摘要

随着移动智能设备应用的日益广泛，以及其综合性能的显著提高，在实际应用中，我们对应用程序有了更高的要求，希望某些应用程序具有这样的能力：应用程序可以携带当前界面和相关的上下文数据从一个设备迁移到另一个设备，并且能够继续执行刚才的任务。这种程序不再被单一的计算机或者用户所束缚，它能够在网络上自由的漫步。并且这种程序还可以为一个群组的用户提供服务，收集人们的输入信息，便于用户间的交互。能够迁移执行程序的能力对移动计算来说特别有用：当用户在不同的计算环境间移动时，可以随身携带自己的应用程序任务。

平台是一组硬件和操作系统的集合，有时平台间的差异会十分巨大：显示屏尺寸的大小、分辨率和颜色数的差异，计算能力的强弱，交互手段的不同，支持语言的种类等。这些差异带来了一些界面设计上的不同。如何克服这些变化带来的困难，使程序开发人员在设计的初期只是关注于需要解决的核心任务，而不去过多地考虑各种平台的特性，对我们来说是一个巨大的挑战。与此同时，界面设计者还期望可以将原来平台的开发经验和知识方便的应用到新的平台的软件开发中。

本文提出了基于自适应表示组件的表示层模型，来作为多平台界面设计的方法。该表示层模型的核心是各模型间的抽象、映射、转换和具体化，主要通过对数据对象、事件路由和抽象界面描述的配置，来实现数据映射、事件映射和界面映射，然后经过界面生成器的处理转换，将抽象界面转化为具体界面，实现了界面的自动生成，提高了程序的重用性和可用性。

另外本文提出了一个界面迁移中的资源满足程度评价方法：为界面加入了特定格式的数据结构（该数据按照特定的规则进行分组），并以此数据为参数建立了相应的算法，来计算目标设备的资源是否满足界面迁移的要求，从而对界面迁移的可行性做出判定，并在界面迁

移之前为界面调整提供相关信息。

关键词： 抽象界面　表示层模型　统一数据网关　界面迁移　资源评价

ABSTRACT

As the expansion rate of mobile platforms is becoming greater and their capabilities have considerably improved, there is now a need that the applications for mobile platforms have this characteristic: it can migrate from one machine to another, taking their user interface and application contexts with them, and continue from where they left off. Such applications are not tied to one user or one machine, and can roam freely over the network, rendering service to a community of users, gathering human input and interacting with people. The ability to migrate executing programs has applicability to mobile computing as well. Users can have their applications travel with them, as they move from one computing environment to another.

A platform is generally defined as a specific combination of hardware and operating system. Sometimes, the capabilities of each platform are very different: the devices differ in screen size, resolution or color number, capacity of computation, interaction mode, languages, and so on. How to solve the difficulties brought by these differences, and pay attention to the main tasks, is a great challenge. At the same time, UI designers expected to be able to reuse their knowledge of a given version of the system when designing the same item for another platform.

We propose an approach called *a presentation model bases on adaptive module* as method to build user interfaces for multiplatform systems. The essence of this approach consists in composing several functions of abstraction, reflection, translation. By configure the data object, abstract interface, event route table, we gained the mapping of data, event, and interface. Then with the UI generator, we get the final UI that meet our requirements.

We also propose an evaluation method of resource satisfaction in interface migration. This paper adds a data structure to the User Interface (this data structure is grouped by certain rules), and establishes a corresponding algorithm which regards this structure as the parameter. The algorithm can judge the feasibility of interface migrating through the calculation of the target platform's resources, and provides

related information for the UI adjustment before UI migration.

Keywords: Abstract Interface, Presentation Model, Uniform Data Gateway, Interface Migration, Resource Evaluation

原创性声明和关于学位论文使用授权的声明

原 创 性 声 明

本人郑重声明：所呈交的学位论文，是本人在导师的指导下，独立进行研究所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的科研成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本声明的法律责任由本人承担。

论文作者签名：郭志强　　日　期：2007.4.5.

关于学位论文使用授权的声明

本人完全了解山东大学有关保留、使用学位论文的规定，同意学校保留或向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅；本人授权山东大学可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或其他复制手段保存论文和汇编本学位论文。

(保密论文在解密后应遵守此规定)

论文作者签名：郭志强　导师签名：史高华　日　期：2007.4.5.

1 绪论

1.1 研究背景

计算和通信技术的迅速发展，使计算机已经远远超出了其传统上的作用，而不仅仅作为实验室中科学计算的工具，计算机正以多种形态存在于我们的生活空间，并发挥其信息处理、存储、通信的作用。计算已经从一种稀缺独特的资源，演化为丰富平常的资源。目前，计算机的一项主要任务就是帮助提高人们日常工作生活的质量。因此，如何使计算和通信无所不在并成为普通用户都能方便享用的服务，成为研究者所关注的课题，即Pervasive Computing。这个概念的中文译法很多，有普及计算、泛在计算、无所不在的计算、普适计算等，其中“普适计算”得到普遍认可[45]。

从字面看，普适计算可以解释为计算的普及性和适应性。前者指网络互联的计算设备以各种形式形态渗透到人们的生活空间，成为人们获得信息服务的载体——即信息空间普遍存在；后者指信息空间能以适合用户的方式提供能适应变化的计算环境的连贯的信息服务——即信息服务方便适用。普适计算的目标就是：在计算和通信无所不在的基础上，建立以人为中心的计算环境。普适计算有时也被称为Ubiquitous Computing、Proactive Computing和Calm Computing等，其理念都是创造一个以人为本的信息服务新环境。

普适计算力图将以计算机为中心的计算、转变为以人为中心的计算。这种转变将极大地促进信息技术在全社会的普遍应用，具有重要的战略意义。从20世纪90年代后期开始，世界各先进国家看到了数字技术将对未来社会生活产生革命性的影响，纷纷投入大量资源，在该领域展开深入的研究。普适计算研究具有重要的理论意义和很高的产业价值。

普适计算引发了对一种全新计算模式的探索，具有鲜明的交叉学科的特点。即便仅考虑计算机学科的问题，也涉及几乎整个学科的各个层面。

各种计算设备，尤其是移动手持设备加入到计算环境中，对界面的开发提出了许多新的挑战。一个重要的挑战是如何利用现在各种设备的现有资源，统一开发适应不同设备特性的应用界面，在保持各个平台界面一致性的前提下，一方面避免针对不同设备的特性单独开发各自的界面，另一方面提高现有的开发经验和

资源的重用性。

解决问题的困难在于各种设备的硬件资源的差异性（通讯能力、计算能力、显示能力、存储能力等）。这些差异带来的问题是，普通的单一界面无法适应种类繁多且差异巨大的各类硬件平台。而如果为每个平台单独设计界面，且不说其工作量的巨大，还会导致各平台间的不一致性，这将为多平台应用的发展增添巨大的障碍。并且，一般的软件系统无论在开发还是运行维护阶段，用户界面的设计和修改都占去了大量的时间。开发人员需要花费大量的精力来设计和修改用户界面，而且这些工作通常都是繁杂和重复性的，不仅降低了效率，也造成代码难以维护和扩展。如何克服这些困难，使程序开发人员在设计的初期只是关注于需要解决的任务，而不去过多地考虑各种平台的特性，可以将原来的开发经验和知识方便地移植到现在的开发过程中，是必须解决的问题。

如果将界面的描述与实现分离，对界面生成相关的元素进行某种层次上的抽象，该抽象描述将各种平台的实现细节隐藏，提供统一的抽象界面描述方法，开发人员就能够使用统一的抽象界面描述方法来描述任务，而不必考虑各个平台间的差异。同时，这些抽象的表示能够根据设备的不同特性自动生成设备相关的界面。

解决问题最主要的思路是使用模型驱动的思想，利用统一的抽象界面描述方法来描述任务，通过模型的转换逐步具体化，最终产生基于某个具体目标平台的界面。

本文针对上述目标，对现有的工作做了分析，提出了抽象程度更高且能满足开发需要的自适应表示组件，作为开发人员描述任务的最基本的元素。在此基础上，通过对数据的统一处理，使得自适应表示组件更具有普适性，有效地支持界面的自动生成。

1.2 本文主要工作与创新点

我们给出了一个基于自适应表示组件的表示层模型，并给出了各个模型的描述以及对数据对象、事件路由和抽象界面描述的配置和界面生成的工作过程，最后通过一个简单的示例来说明。

另外，本文尝试提出了一种界面资源的表示及评价方法，并讨论了如何利用

该方法，根据上下文判断目标设备是否满足应用程序的迁移需要，并能在一定条件下适当调整界面以使应用程序能够实现迁移等相关内容。

本文的工作及创新之处：

- 自适应组件在平台间和平台内两个层次上的自适应性，使其可以更好地适应不同平台和不同模态的变化；通过对模型进行动态的配置，实现了界面的自动生成，提高了程序的重用性和可用性。
- 通过表示层数据和应用数据的不同描述和转换，可以更好地屏蔽由于数据类型不同带来的变化，支持组件操作的普适性。
- 为界面加入了资源需求的描述机制，建立了资源需求列表，并为资源需求列表提出了一种特定的数据结构，以便于清晰表达资源需求；利用上述数据提出一个资源需求满足程度评价算法；利用该数据结构和算法实现了自适应组件的平台间的自适应特性。

1.3 本文结构

全文的组织结构如下：

第一章主要介绍了本文的研究背景、研究目标及内容安排。

第二章主要介绍了一些界面开发的相关知识，以及目前多平台界面开发这一研究方向的研究现状等内容。

第三章介绍了表示层模型，对涉及到的抽象界面模型、表示组件模型、数据模型等几个具体模型，以及统一数据网关等内容作了简单介绍。

第四章介绍了模型的工作过程，包括几个模型的配置及数据、界面、事件的映射等具体内容。

第五章介绍了自适应表示组件的自适应特性及部分实现细节。

第六章引入了一个用于表示资源需求情况的数据结构，并介绍了基于资源评价的界面调整方法。

第七章对本文进行了总结。

2 UI 开发相关知识

2.1MVC

MVC 模式是"Model-View-Controller"的缩写，即"模式-视图-控制器"。MVC 应用程序总是由这三个部分组成。Event(事件)导致 Controller 改变 Model 或 View，或者同时改变两者。只要 Controller 改变了 Models 的数据或者属性，所有依赖的 View 都会自动更新。类似的，只要 Controller 改变了 View，View 会从潜在的 Model 中获取数据来刷新自己。MVC 模式最早是 smalltalk 语言研究团提出的，应用于用户交互应用程序中。MVC 模式是一个复杂的架构模式，其实现也显得非常复杂。但是，我们已经总结出了很多可靠的设计模式，多种设计模式结合在一起，使 MVC 模式的实现变得相对简单易行。

2.1.1 MVC 设计思想

Model-View-Controller，即把一个应用的输入、处理、输出流程按照 Model、View、Controller 的方式进行分离，这样一个应用被分成三个层——模型层、视图层、控制层。

视图(View)代表用户交互界面，对于 Web 应用来说，可以概括为 HTML 界面，但有可能为 XHTML、XML 和 Applet。随着应用的复杂性和规模性，界面的处理也变得具有挑战性。一个应用可能有很多不同的视图，MVC 设计模式对于视图的处理仅限于视图上数据的采集和处理，以及用户的请求，而不包括在视图上的业务流程的处理。业务流程的处理交予模型(Model)处理。比如一个订单的视图只接受来自模型的数据并显示给用户，以及将用户界面的输入数据和请求传递给控制和模型。

模型(Model)：就是业务流程/状态的处理以及业务规则的制定。业务流程的处理过程对其它层来说是黑箱操作，模型接受视图请求的数据，并返回最终的处理结果。业务模型的设计可以说是 MVC 最主要的核心。

控制(Controller)可以理解为从用户接收请求，将模型与视图匹配在一起，共同完成用户的请求。划分控制层的作用也很明显，它清楚地告诉你，它就是一

个分发器，选择什么样的模型，选择什么样的视图，可以完成什么样的用户请求。控制层并不做任何的数据处理。例如，用户点击一个连接，控制层接受请求后，并不处理业务信息，它只把用户的信息传递给模型，告诉模型做什么，选择符合要求的视图返回给用户。因此，一个模型可能对应多个视图，一个视图可能对应多个模型。

模型、视图与控制器的分离，使得一个模型可以具有多个显示视图。如果用户通过某个视图的控制器改变了模型的数据，所有其它依赖于这些数据的视图都应反映到这些变化。因此，无论何时发生了何种数据变化，控制器都会将变化通知所有的视图，导致显示的更新。这实际上是一种模型的变化-传播机制。模型、视图、控制器三者之间的关系和各自的主要功能，如图 2-1 所示。

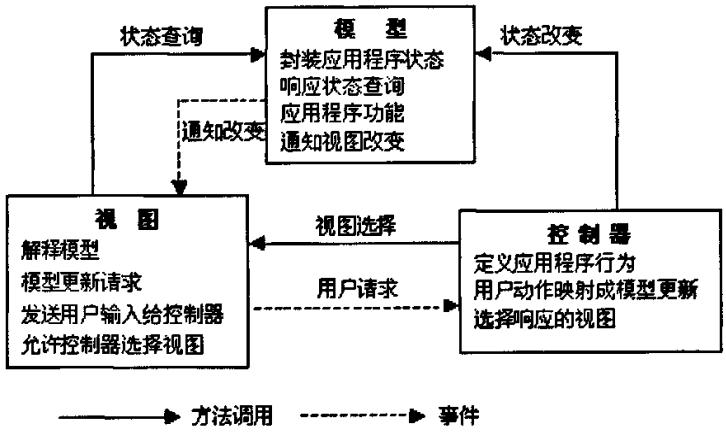


图 2-1 MVC 组件类型的关系和功能

2.1.2 MVC 设计模式的扩展

MVC 模式具有极其良好的可扩展性。它可以轻松实现以下功能：

（1）实现一个模型的多个视图；

（2）采用多个控制器；

（3）当模型改变时，所有视图将自动刷新；

（4）所有的控制器将相互独立工作。

这就是 MVC 模式的好处，只需在以前的程序上稍作修改或增加新的类，即可轻松增加许多程序功能。以前开发的许多类可以重用，而程序结构根本不再需要

改变，各类之间相互独立，便于团体开发，提高开发效率。下面讨论如何实现一个模型、两个视图和一个控制器的程序。其中模型类及视图类根本不需要改变，与前面的完全一样，这就是面向对象编程的好处。对于控制器中的类，只需要增加另一个视图，并与模型发生关联即可。该模式下视图、控制器、模型三者之间的示意图如图 2-2 所示。

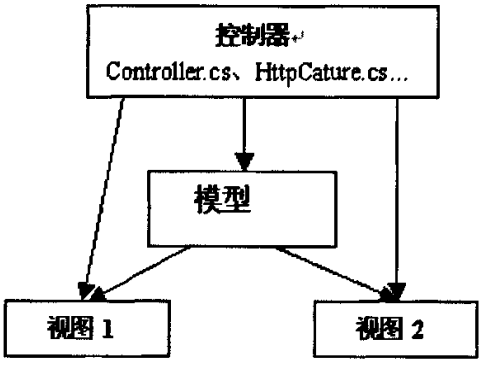


图 2-2 视图、控制器、模型三者之间的关系示意图

同样也可以实现其它形式的 MVC 例如：一个模型、两个视图和两个控制器。从上面可以看出，通过 MVC 模式实现的应用程序具有极其良好的可扩展性。

2.1.3 MVC 的优点

大部分用过程语言比如 ASP、PHP 开发出来的应用，初始的开发模板就是混合层的数据编程。例如，Web 应用中直接向数据库发送请求并用 HTML 显示，开发速度往往比较快，但由于数据页面的分离不是很直接，因而很难体现出业务模型的样子或者模型的重用性。产品设计弹性力度很小，很难满足用户的变化性需求。MVC 要求对应用分层，虽然要花费额外的工作，但产品的结构清晰，产品的应用通过模型可以得到更好地体现。

首先，最重要的是应该有多个视图对应一个模型的能力。在目前用户需求的快速变化下，可能有多种方式访问应用的要求。例如，订单模型可能有本系统的订单，也有网上订单，或者其他系统的订单，但对于订单的处理都是一样，也就是说订单的处理是一致的。按 MVC 设计模式，一个订单模型以及多个视图即可解决问题。这样减少了代码的复制，即减少了代码的维护量，一旦模型发生改变，

也易于维护。

其次，由于模型返回的数据不带任何显示格式，因而这些模型也可直接应用于接口的使用。

再次，由于一个应用被分离为三层，因此有时改变其中的一层就能满足应用的改变。一个应用的业务流程或者业务规则的改变只需改动 MVC 的模型层。

控制层的概念也很有效，由于它把不同的模型和不同的视图组合在一起完成不同的请求，因此，控制层可以说是包含了用户请求权限的概念。

最后，它还有利于软件工程化管理。由于不同的层各司其职，每一层不同的应用具有某些相同的特征，有利于通过工程化、工具化产生管理程序代码。

2.1.4 MVC 的不足

MVC 的不足体现在以下几个方面：

（1）增加了系统结构和实现的复杂性。对于简单的界面，严格遵循 MVC，使模型、视图与控制器分离，会增加结构的复杂性，并可能产生过多的更新操作，降低运行效率。

（2）视图与控制器间的过于紧密的连接。视图与控制器是相互分离，但确实联系紧密的部件，视图没有控制器的存在，其应用是很有限的，反之亦然，这样就妨碍了他们的独立重用。

（3）视图对模型数据的低效率访问。依据模型操作接口的不同，视图可能需要多次调用才能获得足够的显示数据。对未变化数据的不必要的频繁访问，也将损害操作性能。

（4） 目前，一般高级的界面工具或构造器不支持 MVC 模式。改造这些工具以适应 MVC 需要和建立分离的部件的代价是很高的，从而造成使用 MVC 的困难。

2.1.5 MVP（Model View Presenter）设计模式

Model-View-Presenter 设计模式实际上就是我们已经熟悉的 MVC 设计模式的一个最新版本；两者的主要区别是 MVP 真正将 UI 从应用程序的域/服务层中分离。在 MVC 中 view 直接处理相关的界面事件，比方说，键盘鼠标事件， 选择框被选中，按钮被按等等。而在 MVP 中 view 接收到事件，然后会将它们传

递到 Presenter，如何具体处理这些事件，将由 Presenter 来完成。从 class diagram 上来看，就是 Presenter 有 View 和 Model 的引用，Presenter 负责来管理其他两个模块。跟据两者不同来看，MVC 比较适合用来开发 components，而 MVP 比较适合进行 applications 的开发，因为使用 MVP 导致绝大部分逻辑代码集中在 Presenter，而 view 变得非常简单，适当采用良好的编码风格，可以让毫无经验的编码人员稍加培训立刻上岗，大大加速开发 view 的速度。

2.2 人机交互相关知识

人机交互技术(Human-Computer Interaction Techniques)是指通过计算机输入、输出设备，以有效的方式实现人与计算机对话的技术。它包括机器通过输出或显示设备给人提供大量有关信息及提示请示等，人通过输入设备给机器输入有关信息、回答问题等。人机交互技术是计算机用户界面设计中的重要内容之一。它与认知学、人机工程学、心理学等学科领域有密切的联系。人机交互的发展经历了指示灯和机械开关组成的操纵界面、80 年代的由终端和键盘组成的字符界面、90 年代 PC 和工作站使用的由多种输入设备和光栅图形显示设备构成的图形用户界面(GUI)、所见即所得的 WIMP (W-windows、I-icons、M-menu、P-pointing devices)界面、VR 技术(发展方向)等阶段。

人机界面 (User Interface) 又称用户界面，是计算机与人之间交流的接口。人机界面的发展从最早的计算机采用手工操作到 DOS 等操作系统采用的命令，发展到 Windows 系列采用的图形用户界面。其中，图形用户界面是介于人与计算机之间，人与机器的通信。计算机发展决定了计算机从科学计算机型发展为无处不在的计算机，人机溶合，提高了交互效率。人机界面(HCI)包括软件和硬件。HCI 是设计、评估和执行交互计算机系统以及研究由此而发生的相关现象的。HCI 是未来的计算机科学。我们已经花费了至少 50 年的时间来学习如何制造计算机以及如何编写计算机程序。下一个新领域自然是让计算机服务并适应于人类的需要，而不是强近人类去适应计算机。通常，人机界面的设计和开发在整个系统的研制中占 40%~60%的比重。

人机交互是研究人与计算机之间交互的技术。而多媒体人机交互技术是多媒体技术和人机交互技术的结合。信息表示的多样化和如何通过多种输入输出设备与计算机进行交互是多媒体人机交互技术的重要内容。多媒体人机交互是基于视线跟踪、语音识别、手势输入、感觉反馈等新的交互技术。

多媒体人机交互方式是多种多样的,其中

输入方式包括：

- 键盘输入，这是比较传统的方式；
- 鼠标输入，是图形用户界面的重要输入方式；
- 手写输入，如手写汉字识别,"平板电脑"；
- 语音输入；
- 触摸屏输入；
- 数字化仪输入，适用于 CAD/CAM 系统；
- 扫描输入，如条形码、扫描仪、光电阅读器；
- 三维输入：数据手套、三维鼠标、力矩球等；
- 视觉输入，如摄像设备，机器人的视觉。

输出方式包括：

- 显示终端输出，最重要的输出工具；
- 声响输出；
- 打印输出，标准输出设备之一；
- 三维输出，例如投影显示器、头盔显示器、电视眼境等。

多媒体人机交互技术应用于软件界面设计、自然语言人机交互、输入输出装置的设计、计算机辅助设计和制造（Computer Aided design CAD/Computer aided manufacturing CAM）等多个领域。

人机交互界面设计应遵循以下原则：

- 用户原则。人机界面设计首先要确立用户类型。划分类型可以从不同的角度，视实际情况而定。确定类型后要针对其特点预测他们对不同界面的反应。这就要从多方面设计分析。
- 信息最小量原则。人机界面设计要尽量减少用户记忆负担，采用有助于记忆的设计方案。

- 帮助和提示原则。要对用户的操作命令作出反应，帮助用户处理问题。系统要设计有恢复出错现场的能力，在系统内部处理工作要有提示，尽量把主动权让给用户。
- 媒体最佳组合原则。多媒体界面的成功并不在于仅向用户提供丰富的媒体，而应在相关理论指导下，注意处理好各种媒体间的关系,恰当选用。

人机界面设计是系统设计过程的一部分，所以必须结合到现代系统开发方法中去。目前的系统开发方法对界面设计问题和用户关注太少，以致用户批评持续不断。界面设计共同课题是让用户关心和介入。其目的在于促进人在系统开发中的参与与作用。

人机界面设计下一代方法是交互的集成方法。 它将大量地使用语音、自然语言和高级图形，也可用其它交互媒体，如眼的动作和手势、姿态等，还可用三维图像以生动地引导解释交互和任务。

2.3 相关工作

在多平台UI开发中，目标平台可能存在以下约束和限制：

- 目标平台的屏幕分辨率较低。
- 目标平台的屏幕分辨率与原平台相似，但是UI需要显示的更大一些，或者UI元素的距离需要拉远一些（例如目标平台是触摸屏），或者屏幕的一部分需要用来做其它用途（如使用屏幕虚拟键盘）。
- 目标平台的可用窗口部件较少，因为运行的版本是简化版。
- 在目标平台上，某些窗口部件可用性较低，因为该平台缺少键盘等工具。

由于上述原因，以及操作系统的差异，导致为单一平台设计的UI，通常无法直接运行于其他平台上，也不能通过简单的代码转换来实现。目前在多平台 UI开发中，主要有以下几种技术：

- 为每一种平台分别开发相应的UI。利用各类开发工具，例如VB\JAVA\C++这样的面向对象语言可以方便的为各类平台分别开发相应UI。此技术的缺点在于多个目标平台间的UI的一致性难以保证。
- 开发一个单一的UI，此UI利用通用客户端（浏览器）或虚拟工具例如Java Swing，可以运行在多个平台上。

◆ 为UI开发单一的描述（文档）。此方法是第二类方法中的通用客户端方法的一种扩展。例如XML文档（配合CSS或者XSL样式表）。

面向对象语言（VB\JAVA\C++）用于界面开发特别自然，界面的元素显示地出现在界面上，有些语言具有界面生成器。这些语言的功能非常强大，但是对于开发适用于不同平台且支持重用的界面就有了不足，用户需求的变更往往体现在代码级的改变，界面的改变通常会引起许多其它模块的改变，可能带来大量代码的修改，甚至导致软件结构的重新设计。

有些研究基于桌面机的代码，将原来设计中因为平台的变化不再适应的部分通过一定的转换规则转换成新的实现方式，其中有分页、交互子的选择和转换、逻辑窗口和交互子的分布等。这些方法停留在语法层面的转换上，可能会产生出一些没有意义的或混杂的界面。

还有许多专门用于界面开发的工具和语言，如 UIML, XIML, DLL DIALOG, ROMA 等，能够将界面的定义和实际的渲染区分开，有些研究考虑到了适合不同上下文的界面的重用问题，通过现有的界面的集合运算（分解、合成、重组）来生成需要的新的界面，但是都没有从任务功能角度进行抽象来支持重用的考虑。

为了开发适用于多种平台的应用界面，需要进行更高层次的抽象，屏蔽各种设备的不同特性带来的变化。所以使用不同抽象层次的模型驱动方法是当前比较流行的界面开发方法。在模型驱动的方法中，使用较高层次的模型来描述用户的需求，通过模型的转换，逐渐加入平台信息，获得更加具体的模型，直到各种平台上最终界面的生成。在这个领域内，CAMELEON 模型是很多研究遵循的参考模型，Paterno 等人对与语言对应的基本表示元素进行抽象，这个抽象屏蔽了基本表示元素的各种具体实现方式的不同。使用这些基本表示元素和它们之间的时序关系来构建任务模型，生成每一个表示模型和对话模型。重用的是基本的表示元素，任务层次的语义在转换过程中已经逐渐消失，重用的层次不够。

本文工作从更高的层次上对完成任务的功能进行抽象，根据交互的特点，虽然交互的方式千差万别，但从完成任务的本质上是一致的，如可以抽象出在界面中为了完成交互经常使用的“选择”、“输出”、“对象编辑”等基本任务。并且发现完成这些任务的规则是一致的，即完成一个任务的方法、步骤和涉及的事件类型是相似的。这给了我们的研究一个启发，可以把这些基本功能作为研究的对象，

使这些功能能够支持各种平台特点的实现，再加上其它机制的支持，就可以解决多平台的界面的自动生成，支持迁移界面的开发，进而支持分布界面的生成。

我们把这些完成界面任务的基本功能定义为自适应的表示组件。自适应一方面体现在对各种数据实现的适应性，表示组件在在完成它的功能时，不关心被处理的数据对象，只关心处理的规则，使得完成一定功能的表示组件更具有通用型和一般性。另一方面表现在对平台的适应性，可以根据平台的信息和组件内部各种操作之间的倚赖关系，决定一个组件在具体平台上的实现和布局。开发人员可以使用这些自适应的表示组件来描述所需要完成的任务，不必关心平台的细节。

Fabio Paterno[1][2]等人在复杂界面迁移中，主要针对资源差异较大的平台间的界面迁移，将原界面分解并根据目标平台的特性进行重组；Murielle Florins[7]等人在将用户界面简化作为多平台系统设计方法中，讨论过另一种界面迁移情况，即如何将用户界面迁移到某些特定目标平台，例如输入设备为触摸屏（用手指对触摸屏操作），要求界面适合此输入方式。这两类方案中，目标平台都被限制在各自适合的范围内，因此并未对目标设备的资源进行评价。目前随着移动设备的不断发展，其计算能力、显示能力、输入输出能力已经与数年之前的移动设备不可同日而语，多数智能移动设备已经具备显示较为复杂的界面的能力，并有很多设备支持 Java 虚拟机等运行平台，这使得单个系统支持多个平台成为可能。这些平台具有相当程度的计算和输入输出能力，但其交互手段与桌面计算机有许多不同。本文基于目前设备种类繁多、特性各有不同，且拥有较为丰富资源（与前几年的设备相比较）的情况下，尝试提出一个资源表示和评价方法，并采用小幅度调整用户界面的方法来使界面适应存在资源差异的各类平台。

2.4 本章小结

本章介绍了 MVC 和 MVP 软件开发模型，以及界面开发中需要关注的一些问题。针对目前多平台界面开发现状，提出了一个基于自适应组件的多平台界面模型。

3 表示层模型

将界面的描述与实现分离后，对界面生成相关的元素进行抽象，该抽象描述将各种平台的实现细节隐藏，提供统一的抽象界面描述方法，开发人员就能够使用统一的抽象界面描述方法来描述任务，而不必考虑各个平台间的差异。同时，采用自适应表示组件自动生成平台相关的界面。大多数应用程序的表示层对于该应用程序的成败常常都是至关重要的。表示层实际上代表了用户和应用程序其余部分之间的接口。如果用户与应用程序的交互方式不能使他们以高效和有效的方式执行自己的工作，那么应用程序在总体上的成功将大打折扣。

表示层模型主要通过对数据对象、事件路由和抽象界面描述的配置，来实现数据映射、事件映射和界面映射，然后经过界面生成器的处理转换，最后完成界面的自动生成。在我们的表示层模型中，主要涉及抽象界面模型、表示组件模型、数据模型等几个具体模型，以及完成模型之间转换的组件选择事件分发模块和具体界面生成器。另外，还有支持表示模型和应用语义连接的表示层控制器、业务处理单元和统一数据网关等部分，见图 3-1。

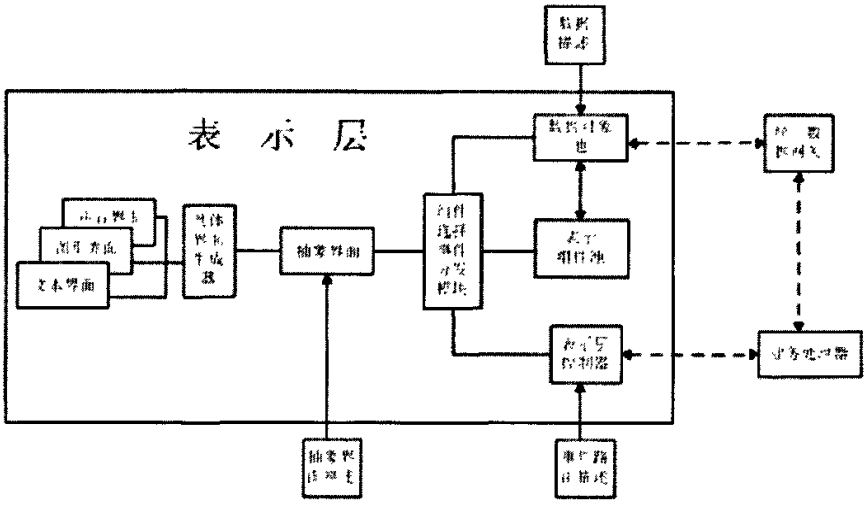


图 3-1 表示层模型

首先介绍使用到的几个模型的功能和描述，来说明它们在整个表示模型中的作用。

3.1 抽象界面

抽象界面是对具体用户界面的抽象，不考虑界面的具体实现，从概念层次上考虑完成一个界面功能所需要的任务划分，决定每个任务对应的表示组件及其操作的数据对象。另外描述这些表示组件之间的时序关系和数据交换关系，提交给具体界面生成器，用来决定这些表示组件在界面上的布局和出现顺序。表示组件只关心完成的功能、操作的数据对象和处理的事件，不关心组件的具体实现方式。抽象界面模型可以通过（表示组件集合，表示组件之间的依赖关系，数据集合，表示组件和数据的映射关系，界面事件）来描述。

其中，表示组件集合是完成这个界面功能所需要的所有表示组件的集合，主要描述了完成一定的界面功能的功能划分。

组件之间的依赖关系主要指组件之间的时序关系。时序关系给出了完成一个功能所需要的各个组件之间时间上的先后顺序关系；

数据集合是完成界面功能所使用到的所有数据对象的集合，一个界面操作的所有数据放到一个共同的数据对象池中，各个组件之间共享这些数据，通过组件和数据之间的映射和组件之间的时序关系来完成数据在各个组件之间的流转。界面的数据集合由界面初始化模块加载，由统一数据网关同步；

组件和数据的映射关系用来描述组件所使用到的数据，为组件和数据对象之间建立映射来，如下列代码描述了一个数据列表组件，第一行中的代码“metadata=″client.client″”表示组件 EventTable 与数据表 client.client 建立了数据映射。

```
<container id="container" fieldrows="9" fieldcolumns="8">
<table id="EventTable" title="客户列表" metadata="client.client" fieldrows="1" >
  <column id="name" title="客户名称" editable="false" width="150" />
  <column id="address" title="地址" editable="false" width="150" />
  <column id="phone" title="联系电话" editable="false" width="150" />
  <column id="typeid" title="客户类型" editable="false" width="150">
  <assist id="typeid" title="" refmodel="event.ArchRefModel" enable="true" />
  </column>
</table>
```

界面事件描述整个界面中引起界面变化的事件以及需要转发到各个组件的事件。这些事件按照不同种类，提交给表示层控制器，由表示层控制器根据不同的分类进行相应的处理。

3.2 表示组件

基本的表示组件是界面上的基本输入输出交互元素。按照类型分为如下各类：Text/文本框、password/口令输入框、combobox/下拉框、int/整数、decimal/浮点数、date/日期、datetime/时间、radio/选择框、checkbox/多选框、file/文件、color/颜色等。另外，还有其他特定的输入/输出组件，包括 voice /声音、picture/图像等。而自适应表示组件由基本表示组件构成，对应界面上一个相对独立的功能区域，完成一个基本的交互操作，如“选择”、“编辑对象”等。自适应表示组件根据所完成的功能，定义对应的事件集合和内部的操作序列；根据运行环境的变化，决定以合适的交互方式完成数据的输入、输出和事件处理功能。

自适应表示组件模型可以使用（ID，功能，平台信息，数据对象，事件集合）描述。

其中，ID 是组件标示符；

功能描述自适应表示组件所完成的交互目标，如“选择”，“编辑对象”等；

平台信息描述表示组件运行的环境。自适应表示组件对平台信息是自适应的，它可以根据接收到的平台信息和操作的数据对象来决定自己完成功能所需要的具体的表示形式。如编辑一个对象集合“表”，在桌面机上可以采用表的形式来编辑，在手持移动设备上由于显示空间的不足，不能采用同样的方式，只能采用分页的方法，每一个页面编辑一个对象，同时在这些页面之间自动建立导航关系；

数据对象是指自适应表示组件所要显示和处理的数据；

事件集合是自适应表示组件为了完成它的功能所涉及的相对固定的事件集合，如“编辑对象”组件为了完成功能定义了一组与功能相关的事件“增加”，“删除”，“修改”等，同时完成这些通用事件和具体数据对象操作的映射关系；

表示层数据对象模型描述完成界面功能所使用到的数据，是自适应表示组件处理的数据模型。主要分为两种类型，一种是实体，对应着简单的对象；另外一种是实体集合，对应着一组对象的集合。表示组件处理的对象的值采用同样的（key, value）对的方式存放到数据对象池中，对象的各个属性的类型以及属性之间的关系在统一数据网关内来描述，表示组件根据属性的类型来决定如何显示这

些值。数据由统一数据网关同步；

3.3 控制器

控制器分为两个部分：表示层控制器和业务处理单元。

表示层控制器：是表示层的主要控制部件。定义所有界面之间的逻辑关系，完成一个界面内部的功能：初始化表示组件，控制它们的生命周期；负责来自于表示组件和业务处理单元的各种事件的采集、分类、处理。

业务处理单元：负责后台业务的处理，接收来自表示层控制器的事件，根据业务内部的规则，调用相应的模块进行处理。处理完毕后，将结果数据返回给统一数据网关，将应用产生的中间事件返回给表示层控制器。

3.4 统一数据网关

统一数据网关不考虑数据本身的含义，将数据的类型、属性、操作、参数等按一定格式进行存储。各类异构数据，如来自于不同的数据库、e-mail、PDF、应用程序的中间结果等数据，采用相同的方式来表示数据。采用树状结构描述数据对象的属性和操作，如图 3-2，数据对象通过一个三元组（属性，操作，参数）来统一定义。其中，属性类型供表示组件使用，用来决定如何显示相应的属性值；操作也可以作为访问的内容，支持在表示组件内部对数据的操作的显示，同时用作和表示组件的通用事件建立映射；参数用来描述数据对象和不同数据源存取格式之间的关系。

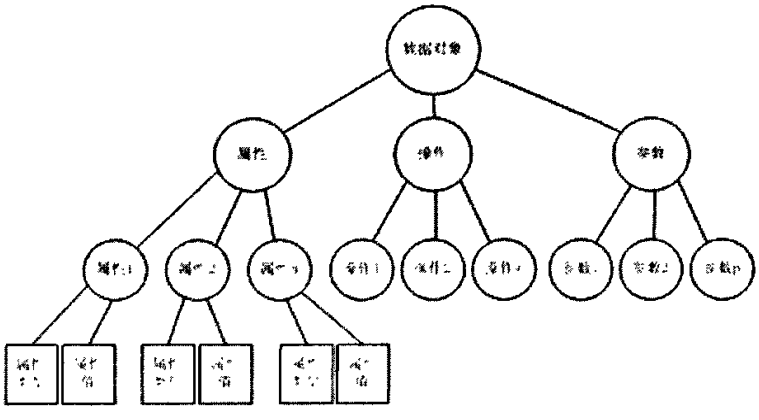


图 3-2 统一数据网关中的数据模型

3.5 本章小结

本章介绍了表示层模型的结构，解释了模型各部分（抽象界面、表示组件、控制器、统一数据网关）的具体含义。

4 模型的工作过程

上一章介绍了表示层模型的构成情况，现在我们来了解一下这个模型是如何工作，并实现界面的自动生成的。该表示层模型是通过对数据对象、抽象界面描述和事件路由的配置，分别产生数据描述、抽象界面描述和事件路由配置描述，并在上述描述中分别建立数据映射、界面映射和事件映射。在配置完成后，根据映射关系等相关信息，确立数据来源和事件处理规则，生成具体界面。

4.1 模型的工作过程

1) 配置抽象界面（确定所需要的抽象表示组件以及这些组件之间的依赖关系，和所操作的数据对象）；
2) 配置事件路由（表示层控制器对每一个事件的响应）；
3) 配置数据对象（建立数据对象池和统一数据网关之间的映射，将表示组件处理的数据按照（key, value）对的形式存放到数据对象池中）；
4) 界面调整（表示组件按照完成的功能，运行的平台信息，数据对象的配置中的属性个数和属性类型，确定组件的表现形式和组件内部的布局）；
5) 界面生成（生成器根据抽象界面描述和平台信息，确定整个界面的时序关系和界面的布局）。

工作流程如图 4-1 所示：

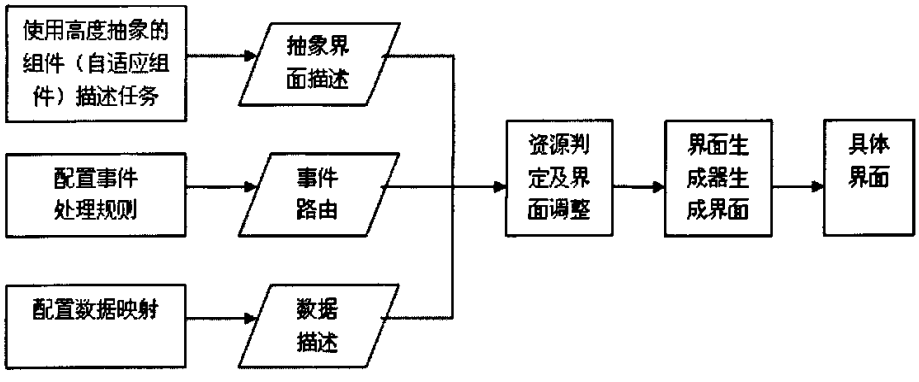


图 4-1 模型的工作流程

4.2 模型的配置

配置数据对象。数据描述的作用是设置数据的来源、格式等相关信息：

（对象名；

属性 1，类型；

属性 2，类型；

……

属性 n，类型；

对象索引（给出对象存放的地址）

）

配置抽象界面，即抽象界面的描述，它对所用到的抽象表示组件的功能以及所涉及的对象和事件等内容作出描述：

（抽象表示组件 1

功能；

对象 1，对象 2…

事件 1，事件 2…

抽象表示组件 2

功能；

对象 1，对象 2…

事件 1，事件 2…

……

抽象表示组件 k

……

依赖关系

抽象表示组件 1.事件 enable 抽象表示组件 2

……

事件

事件 1；

事件 2；

……

)

配置事件路由，即设置界面相关组件事件响应规则及内容：

(抽象界面事件 1：

抽象界面事件 2：

……

抽象界面表示组件 1. 事件 1：

……

抽象界面表示组件 2. 事件 1：

……

)

4.3 建立映射信息

通过建立统一数据网关和数据对象池两种模型之间的映射，来完成的数据转换。映射关系在配置数据对象时开始建立，映射主要描述数据对象池中的（key, value）对和统一数据网关中的数据属性之间的对应关系。反过来，数据池中的（key, value）对可以按照同样的对应关系存放到统一数据网关中的数据对象中。

通过抽象界面的功能划分选择相应的表示组件，同时建立表示组件通用事件和数据对象操作之间的映射。

抽象界面和具体界面之间的具体界面生成器主要将抽象界面内部的抽象表示组件之间的依赖关系描述出来，根据具体界面的模态和运行环境，生成具体的界面模态和布局。同时，建立抽象界面的事件和具体界面事件实现的对应关系。

表示层控制器的事件分发，通过事件路由配置来完成。抽象界面内部涉及的所有事件按照规则传递给表示层控制器，根据规则，抽象界面上的事件分成三种主要类型：界面本身的事件，组件之间的事件和组件内部的事件。这些事件的处理响应，有的可以通过抽象界面的描述直接得到，不能得到的首先由系统根据自检的原则设置缺省值，具体运行时通过对表示层控制器的事件路由设置完成。

4.4 模型实例

现在我们通过一个简单的客户名单管理的例子来说明上述模型的工作过程。首先从一个客户表中选择一个对象，然后对某个客户信息进行编辑。这需要两个表示组件“选择”和“对象编辑”和两个对象“全体客户”、“客户”来完成。工作的起始点是抽象界面和数据对象的描述：

4.4.1 抽象界面的简化描述

```
抽象界面＝（选择组件
            选择
            全体客户
            “选择”、“保存”、“取消”
          对象编辑组件
            对象编辑
            客户
            “修改”、“取消”
         //设定时序关系：选择客户后，才能进行修改，否则编辑组件无效
         //参见附图 8-1、附图 8-2
          选择组件.选择  enable  对象编辑组件              ）
```

4.4.2 数据对象的简化描述

```
全体客户＝（名字＝“全体客户”
           属性   all
           Index ＝ 统一数据网关的地址
         ）
客户     ＝ （名字 ＝ “客户”
           属性  all
           Index ＝全体客户【index】
         ）
```

4.4.3 控制器的简化描述

由抽象界面的描述，可以自动获得表示层控制器的描述

表示层控制器＝（初始化选择组件

选择组件.选择： 初始化对象编辑组件

选择组件.保存：

选择组件.取消：

对象编辑组件.修改：

对象编辑组件.取消：

）

同时，也可以通过组件选择事件分发模块选择两个表示组件“选择”和“对象编辑”，将对应事件分发到各自的表示组件中。表示组件根据获得的这些信息和运行平台的信息，决定用合适的方式来完成自己的功能。

具体界面生成器可以获得两个组件之间的时序先后关系，即“选择组件”的出现事件先于“对象编辑”组件。具体界面生成器根据平台信息选择合适的方式来描述这个关系。

4.4.4 部分关键代码

以上几个描述在具体实现中的部分关键代码如下（具体代码参见附录8.2）：

抽象界面描述：

```
<?xml version="1.0" encoding="GB2312" ?>
<page title="Meta_use_case1" size="500;400">
  <!-- 事件映射，描述事件和页面上控件的对应关系，以及事件的类型-->
  <event_map javacode="event.Meta_use_case1Event">
  <event id="modify" address="head.modify.actionPerformed" />
  </event_map>
  <!-- 页面上下文，描述数据和页面的对应关系，以及初始化数据-->
  <context />
  <!-- 页面视图描述，描述视图上模版的布局-->
  <head id="head">
     <button id="modify" text="修改" />
  </head>
```

```
<body id="body">
    <table id="EventTable" title="客户列表" metadata="client.client">
        <column id="name" title="姓名" editable="true" width="150"/>
        </column>

        <column id="address" title="地址" editable="true" width="150"/>
        </column>

        <column id="phone" title="电话" editable="true" width="150"/>
        </column>

        <column id="typeid" title="类型" editable="true" width="150">
        <!—此项数据可辅助录入，即可以从相关列表 gw 内选择适当内容-->
        <assist id="typeid" title="gw" refmodel="event.ArchRefModel"
        enable="true" />
        </column>
    </table>
</body>
</page>
```

表示层控制器处理过程，即事件处理：

```
public void processUIEvent(Page page, PageEvent ev) {

    try {
        //选择
        if (ev.getId().startsWith("select")) {
            select(page);
        }
        //保存
        else if (ev.getId().startsWith("save")) {
            save(page);
        }
        //取消
        else if (ev.getId().startsWith("cancel")) {
            cancel(page);
        }
        //处理其他情况
        else {
            default(page);
            }
        }
        //错误处理
        catch (Exception e) {
```

```
            e.printStackTrace();
            JOptionPane.showMessageDialog(null, e.getMessage(), "错误",
            JOptionPane.INFORMATION_MESSAGE);
        }
}
```

4.5 本章小结

本章简要介绍了表示层模型的工作过程，描述了数据对象、抽象界面描述和事件路由的配置方法，并通过部分代码加以说明。

5 自适应表示组件

在我们的表示层模型中用到了表示组件，自适应的表示组件则是一种增强的表示组件，它能更好的满足我们的要求。自适应的表示组件包含两个层面的含义：在指定的平台上运行时，不区分处理数据的语义，按照同样的规则显示数据对象池中的数据，并且有自动布局的能力；在不同平台间运行时，保持不同平台之间的实现上的一致性。

界面是由若干功能区域组成的。功能区域具有很大的相似性。比如对象编辑、表格、查询、高级查询、树等，就是常见的功能区域（参见附图 8-5、附图 8-6）。表示组件对应着一个界面或界面的一部分，是完成界面自动生成的重要环节。本文提出了自适应的表示组件的概念，自适应组件就是这些功能区域的基本实现，它由最基本的界面元素组合而成。

5.1 平台内的自适应性

对于自适应表示组件而言，它有能力处理如下的不确定性：

数据模型不确定性。每个自适应表示组件在运行时期，总是对于一个或者多个元数据进行界面管理的，这些元数据代表着该组件的服务/操作对象，但是，一个自适应组件在运行之前，他并不知道自己绑定的、或者对应的元数据。

所包含界面控件的不确定性。由于自适应表示组件在运行事先并不知道所管理的对象，也不知道该对象包含哪些属性，因而，也不知道自己应该包含哪些种类的哪些基本控件来满足数据显示、数据输入的要求。但是，自适应表示组件总是包含不特定多个各种类型的基本控件的。它在运行时刻，首先会根据配置文件初始化自己。

位置布局的不确定性。自适应表示组件在运行初始化的时候，才知道自己在页面中的位置，以及所包含的各个基本控件的布局方式。这些信息都是在配置文件中给定的。

支持事件的不确定性。对于某种类型的自适应表示组件，其所支持的事件是基本确定的，但是，在该类组件的运行实例中需要支持哪些事件，是事先不知道的，需要在运行初始化的时候，从配置信息中获得。

表示组件根据配置文件到统一数据网关内读取用户所定义使用的这些属性的类型，根据属性的类型来决定采用哪些基本的交互子来显示每一个属性；根据用户输入的顺序决定属性的显示顺序；根据平台信息和属性的个数以及属性的显示权值来决定组件显示时的布局。

自适应的组件对显示的内容不加区别，主要是对数据对象池中内容的显示。对象数据按照（key，value）对的形式组织，（key，value）对和数据的对应的属性和属性值之间使用映射机制根据用户输入的数据描述来建立。这种映射可以是全部属性的映射，也可以是部分属性的映射。由用户根据需要决定那些参与交互的属性。表示组件只是将对象的这些（key，value）对按照顺序和相应的类型来显示。这种处理使得组件显示时不关心显示内容的语义，可以使用同样的方法来显示所有的对象数据。

如果用户需要修改显示结果，如将显示的属性由 10 个改为两个，则只需要修改对象的描述文件，将原来对象的 10 个属性简单地改为两个即可，不需要作任何其它地改动。表示组件的显示已经根据属性和（key，value）对的映射机制自动改变。

如果用户显示的内容由个人信息改变成了产品信息，只需把界面描述中组件操作的对象由个人信息对象改变为产品信息对象就可以实现。

表示组件对输入的支持机制

对于界面而言，输入行为是其重要行为，用户需要输入的包括数据、操作两种类型的信息。对于数据的界面录入方式，可以分为如下几种：

[1] 用户直接录入：用户直接输入原始的、基本信息。之所以称这些数据是原始的，是指，这些被输入的基本数据，原本没有被保存在信息系统中，或者本系统中。用户只能手工逐项输入这些数据信息。这些基本信息可以被划分为若干不同的类型，比如字符串、整数、浮点数、日期等。参见附图 8-3 中前三项信息的输入。

[2] 辅助选择性录入：所谓辅助选择性录入是指，所要输入的数据并非是绝对原始的，而是原本存在于本系统中，但是，需要用户从中选择确定相关的数据项。参见附图 8-3、附图 8-4。

[3] 相关性自动录入：是指，用户需要输入的某些字段属性并非孤立的，而

是与本次输入任务中已经输入的某些字段属性相关的，根据相关性逻辑，就可以确定该字段属性的值。也就是说，这类字段的值不需要用户再次手工输入，系统可以根据相关性逻辑自动计算。

[4] 数据源导入：是指，用户需要输入的数据信息，原本已经存在于本系统之外的其他系统中，特别地，数据库中，需要导入到本系统中。

[5] 文件数据导入/引入：是指，用户需要输入的数据信息，原本已经存在于文件系统中，需要导入到本系统中。

对于上述输入过程，除了最后一种文件数据的导入/引入之外，对于前四种，SmartBiz 都实现了语义无关的输入：

用户直接录入

用户通过与基本控件的交互，输入相关类型信息。基本控件本身并不需要了解所输入数据的语义。模板是由不特定多个基本控件组成的，当各基本控件直接输入完数据之后，自适应模板将各个基本模板的数据，自动收集到元数据模型中。

辅助选择性录入

注意基本控件中的 assist，它是一种特殊的控件，他实际上实现的不是用户的直接输入。当用户选择该控件的时候，该控件显示来自后台的数据列表，用户在这些列表中选择一个/几个，作为该项的输入。我们称这种方式为辅助录入/参照录入。assist 控件的基本数据模型是元数据，其实现机制决定了，这种输入方式是语义无关的。Assist 的工作依赖于配置文件，事先他并不知道所要选择的元数据是什么，或者也不知道所要选择的对象集合是什么。也不知道要返回给被录入字段的类型是什么，所有这些都是通过配置文件确定，并且在运行时刻才起作用的。

相关性自动录入

当某些字段属性的数据被确定的时候，系统根据相关性规则表达式，自动计算相关字段的数值，完成录入过程。由于采用了自动表达式计算机制，并且参与计算的各个变量来自于元数据的不同属性，因此，该自动录入过程是一种语义无关的录入过程。例如，一个人的生日信息可以从身份证号码中得到。实例如下（第 3 行中生日字段的信息来自身份证的右数第三到第 8 字节）：

```
<EventMap>
    <EventDesc>
```

```
        <Event ID=" " Src = " Page.Body.dataedit. IDNo.ValueChange " >
        <!生日信息 Detail.Birthday 从身份证号 Detail.IDNo 中取得>
        <Process  "Detail.Birthday = SubStr(Detail.IDNo,3,8)" />
    < EventDesc/>
</EventMap>

    <dataedit id ="detail" title="详细信息" columns="2" rows="8" metadata ="client">
    <text id="name" title="客户名称" enable="true" maxLength=" 32" />
    <text id ="address" title="地址" enable="true" />
    <text id=" zone" tile=" 地区" />
    <text id ="IDNo" title="身份证号" enable="true" />
    <date id ="birthday" title="出生日期" enable="true" />
    <assist id="typeid" title="客户类型" refmodel="ArchRefModel" enable="true" />
    </dataedit>
```

数据源导入

严格地说，数据源的数据导入过程，并非界面过程，而是后台过程。对于不同数据源之间的数据导入过程，以及导入过程中的转换、过滤等处理过程，系统都可以做到语义无关的、通用的、统一的支持。

表示组件通过对所有（key，value）对的值拷贝到数据池中来接受用户对属性值的修改，然后将数据池中的数据和数据网关中的值定期进行更新。

5.2 事件处理机制

组件根据完成的功能来确定一个事件的集合，用户根据需要来选择这个集合的一个子集或全部。对事件的处理主要包括事件的采集和分类，然后根据相关协议将这些事件传送到表示层控制器，由表示层控制器根据内部的规则来确定如何处理这些事件。

5.3 自适应组件的重用机制

在不同的设备平台上，已经由许多成形的组件来支持特定平台的开发。为了有效地利用这些现有的资源，我们将不同平台上完成同样功能的具体表示组件集中在一起，通过语义进行更高层次的抽象，这样就支持了一个功能在不同平台的不同实现，对抽象界面而言，屏蔽掉了各种实现的细节，使开发人员只是关注于

任务功能的划分，而不必考虑运行环境的细节。

5.4 平台间的自适应性

一个表示组件需要支持各种平台，并且保证各种实现的一致性，主要依靠两个机制。第一个是界面的拆分机制，在桌面机上可以完整显示的对象，不一定能在目标设备上完整显示，需要拆分为多个部分来显示。这时为了避免硬性划分带来的问题（硬性划分可能会产生一些无意义的界面），在对象数据内部的属性之间可以描述某些属性之间的逻辑关系，如某些必须放到一起显示的属性可以使用Group来描述它们之间的逻辑关系等。

第二种机制是界面调整机制。同样的功能可以由不同的组件来实现，例如输入一个文本信息，既可以用文本输入框，又可以用列表选择框，如果该信息有特定的特点，例如是地理位置信息，还可以采用地图选择式输入。另外，界面的布局也对显示有一定的影响。

随着相关技术的快速发展，移动数字设备的计算、存储、显示、输入能力均有了质的飞跃。移动平台各项资源虽然仍无法与普通桌面平台相比，但是其各项指标均已达到较高水准，其实际意义在于，相比以前几乎所有程序都必须重新开发才能在手机上运行，目前部分程序在高端硬件的支持下，已经可以通过虚拟机的支持在移动平台上运行。但是由于移动平台的特殊性，跨平台的应用不仅受到资源不足的困扰，更是受到平台间设备资源差异的巨大影响。举例来说，普通的输入界面如果想在使用触摸屏的移动设备上使用，必须考虑到触摸屏操作精度低的特点，需要对界面的组件布局等各项内容作出调整，以此来实现组件乃至界面的自适应。在这种情况下，我们提出了一个基于资源评价的界面调整方法，来解决这个问题。

5.5 本章小结

本章提出了自适应表示组件的概念，并对其自适应的特性做了描述，分别介绍了平台内和平台间的自适应特性，以及自适应组件的事件处理机制和重用机制。

6 基于资源评价的界面调整

在上文中提到，自适应组件能够适应不同的平台。我们提出了一个基于资源评价的界面调整方法，在此过程中对无法在目标平台使用的组件进行调换，来实现组件的跨平台自适应特性。方法的思想是为界面及界面中的各类元素加入一个描述其资源需求的数据结构，当界面需要迁移时，使用评价函数，对资源需求列表与目标资源作对比计算，根据计算结果，评价目标设备是否能满足界面对资源的需求。并且根据评价结果，尝试使用低资源需求的组件替换资源需求得不到满足的组件，并对界面布局等做出调整。整个评价过程如图 6-1 中所示。

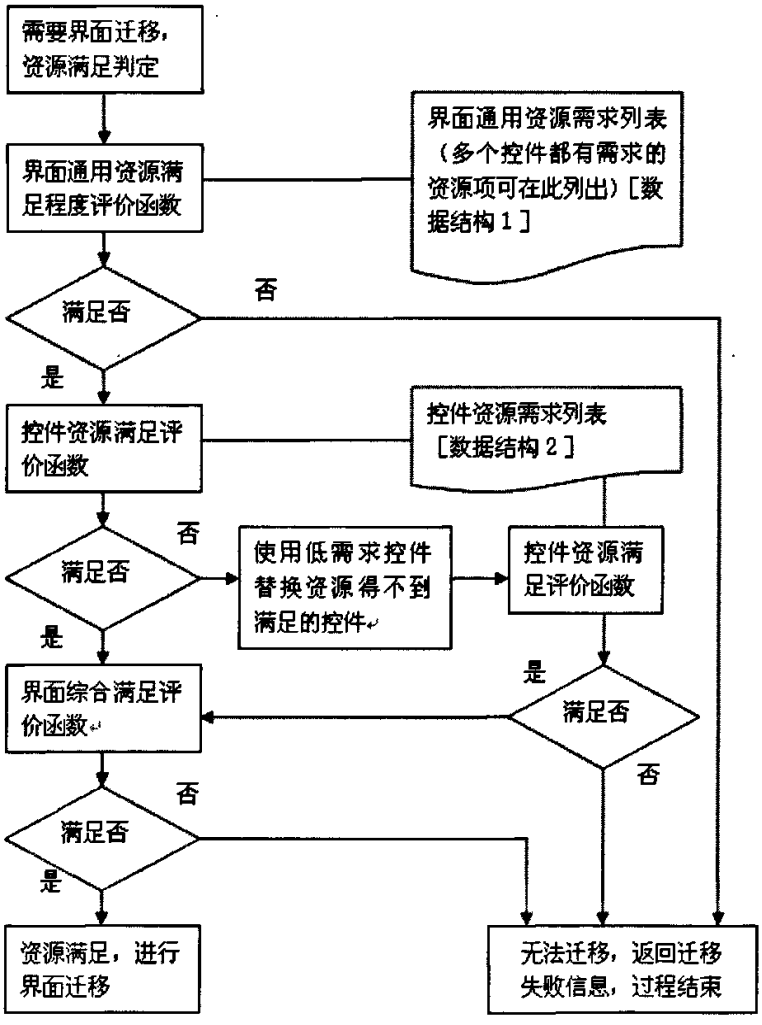


图 6-1 界面资源评价过程

6.1 数据结构及分组规则

6.1.1 建立数据结构

为了描述界面及其组件的资源需求情况，我们建立了如下几个资源需求列表，其结构基本相同，在下文中有具体说明。

界面（通用）资源需求列表（图 6-1 中的数据结构 1）。这一数据主要用来描述整个界面中多个控件共同需求的某些资源项，例如对屏幕分辨率（而不是可用屏幕区域大小）的需求，对操作系统的需求等，其目的是减少数据结构 2 中的资源项的数量，简化计算过程。

控件资源需求列表（图 6-1 中的数据结构 2）。此数据用来描述每个控件对资源的需求情况，即每个控件应当有一个资源需求列表。因此，如果多个控件对某项资源均有需求，把该需求写入数据结构 1 中，能大大减少该资源项在控件资源需求列表中出现的次数（同时简化了后边的控件资源满足情况判定过程）。

6.1.2 对资源进行分组

移动设备跟桌面设备相比，各类资源都十分有限。移动设备资源多种多样，例如显示器、CPU、内存、外存、键盘、触摸屏、操作系统等等，但是他们对控件运行所起的作用无法用一个简单的需求比例来描述，举例来说，合适的操作系统是控件运行的必要条件，而键盘和触摸屏不能算是必要条件，但是输入设备多数情况下也是必不可少的，可能并非同时需要这些输入设备，但至少需要其中一个。如何描述这些资源，对我们能否更好的判断出资源是否满足控件运行的需要来说，显然十分重要。通常这类资源是按照表 6-1 中的格式描述的（表 6-1 见下页）。

如果我们按此格式描述资源，并对资源满足情况进行判断，就会有一些潜在的问题难以解决，比如，如何判断键盘、鼠标、触摸屏等的需求是否被满足，满足到什么程度。因此，我们需要一个更好的资源需求描述方法。

经过仔细分析，我们发现，各种资源在必要性和需求程度上各有不同，例如，2 种操作系统只要有 1 项就可以完全满足需求，而键盘、鼠标、触摸屏等设备资源的需求情况则复杂的多，有些可能只要其中一项就可完全满足需求，有些可能

需求其中的两项或者 3 项才能完全满足需求，另外一些则可能必须需求其中几项，其他项有则更好满足需求。正是这种需求的复杂性，给我们描述资源带来了很多困难。

资源类型	资源需求	是否必须	说明
显示器分辨率	320*240 到 640*480	必须	320*240，满足程度 50%；640*480，满足程度 100%；
尚未使用的屏幕空间大小	120*100	必须	120*100，满足程度 100%；
操作系统	WinCE1.0 或 Palm2.0	必须	有这 2 种操作系统中的任意一种，满足程度 100%；
可用内存	1MB	必须	1MB，满足程度 100%；
键盘		非必须	满足程度 40%；（键盘+鼠标，满足程度 100%；）
触摸屏		非必须	满足程度 80%；（键盘+鼠标，满足程度 100%；）

表 6-1：一个输入日期的控件 Component1 的资源需求表

考虑到各种资源的共性，我们提出一个对资源进行分组的方法。下面我们来具体说明这个分组方法的细节，以及它为什么要把若干资源合并为一类的原因。

分组规则

为了能严谨的对资源进行讨论和分类，我们首先对分组方法提出以下要求，并对这些要求作出解释：

（1）资源的分组应该没有重复，即一种具体资源只能出现在一个分类中。这个要求比较容易理解，在此就不再多做讨论。

（2）分类应该覆盖到控件所需求到的所有资源，即需求的资源必须出现在某个分类中。这一要求保证我们的分类不会漏掉某种资源。

（3）如果某种资源是必须的，也就是说控件的运行必须要求该种资源，则

其应当单独分组。这一要求是为了便于计算资源的满足情况，具体作用见后面资源满足情况求解函数。

（4）可以互相替代的资源分为一组，对各种资源的需求以百分比方式列出。这一要求的主要作用是将可以互相替代的资源视为一种资源，参与最后需求满足值的计算。

（5）分组内各项资源的关系在分组名称后列出。这一项主要说明了如何计算该组的满足值。例如，同时支持 WinCE 和 Palm 操作系统的控件，只要满足其中一项，该组的满足值就应该为 100%；而对于键盘和触摸屏，虽然有其中一项就能满足基本需求，但是很可能两项同时满足能得到更好的效果。

（6）每个分组都是该控件必须的，即每个分组的满足值都应当大于零。分组内只要有一项得到满足，该组满足值都不应为零。

分组后的数据结构

在要求列出后，我们的资源分组表示结构如表 6-2 所示：

分组名称	组合方法	子项名称	所需资源类型	所需资源属性	满足程度
		…	…	…	…

表 6-2：资源分组表示结构

其中，每个分组中可能有 1 到多个子项，其数目可变。满足程度是指若目标设备可满足该项资源需求时，该分组的整体资源满足程度，它应当是界面设计者给定的值，或者对于某一特定的编程环境（如Ｊａｖａ，ＶＢ）而言，各类控件的资源需求可以是一基本固定的列表。组合方法则是指，如何由多个子项的满足程度值求得该分组的满足程度值，本文中使用到了 MAX 和 ADD 两种方法，如果有需要，还可加入其它函数，其开放性能够保证分组方法的扩展能力。关于 MAX 和 ADD 方法的具体含义如下：

函数 MAX 的定义为：

MAX（X1，X2，X3，…）=参数（X1，X2，X3，…）中最大的一个

函数 ADD 的定义为：

ADD（X1，X2，X3，…）=X1 + X2 + X3 + …，当 X1 + X2 + X3 + … <1

或

ADD（X1，X2，X3，…）=1 ，　　当 X1 + X2 + X3 + … >=1

分组后的数据结构（例 1），按照该分组描述格式， Component1，其资源需求描述如表 6-3 所示：

分组名称	组合方法	子项名称	所需资源	所需属性	满足程度
分组 A	MAX	A1	屏幕分辨率	320*240	50%
		A2	屏幕分辨率	640*480	100%
分组 B	MAX	B1	操作系统	WinCE1.0,1	100%
		B2	操作系统	Palm2.0,1	100%
分组 C	MAX	C1	可用内存，	1MB	100%
分组 D	ADD	D1	键盘	有键盘（以下记为 1）	40%
		D2	触摸屏	1	80%

表 6-3：Component1 资源需求分组描述表

从此例子容易看出，四个分组 A、B、C、D 分别是屏幕、操作系统、内存、输入设备四类资源。在需求描述相对简单的情况下，我们的分组方法与传统的按资源类型描述方法趋于一致。

分组后数据结构实例 2

有的时候，某些资源的需求情况比较复杂，这时，就无法简单的把相同类型的资源分为一组。如例 2，Component2,一个小游戏控件，用键盘方向键负责控制人物前后左右移动，鼠标负责改变人物面对的方向，如果用触摸屏代替鼠标，则其只能达到鼠标一半的效果。在本例子中，因为键盘必须满足,，而鼠标和触摸屏则至少具有一个，因此，我们将其分为了 2 组，可以互相替代的 2 项分到一个组中，而把不可或缺的键盘单独分组，这正是我们前边所提的第三条要求。因为若将其与 E 组合并，则整个分组的满足值求解将变得复杂，如果不这样分组，在项数较大的情况下，分组的满足值将无法用简单的操作来描述。因此，其资源需求列表如表 6-4 所示：

分组名称	组合方法	子项名称	所需资源	所需属性	满足程度
分组A	MAX	A1	屏幕分辨率	320*240	80%
		A2	屏幕分辨率	800*600	100%
分组B	MAX	B1	操作系统	WinCE1.0	100%
		B2	操作系统	Palm2.0	100%
分组C	MAX	C1	可用内存	16MB	100%
分组D	MAX	D1	键盘	1	100%
分组E	MAX	E1	触摸屏	1	50%
		E2	鼠标	1	100%

表6-4：Component2资源分组描述表（例2）

前边所述的几个需求列表，均采用这种分组方法来描述具体需求信息。

6.2 资源需求满足程度计算方法

6.2.1 定义满足函数F（X）

我们把F（X）定义为一个重构函数，即X可以是整个界面，可以是界面上的一个组件，也可以是组件数据结构中的一个数据分组，甚至是分组中的一个资源子项。这样我们就可以使用F（X）统一的表示X的需求在目标设备中得到满足的情况（但是当X类型不同时，其实际求解函数是不同的）。

F（X）=X的资源需求得到满足的程度值（在下文中我们简称为X的满足值）

6.2.2 定义一个目标平台

现在，只要给定一个具体的移动设备平台，每个分组的资源满足情况都可以很容易的求出，因为每个子项的满足值已经给出，而如何利用每项的满足值，求得该组的满足值，其方法也已经给出（即例子中的组合方法），为了详细的说明F（X）的定义及计算方法，我们定义一个目标设备智能手机M1，参数如表6-5所示：

屏幕分辨率	内存	操作系统	Java	键盘	触摸屏	鼠标
640*480	32M（用户可用内存 16M）	WinCE2.0（向下兼容 WinCE1.0）	支持	有	有	无

表 6-5：智能手机 M1 参数表

6.2.3 当 X 为子项时的满足值求解函数定义及实例

当 X 为子项时，F（X）求解如下：

F（子项）=数据结构中该子项的满足程度值（当该子项所需资源被满足时）

或

F（子项）=0（当该子项所需资源不被满足时）

例 2 中的子项满足值求解如表 6-6 所示：

分组名称	组合方法	子项名称	所需资源	所需属性	满足程度	目标设备	子项满足值
分组 A	MAX	A1	屏幕分辨率	320*240	80%	640*480	F（A1）=80%
		A2	屏幕分辨率	800*600	100%	640*480	F（A2）=0%
分组 B	MAX	B1	操作系统	WinCE1.0	100%	WinCE2.0	F（B1）=0
		B2	操作系统	Palm2.0	100%	WinCE2.0	F(B2)=100%
分组 C	MAX	C1	可用内存	16MB	100%	16MB	F(C1)=100%
分组 D	MAX	D1	键盘	1	100%	1	F(D1)=100%
分组 E	MAX	E1	触摸屏	1	50%	1	F（E1）=50%
		E2	鼠标	1	100%	0	F（E2）=0

表 6-6：例 2 中的子项满足值求解示例

6.2.4 当 X 为分组时的满足值求解函数定义及实例

当 X 为分组时，F（X）求解如下：

F（分组）=组合方法函数（F（子项1），F（子项2），F（子项3，）…）

我们将前边所求得的子项满足值代入该函数，可以求得：

分组A的满足值

F（A）=MAX（F（A1），F（A2））

=MAX（80%，0）

=80%

分组B的满足值

F（B）=MAX（F（B1），F（B2））

=MAX（100%，0）

=100%

分组C的满足值

F（C）=MAX（F（C1））

=MAX（100%）

=100%

分组D的满足值

F（D）=MAX（F（D1））

=MAX（100%）

=100%

分组E的满足值

F（E）=MAX（F（E1），F（E2））

=MAX（50%，0）

=50%

6.2.5 当X为控件时的满足值求解函数定义及实例

在上文中我们已经求可以求得分组的满足值，那么，只要我们合理的将每个分组的满足值进行组合，就可以得到整个控件的满足值。同样的，也可以得到整个界面的满足值。

当参数X为控件时，函数F（X）应该具有的属性

在已经求得分组满足值的条件下，如何求得控件的满足值呢？我们首先想到

了常用的求平均数函数。那么这个函数符合我们的要求吗？很快我们就发现其不适合之处。我们的分组中，只要有一项不能满足，我们的控件就无法运行。然而求平均数的函数却无此特性。因此，我们需要另寻他路。为了便于筛选，我们觉得应该先把该函数应该具有的属性列出：

（1）函数值应该在 0 到 1 之间，0 表示完全不满足运行需求，而 1 则表示完全满足运行需求。

（2）如果其中 1 项或多项参数为 0，即该项不满足运行需求，函数值应该为 0。

（3）对单一参数来说函数值应该为单调增函数，即其他参数不变的情况下，一个参数变大，函数值就变大，反之亦然。

（4）如果所有项均完全满足，即所有参数为 1，则函数值也应该为 1。

乐观评价函数

经过分析和筛选，我们首先选择了如下函数：

F1=各项分组满足值之积开 N 次方

其中 N 为分组数量，F 为满足函数。如例 2 中，F1(Component1)=[（0.8*1*1*1*0.5）然后开 5 次方]=[0.4 开 5 次方],即为最后得出的 Component1 的满足值。

但是我们发现，此函数求得的满足值，总大于其中的部分分组的满足值。例如，有 4 个分组，满足值分别为 0.1,0.1,0.1,1 时，最后得出满足值约为 0.18。当 4 个非分组满足值分别为 1，1，1，0.1 时，最后得到的满足值约为 0.58。我们觉得此函数得到的结果过于乐观，因为 4 分组中的 3 项满足情况非常低的情况下，其综合满足情况，在实际情况中看来应当几乎难以满足要求；即使 3 项为 1，一项为 0.1 的情况下，实际情况中运行条件也相当不利，但此函数得到的结果却无法反映这一点。因此我们将此函数称为乐观评价函数，得到的满足值称为乐观满足值。

悲观评价函数

考虑到乐观评价函数的不足，我们尝试了另外一个函数：

F2= F（A）*F（B）*…

即 F2 等于各项分组满足值之积。例如，4 个分组满足值分别为

F(A)=0.1, F(B)=1, F(C)=1, F(D)=1 时，F2=0.1*1*1*1=0.1。我们发现，由于 F（X）<=1，而根据定义，F2=F（A）*F（B）*…，因此 F2 总是小于或等于其中任意一项。此函数得到的满足值总是偏低。因此，我们将此函数称为悲观评价函数，得到的满足值称为悲观满足值。

根据实际情况和需要，我们可以自由选择乐观评价函数或者悲观评价函数来进行计算。通常在目标设备资源较低的情况下，我们可以选择悲观评价函数。

这样我们就求得了单个控件的资源满足情况评价值。

6.2.6 当 X 为界面时满足值求解

对于界面资源满足情况的评价，其数据结构和计算方法与控件资源满足情况评价基本一致，可以用相同函数求解。值得注意的是，前边已经说过，如果某项资源（如操作系统等）被多个控件需求，将其列入界面通用资源描述列表，在控件资源描述表中就不用再列出。通常情况下，很多资源项都可以列入界面通用资源需求列表，因此，控件资源满足情况判定中的数据项通常不会太多，计算也不会过于复杂。

6.3 界面调整

当某个（或某些）控件的资源需求得不到满足时，如果可以找到对资源需求较低（低到目标设备的资源可以满足其要求）而且能够实现所需功能的控件，则可以考虑用该控件进行替换。如图 6-2 所示，几个控件都可以用来输入国家信息，图 6-2-B 是组合输入框，可以用鼠标来从下拉列表中选择需要输入的国家名称，也可以在组合输入框中直接输入国家名称，图 6-2-C 中的控件可以在世界地图上用鼠标点某个地区来选择国家。更进一步的，可以用图 6-2-A 中的最简单的输入框输入国家名称，这也可以实现输入国家名称的功能，但是它没有其他两个控件使用方便，牺牲了一定的易用性换取了资源需求的降低。

A 文本框输入　　B 组合框输入　　C 图形点选输入

图 6-2 输入控件

如果找不到合适的控件进行替换，则判定界面迁移所需资源得不到满足，界面无法迁移。

另外，要考虑目标设备的显示器的特点，判断是否需要对界面结构和布局进行修改和调整。例如，界面迁移之前运行的系统，其显示器是横置的（即宽度大于高度），而目标设备的显示器则是竖置的，界面结构和布局便需要调整来适应目标设备；或者，在前边的过程中我们对某些控件进行了替换，界面发生了变化，需要对其进行调整以适合显示，等等。

6.4 模拟试验及性能分析

6.4.1 模拟试验

为了验证算法的可行性，我们作了一个模拟平台来测试算法。首先，我们预定义了 3 个不同的模拟平台及一个简单的输入界面，在每个模拟平台上分别测试该界面的资源需求得到满足的情况，并输出最终的显示结果。其中的测试界面是一个输入信息的界面，主要是用来输入国家信息，可以使用鼠标/触摸屏在列表框中直接选则，或者在组合框的下拉列表中选择，也能够使用键盘在组合框的输入框中，或者在文本输入框中输入国家名称。3 个模拟平台的配置等信息如图 6-3 所示：

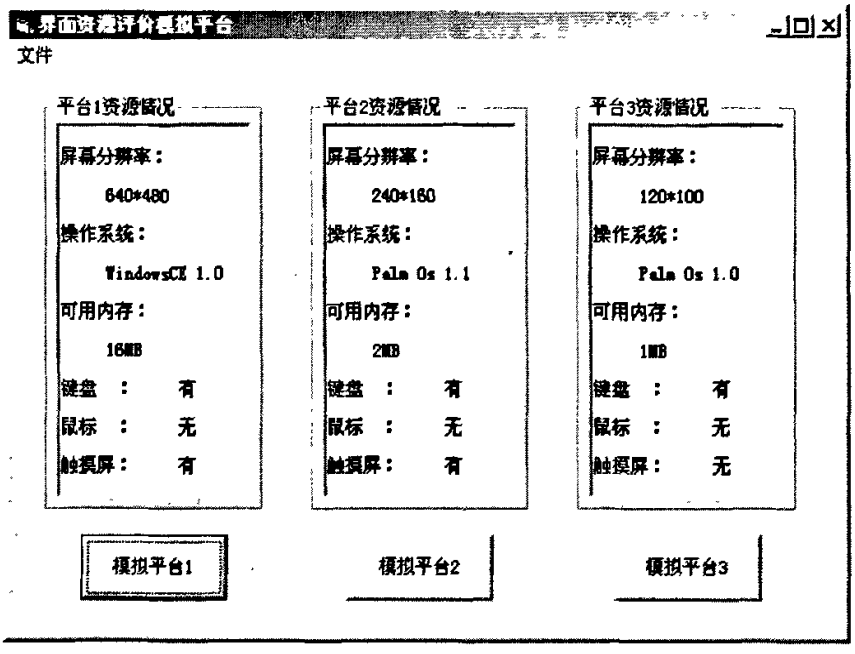


图 6-3 三个模拟平台的信息

在第一个平台上，由于该平台的资源对于控件 1、2 的需求都能很好的满足，因此选择利用控件 1、2 组合形成输入界面，达到最佳效果。控件 3 使用不够便捷，因此不予采用，见图 6-4（其中左边的文本框显示的是算法运算得到的评价结果等信息，右边是模拟平台的输出结果，下同）：

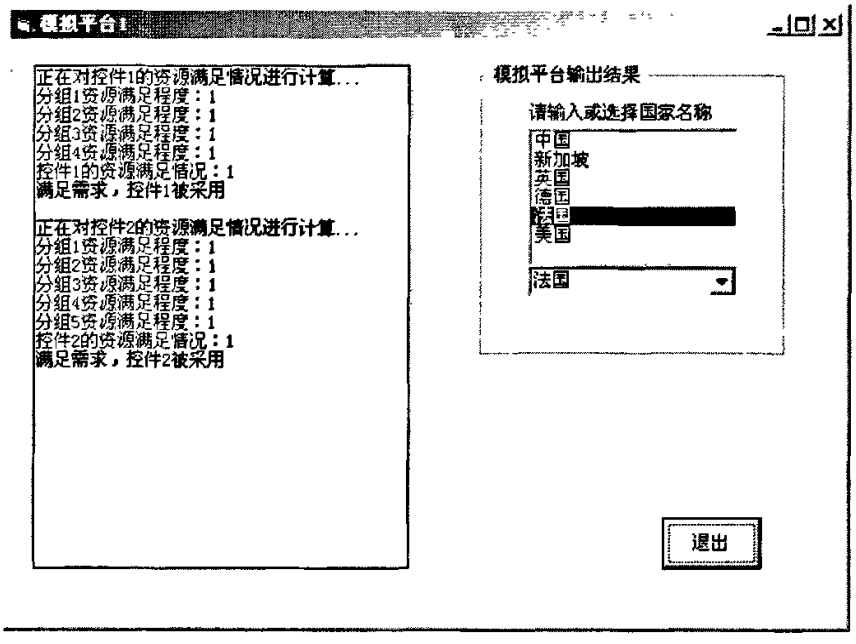


图 6-4 模拟平台 1 运行结果

在第二个平台上，控件 1 的需求不能得到满足，因此，根据替换原则，选择可以满足基本需求的控件 2 形成输入界面，见图 6-5。

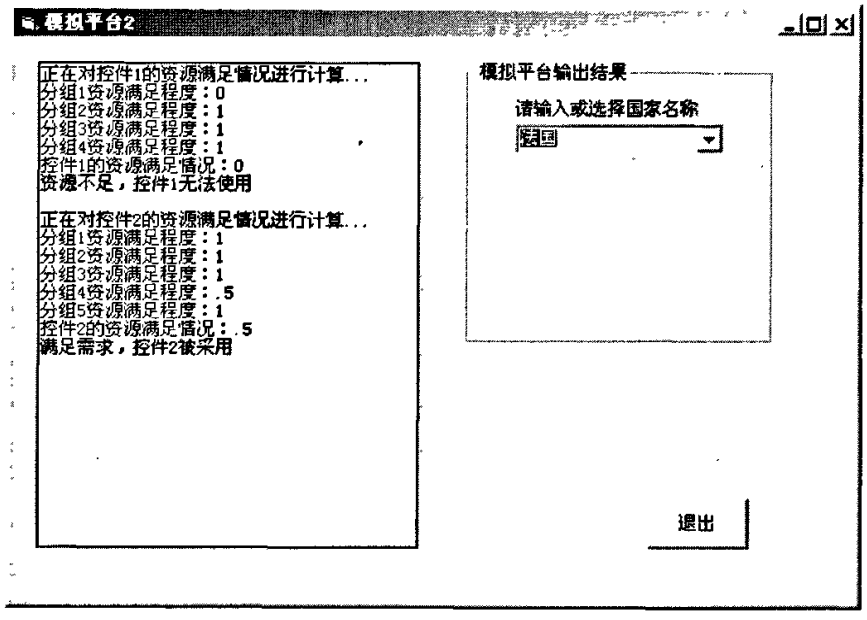


图 6-5 模拟平台 2 运行结果

在第三个平台上，控件 1、2 的需求均不能得到满足，因此只能选择利用资源需求低但效果较差的控件 3 形成输入界面，见图 6-6。

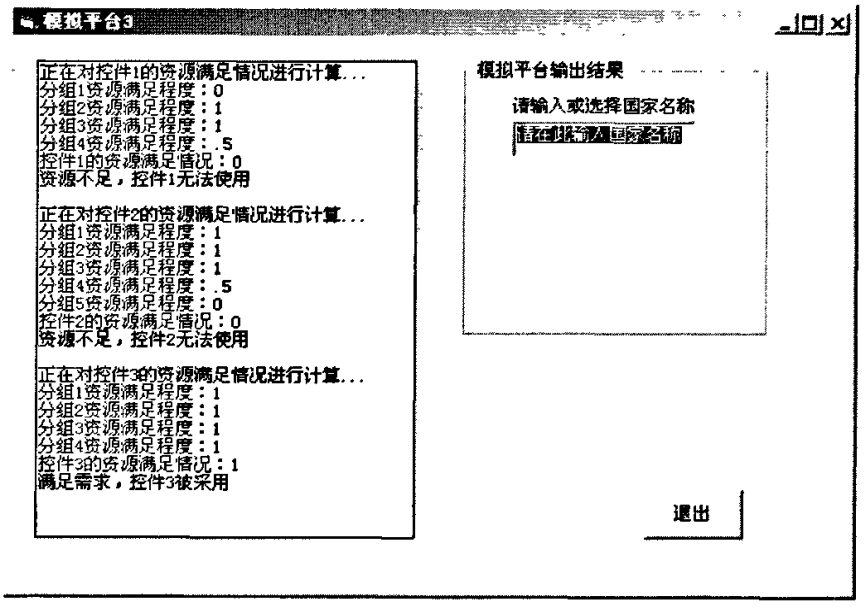


图 6-6 模拟平台 3 运行结果

6.4.2 部分关键源代码

```
//满足程度组合方法：Max，取最大的满足值作为函数值
Private Function Max_cal(X, Y, Z)
    Dim F As Single
    If X > Y Then F = X Else F = Y
    If Y < Z Then F = Z
    Max_cal = F
End Function

//满足程度组合方法：Add，取满足值之和与1之间较小的数作为函数值
Private Function Add_cal(X, Y, Z)
    Dim F As Single
    F = X + Y + Z
    If F > 1 Then F = 1
    Add_cal = F
End Function

//悲观评价函数
Private Function Fbg(A, B, C, D, E)
    Fbg = A * B * C * D * E
End Function

//控件A的资源满足情况求解,A1,A2,A3,A4,A5分别是5个资源组的资源满足值,默认值为1
Public Function cal_A()
//定义资源分组中各资源项满足值变量，暂定为每个资源组不超过3项，默认值为0
    Dim X, Y, Z As Single
//三维数组 A(5,5,5)为控件A的资源需求列表
//info1~info6为模拟平台相关资源信息
//求解各资源分组
    If A(1, 1, 1) > info1 Then
            X = 0
            If A(1, 2, 1) > info1 Then Y = 0 Else Y = A(1, 2, 2)
        Else
            X = A(1, 1, 2)
    End If
    A1 = Max_cal(X, Y, Z)

    If A(2, 1, 1) > info2 Then X = 0 Else X = A(2, 1, 2)
    If A(2, 2, 1) > info2 Then Y = 0 Else Y = A(2, 2, 2)
    A2 = Max_cal(X, Y, Z)

    If A(3, 1, 1) > info3 Then X = 0 Else X = 1
    A3 = Max_cal(X, Y, Z)
```

```
    If A(4, 1, 1) > info4 Then X = 0 Else X = A(4, 1, 2)
    If A(4, 2, 1) > info5 Then Y = 0 Else Y = A(4, 2, 2)
    If A(4, 3, 1) > info6 Then Z = 0 Else Z = A(4, 3, 2)
    A4 = Max_cal(X, Y, Z)
    //调用悲观评价函数求解控件资源被满足情况
    cal_A = Fbg(A1, A2, A3, A4, A5)
End Function
```

6.4.3 算法性能分析

评价方法是基于界面资源需求列表来计算的，需求列表在我们的模拟平台中用 3 维数组表示，但是每个控件的资源需求实际是一个线性列表（在通常的描述方法中，采用一维数组就可以描述该需求），通常由具体控件来决定，或者由 UI 设计者指定，可以计为有 m 项；每个界面由多个控件组成，可记为 n；界面自己的通用需求列表其计算复杂度与单个控件求解相当，可视为共需计算 n+1 个控件；因此本算法的时间复杂度为（n+1）*m，即其时间复杂度约为 n^2。

6.4.4 算法特点

本算法主要是针对目前移动设备资源已接近桌面设备，并日趋丰富的情况，为相近平台（特别是同样支持 Java 虚拟机等技术的平台）下的界面自适应、界面迁移提供一个简单便捷的方法，对界面自动生成中存在的一些问题提出一个求解思路。对于相差较大的平台间的界面迁移，本算法并不适合。表 6-7 是对目前几种常见的方法的比较（见下页）：

方案	界面分解/重构的方法	常见的界面调整方法	基于资源评价的界面调整
适合范围	差异较大的平台间的界面迁移	平台间差异较小的情况	平台间差异较小的情况
实现方式	将界面分解，按照目标平台的条件重构界面	使用合适的组件替换无法正常使用的组件	根据资源评价结果，自动替换组件
综合评价	优点是能够适应异构平台；缺点是算法和实现较为复杂，界面分解容易得到无意义界面	优点是实现较为简单；缺点是适应面窄，并且组件的选择和替换工作自动化程度不高	优点是能够自动判定和选择替换组件；缺点是仍然限制在相近平台间的界面迁移范围内

表 6-7：几种界面迁移方法的比较

6.5 本章小结

本章提出了一种资源描述机制，以及一种基于资源评价的界面调整方法，在此过程中对无法在目标平台使用的组件进行调换，来实现组件的跨平台自适应特性。该方法是对普通的界面组件替换/调整方法的改进，它基于资源需求满足情况评估算法，能够对目标平台是否满足特定组件的资源需求做出量化判断。

7 结束语

7.1 论文工作总结

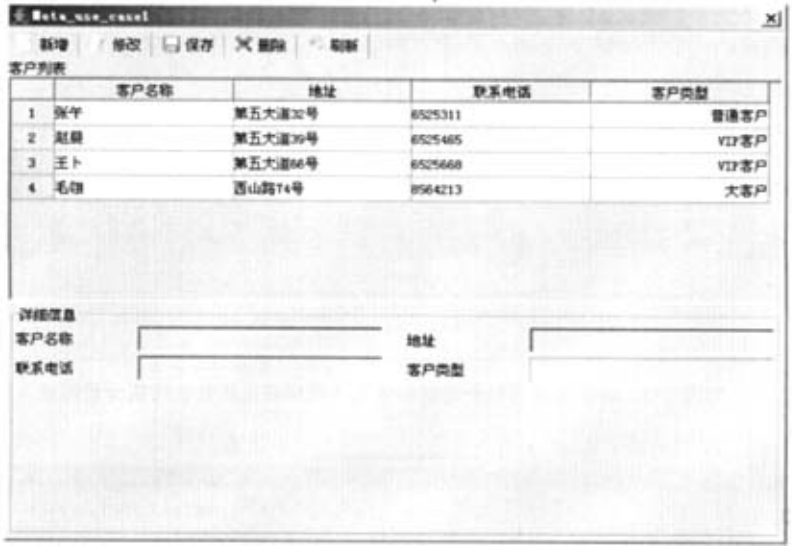
本文分析了现有的界面开发工具和思想，采用模型驱动的思想，提出了一种表示层模型的理论架构。在表示层模型中，通过统一数据网关，屏蔽了各种不同数据给程序开发带来的干扰，支持了表示组件的语法层面的自适应性；通过对不同平台现有的表示组件的抽象，使用完成的功能来描述这些表示组件，屏蔽了各种平台对表示功能实现的细节，支持抽象界面模型的平台无关的任务分解；通过表示组件之间和表示组件内部的两层依赖关系的描述，可以支持语义相关的界面分解和布局；通过对实现功能的交互子的评价和选择，可以支持表示组件在不同平台的实现，可以支持界面的迁移；通过将控制划分为表示层控制器和业务处理单元，很好地实现了应用语义和表示分离。满足了用户界面开发工具所希望具备的几个特征。另外，本文提出了一种界面资源的表示及评价方法，并可使用该方法调整界面以适应不同平台，这一方法部分实现了组件的平台间自适应特性。

7.2 未来研究方向

未来工作主要是对自适应的表示组件的内部实现机制进行更加深入的研究，同时，使表示组件之间的关系的描述和在不同平台的处理更具有普适性。另外，在某些情况下，目标设备的不同，会带来界面资源需求的变化，一组资源需求列表就不足以完美的描述界面对资源的需求情况。例如目标设备使用触摸屏作为输入设备，用手进行操作和用手写笔对触摸屏进行操作相比较，因为操作精度相差甚远，其对目标设备屏幕分辨率和界面布局的要求也各不相同。此时需要多组资源需求列表来描述此类需求。这是我们下一步的研究方向。

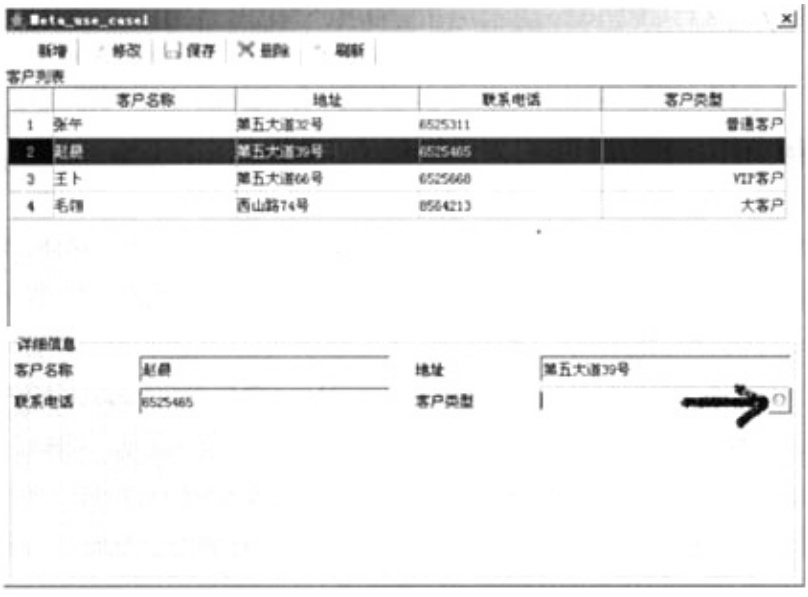
8 附录

8.1 附图

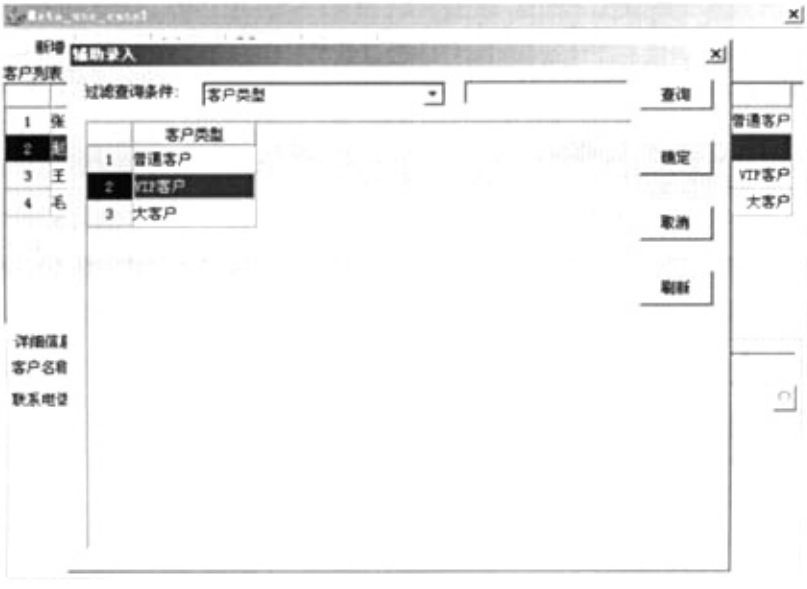


附图 8-1：当选择组件未执行选择操作时，编辑组件处于 disable 状态.

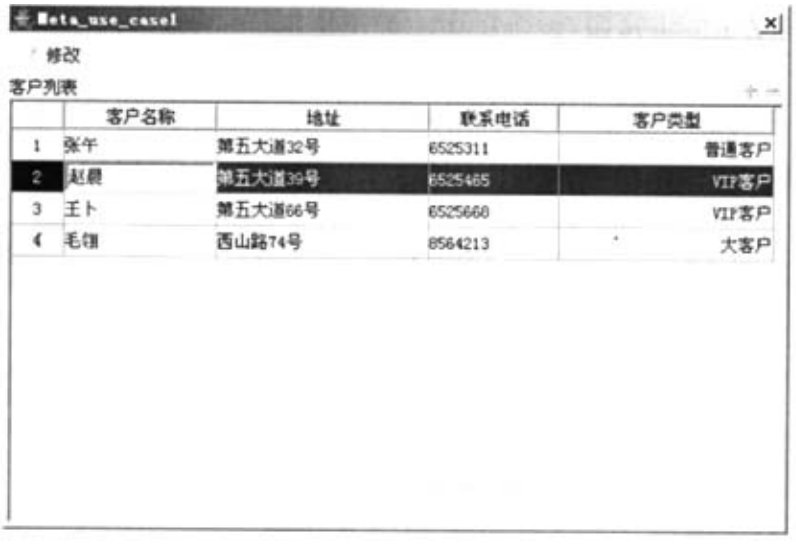
附图 8-2：当选择组件执行选择操作后，编辑组件处于 enable 状态



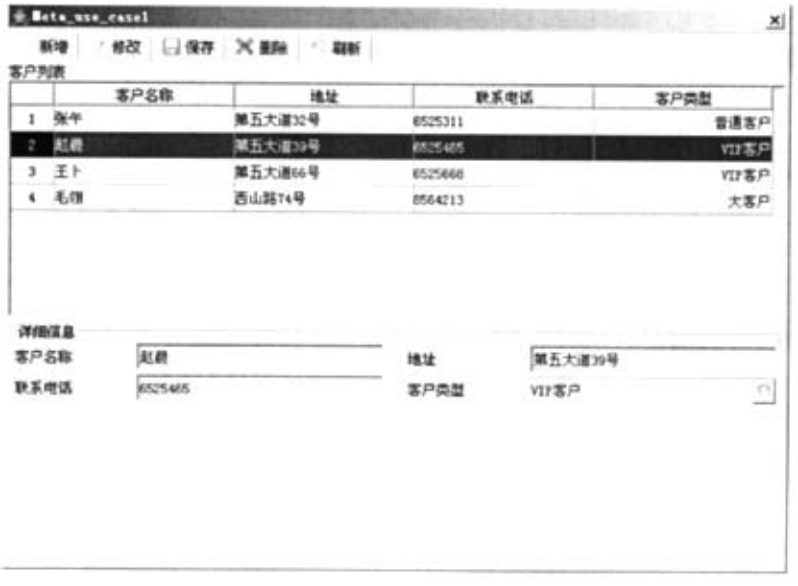
附图 8-3：部分信息可以使用辅助录入（图中箭头所指的按钮为辅助录入按钮）



附图 8-4：辅助录入的窗口



附图 8-5：功能区域 - 表格（图中“客户列表”框中的部分，可以在表格中修改表内数据）



附图 8-6：功能区域 - 选择（图中“客户列表”框中的部分，不能在该区域直接修改表内数据，只能进行选择），和功能区域 - 编辑（图中“详细信息”框中的部分）

8.2 附程序源代码

Meta_use_case1Bo.java

```
//界面初始化，并完成界面显示
package bo;
//加载 Java 相关模块
import java.io.File;
import java.io.IOException;
import java.rmi.RemoteException;
import java.sql.SQLException;
//加载本模型相关模块
import xingx.smartbiz.*
//具体代码
public class Meta_use_case1Bo extends BusinessObject {
 private static final String entity_name = "client.client";
 //信息增加操作* @throws BizException
 public Entity add(Entity en) throws BizException {
     try {
         super.getDefaultDataAdapter().insert(en);
     } catch (SQLException sql) {
         throw new BizException(
                 "BasebidHistorytechBo.add occurs SQLException---"
                         + sql.getMessage(), sql);
     } catch (MetaDataException mde) {
         throw new BizException(
                 "BasebidHistorytechBo.add occurs MetaDataException---"
                         + mde.getMessage(), mde);
     }
     return en;
 }
 //信息修改、删除等部分与上述代码结构一致，在此省略
}
```

Meta_use_case1Event.java

```
//具体事件的处理
package event;
public class Meta_use_case1Event extends UIEventHandle{
 String state=null;
 public void onOpen(final Page page) {
     try {
         this.find(page);
         //设置编辑模班为不可编辑
         final ObjectEditTemplet edit_template = (ObjectEditTemplet) page
         .getSubControlByID("body.detail");
```

```
            edit_template.setEnabled(false);
            //向表格的事件添加换行事件
            final TableTemplet tt = (TableTemplet) page
                    .getSubControlByID("body.EventTable");
            tt.getSelectionModel().addListSelectionListener(
                    new ListSelectionListener() {
                        //当选择列表中的一行时，在下面显示通知单信息
                        public void valueChanged(ListSelectionEvent e) {
                            // 得到当前选择的记录**********

                            edit_template.setEnabled(false);

                            if (tt.getSelectedData() == null
                                    || tt.getSelectedData().size() == 0) {
                                edit_template.clear();
                            } else {
                                Entity en = tt.getSelectedData().get(0);
                                EntitySet ds = new EntitySet();
                                ds.add(en);
                                edit_template.setEntities(ds);
                            }

                        }
                    });
        } catch (Exception e) {
            JOptionPane.showMessageDialog(null, e.getMessage(), "错误",
                    JOptionPane.ERROR_MESSAGE);
        }
    }
    public void processUIEvent(Page page, PageEvent ev) {
            try {
                //保存
                 if (ev.getId().startsWith("save")) {
                    modify(page);
                }
                   //新增
                 if (ev.getId().startsWith("add")) {
                    add(page);
                }
                   //修改
                 if (ev.getId().startsWith("modify")) {
                    edit(page);
                }
                 //删除
```

```
                if (ev.getId().startsWith("delete")) {
                    delete(page);
                }
                //刷新
                else if (ev.getId().startsWith("refresh")) {

                    this.find(page);
                     //设置编辑模班为可编辑
                    ObjectEditTemplet edit_template = (ObjectEditTemplet) page
                    .getSubControlByID("body.detail");
                    edit_template.setEnabled(false);
                    state = null;
                }
            } catch (Exception e) {
                e.printStackTrace();
                JOptionPane.showMessageDialog(null, e.getMessage(), "错误",
                        JOptionPane.INFORMATION_MESSAGE);
            }
        }
     //修改
    private void edit(Page page) throws RemoteException {
        TableTemplet tt = (TableTemplet) page
        .getSubControlByID("body.EventTable");
       EntitySet es = tt.getData();
       Entity sel = tt.getSelectedData().get(0);
       if (sel == null) {
        JOptionPane.showMessageDialog(null, "请选择相应记录", "提示",
                JOptionPane.INFORMATION_MESSAGE);
            return;
       }
       //设置编辑模班为可编辑
    ObjectEditTemplet edit_template = (ObjectEditTemplet) page
    .getSubControlByID("body.detail");
    edit_template.setEnabled(true);
       state = "edit";
    }
    //其它操作略
}
```

参考文献

[1] Silvia Berti, Fabio Paterno. Migratory MultiModal interfaces in MultiDevice environments. Proceedings of the 7th international c onference on Multimodal interfaces. New York :ACM Press. 2005. 9 2-99.

[2] Giulio Mori, Fabio Paterno, Carmen Santoro. Design and Developme nt of Multidevice User Interfaces through Multiple Logical Descr iptions. IEEE Transactions on Software Engineering. Piscataway: IEEE Press, 2004. 507-520.

[3] Jan Van den Bergh, KarinConinx. Towards Modeling Context-Sensiti ve Interactive Applications: the Context-Sensitive User Interfac e Profile (CUP). Proceedings of the 2005 ACM symposium on Softwa re visualization. New York: ACM Press. 2005. 87 - 94.

[4] Brad Myers, Scott E. Hudson, Randy Pausch. "Past, Present, and F uture of User Interface Software Tools". ACM Transactions on Co mputer-Human Interaction. March 2000. Vol. 7, No. 1, P3-28

[5] Jeffrey Nichols, Kai Richter, Krzysztof Gajos, The Many Faces of Consistency in Cross-Platform Design: A Whitepaper. Proceedings of CHI 2006 Workshop "The Many Faces of Consistency in Cross-Platform Design" , p9-18

[6] Giulio Mori, Fabio Paternò, Carmen Santoro, Tool Support for Designing Nomadic Applications. *IUI' 03*, January 12 - 15, 2003, Miami, Florida, USA.

[7] Murielle Florins, Jean Vanderdonckt, Graceful Degradation of User Interfaces as a Design Method for Multiplatform Systems. *IUI' 04*, January 13 - 16, 2004, Madeira, Funchal, Portugal.

[8] Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta, Applying Model-Based Techniques to the Development of UIs for Mobile Computers. *IUI' 01*, January 14-17, 2001, Santa Fe, New Mexico, USA.

[9] Tim Clerckx, Kris Luyten, and Karin Coninx , DynaMo-AID: a Design Process and a Runtime Architecture for Dynamic Model-Based User Interface Development.

[10] Alan Dix, Janet Finlay, Gregory D. Abowd. Human-Computer Interaction. Harlow, England. Prentice Hall. 2004. P288-315

[11] UML Use Case Diagrams , Engineering Notebook , C++ Report , Nov-Dec 1998

[12] Shari Lawrence Pfleeger. Software Engineering Theory and Practice(Second Edition). Pearson Education. 2001. P195-254

[13] Brad Myers, Scott E. Hudson, Randy Pausch. "Past, Present, and Future of User Interface Software Tools" . ACM Transactions on Computer-Human Interaction. March 2000. Vol. 7, No. 1, P3-28

[14] Frank Zhao, Qiong Liu. "A Web Based Multi-display Presentation System". Proceedings of the 12th annual ACM international conference on Multimedia. Oct. 2004

[15] Ali M.F., Pérez-Quiñones M.A., Abrams M.: Building Multi-Platform User Interfaces with UIML. In: Seffah, A., Javahery, H. (eds.): Multiple User Interfaces: Engineering and Application Framework. John Wiley, Chichester (2004) 95-118

[16] Eisenstein, J., Vanderdonckt, J., Puerta, A.: Model-Based User-Interface Development Techniques for Mobile Computing. In Lester J. (ed.): Proc. of 5th ACM Int. Conf. on Intelligent User Interfaces IUI' 2001 (Santa Fe, January 14-17, 2001). ACM Press, New York (2001) 69-76

[17] Göbel, S., Buchholz, S., Ziegert, T., Schill, A.: Device Independent Representation of Webbased Dialogs and Contents. In Proc. of IEEE Youth Forum in Computer Science and Engineering YUFORIC'01 (Valencia, November 2001). IEEE Computer Society Press, Los Alamitos (2001)

[18] Chu, H., Song, H., Wong, C., Kurakake, S., Katagiri, M.: Roam, a Seamless Application Framework. Journal of System and Software 69, 3 (2004) 209-226

[19] Sophie Lepreux1, 2, Jean Vanderdonckt1, Benjamin Michotte 1, Visual Design of User Interfaces by (De)composition.

[20] Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J.: A Unifying Reference Framework for Multi-Target User Interfaces. Interacting with Computer 15, 3 (2003) 289-308

[21] Hui Wang, Fengjun Zhang, Hong Jin, Danli Wang, Hongan Wang, Guozhong Dai. Desinging Human-Centered Ubiquitous Computing Journal of Integrated Design & Process Science 2005 Vol.9 No.2 33-46

[22] Xu Jungang, Wang Hongan, Dai Guozhong. The Implementation of an Enterprise Application Integration Platform Chinese Journal of Electronics, 2004Vol.13, No.1 122-126

[23] Wang Hongan, Wang Qiang, Wang Kun, Jin Hong, Dai Guozhong. Concurrency Control Protocol for Dynamic Priority Scheduling in Hard Real-Time Databases Chinese Journal of Electronics 2006 Vol.15, No.1 1-6

[24] Luan Shangmin, Dai guozhong, LI Wei. A Programmable approach to revising Science China Ser.F Information Sciences 2005 2005 Vol.48 No.6 681-692

[25] 李杰, 田丰, 戴国忠. 笔式用户界面交互信息模型研究 软件学报 2005年1月 第16卷第1期 51-57

[26] 田丰, 秦严严, 王晓春, 敖翔, 王宏安, 戴国忠. PIBG Toolkit: 一个笔式界面工具箱的分析与设计 计算机学报 2005年6月 第28卷第6期 1036-1042

[27] 王丹力, 华庆一, 戴国忠. 以用户为中心的场景设计方法研究 计算机学报 2005年6月 第28卷第6期 1043-1047

[28] 王常青, 邓昌智, 马翠霞, 华庆一, 戴国忠. 基于分布式认知理论的扩展

资源模型 软件学报 2005年10月 第16卷第10期 1717-1726
[29] 李杰，秦严严，田丰，戴国忠. CoPenML:基于XML的笔式用户界面构件体系结构 计算机研究与发展 2005年7月 第42卷第7期 1143-1153
[30] 马翠霞，王宏安，戴国忠，陈由迪. 基于手势和草图的概念设计协同交互的研究 计算机研究与发展 2005 第42卷第5期 856-861
[31] 马翠霞，戴国忠，滕东兴 陈由迪. 概念设计中基于笔式手势的交互计算研究 软件学报,2005 第16卷第2期 303-308
[32] 栾尚敏，戴国忠,李未. 知识库更新的一种可编程实现的方法 中国科学 E辑 信息科学 2005.7.5 Vol.35 No.8 785-797
[33] 付永刚，张凤军，戴国忠. 双手交互界面研究进展 计算机研究与发展 2005.4 第42卷第4期 604-613
[34] 王常青，王绪刚，马翠霞，邓昌智，戴国忠. 使用概率规则文法评估人机界面可用性 计算机辅助设计与图形学学报 2005.12 第17卷第12期 2709-2715
[35] David Hill，选择正确的表示层体系结构，http://www.microsoft.com/china/MSDN/library/architecture/choospreslayer.mspx?mfr=true
[36] 赵慧勤. 网络信息资源组织—元数据. 情报理论与实践， 2000.6
[37] 张敏，张晓林. 元数据的发展和相关格式. 四川图书馆学报， 2000.2
[38] 黄伟红，张福炎 基于XML/RDF的MARC元数据描述技术. 情报学报， 2000.8
[39] 程变爱. 试论资源描述框架（RDF）—— 一种极具生命力的元数据携带工具. 现代图书情报技术， 2000.6
[40] 林海青. 数字化图书馆的元数据体系. 中国图书馆学报， 2000.4
[41] 梅宏，陈锋，冯耀东，杨杰. ABC:基于体系结构面向构件的软件开发方法. 软件学报 Vol. 14, No. 4, 2003
[42] 张伟，梅宏. 一种面向特征的领域模型及其建模过程. 软件学报 Vol. 14, No. 8, 2003
[43] 向俊莲，杨杰，梅宏。基于软件体系结构的构件组装工具ABC-Tool. 计算机研究与发展， Vol.41, No.6, 2004
[44] 邹盛享，张伟，赵海燕，梅宏. 面向软件产品家族的变化性建模方法. 软件学报 Vol. 16, No. 1, 2005
[45] 史元春,普适计算：营造以人为本的信息服务新环境,中国计算机学会通讯
[46] 牟世忠，表示层自动化，2006
[47] 蓝魔时代 http://www.51freely.com/06/mblog/default.asp

致　谢

在此衷心的感谢史清华副教授和朱方金讲师。感谢他们孜孜不倦的教诲和悉心的指导。在研究生学习期间，史老师在学习上和科研上都给了我极大的指导和帮助，他以丰富的知识和实际应用经验，开阔了我的眼界，拓宽了我的知识面，使我深刻地体会到“学海无涯”的道理。史老师谦虚好学的精神促使我更加端正学习和研究的态度。朱老师敏捷的思维和不断创新的精神常常给我以启发，使我在研究生学习期间受益匪浅。我还要感谢计算机科学与技术学院的所有教师们，柴乔林、韩芳溪、肖宗水、曾广周、沈磊、张彩明、徐秋亮等。我是学院所有老师的学生，是老师们悉心的指导、无私的关怀使我的求学经历充实而愉快。除了感谢他们三年来对我的帮助之外，我还要感谢他们在本篇论文的前期准备以及写作过程中给予我的种种指导和帮助。

研究生期间，我得到了计算机应用专业同学们的热情帮助和支持，他们是：陶世忠、赵磊、张佃昌、唐小东，陈云瑞，李少玲，程秋云等。他们的机智，常常启发了我的思想；和他们和睦愉快的相处，使我获得了最佳的学习环境，在此一并表示感谢。

感谢所有关心和帮助我的人！

攻读硕士期间发表的学术论文目录

[1]《界面迁移中的控件资源需求满足程度评估方法》山东大学学报工学版 2007 年增刊 96-99　第一作者