基于UML的面向对象的图形用户界面设计模型*)

孙晓平 郭腾冲 魏明珠 涂序彦
(北京科技大学信息工程学院 北京100083)

A UML-Based Object-Oriented Graphic User Interface Design Model

SUN Xiao-Ping GUO Teng-Chong WEI Ming-Zhu TU Xu-Yan
(Information Inst., Univ. of Sci. &Tech Beijing, Beijing, 100083)

Abstract GUI development is often large, complex and difficult. But there are few methods and tools to describe GUI requirement specifications, GUI layouts and GUI tasks in software design. This article discusses GUI modeling and introduces a UML-based Object-Oriented GUI model composed of Frame Controller, View Model and Core Interface(FVI mode), which supports object-oriented requirement specification and provides a layered, modularized and iterative Object—Oriented GUI design model in terms of GUI Layouts and GUI dynamic interaction tasks. Through an instance of the model, we demonstrate that utilizing UML to implement Object-Oriented FVI mode incorporates GUI design into software process, which improves the integrity and consistence of the software design.

Keywords Object-oriented GUI design model, UML, Object-oriented design, FVI model

1. 前言

图形用户界面(GUI)最早由Macintosh操作系统所采用,后被各种操作系统如Unix,Window所借鉴.目前GUI已经形成了一系列标准的图形用户界面模式.图形用户界面开发的工作量相当大,它占到了软件设计实现工作量的40—50%[1].同时,GUI的设计与实现又存在各种困难[2].在设计时难以真正理解用户需求;用户界面层次结构复杂,缺乏有效的描述方法;界面的任务复杂,缺乏清晰的设计工具;对面向对象的GUI设计实现支持不够;界面设计开发涉及多个领域,如文档的编写,美工设计,标准化,国际化和性能优化等[3].GUI在实现时也面临设计文档的不明确,实现的复杂性,程序健壮性难以保证,难于测试和维护等诸多问题.

事实上,在很多软件的设计阶段,由于缺乏行之有效的用户界面设计手段,界面设计由实现人员直接编码完成,从而导致了实现与用户需求之间的差距.为解决这一问题,研究人员先后尝试了各种方法和手段,如基于模型的用户界面定义,基于形式化或半形式化方法的用户界面设计和自动生成等.本文分析了用户界面建模,提出一种基于Frame Controller—View Model—Core Interface(FVI)构架的面向对象的GUI设计模型,可以对用户界面的需求定义,用户界面的层次结构以及用户界面的交互任务进行划分和描述,它是一个自顶向下不断迭代的设计模型,可以对复杂的GUI设计进行有效的组织和设计.本文结合一个CAD软件的设计讨论了利用统一建模语言UML实现面向对象的FVI框架模型的设计.通过将UML与GUI模型结合起来,为GUI设计的各个阶段提供了有效的文档化、可视化设计语言,将GUI设计与系统软件整体设计紧密地结合起来,提高了系统的可测试性、可维护性和可扩展性.

2. GUI模型

GUI设计和实现是软件系统开发中的一个重要的部分.用户界面是指软件系统与使用者之间的交互.它为用户提供各种形式的输入,将用户的输入信息进行转换后,传给核心模块进行处理,并将处理结果以可理解的方式反馈给用户.它介于用户和核心应用之间.设计既要针对使用者,又要适应核心模块.GUI建模技术为设计者提供了一个折衷的方式,力图从各个层面去满足使用者和核心模块的需求.

GUI建模与核心应用建模既是相互独立的,又有着紧密的耦合关系.通常GUI模型抽象为三个部分:界面的表现模型,即与使用者间的接口;界面构件的对话过程,即用户界面构件之间的交互以完成用户任务;核心应用,即完成应用业务逻辑的功能模块.几种主要的GUI模型如,Seeheim模型[4],MVC(Model-View-Controller)模型和PAC(Presentation - Abstraction - Controller)[5]都基于这样的基本思想.下面对最基本的Seeheim模型进行简要说明.

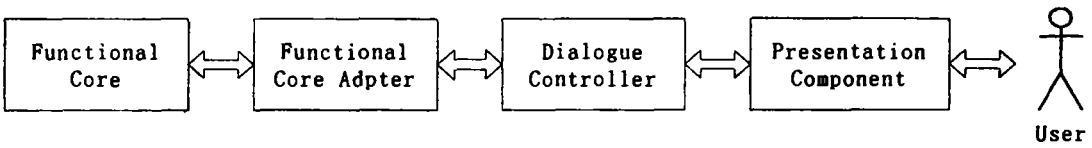

图1 Seeheim模型

*)本文获得国家863-511-944-019高技术计划项目支持.孙晓平 硕士研究生,研究方向:人工智能、Agent、软件工程.郭腾冲 硕士研究生,研究方向:人工智能、数据挖掘、多媒体技术.魏明珠 硕士研究生,研究方向:人工智能、人工生命.涂序彦 博士生导师,主要从事人工智能、大系统控制论、智能管理的研究.

2.1 Seeheim 模型

Seeheim 模型将软件体系结构分为4个部分：核心模块(Functional Core)，核心应用接口(Functional Core Adapter)，对话控制器(Dialogue Controller)，界面构件(Presentation Component)。

Functional Core 对领域应用进行建模。Functional Core Adapter 为用户界面与核心应用之间建立一个缓冲区，以减少二者之间的耦合。它通过一些交互协议为用户界面与核心应用之间提供同步或者异步的数据交换。Dialogue Controller 是 Seeheim 模型中的核心部分。它通过界面构件接收来自用户的各种输入请求，通过转换后利用核心应用接口与核心模块进行数据交换，保证多个视图间的一致性，以完成特定的用户任务。在 Dialogue Controller 中可以嵌套定义 Seeheim 子模型。这样可以从不同粒度上对 GUI 系统进行建模。Presentation Component 对界面构件的具体交互动作和输入输出进行设计。

Seeheim 模型中的核心是 Dialogue Controller。可视化部分由 Presentation Component 来描述，它只负责接受用户的输入。而用户任务很多是直接针对用户界面构件本身的，在 Dialogue Controller 中控制界面的显示行为使系统的设计不够清晰。

2.2 Frame Controller－View Model－Core Interface 模型

本文在 Seeheim 的基础上，结合大系统控制论中的多级分散控制建模思想[6]，提出一种 FVI 模型(Frame Controller－View Model－Core Interface Model)。FVI 是一种以视图(View Model)为核心，包括框架控制器(Frame Controller)和核心应用接口(Core Interface)的面向对象的 GUI 设计模型。视图(View)对用户界面的可视部分进行描述。Frame Controller 通过对各个视图进行调度来完成由用户发起的特定的任务。Core Interface 为视图部分与核心应用模块提供信息交互的接口。

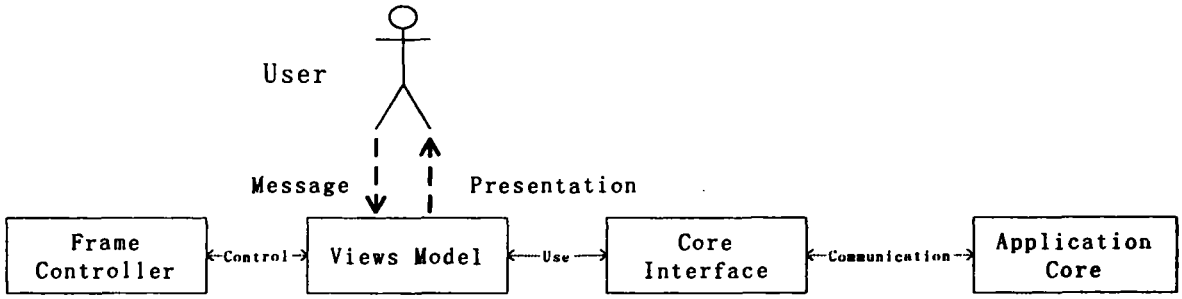

图2 Frame Controller Views Mode Core Interface 模型

视图(View)是用户界面布局中能够完成某一类功能的区域。它接受用户的输入，并为用户提供可视化信息。它可以是一个编辑窗口，一个对话框或一个 Web 页面。采用多级递阶的设计思想，View Model 从逻辑功能上将 GUI 分解为各个视图(View)，每个视图又可以分解为多个子视图(Sub-View)。子视图是对上一层视图的进一步分解和细化。在物理实现时，子视图并不一定是父视图的一部分。它的静态特性可以包含视图的大小、位置和可见性等与视图自身表现形式有关的属性。视图的动态行为包括视图内部的动作和与其他视图间的协作。视图是 FVI 模型设计中的核心，不同层次的视图在不同粒度上完成的用户功能中的某一部分或全部。利用 View Model 可以对复杂的界面进行模块化、层次化的、面相对象的描述。

Views 是 GUI 模型中唯一直接与用户打交道的部分。View 的建模以消息响应为核心。它通过消息响应过程对用户事件(Event)进行处理，如调用核心应用或改变界面形式等。当 View 与其他视图进行交互时，将用户消息发送给 Frame Controller 进行调度。View Model 是用户界面设计中的一个抽象模型，它并不具体指明各个视图的表现形式，从而提高了设计的通用性和灵活性。View 在与核心业务进行交互时，通过调用 Core Interface 中的方法将数据传给 Application Core，或从 Application Core 中获取数据。

当用户完成一项任务涉及到若干视图时，Frame Controller 负责各个视图间的切换调度。它接受从 Views 发送来的消息，由消息响应函数负责对相关的视图进行控制。相对于 Views Model，Frame Controller 是一种粗粒度的用户交互模型。它负责将用户要完成的任务分解，然后映射到用户界面的各个视图中去。视图内部的用户交互动作则由 View Model 去描述。Frame Controller 在系统中是非可视化的，它只负责协调各个视图间的调度。在 Frame Controller 中可以定义子控制器(Sub－Frame)，它嵌套在上层的 Frame Controller 中，为相应层次的视图提供服务。Frame Controller 的设计也是一个逐层细化，分级递阶的过程。

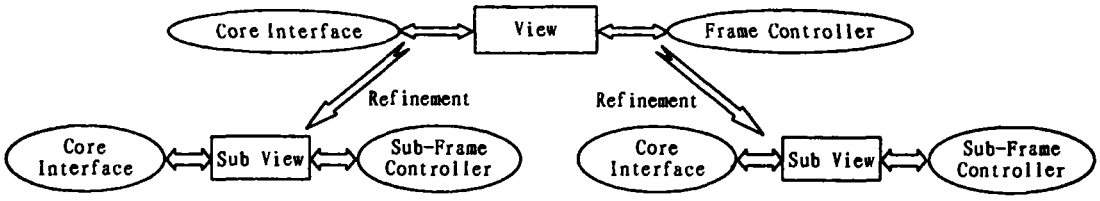

图3 FVI 模型层次结构

Core Interface 为用户界面中的视图(Views)提供了与核心应用进行数据交互的接口。尽管以往的各种设计模型都试图将用户界面与核心应用分离开来，以减少两者之间的耦合。但由于现代操作系统和程序设计语言中大量地使用了回调函

数(Callback function),导致在用户界面与核心应用的实现相互交叉包含.Core Interface 在用户界面与核心应用之间建立一个缓冲区,减少了界面与核心应用的耦合,提高了代码的可维护性和可重用性.

统一建模语言 UML 是近几年业界形成的面向对象的通用建模设计语言,它结合了几种面相对象设计方法,形成了一套从需求分析到详细设计的标记语言,目前已经成为 OOA 设计领域事实上的标准[9,10].利用 UML 实现 FVI 模型,可以将 GUI 设计与核心业务的设计融合起来[11,12].FVI 模型可以从 UML 的 Use Case 导出.View Model 的动态行为可以用 UML 的状态转移图、顺序图和活动图描述.Frame Controller 可以使用协作图,状态转移图,顺序图和活动图.Core Interface 可以使用 UML 的静态视图来描述.

UML 语言提供了模型管理视图,用以描述系统各种模型之间的关系.通过模型管理视图提供的机制,系统设计者可以将各个模型元素有机地分解为各个不同层次的包,从而从不同的层次粒度上对系统模型间的关系进行描述,极大地提高了系统设计的可读性和可维护性.UML 这种层次化、模块化的管理机制非常适合于描述逐层递阶的 FVI 模型.

UML 为系统设计提供了以用例为中心的、不同层次、不同角度的建模工具,并为各种模型提供了关联交互的机制,从而可以对系统进行一个统一的、系统的描述和定义,使各种建模手段有机地结合起来,提高了系统设计阶段的效率和质量,也为实现人员理解设计思路提供了统一的交互语言.我们以一个应用实例来说明使用 UML 实现 FVI 模型.

3. 应用实例

本文中的应用实例是一个 CAD 软件.它提供几种绘制简单图形的工具,使用户可以在电子图纸上进行绘图,包括绘制直线,圆,矩形和书写文字.用户可以用鼠标拖动图形元素来改变尺寸、位置和形状,并能删除和添加图形元素.根据上述需求,我们首先得出系统的用例图.

通过图4,可以很清晰地将系统所要实现的基本功能描述出来,从而描述界面以及界面与业务逻辑的关系,通过用例图可以进一步对系统的功能模型和用户界面模型进行定义和描述.图5则为系统基本的类图.

在实例中,Application Core 的功能包括存储所创建的图形信息,如位置、大小、颜色等,并提供与各种图形相关的核心

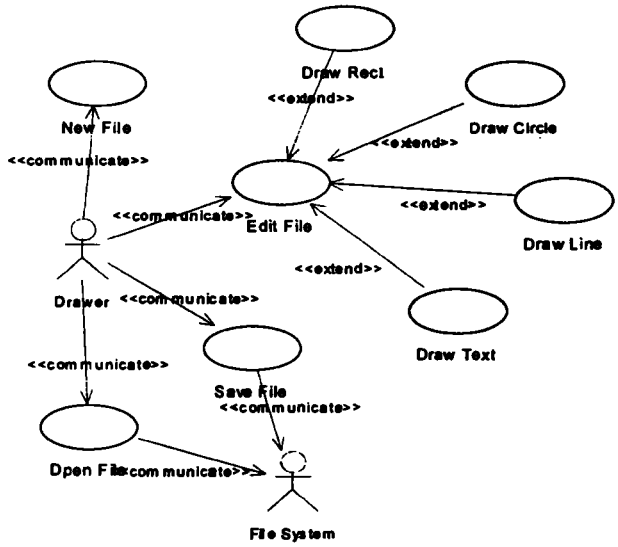

图4 CAD 软件的用例图

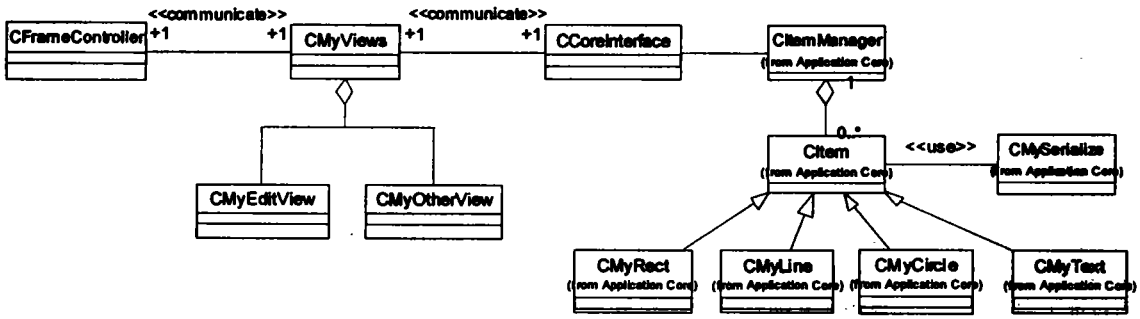

图5 CAD 软件静态类图

算法,如位置检测、覆盖检测等.设计中将几种图形元素抽象为类,提供一个管理类,对一个图形文件中各个图形元素进行存取、删除和遍历机制.CMySerialize 负责对磁盘文件,数据库和网络的存取.

根据 FVI 模型,GUI 进一步划分为三个模块,即 View Model、Frame Controller 和 Core Interface(图6).Frame Controller 控制用户界面布局,各个视图(Views)之间的切换.View Model 对用户界面中的各个视图(View)进行抽象描述.Core Interface 为 View Model 提供与核心应用交互的接口.

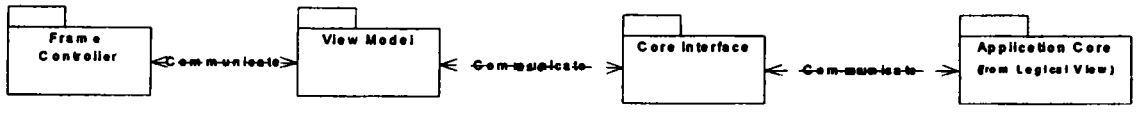

图6 CAD 软件设计的 FVI 模型结构设计

通过用例视图可以进一步对 View Model 进行分解细化为5个视图,分别对应用例图中的各个用户任务.Command View 用来说明与用户命令有关的界面.Edit View 包含4个子视图 Draw Line View,Draw Rect View,Draw Line View 和 Draw Text View.每个视图完成一个特定的用户界面任务.

如 Open File View,它说明了 Open File View 所要提供

给用户的信息，以及用户选择要打开文件的路径，在 Open File View 定义中定义了与类图，顺序图以描述其静态特性和内部行为。图6为用户打开一个文件时的动作描述，它包含在 Open File View 的 Package 中，Open File View 获取用户输入的文件名，通过 Core Interface 传给核心应用，核心应用成功打开文件后返回，Open File View 改变自身的显示状态。然后通知 Frame Controller，由 Frame Controller 控制下面的工作。在具体实现时可以采用 Windows 操作系统提供的 File Open Dialog，也可以采用自己实现的方式，这里并未做出具体说明。Open File View 可以是由其他界面组件激活的，如点击菜单和按钮条上的按钮，这些动作在 Command View 中进行描述。

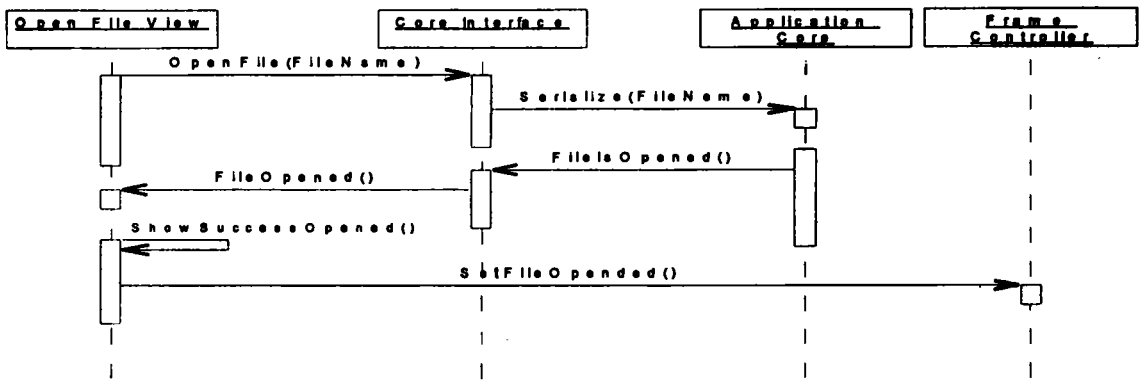

图7 View，Core Interface 与 FrameController 的交互

Frame Controller 对用户在各个视图间的切换调度进行描述。它接收来自各个 View 的事件，Frame Controller 在系统中是非可视化的，它只负责协调各个视图间的调用。如 Open File View 在打开文件成功后，Edit View 需要将文件内容显示，Command View 需要提供新的菜单和工具栏。这一过程可以用顺序图进行描述(图8)。在 Frame Controller 中只描述各个 View 之间的调用，不对 View 内部的动作进行描述，如 Edit View 对 LoadFile()的响应将在 Edit View 中进行描述。

Edit View 是 CAD 软件的核心界面。用户在 Edit View 中对图形文件进行创建和修改。我们采用层次化的方法将 Edit View 分解为4个子视图。Edit View 负责将图形文件载入显示，并将图形元素的信息传给核心应用，实现其他通用的图形操作，如移动图形元素。四个子视图分别负责四种图形元素的编辑和修改。这样划分可以使软件结构更加清晰，利用面向对象的实现，也便于扩展。如需要增加绘制曲线的功能时，只需要增加一个负责绘制曲线的子视图即可。

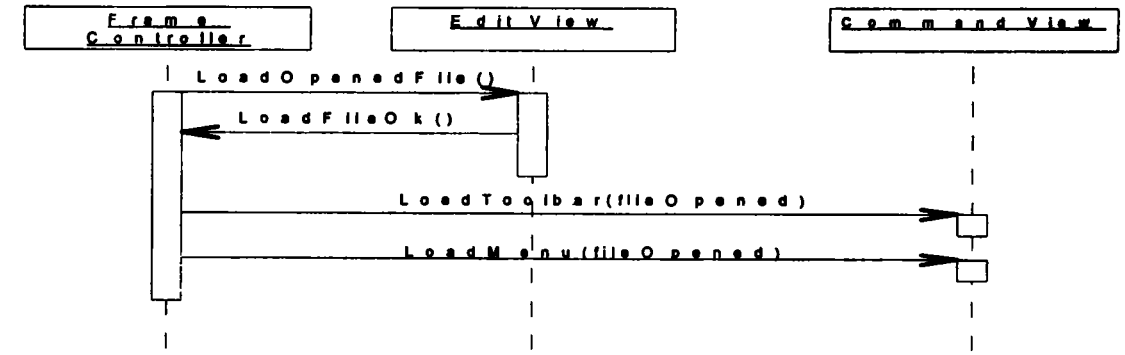

图8 Frame Controller 调度设计

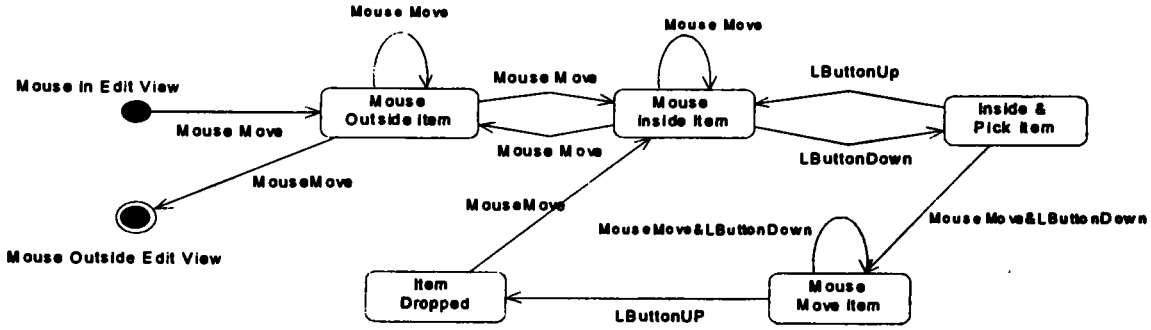

图9 视图内部交互设计

在具体实现时，这四个视图实际上可以在一个窗口中实现，Frame Controller 中也将增加一个子控制器负责对四个子视图进行调度。Edit View 中的各种内部动作可以用状态转移图(图9)进行描述。如在 Edit View 中定义移动图形元素的

过程，当用户在某个图形元素上移动鼠标，并且按住鼠标左键时，图形元素随着鼠标的移动而移动。

UML 对面向对象设计的表述能力相当强大，可以较好地描述 FVI 模型的各个层面。FVI 的层次化和模块化结构可以较好地与 UML 各个粒度的设计工具相结合，较好地融入到整个软件的设计过程中。通过 UML 用例图可以很清晰地导出 View Model，Frame Controller 以及 Core Interface。利用 UML 的包 Package 对模型进行分解细化。

当然，在使用 UML 进行 GUI 设计时，也存在着一些局限和问题。比如，UML 用例图对用户界面需求的描述不够清晰和明确，这导致在由用例图导出界面模型时容易产生偏差。用例图与其他静态视图和动态视图的对应关系没有明确描述。UML 对并发过程控制的约束关系的描述能力较弱，在使用 UML 描述用 View 和 Frame Controller 内部的动态行为时，存在着设计者的隐含预定和假设。实际上，这些问题都是由于 GUI 设计的特殊性所造成的，为此众多学者也提出了各种方法和手段，如 Essential User Case 设计方法[7]，使用 Petri 网或形式化语言描述界面动态行为等[8]。但在实际应用中，UML 被认识是最有效、最直接的设计建模语言。

结束语 FVI 模型以视图 View 为设计核心，采用逐层递阶的思想，反映了在用户界面设计中自顶向下的设计思路，为 GUI 界面的需求分析、界面结构化和层次化设计，以及界面动态交互任务设计提供了逐步细化的面向对象的设计框架模型。FVI 模型将 GUI 设计的中心放在了 View Model 上，将 GUI 逐步分解为层次化和模块化的子视图，可以从各个层次和角度去描述 GUI 界面的层次、结构和动态交互。相应地把用户任务看作是由多个视图的内部行为的组合，给予可视化组件更多的自主权，适合于面向对象的系统设计和实现。

在实际应用中，使用 UML 设计 GUI 模型，为设计人员提供了有效的 GUI 设计语言，可以较好地弥合界面设计与核心应用之间的分歧，使设计人员可以使用统一的设计语言和工具对软件进行整体规划和设计，从而提高了软件设计的质量和效率，减少了设计人员对需求的理解偏差、设计人员之间的理解偏差，以及设计人员与实现人员之间的理解偏差，有利于软件体系结构的扩充和维护，从而提高软件开发的质量、健壮性、可测试性、可维护性和可扩展性。

GUI 设计中还包括大量的图形界面交互行为的设计，如图形控件交互设计、对话框交互设计等等。限于篇幅有限，本文未作详细讨论。实际应用中利用 UML 语言可以较好地描述界面构件的动态交互行为。

通过实际应用，我们感觉到使用 UML 实现面向对象的 FVI 设计模型，明确了 GUI 设计在软件体系结构中的位置，为设计人员提供了 GUI 设计的依据和方法，提高了软件设计文档的完整性和一致性。尽管使用 UML 描述 GUI 交互行为存在一定的局限性，但考虑到 UML 语言的特点以及 GUI 设计与核心应用设计的统一，UML 仍是最有效的 GUI 模型描述工具。对于 UML 在 GUI 设计上的局限，可以通过对 UML 扩展的方式加以弥补，以不断完善对 GUI 设计的支持。

参考文献

1 Myers B A, Hudson, Pausch. Past, Present, and Future of User Interface Software Tools. ACM Transactions on Computer-Human Interaction, 2000, 7(1): 3～28
2 Myers B A. Why are Human-Computer Interfaces Difficult to Design and Implement?: [Research Reports, CMU-CS-93-183]. Computer Science Department, Carnegie Mellon University, 1993
3 Bomsdorf B, Szwillus G. Tool Support for Task-Based User Interface Design. A CHI'99 Workshop, 1999
4 Coutaz J. Software Architecture Modeling for User Interfaces, in Encyclopedia on Soft-ware Engineering, Wiley, 1993
5 Hussey A, Carrington D. Comparing Two User-Interface Architecture: MVC and PAC. TECHNICAL REPORT NO. 95-33, Software Verification Research Centre, Dept. of Computer Science, The University of Queensland
6 涂序彦. 大系统控制论. 国防工业出版社, 2000
7 Constantine L L. Essential Modeling: Use Cases for User Interfaces. ACM Interactions, 1995, 2(2): 34～46
8 Elkoutbi M, Keller R K. User Interface Prototyping based on UML Scenarios and High-level Petri Nets. Application and Theory of Petri Nets 2000 (Proc. of 21st Intl. Conf. on ATPN), Aarhus, Denmark, Springer. LNCS, 2000. 166～186
9 Rumbaugh J, Jackbson I, Booch G. The Unified Modeling Language Reference Manual. Addison-Wesley, MA, 1999
10 OMG. OMG Unified Modeling Language Specification. in Web Site, No. Version 1.3, Object Management Group, 1999. (URL http://www.omg.org)
11 da Silva P P, Paton N W. UMLi: The Unified Modeling Language for Interactive Applications. UML2000 - The Unified Modeling Language. Advancing the Standard. Third Intl. Conf. York, October 2000, Proceedings
12 Silva P P da, Paton N W. User Interface Modeling with UML. The 10th European-Japanese Conference on Information Modeling and Knowledge Representation, May 2000
13 Coutaz J. PAC, an Object Oriented Model for Dialog Desgin. In: H. I. Rullinger, R. Shacked, eds. Human-Computer Interaction-INTERACT'87, Elsevier Science Publishes, 1987
14 Griffiths T, McKirdy J, Forrester G, et al. Exploiting model-based techniques for user interfaces to database. In: Proc. of VDB-4 Italy, May 1998. 21～46

（上接第107页）

和使用，并从根本上实现了可扩展操作系统的高效性和柔性。

参考文献

1 Anderson T. The Case for Application-Specific Operating Systems. In: Third Workshop on Workstation Operating Systems, 1992. 92～94
2 Banerji A, Tracey J M, Cohn D L. Protected Shared Libraries -- A New Approach to Modularity and Sharing. In: Proc. of the 1997 USENIX Technical Conf. 1997
3 Bershad B N, et al. Extensibility, safety and performance in the SPIN operating system. In: Proc. of th Fifteenth ACM Symposium on Operating Systems Principles, Dec. 1995
4 riceno H M. UNIX Abstractions in the Exokernel Architecture. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 1997
5 Candea G M, Jones M B. Vassal: Loadable Scheduler Support for MultiPolicy Scheduling. In: Proc. of the 2nd USENIX Windows NT Symposium, to appear, 1998
6 Hildebrand D. An architectural overview of QNX. In: Proc. of the Usenix Workshop on Micro-Kernels and Other Kernel Architectures, April 1992
7 Deutsch P, Grant C A. A Flexible Measurement Tool For Software Systems. Information Processing 71
8 Engler D R, Kaashoek M F, O'Toole J. Jr. Exokernel: an operating system architecture for application-Specific resource management. In: Proc. of the Fifteenth ACM Symposium on Operating Systems Principles, Dec. 1995
9 Kaashoek M, Engler D, Ganger G, et al. Application performance and flexibility on exokernel systems. In: Proc. of the Sixteenth ACM Symposium on Operating Systems Principles, 1997. 52～65