第 34 卷 第 4 期 计 算 机 工 程 2008 年 2 月
Vol.34 No.4 Computer Engineering February 2008
·软件技术与数据库· 文章编号：1000—3428(2008)04—0107—04 文献标识码：A 中图分类号：N945

MVC体系架构从模式到框架的持续抽象进化

刘 宁，陆荣国，缪万胜
(中国航空无线电电子研究所第五研究室，上海 200233)

摘 要：针对有互动响应要求的复杂软件系统的设计和开发，提出一种“(组合)模型-视图-控制器”软件体系架构框架的设计方案。该方案基于进化式软件架构设计理念体系，运用以持续抽象策略为导引的进化式迭代方法，对传统的模型-视图-控制器体系架构模式进行了进化式抽象迭代，并在SAL Studio 1.0 项目中予以应用。结果证明，由该方法得出的设计方案提高了相应软件体系的可靠性、可扩展性和可复用性，促进了软件体系的技术进化，能优质、有效、可靠地完成软件开发。
关键词：进化式软件架构设计理念体系；架构模式；架构框架；持续抽象

Continuous Software Abstracting Evolution from MVC Architecture Pattern to MVC Architecture Framework

LIU Ning, LU Rong-guo, MIAO Wan-sheng
(The 5th Branch, China National Aeronautical Radio Electronics Research Institute, Shanghai 200233)

【Abstract】To design and develop the complicated software system requiring interactive response, a new multi-dispatch-based behave-type-binding and knowledge-decoupled architecture framework pattern is educed. It is based on the methodology idea system and leaded by continuously-abstracting strategy. An evolutionary iteration process is implemented on the traditional MVC architecture pattern. Integration Signals and Logic Tool SAL Studio 1.0 are implemented to show how to implement the new evolutionary design methodology idea system in a real project. The results demonstrate that the new solution is more reliable, flexible, dynamic and reusable for the software design and development.
【Key words】evolutionary software architecture design idea system; architecture pattern; architecture framework; continuously-abstracting

软件设计方法学的一个经典例子是架构模式模型-视图-控制器模式(Model-view-controller, MVC)[1-3]，本文基于进化式软件架构设计理念体系[4]，通过对MVC架构模式应用持续抽象策略(抽象对于软件设计的重要性可以参考文献[5])，设计和实现了新的基于多派遣行为类型绑定和知识解耦的MVC架构框架，给出了以这种进化式方法学理念体系领导软件进化的实践情况。

1 MVC 架构模式的历史起源

MVC架构模式由文献[6]首次提出，在Smalltalk-80 环境下实现[2]，为交互式软件系统的设计提供了一种基础。著名的基于MVC架构模式的系统有MacApp[7]、ET++[8]、Spring(http://www.springframework.org)、微软的基础类库和Struts(http://struts.apache.org/)。

2 从 MVC 架构模式到 MVC 架构框架的进化

2.1 命名简述的进化

(1)MVC 架构模式

MVC架构模式将交互式应用程序分为 3 个组件：包含核心功能和数据的模型、显示数据的视图、处理输入的控制器。视图和控制器共同构成用户接口，通过变更传播机制确保用户接口和模型的一致性[9]。

(2)MVC 架构框架

MVC 架构框架建设一种抽象的，可被多态实现的 MVC 三元系统架构元素的协作关系。它也包含模型、视图、控制器三元素，但三元素是抽象的，只描述参与协作和组合的抽象特征，其领域特征被延后到实现决定。系统组合和协作将通过“框架”式的控制反转由 MVC 架构框架接管。模型只是一种关于属性和服务的抽象代理，这些属性和服务有待在框架在现实软件系统中被多态地实现为子系统的时候予以具体实现。同时，模型抽象元素也在实现构造认为有必要的情况下，负责参与实现反转控制的协作和组合；视图元素是一种系统边界接口抽象代理，负责事件采集和状态表示，必要时参与实现反转控制的协作；控制器则是由视图通向模型的抽象类型化行为多派遣机制的控制者。

2.2 MVC 架构的协作机制的进化

广义MVC三元素的协作机制有多种类型，已知的有简单的非组合式的三元协作机制[9]，针对可组合式MVC元素的通过控制器继承层次结构连接不同模型视图对的协作机制[5]，以及本文提出的针对可组合的抽象MVC的基于多重派遣的行为类型绑定和消息解耦的协作机制。不同的协作机制定义不同的MVC架构模式/框架的实现构造。

2.3 应用环境的进化

(1)MVC 架构模式具有灵活人机接口的交互式非行为类型敏感组合的应用程序。

(2)MVC 架构框架具有多态系统内外互动机制的行为类型敏感的组合式体系。

2.4 解决方案的进化

(1)MVC架构模式[9]是非行为类型敏感组合的，对MVC各元素协作逻辑无类型控制，有较大的实现自由，但也有较

作者简介：刘 宁(1967－)，男，高级工程师、硕士，主研方向：面向对象软件体系，软件进化，设计模式；陆荣国，研究员；缪万胜，高级工程师
收稿日期：2007-04-15 **E-mail：**stormking@yeah.net

大的模糊性和不稳定性，不利于确保系统的健壮性[9]。

(2)MVC架构框架是基于多派遣行为类型接管对MVC三元素协作和组合的控制，引入基于观察者模式的信使类型，将协作知识从3个骨干元素中剥离，使协作知识的抽象和多态扩展成为可能，解耦协作的实现知识，增强设计的可靠性和可理解性。

MVC框架对MVC模式进行能力抽象，归纳出4种形成框架的抽象元素和一种刻画架构框架多态性的构造机制：

1)抽象模型。抽象模型是核心行为和状态[10]的抽象代表，所以不在框架层实现领域核心行为和状态。由于领域的状态和行为是可分类和可组合的，因此可识别出一种组合抽象，而抽象模型也将通过多派遣行为类型敏感的组合模式和观察者模式实施抽象组合机制。组合机制使框架能刻画更复杂的系统。这其实是一种模型组合-视图-控制器系统。抽象的视图元素是不可组合的，系统组合将完全由模型的组合关系决定，以此降低系统复杂性。待实现的领域行为和维护领域状态的机制将被推迟到最终实现系统实现。抽象模型只处理协作与组合。实现构造机制将贯穿在抽象模型的组合中。

2)抽象视图。抽象视图表示抽象模型，其具体实现不在框架层完成，而被推迟到最终实现系统。抽象视图在框架层的目的是参与框架协作。和MVC模式不同，在框架层视图被禁止组合。

3)抽象控制器。抽象控制器响应来自系统边界的输入，具体实现不在框架层完成，被推迟到最终实现系统。抽象控制器在框架层的存在是为了参与框架协作。和MVC模式不同，控制器在框架层被禁止组合。

4)抽象信使。抽象信使元素用于行为类型绑定的合法类型甄别和行为派遣，类型实现推迟到最终实现系统实施，以提供扩展空间。

由于MVC架构框架的多态扩展建立在多派遣行为类型绑定和知识解耦机制的基础之上，框架的协作机制由实现构造定义，抽象模型元素表达系统的本质行为和状态，因此组合抽象将由模型元素负责构造。抽象视图状态变更由抽象模型发起；抽象模型状态变更由抽象控制器发起；抽象控制器只维护相关抽象模型列表，不维护其他抽象状态，其行为由来自抽象视图的事件进行基于行为类型的多派遣触发和传递。行为类型由信使(Envoy)定义，并通过信使类型识别来定义元素间行为派遣。通过限制部分行为的重载，框架因此能够对各元素间与框架总体有关的协作和组合进行全面的接管，而且这种接管和各元素的具体实现知识是无关的，从而使可靠的多态实现成为可能，同时也确保了核心抽象所控制的组合和协作机制的同构和可靠，加强了系统的可理解性，扩展了框架运行时的动态重用能力。

2.5 结构的进化

(1)MVC模式[9]

由图1~图4可见，MVC模式各架构元素间存在双向关联关系和依赖关系，设计没有充分实现抽象和解耦，增加了最终实现系统的复杂性和不可理解性，降低了可扩展性。

类	协作者
模型	视图
责任	
提供应用程序功能核心 注册视图和控制器 公布有关数据变更的相关组件	

图1 模型元素CRC卡

类	协作者
视图	控制器 模型
责任	
创建和初始化相联的控制器 对用户展示信息 实现更新过程 以模型恢复数据	

图2 视图元素CRC卡

类	协作者
控制器	视图 模型
责任	
接受用户输入作为一个事件 将事件翻译成对模型的服务请求，或对视图的展示请求 如果需要的话，实现更新过程	

图3 控制器元素CRC卡

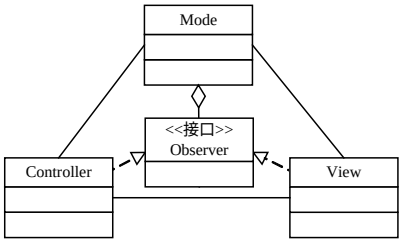


图4 MVC模式类结构

(2)MVC架构框架

由图5~图9可见，抽象后的设计消除了框架架构元素间的双向强类型关系。其中，图9(1)是抽象视图元素与抽象控制器元素之间行为类型敏感的多派遣机制的实现结构；图9(2)是抽象控制器元素与抽象模型元素之间行为类型敏感的多派遣机制的实现结构；图9(3)是抽象模型元素与抽象视图元素之间行为类型敏感的多派遣机制的实现结构；图9(4)是封装于骨干抽象模型元素内的抽象模型元素与抽象子模型元素之间行为类型敏感的多派遣机制的实现结构。

抽象类	协作者
抽象模型	抽象信使 抽象视图 作为被组合元素的抽象模型
责任	
维护通过行为类型绑定的抽象模型元素组合 维护通过行为类型绑定的抽象视图元素集 确立基于行为类型绑定的信使传递机制	

图5 抽象模型元素的CRC卡

抽象类	协作者
抽象视图	抽象控制器 抽象信使
责任	
维护通过行为类型绑定的抽象控制器元素集 确立基于行为类型绑定的信使传递机制	

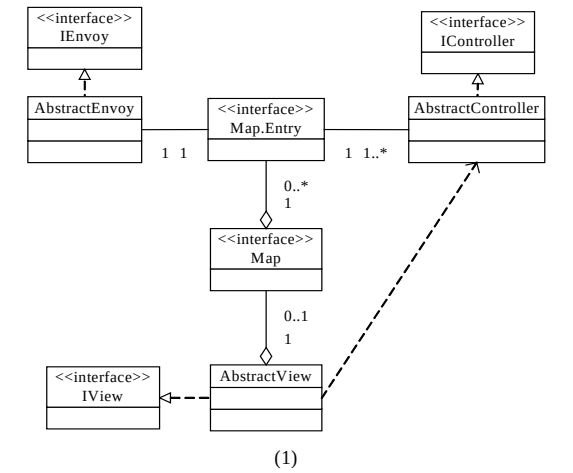
图6 抽象视图元素的CRC卡

抽象类	协作者
抽象控制器	抽象模型 抽象信使
责任	
维护通过行为类型绑定抽象模型元素(模型组合被封装在参与抽象控制器互动的抽象模型元素内，保持框架架构层次的结构简单性) 确立基于行为类型绑定的信使传递机制	

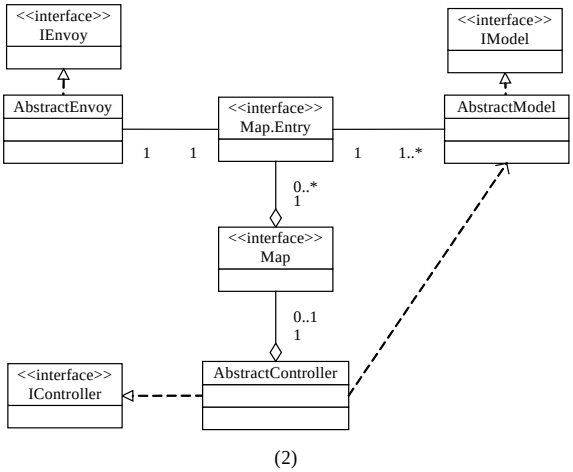
图7 抽象控制器元素的CRC卡

类	协作者
抽象信使	无
责任	
构造行为类型绑定的多派遣实现构造机制	

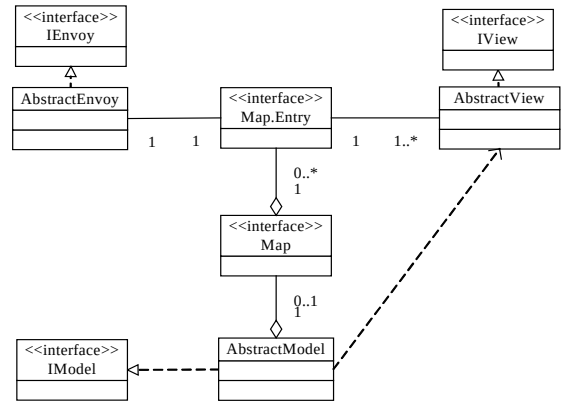
图8 抽象信使元素的CRC卡



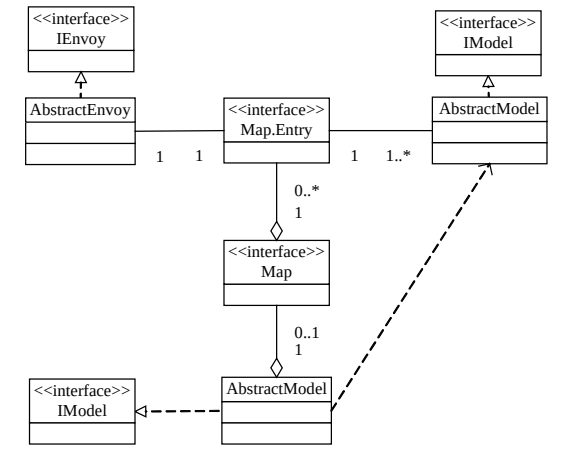
(1)



(2)



(3)



(4)

图 9　MVC 架构框架静态结构

MVC 架构框架解耦了对于相互知识的双向依赖，使元素间完成协作所需的知识依赖变成单向的：抽象模型只需要抽象信使、抽象子模型和抽象视图的抽象知识(框架本身的运作与最终实现系统的知识无关)，抽象模型及子模型只须了解抽象信使和抽象视图的抽象知识，而抽象视图只须了解抽象信使和抽象控制器的抽象知识，抽象控制器只须了解抽象信使和抽象模型的抽象知识(由于抽象模型组合机制被封装在抽象模型内，因此抽象控制器并不需要了解组合信息)。

进化后的架构框架达成 2 个层次的知识解耦：

(1)MVC 各元素间不存在聚合关联之类的强关系，模型的自组合也无须了解具体模型知识就能完成多派遣协作，只存在较弱的调用依赖关系，而且这种关系是单向依赖的，使各协作元素间能够知识解耦。

(2)架构框架抽象无须了解任何最终系统的实现知识即能自治运行。核心系统虽然对多派遣行为类型敏感，但核心系统本身无须了解具体行为类型的实现知识即可运作。

2.6　动态特性的进化

(1)MVC 架构模式

MVC 架构模式的动态协作序列图(图 10)是一个强耦合架构，这使系统缺乏可扩展性和灵活性，即系统设计未达成良好的抽象。

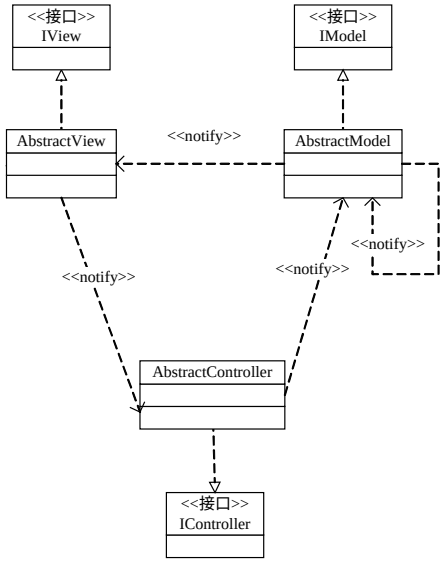


图 10　MVC 架构框架变更传递依赖性

(2)MVC 架构框架

图 11 例示了 MVC 框架的抽象多派遣运行时的动态协作过程。由图 11 可见，和 MVC 模式三元素交叉协作时需要维护强关联关系不同，在 MVC 框架的抽象协作层次中，MVC 的 3 个抽象元素间只有一种针对抽象元素的 notify 调用依赖关系，这种依赖关系又通过 AbstractEnvoy 行为类型实现了多重派遣以及多态解耦。这种解耦结构不是无类型解耦，而是依循“实现构造”的建设性行为类型敏感解耦。

2.7　实现的进化

MVC 模式实现请参考文献[9]，MVC 框架已被应用于中国航空无线电电子研究所 SAL STUDIO1.0 项目中，通过运用新的体系架构框架，有效控制了系统的复杂性，适应了一系列客户需求变更，成功地完成了软件开发(原项目长期没有开发进展，运用新设计方案后，在 1 年之内有效可靠地完成了开发工作，通过验收，获得嘉奖)。

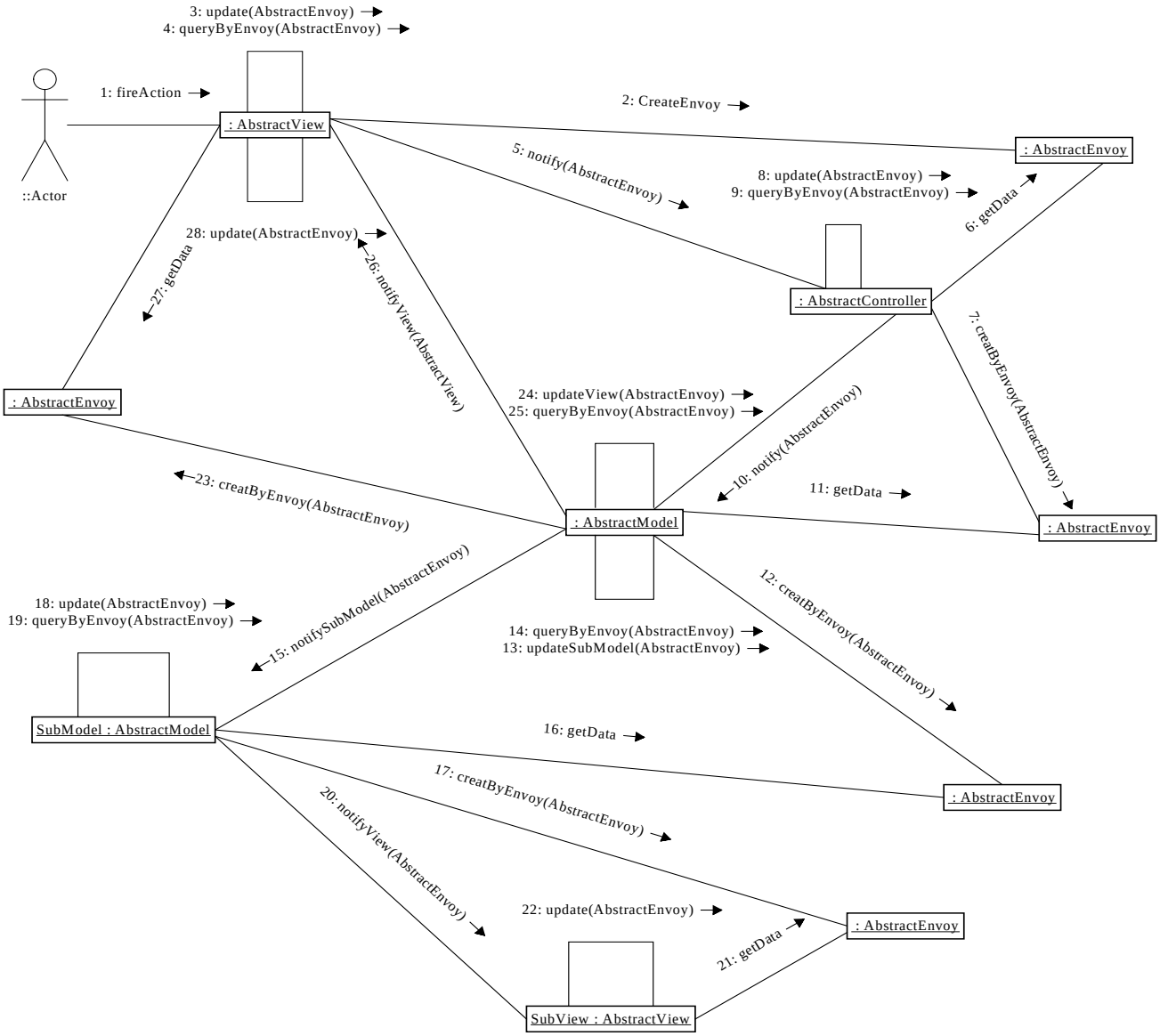


图 11　MVC 架构框架运行时的抽象多派遣动态协作

3　结束语

本文应用进化式软件架构设计理念体系[1]，对MVC架构模式实施以持续抽象策略为导引的进化式技术迭代，开发了一种基于多派遣行为类型绑定和知识解耦的MVC架构框架，通过在实际开发工作中运用新的体系架构框架，有效控制了系统的复杂性，适应了一系列客户需求变更，成功地完成了软件开发，验证了文献[9]的方法学理念体系对改进软件开发的效率和促成可进化软件系统的成功进化的作用。

参考文献

[1] Gamma E, Helm R, Johnson R, et al. Design Patterns: Elements of Reusable Object-oriented Software[M]. [S. l.]: Addison Wesley, 1994.

[2] Krasner G E, Pope S T. A Cookbook for Using the Model-view Controller User Interface Paradigm in Smalltalk-80[J]. Journal of Object-oriented Programming, 1988, 1(3): 26-49.

[3] Althammer E, Pree W. Design and Implementation of a MVC-based Architecture for Ecommerce Applications[Z]. [2006-08-05]. http://citeseer.ist.psu.edu/443079.html.

[4] 刘　宁, 陆荣国, 缪万胜. 进化式软件架构设计理念体系[R]. 中国航空无线电电子研究所, 2006-07.

[5] Gamma E, Helm R, Vlissides J. Design Patterns: Abstraction and Reuse of Object-oriented Design[C]//Proceedings of ECOOP'93. Kaiserslautern, Germany: [s. n.], 1993.

[6] Reenskaug T, Wold P, Lehne O A. Working with Objects: The OOram Software Engineering Method[M]. [S. l.]: Manning Publications Company, 1996.

[7] Macitosh Programmer Workshop Pascal 3.0 Reference[M]. Cupertino, CA: Apple Computer Inc., 1989.

[8] Gamma E. Object-oriented Software-Entwicklung am Beispiel von ET++: Design-Muster, Klassenbibliothek, Werkzeuge[M]. Berlin: Springer-Verlag, 1992.

[9] Buschmann F, Meunier R, Rohnert H, et al. Pattern-oriented Software Architect: System of Patterns[M]. [S. l.]: John Wiley & Sons Ltd., 1996.

[10] Doug L. Concurrent Programming in Java: Design Principles and Patterns[M]. 2nd Ed. [S. l.]: Addison-Wesley, 1999.