一个可构造的反演状态模式*

何克清[1], 应 时[1], 何 非[2]

[1](武汉大学 软件工程国家重点实验室,湖北 武汉 430072);

[2](国立先端科学技术研究大学,日本)

E-mail: hekeqing@public.wh.hb.cn

摘要: 为了构造可扩充、可配置、可控制的状态模型,提出了一个可构造的反演状态模式.基于体系结构的反演模式和角色对象模式,讨论了状态设计模式中结构和行为的反演方法及其手段;给出了其反演模式体系结构的元级和基本级对象结构;使用MOP(meta object protocol)规程给出了元级和基本级的关系,讨论了反演状态模式的元对象反演到基本级对象群的"截取和求精(interception 和 reification)"机制.此外,还给出了一个应用可构造的反演状态模式的「记账凭证处理型」应用事例.

关键词: 角色模型;状态设计模式;反演模式;角色对象模式;UML(unified modeling language)

中图法分类号: TP311 **文献标识码**: A

近年来,软件模式(software pattern)已成为软件工程研究和应用的一个热点.1995年,Erich Gamma等4人(学界中通称GoF'95)在"Design Patterns,Elements of Reusable Object Oriented Software"[1]的专著中首先提出了设计模式(design patterns)的概念,并给出了23个典型的设计模式.文献[1]在学术界和产业界都产生了很大的影响,极大地促进了近代软件工程学的发展,它获得了"Software Development Productivity Award"大奖,成为近30年来最优秀的软件工程著作.Frank Buschmann等人于1996年在名为"Pattern-Oriented Software Architecture:a System of Patterns"[4]的专著中提出了软件体系结构模式的概念,同时提供了一些可重用的软件体系结构设计经验、方法及模式.UML(unified modeling language)[2]已被国际软件界广泛应用,为软件模式的结构设计和实现提供了有力的手段.

本文基于体系结构的反演模式(reflective pattern)[4]和角色对象模式(role object pattern)[3],讨论了求精和细化状态设计模式的方法和手段,提出了一个可构造的反演状态模式.

对于可以枚举的状态空间及其管理的一般问题,GoF'95在文献[1]中提出了著名的软件状态设计模式.它成功地描述了状态模型的管理、控制和应用的逻辑,在设计中实现了分离状态管理和复杂的状态转换所执行的具体行为.但是,对于如何构造可扩充、可增殖的状态空间、状态转换,如何配置状态模型,文献[1]并没有给出明确的表述,而是完全委托给利用者自己解决.Frank Buschmann在文献[4]中提出了体系结构的反演模式,但是并没有解决状态空间和状态转换的构造、配置和控制的问题.本文讨论了如何增殖式地构造可以扩充的状态空间、状态转换模型的方法;给出了配置和控制状态转换模型的手段,反演状态模式将求精的状态设计模式划分为元级(meta level)和基本级(base level)对象两个部分.本文使用MOP(meta object protocol)规程描述了元级

* **收稿日期**: 2000-09-06; **修改日期**: 2001-04-17

基金项目: 国家教育部科学技术重点规划资助项目(99188);武汉市科学技术规划资助项目(20011001001)

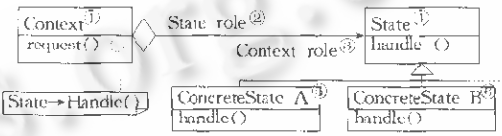
作者简介: **何克清**(1947-),男,湖北武汉人,博士,教授,主要研究领域为软件工程方法论,软件开发环境,软件模式,软组合件,软件框架;**应时**(1965-),男,湖北武汉人,博士,副教授,主要研究领域为软件工程,组件重用;**何非**(1974-),男,湖北武汉人,硕士生,主要研究领域为软件工程方法,软件模式及其形式化.

和基本级的关系,讨论了反演状态模式的元级对象反演到基本级对象模型的"截取和求精"的机制.还给出了一个应用可构造的反演状态模式的「记账凭证处理型」应用事例.

1 GoF'95 的状态设计模式

对于可以枚举的状态空间的管理及其应用的一般问题,GoF'95 提出了著名的软件状态设计模式.如图 1 所示,根据使用者的要求,当 Context 对象的内部状态发生变化时,执行状态转换的对应行为.但是,我们不在 Context 对象的内部给出该行为,而是通过导入状态对象来给出对应的行为.从而达到状态管理和状态转换所执行的具体行为的隔离.

在状态模式中,对象的行为依赖于状态.对象往往具有多个状态及其对应的行为,我们运用抽象的原理,定义多个状态及其对应行为的公共界面,即抽象状态类.作为这个抽象类的子类,我们再具体定义各个不同的状态及其对应行为的 Concrete 状态对象,并且在 Context 抽象类或状态抽象类中定义这些状态转换的规则.



①Context对象,②状态角色,③Context角色,④状态对象,⑤具体状态A,⑥具体状态B.

Fig. 1 State pattern of GoF'95
图1 GoF'95的状态模式

这是一个软件设计模式,其主要目的是描述状态空间的管理和应用的逻辑,分离状态管理和复杂的状态转换所执行的具体行为的设计知识.但是,对于如何构造一个可扩充的、可增殖的状态空间、状态转换模型以及控制和配置状态模型,却没有给出明确的表述,而是委托给利用者自己解决.

2 问题的提出

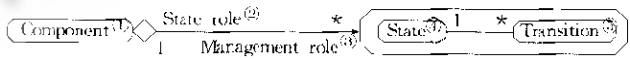
在相当广泛的一类应用问题中,问题的状态空间、状态转换的构造和管理是设计其软件系统的技术关键.而且问题的状态空间往往是经常变化和变更的,需要不断地修改、扩充和增殖.因此,可扩充问题的状态空间的构造、设计和控制是一个十分复杂的问题.

首先,我们分析可以枚举的状态空间及其管理的一般问题,如图 2 所示,给出了一个抽象的状态管理的角色模型[5,6].一个状态组合件角色对象管理着多个状态和多个状态转换角色对象,其中,集约关系(aggregation)描述了多个状态所组成的状态空间是可以配置的,通过状态转换将它们有序地配置起来.状态转换所对应的处理行为可以描述如下:

处理—{行为:{{状态$_1$,…,状态$_n$},

行为() 行为() 行为()

状态$_1$---→状态$_2$,状态$_2$---→状态$_3$,状态$_n$---→状态$_n$}}.



①组合件,②状态角色,③管理角色,④状态,⑤转换,⑥角色对象,⑦使用关系,⑧集约关系.

Fig. 2 The role model of management of state space
图2 状态空间管理的角色模型

但是,对于可以构造和动态扩充的状态空间、状态转换及其配置与管理如何模式化,还是一个十分值得研究的问题.

为了研究这个问题,我们在 GoF'95 状态设计模式的基础上讨论如何求精和扩充,以明确地表述状态模型的构造、动态扩充、配置和控制的新模式.

3 可构造的映射状态模式

我们一方面需要保持GoF'95状态设计模式中状态模型的管理及其应用中状态转换与所执行的具体行为相隔离的思想,同时也希望具有状态模型的可构造、可扩充、可增殖及其配置、控制的开放型特点.为此,我们采用了以下基本思想来讨论可构造的反演状态模式的分析和设计:

(1) 基于Frank Buschmann等人的反演模式,讨论可构造的反演状态模式的开放结构.该模式由两级组成,即模式的元级和基本级.元级描述状态模型中结构配置与行为控制的知识;基本级描述状态模式的应用逻辑,其对象的实现使用元级的对象.

(2) 在元级中,为了构造状态模型,我们引入了Dirk Baumer等人的角色对象模式来构造和动态地扩充状态模型的元对象(meta object)模型,而且将角色对象模式和反演模式结合使用.

(3) 可构造的反演状态模式的基本级为GoF'95状态设计模式.

3.1 反演模式的可构造体系结构

对于诸如可变更、可适应、可扩充的状态空间、状态转换的应用系统来说,软件系统的设计应该采用开放式的体系结构.在开放式体系结构中,首先应定义并设计代表系统的结构、行为、状态等基本特征的语义结构,即元模型(meta model).而且根据系统的环境和条件,由于软件系统自身的变化是可以预测的,因此我们在开放式体系结构中组入可以向用户开放的支持软件自身变化和扩充的机构.当系统的要求发生变化时,软件自身就能够适应所要求的变更和扩充,这是因为我们已经在体系结构上设计了"变更与扩充".

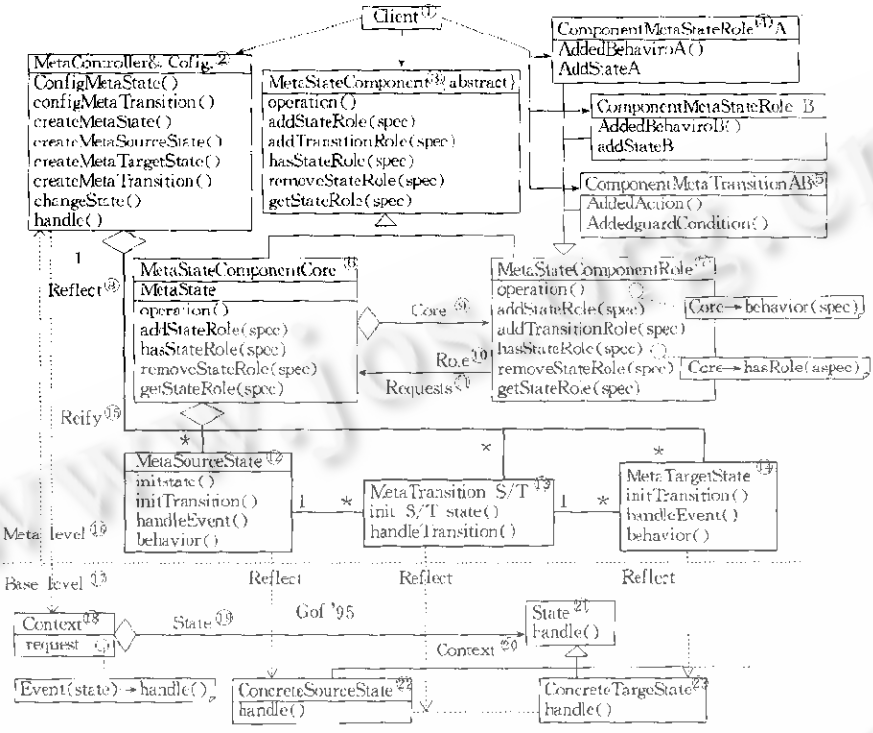
Frank Buschmann等人提出的体系结构的反演模式提供了一个动态地改变软件系统结构和行为的机制.它将应用问题划分为两个级别,即元级和基本级.元级中元对象群是开放的,一个元对象描述一个或多个基本级对象的内部结构和/或行为的信息,能够动态地管理对应的基本级中对象的界面.元对象和基本级对象之间的反演关系,使用一个元对象规程MOP描述.用户使用MOP定义、构造、扩充和变更元对象集.一般而言,在元对象和基本级对象之间,一个MOP应该提供以下的交互式规程:(1) 能够一对一或一对多地、静态或动态地追加元级对象到一个基本级对象群;(2) 提供屏蔽程序的行为、结构、数据、关系信息的手段;(3) 通过"截取(interception)和求精(reification)"机制,将元级的计算界面反演为基本级对象的可执行行为;(4) "变更与扩充"行为和结构的元对象集屏蔽基本级中对象行为和结构的变更信息.

本文应用反演的体系结构,将状态空间及其状态转换问题的开放结构划分为元级和基本级两个部分.我们在元级中组入Dirk Baumer等人的角色对象模式,静态或动态地定义、追加元级状态对象的角色,增量式地构造元级对象模型.也就是说,我们视元级的状态、状态转换对象模型为一个状态组合件,通过不断地定义Concrete组合件角色对象,并按照GoF'95的Decorator设计模式[1],向状态组合件核心即元级的对象模型增加元级状态对象、元级状态转换对象,达到构造元级的状态对象模型的目的.反演模式使得元级对象可以控制与配置基本级对象的状态和状态转换,分离了状态模型的控制与复杂的状态转换所对应的具体行为的变化.对于构造出的元级状态对象模型,反演模式在元级对象和基本级对象之间使用MOP进行配置、控制与反演.

3.2 可构造的反演状态模式

如图3所示,可构造的反演状态模式的开放式结构由元级和基本级构成.在元级对象模型中,我们设计集约元级对象组(源状态、状态转换关系、目的状态)桥接Dirk Baumer角色对象模式和

Frank Buschmann 映射模式来构成.我们还设定可构造的反演状态模式的基本级为 GoF'95 状态设计模式.



①客户,②元控制及配置,③元状态组合件类,④元状态组合件角色,⑤元转换组合件,⑥元状态组合件核心,⑦元状态组合件角色类,⑧反演,⑨核心角色,⑩角色,⑪请求消息,⑫元源状态,⑬元转换S/T,⑭元目标状态,⑮具体化,⑯元级,⑰基本级,⑱Context对象,⑲状态角色,⑳Context角色,㉑状态对象,㉒具体源状态,㉓具体目标状态.

Fig. 3 Constructive pattern for state reflection
图3 可构造的反演状态模式

模式的目的和动机

(1) 在状态模型确定之前,我们需要根据状态转换问题的规范来构造状态空间及其对应的状态转换、状态转换规则.(2) 通过增加、删除等操作动态地扩充状态模型.(3) 在体系结构层,强化状态模型的增殖、变更与维护机制.(4) 对于定义了的状态空间、状态转换及其对应的行为,我们还需要根据问题的要求,配置状态模型.使用时,根据事件要求控制状态转换模型.(5) 隔离用户和状态模型的变更与扩充,给用户一个始终一致的状态模型的操作界面.

模式的结构

如图 3 所示,模式的元级由构造可扩充的状态模型和状态模型的配置、控制及反演两部分组成.为了构造可扩充的状态模型,我们适用角色对象模式,设计由一个抽象的元级状态组合件类、一个元级状态组合件角色(role)类、一个元级状态组合件核心(core)类、两个组合件当前元级状态 A、B 对象和一个当前元级状态转换对象组成的元结构;为了控制和配置状态空间,我们适用反演模式,设计由一个元级控制和配置类以及源状态、状态转换关系、目的状态的元集约对象组成的反演结构及其元级对象反演到基本级对象模型的"拦截(interception)和求精(reification)"的机制.

我们设定元级状态组合件核心类和元级对象控制和配置类都是(源状态、状态转换、目的状态)

元级对象组的集约,在元级中实现以上两部分的连接.

(1) 构造可扩充的状态模型

元级状态组合件类:定义状态空间的所有状态处理的抽象界面.为元级状态组合件角色类提供诸如 adding,removing,testing,querying 等规程.

元级状态组合件核心类:实现元级状态组合件类的界面,包括角色管理规程;建立当前状态角色的实例对象;管理它的角色类等.

元级状态组合件角色类:给元级状态组合件核心类提供一个 decorator 式访问;根据它的核心属性实现元级状态组合件类的界面.元级状态组合件角色类和元级状态组合件核心是抽象的元级状态组合件的子类,它们都继承了父类构造元级状态空间的操作 addStateRole(spec),addTransitionRole(spec),hasStateRole(spec),removeStateRole(spec),getStateRole(spec),operation().

组合件的当前元级状态角色 A,B 和元级状态转换 AB 对象:对于状态组合件类界面,实现一个特定状态 Context 的扩充;根据元级状态组合件核心类的要求,建立当前对象.

利用者使用以上状态角色对象模式,对元级状态组合件核心类增加原级状态 A、B 及其状态转换 AB.即元级状态转换模型以增殖的方式增加{源状态、状态转换(事件名[guard 条件]/活动)、目的状态}组的元级状态模型单位.元级组合件核心类集约多个这样的单位,只要模型的状态空间是可以枚举的,我们就可以使用该模式来构造元级状态转换模型.

对元状态组合件核心类和状态组合件角色类的协调行为进行分析:(1) 元级状态组合件角色类传送请求给它的组合件核心类;(2) 元级状态组合件核心类管理并且实例化状态组合件角色类.

利用者以下列方式在状态组合件核心类和角色类之间进行交互式行为协调:(1) 一个利用者可以用自己所需要的状态及其转换角色来扩充元级状态组合件核心类,直到结束;(2) 利用者请求核心类,要求在一个角色工作时,核心类运行该角色,并返回信息给利用者.

(2) 状态模型的配置、控制及反演

在构筑了状态、状态转换空间之后,我们适用反演模式,并根据问题的规格需求,使用 MetaController&Cofig.类来配置状态转换模型.即配置多个{源状态、状态转换(事件名[guard 条件]/活动)、目的状态}组的相互关联,确定系统的状态转换规则.

MetaController&Config.:(a) 配置元级,实例化,初始化 concret 元级状态和元级转换子类;(b) "截取"所有的消息转送给 concret 对象;(c) 控制到 concret 状态元级对象的访问,并委托它处理被"拦截"的消息;(d) 实现状态转换,通过一个元级状态转换对象改变当前元级状态对象.为此,我们给出 MetaController&Cofig.类的如下操作:ConfigMetaState(),configMetaTransition(),createMetaState(),createMetaSourceState(),createMetaTargetState(),createMeta Transition(),changeState(),handle().

MetaState 类:(a) 定义一个处理一个事件的界面,该事件描述一个依赖于状态的服务;(b) 定义一个初始化基本级中状态对象的界面;(c) 定义一个初始化自己本身的方法.

MetaTransition 类:(a) 定义一个处理转换的界面;(b) 定义初始化自己本身的方法.

一旦元级对象模型配置完毕之后,用户再也没有必要直接去处理这些元级状态类和状态转换.元级对象和基本级对象之间的反演关系使用元对象规程 MOP 加以描述.

3.3 可构造的反演状态模式与 GoF'95 的状态设计模式

我们认为,GoF'95 状态设计模式可以作为可构造的反演状态模式的基本级模型(base level model).如图 3 的下部所示,我们使用元级对象规程 MOP 作为 Controller&Cofig.类对象的一个反

演,对应于 GoF'95 状态设计模式的 Context 类;作为 SourceState 和 TargetState 类对象的一个反演,对应于 GoF'95 状态设计模式的 ConcreteSourceState 和 ConcreteTargetState 类.作为 Transition S/T 类对象的一个反演,可以隐含地对应于 GoF'95 状态设计模式的状态转换关系.对于已经构造和配置完毕的状态转换模型,GoF'95 状态设计模式的 Context 类接受利用者的状态服务事件,根据系统的状态转换规则执行相应的(源状态、状态转换(事件名[guard 条件]/活动)、目的状态)组.

4 可构造的反演状态模式在「记账凭证处理型」问题中的应用

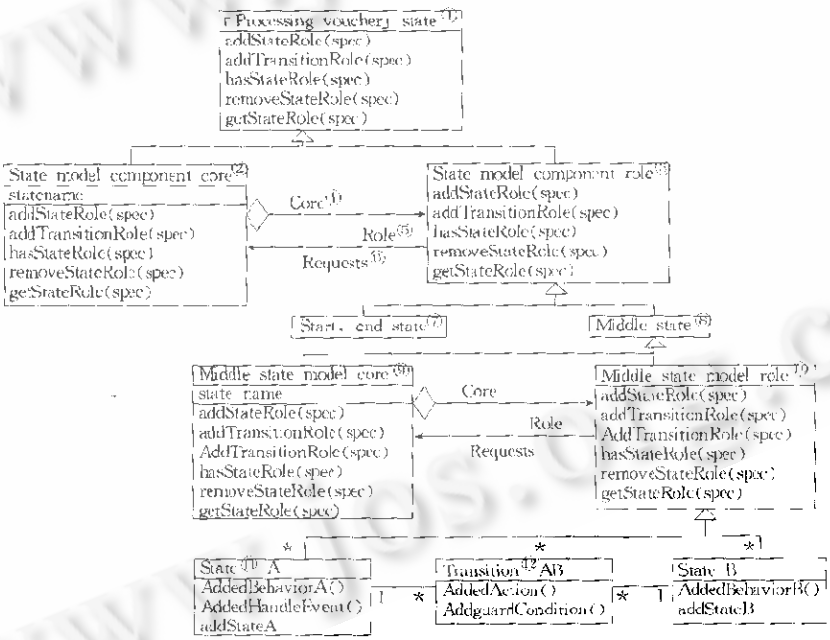
我们给出了如图 4 所示的「记账凭证处理型」的一个应用例子.问题具有如下的特点:

(1) 不论对于什么样的「记账凭证处理型」问题,其初始状态、最终状态都是存在的.

(2) 不同的应用问题可以存在不同的中间状态空间及其状态转换所对应的行为.

(3) 状态空间及其状态转换的行为规范构成一个「记账凭证处理型」业务组合件.

(4) 利用者可以根据自己的需要,构造状态空间及其转换.



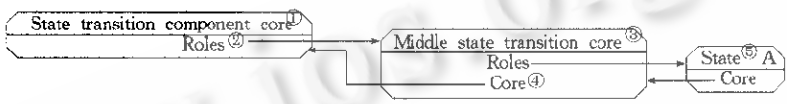
①「记账凭证处理型」状态模型组合件,②状态模型组合件核心,③状态模型组合件角色,④核心,⑤角色,⑥请求消息,⑦初始、最终状态,⑧中间状态,⑨中间状态模型核心,⑩中间状态模型角色,⑪状态,⑫转换.

Fig. 4 Meta level object model for processing voucher
图4 「记账凭证处理型」元级对象模型

因此,我们应用可构造的反演状态模式,设计了如图 4 所示的「记账凭证处理型」元级对象模型.根据「记账凭证处理型」问题的特点(1)和(2),我们将「记账凭证处理型」状态模型组合件角色的子类设计为初始、最终状态角色类和中间状态角色类.进而,我们对于「记账凭证处理型」的中间状态角色类再一次应用可构造的反演状态模式,设计了中间状态模型核心类和中间状态模型角色类,它们是开放式体系结构中的"变更与扩充"机构.对于中间状态模型角色类,我们还设计了状态 A、状态 B、状态转换 AB 这 3 种角色子类.

我们应用「记账凭证处理型」可构造的反演状态模式,具体实现了如下"销售记账凭证付款"处理的状态转换模型:初始状态、销售款登录()、销售款状况检查状态;销售款状况检查状态、请求()、请求状态;销售款状况检查状态、隔月支付()、再请求状态;销售款状况检查状态、抵消预付金()、进款状态;请求状态、超过预定现金的支付日期()、再请求状态;请求状态、现金支付()、进款状态;请求状态、委托自己进账()、委托自己转账状态;委托自己转账状态、不能进账()、再请求状态;委托自己转账状态、进账确认()、进款状态;

初始状态、预先支付()、预支付状态;预支付状态、抵消预付金()、进款状态;进款状态、履行契约()、最终状态.如图5所示,在运行时,该模式引导一个角色对象和核心对象的动态执行链.中间状态模型中的角色对象状态A的角色动态地追加给中间状态模型核心对象,进而动态地追加给"销售记账凭证付款"的状态模型组合件核心对象.



①状态转换组合件核心,②角色,③中间状态转换核心,④核心,⑤状态.

Fig. 5 Object relationship for adding objects dynamically

图5 动态地增加角色的对象关联示意图

5 小结和今后的工作

本文基于体系结构的反演模式和角色对象模式,对求精和细化地扩充状态设计模式进行了讨论,提出了一个可构造的反演状态模式.

由于状态设计模式旨在描述状态模型的管理和控制逻辑及其使用,分离状态管理和复杂的状态转换所执行的具体行为的设计,而如何构造可扩充、可增殖的状态空间、状态转换,如何配置状态模型,则并没有给出明确的表述,而是完全委托给利用者自己来解决.本文基于角色对象模式给出了如何增量式地构造可以扩充的状态空间、状态转换模型的方法和手段;基于体系结构的反演模式明确地给出了如何配置和控制状态转换模型的手段.反演状态模式包括元级和基本级两个部分,使用MOP规程描述了元级和基本级的关系.本文讨论了反演状态模式的元对象反演到基本级对象模型的"截取和求精"的机制,还给出了可构造的反演状态模式在「记账凭证处理型」问题中的应用事例,叙述了构造和配置「记账凭证处理型」状态模型的方法及其状态转换模型,为状态设计模式在「记账凭证处理型」问题逻辑中的应用提供了一个求精和细化的事例.

作为今后的工作,可构造的反演状态模式的元对象反演到基本级对象的"截取和求精"的机制的实现、行为细化与状态-事件-活动的关系、状态空间浏览(state space explorer)模式等课题还有待于进一步的研究.

References:

[1] Gamma, E., Helm, R., Johson, R., *et al*. Design Patterns, Elements of Reusable Object-Oriented Software. Tokyo: Soft-Bank Publishers, 2000. 325~333.

[2] OMG. Unified Modeling Language V. 1.3 Specification (draft). USA, 1999. http://www.omg.org.

[3] Baumer, D., Riehle, D., Siberski, W., *et al*. The role object pattern. In: Proceedings of the PLoP'97. 1997. 87~96. http://www.ubs.com/abilab.

[4] Buschmann, F., Meunier, R., Rohnert, H., *et al*. Pattern-Oriented Software Architecture: a System of Patterns. Tokyo: Toppan Publishers, 1999. 189~215.

[5] He, Ke-qing, Jiang, Hong, He, Fei, *et al*. Extended UML with role modeling. Journal of Wuhan University (Natural Science), 2001,6(1-2):175~182.

[6] Xu, Yong song, He, Ke qing, Ying, Shi. Role modeling method for software pattern at knowledge level. Journal of Wuhan University (Natural Science), 2001,6(1-2):200~203.

A Constructive Pattern for State Reflection

HE Ke-qing[1], YING Shi[1], HE Fei[2]

[1](*State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China*);

[2](*National Advanced Institute of Science and Technology, Japan*)

E-mail: hekeqing@public.wh.hb.cn

Abstract: In order to construct extendable, configurable and controllable state model, a constructive pattern for state reflection is proposed in this paper. Based on the reflective pattern of software architecture and role object pattern, the reflection approach for structure and behavior in the pattern of state design is discussed. At meta level and base level, the object structures for architecture of reflective pattern are designed. The relationship between the meta level and the base level is presented with the protocol of MOP (meta object protocol). The mechanism of interception and reification for object reflections from the meta level to the base level is discussed in the pattern of state reflection. Finally, an example of voucher using the pattern of state reflection is given in this paper.

Key words: role model; pattern of state design; reflective pattern; pattern of role object; UML (unified modeling language)

* Received September 6, 2000; accepted April 17, 2001

Supported by the Key Sci-Tech Project of the Ministry of Education of China under Grant No. 99183l; the Sci-Tech Project of Wuhan Municipal Government under Grant No. 20011001001