一种基于模型驱动的笔式界面开发框架研究

陈明炫[1,2],邓昌智[1],任 磊[3],田 丰[1],戴国忠[1]
(1.中国科学院软件研究所人机交互技术与智能信息处理实验室,北京 100190;
2.中国科学院研究生院,北京 100049; 3.北京航空航天大学自动化科学与电气工程学院,北京 100191)

摘 要: 本文针对笔式用户界面开发的需求个性化、平台多样化和开发难度大的问题,提出了一种基于模型驱动的笔式界面开发框架.首先,提出了一种笔式用户界面开发架构,对笔式用户界面的特点进行抽象与描述;然后基于模型驱动开发方法提出了笔式界面的平台无关模型 PIPIM 以及笔式界面的平台相关模型 PIPSM,并建立了 PIPIM 与 PIPSM 之间的转换方法;最后,给出了一个基于模型驱动的笔式界面开发工具 Iris,并使用 Iris 开发了一个笔式个人信息管理系统.应用实例表明:该基于模型驱动的笔式界面开发框架能够有效支持笔式界面开发、降低开发复杂性.

关键词: 模型驱动; 笔式界面; 软件开发方法; 人机交互
中图分类号: TP391 **文献标识码:** A **文章编号:** 0372-2112 (2011) 02-0268-07

Research on a Model Driven Development Framework for Pen-Based User Interface

CHEN Ming-xuan[1,2], DENG Chang-zhi[1], REN Lei[3], TIAN Feng[1], DAI Guo-zhong[1]
(1. *Intelligence Engineering Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing* 100190, *China*;
2. *Graduate University of Chinese Academy of Sciences, Beijing* 100049, *China*;
3. *School of Automation Science and Electrical Engineering, Beihang University, Beijing* 100191, *China*)

Abstract: In order to make the pen-based user interface easy to develop with personalized requirements and diverse devices, we propose a model driven development framework. For a pen-based user interface, firstly we present a general development framework, then build a platform independent model and a platform specific model based on the model driven architecture. Furthermore, we introduce the transformation way from the former model to the latter one. At last, a toolkit named "Iris" is provided to support the development. The example application built by "Iris" shows that the model driven development framework can benefit the development of a pen-based interface and reduce the complexity of the development efficiently.

Key words: model driven; pen-based user interface; software development method; human-computer interaction

1 引言

随着计算环境与交互设备的发展,笔式用户界面以其自然的交互特征和便携轻巧的物理特性广泛应用于台式机、平板电脑和智能手机等多种终端,面向多种以自然交互为中心的应用,例如制定体育训练计划、制作和播放教育课件以及个性化个人信息管理等,笔式用户界面迅速成为新一代用户界面研究的热点[1],然而软件开发者在开发笔式用户界面时,面临众多困难和挑战.第一,笔式用户界面面向大众用户和广泛应用,开发者希望为用户定制个性化的界面和服务;第二,笔式用户界面适合多种平台,在平台变更时,开发者希望把已有软件在新平台上快速实现;第三,笔式用户界面本身在界面和交互的特殊性也增加了开发的复杂度.原有的软件开发方法,在需求和平台变更时,都要更改设计和代码,难以保证设计文档和最终代码的一致性,难以对原有设计有效复用,在开发单一平台下的笔式用户界面也具有一定的难度.如何能方便地定制个性化的笔式用户界面,如何能有效提高笔式用户界面在平台间迁移的效率,如何能降低笔式用户界面的开发复杂度,这些问题成了笔式用户界面开发的关键问题.当前 OMG 提出的模型驱动软件开发框架(Model Driven Architecture, MDA)对降低软件开发的复杂度和解决软件复用问题提供了一个很好的方法和思路[2~4].

收稿日期:2009-12-02;修回日期:2010-04-04
基金项目:国家 973 重点基础研究发展规划(No.2009CB320804);国家自然科学基金(No.U0735004);国家 863 高技术研究发展计划(No.2009AA01Z337, No.2008AA01Z303)

本文针对笔式用户界面需求的个性化、平台多样化以及开发难度大的问题,提出一种笔式用户界面架构,在此基础上结合 MDA 开发思想,建立了模型驱动的笔式用户界面的开发框架,定义了笔式界面的平台无关模型、相关模型以及平台无关模型到相关模型的转换方法,给出了笔式界面开发的整个流程,并实现了基于此方法的笔式界面开发工具.

2　相关工作

笔式用户界面是 Post-WIMP 界面的一种主要界面形式,它采用的基于纸笔的交互隐喻赋予用户极大的发挥空间,现实操作中的自由勾画、手势会意这些特点都被自然地借鉴过来,从而实现了自然高效的交互方式[5],使其逐渐能够应用在人们日常生活中的信息捕捉、信息交流、设计制造以及艺术表现等不同的领域.

但是笔式用户界面所具有的一些特征决定了笔式用户界面开发的复杂性.首先,从用户角度看,笔式用户界面具有很强的个性化特点,由于笔式用户界面应用于不同的大众人群,他们具有不同的知识文化背景、不同的交互习惯和偏好,因此对笔式界面的呈现、界面的交互技术有着不同的要求;其次,从软件功能角度看,笔式用户界面应用的众多领域,对软件功能有着不同的需求,例如在教学、体育训练和办公领域的应用体现了笔式用户界面具有功能需求的多样性[5];再次,从软件适应性角度看,笔式交互设备和使用环境日渐多样化,计算能力、显示尺寸、系统平台差异等因素对笔式用户界面提出了跨平台、跨系统等新的要求,这对笔式用户界面开发提出了快速适应性的要求;最后,从界面本身特征角度看,笔式交互界面具有不同于一般 WIMP 界面的的众多特征[5],这使得软件的界面和交互设计有着比 WIMP 界面更多的考虑因素,增加了软件开发的复杂性.因此,需要一种面向笔式用户界面的软件开发方法,它既能满足界面的用户个性化、功能多样化的要求,又具有快速适应能力和能反映界面特征的完整描述能力.

在当前的软件开发方法中,专门针对笔式界面系统的开发方法比较少[6].传统的结构化开发方法虽然能够建立满足需求的系统,然而其前提是需求的相对稳定,当需求变动时,则需要花费大量的成本重新设计和开发,而且因为用户无法参与其中,增加了开发的风险;快速原型法虽然能够让用户在开发过程中积极参与,快速反映用户对界面和功能的需求,然而面对复杂的笔式交互设计时,无法做到快速开发;面向对象开发方法以对象建模为基础,通过建立对象模型,能真正反映用户的需求,然而无法满足个性化定制和跨平台等带来的效率要求.OMG 提出的 MDA 方法可以很好的满足个性定制、平台迁移的要求,保证了用户和设计者在平台无关层次合作对系统的设计,并提高了软件的开发效率.当前有多种基于 MDA 思想的开发方法和工具,然而他们或者偏重于描述系统的内部业务流程,如 UML 语言,或者面向特定的领域,例如基于 Web 的三层/多层体系结构软件的开发[7,8],或者集中在 MDA 对 WIMP 界面模型构建和模型之间的映射关系上[9~12],很少关注在 Post-WIMP 界面上的设计开发[13],在交互方面都无法描述交互的多样性、无法迅速构建个性化的界面对象,对于轻量级的笔式界面,开发周期还是相当长[13].

本文提出了一种笔式用户界面开发框架,在此基础上结合模型驱动开发的思想,提出模型驱动的笔式界面开发框架,给出了建立平台无关模型和平台相关模型以及平台无关模型到相关模型转换的方法,并实现了一个基于模型驱动的笔式界面开发工具,能够有效满足笔式用户界面对用户个性化和功能多样化的要求,能完整描述笔式界面的交互方法,又因为平台无关模型保留了系统的抽象描述,在跨越平台时只需要模型间的转换就可以快速进行平台间迁移,有效提高了笔式用户界面的开发效率.

3　一种笔式用户界面架构

笔式用户界面的开发侧重于界面与交互设计,从输入角度看,Ink 作为原始输入被转化为交互原语,进一步被解释为笔迹或具有语义的笔手势,对于笔迹的理解和笔手势的识别都涉及到众多的算法.从输出角度看,界面需要渲染多样化的界面对象,需要呈现丰富的多媒体内容.为指导开发者理解笔式用户界面,降低开发的复杂度,同时使笔式界面具有很好的可扩展性,我们提出了一种笔式用户界面开发框架,如图 1 所示.该框架将笔式用户界面分为三个层次:核心层、平台层

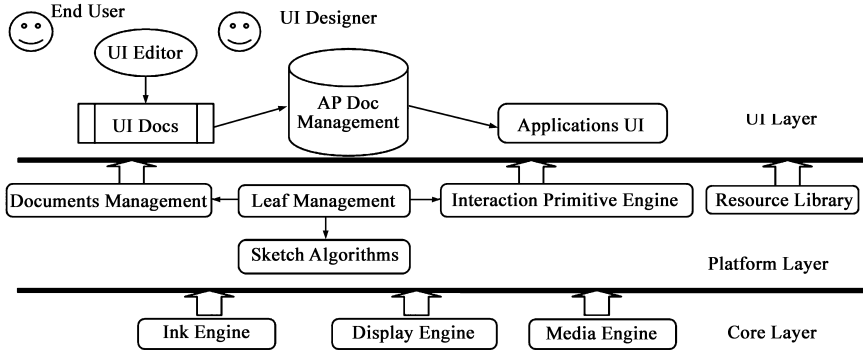


图1　一种笔式用户界面架构

和界面层.

核心层主要负责包括 ink 引擎、display 引擎、media 引擎;ink 引擎封装了来自用户的笔或鼠标事件,生成笔原语,典型的笔原语[4]包括:点击、长击、划线以及长击划线.display 引擎包含各类渲染工具例如画笔刷子以及渲染对象例如矩形椭圆等,负责对界面元素进行渲染;media 引擎对多媒体对象进行解码,例如图片、音频、视频、动画等.核心层封装了输入和输出的核心算法和功能.

平台层负责为上层应用提供相应服务,包括文档结构管理、叶节点管理、算法库、交互原语引擎以及外部资源库.文档结构管理模块管理笔式用户界面中常用的文档结构,例如卡片结构、页结构等;叶节点管理模块负责管理笔式用户界面的常用框,例如文本框、笔迹框、算术框、几何框等;算法库封装了若干笔迹算法,例如笔迹识别、切分、理解等算法;交互原语引擎通过核心层的 ink 引擎得到笔原语,根据上下文得到用户正在操作的交互对象或应用对象,将原语与对象封装为具有语义信息的动宾短语形式的交互原语;外部资源库负责维护开发过程中用到的图片、动画、音频、视频等资源.

用户界面层用来实现和管理不同个性化的界面,设计者和用户一起参与设计,根据不同用户的要求通过设计工具完成应用程序的界面和交互设计,形成设计模型并进行统一管理.设计模型可通过编码或者模型转换的方法产生应用程序界面.

由上可知,该框架将笔式用户界面的各功能单元提取出来,形成模块并组织分层,即按依赖关系对模块由底向上进行了划分,完整地描述了笔式用户界面的特点,可以用来指导部分笔式界面的开发,降低了开发的复杂性.

4 模型驱动的笔式界面开发

为了方便笔式界面的个性化定制和快速的平台间迁移,进一步降低软件系统开发的复杂性,提高开发的效率,我们结合 MDA 思想,设计了一种基于模型驱动的笔式用户界面开发框架.

MDA 开发方法将软件系统的模型分为平台无关模型(Platform Independent Model,PIM)和平台相关模型(Platform specific Model,PSM),其中平台无关模型 PIM 是对工作流程的高层次的抽象,其中不包括与实现技术相关的信息;平台相关模型 PSM 是跟特定平台相关的模型.PIM 和 PSM 通过转换规则将它们统一起来,以这样的方式来解决软件需求变更带来的问题;同时,它将业务层和实现层分离,能够很好地做到业务模型的重用,降低不同实现环境的工作量,缩短开发周期.图 2 表达了 MDA 的开发过程[12].

由于 MDA 开发方法并未给出具体的抽象模型和具体模型的类别和建立方法.基于上述 MDA 的思想,结合笔式用户界面开发框架,首先建立笔式用户界面开发过程中的平台无关模型和相关模型的结构.

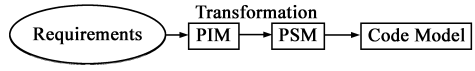


图2 模型驱动开发过程

4.1 笔式界面平台无关模型

为将模型驱动思想运用到笔式界面架构中,应首先抽取出笔式界面的平台无关模型(PIPIM).本文从提出的笔式界面架构出发,提出四种描述界面和交互的与实现技术和平台无关的模型,分别是用户模型、场景树模型、界面表示模型和交互模型,其业务逻辑集中于用户界面层.

首先,界面设计者从最终用户获得初步的需求,以故事书的形式建立用户的概念模型.由于笔式用户界面面向的是普通大众,讲故事的方式比较符合其认知特征,可以帮助设计人员准确定位用户的需求,这种方法的好处是比较简洁明了,适合与用户交流.用户概念模型表示用户对实际业务中事物的描述,它是对软件系统需要解决问题进行高度抽象和概括的结果.

其次,从用户概念模型出发,提取故事书每一页之间的关系,形成场景树模型(Scenario Tree Model,STM).场景树中每个树结点代表一个交互场景,整棵树表示用户完成交互任务的界面之间的逻辑关系.用几组界面或图示描述场景,并且通过箭头联系各个界面,表明场景切换与控制转移.由于每个场景内发生不同的交互任务以及场景之间存在依存关系,在此我们将场景分为根场景、管理场景、兄场景和叶场景.其中根场景是指应用的初始场景,存在是唯一的;管理场景负责管理若干子场景;兄场景是指与某场景存在同数据不同显示方式的场景;叶场景是指用户交互内容的场景;场景树可描述如下:< *STM* > :: = { < *Scene* > }, *Scene* :: = < *RootScene* > | < *ManagementScene* > | < *BrotherScene* > | < *LeafScene* > .

然后对每个场景中的界面进行设计,提取该界面的界面元素,形成界面表示模型(Interface Presentation Model,IPM).给定界面的界面元素包括界面背景、交互对象和应用对象,可表示为:< *IPM* > :: = < *BK* > { < *Interaction Object* > } { < *Application Object* > };在笔式用户界面领域,交互对象主要是指笔式交互 PGIS(Paper、Gadget、Icon、Sketch)范式内的 Gadget 和 Icon;应用对象是最终用户使用笔式交互的内容管理容器.笔式界面中应用对象是指各种交互框:包括符号框、Sketch 框、媒体框,负责完成用户对交互内容的操作.

最后设计场景中的交互任务,形成交互任务模型(Interaction Task Model,ITM).交互任务模型定义了在给定场景中,通过交互达到的目标,以及实现目标过程中用到的界面元素及对其进行的操作.根据笔式交互的特征[5],给定界面下的交互任务模型可以描述成:$< ITM > ::= \{< Goal > \ < Pen\ Primitive > \ < PM >\}$.其中 *Goal* 表示交互达到的目标,*Pen Primitive* 表示笔交互原语,IPM 为交互中所用到的界面元素,其可在界面表示模型中详细定义.

该平台无关模型描述了笔式界面的界面间关系、界面元素以及用户通过笔与界面之间的交互过程,每个模型都与具体实现细节无关.

4.2　笔式界面平台相关模型

为实现平台无关模型的业务逻辑,需要在具体平台上实现必要的组件服务,形成平台相关模型.我们从提出的笔式界面开发框架中提取了出核心模块,形成笔式界面平台相关模型(PIPSM),包括运行控制模型、场景管理模型、应用文档模型、核心引擎模型以及交互对象库.

运行控制模型(Running Control Model,RCM)主要负责控制整个应用程序的运行,主要包括如下几方面:程序的初始化,包括 Ink 引擎、Media 引擎、Display 引擎和所用资源等;初始化根场景,进入根场景以循环的方式等待交互命令;程序退出,包括释放引擎和资源,其模型结构如图 3 所示.笔式界面的平台相关运行控制模型以工程模板的形式出现.

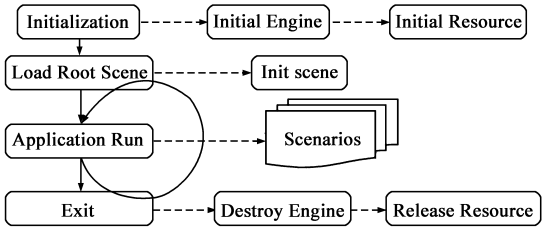


图3　运行控制模型

场景管理模型(Scene Management Model,SMM)负责给定场景的管理工作,具体包括场景初始化、场景渲染子模型、对象管理子模型和交互状态子模型.场景渲染子模型(Scenario Render Model,SRM)主要负责渲染给定场景中的界面元素,顺序为渲染背景、渲染交互对象和渲染应用对象.对象管理子模型(Object Management Model,OMM)主要用来管理场景中应用对象和交互对象,提供两个对象容器和增加、删除、更改、查询等服务.交互状态子模型(Interaction State Model,ISM)负责将 Ink 交互原语信息进行分析处理,根据上下文,转化为应用相关的具有语义的交互原语,并调用相关处理函数.笔式界面的平台相关场景管理模型以目标平台代码模板的形式出现.

应用文档模型(Application Document Model,ADM)主要用来管理笔式用户界面典型的文档组织结构,例如卡片结构和页结构,向笔式界面开发人员提供文档的数据结构和使用接口,对于用户自定义的文档结构,可以通过自定义方式进行添加.笔式界面平台相关应用文档模型以库的形式为上层提供文档解析服务.

核心引擎模型(Core Engine Model,CEM)包含三个核心引擎:Ink 引擎、Media 引擎和 Display 引擎.交互对象库以库的形式提供了若干预定义的交互对象供上层开发直接使用.笔式界面平台相关核心引擎模型以库的形式为上层提供支持.

各模型之间存在依赖关系,例如运行管理模型初始化根场景后,进入根场景,其场景管理模型通过应用文档模型得到应用对象的数据格式,在初始化交互对象和应用对象后,将对象交给对象管理模型进行管理;场景管理模型从用户处得到原始输入,交给交互引擎模型进行处理;得到该输入的交互引擎模型将输入解析为笔原语,同时从对象管理模型中得到用户当前交互的对象,形成交互原语,执行对应的任务.

4.3　笔式界面平台无关模型到相关模型的转换

笔式界面平台无关模型以文本形式保留了设计的结果,平台相关模型以代码或者组件形式在给定平台下得到了部分实现,而最终的完整系统实现需要平台无关模型与相关模型结合,形成系统的平台相关完整模型,为此需要利用平台无关模型到相关模型的转换技术.我们采用以 XML 语言描述平台无关模型,基于标记的代码转换方法.平台转换工作在平台相关模型方面主要集中于向场景管理模型的转换,因为场景管理模型以模板代码的形式对模板场景的初始化、销毁等工作做了封装,并包含了对象管理子模型、场景渲染子在模型和交互状态子模型,在平台转换时,不仅需要根据设计的场景实例化场景管理模型,还要初始化场景对象、赋予三个子模型对应的值或属性,设置交互状态等.一个通用的转换可定义为 $Transformation = < Input > + < Process > + < Output >$;*Input* 是转换的输入,*Process* 为转换的处理过程,*Output* 为转换的输出结果.

4.3.1　场景树模型到场景管理模型的转换

该转换的目标是根据设计创建各场景,形成场景管理模型实例.转换的 Input 为一棵平台无关场景树模型,Output 为若干场景管理模型实例,表示为 $Input = < STM >$,$Output = \{< SMM\ instance >\}$,*Process* 包括创建一连串 SMM 实例并设置其属性映射,以及约束条件,形式上 $Process = \{< SMM\ instance\ creation > + < Attribute\ Mapping > + < Constraints >\}$,其中 $Attribute\ Mapping = < ID\ mapping > + < Name\ mapping > + < Scene\ type\ mapping >$,将

STM 中的 ID 值、场景名和类型属性值设置到 *SMM* 实例中. *Constraints* = < *Attribute Type Constraints* > + < *Attribute Value Constraints* > + < *ID Unique Constraints* > + { < *Parent-Child Constraints* > | < *Sibling Constraints* > },其中 Attribute Type Constraints 约束了场景三属性类型在映射前后一致;Attribute Value Constraints 约束了场景三个属性值在映射前后一致;ID Unique Constraints 约束了场景的 ID 属性的全局唯一性;Parent-Child Constraints 和 Sibling Constraints 约束了映射后的场景间的父子或兄弟关系不变.

4.3.2 界面表示模型到场景管理模型的转换

该转换的目标是在根据设计中的界面表示,在平台相关模型上创建界面的可视化表征,既背景、交互对象和应用对象.转换的 *Input* 是所有场景的平台无关界面表示模型 *IPM*, *Output* 为各场景的场景管理模型实例的初始化代码,形式上描述为 *Input* = < *IPM* >, *Output* = { < *Scene Init Code* > }. *Process* = { < *Scene Background Attribute Mapping* > + < *IOs creation*, *IO Attribute Value mapping*, *put into OMM* > + < *AO Init Function Creation* > + < *Load Data Function Calling* > } + *Constraints*. Process 包括约束条件和若干代码:读取并设置背景属性、读取并创建交互对象(IO)、设置交互对象属性并放入对象管理子模型、创建应用对象的初始化例程、调用数据读入例程. *Constraints* = < *Attribute Type Constraints* > + < *Attribute Value Constraints* > + < *IO ID Unique* > + < *Init Code Position Constraints* >,前两个约束了界面对象的属性和值在转换前后的一致性,IO ID Unique 约束了交互对象的 ID 号在全局的唯一性, *Init Code Position Constraints* 约束了转换生成的代码插入场景的位置为场景初始化位置.

4.3.3 交互任务模型到场景管理模型的转换

该转换的目标是根据已设计完成的交互任务模型,在平台相关模型中创建界面的交互过程及响应代码.转换的 Input 是所有场景的平台无关交互任务模型 ITM, Output 为各场景的场景管理模型实例的交互原语判断及函数调用代码. 形式上, *Input* = < *ITM* >, *Output* = { < *Pen Primitive Decision Code* > + < *Task Function Calling Code* > }. *Process* = < *Goal-Function Creation* > + < *Interaction Primitive Parsing* > + < *Pen Primitive Decision* > + < *Interface Objects Decision* > + < *Task Function Calling* > . Process 首先根据 ITM 的 Goal 创建任务函数代码,然后解析 Pen Primitive 和界面对象 ID、创建判断 Pen Primitive、判断界面对象以及调用任务函数的代码.该转换形成的代码属于场景管理模型的交互状态子模型.

4.4 模型驱动的笔式用户界面开发框架

基于上文提出的笔式用户界面开发框架、开发过程模型及模型转换方法,本节详细论述模型驱动的笔式界面设计开发的详细流程,以指导笔式用户界面的开发.我们将设计开发过程分为需求分析、平台无关模型设计、模型转换和运行评估测试四个阶段.设计和实现阶段同开发过程模型成对应关系,每个阶段设计的文档资源作为下一个阶段的设计的输入,保证设计开发流程的完备性和继承性.平台无关模型以文本文件的形式保留了设计的结果,使得平台变更时保留原有的业务逻辑从而得到复用,进而减小了开发成本,提高了开发的效率.又因为平台变更时采用模型转换的方法,保证了平台无关模型成为设计的唯一来源,避免了传统开发过程中,设计与实现逐步偏离的缺点.

以下逐一阐述笔式用户界面设计开发每个阶段详细的流程.基于模型驱动的笔式界面开发分五个阶段,分别是分析需求、建立平台无关模型、从平台无关模型转换到相关模型、整合平台相关模型和二次开发,开发完毕的系统编译链接及测试.

在分析需求阶段,界面设计人员听取用户叙述其工作内容,掌握业务逻辑,并了解用户的思维模型,包括目的、动机、期望等,全面捕获场景.该阶段确定用户的详细需求、完成的任务以及用户本身的特征,以故事叙述的方式形成故事书[9];

在建立平台无关模型阶段,依据需求阶段的故事书,使用场景设计工具,设计场景树、各个场景的界面元素和场景状态图.在设计场景树时,使用场景设计工具,用几组界面或图示表示应用的场景树,并且通过箭头联系各个界面,表明场景切换和场景之间的逻辑关系;在场景设计工具中,必须明确场景的类别:根场景、管理场景和子场景,以建立场景之间的关联关系,保证映射到代码模型时,场景之间正确的跳转关系.下一步设计笔式应用中的界面对象,包括背景、应用对象和交互对象.设置背景的大小、颜色、背景图以布置场景;设计笔式界面中的应用对象,即各种交互框:包括符号框、Sketch 框、媒体框,用设计工具以可视化形式设计其表征属性,包括:大小、颜色等.设计每个场景中的交互对象时,使用设计工具,确定交互对象的类别,在界面中的逻辑位置,以及大小、颜色、文字、可见性等表征属性.最后设计笔式应用的交互任务模型,主要包括场景中的交互任务、场景之间的切换,形成场景任务列表.完成的场景任务需要同 Storyboard 中确定的场景任务需求保持一致.设计应用系统中场景任务需要的交互原语;根据笔原语类型,设计完成交互任务的交互方式,并形成每个场景中的交互原语列表.设计好的平台无关模型存储为 XML 格式.

在模型转换阶段,选择目标平台,根据平台无关模型设计结果,通过模型转换工具,通过读取该平台的场

景管理模型的模板,将平台无关模型中的场景树、界面对象和交互任务等转换为目标平台代码,每个场景形成一个场景管理模型实例.

下一步整合平台相关模型,将每个场景管理模型实例嵌入该平台的运行控制框架中,此时形成一个包括运行控制模型、场景管理模型、应用文档模型、核心引擎模型和交互对象库的完整应用框架,供二次开发.开发完成后形成可执行的应用程序,供测试人员测试评估.

由以上开发过程看出,对给定系统的设计开发的重点已转移到平台无关模型的建模阶段,该阶段由设计工具提供支持,设计人员和用户一起参与该过程,因用户的参与使得需求与用户要求一致.其余阶段采用自动化的方式形成平台相关的代码,它们是界面开发的大部分内容,因此也减少了二次开发的工作量.在需求变更时,只需要更改平台无关模型并重复该过程;在平台变更时,只需要保留原有设计,重新选择目标平台,并转换到该平台即可,极大提高了软件开发的效率,缩短了开发过程的周期,降低了开发过程中潜在风险.

5　基于模型驱动的笔式界面开发工具和开发实例

为支持上述基于模型驱动的笔式界面开发框架,我们建立了面向笔式界面的开发工具 Iris.该工具提供平台无关模型建模以及平台无关模型到相关模型转换两种功能,以可视化建模方式提供场景树、界面以及交互任务的建模功能,建模结果以 XML 格式保存;通过读取建模结果并选择目标平台,进行模型转换.我们以笔式个人信息管理系统开发为例讲述工具的使用方法.

5.1　需求分析

笔式个人信息管理系统主要为用户管理文件、安排计划日程、记录日常信息等功能,用户能够在日常生活中以纸笔交互方式去管理个人信息.为了减少用户的认知负担,我们采用实物界面,以日常办公室作为隐喻.

从用户特征角度看,用户为知识工作者;其管理个人信息的活动是重复、灵活、结构化和非结构化结合的模式.因此很适合通过办公室隐喻表现其活动场景.用户办公室场景具有办公桌、书架等实物,桌面上摆放着多堆文件,日程安排,日记本和邮件等.用户可以在办公室将桌面非结构化管理的文档归档到书架上,也可以从书架上取下参考的文档.用户在桌面上可以阅读、书写、批阅文件;可以从堆里找出需要的文件;可以按照类别整理堆资料;也可以将多个堆文件合并为一堆文件;用户处理完成后将文件放入到堆上;如果归档文件,可以将文件放入到书架上.

5.2　场景树、用户界面和交互设计

(1)场景树设计　Iris 在左侧提供了场景结点和链接等建模元素,用以建立不同类型的场景结点及其之间的关系.笔式个人信息管理系统的场景树以办公室场景为根场景,包含桌面、堆、书架、夹、日历、日记、电子邮件、程序管理等子场景,如图 4 所示.

(2)用户界面设计　每个场景都有相应的用户界面设计.Iris 在界面设计场景的右上方为用户界面设计提供了界面设计的建模元素,设计结果如图 5 所示.

图4　用Iris 设计场景树示例

图5　用Iris设计界面部分示例

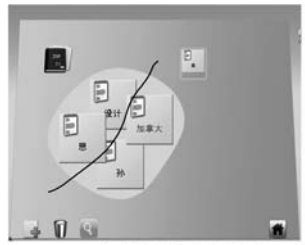
(3)交互任务设计　每个场景有相应的交互任务设计.Iris 在交互任务设计场景的下方提供了交互任务的建模元素,其设计如表 1 所示.

表 1　桌面场景的交互设计

Interaction Primitive		Interaction Task / Action
Pen Primitive	Interface Object	
Hold-Line	Background	Create a file
Stroke	File	Move a file
Stroke	Pile	Move a Pile
…	…	…

5.3　模型转换和二次开发

Iris 产生的文档包括场景树、各场景的界面设计图和交互任务图,Iris 为用户提供了几种目标平台,在选择好目标平台后,通过模型转换,形成基本的程序框架.用户可利用该基本程序框架进行二次开发.图 6(*a*)显示了桌面场景的运行效果,图 6(*b*)显示了笔交互后的界面效果.



(*a*) 桌面场景运行时界面

(*b*) *a*图中笔手势交互后的界面

图6　用Iris开发的笔式个人信息管理系统实例运行界面

以上实例表明,基于模型驱动的笔式界面开发框架及其开发工具 Iris 能够对笔式用户界面的用户模型、界面设计和交互设计进行统一建模,能够有效满足笔式用户界面对用户个性化和功能多样化的要求,能完

整描述笔式界面的交互方法,又因为平台无关模型保留了系统的抽象描述,在需求变更时,只需要更改平台无关模型并进行相应的转换,保证了设计与实现的一致性;在跨越平台时只需要更换目标平台并进行模型间的转换就可以快速进行平台间迁移,保证了设计的可复用,有效地提高了笔式用户界面的开发效率.

6 结论和下一步工作

本文针对笔式用户界面开发的方法问题,采用模型驱动的软件开发思想,提出了一种基于模型驱动的笔式用户界面开发的框架,包括笔式界面开发平台无关模型、平台相关模型以及平台无关模型到相关模型的转换方法,并给出了详细的开发流程,基于该框架实现了笔式用户界面开发工具 Iris,并通过 Iris 构建了一个笔式个人信息管理系统实例,该实例表明模型驱动的笔式界面开发方法能够有效满足笔式用户界面对用户个性化和功能多样化的要求,保证设计与实现的一致性,保证了设计结果的可复用,有效减少了开发的复杂性,降低了开发的风险,提高了开发的效率.

未来的工作将进一步完善基于模型驱动的笔式用户界面开发模型和开发流程,提高代码的自动化效率,进一步提高笔式用户界面开发的效率和质量.

参考文献:

[1] 董世海,王坚,戴国忠.人机交互和多通道用户界面[M].北京:科学出版社,1999.

[2] Botterweck G. A model-driven approach to the engineering of multiple user interfaces[A]. Proc MoDELS'06 Workshop on Model Driven Development of Advanced User Interfaces[C]. Berlin: Springer Press. 2007. 106 – 115.

[3] Balagtas-Fernandez F T, Hussmann H. Model- driven development of mobile applications[A]. In Proc. 2008 23rd IEEE/ACM International Conference on Automated Software Engineering[C]. Washington: IEEE Computer Society. 2008. 509 – 512.

[4] Funk M, Hoyer P, Link S. Model-driven instrumentation of graphical user interfaces[A]. Proc 2009 2nd International Conferences on Advances in Computer-Human Interactions[C]. Washington: IEEE Computer Society. 2009. 19 – 25.

[5] 田丰,秦严严,王晓春等.PIBG Toolkit:一个笔式界面工具箱的分析与设计[J].计算机学报,2005,28(6):1036 – 1042.
Tian Feng, Qin Yan-yan, Wang Xiao-chun, et al. Analysis and design on PIBG toolkit: a pen-based user interface toolkit[J]. Chinese Journal of Computers. 2005, 28 (6): 1036 – 1042. (in Chinese)

[6] 李杰.笔式用户界面开发方法研究[D].北京:中国科学院软件研究所,2004.

[7] Cook S, Jones G, Kent S, et al. Domain-Specific Development with Visual Studio DSL Tools[M]. Boston: Addison-Wesley Professionals, 2007.

[8] Kalnins A, Vilitis O, Celms E, et al. Building tools by model transformations in Eclipse[A]. Proc DSM'07 Workshop of OOPSLA 2007[C]. Montreal: Jyvaskyla University Printing House, 2007. 194 – 207.

[9] Aquino N. Adding flexibility in the model-driven engineering of user interfaces[A]. Proc 1st ACM SIGCHI Symposium on Engineering Interactive Computing Systems[C]. New York: ACM Press, 2009. 329 – 332.

[10] Caplat G, Sourrouille J L. Model mapping using formalism extensions[J]. IEEE Software, 2005, 22(2): 44 – 51.

[11] Almendros-Jiménez J M, Iribarne L. An extension of UML for the modeling of WIMP user interfaces[J]. Journal of Visual Languages and Computing, 2008, 19(6): 695 – 720.

[12] Kleppe A, Warmer J, Bast W. MDA Explained-The Model Driven Architecture: Practice and Promise[M]. Boston: Addison-Wesley Professional, 2003.

[13] Myers B, Hudson S E, Pausch R. Past, present, and future of user interface software tools[J]. ACM Transactions on Computer-Human Interaction, 2000, 7(1): 3 – 28.

作者简介:

陈明炫 男,1982 年出生于内蒙古包头.中国科学院软件研究所博士研究生.研究方向为个人信息管理、人机交互技术.
E-mail: mxchen04@gmail.com

邓昌智 男,1978 年出生于湖南永州.中国科学院软件研究所博士.研究方向为个人信息管理、人机交互、信息可视化.