式的指南，它包含了一系列的软件架构模式，但是只适合软件架构模式、设计模式及成例等设计阶段的模式。要做到从需求到代码实现的软件自动化生产，还需引入需求模式，与软件架构模式一起组成软件模式系统。

需求模式将需求分成不同的领域，将这些领域的模式纳入模式系统，可以产生特定需求领域的模式系统，涵盖了从分析到使用具体编程语言实现特定的设计的大部分软件开发过程。

引入需求模式之后的模式系统解决了软件自动化生产过程中的统一的沟通方式，而模型驱动架构生产方式解决了软件生产过程中软件环境变化的问题。

模型驱动架构中对模式的使用不仅限于转换过程，在平

台无关模型的建立过程以及平台相关模型的生成过程中都会用到。平台无关模型的建立需要使用形式化方法的无歧义性来描述系统,需求模式的引入方便了形式化的描述,为需求向体系结构的转换做了铺垫。平台相关模型的生成需要结合具体平台的技术,需要使用平台提供的架构。

软件开发从来都是一个需要创造性思维的工作,软件技术的进步从来都是渐进的。软件需求的变动会带来软件体系结构的变动,在这一过程中计算机很难做出合适的选择,需要软件开发人员进行思考决策。基于模式的软件开发方法在需求分析阶段就要进行开发,因此需要特定领域的专家和分析人员进行开发。在设计阶段需要对不同平台、不同系统架构和使用的变换工具的变换定义非常熟悉,因此需要专业开发人员,很难在初级开发人员中普及。软件自动化开发方法会遇到的 4 点固有困难在软件环境发生变化时会使得自动开发更加困难,这些都导致了软件自动化开发方法的研究进展缓慢。

结束语 软件开发和设计过程涉及许多不同种类和不同层次的思维过程。程序员所进行的大量工作都集中在抽象层次较低的软件编码上,他们过于习惯性地依赖于特定的解决方案,没有针对性地检查该方案是否恰当有效。研究软件生产自动化并不是完全的自动化,需要开发人员进行思考决策。基于模式的软件自动化生产方法还有很大的发展空间,现在的模式选择和决策等都依靠程序员的经验和思考,往往会造成很多主观错误。因此,转换决策的制定、模式选择时考虑因素的优先级别等都是以后研究的方向。设计模式阶段需要的通用框架也是以后的研究方向。本文分析了模式驱动的软件架构设计在电子商务应用领域的可行性,进一步的具体实现是以后的研究方向。

参 考 文 献

[1] Model-driven Architecture[EB/OL]. [2017-08-25]. https://en.wikipedia.org/wiki/Model-driven_architecture.

[2] Software factory[EB/OL]. [2017-08-26]. https://en.wikipedia.org/wiki/Software_factory.

[3] 郭新峰,马世龙,吕江花,等. 需求变更自动化管理模型与实现[J]. 计算机系统应用,2015,24(4):11-18.

[4] 刘奎,宋淼,陈一飞,等. 基于软件模式的 PIM 到 PSM 的模型变换[J]. 计算机技术与发展,2006,16(10):74-76.

[5] BUSCHMANN F,MEUNIER R,ROHNERT H,et al. Pattern-Oriented Software Architecture(Volume 1):A System of Patterns [M]. New York:John Wiley & Sons,1996.

[6] MICHAEL J. Problem Frames:Analyzing and Structuring Software Development Problem [M]. Addison-Wesley,2001.

[7] 模式[EB/OL]. [2017-07-12]. http://www.miel68.com/zhuanti/moshi.htm.

[8] 模式[EB/OL].[2014-06-24].http://www.baike.com/wiki/模式.

[9] STEPHEN W. Software Requirement Patterns [M]. Microsoft Press,2014.

[10] FOWLER M. 分析模式[M]. 北京:机械工业出版社,2004.

[11] ALEXANDER C. The Timeless Way of Building [M]. Oxford University Press,1979.

[12] ERICH G,RICHARD H,RALPH J,et al. Design Patterns-Elements of Reusable Object-Oriented Software [M]. Addison-Wesley,1995.

[13] BUSCHMANN F,HENNEY K,SCHMIDT D,et al. Pattern-Oriented Software Architecture(Volume 5):On Patterns and Patterns Languages [M]. New York:John Wiley & Sons,2007.

[14] 丁博,王怀民,史殿习. 构造具备自适应能力的软件[J]. 软件学报,2013,24(9):1981-2000.

[15] KRAMER J,MAGEE J. Self-Managed systems:An architectural challenge[C]// Proceedings of the Conference on the Future of Software Engineering. 2007.

[16] RAMIREZ A J. Design patterns for developing dynamically adaptive systems [M]. Michigan State University,2008.

[17] SCHMIDT D,STAL M,ROHNERT H,et al. Pattern-Oriented Software Architecture(Volume 2):Patterns for Concurrent and Networked Objects[M]. New York:John Wiley & Sons,2001.

[18] GOMAA H,HUSSEIN M. Software reconfiguration patterns for dynamic evolution of software architectures [J]. Fourth Working IEEE/IFIP Conference on Software Architecture,2004 (WICSA 2004). 2004.

[19] WEGNER P. Research Directions In Software Technology[C]// Proceedings of The 3rd International Conference on Software Engineering. 1978.

(上接第 47 页)

[34] SALTON G,WONG A,YANG C S. A vector space model for automatic indexing [J]. Communications of the ACM,1975,18(11):613-620.

[35] 梁栋. 基于深度学习的目标识别研究及其多机器人编队应用[D]. 哈尔滨:哈尔滨工业大学,2015.

[36] LONG M,GAGE A,MURPHY R,et al. Application of the Distribued Field Robot Architecture to a Simulated Deming Task [C]// Proceedings of the International Conference on Robotics and Automation. IEEE,2005:3204-3211.

[37] KOWDIKI K H,BARAI R K,BHATTACHARYA S. Leader-follower Formation Control Using Artificial Potential Functions:a Kinematic Approach[C]// Proceedings of the International Conference on Advances in Engineering,Science and Management. IEEE,2012:500-505.

[38] DU Z,HE L,CHEN Y,et al. Robot Cloud:Bridging the power of robotics and cloud computing[J]. Future Generation Computer Systems,2016,21(4):301-312.

[39] 张小俊,刘欢欢,赵少魁,等. 机器人智能化研究的关键技术与发展展望[J]. 机械设计,2016(8):1-7.

[40] 王光君. 基于云计算的自主心智发育机器人研究[D]. 济南:山东大学,2015.

[41] JI B,LI S,WANG G,et al. Could cloud technology be useful in autonomous mental developmental robotics? A case study[J]. International Journal of Robotics and Automation,2016,31(3):206-444.

[42] 张恒,刘艳丽,刘大勇. 云机器人的研究进展[J]. 计算机应用研究,2014,31(9):2567-2575.