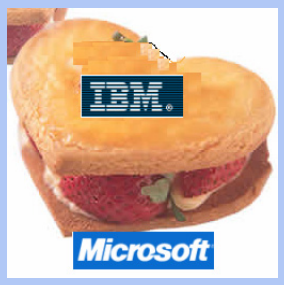
第 25 期（2003 年 5 月）

【新闻】

1 Rational能保持中立吗？…



能保持中立吗

【方法】

7 使用用例组织需求－识别用例

20 基于角色访问控制的UML表示

28 关系数据库访问层 一种模式语言

56 以用途为中心的Web应用工程

72 RUP对用户界面设计的支持

【人件】

83 中国“人件”非正式调查

X-非程序员 Programmer
软 件 以 用 为 本

投稿：editor@umlchina.com
反馈：think@umlchina.com

http://www.umlchina.com/

Rational 能保持中立吗?

[2003/5/1]

IBM 出资$21 亿收购 Rational 软件公司已经过去半年了，业界仍然是余震未消。IBM 坚持 Rational 仍然保持其企业软件设计、测试工具的平台无关性，就和过去一样。但分析家、顾客和合作伙伴－尤其是那些工作在 Microsoft 平台上的－对此表示怀疑。

Rational 公司的竞争对手最近生意渐好，来自过去 IBM Rational 顾客的咨询不断，有些甚至已经选择了他们的产品，尽管其中很多并不能提供 IBM Rational 那样的性能和扩展性。

现在，建模、SCM 和测试等工具的顾客有两种选择－购买 Rational 的产品，这意味着同时也投身于 IBM 的阵营（看看 Lotus 和 Tivoli 的情况就知道了）；或者选择还可以的小一些的公司的便宜一些的产品，它们声称会填补平台无关工具市场上的空白，尽管现在看来难度还不小。

Gartner 公司研究分析员 Jim Duggan 认为对过去的 Rational 及其顾客来说，这次收购也许是一件好事，利用 IBM 的资源，工具会得到很大的改进，当然，21 亿是另外一个很好的原因。不利之处在于，作为以平台中立而成功的 Rational 来说，现在显然成了一家公司的分部，而这家公司的目标是简化 Java 应用的开发。

“会有一些忧虑”，关于 IBM 接管 Rational 这件事，Duggan 说，“但是到目前为止，传递出来的信息是‘相信我们’。”

RATIONAL 和 MICROSOFT

和工作在 JAVA 环境上相比，工作在 Microsoft .NET 环境上的开发团队要迷惘得多。“过去，Rational 找到了在 IBM 和 Microsoft 中间的平衡点，并且很好地维持了五六年，” Duggan 说，“但显然，Microsoft 对现在的情况不会满意”。

Duggan 认为，失去 Rational 这样一个销售渠道，对 Microsoft 来说可能是比失去这些工具本身更重大的损失。“在三层套件（three-piece suit）上，他们不可能有同样档次的销售人员〔这是来自其它厂商的说法〕。”对于各个单独的工具，Duggan 认为其它销售商可以提供价格更优惠甚至性能也更好的产品，但是 Rational 的优异在于它的整个工具套件。

“Rational 将试图长期保持和 Microsoft 的关系，但是，作为 IBM 的分部，这简直是不可能的”， Duggan 说。

对 Prashant Sridharan 来说，至少现在，IBM 的 Rational 和 Microsoft 之间一切良好。Sridharan 是负责 Visual Studio 的产品经理。“我上周才和 Rational 公司的同行见面，我们关系不错，大家都是好拍档”。

Sridharan 认为 Rational 是 Microsoft 的重要合作伙伴，并且仍将获得 Microsoft 产品发布前的早期版本。作为 Microsoft 企业部的官员，Sridharan 认为，“Rational 是进入企业应用领域的渠道，是 Microsoft 的重要伙伴，当然，不是唯一的”，他举了 Mercury Interactive 公司的例子，和 IBM Rational 一样，该公司的测试工具同样面向企业用户。

Eric Schurr，前 Rational 市场部负责人，现在负责 Rational 软件产品市场的 IBM 副总裁，认为 IBM 对非 IBM 平台的支持“允许我们为更大范围的顾客服务，我们将继续支持其它平台。”

对于那些担心 IBM 最终会将自己拉上 IBM 平台的顾客，Schurr 说，“不用担心，IBM 不会强迫顾客干任何事情。我们没有摆木马计，也不会强迫 ClearCase 的顾客来使用 WebSphere。但当顾客们看到集成所带来的利益，也许他们会愿意尝试一下。”

Serena Software 公司的 Mark Woodward 认为 Rational 顾客将会被转移到 IBM 平台上的，他提醒，IBM 已经把 ClearCase Lite（一个需求管理工具）（译者注：CC 应该是配置管理工具，Rational 的需求管理工具应该是指 RequisitePro，但原文如此）集成到 WebSphere 应用开发工具包中。Woodward 说，他已经看见了 Rational 工具被纳入 IBM IDE 的未来。

Woodward 补充，IBM 同样说过 Tivoli 和 Lotus 要保持中立的话，“但蜜月总是短暂的，他们可以称它为分部，但和公司其它部门的紧密集成却不可避免。”

Gartner 公司 Duggan 认为，“即便 Rational 保持中立，它也不会成为 Microsoft 的渠道。而且，并没有一个好的替补。Borland 是 J2EE 阵营的，Mercury 还行，但规模却只有 Rational 的 1/3。”

复杂的关系

Microsoft 公司 Sridharan 补充，他也听到了顾客们对 Rational 和 IBM 紧密结合的担心，他说，“占在付费者的角度出发，我理解这些担忧。但是要知道，商业的关系足够复杂，在某些领域 IBM 和 Microsoft 仍然是伟大的拍档，尽管我们在其它领域存在竞争。”

但是，Embarcadero Technologies 公司的产品设计和建模负责人 Greg Keller 认为，这场收购和 Borland 公司对 TogetherSoft 的收购一样，都是 UML 建模工具和 Java 的结合。

“很显然，两次收购的焦点都致力于简化基于 J2EE 的应用开发，”Keller 认为，“IBM 有操作系统、数据库、应用框架，还有加快市场化的 JAVA 应用的模型驱动工具。这正是 Microsoft 对.NET 世界的设想。这是一场经典的‘可口可乐 Vs 百事可乐’之战。”

MKS 公司的总执行官 Michael Harris 认为，尽管业界人士认为 Rational 是为数不多的“企业级”工具集的开发商，但“除非你不知道你的运行平台，你不是企业用户。这里我们说企业，意思是指工具对企业的影响有多大？并不是说一个厂商必须要有各种工具，企业只需要使用已经选择的工具进行工作。”MKS 销售配置管理和版本管理工具，是 Rational 的竞争对手。

Harris 认为顾客不会花钱来替换他们已有的工具；他们需要工具商理解企业所面临的挑战，并帮助他们解决问题。

IBM 的 Schurr 承认 Rational 曾经一度支持 UNIX 超过 Windows，但是情况早在三四年前就发生了变化。Schurr 认为竞争对手“总是试图为了自己的利益来歪曲〔这次收购〕，但市场将会见证我们的所作所为。”

Schurr 认为 IBM Rational 覆盖了最佳实践、测试、配置管理以及通用建模工具等各领域的工具集并没有受到底层技术的约束。“认为 Rational 的一切都会变成 Java 的，这种论断忽略了我们所做的工作。”

但 Duggan 认为不可能指望 IBM 维持和 Microsoft 的长期合作。“你必须期待比 Rational 更好并且工作在.NET 上的产品。”

（自 SDTimes，风自由 摘译，不得转载用于商业用途）

用 Select Scope Manager 进行自动极限编程（XP）开发

[2003/4/24]

日前，Select Business Solutions 公司发布 Select Scope Manager，支持 XP 开发。

Select Scope Manager 不仅支持诸如快速决策和扩展的开发反馈的这些 XP 原则，还通过故事（Story）和任务(Task)的管理实现 XP 项目跟踪的自动化实现。

Select Scope Manager 帮助开发者自动化他们的 XP 开发过程，加速业务关键应用的交付。使用 XP 方法，开发的时间和成本都会节省，这些已经得到证明。现在，通过使用 Select Scope Manager，组织可以最低限度和最小范围地，用最少的文档管理好软件的开发过程，并且通过工具来维护软件的知识产权。

来自 Select Business Solutions 公司市场和开发部门的 Hedley Apperly 说，“我们在开发过程中使用了 Select 的其它额外经验来帮助缩短开发周期、节约成本，我们在开发 Select Scope Manager 的过程中成功地使用了 CBD、UML 和 SSADM 等技术”， Apperly 还补充说，“Select Scope Manager 是专门支持 XP 过程和项目的第一个也是唯一一个工具集”。

Select Business Solutions 公司的总裁 Don Hanson 说，“Select Scope Manager 的第一批用户给了我们相当不错的评价，我们非常开心、非常兴奋。初始 alpha 和 beta 版本的测试反馈表明，Select Scope Manager 用户的投资得到了迅速的回报，总的开发时间和开发成本大幅度缩减，并且程序质量得到了提高”。 Select Scope Manager:

- 支持故事（stories）、idea、文档和分析的保存；
- 帮助开发团队进行项目估算以及细化任务；
- 可以在“wizard”的帮助下快速创建 XP 故事；
- 帮助开发者理解在开发递增（when planning your increments）时的“What-If”场景；
- 为小组成员分配任务；
- 自动生成 Microsoft Project 计划；
- 跟踪任务，显示开发人员当前工作并标识出没有分配的任务；
- 清晰地监控开发速度、超支情况，以及还有多少未完成的工作；

（风自由 摘译，不得转载用于商业用途）

Embarcadero 宣布支持 Microsoft Visual Studio .NET 2003 和.NET Framework

[2003/4/24]

Embarcadero 的 UML 设计和建模产品 Describe Version 6.0 引入了对 Microsoft Visual C# .NET 的支持，以及对 Microsoft Visual Studio .NET 的全面指南。

Embarcadero(R) Technologies, Inc. (Nasdaq: EMBT)，应用程序和数据库生命期管理解决方案的领先者，今天宣布公司基于 UML 的集成建模开发环境（Integrated Modeling Development Environment，IMDE）产品 Describe(R) 将在即将发布的 6.0 Enterprise 引入对 Microsoft Visual C# .NET 语言的支持。除了对 Visual C# .NET 的增强之外，公司计划在 Microsoft Visual Studio .NET 2003 集成开发环境（IDE）内部提供全面的 IMDE 支持，使 Visual Studio 应用建模者和应用开发者在 Visual Studio .NET 2003 IDE 内能无缝使用 Describe 的直观而强大的建模特性。

Describe 的行动展示了 Embarcadero 为广阔领域的客户提供 end-to-end，平台独立，模型驱动分析设计环境的承诺。

“在开发项目时，可能有许多涉众，从 Visual C# .NET 开发人员到 SQL Server DBA 到业务单元经理，清楚传递应用架构的信息是至关重要的”，Microsoft 平台和推广部首席产品经理 Prashant Sridharan 说。

Beta 用户参与

欲获取关于 Embarcadero Describe for Visual Studio .NET 2003 beta 用户程序的进一步信息，请致信 Describebeta@embarcadero.com

（think 摘译，不得转载用于商业用途）

Eclipse, UML 齐头并进

[2003/3/31]

由 IBM 率先提出的 Eclipse 开放源码工具又有新的进展，与此同时，OMG（Object Management Group）的 UML（Unified Modeling Language）2.0 也有新的突破。

本周一，Eclipse 协会宣布正式推出 CDT（C/C++ Development Tools）1.0 版，并将其称为是向集成的跨平台的开源 IDE 迈出的重要一步。几乎与此同时，在上周奥兰多举行的 OMG 大会上，UML2.0 规约的部分内容朝着标准的方向又靠近了一步。

Eclipse CDT 1.0 框架将被用于集成多方的 C/C＋＋工具。Eclipse 称，CDT 的发布意味着 Eclipse 对 C 和 C＋＋的支持像对 Java 一样强大。

从 http://www.eclipse.org/tools/downloads.html 可以下载到 Eclipse CDT 1.0 版。这项免费的技术提供一个集成各种工具的平台，主要特色是其 plug-in-based 框架，旨在更易于构建、集成和使用软件工具，工具开发商可以共享这些核心集成技术。

在另一条战线上，OMG 投票通过了 UML2.0 基础架构、对象约束语言和图交换协议，这表明这些规范已最终定稿，但关于图的表示的 superstructure 还未确定。

OMG 标准的负责人 Fred Waskiewicz 称,UML 是分析和设计应用软件的建模语言，同时也是 OMG 模型驱动的体系结构方法中重要的组成部分。

Waskiewicz 表示，OMG 的批准意味着提案已经被 issuing task force 和 OMG 架构委员会（OMG architecture board）评审，并将由 OMG 平台技术委员会（OMG platform technology committee）及理事会投票通过。

Waskiewicz 称 UML2.0 将于六月被最后通过。UML2.0 关注的重点在于更好地集成语义学以及对扩展性的更佳支持。利用 UML2.0，核心的开发成果（译者注：MDA 中的平台无关模型）和设计代码可通过 profiles 被扩展到不同的平台，如 Web 服务、Java 以及 CORBA (Common Object Request Broker Architecture)。UML2.0 所做的就是提供实现这一切所需要的语义和机制。

（Crestress 摘译，不得转载用于商业用途）

《人件》— 老板，别把开发人员当成牲口

新闻

《人件》提供预订>>

相关文章

《人件》实践之：SAS公司
关注程序员自己的文化——专访Tom DeMarco
别把开发人员当成牲口
Brooks在《人月神话》中的评论
Alan Cooper的评论
办公室空间，下一场革命
《人件》在计算机行业的实践（待续）
人的问题：关于《人件》
Edward Yourdon的评论
开发人员是人吗？

各版本封面

英文，日文，德文

首页 购买 留言板

使用用例组织需求－识别用例

think 著

查看评论

识别 actor，要注意两个字：边界。

识别用例，要注意两个字：价值。【1】

图 1 齐秦歌曲 “边界”、“价值”

“价值”这两个字也是用例技术区别于之前的技术的关键所在。

你厌烦了 56K Modem 上网的速度，决定去申请装 ADSL。你到营业厅填好单，和证件一起交给营业员，营业员看看你的证件，接过单子开始操作电脑。如果你能站到营业员身后，你会看见营业员打开一个“受理”菜单，选择“ADSL”，弹出一个带有好多个 Tab 的框框，营业员把你填的那些东西哔哩叭啦输进去。然后告诉你回家等，施工人员两个小时内就到。你高兴极了（可以在线看电影了），转身出门。营业员吁一口气，想“又弄完一个”，看看电脑上的时间，还有 1 个小时才下班.....。【2】

如果在识别用例上有什么问题，不妨想一想上面所描述的这个场景。想一想这件事情为“你”提供了什么有意义的价值，为营业员提供了什么有意义的价值。

用例的定义

用例的定义有很多种版本，我们选择 RUP 的版本：A use-case instance is a sequence of actions a system performs that yields an observable result of value to a particular actor. A use case defines a set of use-case instances.（“用例实例是系统执行的动作序列，这些动作将生成特定 actor 可见的价值结果。一个用例定义一组用例实例”）。【3】

理解这个定义，可以很好地帮助我们解决在应用用例技术的过程中可能会碰到的各种问题。

（1）可见的价值结果。用例必须提供可见（可度量）的价值结果。

图 2 单个步骤不能为 actor 带来可度量价值

如上图：在一个商品销售系统中，“设定查询条件”、“选择商品”等等并不能为“会员”提供可度量的价值。其实它们只是“检索商品”里的几个步骤，“检索商品”才可以为“会员”提供可度量的价值。我们可以这样想：“会员”如果仅仅是“设定查询条件”，有助于帮助他完成工作吗？有助于帮助他领到工资吗？答案是“否”。正确的用例如下：

图 3 “检索商品”对会员有可度量价值

以上举例是要说明用例必须为主参与者提供可度量的价值，并非想说明“设定查询条件”、“选择商品”之类的就不是用例。读者可以自己思考什么情况下“设定查询条件”可以作为用例。【4】

上面就是一种把步骤（actor 的动作）当作用例的错误情况，还有另一种把步骤当用例的情况：

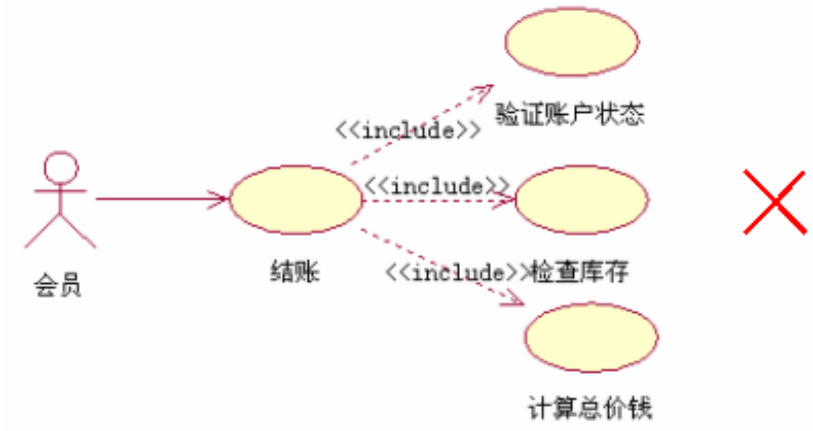


图 3 错误的 Include 关系

上图的本意是：当会员要求结账时，系统要先验证账户的状态，检查所订商品是否有库存，还要计算出整个订单的总价钱。实际上“验证账户的状态”等三个的主语是“系统”，并非用例，而是“结账”用例中的步骤。这种错误在用例初学者中非常容易出现：

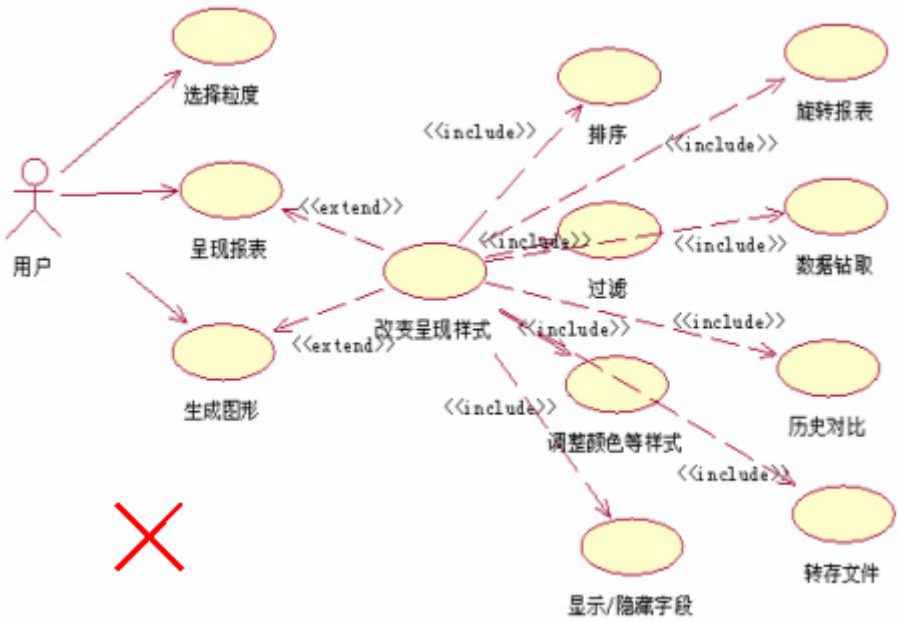


图 4 糟糕的用例图

上图是另一个项目中的用例图，犯了上面所有的错误。实际上用例只有一个：用户－－做报表。

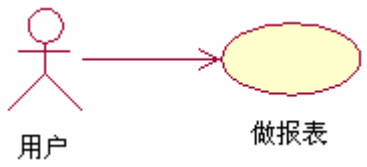


图 5 其实用例只有一个

（2）特定 actor。这个特定 actor 就是用例的主 actor。

图 6 把用例当作数据传送中介

网络告警处理系统。采集系统（一个外系统）发送告警信息，系统分析告警信息，存储，分类显示。操作员如果在工作位置上，他会处理这些告警信息。发送的告警是为了让操作员处理的，这就是箭头指向操作员的意思。问题在于：“发送告警”这一用例并不需要等待操作员处理了才有价值，系统存储、显示以后，采集系统的任务就完成了。操作员也不是这个用例的辅助 actor。如果操作员怠工不处理告警，责任在操作员，不关采集系统的事。可以把用例看作一份契约，“发送告警”这份契约，不涉及操作员。

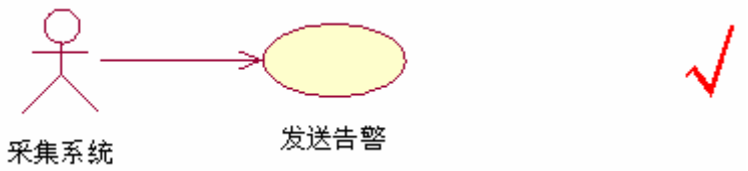


图 7 不关操作员的事

这是一种情况，还有一种误把辅助 actor 当作主 actor 的情况：

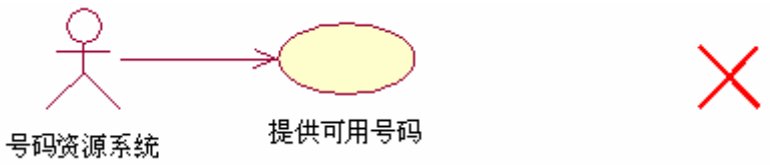


图 8 号码资源系统成了主 actor

某通信公司的营业受理系统。在识别 actor 阶段，识别出“号码资源系统”会和当前讨论系统（System under Discuss，SuD）“营业受理系统”交互。号码资源系统确实可以向“营业受理系统”提供可用号码的信息。所以开发人员【5】提出了一个“提供可用号码”的用例。问题在于：“号码资源系统”不会无缘无故地“提供可用号码”，只是营业员在“受理新装电话”，“受理改号”…时，可能会请求“号码资源系统”提供可用号码。实际上，“号码资源系统”是这些用例里的辅助 actor，“号码资源系统提供可用号码”只是某个用例里面的一个步骤：

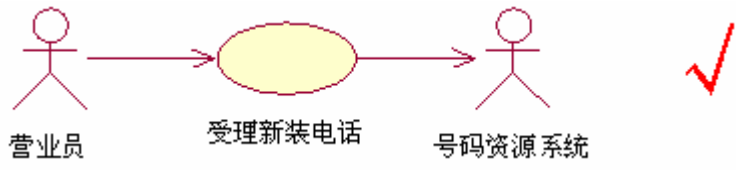


图 9 正确的用例图

（3）系统执行

这几个字说明，用例表示的就是系统的需求，把各种需求组织到用例中形成的用例文档就是需求文档，不必再寻求传统形式的需求。因为用例就是需求，所以用例的名字要准确地传达系统应该做什么。

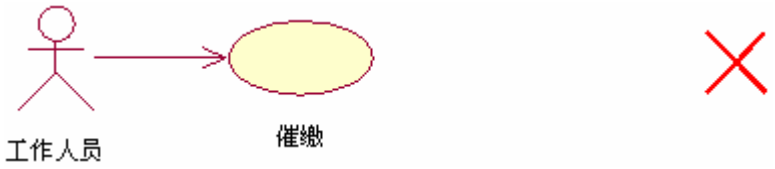


图 10 不恰当的用例名

某种汽车服务的计费系统。当有些用户未能按时缴费或银行托收不成功时，工作人员要负责催缴。所以开发人员提出“催缴”用例。但实际上工作人员需要使用系统做的只是“查看应催缴用户清单”，实际的催缴（打电话，投递催缴信等）是通过另外的渠道完成的。正确用例为：

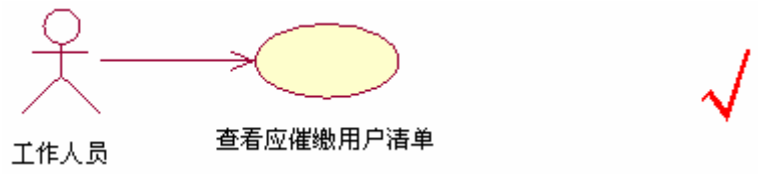


图 11 恰当的用例名

注意，这里还有一个较常看到的误解：认为使用系统完成的是系统用例，那么不使用系统但现实中要完成的就是业务用例了？如下：

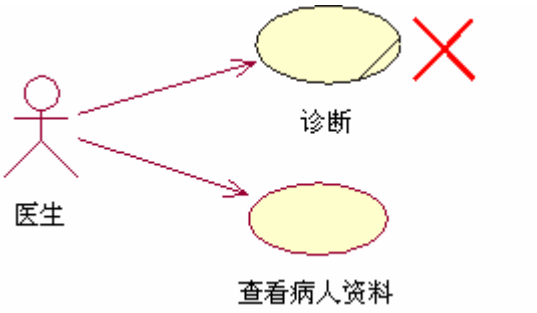


图 12 “诊断”变成了业务用例

这是一个医院管理系统。医生不是通过系统来“诊断”，但医生诊断是现实存在的事情，所以分析员就画了个业务用例“诊断”。业务用例当然不是这个意思。有关业务用例，我们将在后文讲述。

要准确地给用例一个名字，并不容易做到【6】。往往从给用例起的名字上就能看出作者是否真正理解了用例技术。恰当名字的用例集合能够鲜活地呈现出系统的概貌和价值。

用例的名字应该是动宾结构，如：“打电话”，动词前面可以加状语，宾语前面可以加定语，“在优惠时段打电话”、“打长途电话”。

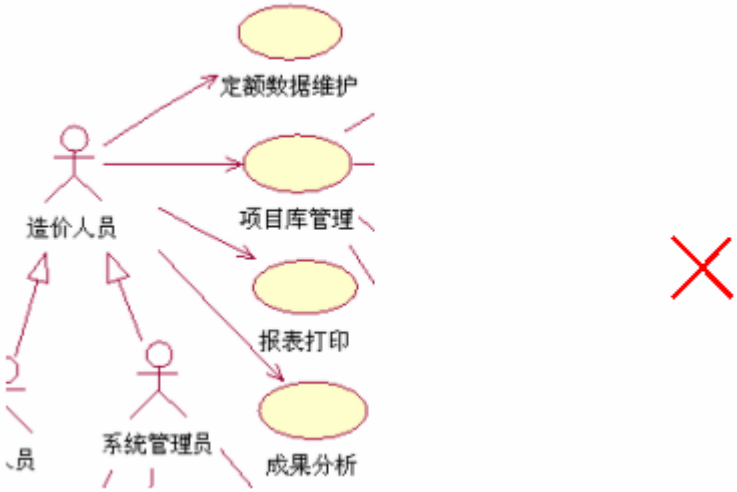


图 13 不是动宾结构

上面的错误可能都知道，改装成动宾结构还是比较容易的。这里还要提防的是使用一些弱动词、弱名词形成的“伪动宾结构”【7】

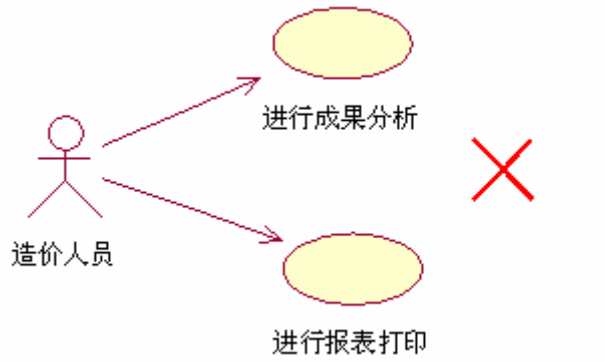


图 14 弱动词构成的伪动宾结构

用例的名字要能够传达业务信息，不要使用没有业务意义的名字，下面这张 Rose 模型的浏览器截图是一个错误的极端：

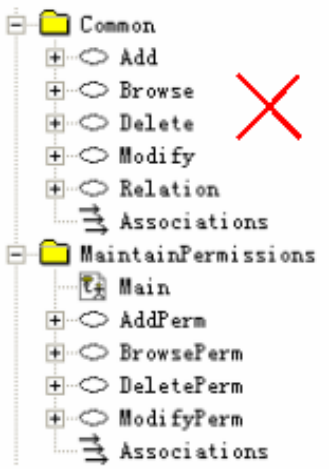


图 15 放之四海皆准的名字

作者给用例起名叫 Add，Browse，Delete，Modify 等。然后在繁衍（扩展）出 AddPerm….这也是正确的，不是吗？因为任何业务落实到数据这一层，都是增加、修改、删除、读取（CRUD）。但我们是不是应该从这个角度去识别用例呢？如果带着这副有色眼镜去看，世界上的一切都是 CRUD。【8】

（4）一组用例实例，动作序列。

经常有人问，用例的粒度应该多大才合适？从定义中可知，用例要包括一组用例实例（或称路径、事件流），而一条路径里又有一个动作（步骤）序列，这是用例粒度的底线。

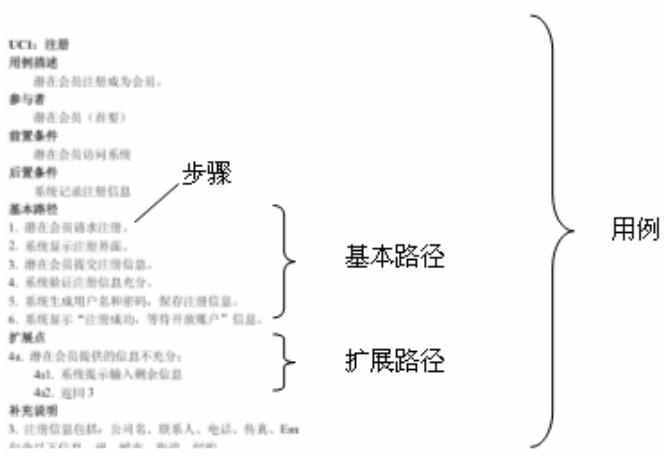


图 16 用例有若干路径，路径有若干步骤

在这之上，只要边界没有改变，多大粒度的用例在技术上都是对的，甚至大到整个系统只有一个用例“使用系统”，只不过这样的用例没有什么用。所以，严格来说，用例不存在粒度问题。

经常出现的粒度问题是 CRUD 用例，也就是所谓的“四轮马车”：

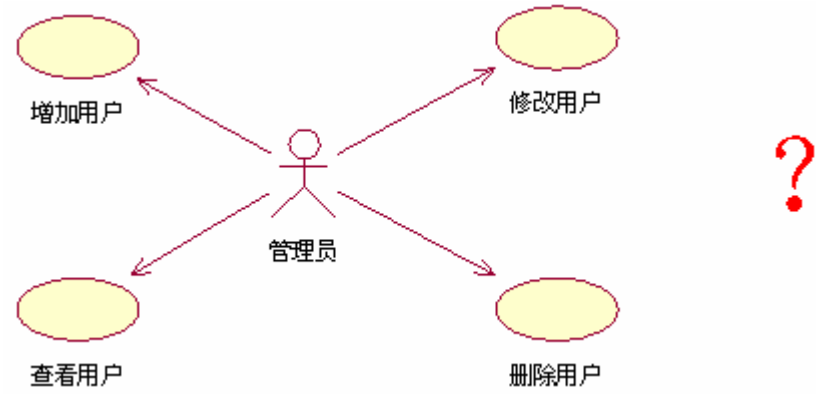


图 17 四轮马车，到底是一个用例好还是四个用例好？

碰到这种情况，首先要先检查“增加××”…这些用例名是否合适，它能形成单独的价值吗？为什么要“增加××”？还是背后另有业务意义？

图 18 增加宿舍？

“增加宿舍”指什么？新盖一栋宿舍楼？原来－－开发人员说的是指定某个物业作为宿舍使用时，宿舍的数目增加了，“增加宿舍”背后隐藏着“调配物业”。那么，叫“增加宿舍”就不可以吗？未必，如果物业室的工作人员平时在业务上都使用“增加宿舍”，叫“增加宿舍”也没什么问题。

如果确实是管理数据的简单用例。可以把这几个用例合并成一个用例“管理××”即可：

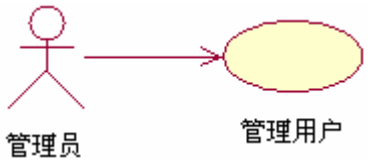


图 19 简单情况一个用例即可

不管是 C、R、U、D，都是为了完成“管理”的目标。这样做有时还是有问题。例如一个生产系统里，就有很多种基本数据要维护；即使是一个普通的系统，也有许多参数要配置。就算每种数据一个“管理××”，用例的个数也很容易就上去了。其实，甚至很多种基本数据的管理都可以用一个用例表示：

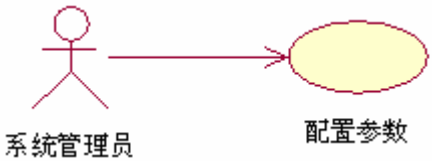


图 20 很多种数据也可以用一个用例

这并不是绝对的，如果在书写用例文档时发现“管理××”用例里有某种数据或某条路径有特殊性，并不妨碍把它独立出来单独作为一个用例处理：

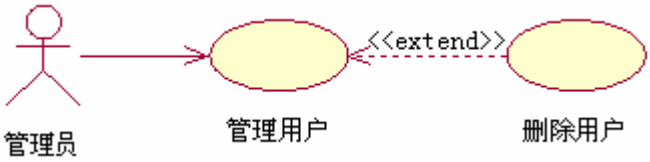


图 21 可以把复杂的分支独立出来单独处理

识别用例的过程

上面的段落实际上已经把一些识别用例时常见的错误组织到对定义的理解中。真正理解了用例是什么，识别用例就不是一件难事了。如何寻找用例呢？

（1）很多教科书上都列了一些大同小异的思路指引：

对于各个 actor，哪些任务会涉及到系统？

是否需要将系统中发生的某些特定事件通知给 actor？

此 actor 是否需要将突发变更或外部变更通知给系统？

actor 是否需要支持和维护系统？

actor 是否需要在系统内修改或创建什么信息？

....

但是在不同的项目中，系统涉及的领域和业务是多种多样的，所以上面的思路只可以作为一种启发，无法涵盖各种各样的用例。根本的思路还是要针对每个 actor 询问以下问题：actor 需要使用系统达到什么目标？

（2）如果前面有业务建模阶段，可以从业务用例的细节中来推演系统用例。通常需要使用系统的业务 worker 会映射成系统 actor，业务 worker 的活动映射成系统用例。以本文开头所说的装 ADSL 为例：

用例名：装 ADSL

主 actor：用户

范围：××通信公司

…..

用户把申请单交给营业员

*营业员*检查用户有申请资格

*营业员*受理

*施工人员*施工

*计费部门*给用户建账户

…..

由上可以考虑“营业员”、“施工人员”、“计费部门”为系统 actor，“受理”、“建账户”是可以通过当前讨论系统来完成的，可以作为系统用例。

但要注意，这种映射并非一一对应或者“是/否”这么简单。例如：施工人员“施工”不是全部通过系统完成的，和系统有关的只是“返回施工结果”。特别是在 web 时代，很多原来由 worker 完成的工作将自动进行，actor 也相应发生变化。

需要注意的几点

1. 不要急于添加细节。

这个阶段的任务是从面上给出反映系统概貌的用例。用例的粒度也应该是宁粗勿细的，而且不应该急于描述用例的细节。在这个阶段容易走形的原因之一就是开发人员按耐不住直接在用例图上添加各种细节，以从中获得一种虚假的安全感。

图 22 虚假的安全感

2. 不要使用关系

这一点和上面一点实际上是一回事。没有细节，哪里出来的关系。布满关系的用例图极有可能是低劣的。

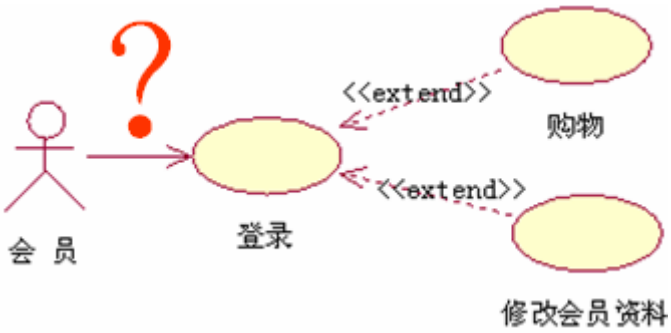
注释

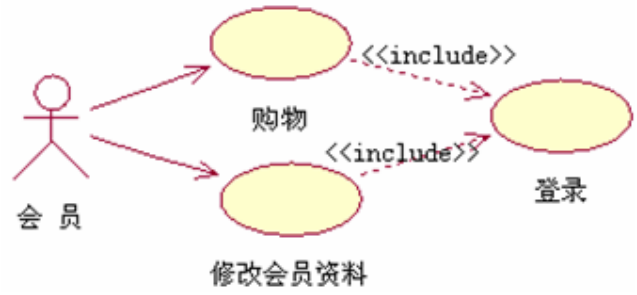
【1】还有一个候选词：契约。

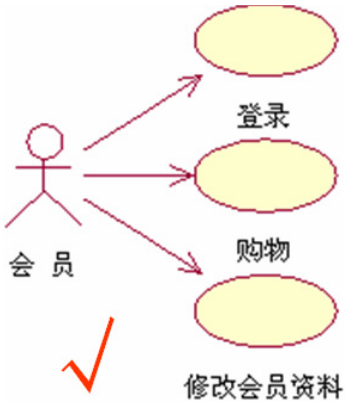
【2】本文的例子全部来源于真实的项目，隐去一些信息。

【3】RUP2002 评估版，http://www.rational.com/products/rup/tryit/eval/gen_eval.jsp。

【4】这里还带出一个经常被问到的问题，“登录”算不算一个用例？关于“登录”，经常有以下几种处理方法：







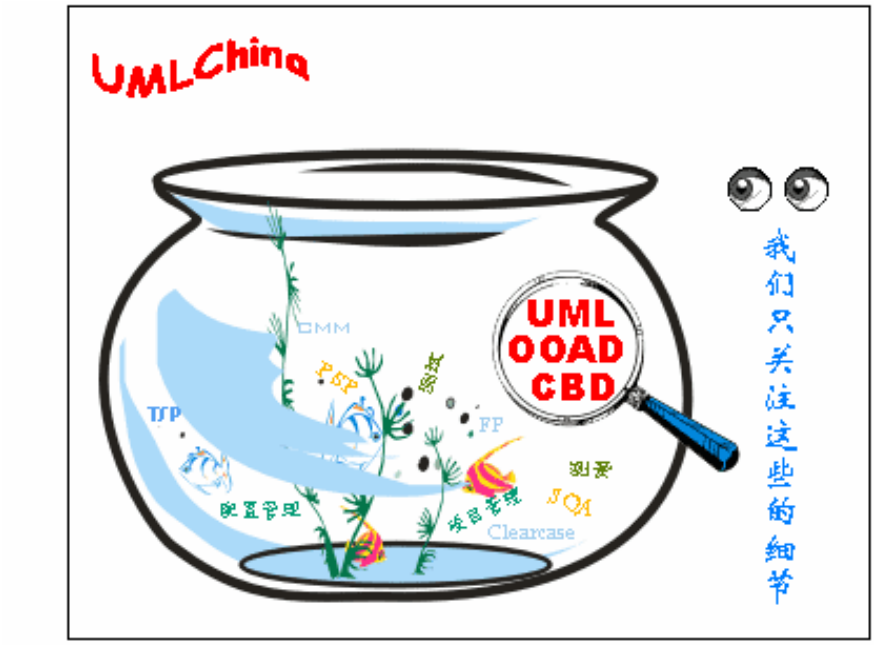
这三种表示方法在技术上都正确，只是写出文档时文档的结构不同而已。第 1 种：把“购物”、“修改会员资料”…作为“登录”的扩展。把“登录”作为基用例，“购物”等（上图所给的用例只是一小部分）作为“登录”的分支，在业务上确实很勉强。第 2 种：把“登录”作为各个需要经过登录步骤的用例的子用例，让各用例共享这些步骤。这是正确的，严格说来登录本身确实不能构成独立的价值，但第 2 种方法也带来了很多 Include 关系，所以类似“登录”这样的入口用例常可以处理成第 3 种：把“登录”看作一个单独的用例，其他用例以“已经登录”为前置条件。第 3 种是最合适的方案。

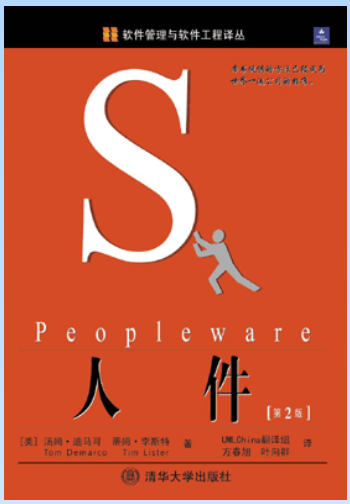
【5】良好的用例开发团队应该有三种人组成：开发人员、业务专家、最终用户。目前的现实中，大多是由开发人员唱主角，后两者断断续续的协助。

【6】可以想一想设计模式中的“Decorator”、“Visitor”、“Bridge”等名字。

【7】“进行××××”的问题，大家感兴趣可以看一看你自己的或者你能看到的“需求规格说明书”（加引号是因为现实中很多所谓的“需求文档”里说的并不是需求，而是设计甚至是无用之物），查找看看里面有多少个“进行”。“进行新装电话单的录入”、“进行统计分析运算”等等。

【8】CRUD 用例泛滥的原因可能是：（1）容易找。相对于要看出 CRUD 背后的业务，找 CRUD 是容易的。（2）容易实现。所以我们平时看到的许多系统都是数据管理工具，一打开就是“新增××”，实用性很差。





Tom Demarco，Tim Lister

翻译：UMLChina 方春旭 叶向群

http://www.peoplewarecn.com

人件中文版网站

Frederick P. Brooks, Jr--《人月神话》第 19 章

近年来，软件工程领域的一个重大贡献是 DeMarco 和 Lister 在 1987 年出版的《人件》，我衷心地向我的读者推荐这本书。

Edward Yourdon

《人件》在 1987 年出版后，立即成为最畅销的作品。当人件第一版出版时，我写了一份评论，“我强烈推荐你买一份《人件》给你或你的老板，如果你是老板，那么为你部门的每一个人买一份，并给自己买一份”。这建议在 12 年后的第二版依然有效，并且更加热烈。

Mark A. Herschberg

我只学了办公环境的章节，就辞掉了原来的工作！

Joel Spolsky

我想，微软成功的原因之一就是公司里的所有经理都读过《人件》

Steve McConnell

由于本书第 1 版的赫赫声名，新版的《人件》是我不用看就会决定购买的少数几本书之一。

预订中文译本>>

基于角色访问控制的 UML 表示

Michael E. Shin、Gail-Joon Ahn 著，UMLChina 译

查看评论

摘要

在基于角色访问控制（role-based access control，RBAC）中，权限和角色相关，用户被当作相应角色的成员而获得角色的权限。RBAC 背后的首要动机是为了简化管理。已经有文章介绍了一些基于角色系统的开发框架，但目前很少有文章使用系统开发者或软件工程师易于理解的方式来阐述 RBAC。统一建模语言（UML）是一种通用的可视化建模语言，我们可以使用它阐述、可视化和文档化软件系统的组成部分。本文使用 UML 表示 RBAC 模型，缩短了安全模型和系统开发之间的鸿沟。我们使用三种视图表述 RBAC 模型：静态视图、功能视图、动态视图。另外，我们简短地讨论了将来的方向。

1 介绍

在 RBAC 中，权限和角色相关，用户（user）被当作相应角色（role）的成员而获得角色的权限（permission）。这大大简化了权限的管理。角色针对组织中的各种功能创建，用户依据他们的责任和资历被指派角色。用户被指派的角色可以容易地从一个跳到另一个。用户对信息的访问在指派角色的基础上被管制。自从 RBAC 被广泛接受，许多安全领域的研究者和安全系统开发者已经花了很多时间来开发基于角色的系统。一些文章介绍了一些基于角色系统的开发框架 [2, 11 , 12]。这些以前的工作有一些很难被开发人员理解，因为它们太抽象、太形式，另外一些则是关注面向应用或专门领域框架的具体解决方案。这些框架都不足够为系统开发者给出一份合理的蓝图。

我们的主要目标是缩短安全模型和系统开发之间的鸿沟。本文使用通用的可视化建模语言 UML 来表示 RBAC 模型。我们选择 UML 的原因是它已经成为建模的标准语言。我们的表示包括 RBAC 模型的静态视图、功能视图、动态视图。

本文按以下方式组织：在第 2 部分，我们描述一种广为人知的基于角色访问控制模型，通常称为 RBAC96。第 3 部分简短介绍 UML。第 4 部分用 UML 表示 RBAC96 模型。第 5 部分给出结论。

2. RBAC 模型

作为有希望取代传统自主式访问控制（DAC）和强制式访问控制（MAC） [3,4, 6, 9]的替代者， RBAC 目前已经得到了很大的关注。RBAC 的策略基于角色，可以使用映射组织结构的方式来阐述安全策略。Sandhu 等发布了称为 RBAC96 的 RBAC 通用模型家族[9]。图 1 展示了家族中最通用的模型。关于在开发这个模型家族时所作设计决定的一些动机和讨论可参见[9]。

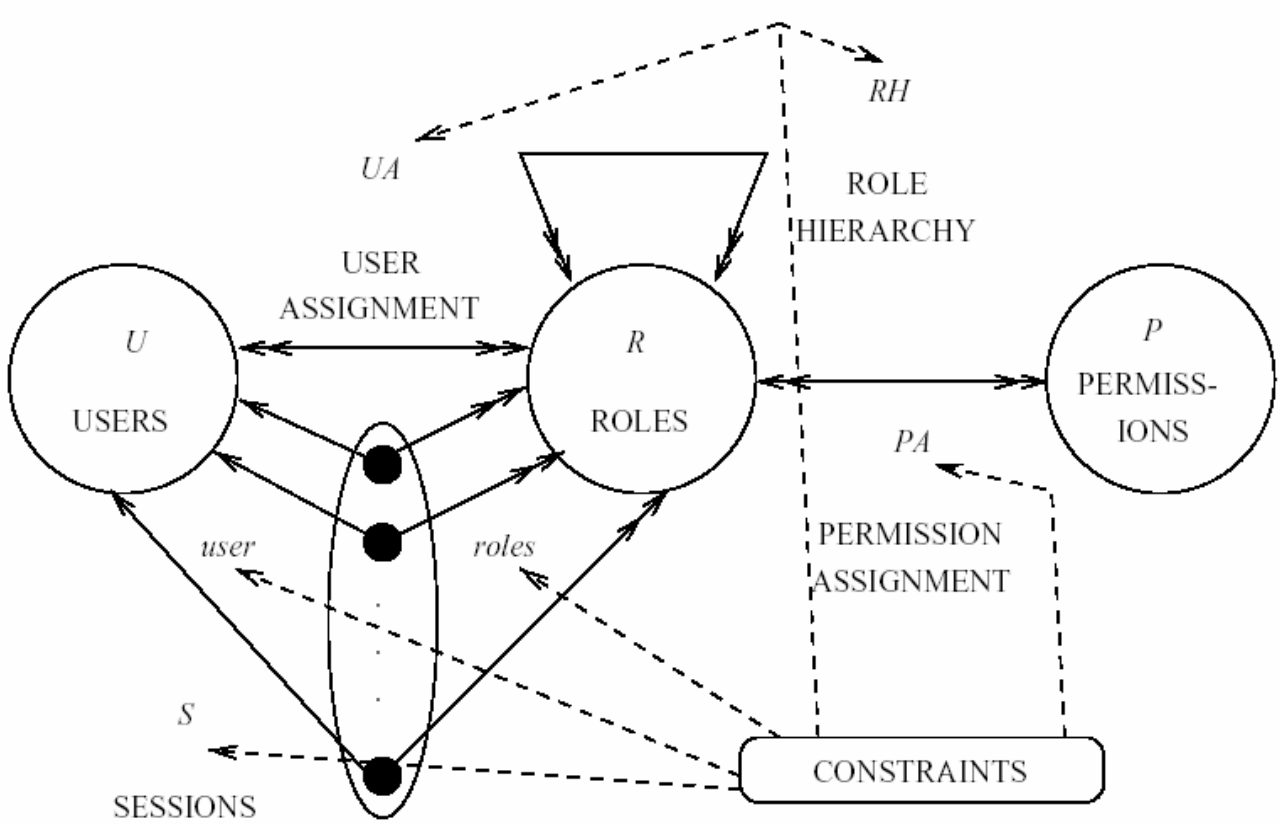


图 1 RBAC 模型

图 1 显示了 (regular)管制对数据和资源的访问的角色和权限。直观上，一个用户是一个人或一个自治的 agent，一个角色是一项在组织中的工作功能或工作头衔。而权限是对系统中一个或多个 object 的特定访问模式的许可或执行某些动作的特权。角色以偏序关系≥组织，如果 x≥y 那么角色 x 就继承了角色 y 的权限。x 的成员也意味着是 y 的成员。每次会话（ session）把一个用户和可能的许多角色联系起来。用户建立一次会话，激活一些他或她是成员（直接获得或通过角色继承的方式间接获得）的角色子集。RBAC 模型有以下组成部分，这些组成部分从以上的讨论中形式化。

一个用户可以成为很多角色的成员，一个角色可以有许多用户。类似地，一个角色可以有多个权限，同一个权限可以被指派给多个角色。每个会话把一个用户和可能的许多角色联系起来。一个用户在激发他或她所属角色的某些子集时，建立了一个会话。用户可用的权限是当前会话激发的所有角色权限的并集。每个会话和单个用户关联。这个关联在会话的生命期间保持常数。一个用户在同一时间可以打开多个会话，例如，在不同的工作站屏幕窗口各一个。每个会话可以有不同的活动角色。会话的概念相当于传统的访问控制中主体（subject）的标记。一个主体是一个访问控制单位，一个用户在同一时间可以拥有多个不同活动权限的主体（或会话）。

3. UML 概览

UML 是通用的可视化建模语言，我们可以使用它阐述、可视化和文档化软件系统的组成部分。现在 UML 已经是软件工程的标准语言。UML 包括用例建模，静态建模和动态建模。在用例建模中，通过用例和执行者阐述系统的功能需求。用例由执行者引发，描述了执行者和系统之间的交互。静态建模提供系统的结构视图信息，在这个视图中，通过类、类属性和与其他类的关系（ relationship）表示。关系包括关联（association）、泛化/特化（generalization/specialization）、聚合。动态建模表现系统的行为视图。它可以通过对象协作图、顺序图或状态图来描述。对象协作图和顺序图用来描述对象和其他对象在执行用例时的协作。依赖于状态的对象使用状态图来描述 [5]。

本文中，我们使用类图、用例图和对象协作图分别作为 RBAC 模型的静态视图、功能视图、动态视图。在本文的剩下内容中，我们使用了 [1, 7, 10]中介绍的 UML 标记符。

4. 基于 UML 的 RBAC 表示

RBAC 中的组成部分是用户、角色、权限、会话和约束。为了使用 UML 表示 RBAC 模型，我们使用对象类的标记分析每个组成部分。在本文后面部分中，我们的分析分为三个不同的视图：静态视图、功能视图和动态视图。

4.1 静态视图

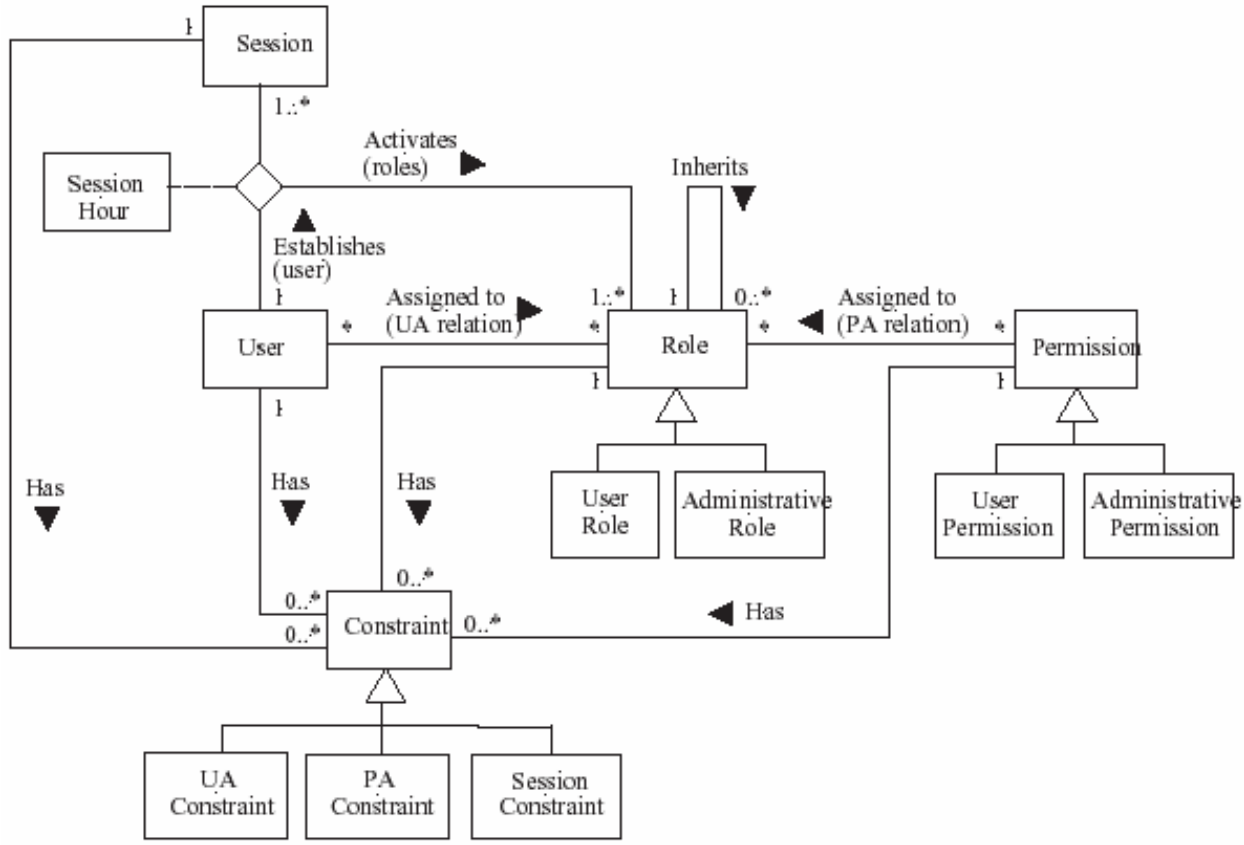


图 2 类图：概念静态模型

RBAC 的概念静态模型如图 2 所示，包括类、类之间的关系和关系的势。基本的实体是用户、角色、权限、约束和会话类。角色和权限类分别特化出两种类别：使用者和管理者。这个特化依赖于用户的资历。RBAC 模型的约束可以有很多种形式，依赖于应用系统。为了简化分析模型，我们静态模型中的约束只有三个：用户约束、权限约束、会话约束。另外，静态模型还有一个特殊的类 session hour，当用户建立一个会话来激活他/她的角色时，该类被使用。这对表达基于会话的约束很有用。例如，一个组织可以要求用户只能在某些时间建立会话。为了强化这类约束，我们需要跟踪每个会话的会话时间。实体类的属性在图 3 中定义。

«entity» User	«entity» Role	«entity» Permission	«entity» Session	«entity» SessionHour
user : String roleList: List	role : String subrole: String	permission : String roleList: List	sessionName String roleList: List	sessionName String startTime: List

图 3 实体类的属性

在静态视图中，UA 关系和 PA 关系用多对多的“assinged to”关系表示。用户－会话关系看作一个用户可以建立一个或多个会话，在常数的会话生命期中每个会话激活至少一个或多个角色。角色继承关系表示为一个角色 inherits 另外的角色。

4.2 功能视图

本文中，我们也作了更具体的功能需求来表示在图 2 中没有清晰定义的、RBAC 系统应该提供的功能。使用用例模型描述的功能视图如图 4 所示，有三个执行者：安全管理员（security administrator）、用户（user）和角色领域工程师（role domain engineer）。角色领域工程师从可能组织一组权限的应用系统中抽取出基本的知识，构造角色层次和指定约束。安全管理员管理基于角色的系统，指派用户到角色和指派权限到角色。用户是真实的人或外系统，可以建立会话、请求权限许可和关闭会话。

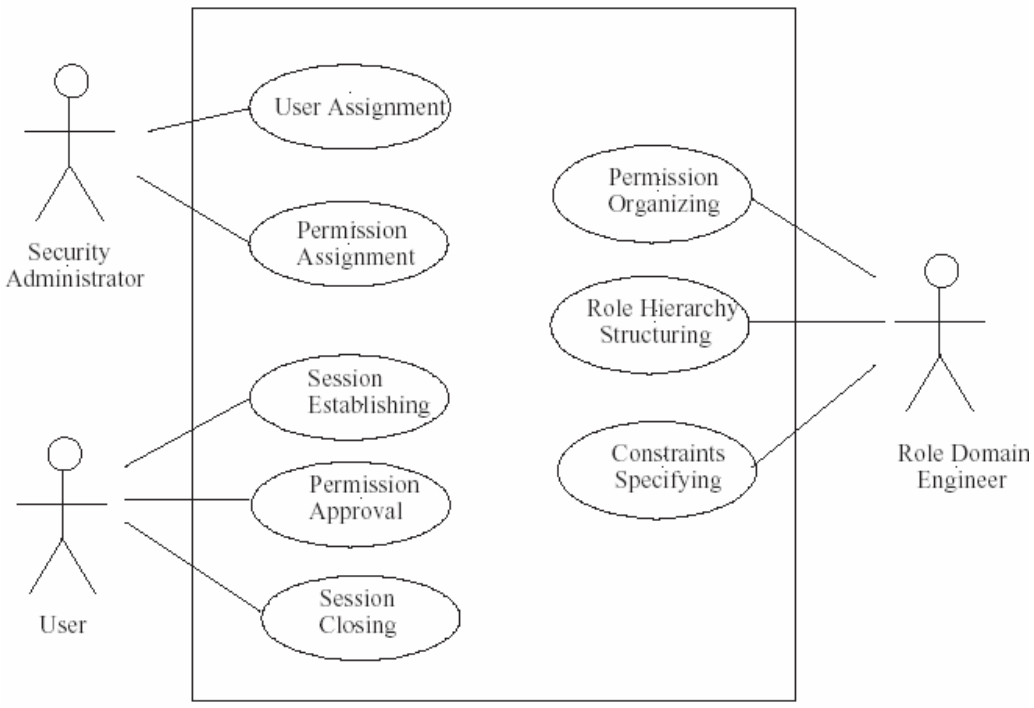


图 4 用例模型

下面是建立会话（session establishing）用例的简要叙述：

用例：建立会话

执行者：用户

前置条件：系统空闲

说明：用户提交信息以建立会话；系统显示用户可以激活的角色；用户选择要激活的角色；系统以用户所选择的角色激活一个会话。

在以所选择角色建立会话之后，用户可能需要访问需要基于角色信息的授权过程的系统资源。换句话说，和角色相关的权限应该得到系统的许可。以下是“权限许可”用例的简要描述：

用例：权限许可

执行者：用户

前置条件：用户的会话已经激活

说明：用户提交权限许可信息，系统通知用户权限是否许可。

我们也可以考虑其他情况。例如，我们可能需要额外的功能来让安全管理员监视会话或让用户查询自己的状态。

4.3 动态视图

在动态视图中，用例被精化以显示参与用例的对象之间的交互。“建立会话”的协作图如图 5 所示。

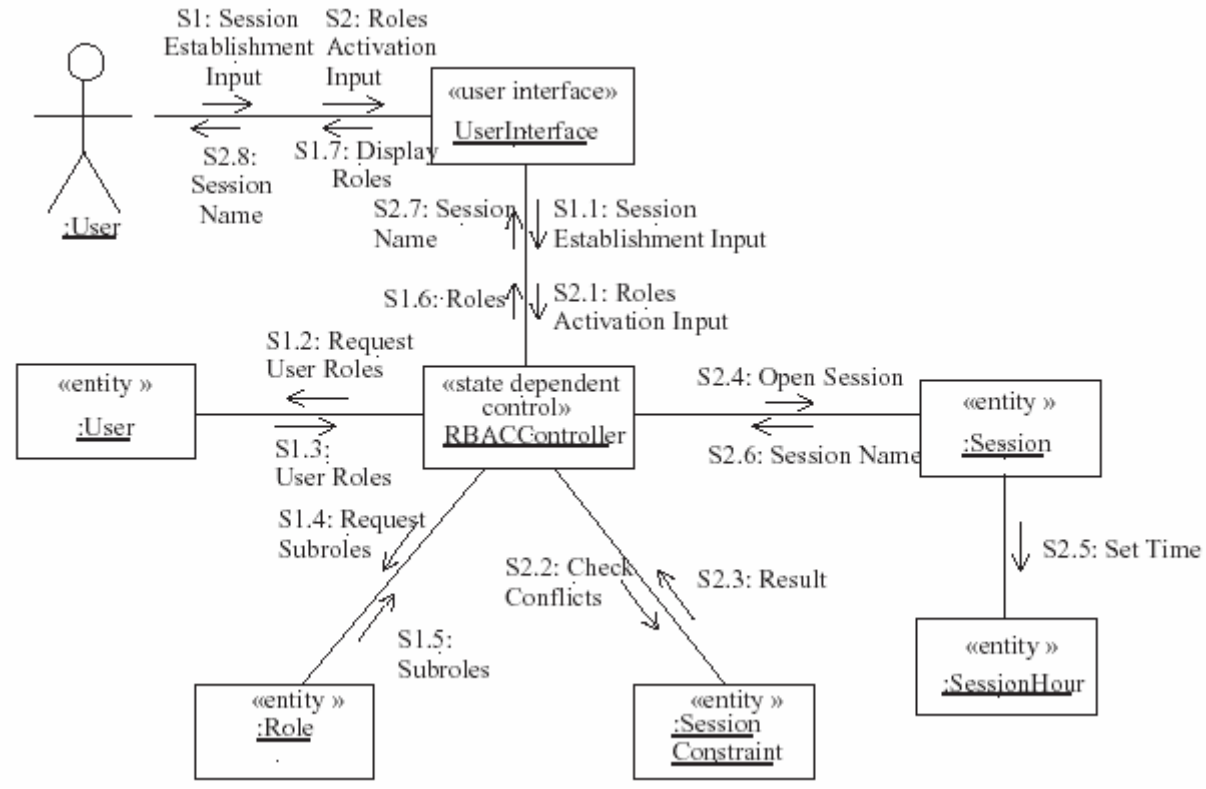


图 5 协作图：建立会话

用户通过 UserInterface 引发用例， RBAC Controller coordinates 协调对象之间的交互。“权限许可”的协作图如图 6 所示，需要一个前置条件：会话已经被激活。

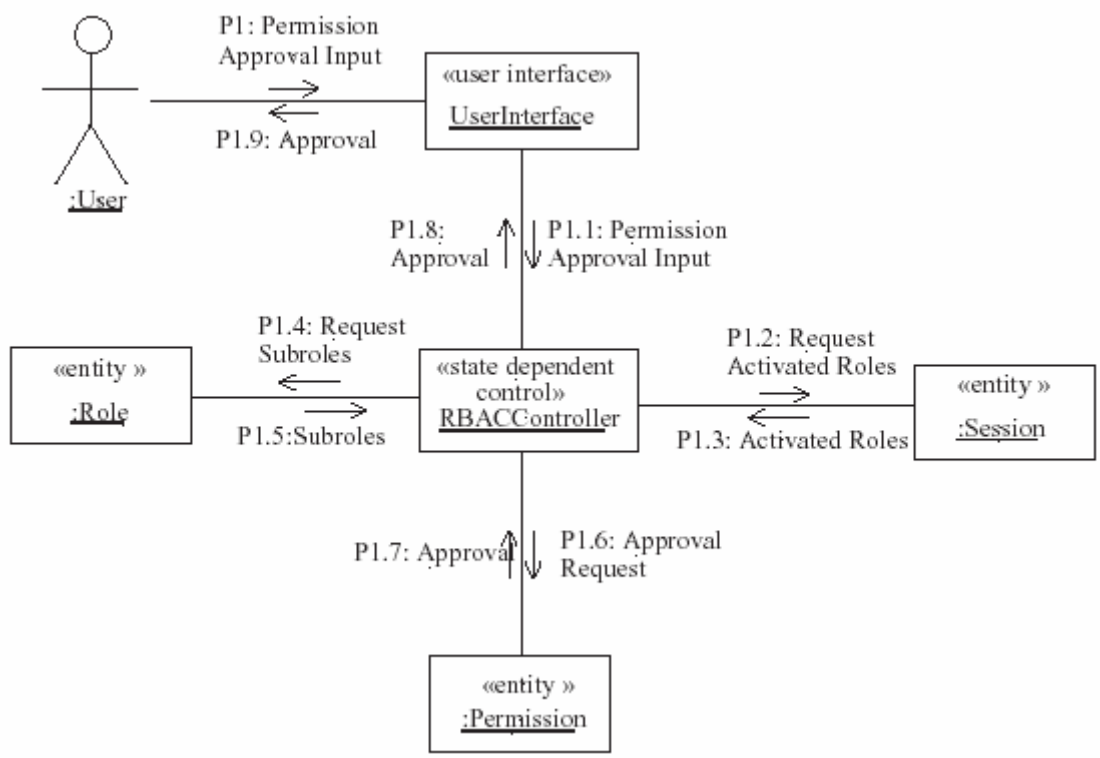


图 6 协作图：权限许可

5. 结论

本文中，我们简要描述了广为人知的基于角色访问控制模型。我们使用可视化建模语言 UML 来描述这个模型。这是使用建模语言描述 RBAC 模型的第一次尝试。我们相信我们的工作可以帮助系统开发者更容易地理解 RBAC 模型和建立基于角色的系统。我们也识别了在安全模型的初期被排除的有用功能和约束。在此工作的基础上，我们将研究基于 UML 的模型如何能适应于阐述 RBAC 的组成部分。也包括如何用可能的 UML 扩展表示角色继承和约束。因为模型通过简化细节来帮助我们理解系统，这个方向应该比较实用。

致谢

感谢 Jaehong Park 的帮助审阅此文。

参考文献

[1] G. Booch, J. Rumbaugh, and I. Jacobson. The Unified Modeling Language User Guide. Addison Wesley , 1999.

[2] Pete Epstein and Ravi Sandhu. Tow ards A UML Based Approach to Role Engineering. In Pr oceedingsof the 4th ACMWorkshop on Role-Based AccessControl, pages 135-142, Fairfax, VA, October 28-29 1999.

[3] David Ferraiolo and Richard Kuhn. Role-based access con trols. In Proceedings of 15th NIST-NCSC National Computer Security Conferenc e, pages 554-563, Baltimore, MD, October 13-16 1992.

[4] M.-Y. Hu, S.A. Demurjian, and T.C. Ting. User-role based security in the ADAM object-oriented design and analyses environment. In J. Biskup, M. Morgernstern, and C. Landwehr, editors, Database Security VIII: Status and Pr ospects . North Holland, 1995.

[5] Hossan Gomaa. Object Oriented Analysis and Modeling for Family of systems with the UML. T echnicalReport, ISE Dept. George Mason University, 1999.

[6] Imtiaz Mohammed and David M. Dilts. Design for dynamic user-role-based security. Computers & Security, 13(8):661-671, 1994.

[7] J. Rumbaugh, G. Booch, and I. Jacobson. The Unified Modeling Language Reference Manual. Addison Wesley , Reading MA, 1999.

[8] Ravi S. Sandhu. Lattice-based access control models. IEEE Computer, 26(11):9{19, November 1993.

[9] Ravi S. Sandhu, Edward J. Coyne, Hal L. Feinstein, and Charles E. Y ouman. Role-based access con trolmodels. IEEE Computer, 29(2):38{47, February 1996.

[10] National Soft w areet al. Unified Modeling Language Notation Guide, Version 1.1. September 1, 1997.

[11] Charles Y ouman, Ed Coyne, and Ravi Sandhu, editors. Pr oceedingsof the 2nd ACM Workshop on R ole-Based Access Control, Nov. 6-7. ACM, 1997.

[12] Charles Y ouman, Ed Coyne, and Ravi Sandhu, editors. Pr oceedingsof the 3rd ACM Workshop on R ole-Based Access Control, Oct. 22-23. ACM, 1998.

关系数据库访问层 一种模式语言（关键模式）

Wolfgang Keller、Jens Coldewey 著，Happy 译

查看评论

摘要

关系数据库访问层可以帮助您设计使用关系型数据库的应用程序，并且在业务对象级别反映关系运算，这种应用程序被称作数据驱动或表示[Mar95]。这种系统并不需要是面向对象的，您完全可以使用第三代语言（3GL）来实现，因此，本文的模式语言将忽略映射继承和多态的特性。

本文包含框架模式和几个关键的实现模式。PLoP 学报[Kel+96b]包含了完整的模式语言，包括适配器和优化模式。

简介

很多业务信息系统拥有一个简单的数据模型，即使他们可能会有 30 或更多个实体表，但是很少发现继承或者复杂的关联。对于复杂的情况，通常是封装在应用程序的核心。用关系运算来为这些系统建模是一个很好的主意。

来看看下图 1，从一个订单管理系统中摘录的片断，在左上角，有处理发货单的实体，这个片断遵循第三范式（3NF），在因数分解级别，数据分析家经常这么用。假设你使用这种逻辑数据模型来定义你的物理数据库表，系统当然可以正常工作，但是性能却是不敢恭维。

剖析这个系统，你将发现很多多余的数据库操作，同时，也会发现数据库操作过于缓慢的原因是大量的表连接或者是不必要地移动大量数据。为了提高性能，你可以在物理数据模型中不用太过范式化。对数据库内容的统计分析表明 90%的订单拥有不超过 5 个订单项（OrderItems），因此可以将前 5 个 OrderItems 放在 Order 表中，为包含剩余 10%，可以创建一个 OrderItemOverflow 表，如图 1 右上角所示，此外，你还可以将物品的属性 ArtPrice 和 ArtName 集成到 Order 表中。这样，最后的数据库设计使得访问两个表，一个 Order 表，一个 Customer 表，就可以读取 90%的发货单，而其他的连接都可以去掉了。

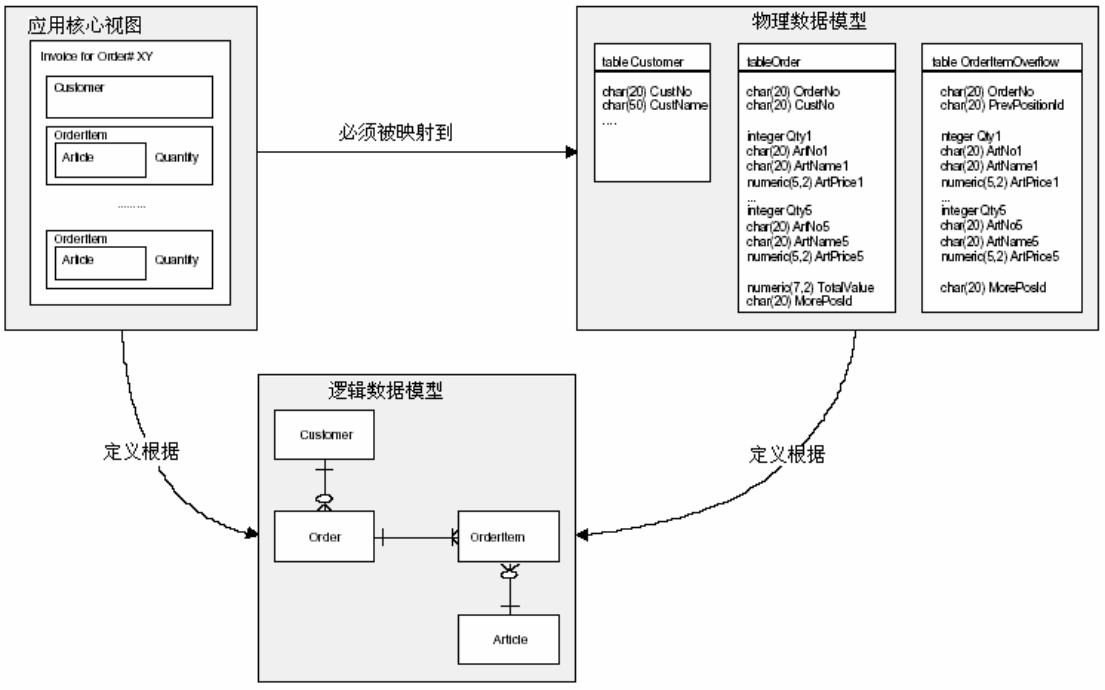


图 1：订单处理系统的部分

现在假设你在应用程序的核心代码中嵌入了 SQL 语句，为适应新的表结构，你将不得不重写其中的某个部分，此外，处理这些溢出表(OverflowTable)，将使 SQL 代码暴露出来，最坏的是，你还不得不在每次数据库结构更新时重复这些过程。

模式语言图

关系数据库访问层框架模式定义了它的部件的角色和职责，同时也阐述了三个关键抽象，层次视图、物理视图和查询代理。

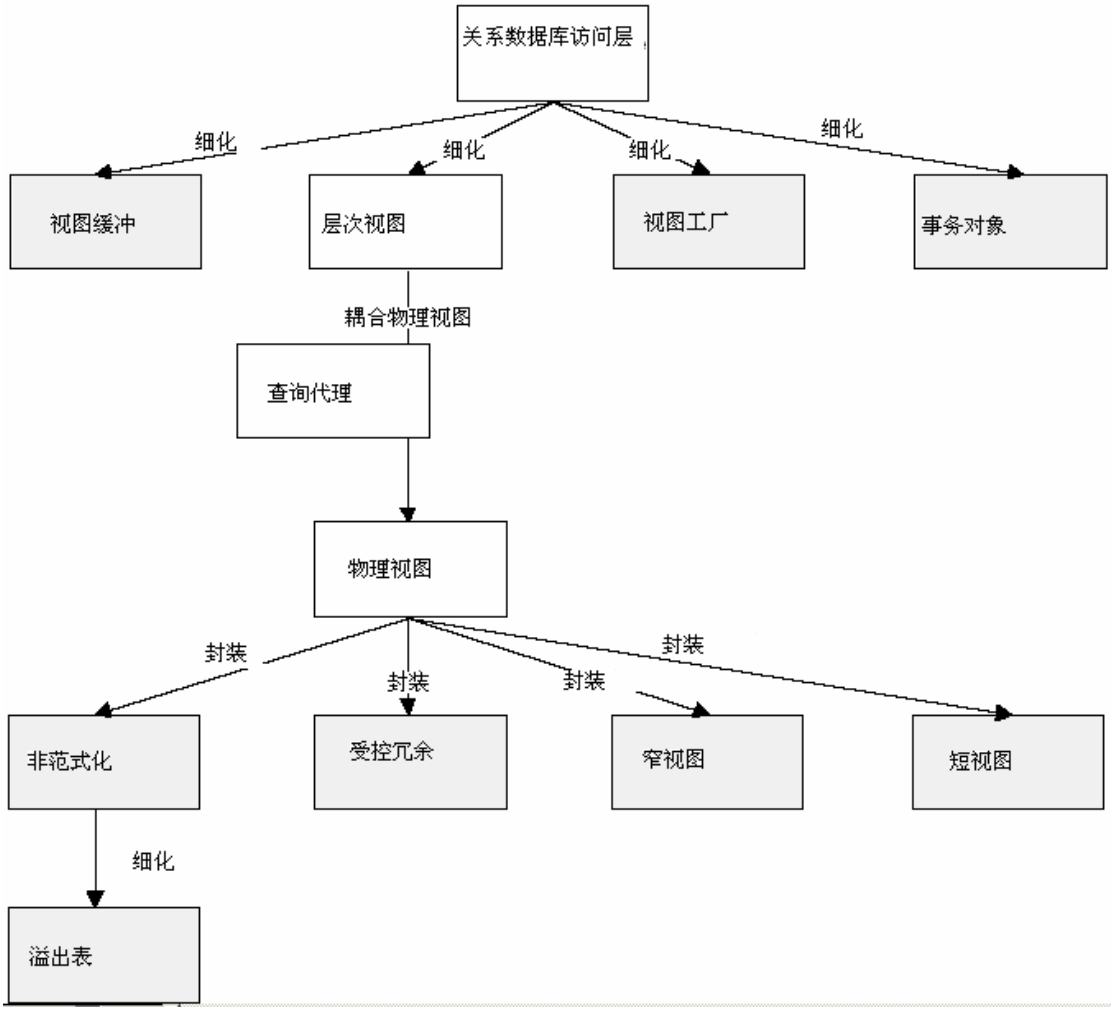


图 2：该模式语言的映射图。白框中的模式表示在本文描述的模式，灰框中的模式只在 PLoP 学报中描述[Kel+96b]。

数据库访问层的相关工作

这个框架为那些将数据库以关系型方式使用的应用程序提供了一个数据库视图接口，其它的持久化框架和模式语言都是关于对象的持久化，例如对象－关系的访问层[Bro+96, Col+96, Kel+96a]，它使用一个关系型数据库；还有对象访问层，它使用一个面向对象数据库[Col97]。图 3 显示了这些不同访问层的区别。

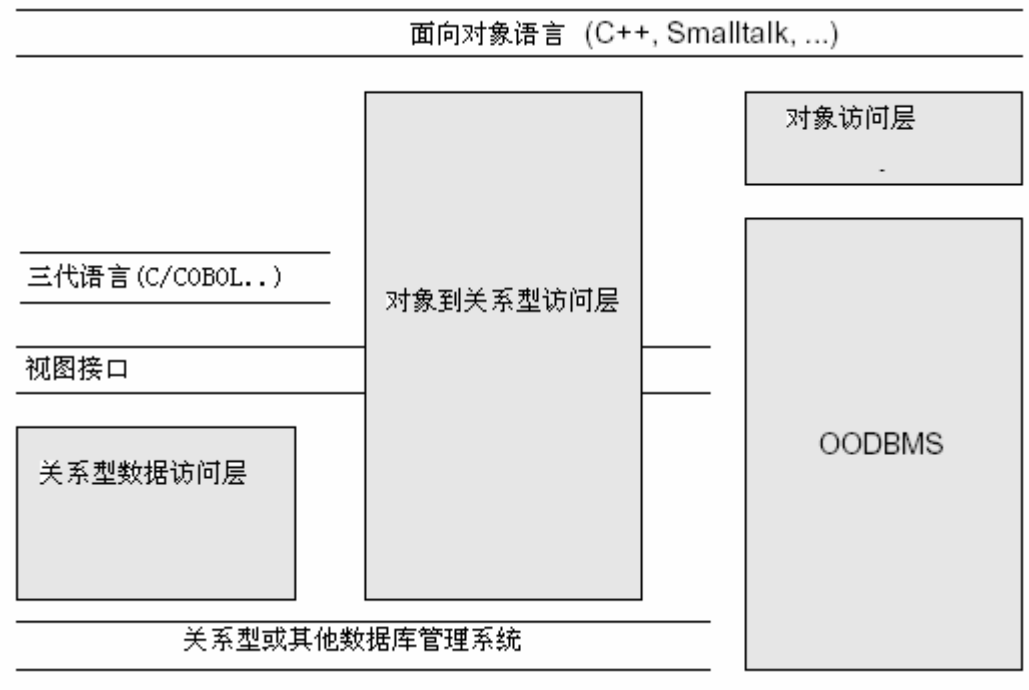


图 3：三种不同的数据库访问层

符号约定

我们使用 OMT[Rum+91]来表示对象图，带下划线的其他模式参考一个相关的模式。如果一个模式参考后面跟着一个引用，例如[GOF95]，你可以从相应参考论文中找到它。

框架模式

模式：关系数据库访问层

环境

你如果正在编写一个类似上面订单处理系统的业务信息系统，关系型运算比较适合这个域逻辑，最后的数据模型非常简单，很少使用继承，而将关系模型映射到面向对象表示的工作巨大，得不偿失。

问题

如何访问关系型数据库？

先决条件

- 业务分离与成本：数据库编程很复杂，应用逻辑也是如此，都需要有灵活的解决方案。将它们混合起来的复杂程度要高于它们两者复杂度简单的组合。最容易的方法是将应用编程和数据库编程分离开，两者都更易于实现和测试。另一方面，引入新的层，将增加类的数量，并加重设计和实现的工作，需要花大力气在提高维护性和性能调优上。

- 易用性与功能：如果你决定要封装数据库，最后的接口必须易于使用。而另一方面，数据库接口的复杂度将影响它的功能，因此，接口的封装既要易于使用，功能也要足够强大以满足你的项目。

- 性能：要想业务信息系统达到一个可接受的性能，数据库优化是关键的。因为数据库比一个主处理过程要慢几个数量级，调优的操作将集中在数据库访问上。调优是一个迭代的过程，为优化数据库访问，你可以改变存储的物理参数，以及表的分布或访问数据库的 API。

- 灵活性与复杂性：因为数据库调优非常关键，需要对数据库有一层封装，允许在应用核心不变的情况下，频繁改变底层的数据模型。因此，系统的灵活性越高，复杂性就越大。

- 旧系统与优化设计：很少从零开始设计一个业务信息系统，而是需要联接到一个旧系统，并且根本就不允许碰它。通常你不可能替代所有的遗留代码，因为这种方式有很高的风险并代价不菲。但是，遗留数据的结构很少符合你的需要——如果还有结构的话？你也许不得不将不同年代的数据库技术桥接起来。为保证应用的可维护性，你不得不封装遗留的访问接口，尤其在重建工程中，这是一个强烈的先决条件。

解决方案

使用一个分层的架构，包含两层。逻辑访问层提供稳定的应用核心接口，而物理访问层访问数据库系统，后者可以适应修改性能的需要，在两者之间使用一个查询代理将他们耦合起来。

结构

图 4 显示了关系数据库访问层的类。逻辑访问层提供缓冲和事务管理的类，物理访问层提供访问数据库系统的接口。后者细分成表示数据访问的物理视图和 Database 类，Database 类用来封装数据库管理的调用。这两层可以直接硬连接，或采取更好的方式——用一个查询代理居于逻辑和物理访问层之间。

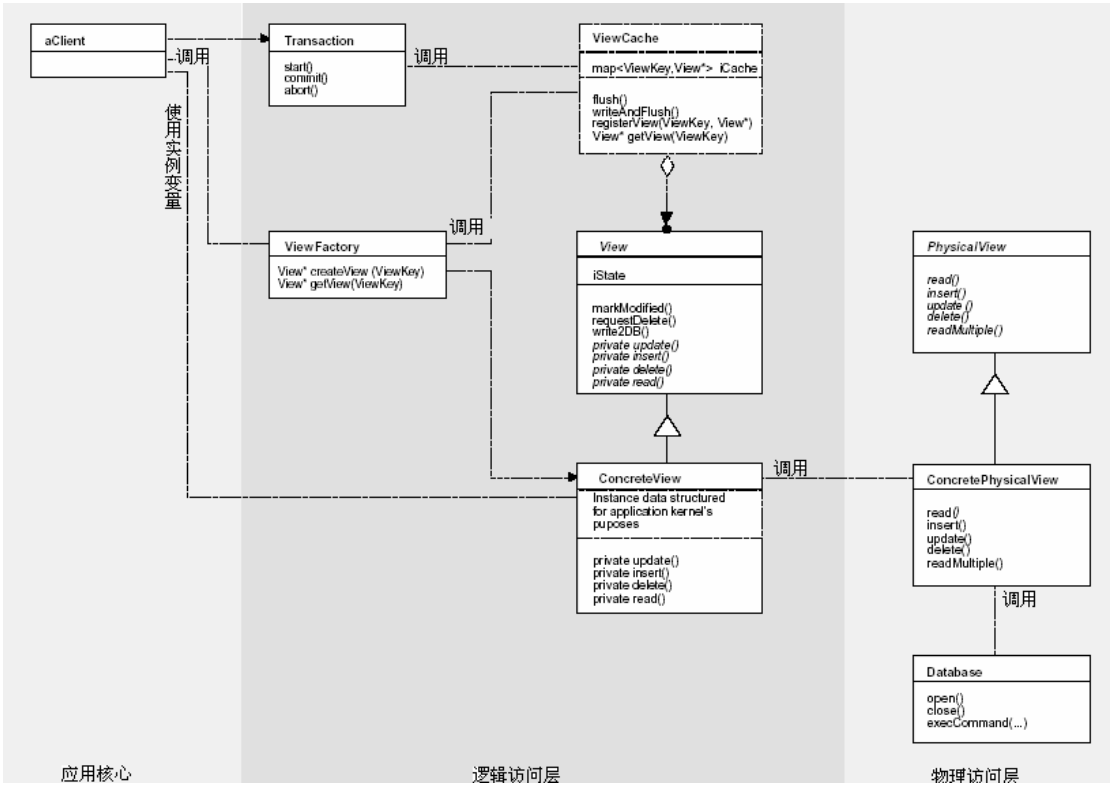


图 4：关系数据库访问层框架的结构。客户端只访问逻辑访问层，而逻辑访问层使用物理访问层联接到数据库。

参与角色

事务（Transcation）

- 提供一个允许事务开始、提交和回滚的接口；
- 它在每个事务开始前创建，并在提交或回滚后销毁。这种用法类似于定义在[ODMG96, chapter2.8]中的事务对象。

视图工厂(View Factory)

- 转送由键值标识的数据。它提供 createView()和 getView()方法，分别来创建新视图和激活已有视图。客户端只有通过这两个方法才能得到视图的引用；
- 使用一个视图缓冲（View Cache）[Kel+96b]来避免创建在当前事务环境中已经存在的视图；
- 允许标识 ConcreteViews 的判定，用一个抽象的键值类 ViewKey 为所有的键提供标准接口；
- 它是一个 Singleton 对象[GOF95]。

视图缓冲（ViewCache）

- 避免数据装入两次，它是一个键值索引的视图包容器，形成访问层的缓冲；
- 提供 writeAndFlush 方法，可以将所有修改过的视图使用 write2DB 方法写入数据库。事务对象在提交时，会调用 writeAndFlush，而中止事务将调用 flush()来清空 ViewCache。

具体视图（ConcreteView）

- 它是逻辑数据库模型的层次视图（Hierarchical View），有若干个具体视图类。每个类适应于应用核心某个用例，ConcreteView 成员的类型是应用程序数据类型，而非数据库类型。
- 知道如何通过具体物理视图（ConcretePhysicalViews）将它们自己写入数据库。具体视图将监视它的内部状态，如果它从视图缓冲接收到一个 write2DB 消息，将调用私有的 update()、insert()或 delete()方法。这些方法中即可以强制编码调用一个相应的具体物理视图，也可以调用一个查询代理（Query Broker）。

视图（View）

- 为具体视图(ConcreteView)定义抽象的协议，参见层次视图模式；
- 提供一个 markModified()方法，在当前事务提交时触发数据库更新；
- 提供 requestDelete()方法，在事务结束时触发数据库删除。不要将这个方法和析构函数混淆起来。析构函数只是将对象从内存清除，而 requestDelete()是在事务结束后将对象从数据库（同时也从内存中）删除。为了避免悬空引用，requestDelete()方法应该是唯一删除数据库记录的方法。

物理视图（PhysicalView）

- 为具体物理视图（ConcretePhysicalViews）定义统一的协议；
- 捆绑数据库访问函数，封装数据库行为并将数据库错误码转换成应用级别错误码；

具体物理视图（ConcretePhysicalView）

- 包装一个物理数据表。它也可以包装数据库视图，如果数据库不支持视图的直接更新，具体物理视图还要提供适当的写命令；
- 包装数据库优化。如果使用非范式化、受控冗余或溢出表，这种情况下，具体物理视图可以映射多个表；
- 可以从元数据信息生成，如数据库的表结构。

数据库（Database）

- 封装数据库管理系统。提供开始数据库联接的方法，以及处理数据库命令，接收结果集等方法。

动态行为

我们将讨论这个模式的动态行为，实现框架的不同方面。

实现

- 对待大量更新（Mess Update）。大量更新语句的形式诸如“update..where”，使用一条查询操纵一组记录。很难将这些语句集成到视图缓冲中。合并大量更新的意思是：读入大量数据到视图缓冲，单条操纵记录并在写回数据库时，每次写一条，这个方法比直接在数据库处理要慢得多。

- 批处理需要特别对待。有一些模式描述批处理方式访问数据库，不过还有待挖掘；

- 多重查询：我们忽略了多重查询，可以在短视图（ShortView）和窄视图（Narrow View）中找到更多信息。

- 游标稳定性：将大量数据读取操作提交到 BFIM（前映像）一致性检查是具有理论可行性的。这将提供二级事务一致性（游标稳定性[Gra+93]），而非一级（浏览一致性）。大量数据读取通常用来填充列表框（参见短视图）。它们具有诸如“select <fileds> from … where”的形式。在提交时检查它们的一致性意味着在事务中，重新读取所有已读记录，并和它们先前的映象比较。如果有一条记录不同，将不得不终止事务。这不仅仅对性能是一个严重的威胁，同时对一致性也没有什么价值。大多数情况下，用来填充列表框的记录不会扮演破坏任务一致性的角色。因此，在事务中，对于不参与计算的数据，通常使用浏览一致性就足够了。

- 具体物理视图和动态 SQL：如果数据库系统支持动态 SQL 而没有什么运行时的负担，你可以跳过具体物理视图而使用查询代理来生成相应的 SQL 语句。静态的 SQL 由具体物理视图提供。

- 数据库联接尽可能长时间地保留。为每个事务建立一个新联接将导致槽糕的性能。

- 在这个架构下，强烈建议不使用触发器和包含业务逻辑的存储过程。一个视图缓冲将无法被通知到数据库中自治的变化，因此存储过程可能会导致缓冲的一致性问题。同样，触发器也有类似问题。因为它们工作于物理数据模型，很难转换到应用核心的逻辑层，不过也可以为了提高数据访问速度（见物理视图）使用受限的存储过程。

结论

- 业务分离：访问层对事务、数据库访问、缓冲形成了封装良好的子系统，应用核心使用逻辑层接口，无需对数据库的访问有了解。

- 工作强度：实现一个关系数据库访问层根据包含的特性不同，需要 0.5 到 35 个人年，使用生成器和依赖硬编码比构建维护工具的代价和查询代理的代价要低，在你决定不同的选择前，充分考虑预期的变化、市场推广和软件的生命周期。

- 易用：这个访问层并没有将关系模型转换到一个面向对象视图，因此应用核心必须工作在关系视图上。你应该仔细考虑是否这种数据驱动的方式适合你的应用逻辑，看看是否你的工程更适合使用对象－关系的访问层。当应用核心确实需要一个隐式的映射时，避免构建一个更复杂的访问层并不是一个好的想法。

- 灵活性：如果你使用一个查询代理，你可以通过增加新的具体物理视图来维护并调优数据库，而不用修改应用核心代码。在底层物理数据库改变时，应用代码保持稳定。

- 复杂性：这个访问层包含最简单的类。查询代理是成本最高的部分，因为它包含一个复杂的树匹配算法。忽略它可以导致一个简单的适配器层，但是灵活性降低了。

- 性能：你要为映射付出一点运行时性能的牺牲，不过访问层可以使用缓冲来优化并易于调优，它将使用快速的处理器周期，而避免缓慢的 IO 处理。

- 遗留数据：你也许会使用访问层来降低现有应用的物理、逻辑数据模型间的耦合，这对重构遗留系统非常有用。首先，在代码中插入一个数据访问层，这是一个具有可控风险的单一步骤，接着，开始在不同的项目中重写数据库和应用核心。

变种

- 舍去视图缓冲：如果你不需要长事务，可以舍去视图缓冲。这对简单的对话框系统是可行的，这种系统在每个事务中一般之处理一条记录。但是如果应用核心有一些影响多条记录的事务时，就应该使用视图缓冲。

- 对于在事务监控器之上实现用户事务，一个缓冲机制是非常自然的选择，例如 IMS、CICS 或 UTM。当用户事务包含多步对话框才完成，事务监控器为每一步开始一个新的事务，使用视图缓冲可以让你收集到一个用户事务中所有的写数据库操作，接着，它们可以作为一个技术上实际的事务执行，可以对这些多步的对话框操作保持事务一致性。

- 使用非关系型数据库：物理访问层可以封装非关系型数据库和文件格式，例如 IMS-DB，CODASYL 或 VSAM。甚至可以将它改写以适应若干不同的数据库技术，隐藏对遗留数据的访问。

已知应用

VAA数据管理器规范使用这个模式，并带有元数据编辑器和对层次数据库系统复杂的映射[VAA95]。VAA数据管理器是从Württembergische Versicherung[Würt96]的数据管理器派生而来的。

Denert在[Den91,pp.230-239]中简单描述了这个模式语言的一些基本思想，sd&m的许多项目都使用这个模式的各种变种，包括Thyssen、Deutshche Bahn和HYPO银行[Kel+96a]。

CORBA持久对象服务（POS）[Ses96]指定持久对象使用一个代理（持久对象管理器）来将数据写入任意数据存储中（持久数据服务）。

相关模式

这个模式一个分层（Layer）的应用[Bus+96,pp.31]。视图工厂一个抽象构造器[Lan96]的应用。

[Bro+96]和[Col+96]阐述了如何扩展这个模式，向应用核心提供面向对象的视图。Brown和Whitenack[Bro+96]使用一个代理来降低层之间的耦合，而[Col+96]描述了一种直接连接的方法。

一些实现模式

模式：层次视图

示例

考虑图5中，订单处理系统的细节。其中的发货单可能具有右图所描述的情况，注意这个发货单有两级间接的层次结构。一个用例可能就是从一个订单号开始并浏览它的各条订单项和物品。

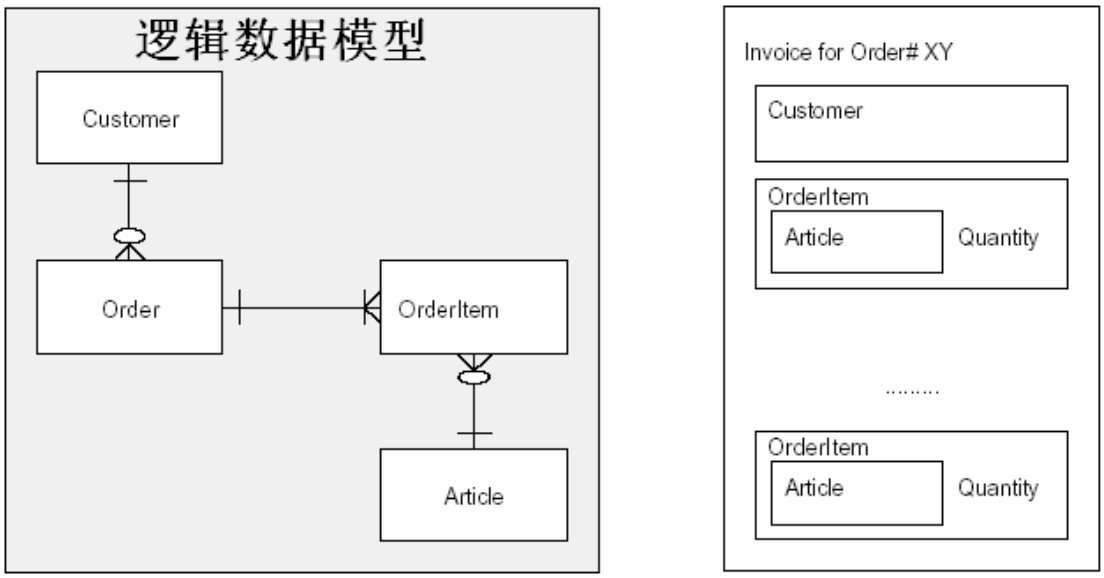


图5：我们订单处理系统数据模型的详细。左边用第三范式表示的 ER图，右边是发货单的结构，由这些实体构成。

环境

你已经决定使用关系数据库访问层来降低物理数据库和应用核心的逻辑数据模型之间的耦合。

问题

数据库访问层向应用核心提供什么接口？

先决条件

- 复杂度与易用性和开发成本：接口应该简单易用而且要有足够的功能来满足必要的数据库操作。因为应用核心反映域问题，你不会想用数据库特定的代码来打乱它，所以这个接口应该在某种程度上反映适当域级别的数据抽象。提供一个包含所用数据库功能的接口意味着重新实现大部分数据库管理系统，这个代价太高了，所以，SQL风格的接口不大可行。

- 灵活性与开发速度：在数据库访问时，使用逻辑数据模型作为指导对于短期基本功能实现可能是最简单的解决方案。但是，性能调优和维护迫使你要经常改动物理数据库的布局，同时我们也不想改变应用核心，所以我们需要一种和物理数据模型无关的接口。

- 性能：理论上，第三范式对于关系运算是最好的，但是在物理模型中用它将导致很差的性能。

- 大量问题：一个大型的数据模型包含上百甚至更多的实体，手工地为数百个实体编写包装函数或嵌入的SQL代码是一个无聊而又代价昂贵的任务，枯燥的工作还总是伴随错误发生。一般的解决方法是在数据库编程中使用宏、生成器或模板。

解决方案

根据域问题空间来表示接口，那就成为一个关系数据模型。从数据模型的一点（或实体）开始，并使用外键关系来行走到其他感兴趣的点。在行走过程中形成形成了一个有向无环图（DAG），每个节点标记上实体名，相关属性和选择谓词，每条边标记上所使用的外键以及它的粒度（一对一或一对多）。

结构

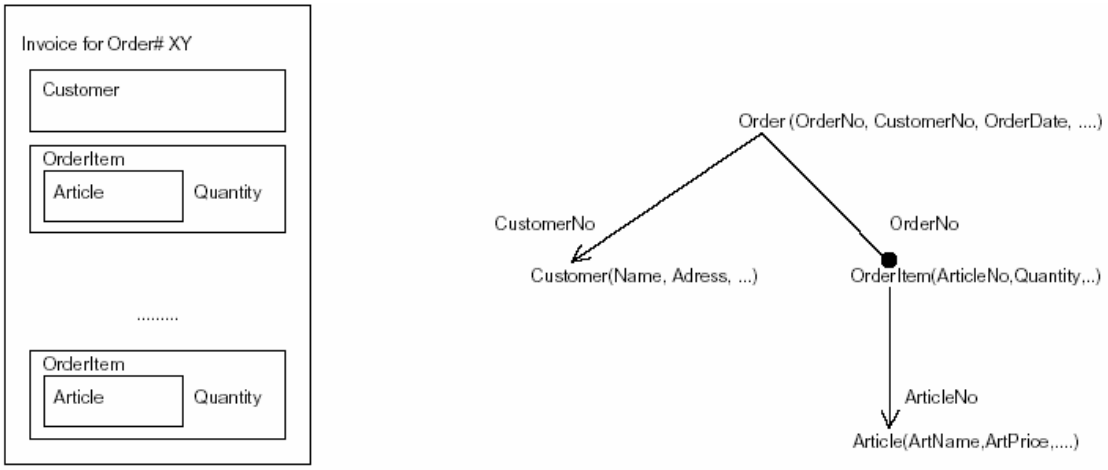
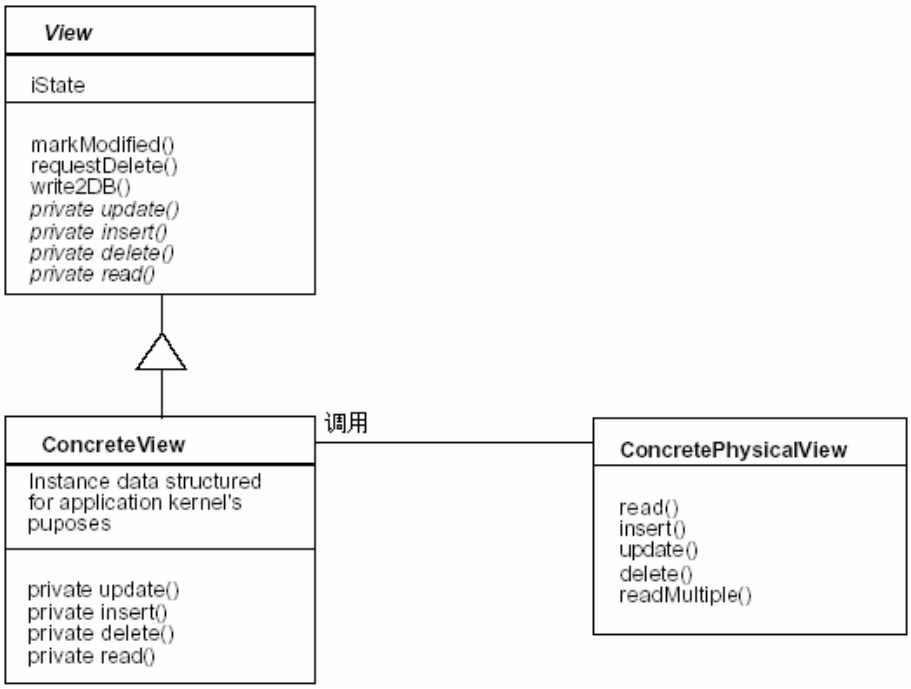


图6显示了相当于左边发货单的图表示

要将这个图转换成一个层次视图，从View派生一个ConcreteView，这个ConcreteView是DAG的根，再为任何一个节点定义一个域级别的类，使用聚集来实现图中to one的关系，使用包容器来实现图中to many的边。一个ConcreteView通过这种方式构建，并填充以来自ConcretePlysicalView域级别属性，适当的ConcretePhysicalView可以使用硬编码方式调用或使用查询代理。

数据库访问层应该能够对所有ConcreteViews统一对待，因此，View定义了它们共通的接口去访问访问层中其他的类。



实现

你可以使用文本文件或者一个特定的工具[Wurt96]来定义ConcreteViews的结构，这允许为静态类型语言自动生成ConcreteViews类，而对于动态类型语言可以进行运行时定义。

结论

- 继承和多态性：访问层没有内嵌继承和多态性的处理，看看这是否适合你的问题域。
- 接口复杂性：这个接口是最小化的，因为它仅提供应用核心所需的基本特性。然而，你不得不花点努力在生成器或模板上，这在调优时会对你有所回报，但同时，在你达到真正目的前，需要经过几个维护周期。一旦你完成了生成器，那么你定义新的ConcreteViews将会是几分钟的事情。
- 接口风格：一个使用层次视图的应用程序依据逻辑数据模型的结构，逻辑数据模型决定了使用它的代码的结构。而对于一个对象－关系访问层中，对象模型依照域的内部结构。
- 易用性和应用核心的需求：层次视图只反应特定域逻辑，它们是相应用例所需要的，因为层次视图封装了数据库特定功能，所以调用访问层是非常简单的。
- 性能和灵活性：层次模型完全降低了应用核心和物理数据模型的耦合度，这允许你任意调整数据库而无需影响应用核心代码。最后的性能总体所得要高于引入层次视图这一间接层的性能损失。
- 大量问题：最坏的情况是每个用例都需要一个或更多的ConcreteView，这会导致大量的类产生：大量代码，大量符号表等等。不过，使用模板、宏或代码生成器，ConcreteViews是足够一般的类。

示例方案

程序段1显示了发货单示例的声明，程序段2包含处理发货单的代码。

```
struct Customer {
CustomerKeyType iCustNumber;
... // 逻辑数据模型中客户其他属性
};
```

```
struct Article {
ArticleNumberType iArticleNumber;
... // 其他属性
};
struct OrderItem {
Article iArticle;
QuantityType iQuantity;
};
class OrderInvoiceView : public View {
public:
OrderKeyType iOrder;
Customer iCustomer;
Vector<OrderItem> iItems; // 其他包容器也可以
Money iSumOfInvoice;
private:
// 私有方法用以读取和写入数据
// 到PhysicalView
    virtual void update ( void );
    virtual void insert ( void );
    virtual void remove ( void );
    virtual void read ( void );
};
```

程序段2的代码和数据库无关，它是依据逻辑数据模型的，非范式化的物理数据模型对于应用代码是不可见的。这里只有两行代码涉及到持久化：ViewFactory::getView()命令从访问层得到数据。pos->markModified()方法为SumOfInvoice作上标记，以让它把自己写回数据库。

```
void Order::processInvoice ( OrderKeyType anOrder){
    // 从数据库得到数据，我们只指定主键，剩下的让访问层去做
    OrderInvoiceView * pInvoice =
        ( OrderInvoiceView*)ViewFactory::getView( anOrder );
    // 处理发货单项
    ItemIterator itemIter = pInvoice->iItems.begin();
    for(; itemIter != iItems.end(); itemIter ++){
        itemIter->iSumOfInvoice +=
        (itemIter->iQuantity *
        itemIter->iArticle.iArticlePrice);
    }

    // 视图改变了，作标记
    pInvoice->markModified();
}
```

程序段2：processInvoice的实现。这个例子演示了订单中订单项的遍历和所有项目iSumOfInvoice属性的求和。注意，我们遍历了逻辑数据模型中的两个间接层。为简便起见，我们舍去包围在Order::processInvoice外面的事务处理和一些明显的类型定义。

变种

- 许多应用大多是一些简单用例的集合，它们只需要单一级别的间接（就像实体和它依赖的实体）。这种情况，ConcretePhysicalViews封装了数据库访问代码并向应用提供足够的简洁接口，省掉查询代理和层次视图。不过这对那些可能涉及到上10个实体的复杂用例不太适合，例如，保险应用等。

- 一个更复杂的变种是允许获取历史数据，当你不仅仅对一个合同的当前状态感兴趣，而且对它在某一个指定时间的状态感兴趣时，需要这种变种，为浏览数据模型，你不得不为表示基于时间的浏览而增加条件和浏览边。保险公司经常需要这样。

已知应用

VAA，应用于德国保险公司的一个标准构架，结合时间浏览使用这个模式[VAA95]。相应的数据管理部件目前在构建中，Württembergische Versicherung[Würt96]开发了一个使用层次视图的数据管理器以及一个定义它们的工具。

许多sd&m的项目使用层次视图模式的简单变种(1:n views)，生成这些视图[Den91]。

相关模式

你可以使用一个查询代理来降低层次视图和底层物理数据库的耦合度。使用一个视图缓冲来避免对同一物理数据的重复数据库访问。

模式：物理视图

环境

你已经决定使用关系型数据库访问层。你使用层次视图作为应用核心的接口，并且你已决定，不将数据库的访问放在ConcreteViews。

问题

如何提供一个易用的访问物理数据库表的接口？

先决条件

- 简单化与性能：为达到一个良好的性能，你不得不使用非范式化、可控冗余或溢出表来优化物理表的布局。但是，对于这些技术如果没有限制地使用会打乱处理物理数据结构代码并使数据库访问变得复杂，尤其是溢出表将导致错综复杂的代码。除了这些复杂的优化，你还想有一个易用的接口和一些可维护的类。

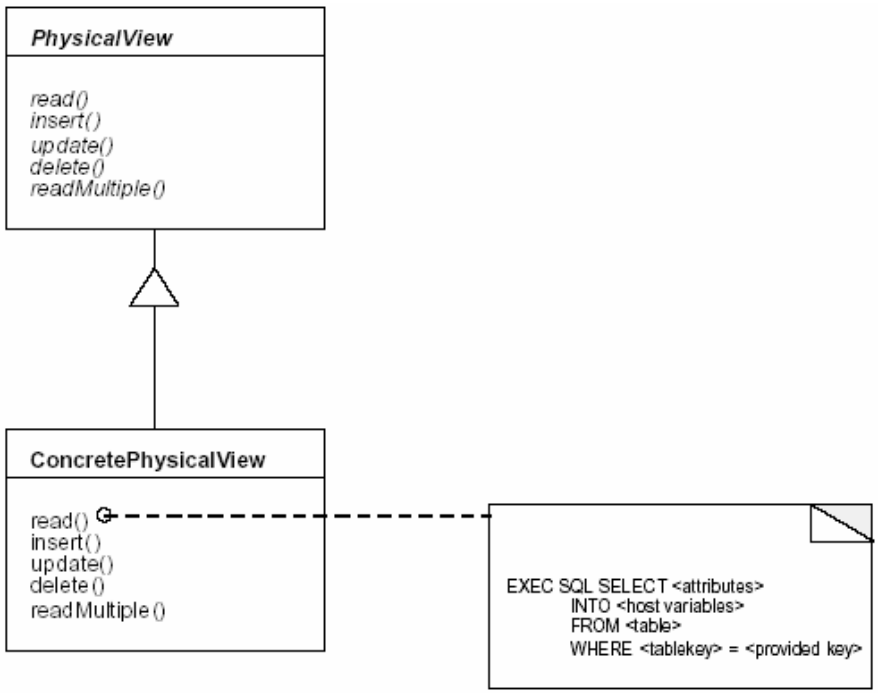
- 灵活性：大多数据库向你提供了使用静态还是动态SQL的选择，因为数据库预编译、优化静态SQL查询，这通常减少服务器复杂性以得到较好性能。一些数据库管理员只允许在服务器上使用静态SQL。另一方面，动态SQL更加灵活并且更适应数据库结构的变化，很容易在开发中使用。为满足高性能的需求，你也许想使用底层数据库API，访问层高层API应该不知道这些细节。

- 大量问题和开发成本：因为大型数据库可能有上百个表，你也许想有一个构建这些接口的自动过程，宏、生成器或模板都可以减轻这个工作。然而，一个通用的接口开发要花费更多的设计努力，因为它必须要考虑到各种情况。

解决方案

用一个ConcretePhysicalView封装每个表和视图，同样用这些类来封装溢出表。为提供一个统一的接口，从PhysicalViews派生出ConcretePhysicalViews，它使用类似PhysicalViews的记录结构来存储它们的实例数据，主要区别是封装一个物理表或视图，还是封装多个表、视图。

结构



实现

封装什么？每个ConcretePhysicalView应该封装一个简单的SQL语句和它对应的溢出表。因为层次视图指向多个ConcretePhysicalView，你可以选择是使用SQL来连接数据库的两个表，还是在访问层连接它们。一个比较好的入手点是为每个“根表”，例如客户表和物品表，定义一个ConcretePhysicalView，，另外，为每个“复合实体”定义一个ConcretePhysicalView，你已为它们定义了数据库视图，例如订单/订单项关系。如果你使用一个查询代理，你可以分析这些决策，寻求更多的候选实体。

封装只读视图：为保持物理视图尽可能简单，你应该考虑哪些ConcretePhysicalViews有权限更新它们已读取的数据。表示数据库视图的物理视图，以及大多数数据库都不支持写视图操作。因此，如果涉及到几张表，一个只读访问的物理视图要比读写访问来得简单。一个好的主意是完全让一个物理视图拥有对某个表的写权限。

编程工具：ConcretePhysicalViews是一般性的类，使用一个生成器或宏技术去实现它们，当然也可以考虑模板。

使用存储过程和其他API：大多数数据库提供存储过程以进行数据库服务器上的运算。因为物理视图是直接工作在数据库上的，因此可以使用存储过程或是其他任何访问数据库元组的API来实现。使用这个方案，你可以写出灵活的优化，例如溢出表，它是数据库代码形式的，而非某种主机语言加上嵌入SQL。但是，这将增加数据库服务器的负载，并且你要确定，所有的应用都要遵从这样的架构。

结论

- 简单性：物理视图隐藏了优化和数据库编程的复杂性。因为它没有其他责任，也很容易实现。尽管如此，附加的层增加了附加的类。如果你计划舍去查询代理，而直接硬联接到ConcreteView的话，你应该考虑清楚，是否增加一个层更简单或者是否ConcreteView本身应该访问数据库。后者导致较少的类，但是相应的，灵活性也降低了。

- 灵活性：因为物理视图封装了数据库代码，使用何种API去访问数据库是它们的选择。你可以将不同数据库访问API分成几组类。如果你想在运行时试验不同的数据库访问技术，你就可以使用桥(Bridge)[GOF95]来动态切换访问模式。

- 封装：物理视图使你无需影响上层而优化物理数据库结构。简化了调优过程并导致更佳的性能。大量的附加间接级在这里可以微不足道的。

- 大量问题：很容易定义一个生成器来构建一个粗略版本的ConcretePhysicalView。只要你不使用溢出表，你只需要将相应的SQL语句包装起来，更复杂的生成器可能需要处理溢出表等问题。

变种

如果你选择直接联接ConcreteViews和ConcretePhysicalViews，你可能将ConcretePhysicalViews作为ConcreteViews的方法。不过这个方法灵活性较低，因为你无法重复使用同一个ConcretePhysicalView。

你也可以使用物理视图来封装非SQL数据库和文件系统，例如ADABAS、IMS-DB、CODASYL和VSAM等。我们前面提到了，你可以使用这些变种来在遗留系统上构建关系型的应用。

示例解决

图7显示了两个物理视图的DAG定义，这是我们的发货单例子所需的。它们对应到物理数据库结构而且也解析溢出表（见图1）。为了简化OrderPhysicalView，应该只对Order和OrderItem数据赋以更新访问权限，由其他的物理视图改变Ariticle表，

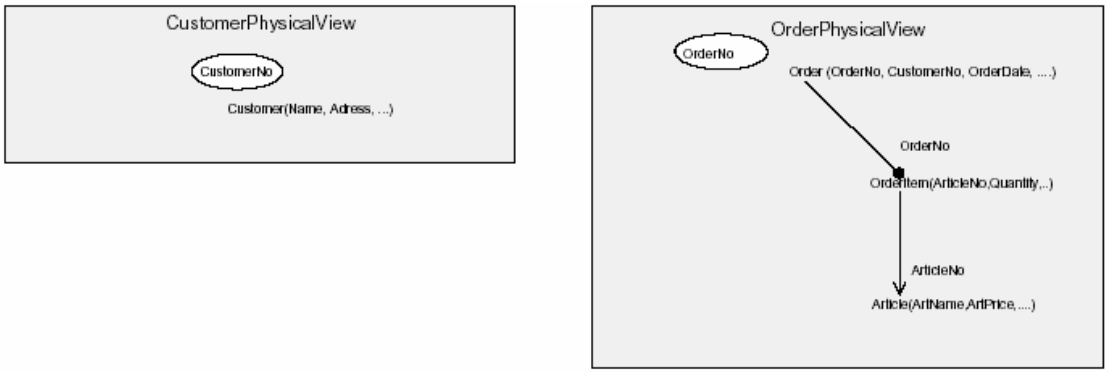


图7：两个ConcretePhysicalViews的DAG定义。注意OrderPhysicalView封装Order表和它的溢出表OrderItemOverflow，而CustomerPhysicalView只封装了一个Customer表，参见图1的物理表结构。

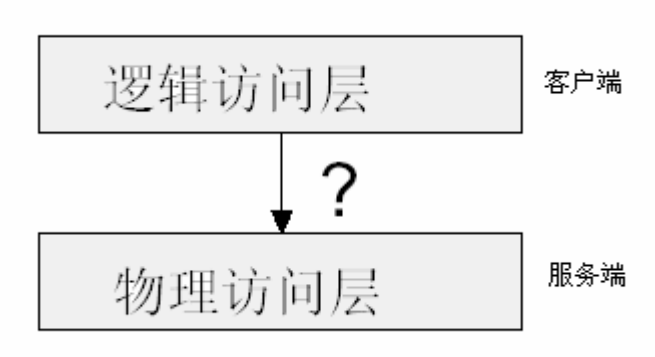
相关模式

对优化更深入的讨论，参照[Kel+96b]：溢出表模式的详细描述，如何部分合并表。受控的冗余包含了关于何时赋写权限的讨论。窄视图给出物理视图如以选择数据的提示。

模式：查询代理

示例

考虑前面发货单的例子，OrderInvoiceView描述了逻辑数据结构，而OrderPhysicalView和CustomerPhysicalView描述了相应的物理表。



环境

你已经决定使用关系数据库访问层了，使用层次视图作为应用核心的接口，并使用物理视图来封装数据库访问。

问题

如何连接层次视图和物理视图以进行读取和写入？

先决条件

- 成本与灵活性：最节省成本的方式是通过函数调用直接硬连接：一个ConcreteView知道它必须调用哪个PhysicalViews，你能够使用约定的表描述来生成相应的调用，这在所有的层都非常稳定的情况下会工作得很好，但是，如果有一层不是很稳定，你就需要使用某种方式来降低耦合。在访问层中，我们有比较稳定的层次视图，位于不太稳定的物理视图之上。如果系统足够小，你可以使用一个程序生成器来耦合这两层。不过，如果系统存在的时间将有好几年，这种方式会因为编译和软件发布而产生昂贵的成本，想象一下你将不得不在每当物理数据模型发生变化时，将几兆的数据库访问软件发布到数千个客户端。

- 重用性：虽然物理视图可以迅速改变，它反映数据库的物理结构。因此有可能是若干个应用程序使用相同的物理视图却拥有不同的层次视图。为每个独立的应用编写一个独立的耦合机制使你从重用物理视图中得到的好处无处体现。

- 复杂性：因为硬编码的解决方法不够灵活，你需要一个更复杂的方法。然而，附加的复杂性使你的系统成本增加且难以二次维护。

解决方案

使用一个代理[Bus+96]来联接两个层，层次视图形成查询代理的客户端，物理视图作为代理的服务端。使用DAG来描述服务，并且使用一个树匹配算法来寻找最优匹配。让查询代理装配物理视图并将结果集放在查询结果集包容器中。

结构

代理是降低耦合的一种标准技术，以它的标准结构应用于数据库访问层框架：

- 查询代理和一个标准代理最重要的区别是它通常有多个服务端来处理一个请求，到服务端的映射不是一对一而是一对多的关系。

- 代理通常使用符号名称来标识服务。但我们这里在服务请求和服务端之间是一种1:n的关系，这不大适合。因此，查询代理使用语义描述（DAGs）来表述请求视图。看看图9，左边表示OrderInvoiceView的请求，右边表示相应的服务。为组装ConcretePhysicalViews，查询代理匹配关键字，标记在白色椭圆中。

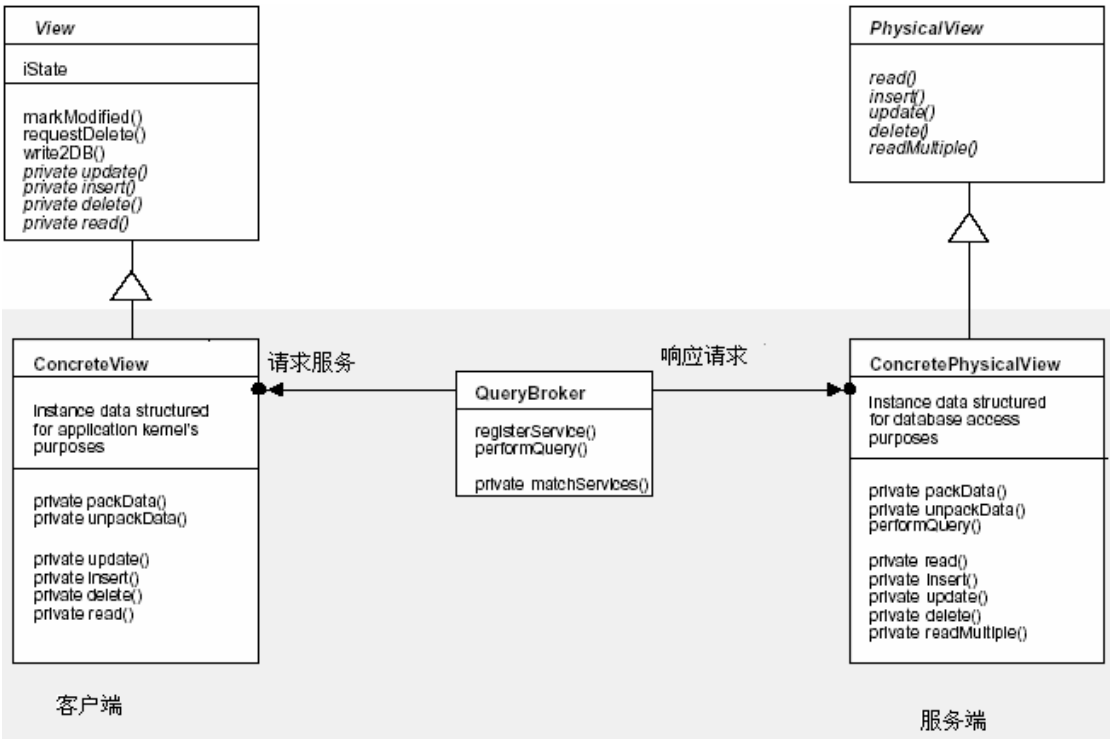


图8：查询代理是一种应用于数据库访问层框架的代理。

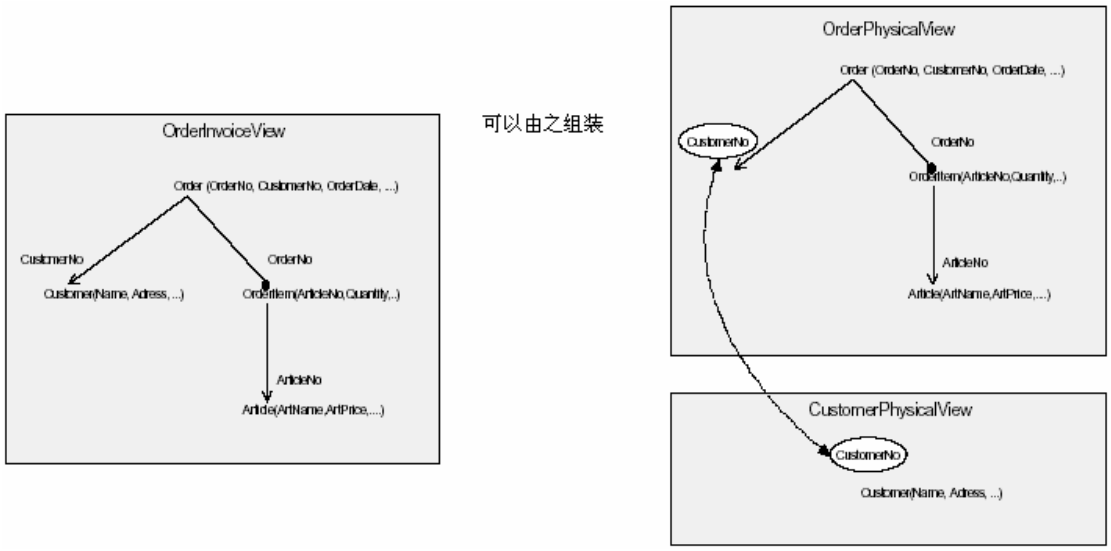


图9：树匹配解析查询代理收到的请求。左边显示了我们订单处理系统的OrderInvoiceView，右边显示了相应的两个物理视图。如果查询代理得到左边表示的请求，那么右边的物理视图将是满足请求的最优方式。

动态行为

在下面的场景中，一个应用核心对象创建一个OrderInvoiceView，产生一个数据库调用read()，查询代理处理read()，并且通过一个matchServices()方法依据已有的服务匹配一个视图描述。QueryBroker将这个请求转发到两个不同的ConcretePhysicalViews：OrderPhysicalView和CustomerPhyscialView，这两个read()从数据库读取数据并将数据打包到结果集包容器中发布。代理必须合并结果集包容器，向OrderInvoiceView发布一个结果。OrderInvoiceView再将数据解包到它的实例变量中。

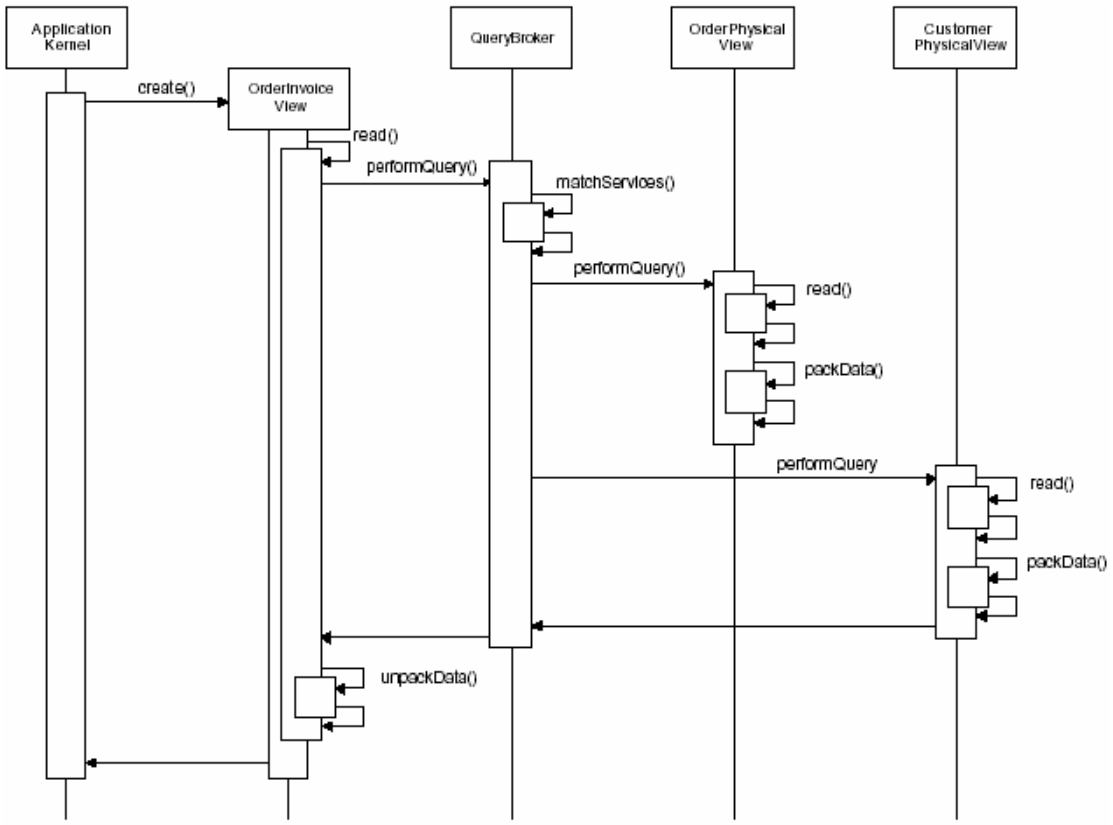


图10：由查询代理获取数据

实现

- 服务端注册：所有的ConcretePhyscialViews必须在第一次数据库访问前注册到QueryBroker中，在系统初始化时要特别注意这一点。你可以使用一个运行时字典、其他一些形式的注册表或者是语言特定的初始化技术。

- 转换数据类型的职责：原始数据库类型和应用数据类型的互相转换有两种选择。例如，你不得不将一个CHAR(20)类型转换到一个OrderKeyType类型，反之亦然。你可以转换ConcreteViews，也可以转换PhysicalViews。如果有一个运行时字典，它包含逻辑数据模型和哪些属性转换到哪些应用数据类型的信息，就可以同时支持这两种选择。

- 树匹配：匹配DAG类似于编译器的代码生成，它必须为程序寻找一个理想的汇编码，所以，你可以使用类似的算法[Aho+86,9.2章]。代理甚至可以为一个请求寻找若干具有不同速度的查询计划，匹配算法必须处理任意多种情况，以得到一个最快的方案。对编译器优化也有类似问题，为代码生成的目的，处理任意语法。

- 查询描述和结果集包容器：你必须找到一个理想的DAG表示法，用以描述查询。一方面，视图应该能够很容易指定它们的请求和服务，另一方面这个表示法应该顺应匹配算法的需要，一个易于解析的文本表示是一个很好的选择。

结论

- 灵活性：查询代理完全降低了层次视图和物理视图的耦合度，在运行时，新的物理视图可能被注册，而且和层次视图的关联也会被改变。

- 复杂度：树结构的结果集包容器、请求描述、树匹配算法使查询代理很难设计，但是代理封装性很高，限制了单一子系统的复杂性。

- 重用性：因为查询代理独立于它联接的视图，你可以将它作为框架的一部分，这比仅仅重用物理视图或生成器更好。

- 成本：查询代理的复杂度使之实现的代价很高，一个运行时字典更增加了成本。不过作为补偿，它实现成一个可重用的框架并应用于多个应用程序中。硬连接耦合的构建代价较低但是使优化更昂贵，并且当你想将软件发布到数千个客户端时，简直是个恶梦。

变种

如果你只使用动态SQL，查询代理可以组装SQL语句而无需使用物理视图，这对于干净的数据库模型工作起来不错，但是对于溢出表的处理就很难。你可以在开发的早期使用这种方式，接着，增加越来越多的使用静态SQL或其他API的物理视图服务器，代理向应用核心隐藏了这些变化，它总是负责在它的注册表中寻找最快的服务。

已知应用

查询代理汇集了多种最好的实际应用：

VAA数据管理器[VAA95]根据逻辑数据模型定义视图，它使用一个自动生成的硬连接来耦合两层。我们在HYPO-Bank中的经验告诉我们无论在哪儿都尽可能使用动态描述。在sd&m的两个项目使用了这种方式其他重要的部分。

sd&m的LSM项目使用动态SQL，移植到静态SQL并最终形成一个元组接口。这个想法来自工作于一个缓慢的数据库服务器上的实际经验，这个项目完全使用一个运行时数据字典来桥接动态查询和预编译查询。

德国警局的Fall/OK项目使用树匹配，这个软件通过在一个大型数据模型上的示例处理查询，数据模型的改变非常频繁。

CORBA持久对象服务[Ses96]也使用了一个代理，应用核心对象将它们的实例数据写入流，然后一个代理（持久对象管理器）将流转发到某个持久对象服务（数据库或其他什么）。持久对象服务可以是任何数据库，对象无需知道它。这也是一种简单的请求和服务端之间一对一的情况。

相关模式

[Bus+96]包含了一个关于代理的更广泛的讨论。

Brown和Whitenack描述了一个代理模式[Bro+96]的基本类，注意，查询代理是更一般的情况。

参考文献

[Aho+86] **Alfred V. Aho, Ravi Sethi, Jeffrey D. Ullman**: *Compilers: Principles, Techniques, and Tools*, Addison-Wesley 1986.

[Bro+96] **Kyle Brown, Bruce G. Whitenack**: *Crossing Chasms, A Pattern Language for Object-RDBMS Integration*, White Paper, Knowledge Systems Corp. 1995. A shortened is contained in: **John M. Vlissides, James O. Coplien, and Norman L. Kerth (Eds.):** *Pattern Languages of Program Design 2*, Addison-Wesley 1996.

[Bus+96] **Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal:** *Pattern Oriented Software Architecture, A System of Patterns*, Wiley 1996.

[Col+96] **Jens Coldewey, Wolfgang Keller:** *Objektorientierte Datenintegration – ein Migrationsweg zur Objekttechnologie*, Objektspektrum Juli/August 1996, pp. 20-28.

[Col97] **Jens Coldewey:** *A Database Access Layer for ODBMS*, In **Akmal Chaudri, Mary Loomis (Eds.)**, *Experiences with ODBMS* [working title], Prentice Hall 1997 (to appear).

[Dat94] **Chris J. Date**: *An Introduction to Database Systems,* Sixth Edition; Addison-Wesley 1994

[Den91] **Ernst Denert**: *Software-Engineering*, Springer Verlag 1991.

[GOF95] **Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides**: *Design Patterns, Elements of Reusable Object-oriented Software*, Addison-Wesley 1995.

[Gra+93] **Jim Gray, Andreas Reuter**: *Transaction Processing, Concepts and Techniques,* Morgan Kaufmann Publishers 1993.

[Kel91] **Wolfgang Keller:** *Automated Generation of Code using Backtracking Parsers for Attribute Grammars*, ACM Sigplan Notices, Vol. 26(2), 1991.

[Kel+96a] **Wolfgang Keller, Christian Mitterbauer, Klaus Wagner**: *Objektorientierte Datenintegration über mehrere Technologiegenerationen*, Proceedings ONLINE, Kongress VI, Hamburg 1996.

[Kel+96b] **Wolfgang Keller, Jens Coldewey**: *Relational Database Access Layers*: A Pattern Language, Proceedings PLoP '96, Allerton Park 1996.

[Lan96] **Manfred Lange:** *Abstract Constructor,* Preliminary Conference Proceedings EuroPLoP, First European Conference on Pattern Languages of Programming, Irrsee, Germany, 1996.

[Mar95] **Robert C. Martin:** *Designing Object-Oriented Applications Using the Booch Method*, Prentice-Hall International, London, 1996

[ODMG96] **Rick G. G. Cattell (Ed.) et. al.:** *Object Database Standard: ODMG-93 -Release 1.2* Morgan Kaufmann Publishers, San Mateo, California, 1996.

[Rum+91] **James Rumbaugh, Michael Blaha, William Premerlani, Frederick Eddy, William Lorensen:** *Object-Oriented Modelling and Design*, Prentice Hall, 1991.

[Sch96] **Johannes Schlattmann:** *Die Anwendungsarchitektur der Versicherungswirtschaft,* Historienkonzept, Entwurf, GDV Bonn, 1996.

[Ses96] **Roger Sessions:** *Object Persistence, Beyond Object Oriented Databases*, Prentice Hall 1996

[VAA95] **GDV:** *VAA - Die Versicherungs-Anwendungs-Architektur*, 1. Auflage, GDV, Bonn 1995

[Würt96] **Württembergische Versicherung:** *Projekt Datenmanager*, private communications 1995 - 1996.

[Wit96] **Andreas Wittkowski**; *Datenbankdesign & Performance*, slide presentation, sd&m Internal Lecture Series, 1996.

《人件》— 老板，别把开发人员当成牲口

新闻

《人件》提供预订>>

相关文章

《人件》实践之：SAS公司
关注程序员自己的文化——专访Tom DeMarco
别把开发人员当成牲口
Brooks在《人月神话》中的评论
Alan Cooper的评论
办公室空间，下一场革命
《人件》在计算机行业的实践（待续）
人的问题：关于《人件》
Edward Yourdon的评论
开发人员是人吗？

各版本封面

英文，日文，德文

首页 购买 留言板

以用途为中心的 Web 应用工程

Larry Constantine、Lucy Lockwood 著，金哲凡 译

查看评论

摘要

本文展现了一种轻量的以用途为中心的设计方法，实践证明该方法在高可用性的基于 Web 的应用设计中特别有效。该方法与传统的面向对象方法完全兼容，也与新出现的敏捷设计技术，例如极限编程完全兼容，它使用了快速的、基于卡片的技术来产生一系列简化的模型，包括用户角色、任务、用户界面内容。整个流程试图解决系统在下述两个方面的矛盾：一方面需要快速设计和增量开发，另一方面需要与适合整套用户任务的用户界面集成。具体的办法是，创建一个导航构架和一个可视的交互设计方案，这个方案的基础是快速、易于理解的任务建模。本文将以一个实际的为授课教师服务的 Web 应用设计为例子来说明这个过程。

关键字

可用性，Web 应用，以用途为中心的设计，敏捷方法，轻量方法，极限编程，迭代开发

（想进一步了解用途为中心的设计，请访问 http://www.forUse.com）

概述－旧瓶装新酒

Web 是新规则的代表，这种说法真令人窒息，尽管如此，职业的开发人员发现，在 Internet 时代之前学到的软件开发知识仍然是有效的。他们认识到，Web 网页是用户界面，HTML 编写就是一种编程，使用浏览器的软件系统能从基本的软件工程规则中获益。

然而，Web 开发在许多方面有其特殊性。尽管在本质上，任何可通过软件实现的功能都可以通过 Web 发布而在浏览器使用，但实际上几乎所有方面都与以往有所不同。由于缺乏标准和惯例，Web 的开发和使用显得更加混乱而复杂。不同的浏览器、平台和连接的不一致性这几种因素加在一起，降低了我们对用户的实际所见和使用感受的控制能力。在一个工程里，多达 70%的工作是为了让理应是标准的特性能够在某种浏览器（如 NetScape）和某个平台上（如 Apple Macintosh）正常工作，而以客户需求为本的工作加起来不到 10%。Web 程序设计风格的多样性使得局面更加混乱，一些被广泛接受的使用习惯也常常会被省略掉，比如将主页连接放在最左边的列，链接以蓝底文本的风格显示等。

三种特性的考虑应该放在 Web 开发的前列：一、不断变快的开发过程；二、基本没有规则和先例的“荒地”应用大行其道；三、用户使用感受至关重要。几乎每个工程都面对着缩短了的产品发布周期和不断增加的、及时发布的压力，但是 Web 重写了快速开发和配置的定义。“Web-Time”已经成为了不顾一切前进和实际不可能的时间期限的同意语。

系统化的注重纪律的实践经验给了我们一些希望，但是对传统方法的盲从会导致许多问题。传统方法基于大量的文档和频繁的发布，强调对已完成的实践和已存在系统的分析，它们可能不能很好地适应 Web 时代新奇的应用开发的需要。而且，几乎毫无例外，在统一过程等“重量级”过程，和它们轻量级的竞争者如极限编程中，用户界面设计和可用性都没有扮演重要的角色。

本文描述了一种用于 Web 应用的、柔性的、模型驱动的方法，其成功之处在于对用户界面设计和可用性的关注。它简明的模型驱动技术很适合新奇的应用，而且能方便地集成到各种在压缩了的时间表上进行的“轻量”或“敏捷”的开发过程中去。

以使用为目的的 Web 应用

可用性和用户的使用感受是判断 Web 应用是否成功的关键性要素。如果顾客不能找到他们需要的东西，他们就无法购买；如果关键信息被埋葬在许多不重要的信息里，用户就会作出不明智的商务决定。设计得不好的用户界面将增加用户的错误，而这可能导致巨大的代价。例如，如果在输入信用卡帐单信息时出了错，可能会需要昂贵的人工善后工作，或者使应有的交易不能成交。据 Web 可用性专家 Jakob Nielson 估计 ，可用性问题在 Web 交易中导致的损失达数十亿美元。

技术支持和客户支持在业务各个方面的开销以火箭速度增长。每当网站或应用程序的可用性问题，引发客户帮助坐席或客户服务部门的电话或 email 时，一次便宜的 Web 对话就演变成了一次昂贵支持服务。而在公司内部网络级别的应用中，难以使用的程序要么索性被扔到一边，要么需要额外的培训。

本文的重点是基于 Web 的应用，这些系统的功能放在 Web 上，通过作为瘦客户的浏览器实例来使用。但是，在基于 Web 的应用和那些在 Internet 上可以使用但仅仅是为了娱乐、随意访问或建立公司标志的站点之间划出一条清晰的界限是困难的。它们相同的思路是在 Web 上发布有用的服务和功能。

在可用性之外，设计的审美水准对用户的感受也很重要。不幸的是，图形设计和审美考虑往往在前期完成，并且影响到整个过程，最终牺牲了可用性。比如，基于客户 CEO 提供的“外观需求大纲”，某著名的图形设计室为核心设计小组设计了新颖的 Web 站点。尽管设计理念已经对特定的客户做了广泛的宣传，并做了很多市场引导的工作，实践证明这种设计在可用性方面是灾难性的，市场的失败与美观设计没有什么关系。通过努力在不修改基本的图形设计的情况下改进可用性，网站生存了下来，但是最后证明只比原来略强一些，仍然不能完全满足客户的需要。最后，公司以用途为中心，对网站内部进行了全面的重新设计。

用途为中心的设计

以用途为中心的设计方法是一种系统化的过程，它使用抽象模型来进行最小、最简单系统的设计，这些系统直接、完全地支持用户需要完成的所有任务。以用途为中心的方法在 1990 年代提出，它是一种经实践证明了的颇具工业实力的设计方法，这种方法被用于各种系统的设计，包括工业自动化系统、消费电子产品以及银行和保险方面的应用。由于它是一种由简单模型驱动的流线型过程，所以有很好的可扩展性，它可以用于只需要少量人月的小工程，也可以用于需要 5 个设计师、19 个开发人员、23 个月的工程如 Step7lite，后者是 Siemens AG 的一个复杂的集成开发环境。在 Web 上，这种技术被全球各地的开发小组成功地使用着，涉及的应用类型广泛，包括电子商务、客户支持、教育和医药信息系统。

用户为中心，还是用途为中心？

以用途为中心方法是作为用户为中心的方法的替代物而发展起来的。表 1 总结了两者一些显著差异。以用户为中心的设计技术松散地连接了人的要素，它深层的哲学是：理解用户并且把他们结合到设计当中。它主要依赖于以下三种技术：通过用户研究确定用户想要什么，在连续的用户界面迭代设计中使用快速的文档原型来得到用户反馈，以及通过用户对工作原型或系统进行的可用性测试来发现可用性方面的问题。尽管这些技术是有用的，但它们都不能替代好的设计。用户研究太容易混淆用户想要的东西和用户确实需要的东西。快速迭代原型对于深思熟虑的系统化设计来说，是一种糟糕的替代品。可用性测试则是一种相对低效的发现问题的方法，而这些问题通常可以通过合理的设计来避免。

Table 1 - Comparison of User-Centered and Usage-Centered Design

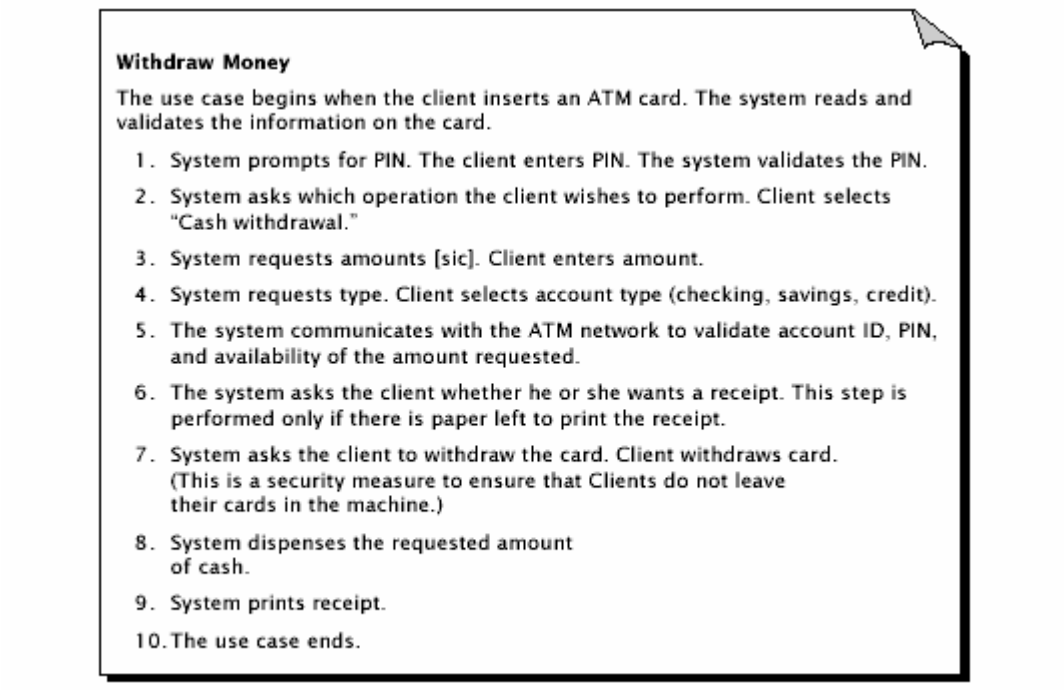
User-Centered Design	Usage-Centered Design
Focus is on users: user experience and user satisfaction	Focus is on usage: improved tools supporting task accomplishment
Driven by user input	Driven by models and modeling
Substantial user involvement ● User studies ● Participatory design ● User feedback ● User testing	Selective user involvement ● Explorative modeling ● Model validation ● Usability inspections
Design by iterative prototyping	Design by modeling
Highly varied, informal, or unspecified processes	Systematic, fully specified process
Design by trial-and-error, evolution	Design by engineering

驱动模型

以用途为中心的设计由三种简单而紧密相关的模型驱动：角色模型，任务模型和内容模型。角色模型捕获用户在系统里扮演的角色的显著特征。任务模型代表了用户相对于系统所需要完成工作的内在结构。内容模型代表了用户界面为了支持任务而需要的内容和组织。

用户角色代表了一种用户可能与系统或 Web 站点发生的关系。这种关系可能包括许多方面：交互的目的和频率，交换信息的方向和信息量，用户作为某种角色对系统的态度等。

任务模型由一套任务案例和描述这些案例之间关系的图组成。任务案例是一种用例。传统的用例是系统的模型，即“一个系统或子系统或类在与外部用户交互时可能产生的动作序列，包括变化的和错误的序列”。通常认为，用例表达了系统和与其交互的角色之间的固定动作和反应。图 1 是一个出版了的传统用例的例子，它明显忽略了用户自己完成的最重要的步骤：把钱取走。



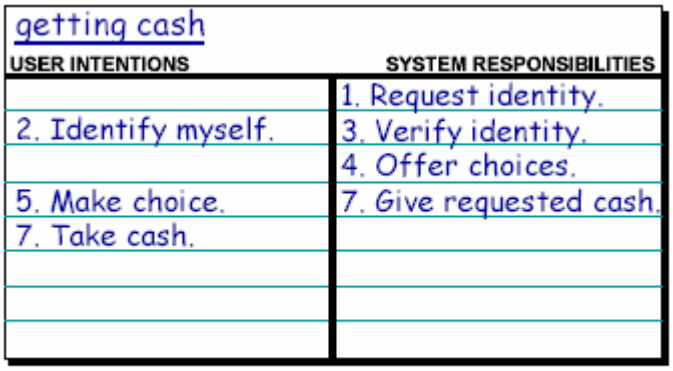
Withdraw Money

The use case begins when the client inserts an ATM card. The system reads and validates the information on the card.

1. System prompts for PIN. The client enters PIN. The system validates the PIN.
2. System asks which operation the client wishes to perform. Client selects "Cash withdrawal."
3. System requests amounts [sic]. Client enters amount.
4. System requests type. Client selects account type (checking, savings, credit).
5. The system communicates with the ATM network to validate account ID, PIN, and availability of the amount requested.
6. The system asks the client whether he or she wants a receipt. This step is performed only if there is paper left to print the receipt.
7. System asks the client to withdraw the card. Client withdraws card. (This is a security measure to ensure that Clients do not leave their cards in the machine.)
8. System dispenses the requested amount of cash.
9. System prints receipt.
10. The use case ends.

Figure 1 – Conventional use case for getting cash from an ATM [Kruchten, 1999]

图 2 是一个简化的任务案例，用于以用途为中心的敏捷设计，它更加短小精悍，更精确地描述了真正与用户有关的任务。任务案例也称为基本用例，是抽象、简单、与技术无关的基本模型，它对用户界面没有任何内定的假设。



getting cash

USER INTENTIONS	SYSTEM RESPONSIBILITIES
	1. Request identity.
2. Identify myself.	3. Verify identity.
	4. Offer choices.
5. Make choice.	7. Give requested cash.
7. Take cash.	

Figure 2 – Abstract use case (task case) in essential form.

基本用例的优点得到了认识，人们普遍认为这种风格对用户界面设计和需求建模都是最好的。

第三种模型－内容模型，有时也被称为抽象原型，因为它是一种抽象的表达－独立于用户界面实现后实际的外观和行为，它描述了用户界面的内容以及内容如何组织成上下文，用户遵循上下文与系统进行交互。抽象原型可以有许多种形式，从网站或应用的可视项目或可访问内容的简单清单，到基于一套规范的组件的高度结构化的模型。不论是那种形式，抽象原型是沟通任务模型和实际的具体原型的桥梁，这些具体的模型包括传统的文本原型和设计草图。通过抓住用户的视角，同时在不同的实现方式上保持开放，抽象方法鼓励创新的外观和交互设计。

流程概述

图 3 显示了以用途为中心的设计过程的逻辑视图。系统角色－系统需要交互的其他软件和硬件系统与作为用户的人类角色分开。三个依次展开的核心模型与其他模型之间有一些连接，其他模型包括：词汇表形式的域模型以及其他更加细化的数据模型，包括包含内部业务逻辑和约束的业务规则模型，描述工作环境显著特征和操作上下文的操作模型。

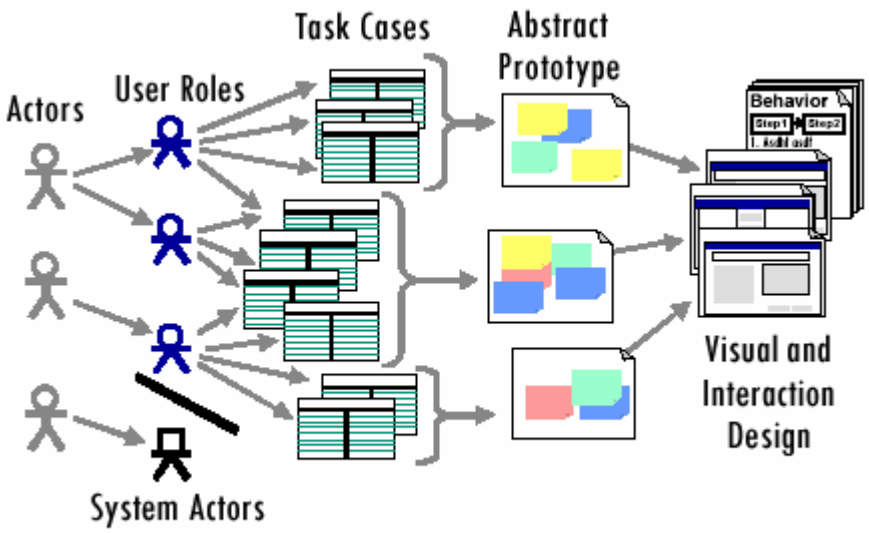


Figure 3 - Logical process of usage-centered design.

概念上说，可以用一种简单直接的关系把最终设计连接到支持用户角色的任务案例。每一个 Web 页、表格和交互上下文对应一个抽象原型，后者支持一簇相互关联的任务案例。实际的按钮、连接、表格、显示以及其他属性都直接从抽象组件中派生出来，后者在所支持的任务案例中实现特定的步骤。同理，这些任务案例支持用户以一定的角色进行与站点或应用相关的活动。

用途为中心的 Web 设计

通过连续的 Web 工程实践，一种轻量的过程产生了，它吸纳了一些建模和设计的窍门。敏捷的用途为中心设计包括了一种快速建模技术，它使用普通的索引卡片来支持头脑风暴、排序和归类，这些办法在极限编程里很流行。在实践中，通过与最终用户、应用领域专家和客户合作建模，搜集、建模和验证用户需求的时间还可以缩短。

敏捷的用途为中心设计流程的要点大致是：

预备步骤

1、 实质目的和猜测：阐明业务和用户的目的，然后进行想象，排除对属性、工具、内容和能力的猜测；

2、 探索性建模：发现问题，找到含糊不清之处以及风险和不确定性所在的区域。

首次迭代：

3、 角色建模：对所有用户角色列表，分出优先级，选择初始的目标子集；

4、 任务建模：对所有任务列表，分出优先级，选择初始的目标子集；

5、 任务归类：根据相关性和全局导航结构的草稿对所有任务分组；

6、 设计：草拟可视和交互方案，对外观设计进行检查和修改；

7、 建立抽象原型：为交互上下文创建内容模型，以支持所选的任务子集；

8、 设计：为所选的交互上下文自己创建用户界面设计细节；

9、 构建：程序实现所设计的部分用户界面；

连续迭代（同样）

3a、检查角色模型，如果需要的话进行修改，选择下一个子集；

4a、检查任务模型，如果需要的话进行修改，选择下一个子集；

5a、检查导航结构，如果需要的话进行修改；

6a、检查可视和交互方案，如果需要的话进行修改；

7a、为交互上下文创建内容模型，以支持所选的下一个任务子集；

8a、为所选交互上下文设计用户界面；

9a、构建新设计的部分。

实质目的和猜测：

Web 工程常常被认为有以下特点：目的含糊并且有不切实际的雄心。在前期阐明目的并进行构想能带来巨大的好处。我们建议开动这样的过程：由管理者、用户、开发者和其他涉众参加，从业务眼光和外部用户的观点，来确定应用目的。业务目的和外部目的可通过头脑风暴发掘，并记录到卡片上，然后卡片可以分类并建立优先级。以同样的思路，我们鼓励参与者分享他们关于系统的属性、功能、工具、内容和能力的想象，但是我们清晰地标识这些想法为“猜测”并把它们放在一边，把它们作为可商榷的想法而不是真正的需求。

探索性建模

设计工作从探索性建模过程开始，其目的是明确用户需求的问题，以及不明确、二义性的区域。这是通过作出角色和任务模型的草稿完成的，精细模型将在以后创建。这样能很快发现遗漏了的必需的信息。

用户界面架构

许多敏捷过程放弃了先行的设计，而使用一种更加先进的方法：先构建一个范围较小的系统，然后通过连续的迭代来逐步求精。不幸的是，当开始规模很小的代码随着需求逐步扩大规模时，原来的构架和代码内部的组织常常变得不能充分支持、或不再适合不断出现的需求。重构意味着对现存的代码进行重新组织和重写来适应不断发展的需求，用迭代扩展的办法开发复杂的软件系统，重构是成功的关键。

用户界面则是另一回事。对用户界面基本结构的事后求精是不可接受的，因为这样会改变系统的面貌，而用户已经学会和掌握了以前的版本。稍微调整一下外观的位置或形式都会给用户带来麻烦。反之对软件内部组件的重构不必影响用户，而重新设计用户界面结构具有不可避免的破坏性。为了拥有高可用性的用户界面，其全局组织、导航和外观感受的设计都必须适合要完成的全部任务。

然而，并不需要先行设计用户界面的所有方面。为了避免根本的重构或在设计进程中出现不一致，需要预先生成一个精心构思的站点应用的全局设计图。为了保证基本的外观风格和行为方式能有效地贯穿整个应用，需要制定一个概括性的设计方案。

导航结构定义了用户界面宏观上如何组织成交互上下文和组，它们如何显示给用户，以及用户如何在其中“航行”。导航结构至少要定义所有的交互上下文及其内部连接，如果能包含更多内容则更好。比如导航结构可以说明用户界面的一部分如何组织到标签笔记本里去，后者可以通过核心分发对话框上的一系列命令按钮来访问。

全局用户界面设计的另一个至关重要的方面是外观和交互方案，即一整套的抽象风格指南，它简要地描述了对在整个设计里都要用到的、反复出现的基本可视元素以及通用的布局和模板，这些要素应用于各种交互上下文。例如，外观和交互方案可能规定一个基本的布局网格，一个修改过的树形浏览器放在最左边作为主导航器，一套可视的控件放在顶端作为次要的导航器。在更加详细的层面，它可能规定一种独特的颜色来标志可编辑域，后者可通过鼠标双击来开始编辑。它可能进一步指明特定的控件集合如何放置在滑出工具条上，后者在鼠标拖过时自动打开，用完后自动关闭。

毫无疑问，良好的导航结构和外观与交互方案必须建立在完整的任务模型基础上，后者包括了所有确定的任务，而不仅仅是那些只在最初或早期迭代中碰到的。

基于卡片的建模

最初的用户角色模型，是一个用户相关于应用或站点可扮演的角色的简单清单。生成这样一个清单的最简单的办法之一是头脑风暴，并把结果记录到卡片上。当大家都认同了初始的清单时，角色已经用简洁的语言直接记录在了标志卡片上，同时记录使用的词语最适合设计问题的描述：角色的上下文，角色交互的特征模式，为了支持角色需要的特殊条件。完成了的卡片被排序，次序体现了它们在设计中的重要性。

在以重要性为序对角色进行复审之后，我们通过头脑风暴生成一个任务案例的清单，同样把它们记录到卡片上。当我们复审和求精这个清单时，通过压缩、合并、删除和添加任务，我们将其命名标准化，以表达用户的基本目的，比如“找到某个产品的替代品”或“为课程计划添加评估”。初始的任务案例建立起来之后，卡片按照需要的频率和全局重要性进行排序。以这两种优先级为根据，我们将它们分成三堆：需要的（先做），想要的（如果有时间的话做），延迟的（下次做）。

现在，我们开始补充细节，但这并不是对所有任务案例都需要。我们跳过一些案例，它们的交互看上去十分明显或是例行公事式的。对关键、复杂、不清楚或有趣的任务案例，我们直接把它们的交互描述写到卡片上。用小卡片工作是一种很好的建模习惯。如果某个过程的描述在卡片上写不下，那就说明要么描述不够抽象简明，要么卡片实际上覆盖了多个需要分别标志的任务。

使用在创建、精华任务案例的过程中积累起来的知识，我们将卡片按其相互关系的强弱分类，用户通过某一角色可以一起执行的一些任务尤其要归在一起。每一类都代表一个应用必须同时提供的功能集合——在同一个页面上、同一个浏览窗口里或至少在紧密关联的交互上下文里。

每一类的任务都可以成为部分用户界面设计的指南。我们主张建立抽象原型，但是潜在的压力可能直接转移到起草实际的纸件原型的工作中去。然后，纸件原型加上一份各种元素行为方式的说明就成为了评估可用性问题和确定问题区域的依据。为了方便和速度，我们主张“协作的可用性检查”，这种办法使用户、客户、设计者和开发者共同参与，另外启发式检查和其他技术也很有效。

实践

我们在一个基于浏览器的教室信息管理系统的设计中使用了本文的方法，实践证明在 Web 时代的敏捷过程中，这些方法特别合适。这个应用是一个复杂的性能支撑系统，它提供钥匙管理、计划管理、教学任务管理等功能，简化和方便了教师们的工作，而它的运行环境紧密地集成在已有的管理系统中，这里充满了各种软件和基于 Web 的教学资源。这个应用使用基于 Web 的技术，客户端的用户界面以 HTML、XML、和 Java JFC/Swing 实现。

风险因素很多，包括一个新成立公司的一支新的、能力未经证明的开发团队，一个外部的咨询团队负责用户界面设计，在很多领域需要技术突破，需求含混不清而且范围频繁改变，另外早期发布、频繁的演示和不可预见的变更也给管理带来了许多困扰。

设计小组与一个博士水准的培训团队协同工作，后者有着丰富的教室授课经验。这个培训团队同时担当了领域专家和最终用户的角色。

最初的用户角色和任务案例清单都是与培训团队协作开发的。总共定义了 14 个用户角色，并设定了优先级。最初的 142 个任务案例包括了 35 个与课程计划相关的案例，他们是系统的关键功能。过程描述－用例的定义主体，仅有少数几个“有趣”的任务案例写了。通过对任务案例分类，建立了包含 40 个交互上下文的导航图草稿。（图 4）

需求中冗余、无规律和二义性的部分，以及设计的问题通过频繁的会议和不断请教培训团队马上被解决，我们称这种模式为 JITR：Just-In-Time-Requirements。两位设计人员中的一位关注需求和范围的问题，另一位则开始初始的设计。导航结构的草稿、导航图和外观/交互方案先由设计小组修改，然后由培训团队检查和修改。

尽管不是百分之一百符合计划，设计过程以 2 到 2.5 个星期的周期进行。当全时的设计工作开始时，初步的导航结构和外观/交互方案已经在第一个周期里完成了。第一版的完整的外观/交互方案在下一个周期完成，之后进行可用性的检查，并在第二次迭代里进行一次主要的设计修正。第三次迭代继续扩展和精化设计，其过程覆盖了工程余下的以用途为中心的设计部分，共 17 周。编码工作遵循一个敏捷、轻量的过程，与外观/交互设计并行进行。

一个可接受的设计必须能让普通教师经少量培训、或不需培训就能马上使用系统。它必须有足够的柔性来容纳各种各样的工作习惯，并且使用起来有效率，因为老师们通常每天只有 15 分钟的时间来安排一天的课程。确定下来的技术上的目标是以可用性的标准为基础的，这些标准包括：

- 最小化窗口管理的开销
- 最小化形式化的动作
- 允许用户简单而直接地在上下文间切换
- 最大化可用的屏幕空间资源，用于文档和数据的显示

对完整的设计的讨论超出了本文的范围－基于这个工作的一系列的对设计的研究已经在 Web 上出版了。第一次设计迭代的结果清楚地表明，在理解用户角色和任务的基础上，早期尽早考虑全局的导航结构和外观/交互方案是至关重要的，这无论在传统的软件工程实践或新的轻量方法中都没有实现。

导航图的主要部分在第二次迭代中定稿，如图 4。图 5 的模型直接来自工程第一个周期实际的设计文档，它显示了全局导航和外观/交互设计方案如何文档化。这些设计文档加在一起，详细说明了用户界面的全局导航的许多问题。

理解老师们扮演的各种角色的要求，并且清楚地解释他们的任务，对于制订快速、柔性、容易学会的在应用的各种工作上下文间导航的方案无疑是很重要的。一旦理解了各种任务的结构，就可以将任务类和交互上下文组织起来，通过一次鼠标单击在用户感兴趣的任务间切换。通过使用一些简单的外观隐喻，比如拇指形标签、hanging folder、文件夹，就生成了一个三层的可视结构，实践证明它比传统的 Windows 风格的 tree view 要更加容易学习和使用。

屏幕空间总是一个稀缺的资源，对运行在中等分辨率的显示器和浏览器上的应用来说尤其珍贵。为了解决这个问题，设计方案包括了让组件展开和自动隐藏的内容。比如，最上层的导航条（拇指形标签）被点中时展开就能提供更多的工作空间。又比如图 4 上的 dynamic workspace，它不仅仅是对 Apple OS-X 的“dock”的模仿。work space 提供了一个紧凑的储物箱，允许用户方便地从各种个人和公共资源中收集需要的文档和片断放到课程计划里，或者分发给学生。

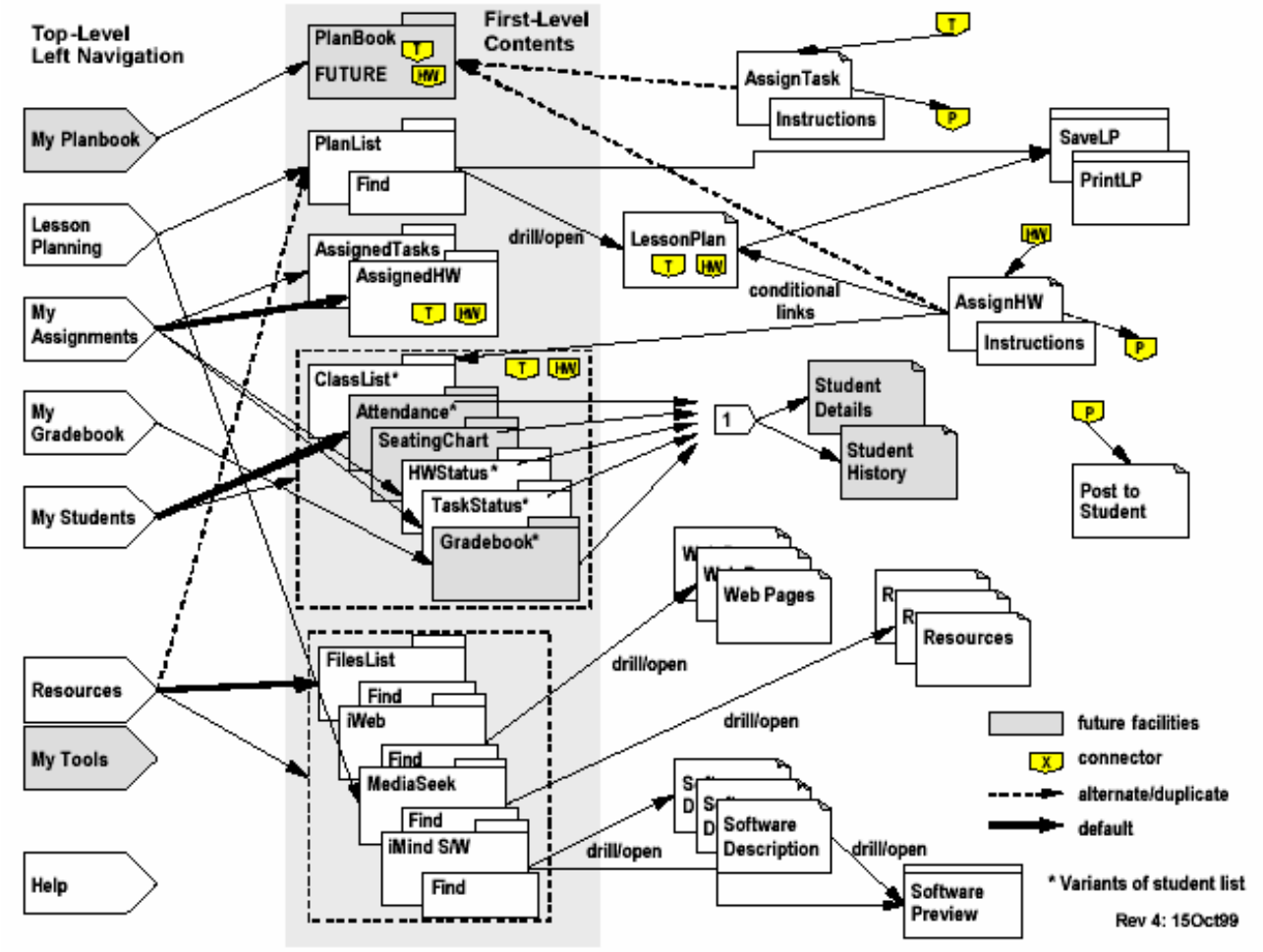


Figure 4 - Main portion of the navigation map for the classroom application.

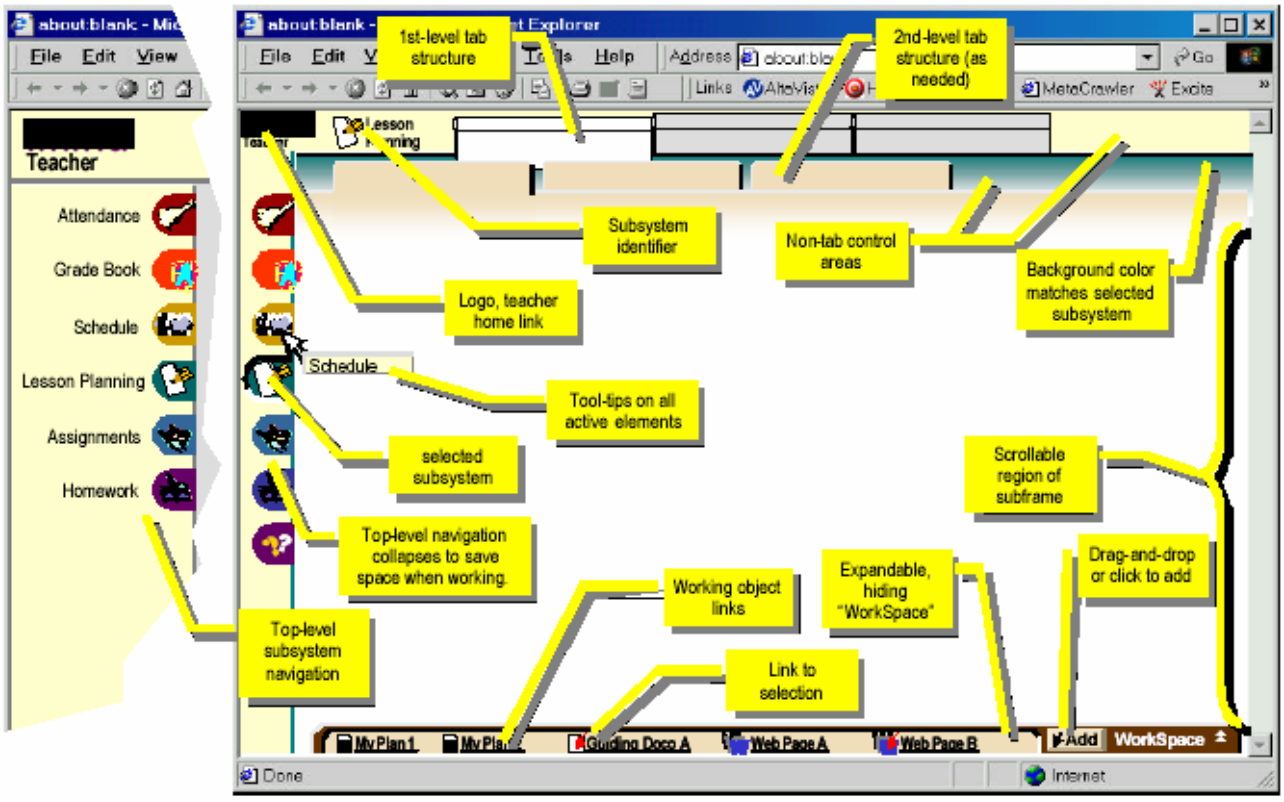


Figure 4 – Navigation architecture and design scheme.

任何产品设计的成功与否，最终都可以以其被接受程度来衡量。通过最少的指导，各个专业的授课老师们都能马上使用系统的功能。实际上，设计者能够完成如此严格的任务：老师们通过平均“一页教程”就能马上高效地使用系统。

学到的教训

我们从这个项目里学到了很多东西。糟糕的管理最终导致了开发这个系统的公司破了产，使得很多雄心勃勃的功能没有完成。尽管管理层反复无常，频繁带来变更，尽管编程资源不断被投入到毫无产出的演示系统中去，课程计划模块和其他一些核心功能还是开发出来并且发布了。

这次和其他敏捷的用途为中心设计的经验清楚地向我们表明，提升可用性决非没有代价和风险，即使使用流线型的方法和压缩的方案时也如此。特殊的训练和技能是必须的，而并不是每个开发团队都能得到这些资源，缺乏敏捷开发知识的团队尤其如此。而且，在编码时，由于不小心、或甚至善意的改变或求精都很容易破坏掉最好的设计。用户界面设计者和程序员之间紧密的配合是重要的，程序员必须完全服从过程并且相信其好处，即使他们并非对每个设计细节都能理解。

柔性、可伸缩的用途为中心的设计过程，是协作的良好起点，但对于极限编程这样的敏捷过程来说，由于他们完全拒绝任何有 BDUF(big design up front 提前的大量设计)迹象的东西，一些规则和原理可能成为其障碍。究竟哪些规则应该被重写，什么原理可能需要折衷，这是最近正在争论和实验的课题。本文展示的概要更应被看作一个草稿，而非最终的模型。

参考文献

References

Anderson, J., Fleek, F., Garrity, K., and Drake, F. (2001) "Integrating Usability Techniques into Software Development," *IEEE Software, 18* (1), January/February.

Beck, K. (2000) *Extreme Programming Explained.* Boston: Addison-Wesley.

Cloyd, M. H. (2001) "Designing User-Centered Web Applications in Web Time," *IEEE Software, 18* (1), January/February.

Constantine, L. L. "Software by Teamwork: Working Smarter," *Software Development, 1* (1), July 1993.

Constantine, L. L. (1994) "Essentially Speaking," Software Development, 2 (11). Reprinted in L. L. Constantine, *The Peopleware Papers.* Upper Saddle River, NJ: Prentice Hall, 2001.

Constantine, L. L. (1995) "Essential Modeling: Use Cases for User Interfaces," *ACM interactions 2* (2).

Constantine, L. L. (1998) "Abstract Prototyping," *Software Development, 6* (10), October. Reprinted in S. Ambler and L. Constantine, eds., *The Unified Process Elaboration Phase.* San Francisco: CMP Books, 2000.

Constantine, L. L. (1999). "Persistent Models: Models as Corporate Assets," Software Development, 7 (11), November. Reprinted in L. L. Constantine, ed., *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001.

Constantine, L. L. (2000a) "Cutting Corners: Shortcuts in Model-Driven Web Development," *Software Development, 8* (2), February. Reprinted in L. L. Constantine, ed., *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001.

Constantine, L. L. (2000b). "Unified Hegemony: Beyond Universal Solutions," *Software Development, 8* (9), September. Reprinted in L. L. Constantine, ed., *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001.

Constantine, L. L. (2001a). "Process Agility and Software Usability," *Software Development, 9* (6), September. An expanded version can be found at http://www.foruse.com/articles/agiledesign.pdf.

Constantine, L. L., and Lockwood, L. A. D. (2001b) "Design Study 1: Active Table-of-Contents Control for Content Navigation and Customization." http://www.foruse.com/articles/designstudy1.pdf

Constantine, L. L., and Lockwood, L. A. D. (2001c) "Design Study 2: Structured Selection with a Multi-Modal Extended Selection List." http://www.foruse.com/articles/designstudy2.pdf

Constantine, L. L., and Lockwood, L. A. D. (2001d) "Design Study 3: Dynamic Workspace for Document Assembly and Navigation." http://www.foruse.com/articles/designstudy3.pdf

Constantine, L. L., and Lockwood, L. A. D. (1999) *Software for Use: A Practical Guide to the Models and Methods of Usage-Centered Design.* Reading, MA: Addison-Wesley.

Constantine, L. L., and Lockwood, L. A. D. (2001) "Structure and Style in Use Cases for User Interface Design." In M. van Harmelan, ed., *Object Modeling an User Interface Design.* Boston: Addison-Wesley.

Constantine, L. L., Windl, H., Noble, J., and Lockwood, L. A. D. (2000) "From Abstraction to Realization in User Interface Design: Abstract Prototypes Based on Canonical Abstract Components." http://www.foruse.com/articles/canonical.pdf

Corlett, D. (2000) "Innovating with OVID," *ACM interactions, 7* (4), July/August.

Ferré, Xavier, Juristo, N., Windl, H., and Constantine, L. L. "Usability Basics for Developers," *IEEE Software, 18* (1), January/February.

Fowler, M. (1999) *Refactoring: Improving the Design of Existing Code.* Reading, MA: Addison-Wesley.

Fowler, M. (2000) "Put Your Processes on a Diet," *Software Development, 8* (12), December. Expanded version on the Web, http://www.martinfowler.com/articles/newMethodology.html

Gottesdiener, E. (1999) "Rules Rule: Business Rules as Requirements," *Software Development, 7* (12), December. Reprinted in L. L. Constantine, ed., *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001.

Jacobson, I., Christerson, M., Jonsson, P., and Övergaard, G. (1992) *Object-Oriented Software Engineering: A Use Case Driven Approach.* Reading, MA: Addison-Wesley, 1992.

Jeffries, R. (2000) "Card Magic for Managers: Low-Tech Techniques for Design and Decisions," *Software Development, 8* (12), December. Reprinted in L. L. Constantine, ed., *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001.

Jeffries, R., Anderson, A. Hendrickson, C. (2001) *Extreme Programming Installed.* Boston: Addison-Wesley.

Kruchten, P. (1999) *The Rational Unified Process: An Introduction.* Reading, MA: Addison-Wesley.

Lockwood, L. A. D. L. (1999) "Taming the Wild Web: Business Alignment in Web Development," *Software Development, 7* (4), April. Reprinted in L. L. Constantine, ed., *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001.

McMenamin, S. M., & Palmer, J. (1984) *Essential Systems Analysis.* Englewood, Cliffs, NJ: Prentice Hall.

Nielsen, J., & Mack, R. L. (eds). (1994). *Usability Inspection Methods.* New York: Wiley.

Norman, D. A., and Draper, S. W. (eds.) (1986 *User Centered Design.* Hillsdale, NJ: Lawrence Erlbaum

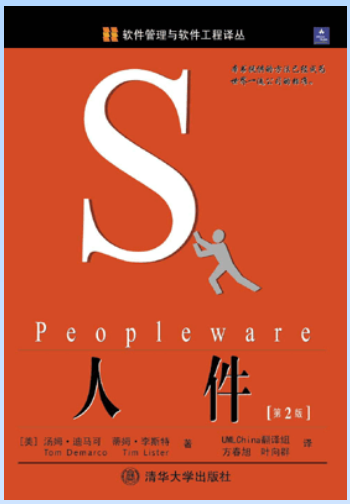
Parush, A. (2001) "Usability Design and Testing," *ACM interactions, 8* (5), September/October.

Pokorny, J. (2001) "Static Pages are Dead: How a Modular Approach is Changing Interaction Design," *ACM interactions, 8* (5), September/October.

Rumbaugh, J., Jacobson, I., and Booch, E. (1999) *The Unified Modeling Language Reference Manual.* Reading, MA: Addison-Wesley.

Thomas, D. (1998) "Web-time Development," *Software Development, 6* (10), October. Reprinted in L. L. Constantine, ed., *Beyond Chaos: The Expert Edge in Managing Software Development.* Boston: Addison-Wesley, 2001.

Windl, H., and Constantine, L. (2001) "Performance-Centered Design: STEP 7 Lite." Winning submission, Performance-Centered Design 2001, http://foruse.com/pcd/



Tom Demarco，Tim Lister

翻译：UMLChina 方春旭 叶向群

http://www.peoplewarecn.com

人件中文版网站

Frederick P. Brooks, Jr--《人月神话》第 19 章

近年来，软件工程领域的一个重大贡献是 DeMarco 和 Lister 在 1987 年出版的《人件》，我衷心地向我的读者推荐这本书。

Edward Yourdon

《人件》在 1987 年出版后，立即成为最畅销的作品。当人件第一版出版时，我写了一份评论，“我强烈推荐你买一份《人件》给你或你的老板，如果你是老板，那么为你部门的每一个人买一份，并给自己买一份”。这建议在 12 年后的第二版依然有效，并且更加热烈。

Mark A. Herschberg

我只学了办公环境的章节，就辞掉了原来的工作！

Joel Spolsky

我想，微软成功的原因之一就是公司里的所有经理都读过《人件》

Steve McConnell

由于本书第 1 版的赫赫声名，新版的《人件》是我不用看就会决定购买的少数几本书之一。

预订中文译本>>

Rational 统一过程对用户界面设计的支持

Chris Phillips、Elizabeth Kemp 著，herman 译

查看评论

摘要

Rational统一过程(RUP)是用例驱动的迭代软件工程过程。RUP中的用户界面设计包括用户界面建模与用户界面原型。本文涉及两种支持映象——可延展的用例表格和UI(用户界面)部件串——它们是连接用户界面建模与用户界面原型的桥梁。它们支持“事件流”故事板，用户界面元素群组和UML(统一建模语言)边界类定义，以及用户界面原型草图。

关键词

用户界面设计，Rational统一过程，用例建模，UML

1 简介

Rational统一过程(RUP)是用例驱动的迭代软件工程过程(Jacobson, Booch and Rumbaugh 1999, Krutchen 2000)。RUP是Rational开发的Objectory (Jacobson, Christerson, Jonsson and Overgaard 1992)的扩展。RUP是一种为软件开发和组件重用提供指导、模板与工具的用户化、对象化的结构。它聚集了广泛适合于各种项目并以达到最终用户要求为目标的最好的实例。

基于RUP的模型用统一建模语言（UML）描述(Fowler and Scott 1997, Quatrani 1998, Richter 1999)。UML由半正式的图表符号集合组成，看起来正在变成面向对象建模语言的标准(Kobryn 1999, Rumbaugh, Jacobson and Booch 1999, UML Revision Task Force 1999)。UML为软件提供的几种视图结构，支持静态与动态建模。符号集包括用例图、活动图、类图、交互图和状态图。

用例是RUP的中心，是后续开发的基础。它们提供一个功能（任务）视图，为与系统交互的角色建模。每一用例描述系统的某一用途，包含交互顺序，(隐含)对象处理。在UML中，用例图表现了系统上下文级的用例。用例图显示系统边界，外部角色，及存在的任何结构关系。尽管活动图可以用来提供概要的工作流图，UML没有提供特殊情况下的用例细节描述语法。

第二节将简要回顾RUP对于用户界面设计的支持。第三节描述用例故事板，介绍辅助用户界面元素定义的扩展表用例表示法。第四节讨论RUP怎样通过UI（用户界面）元素的群组、UML边界类定义和用户界面缩略图支持用户界面原型。

2 RUP 与用户界面设计。

使用RUP进行用户界面设计有两个层面：用户界面建模、用户界面原型。这一活动的主要输入是描述如何使用系统的用例模型。

在用户界面建模的层面上，每一用例都由一个用例故事板来描述，它从概念上描述用户界面如何支持一个用例(Krutchen, Ahlqvist and Bylund 2001)。包含如下特性：

1）事件故事板流：对角色与系统之间交互的高级文字描述。

2）交互图：对象在用例中如何相互作用的细节图示。

3）类图：描述边界类和参与用例实现的媒质

4）可用性需求：关于质量、系统每一使用的需求文本描述。

5）用户界面原型含义：组成用户界面的UI元素含义，包括它们与边界类之间的通讯。

特性2和3是受到直接支持的——类图与交互图都是UML的一部分。

符号	流	角色	用户/系统分离	细节水平	例外	其它问题
活动图	直接	不显示	差	一般	到原处	可视化
文本	暗示	显示	一般	好	不符合	可以加长
表格	直接	显示	好	好	不符合	可以加长

图1：用例表示方式比较

可用性需求（特性4）通常是文本式说明。本研究报告的重点放在对特性1和5的支持。“事件流”故事板将在第三节讨论，用户界面原型的联接放在第四节。

用户界面原型层包括“低精度”与“高精度”的设计、评估。低精度原型(Rettig, 1994)是本文的重点，包括草图与模型，使用纸、笔或可能的“演示”工具，如微软的PowerPoint。

3 用例“事件流”故事板

用例“事件流”故事板是由简短陈述组成的、对角色与系统之间交互的高级文本描述。它逐步描述相应用例(Krutchen, Ahlqvist and Bylund 2001)。尽管描述用例行为可以使用不同的方法。此时，许多评论(比如Cockburn 2001, Fowler and Scott 1997, Richter 1999) 支持用例对详细文本描述的需求。用例可能包含另一用例，通常采用包含主线、支线流的结构格式。表格表示法是更结构化的文本形式，它把角色与系统的动作为分两列。

用户界面描述是对设计的约束，为了避免出现绝对的用户界面描述，在用例细节描述时要小心谨慎。特别是，不要过早决定交互序列、显示及I/O方法。实质上，用例 (Constantine and Lockwood 1999)在探索排除这种可能性。

在近来的一些研究中(Phillips, Kemp and Kek 2001)，人们评估三种看起来在用户界面设计初期有价值的用例表示法。它们是UML活动图、文本描述和表格描述。人们通过所提供信息的水平和总量，及支持活动的用户界面设计初始输入可能值来评估这三种表示方法。

评估结果见图1。三种方法都不理想，进一步来说，图形与文本表示法是互补的。在用文本表示用例时，活动图能提供总的事件流视图。活动图可以显示活动时间顺序和活动发生时的情况。对于支持用户界面设计来说，表格是最佳表示方法，因为它可以显示细节，清楚地分开用户与系统行为，并适度列出其中的交互关系。

图2是用表格格式描述的图书借阅系统中的一个用例。这个应用程序中，用户可以查找图书，要求借阅已预定(借出)的图书，并(或)显示他们的借阅记录。进行每一次借阅或显示借阅记录，系统都要求用户认证。

图2中的图书申请用例开始于系统显示一本书的详细资料，同一本书及所有已出借的册。用户选出其中一册，任意填写最后想借阅的时间，并提并借阅申请。系统检查用户认证，如果是有效用户，确定借阅申请。在每一步，用户都可以要求打印图书详细资料，或是中止这一用例。图2包括一条主线，支线S1、S2和一条例外处理E1(响应无效的用户输入)。注意，用户输入日期(灰底色)可选。

3.1 扩展表格用例描述方式

在上述研究之外，开发了扩展表格用例描述。它与活动图的可视流一起，用表格方式描述细节，并把角色与系统分开。

以图2所表示的用例为例的扩展表格用例描述见图3。UI元素列确定5个工作区(W1~W5)，4个功能元素——选择、提交、打印、取消(支持在其他两列中列出的活动)。它显示工作区如何与任务流联系起来，并且在每一工作区中显示多少数据。后者部分源自对象知识(来自UML类图)，部分来自构造用例的关系分析。

<table>
<tr><td colspan="2">用例：预约借书
角色：图书馆用户
目标：允许图书馆用户申请目前不在馆的图书
前置条件：展示书的细节并且书的状态为“不在馆”
后置条件：无</td></tr>
<tr><td>图书馆用户</td><td>系统</td></tr>
<tr><td>主线流
任何时间，可能
-要求打印书的详细资料
-取消申请
选择一册书，输入借阅生效期限。
提交申请
输入 ID</td><td>认证用户
确认申请，并通知图书馆用户每两周查看书籍是否可借出。
用例结束</td></tr>
</table>

支线流： S1：要求打印输出书的详细资料	打印书的详细资料 用例继续
S2：取消申请 用例结束	
例外处理： E1：无效 ID	显示信息 用例结束

图2：预约借书用例的表格表示方式

这种用例表示法的关键点是它独立于用户界面设计方式。在开发中，用户界面设计可以使用这种方法，既获得说明与任务流，又不依赖系统交互方式。非技术人员也很容易理解。

在通过RUP进行用户界面建模的关系中(第二节)，扩展表格描述支持特性1和5，也就是说，它提供“事件流故事板”，外加支持交互的UI元素需求描述。UI元素是与界面内容相关的抽象部件。在这一层上不定义任何元素的分布与群组，也不定义交互方式或是隐喻(外观&感觉)。功能元素可以用各种方法实现，比如，通过菜单选择，或是“拖拽”方式。

4 与用户界面原型连接。

UI元素接下来必须被分组并且安排以形成可视界面，第一步是安排屏幕空间。在第一个例子中，用类现实的方式描述元素，或是电子“告示贴”(Constantine and Lockwood 1999)，通过物理排列实验，可以达到这一要求。

UI元素群组确定交互空间上的元素的分布及主要布局。采用什么样的交互方式是UI元素群组开发引导过程的重点之一。这组元素可以，打个比方，放入一个直接操作空间，提供多种选项；或是换一种方法，排列成一个简单菜单序列，每一条只有一个选项。

用例：允许图书馆用户申请不在馆的图书

角色：图书馆用户

预处理：展示书的细节并且书的状态为“不在馆”

后续状态：无

图书馆用户	系统	UI元素
主线流 任何时间，可能 -要求打印书的详细资料(S1) -取消申请(S2)		标题 W1 作者 出版商 ISBN 册号 图书编目号* 分布 状态 选择 W2 (mm/dd/yy)
选择一册书 输入一个时间 提交申请 输入ID		提交 W3 用户ID/密码 → E1 W4 确认需求 每两周查询借阅记录

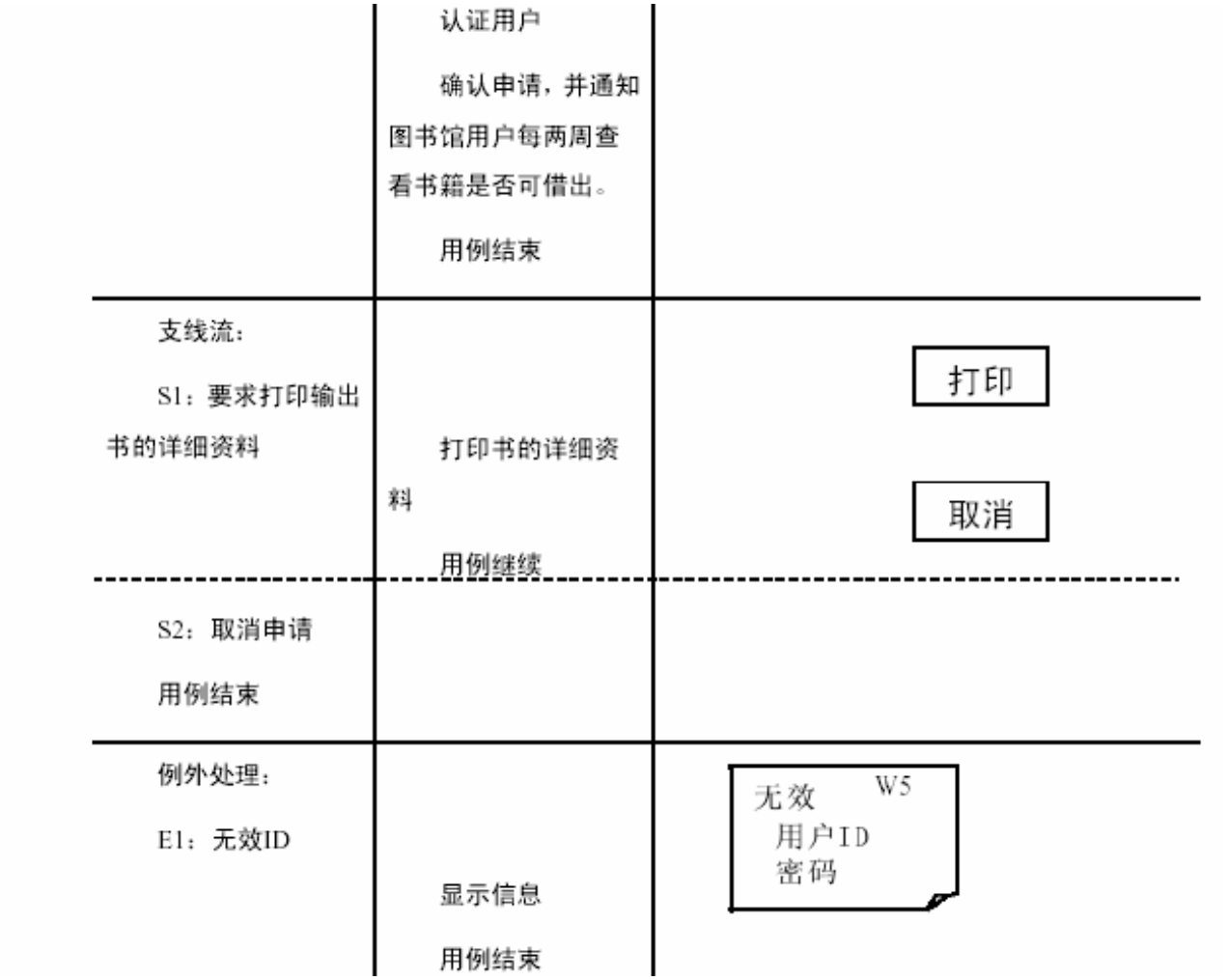


图3：预约借书用例的扩展表格表示方式

图4显示图3所描述的图书借阅用例的GUI(图形用户界面)所有可能的UI元素群。它基于交互的直接操作方式。工作区W1与W3，W4与W5的交迭意味着他们可以共享空间。在软件支持下，这些元素可以直接从表格化用例的UI元素列中选出，通过“拖拽”方式安排以形成群。

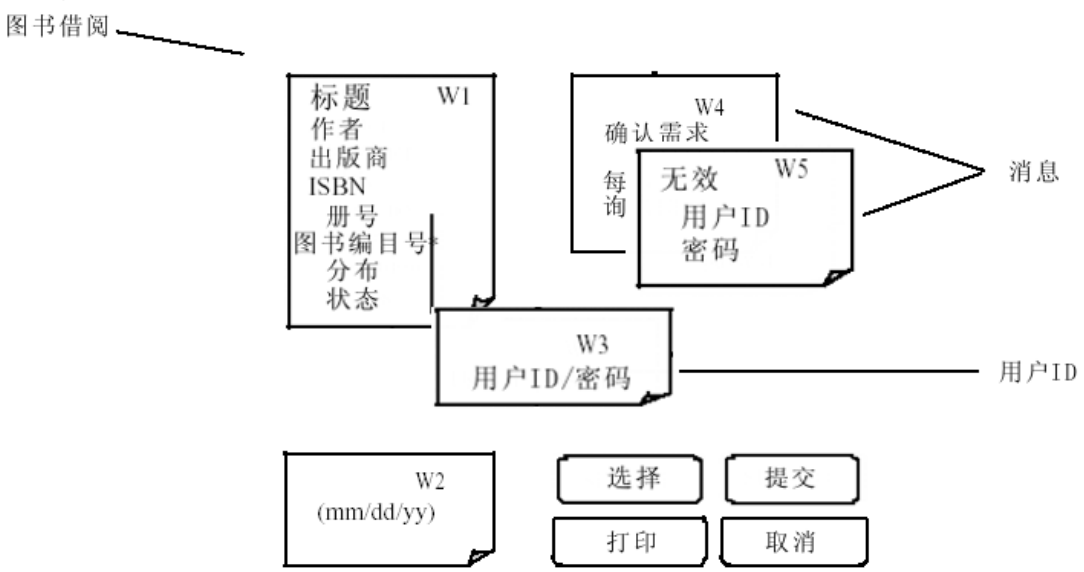


图4：预约借书用例UI元素群组，附带边界类标签。

要注意，这些群组不确定使用什么窗口部件，也不确定它们的行为，只是与UI元素之间的关系有关。所以它们不说明界面“外观&感觉”的细节。隐喻可能在细节层次上被加进来。加入一些注释也很有用，比如收集设计决策，或是提供初始草图与后来的原型时期指示。

图书申请、消息以及用户ID这类标签元素表现为UML边界类。边界类是UML中的一种标准stereotype(其他两种是实体类与控制类)。它表现为与信息收集、陈述相关的角色-系统界面(Jacobson, Booch and Rumbaugh 1999)。

边界类用于隔离用户界面的变化。这一类中的对象有：窗口、表单、窗格、菜单或打印界面。边界类可以通过UML关系图来校验(RUP用例故事板特性2，第二节)。

设计师使用UI元素群组设计用户界面草图与支持用例活动的“低精度”原型。这些草图首先与界面的“外观&感觉”相连，并与空间的分配与规划相关，包括窗口部件的行为与外观，上下文之间的导航。在传统的基于窗口界面中将包括确定主窗口与二级窗口及功能分配。

RUP用例故事板(第二节)中的特性4所表示的可用性需求必需在这一层中考虑，并贯穿剩下的整个执行。包括一些特殊数字需求，如最大执行次数、错误率。除此之外，还有像支持错误恢复与柔性工作之类的定性需求。

图5显示上面讨论的图书借阅用例UI元素所有可能的GUI(图形用户界面)草图。包括对工作区W1，W2和W4(覆盖消息)的描述。四个功能元素表现为按扭。在这一层中，从用例获取数据，选择符合要求的工作区物理尺寸，依照任务流的需求检测工作区与功能元素的位置。可能有其他与系统运行相关的功能元素要加入，例如帮助信息。对于每一用例，这一步骤都要重复，草图不断结合，直到合成用户界面原型。

5 回顾与结论

本文描述了两种支持映象——扩展表格用例和UI元素群组。他们提供与用户界面设计相关的RUP两种中心活动(用户界面建模，用户界面原型)之前的桥梁。这些方法支持事件流故事板，用户界面元素群组，已定义的UML边界类，以及用户界面原型草图。

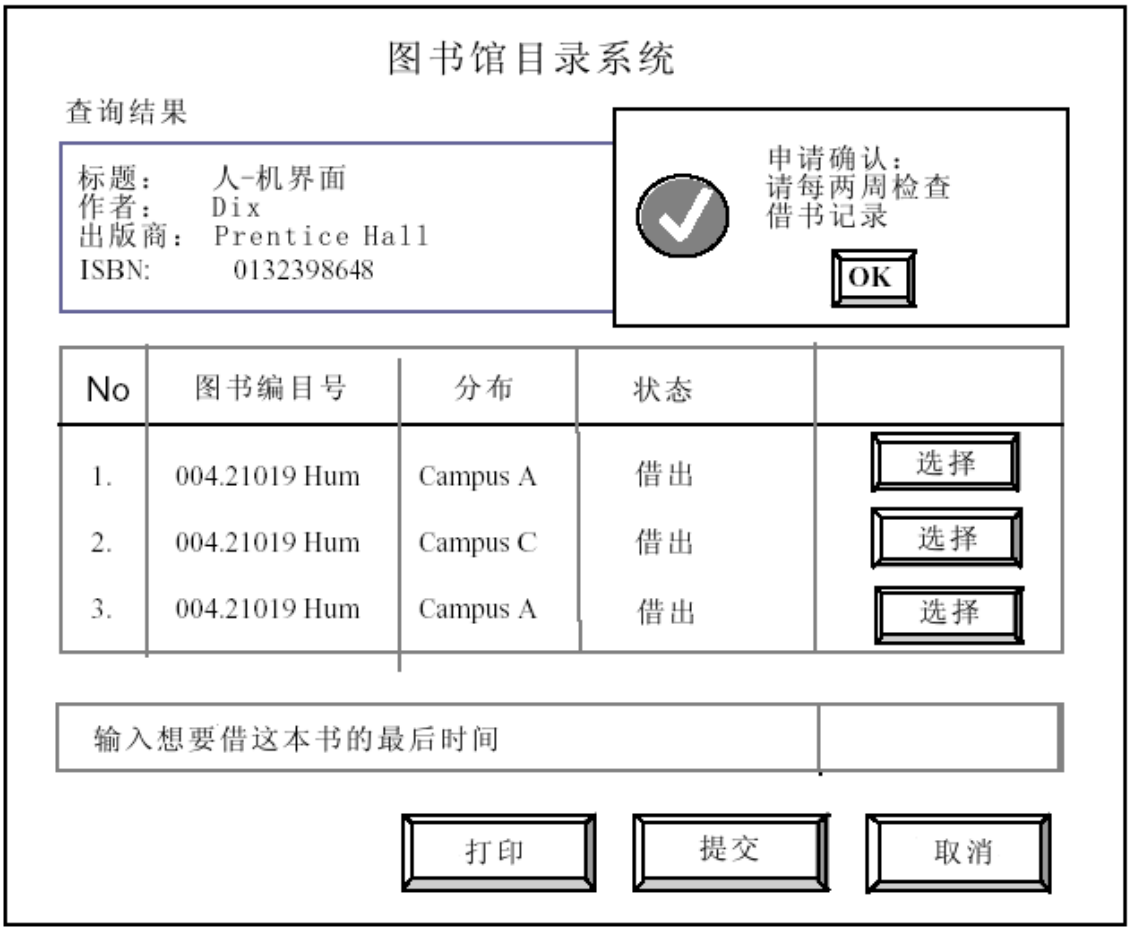


图5：带确认信息覆盖的借书用例UI草图

这些方法有以下几个优势：

表格用例：

- 描述本质任务序列，不关注可选活动。
- 合并所有的“被包含”和“扩展”用例关系。
- 辩识必须控制的例外情况，包括用户输入日期时的错误。

扩展表格用例：

- 提供既图形化又文字化的事件流故事板。UI元素的视觉描述使得它在所有相关开发过程中更容易理解。
- 尽管元素与应用程序的内容相关，它不依赖于任何界面描述方式，没有特定的“外观&感觉”。
- 提供产生一套处理的要素(表示所有相关属性的扩展表格符号)，用于检测执行。

UI元素群组

- 确定界面空间上UI元素的分配与主要规划。
- 支持不同的用户界面风格试验，在广泛水平上研究与屏幕布局之间的关系。
- 适当地直接确定与UI元素的任何特殊排列相关的UML边界对象需求。

总的来说，本文中描述的符号与方法提供一种系统方式，把兼顾图形与文字表述的事件流故事板变为早期的系统原型。通过软件支持，UI元素可以直接从扩展表格用例的第三列中选出，排列到表单群中。我们的工作就是这个过程。

上面的过程在许多应用软件中采用，包括在本文中提到的完整的图书馆系统，大学学生注册系统，台基(kiosk-based)公众信息系统。这之中，其他方式的可测量性还需要完整测试。目前，梅西大学关于用户界面设计的研究生毕业论文正在尝试这一方法。

参考文献

COCKBURN, A. (2001) Writing Effective Use Cases, Addison Wesley, Boston.

CONSTANTINE, L.L. & LOCKWOOD, L.A.D. (1999): Software for Use: A Practical Guide to Models and Methods of Usage-Centered Design, Addison-Wesley, Reading, Massachusetts.

FOWLER, M. & SCOTT, K. (1997): UML Distilled: Applying the standard object modeling language, Addison-Wesley, Reading, Massachusetts.

JACOBSON, I, M. CHRISTERSON, P. JONSSON, & G. OVERGAARD. (1992): Object-Oriented Software Engineering: A Use Case Driven Approach. Addison-Wesley Publishing Company, Reading, Massachusetts.

JACOBSON I., BOOCH G. & RUMBAUGH J. (1999): The Unified Software Development Process, Addison-Wesley, Reading, Massachusetts.

KOBRYN, C. (1999): UML 2001: A Standardisation Odyssey, Comm. ACM, Vol 42, No 10, 29-37.

KRUCHTEN P. (2000): The Rational Unified Process: An Introduction (2nd ed), Addison-Wesley, Reading, Massachusetts.

KRUCHTEN P., AHLQVIST S. AND BYLUND S. (2001): User Interface Design in the Rational Unified Process, in Object Modeling and User Interface Design, ed Van Harmelen, Addison-Wesley, Reading, Massachusetts.

PHILLIPS, C.H.E., KEMP, E.A. & KEK, S.M. (2001): Extending UML use case modelling to support graphical user interface design, Proc. of ASWEC 2001, IEEE, Canberra, Australia, 26-28 August 2001, 48-57.

QUATRANI, T. (1998): Visual modeling with Rational Rose and UML, Addison-Wesley, Reading, Massachusetts.

RETTIG, M. (1994): 'Prototyping for Tiny Fingers', Comms. of the ACM, Vol 37, No 4, 21-28.

RICHTER, C. (1999): Designing Flexible Object-Oriented Systems with UML, Macmillan, Indianapolis, Indiana.

RUMBAUGH J., JACOBSON I. & BOOCH G. (1999): The Unified Modeling Language Reference Manual, Addison-Wesley, Reading, Massachusetts.

UML REVISION TASK FORCE (1999): OMG UML v1.3 Revision and Recommendations document ad/99-06-08, Object Management Group.

《人件》— 老板，别把开发人员当成牲口

新闻

《人件》提供预订>>

相关文章

《人件》实践之：SAS公司
关注程序员自己的文化——专访Tom DeMarco
别把开发人员当成牲口
Brooks在《人月神话》中的评论
Alan Cooper的评论
办公室空间，下一场革命
《人件》在计算机行业的实践（待续）
人的问题：关于《人件》
Edward Yourdon的评论
开发人员是人吗？

各版本封面

英文，日文，德文

首页 购买 留言板

中国“人件”非正式调查

查看评论

1. 这种事情在你们的软件公司情况如何？

一个缺少洞察力的例子：一个老板表露出被他部下的个性吓倒的极端信号。他有一个非常有天赋的员工，每年大部分时间花在回访客户的路上，结果以报销单为生。该员工的一份开支分析报告显示，他在吃饭上的花费与其他出差者不协调，他的餐饮费用比其他出差者多一半以上。在一次公开的例会上，老板因一时愤怒竟然污辱那位员工，说他是一个“食物罪犯”。然而，那个员工的总费用并没有超出标准——无论他在饮食方面的额外费用有多少，他都从其他方面节省回来。此人并没有比其他人花费更多，只是与其他人不同罢了。

——《人件》第 2 章：做吉士汉堡，卖吉士汉堡

2. 你们的软件公司有这样的“催化剂”吗？

几年前我在进行一门设计课程的内训时，有一个高级经理强留我谈话，要求我评估一下上课的一些人（他的项目组成员）。他对一位女士感到特别好奇。很显然他对她的成绩表示怀疑：“我看不出她为项目做了些什么——她既不是一个得力的开发人员或测试人员，也不是有任何其他什么特长的人”。通过一些调查之后，我发现了这个令人迷惑的事实：在她就职这家公司的十二年间，这个令人怀疑的女人从事的项目都取得了巨大的成功。她为项目做了什么是不明显的，但是有她在项目总是成功的。通过在班上对她进行为期一周的观察以及通过和她的同事的谈话，我得出的结论是：她是一个极品催化剂。有她在队员们自然会团结得更好，她协助人与人之间的交流并使大家融洽相处。当她是项目的成员时整个项目变得更加有趣，当我试图将向那个经理解释这个想法时，我感到非常震惊。他恰恰就不承认催化剂作用对一个项目来说是重要的。

——《人件》第 2 章：做吉士汉堡，卖吉士汉堡

3. 你们的软件公司发生过这种事吗？

几年前，我与南加州一个大项目的项目经理交流各自的艰辛历程。他开始叙述将项目和疯狂的时间表压到他下属的身上产生的影响。一是发生的两宗离婚案，其中原因可以直接追溯到与他的人经常加班有关；再者就是一个员工的孩子吸毒，其中原因可能是由于在过去的一年里，孩子的父亲太忙，未能尽到做父亲的责任，最后，测试团队的负责人又神经崩溃。

在他继续叙述这些可怕的事情的时候，我开始意识到，在他这种奇怪的工作方式中，此人一直在炫耀。你也许怀疑，如果再有另外一两个人离婚和一个人自杀，这个项目原本会取得圆满的成功，至少在他的眼里是这样的。

——《人件》第 3 章：维也纳在等着你

4. 你们的软件公司有令人恶心的家具警察（行政人员）吗？

警察心理

家具警察的头目是那个家伙，在你的员工要搬进新办公室前一天，他先走进去徘徊几圈，脑子里开始这样盘算着：

“看看，每件东西都是那么漂亮、一致！甚至分不出是在 5 楼还是 6 楼！但是一旦那些人搬进来之后，一切都要被毁了。他们会挂上照片和个性化他们小小的工作间，一切都将杂乱无章。他们也许要在我心爱的地毯上喝咖啡，甚至就在这儿吃午餐（耸肩），噢，天哪，噢，天哪，噢，天哪……”

这就是颁布规定的那个家伙，规定晚上每张桌子必须保持清洁，并且禁止在隔板上悬挂任何东西，或许公司的日历除外。我们知道一家公司的家具警察甚至专门为咖啡溅出将一个号码列入紧急电话号码清单，而这份清单贴在每部电话的贴花纸上。当有人打那个号码时，

——《人件》第 7 章：家具警察

5. 在你们的软件公司，以下几项关于开发人员工作环境的指标如何？以你们公司的工作环境，团队如果参加作者组织的编码演习，是排在前 1/4 还是后 1/4？

表 8-3 在编码演习中成绩最好与最差者的工作环境

环境因素	成绩在第一个 1/4 的人	成绩在第四个 1/4 的人
1．你有多少专用工作空间？	78 平方英尺	46 平方英尺
2．安静程度可以接受吗？	58%可以	29%可以
3．私密性可以接受吗？	62%可以	19%可以
4．你可以让你的电话不发出声响吗？	52%可以	10%可以
5．你可以转接你的电话吗？	76%可以	19%可以
6．经常有人无故打扰你吗？	38%是的	76%是的

可以看到，成绩排前面 1/4 的人，那些表现得最快又最好的人，他们的工作空间与成绩排后面 1/4 的那些人的工作空间根本不同，前者的工作空间是安静的，更具有私密性，更不容易受干扰，而且工作空间更大。

——《人件》第 8 章 朝九晚五无所为

6. 你们的软件公司发生过这样的事吗？

在我早年做开发人员的时候，我被特许致力于一个由莎伦·温伯格管理的项目，此人现任 CODD 和 DATE 顾问集团的总裁。她是现在我想到的开明管理方面的活典范。在一个下雪天，我从病床上爬起来为参加一次用户演示会整合我们的不太牢靠的系统。莎伦进来后发现我强撑着在控制台上工作，她很快离开，几分钟后端来一碗汤，等我喝完汤后，又对我进行一番鼓励，我问她手头有那么多的管理工作要做，怎么还为这类事情抽出时间，她向我露出她特有的微笑说“汤姆，这就是管理呀。”

——《人件》第 6 章 苦杏仁苷

7. 作为开发人员，你用于“思考”的时间占多少比例？

如果要你负责完成某项任务，你应该花在实际做这项任务上的时间应该占多少比例？不是百分之百。应该预备一些时间用在这些事情上：组织头脑风暴会议，研究新方法，构思如何避免做一些次级任务，阅读，培训，甚至只是消磨时间。

但所有这些都是婆婆妈妈的事情。每个人都知道这些并且都会按照规定行事，对吗？不对。我们是如此一门心思地去“做事”，我们花费少于百分之五的时间用于计划、研究新方法、培训、读书、评估、预算，安排进度以及人事安排等综合活动。（百分之五的数字来自于对一个系统开发项目的分析，但它似乎可以应用得更广，也许可以用于整个工薪阶层。）

——《人件》第 2 章 做吉士汉堡，卖吉士汉堡

8. 你们的软件公司有“工作狂”吗？

工作狂将加班加点，不需要补休。虽然或许他们的工作效率在降低，但是他们会工作过多的时间。如果给他们施加足够的压力，他们会在损害个人生活的道路上走很长的一段路。但是这只是一时的。这样的讯息迟早会传到甚至最专心的工作狂的耳朵里：

慢一点，你做得很好了，

在你死前你不可能完成一切心愿，

虽然今晚在边缘线上非常浪漫，

但是何时你意识到……维也纳在等待你？

——《人件》第 3 章 维也纳在等着你

9. 你们的软件公司有这种事吗？

在过去的一年中，我为一个项目做了一些咨询工作。该项目进展得很顺利，项目经理知道她能够按项目进度表按时提交产品。她被管理委员会叫来做项目进度汇报，她说她可以保证她的产品在最后期限 3 月 1 日准备好，完全根据最初估计的时间按时完成。高层经理认真考虑了这一意料之外的好消息，然后第二天又把她叫来。既然她可以按时在 3 月 1 日完成，他们解释说，那么最后期限就改到 1 月 15 日。

对那些信奉西班牙人理论的经理们来说，一张能够真正按时完成的进度表是没有价值的，因为它没有对工作人员带来压力。更好的做法是有一张毫无希望、不可能按时完成的时间表，它可以榨取员工们更多的劳动。

——《人件》第 3 章 维也纳在等着你

10. 你们的软件公司有这种事吗？

实际上一些公司使用一个公共寻人系统去打断员工——那些在努力思考的人，只是为了找到某一个人：嘭！[电噪音]注意，注意！呼叫保罗?波丘拉克，请保罗?波丘拉克给传呼中心打电话。如果坐在自己的位子坐着不动，你常常可以看到三四十个拿薪水的员工听见第一声响声就抬起他们的头有礼貌地聆听完通告，然后重新低下头回想在工作被打断之前他们正在干什么。

——《人件》第 7 章 家具警察

11. 作为开发人员，你坐的位置有窗户吗？

我们在一幢新建的摩天大楼的第 20 层第一次见到带窗户的走廊平面图，第 20 层的每个方向本来都有极好的风景，但实际上这些风景却没有人看到过。那个大楼里的人也许就和在地下室里工作一样。

用家具警察的眼光来看，地下室的空间确实更好，因为它本身为统一布置提供了更充分的条件。但是开发人员在自然光下工作得更好。他们在有窗户的空间感觉更好，并且会把这种感觉直接转化为更高的工作质量。人们也不想在一个完全统一的空间工作，而是希望按自己的喜好和品味来布置自己的工作空间，这些事实是和人打交道时诸多不方便之处的典型。

——《人件》第 7 章 家具警察

12. 作为开发人员，你的平均办公面积是多少？

为了在圣特雷莎建设新的研究机构，在工程计划之前，IBM 公司不顾所有的工业标准，仔细研究了那些会使用该场所的员工的工作习惯。这项研究是由建筑学家吉拉德?麦克丘在 IBM 公司地区经理的协助下设计的。研究员观察了在当前的工作场所和推荐的工作场所实物模型中进行的工作过程。他们观察程序员、工程师、质量控制人员和经理们从事他们的日常活动。通过研究，他们得出的结论是，为将使用新的工作场所这些员工提供的最小活动空间应符合下列要求：

- 每个人 100 平方英尺的专用空间
- 每个人 30 平方英尺的工作面积
- 采用封闭式的办公室或者 6 英尺高的隔间的形式，以免受到噪音的干扰

——《人件》第 9 章 在空间上省钱

13. 作为开发人员，你的工作经常被打断吗？

工作时段　工作类型　终止工作时段的干扰原因是什么？

2:13—2:17　编码　电话

2:20---2:23　编码　停下和老板聊天

2:26---2:29　编码　来自同事的问题

2:31—2:39　编码　电话

2:41—2:44　编码　电话

——《人件》第 10 章　脑力时间 vs.体力时间

14. 你的软件公司的工作环境是好看还是好用？

我们发现，开发人员述说他们对工作环境问题的关心，却被当作是追求地位而不予考虑，这尤其令人难过，因为更常见的情况是，更高层的管理层在设计员工的工作空间方面犯有追求地位的罪行。努力工作按时交付优质产品的人是不关心办公室的外表的，但是老板有时却会关心。因此我们看到反常的现象是，整个不能工作的空间被特意用长绒地毯、黑色铬合金的家具、比员工还占用了更多空间的玉米植物以及精美的面板来美化。下次如果有人骄傲地领着你参观一个新设计的办公室，努力思考一下他炫耀的是空间的功能还是外表，当然，常常是它的外表。

——《人件》第 12 章 把门带上

15. 作为开发人员，你有过这样的感受吗？

“在所有人上班之前，一大早我已经尽力完成了我的工作。”

“一个晚上干晚一点，我可以做两三天的事情。”

“办公室整天像个动物园，但在下午大约 6 点的时候，一切都安静下来了，你终于可以真正完成一些事情了。”

——《人件》第 8 章 朝九晚五无所为

16. 你们的软件公司要求“统一”吗？

统一性对于不安全的独裁体制（例如教会学校和军队）是如此重要，以至于它们甚至加强了着装规范。短裙长度不一或夹克衫颜色不一都是一种威胁，因此要加以禁止。不允许任何东西破坏长长的几排几乎清一色的队伍。完成任务的问题变成了任务能不能由看起来毫无区别的人来完成的问题。

公司有时也强调着装标准。这些公司也不是如此极端以至于严格要求统一服装，但是它们禁止个人有良好的判断力。如果是这样，它的影响是破坏性的。人们不会谈论或考虑其他事情了。所有有用的工作都停了下来。最有价值的人开始意识到不是因为他们的真正价值而受赏识，意识到他们的工作贡献还不如剪短的头发和领带重要，他们可能会离开。公司剩下的人拖着沉重的步伐继续工作，试图证明拥有合适的人不重要。

——《人件》第 14 章 霍恩布洛尔因子

17. 你们的软件公司招人时举行过“工作试讲”吗，还是人力资源代劳？

这个主意非常简单。你要求一个候选人准备 10 至 15 分钟，跟过去工作有关的某个方面的陈述。它可以是关于一项新技术以及第一次尝试该技术的经验或者是关于历经困难得来的管理教训或者是关于一个特别有趣的项目。候选人选择主题。规定好日期，召集将是新雇员同事的那些人，组成一支小听众队伍。

当然这个候选人会紧张，或许甚至会不愿意经受一次这样的经历。你必须向他解释所有的候选人都会对试讲感到紧张，并且说明你支持举办试讲的理由：看看不同的候选人的沟通技巧，并在雇用过程中给未来的同事一个角色。

试讲结束，候选人走了以后，举行一个任务报告会，每个与会者都要对那个候选人适应工作的能力和他或她是否能够很好地适应团队发表意见。虽然最后是你决定是否雇用那个候选人，但是从未来的同事那里反馈的信息是宝贵的。甚至更重要的是，任何新雇员更有可能被轻易地吸纳进小组，因为小组的所有其他组员都在选择这个候选人的问题上发了言。

....

不久就很清楚，试讲的过程有助于促进一个新雇员和现有团队成员的融合。一次成功的试讲是一种成为同事的证明方式。事物的反面似乎同样也是有效的。失败的试讲对全体团队成员而言是一次士气的鼓舞，他们不断证明受雇为团队工作比他们的简历被我看中更有意义。

——《人件》第 15 章 雇用一个变戏法的人

18. 你们的软件公司有“胶冻团队”吗，引起经理的害怕？

从胶冻时起，这个团队本身已经是他们力量的真正焦点了。他们呆在这个团队中为的是共同的成功，为的是一起完成目标，任何一个目标。再将他们的注意力集中到公司在这个项目中的利益上是无益的，它只会使成功变得毫无意义。

在胶冻团队工作的人非常投入，他们有足够的心理准备去对抗纳瓦隆大炮，只是为了通过养老金信托系统第三版本的认可测试。你得提醒他们，他们正在努力完成的不是“相当于战争的道德价值”（MEOW）。

通常，胶冻团队对他们建立的产品有着共有的情感。参加人员很高兴将他们的名字组合在一种产品上，或产品的一部分上。团队中的个人渴望同事的认可。在产品将要完成时，团队的空间布满了和产品有关的装饰品。

...

经理们不是他们团队的真正成员（详见第 23 章），因此排斥他们的想法比将他们合并成小组的想法更强烈。小组内的忠诚比将小组与公司联系在一起的忠诚更强烈，那么就会有一种可怕的想法，一支紧密团结的团队可能会一团糟，并且将其所有的精力和热情用于竞争。因为这些原因，不安全的经理受到私党的威胁。与一群统一制造的、同样的、可互换的并且未结合的员工一起工作，会使他或她感觉更好一些。

——《人件》第 18 章 整体大于部分的总和

19. 你们的软件公司有这样的老板吗？

一家加州公司的客户信息系统项目。项目的规格说明已经写出来了，而且我们准备着手为内部设计绘制蓝图。老板把我们召集到一起，并且给了每个人一幅标明了远在长岛的一幢办公楼的线路图。他解释说，在那儿有一个空会议室，我们可以不受打断地工作。他会留下来，帮我们处理一切外部干扰，除了非常重要的电话。他告诉我们说：“完成任务后回来。”两个多星期以后，我们带着刚出炉的设计回来，而他在整个期间从未来过电话或顺便来过一次。

.....

这样一个计划会让你和你自己的管理层以及同事争论一番，因为它是一个如此大胆的计划。他们会问你：你如何知道你的人这一分钟没有虚度？你怎么能确定他们不会在十一点钟停下工作去吃午饭，并且在喝酒中度过他们的下午？简单的回答是，可以根据他们带回来的产品知道，可以根据他们的成果了解他们。如果他们带回的是一个经过仔细考虑的、全面的结果，他们就是工作了。如果不是这样的。他们就是没有工作。直接监督对开发人员来说是一个笑话，那是为犯人准备的。

——《人件》第 22 章 思想开放

20. 你们敢这样做吗？

它们鼓励你和家具警察较量，与公司的熵作斗争，打败团队自杀趋势，提高产品质量（即使时间不允许），废除帕金森定律，给正式的方法论松绑，提高你的 E-因素，开放你的思想，

....

一家加州计算机公司，位于程序员区域的传呼系统受到一支鲁莽的游击队的袭击。从此线路一直就断了。因为程序员坐在以前是一个组装池的地方，天花板（和传呼系统的扬声器）高 16 英尺。谁能爬那么高？也许只有霍尔加·丹斯克了。

在明尼尔波利斯，一个大项目的经理拒绝把他的人迁移到新区去。(“新”在这只指更小的和更吵的。)管理者对他的拒绝感到吃惊；他们从没有想到会有这种事。工作人员本应照他们所说的去做。那个经理有不同的理论，即工作人员应该工作。他收集了关于新环境的证据足够说服他自己他们不能在新的工作场所工作。因此经理的恰当职责是说不。如果他是独自一人采取这个立场时，就可能很容易地控制他。但是他不是孤独的。他有霍尔加?丹斯克站在他一边。

——《人件》第 26 章 霍尔加?丹斯克

21. 你们的软件公司讲究喊口号吗，公司的标语是不是到处都是？你们公司的口号和其他喊口号的公司有显著差别吗？还是一律是“诚实、拼搏...”？

按照他们的说法（包括办公室里咖啡杯上的口号、招贴牌、别针、钥匙链以及奖章），这些宣传品是一种超越物质的胜利形式。他们似乎颂扬质量、领导艺术、创造性、团队精神、忠诚度以及许多公司美德的重要性，但是只表现为过分简单的口号，这向人们传递完全不同的信息：这儿的管理层相信只要用海报而不是靠努力工作和管理天分就能促进这些美德。每个人很快明白海报的出现是缺少努力工作和天分的确切信号。

像这些本应是宣传品主题的重要事情已经成为一种耻辱，但是执行起来使之更糟。以某家公司推出的海报为例，上面是一幅柔焦画面：汗流浃背的划桨者齐心协力地在起薄雾的清晨划着桨。画面下可看到部分文字：

团 队 精 神

……允许平凡的人达到非凡结果的原动力

这里海报所说的的“平凡的人”就是你和你的同事。平凡的人。(不要把它太当回事。)至少这些海报在态度上是一致的：同一家公司的有关领导才能的海报告诉我们“领导的速度决定了一群人的速率”。一群人。是的，又是指你们。

宣传品是如此的拙劣，足够使人起鸡皮疙瘩。他们正在损害健康的公司。

——《人件》第 27 章 再论团队自杀

22. 你们的软件公司是不是在推行 CMM?

假设你在为一个中等规模的生产成品商务软件的公司工作。公司主管认为 CMM 不是来自匹兹堡而是来自上帝的旨意。他们已经请 SEI 来评估公司的成熟度等级。评估是完整的，而且你们被招集在公司礼堂，SEI 的评估领导会登上礼堂的讲台宣布新的发现。礼堂大厅内一片寂静……

“女士们，先生们，我们有一些不平常的消息要宣布。我们已经完成了我们的评估工作，并且确定你们的 CMM 等级在第五级上。结果是这个公司已经证明它自己是……第六级！是的，是第六级。在我们来这里之前我们甚至都不知道有第六级。对我们所有人而言，这是不平常的一天。我们感觉似乎我们在等级阶段表上已经发现了一个新的等级。你们是现在人类所知道的最好的软件开发公司。谢谢。祝你们愉快。”

——《人件》第 29 章 过程改进程序

23. 猜想一下：你临死前会想到你现在在写的程序吗？

将你的思绪向前推，请等一下，推到遥远的未来的某一天，你要寿终正寝了。你活到很大岁数，比如说一百零一岁，你正老死在床上。你没有什么不舒服，只是老了。在这个年纪，思绪都是在过去。你正在整理记忆。你扪心自问，我的一生中什么事情是真正重要的而什么又是最不重要的？一点也不惊奇的是多年来彻底萦绕在你脑海中的许多关心的问题（例如，使巨大的“爆竹”（Whizbang）v6.1.1 的第 27 个工作版本保持稳定）不会十分突出地出现在你的临终评价上。不，你很可能想到温馨的家庭关系、儿孙、房子和所有有关房子的记忆。

——《人件》第 34 章 社区的形成

征稿

http://www.umlchina.com/xprogrammer/xprogrammer.htm