第 35 期（2004 年 3 月）

【新闻】

1 视频游戏公司采用SmartDraw进行标准化…

【访谈】

9 我思Argo在

【方法】

16 当企业计算遭遇嵌入式系统

20 探索《探索需求》

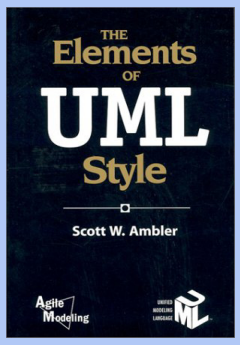
24 《UML风格》中译本节选

34 糟糕界面集锦—界面设计考虑要点

50 批判索尼爱立信的手机设计

61 为什么伟大的技术不能制造出伟大的产品

65 用UML进行超媒体建模



风格

X 非程序员
X-Programmer
软 件 以 用 为 本

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

http://www.umlchina.com/

本电子杂志免费下载，仅供学习和交流之用
文中观点不代表电子杂志观点
转载需注明出处，不得用于商业用途

Gerald Weinberg “探索需求”讲座举行

[2004/3/19]

2004 年 3 月 19 日（周五）上午 10:00-12:00，数十名 UMLChina 听众聆听了 Gerald Weinberg 的“探索需求”讲座。

温伯格(Gerald Weinberg)。对软件业影响巨大的几个人之一。

详细介绍请看他的中文网站（http://www.weinbergcn.com/）。

温伯格系列丛书，已出版的包括：

程序开发心理学>>

系统设计的一般原理>>

系统化思维导论>>

你的灯亮着吗>>

走查、审查与技术复审手册——对程序、项目和产品进行评估>>

成为技术领导者——解决问题的有机方法>>

《探索需求》样章，《非程序员》34 期

类似的讲座将持续举行，感兴趣者请留意 UMLChina 首页的公告。

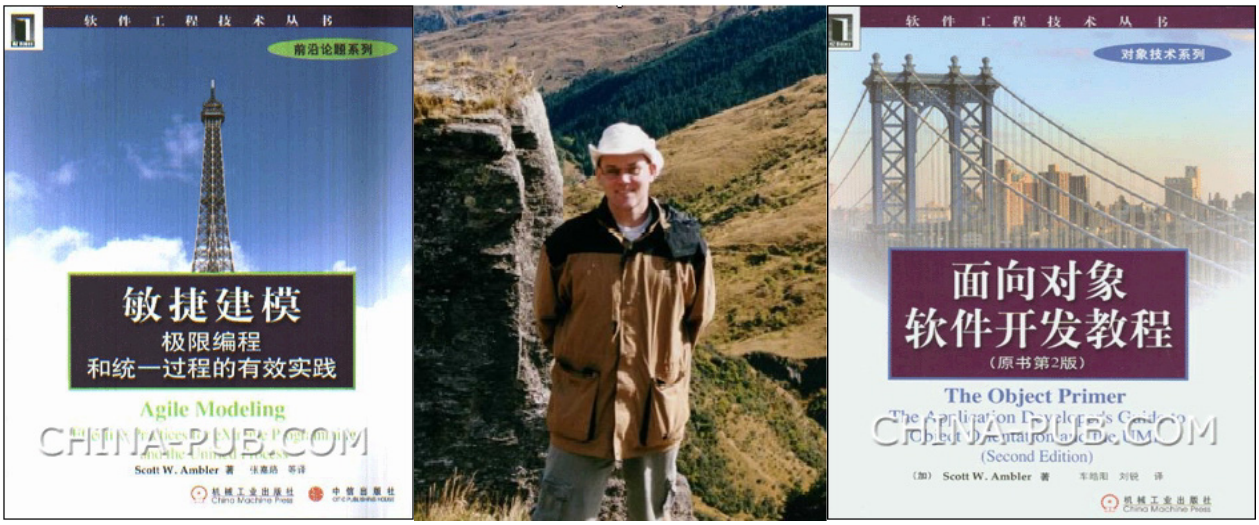
Scott W. Ambler“应用 UML 敏捷建模”讲座举行

[2004/3/11]

2004 年 3 月 11 日（周四）上午 9:30-11:30，数十名 UMLChina 听众聆听了 Scott W. Ambler 的“应用 UML 敏捷建模”讲座。

Scott W. Ambler，对象技术和敏捷方法大师。他是 Ronin International 的总裁，该公司是一家专门提供面向对象过程指导、体系结构建模和 Enterprise JavaBean (EJB)开发的咨询公司。Scott W. Ambler 著有多本对象技术畅销书，今年他的“Agile Database Techniques: Effective Strategies for the Agile Software Developer”获得了 Jolt 奖。

Scott W. Ambler 学习武术，有刚柔流空手道（徒手）的黑带和 Kobudo（器械）的绿带，此外还学习太极拳和瑜伽。他酷爱风景和野生摄影以及国际旅行。



声音记录：

*第一部分（SC4 文件，BT 下载）

*第二部分（SC4 文件，BT 下载）

*SC4 播放器

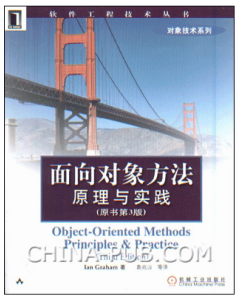
*BitTorrent 客户端下载

类似的讲座将持续举行，感兴趣者请留意 UMLChina 首页的公告。

Ian Graham“对象技术”讲座举行

[2004/2/29]

2004 年 2 月 29 日晚上 19:00-21:00，数十名 UMLChina 听众聆听了 Ian Graham“对象技术”讲座。



Ian Graham，对象技术大师，SOMA 方法（Semantic Object Modelling Approach）的作者。他和 Brian Henderson-Sellers 创立的 OPEN 协会（Object-oriented Process, Environment, and Notation）在 90 年代中期曾经是 OMG/UML 的对抗者，并提出竞争方案：开放式建模语言（Open Modeling Language (OML)） 。 近来他主要关注 web 可用性。他的《面向对象方法原理与实践》（第 3 版有中译本）以及“Migrating To Object Technology”（未有中译本）是面向对象领域的经典名著之一。

Ian Graham 目前是 TriReme International Ltd 的董事，TriReme 为客户提供定制的 UML 培训服务。

Ian Graham 非常喜爱中国文化，修习中国武术已经超过 30 年，精通白眉、太极、少林、咏春等门派的武功。

*Ian Graham 讲座记录

*SC4 播放器

*BitTorrent 客户端下载

类似的讲座将持续举行，感兴趣者请留意 UMLChina 首页的公告。

MagicDraw UML 工具增强报表功能

[2004/3/15]

2004 年 3 月 15 日，No Magic 公司宣布，一个新的报表模块连同软件设计文档模板和用例规格说明模板出现在 MagicDraw UML 建模工具版本 7.5 的新特性中，这个版本将在 3 月 17 日发布。“你可以把规格说明放在 MagicDraw 中，然后产生 RTF 文档，就象出现在 RUP 或 PDF 中一样”，MagicDraw 产品经理 Saulius Kaukenas 说。

这个工具遵循 XML 元数据交换规范，但现在还没有支持 Kaukenas 希望的 UML2.0。在 OMG 讨论过程下，UML2.0 要到九月才会最终通过。

同样，新特性之一是针对开放源码开发者们的一个免费社区版本，包括 UML 类图，Kaukenas 说。

价格没有变化：个人版本，可以绘制，编辑和发布模型，价格是每份 149 美元。标准版本，增加了一个 UML 编辑器和报表功能，每个浮动许可价格是 699 美元。专业版，分别针对语言 Java， C++和 C#的不同版本，增加了一些反向工程功能，每个浮动许可 1299 美元。企业版，有代码生成，反向工程和针对 CORBA IDL， EJB， WSDL 和 XML schema 的图表功能，每个浮动许可 2299 美元。

公司还发布了一个该工具的 Windows 本地版本，替换了以前的 Java 版本。Kaukenas 解释说版本 7.2 确实允许从模型生成 C#代码，并可以从 C#和.NET 公共中间语言（CIL）字节码创建模型。

（自 sdtimes，Windy 摘译，不得转载用于商业用途）

视频游戏公司采用 SmartDraw 进行标准化

[2004/3/15]

SmartDraw.com，Internet 上第一个商业图形软件提供商，今天宣布 Sammy Studios 公司，一个互动娱乐内容的开发商，发行商和全球分销商，选择 SmartDraw 软件用于 UML 设计。

Sammy Studios 购买了 SmartDraw 软件，用于它在圣迪亚戈的游戏开发室和洛杉矶的办公室。

“我们本来打算采用另一种产品进行标准化的，但是它太贵了”，Sammy Studios 公司的游戏引擎和工具主管 Dave Wagner 说，“我们发现 SmartDraw 用来作 UML 设计非常棒，价格也只是（前者的）几分之一。”

Sammy Studios 希望它的第一线原创视频游戏能在今年的晚些时候上市。它也将分销第三方的产品，例如 Sony Playstation 2, Microsoft Xbox, Nintendo GameCube 等等。根据 Wagner 的说法，SmartDraw 将在游戏开发中扮演一种整合的角色。

SmartDraw 软件为用户提供了超过 5 万种符号，图像和模板，包括用于软件设计的一个完整符号集合，包括为 UML，ERD 和数据库，GUI，网络设计和其他符号提供的库。用户可以简单地把符号拖拽到页面，来建立专业风格的 UML 图。

“SmartDraw 很直观，快速，而且好用”，Wagner 说，“人人都喜欢它。”

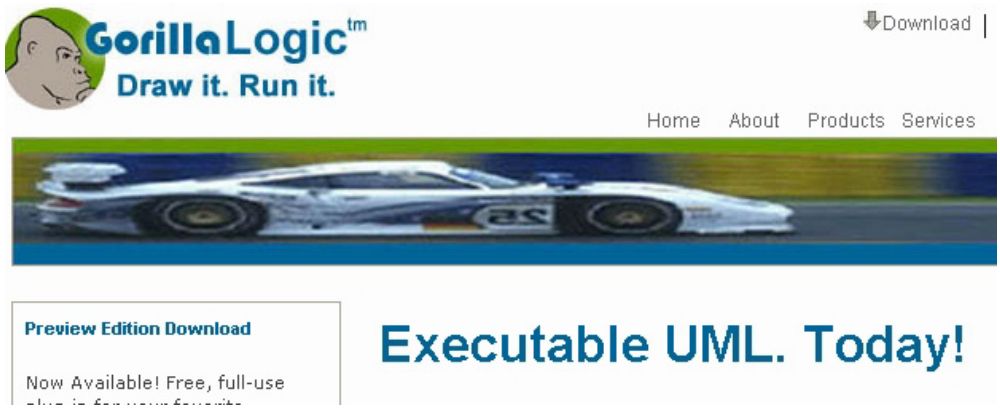
除了 UML 设计之外，Sammy Studios 还将用 SmartDraw 在人力资源部门制作组织图表和其他一些图。Sammy Studios 现在有 130 个员工，他们希望这个数字在（未来的）18 个月里面能翻一倍。

（自 thetsector，Windy 摘译，不得转载用于商业用途）

Gorilla Logic 发布直接执行 UML 业务模型的 Gorilla Execution Engine 预览版本

[2004/3/2]

Gorilla Logic 今天宣布其突破性技术的问世，该技术将帮助建模人员得到可执行的 UML 领域模型。现在，在一个项目的分析阶段，设计人员可以通过演示交互的 WEB 应用演示他们的领域模型。Gorilla eXecution Engine (GXE)的免费预览版本已经提供下载。



该产品的一个 beta 测试员，SUN 公司的高级 Java 架构师 Tim Seltzer 评价，“其中通过设计直接从用户那里得到反馈的能力给我留下了深刻的印象。”，Seltzer 补充说，“顾客可以很容易地在领域模型中验证实体以及相关规则之间的关系。”

Gorilla Logic 公司的创始人及 CEO，Stu Stern，认为，“我们相信 GXE Preview 将成为 UML 设计人员不可缺少的工具，以助于更好地可视化和验证他们的领域模型，能够为 UML 的实践者提供一种降低需求风险的途径，我们觉得非常兴奋。”

Gorilla Logic 正在申请专利的模型分析技术使得设计人员可以直接通过扩展的 UML，而只需很少或者不用编码，就能开发出更加全面和功能丰富的应用。Seltzer 认为，“Gorilla Logic 的技术有潜力提供高价值并且低学习成本的工具集，不仅是在软件生命周期的初始阶段，而且贯穿整个软件生命周期。”

关于 Gorilla Logic

Gorilla Logic 公司(www.gorillalogic.com) 提供简化 eBusiness 应用开发过程的工具、方法论以及服务。Gorilla eXecution Engine (GXE) 通过直接执行业务应用高级别的应用设计模型，实质上已经使得很多编码工作不再需要。Gorilla Logic 同样提供专业服务，帮助软件组织改进其软件工程过程，提供敏捷建模的咨询和培训。

（自 computer-software.press，袁峰 摘译，不得转载用于商业用途）

I-Logix：模型驱动开发理应有测试的一席之地

[2004/2/17]

系统设计和软件开发解决方案的提供商 I-Logix 推出其自动测试生成工具 Rhapsody Automatic Test Generator (ATG),将模型驱动的开发（extending Model-Driven Development，MDD）进行扩展，纳入了模型驱动的测试。I-Logix 声称这是嵌入式软件开发市场上第一个自动测试工具，开发人员可以从 UML 设计中自动生成测试用例。这项突破性的技术将保证系统的开发可以在各个迭代阶段以及最终的产品实现阶段维护设计的完整性、降低错误率并加快推向市场的速度。

Rhapsody ATG 对开发诸如符合 DO-178B 标准的安全关键应用（safety-critical applications）非常关键，它是 Rhapsody MDD 环境的自然扩展。当软件工程师使用 Rhapsody 时，他们可以图形化地定义系统并进行设计，以及执行和验证，并最终从模型中产生面向目标系统的全部的产品代码。

通过自动生成测试用例和在整个设计过程中执行这些用例，Rhapsody ATG 使得软件工程师们的开发过程又上了一个台阶。只要定义好被测试系统的范围－不管是单个类还是一族类－以及它们的接口，Rhapsody ATG 就可以自动地生成测试用例。

另外，Rhapsody ATG 还保证了对 UML 模型测试的高覆盖率。测试用例对状态、变化、方法调用以及模型中使用的事件进行了完全的覆盖。另外，Rhapsody ATG 还足够灵活，可以“读”用户定义的测试用例，分析对代码的覆盖率，并为那些没有覆盖到的部分自动生成测试用例。在宿主机上创建测试场景并在整个开发过程中运行它们，提供了自动化的回归测试。另外，尽早在目标环境中进行测试有利于发现在设计中无法发现的一些问题。

Rhapsody ATG 也可以用来和 I-Logix 的 Rhapsody TestConductor 结合以提供白盒测试的能力。通过执行这些测试用例，用户可以在测试用例运行中依照系统内部的交互，这样可以更容易发现错误。另外，用户可以将生成的测试用例输出到他们自己的测试环境中来进行黑盒测试。

“显然，测试应该是产品开发过程中不可分隔的一部分。行业分析员估计，测试将占开发总时间的 70% 以及总开发支出的 50%，可以看到，这里有很大一部分的成本是可以节省下来的。”市场和公司发展部的副总裁 Neeraj Chandra 认为，“Rhapsody ATG 可以被应用在整个开发周期，不仅可以测试开发人员负责的设计部分，而且还可以用来测试已经部署的应用，这将有助于提高产品的质量。”

I-Logix 的 Rhapsody 是用来解决嵌入式软件开发领域各种难题的基于 UML 的 MDD 环境。它允许嵌入式软件工程师可以可视化地分析、设计、实施及测试基于 UML 的应用。在设计进行中就可以得到高质量的产品代码，并可以得到图形化仿真（animation），这样设计的图就可以在开发的机器上进行调试工作。这个能力使得设计级别的重用成为可能，并压缩了整个开发的生命周期。Rhapsody 具有开放的架构，为其它领先的需求跟踪、配置管理以及测试工具提供了接口。

（自 ebizq，袁峰 摘译，不得转载用于商业用途）

征稿

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

UMLChina 辅助教材

EXPLORING REQUIREMENTS

QUALITY BEFORE DESIGN

BY DONALD C. GAUSE & GERALD M. WEINBERG

发现什么都不是

而发现的过程就是一切

——《探索需求》

中译本预订>>

我思 Argo 在

Alan O’Callaghan 著，车皓阳 译

查看评论

敏捷开发并不是一个新的概念。**Alan O’Callaghan** 采访了比利时的 Martine Devos，他是 Argo 项目的幕后精英。

近来，在贸易出版物的头版和专业会议当中，敏捷过程都非常火。它的基本思想是，软件开发过程自身应该灵活、适应、轻量。它与 Jim Highsmith 和 Kent Beck（XP 运动的领导者）等几个曝光率甚高的美国人有着莫大的关系。但是欧洲拥有自己的专家，敏捷过程并不象有些人想得那样新颖。我与 Martine Devos[1]邂逅，她曾因为对轻量级方法出色的阐述，在大西洋两岸，广为人知，受到众人的尊敬。这次采访过程跨度很大，从她在比利时先锋项目 Argo 中担任的领导角色，一直谈到她把敏捷方式的统一过程引入到英国的开发机构内部，在此之中 SCRUM 开发过程对她产生影响的重要程度。

定义 Argo

Alan：Martine，我们现在已经认识一段时间了。我记得在 1995 年牛津召开的对象技术会议上，你曾经谈到过比利时的项目 Argo。追忆往事，很明显它是现如今被称作敏捷过程的原型。你能回忆一下 Argo 项目是干嘛的吗？

Martine：Argo 是一个半政府性质的组织，它管理比利时佛莱芒团体中的公立的，非教派的学校。它由一个中央管理机构和几百个半自治的本地团体组成，这些本地团体包括董事会和学校。

这个项目实际上在 1994 年就已经启动。那时 Argo 正在经历外部审计，用来仿真在手续、组织结构、责任规定、职责、委托中最大变化发生的可能性。我们已经有了一个预期未来变化的清单，甚至还加入了其它一些没有什么人能猜到的内容。Argo 拥有一名来自五大银行的顾问，他曾经定义过超过 400 个独立的应用程序，这些都是 Argo 需要的。这些应用的需求将会审慎地签署。

当我们接管的时候，觉得如此复杂、野心如此之大的项目如果再按照传统的方式来创建，无疑是愚蠢的。我们要使用架构方法，当设计问题出现的时候，它可以给予足够的变化空间。

我们选择了一个框架来构建应用，它足够灵活，可以处理变化的或新出现的需求，功能的和技术的，从外部加入的或是项目内部适用的。从一开始，我们就想构建出一种环境，它不仅可以在重新组织以及分布式的条件下生存，而且还可以激励用户重新考虑工作方式，思考完成事情更好的作法。

开始

Alan：那么你们是怎么开始的呢？

Marine：我们定下目标开发框架，它是通用组件的微型内核，可以动态作用于知识库。要尽可能地避免硬编码。

我们以三个试飞应用开始，它们为最初的框架提供输入：

- 文档中心，提供对数据和文档的灵活的搜索和检索手段，例如法律文档和议会法令。
- 中央董事会决策支持系统，从最初的草案到最终文本，监控它们在学校和本地董事会的实现情况。
- 一个应用程序，用于导入、检索、存储、路由本地学校和其它相关单位发送过来的大量文档。

项目进行到一半，执行了待解决的重组织问题。尽管变化的性质和程度在项目开始之时基本上都是未知的，但无需多余编码，通过重新配置就能适应现有应用。

项目继续进行。自此又交付了好几种新的应用。框架也扩展了，允许学校通过 Internet 访问中央数据和信息。我们增加了一个 Internet 讨论系统。

应用本身，与新的技术可能性一起，是演化框架需求的主要来源。我们不仅需要使用数据、文档、工作流、Internet 功能来创建最终用户的应用，而且还需要使用同样的框架开发我们使用的工具，来创建和管理这些应用。这样用户就可以使用他们的接口自己来适应应用。

Argo 的最终用户逐渐从开发小组那里接管了应用和配置管理。关键用户指导并编写了部分文档，同时用户小组组织研讨会来讨论如何使技术变得更加有效。最初，这个过程由上进心很强的先锋用户所组成的小型团队来推动，但现在它已经开始充分拥抱用户团体了。用户在开始时都是一些高度的计算机文盲，这实际上是巨大的成功。在其它地方，我很少见到这种情况。渐渐地，知识库中构建并存储了完整的规范，这些应用可以立即执行使用。这样我们就弥补了规范、开发、应用程序使用之间的缺陷。通过这种方式，我们就能够帮助用户思考、重新思考并重新设计这些过程。

Argo 中的经验教训[2]

Alan：正如你所说的那样，Argo 曾经是，而且现在也是先于其时代的。尽管如此，肯定还是会总结出许多经验教训。您能说一说包括有哪些吗？

Martine：我们有三种意见和许许多多小意见。主要的三种意见包括：

- 第一，用户趋向于按照当前他们的工作状况来描述需求。对于他们来说，预测可能性和抉择是很困难的，因为他们并不了解最新的技术。他们要能通过尝试来学习以及做出调整，这需要大量的原型工作。
- 第二，对于一个机构来说，业务模型在某个给定的时间点上可能会很特殊，但诸如数据项、查询、报表、文档管理和路由等一些功能需求变得更加通用了。这种理解导致要确认一些通用组件。
- 第三，独立于实际的业务模型，功能扩展常常会做进可重用资产。

分离组件

Alan：听上去你好象说的是，可以通过把更加稳定，或者说是更加通用的组件从容易发生变化且特定的组件中剥离出来，以达到系统中的灵活性。但你不能猜测，在大型项目的起点就 100%的正确。你是怎么做的？

Martine：我们从一开始就与客户，也就是教育部长商谈采用一套不同的方法，我们知道因为亟待进行的重组问题需求肯定会发生变化。而且由于对计算机的无知，用户并不知道想要求什么，能得到什么。我们寻找一种“敏捷的方法”。

是XP吗？不。极限编程当时还不存在，但我们在两年之后与Ron Jeffreys、Martin Fowler 和Kent Beck一起参加了OOPSLA的第一届pre-XP研讨会。在那里我们讲述了我们的方法，因为它们有一些相似之处。那是SCRUM吗？也不是。但我在PLOPD4[3]中写出了我们使用SCRUM的实践。

是RUP吗？一点也不。它是我们自己的方法，裁剪之后可以适应项目关系人（stakeholder）的需求，有部长（支付费用并期待金钱产生的价值），来自不同部门的用户（他们非常怀疑系统将如何改变他们的工作和生活的方式），开发者（要完成有趣而富有挑战性的工作）和管理人员。

Alan：那么它就是你们自己的方法，很明显，它超前于我们今天了解的敏捷过程。Argo 影响了这些后续的开发吗？

使用敏捷过程

Martine：是的。我们面临一个很实际的问题。在一个非常大的机构里，没有一种工作方法可以在每个人身上都适用。事情实际上都要依赖于正在完成工作的人。有一个部门正在完成伟大的工作。它拥有使用自己工作区的规则簿，这就使得它能做出出色的工作。我们需要一种支持人们真正工作的方法。这就是我们为什么要装配自己的敏捷方法的原因。

我们有一个掌控委员会。它包括所有的决策制定者和来自于各种客户和开发人员的代表。总设计师担任掌控委员会委员。每月一次召开的例会，变成了时间盒（time box）的最后期限。在这些时间盒中，我们使用了以后被称为 SCRUM 的原则。

Alan：告诉我们关于这些原则的事情。

Martine：在最初的时候，你与项目关系人进行头脑风暴，积压了一堆任务。这与收集 story 的 XP 技术大致是等价的。它的目标是在索引卡上或图中描述事物。巧合的是，它包含功能需求、非功能操作需求，等等。使用 XP 你是否可以如我们那般深入，这一点并不是完全清楚的。既然它就是我们创建的东西，最终用户应该能够马上买进。

积压中的不同任务都有优先级，并被分成 Sprints。在 Argo 中，一个 Sprint 就是一个月的时间盒。每个 Sprint 的需求一旦开始就被冻结，开发者可以自由计划、设计、实现和测试，没有任何不必要的间断。在 Sprint 最后，有一个 after-Sprint 展示，描述实现的内容。

这些展示很关键。它们建立信任。你要给用户出示真正的进度，而不只是一堆文档，用户可以自由地建议做出改变和增加需求。然后，掌控委员会就能决定要如何对付这些变化，做出其它的修改，并把它们包含进下一个 Sprint 中。

SCRUM[4]过程按照控制 Sprint 的每日会议来命名。会议按照橄榄球队命名，球员聚集在一起协力将球转移到底线。每一个有时间的人，包括用户，都会参加 Scrum 会议。

在会议当中，我们有所谓的猪和鸡的概念。猪，包括开发人员，他们来驱动会议，完成大部分讲话，因为最为得失攸关的就是他们。而鸡，或者说管理人员，他们在那里提供支持。团队的每一个成员都会给一分钟来做进度报告和未来的工作计划报告。他或她应该要把任何亟待解决的问题都明确地提出来，然后鸡要找出办法解决它们，如果有必要可以离线进行。信任是至关重要的。如果问题隐藏起来或者没有提出来，SCRUM 就不会正常工作。

Alan：你今天回想一下，哪些原则是与今天的 XP 相关的？

Marine：我们使用了大量的你今天可以在 XP 中见到的技术。列出如下：

- 结对编程。
- 单元测试。
- 日创建和集成。
- 作为验证测试使用的，或作为帮助文档输入的 Stories。
- 不要加班。这很正常。毕竟这里是比利时，而不是美国。
- 愿景（Vision）。

愿景在 Argo 项目中非常重要。我们开发了一种方法，其中所有的项目关系人都在一起工作。这真的会帮助我们克服很多阻力，政治方面的。面对一个新的系统，用户经常会感到恐惧。

采取主动使用隐喻和游戏的方法，我们营造出一种氛围，可以很安全地讨论问题，感受不到恐惧。而且，经常有这样的现象发生，他们会发现他们有着相同的问题。这给了他们一起工作的动机和信任。

使用模式

Alan：你曾经提到项目自1994年就已经开始了。巧合的是，GOF于同年写出《设计模式》[5]一书。你也是模式运动的领导，并且曾经担任EuroFlop[6]编程委员会的主席。Argo中使用模式了吗？

Martine：模式浑然不觉地就这样出现了。我们的确大量使用了GOF的设计模式一书。每一个新的小组成员都会收到一份拷贝。一些设计模式节约了我们大量的时间，实际上，它们使得整个方法成为可能。重构，甚至抛弃大量代码是这里的规则，而不是个别现象。

但我们不能只使用设计模式。我们意识到我们使用了Jim Coplien的组织模式（organisation pattern）[7]，并且决定要更多地试用它们。因此我们在某个周末离开，寻找遗漏的内容，试图改进组织结构。我们使用的关键的模式有：

- 设计者也参与实现，我们需要知道材料的潜在价值和弱点，这样设计者就不得不自己来使用它。他们删剪代码。
- 学徒，与设计者共享办公室。
- 结对编程。

Alan：回顾往事，你是否曾经使用过今天出版的敏捷方法中的内容？

Martine：如果问题是“我们是敏捷的吗？”，回答当然是yes。但如果你问的是我们是否使用过某种专有的方法，那么答案就是no。实际上，我宁肯使用下面的语言来描述我们所做的事情，我们仍然在做的事情，我们使用所谓敏捷方法在做的事情，“够用足已”。当请求发生时一些元素变得更加精确。例如，白板绘图有时会变成硬拷贝。但当事情变得清晰时，我们不会为了它而进行绘图。

有时我们做出的绘图永远也不会有任何形式的用途。当需要它们的时候，我们就发明了自己的方法。例如，我们的“蜘蛛”在用户面前变得臭名昭彰。在蜘蛛图中，每个用户都被表示成私有的蜘蛛实体。每个用户都要绘制到其他用户的输出，他收到的、需要的和使用的来自其他用户的输入。这些蜘蛛使我们能够消除组织中大量的负荷。尽管如此，它们永远也不会离开白板变成计算机图形。它们服务完自己的目标，然后消失。

RUP

Alan：在此之后，你曾经为 EDS 工作过一段时间。它是世界上第二大的计算公司吗？你现在还是独立顾问吧。最近，你帮助英国的一家公司克服了“过程瘫痪”。我听说你还帮助它设计了一套自己的敏捷版本的 RUP。我现在对此持有极大的兴趣。你可能知道，我和 Ivar Jacobson 在这个栏目中也曾辩论过两个问题。我说 RUP 是重量级的过程。你到底是怎样应用它的？

Martine：我并不知道使 RUP 敏捷化会有什么困难。有很多关于统一过程的误解。在 CD 或在网站上，可能它看上去会很重，那样做会是一件非常愚蠢的事情。但使用一套类似蜘蛛的方法来做，就完全有可能。也就是说，与项目关系人一起组织开研讨会，进行头脑风暴，讨论来自他人的输入，产生的输出，找出是否真正使用了文档。结果很可能会是，我们以一套统一过程的版本结束，它比我们以前实验过的内容更加理性。我这么说的意思是，更少的负荷，更少激励进度的纸介文档，或者是反而更加真实的进度。

如你所说，我最近与一家英国公司合作。那里有两个阵营，一个想要使用统一过程，另一个则担心使用 UP 会需要他们完成多余的工作，产生更多的文档。因此，我把他们召集到一起，帮助他们发现过程中每个人需要完成的工作。一旦他们的注意力集中在需要完成的事情上，和平就会出现。战争结束了，瘫痪的项目重新开始作用。RUP 给了我们帮助和在一些准则之上工作的框架。我们会正确使用它，或者误用。

Alan：好的，在讨论出版之后，从我收到的邮件来看，我的一些读者将会很高兴听到你这么说。我的一些专栏作家，也会加入进来。但我还不能完全确定，客户最后得到是这样一个过程，一个理智的销售人员会认可的过程。我与你的想法是一致的，我真是太喜欢你的工作了，你安排人员的方式，在开发最核心的地方他们进行的交流。它看上去就象是常识一样，但政治因素和神话经常会蒙蔽开发团队，或者说，他们的管理者，这样他们就需要象你这样真正有经验的协调人员来帮助看一看到底发生了什么。

多谢你能抽出时间与我交谈，Martine。看到并不是所有的敏捷大师们都在美国，这真是令人兴奋，说实话，和他们在一起并不是那么有趣。严肃地说，看到第一个敏捷项目成功的故事是发生在欧洲，这很重要。到目前为止，它可能仍然是敏捷运动中最大的成功。我确信，肯定会有许多开发团队乐于学习你的开发经验。再次表示感谢。Mine's a Guinness。

参考文献

1. Martine Devos的邮件地址是**devos.martine@pandora.be**

2. 在M Devos和M Tilman的下面的文章中全面描述了Argo项目, *A Reflective and Repository Based Framework*, in ME Fayed, DC Schmidt and RE Johnson (eds). *Implementing Application Frameworks*. John Wiley and Sons, 1999

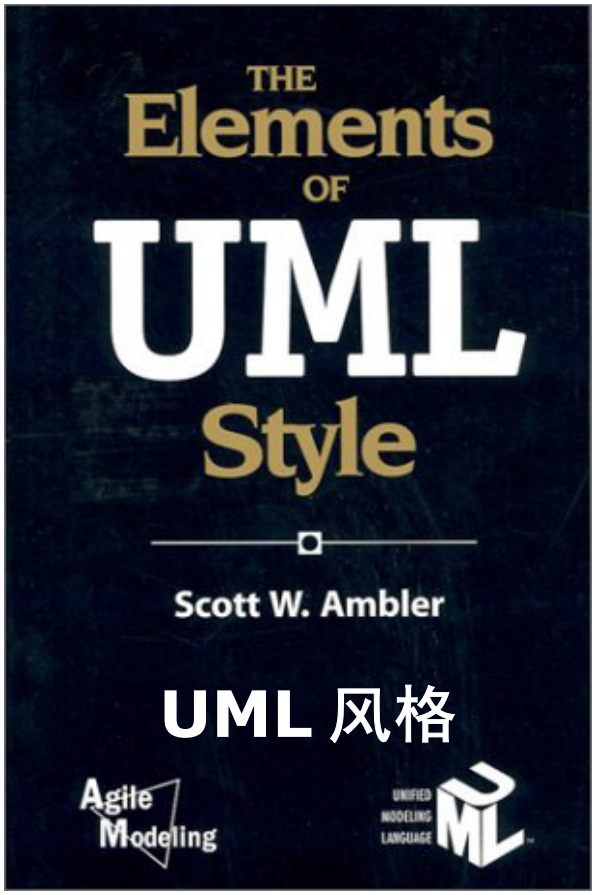
3. N Harrison, B Foote and H Rohnert (eds), *Pattern Languages of Program Design volume 4*. Addison Wesley, 1999

4. M Beedle M, M Devos, Y Sharon, K Schwaber and J Sutherland, *SCRUM: A Pattern Language for Hyperproductive Software Development* in N Harrison, B Foote and H Rohnert (eds), *Pattern Languages of Program Design 4*, Addison Wesley, 1999.

5. E Gamma, R Helm, R Johnson and J Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995

6. 想要查阅EuroPlop 2002的详情，请去**www.hillside.net/patterns/EuroPLoP/**

7. JO Coplien, A Generative Development Process Pattern Language in JO Coplien and DC Schmidt (eds), Pattern Languages of Program Design, Addison Wesley, 1995.



UML 风格

口袋里的 UML

当企业计算遭遇嵌入式系统

Michael Kircher、Christa Schwanninger 著，车皓阳 译

查看评论

摘要

在这份报告里，我们简要回顾了在约束环境中扮演复用角色的现有技术和概念。我们从复用的角度开始，紧接着讨论了嵌入式系统中的典型限制，最后讨论了当前在约束环境支持下复用功能的技术和概念，并以此作为总结。

关键词

复用，模式，框架，组件，面向方面编程

1 简介

嵌入式设备中软件使用得越来越多，系统之间建立了连接，对在小型设备中增加功能的要求也一直在增长，这些都迫使供应商想尽办法来缩短开发周期。对于嵌入式系统来说，只有实现了复用，软件需求才能满足，至少在上世纪八十年代我们就已经开始为此而付出努力了。我们能否做到简单地把概念从企业软件转移到嵌入式系统中吗？这是可以做到的，只要我们注意约束条件就行了。

2 复用

可以以下面两种方式应用复用的概念：

- 架构，架构复用意味着要复用分离的职责结构和它们交互的模式，例如，三层架构。
- 源代码，源代码复用包括库和生成代码。通过剪切和粘贴来复用，通常是不会尽如人意的。

借助下面这些熟知的概念，在不同组合当中都可以支持这两种方式的复用：

- 架构模式仅在架构层次上支持复用。架构模式中最突出的例子是分层模式（Layers pattern）[POSA1]，它在每种协议栈中都有应用，还有微内核模式（Microkernel pattern）[POSA1]，比如说它会在嵌入式操作系统和中间件产品中实现。
- 框架支持架构和源码级的复用。在应用程序上强制运行某种控制流，它们就能对架构产生影响。充当 GUI 框架的 AWT 和网络编程工具包 ACE[Schm03]，都是典型的例子。
- 封装应用程序功能的组件，一般会以库的形式进行实现和复用，但不要强加任何架构方面的内容。为了使组件独立于应用基础架构，潜在地，基础架构服务会在组件中被提取出来。在企业系统中，组件容器实现了这些基础架构服务。但在嵌入式系统中，象 J2EE 或 CORBA[OMG03]这样的组件技术太笨重了。相反，定制的解决方案，领域专用语言会被使用起来；请参阅[Contribution by M. Voelter]。
- 另一种代码复用的方法是划分相关功能分组的库，例如数学函数或操作系统包装外观[POSA2]。

面向方面编程（AOP）[Kicz97]正交于这些概念。职责横切诸如函数或对象之类的模块化制品，AOP 支持职责的封装。这种附加的模块化功能也支持复用，因为只有避免被特定于某种上下文环境的功能“污染”的那些组件才可以做到不加变化地复用。在 AOP 中，术语是“纠缠”，它意味着在一段代码之中实现好几种关注点。

另一方面，既然面向方面允许在单个模块中实现关注点横切，关注点横切的实现就能被复用。

在预编译、装载和运行的过程中，方面与基础代码交织起来。它们使得独立于应用的组件开发成为可能，正如组件模型，无须手工更改组件代码就可以增加基础架构服务。

3 约束环境

这次讨论会的焦点主要与约束环境的定义相关。软件系统的典型约束有：

- 内存印记（footprint）。批量制造设备会节省内存，因为硬件上省下的每一分钱都会赚到上百万资金。
- CPU 周期。CPU 循环周期是稀缺资源，因为 CPU 越快，产生的热量就会越大。
- 通信带宽。在嵌入设备中通信意味着冗余和高可用，这将会导致通信带宽的限制。
- 时间需求。任务和安全关键的应用会严格强制时间相关的需求。
- 强壮性。嵌入设备通常必须要可靠地工作在苛刻的条件下。

约束一般会是非功能需求（不考虑硬实时），大多情况是出于有限可用资源和 QoS 方面的考虑。

所有的环境都必须能够处理约束。企业系统主要需要应对不同层次上的扩展性问题，嵌入式系统会大大受到紧急资源可用性的影响。在当前的软件开发项目中，嵌入式系统已逐渐开始使用企业技术，例如面向对象和组件技术等，企业系统也开始使用模型驱动的架构方法——这在嵌入式系统开发中已有很长的历史。

4 解决方案

由于这次研究会的目的，我们把讨论限制在嵌入式系统上，与企业系统相比，它们一般会有更多的约束。

现在我们回头看第二节中的列表，在嵌入式系统中下面的概念支持复用：

- 架构可以不加限制地复用。前面提到的层和微内核模式适应良好。
- 对于嵌入系统来说，框架需要高度定制，以防无用功能消耗内存。诸如 Strategy[GOF]、Bridge[GOF]和 Visitor[GOF]等设计模式有助于增进适用性和灵活性设计。对于有效复用来说，两者都是需要的。
- 组件可以当成概念使用，但资源集中的容器框架相对来说太过笨重。代码生成有助于将容器功能减少到适用的程度。
- AOP 可以作为技术使用，来实现定制组件模型。横切容器功能必须按方面实现，编织器从功能组件和方面入手生成应用程序。

✠ 资源管理模式[POSA3]在系统运行时处理系统资源的复用。它们描述何时获取何时释放系统资源，如内存、线程、通信连接，等等。

5 结论

在本文中我们简要地介绍了当前支持复用的一些概念，这些概念并不是全部都可以在约束环境中使用。总的来说，增加运行时费用或内存印记的那些技术并不会被使用。但是，象组件模型与 AOP 这样的企业技术却正在渗透嵌入式领域。

6 参考文献

[GoF95]E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns-Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995

[Kicz97]G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda and C. Lopes, J.-M. Loingtier, and J. Irwin. Aspect Oriented Programming. In Proc. of European Conference on Object-Oriented Programming (ECOOP), Lecture Notes in Computer Science Vol. 1241, pp. 220-242, 1997.

[OMG03]Object Management Group, CORBA Component Model Specification, http://www.omg.org/technology, 2003

[POSA1]F. Buschmann, R. Meunier, H. Rohnert, P. Sommerland, and M. Stal, Pattern-Oriented Software Architecture-A System of Patterns, John Wiley and Sons, 1996

[POSA2]D. Schmidt, M. Stal, H. Rohnert, and F. Buschmann, Pattern-Oriented Software Architecture-Patterns for Concurrent and Distributed Objects, John Wiley and Sons, 2000

[POSA3]M. Kircher, P. Jain, Pattern-Oriented Software Architecture-Resource Management and Optimizations, John Wiley and Sons, 2004

[Schm03]D. C. Schmidt, Adaptive Communication Environment (ACE), http://www.cs.wustl.edu/~schmidt/ACE.html, 2003

WILEY

TIMELY. PRACTICAL. RELIABLE.

UMLChina 指定教材

Agile Database Techniques

Effective Strategies for the Agile Software Developer

《敏捷数据》

UMLChina 李巍 译

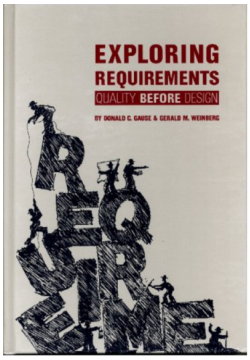
Scott Ambler

探索《探索需求》

think 著

查看评论

有价值的思想，不会凋谢。



【原文】

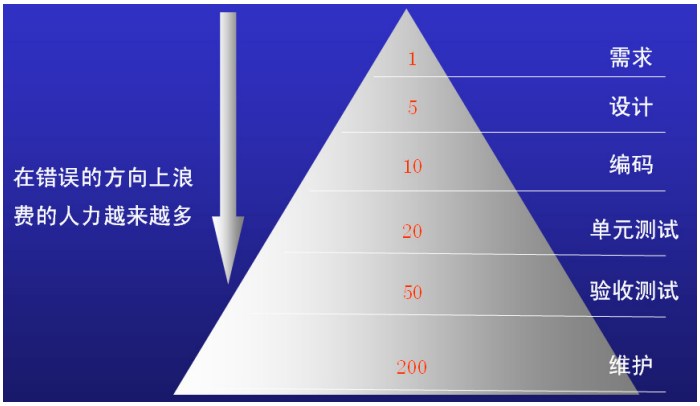
所属章节：序言

软件业——一个倍受这种毫无根据的白日梦折磨的行业。我们乐意引用约翰·冯·诺依曼的话，因为我们的许多客户认为他是软件之父。他们关注到他的名言："如果你根本不知道自己在讨论什么，那么对其强求精确是毫无意义的。"

如果人们不知道他们想要什么，那么无论什么开发过程——不管有多么精确、多么聪明或多么有效率——都不能使他们满意。这就是为什么我们要做需求工作的缘故——只有如此我们才不会设计出人们不需要的系统。

【探索】

何止是毫无意义，简直是巨大的浪费。需求一旦出问题，越往后面做，就意味着在错误的方向上浪费的人力物力越来越多。如果有上帝在天上看到这一切，他应该会为我们这群蚂蚁而叹息吧？



正因如此，需求环节是软件公司最值得改进的环节。需求环节不解决，就象拿着好枪（面向对象分析设计）却打不中目标一样，旁观者首先怀疑的是你的枪不好、你的射击技术不好，而不会首先怀疑你的目标问题。不改善需求，软件开发的结果自然不会好，后续的改进无从谈起。

【原文】

所属章节：序言

美国第 34 任总统艾森豪威尔上将曾经说过，“计划本身什么都不是，而编制计划的过程就是一切”。我们认同这样的说法，并把它推广到需求过程：

产品什么都不是，而开发的过程就是一切。

发现什么都不是，而发现过程（探索过程）就是一切。

这就解释了我们的书名：探索需求。

【探索】

采用“用例”来代入——用例什么都不是，而探索用例的过程就是一切。

用例的发掘过程比结果还要重要。它“掀开了地毯”（引自《人月神话》），迫使你去思考“系统到底是什么”的问题。在寻找执行者的过程中，迫使你去思考软件系统的边界；在寻找用例的过程中，迫使你去思考软件的价值；在书写用例文档的过程中，迫使你去思考哪些是需求（即问题），哪些不是需求而是设计（即解决方案）。

如果你有了这个过程，获得了成果。成果用何种方式表达出来，是不是按照严格的模板，并非那么重要。

【原文】

所属章节：第 1 章

万无一失蟑螂杀灭仪。全部操作步骤为：把蟑螂放在木块 A 上，然后用木块 B 拍它。

以我们的经验，“分析和设计工作平台”类似于“万无一失蟑螂杀灭仪”里的木块 A。“代码字典”类似于木块 B。

本书就是关于你的那部分工作：将蟑螂放于木块上，并在你突然拍打之前让其保持立正。

不管是蟑螂还是需求，难点在于抓住它们并使其立正。与前两个步骤相比，不同的是生成代码和清理被压碎的蟑螂尸体是琐事。

【探索】

开发人员问，“使用用例技术，要思考的东西比以前多了很多，会不会影响进度？”还有开发人员质疑迭代式开发，“这不是自找麻烦吗？按照我们以前的方式，需求，分析，设计，编码，一切按计划进行，不用修改”。——如果以“可运行”作为指标，这些质疑确实有道理，甚至这些表面功夫都可以不用，把微软或 Sun 的样例改改，改成你的系统的名字，就 OK 了。

超越灭蟑仪，就是超越“可运行”的目标，朝着“可用”的目标进发。

比起故障频频的软件，鸡肋软件（可稳定运行，但无价值）的危害更大。用也不是，不用也不是。每天上班 8 小时对着这个东西，实在是对 business worker 的一种摧残。

中国也有和书中类似的笑话：

“你的万灵虱子药怎么用啊？”

“先把虱子捉住，然后抹在它嘴上”

【原文】

所属章节：第 1 章

这就是为什么我们认为，即使你使用了万无一失的 CASE 和 CAD 工具，你还是需要本书所讲的工具。不仅如此，你的 CASE 工具越好，你就越是需要本书所讲的工具。

【探索】

开发人员问，“开发平台提供的框架已经很完善，大部分工作已经做了，我不知道该做什么？”

人做人的事，工具做工具的事。

工具使人解放。工具帮助你从琐事中解脱出来，把更多的精力放在核心问题上。数学公式之于数学家，五线谱之于音乐家，UML 之于软件开发人员，都是如此。模型转换、代码生成，现在又要朝 MDA 进发。它们不能解决和人相关的问题（如：需求获取…）。但我们有理由希望它们能帮助我们解决除此之外的各种问题，即书中的“清理被压碎的蟑螂尸体”等琐事。

你的工具好，恭喜你，同时也要提醒你，新层次的挑战来了。以前可以抱怨工具的理由如今已经不存在，你需要更加集中精力去面对真正的问题。——这也就是 Weinberg 说的“你的 CASE 工具越好，你就越是需要本书所讲的工具。”——更贴切地说：你的工具越好，本书的工具在你的软件开发工作中所占的比重越大。

（待续……）

The Addison Wesley Signature Series

A MARTIN FOWLER SIGNATURE BOOK

PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE

中译本即将上市

MARTIN FOWLER

WITH CONTRIBUTIONS BY
DAVID RICE,
MATTHEW FOEMMEL,
EDWARD HIEATT,
ROBERT MEE, AND
RANDY STAFFORD

UMLChina 负责审校，并指定为训练用教材

Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

UMLChina 指定教材

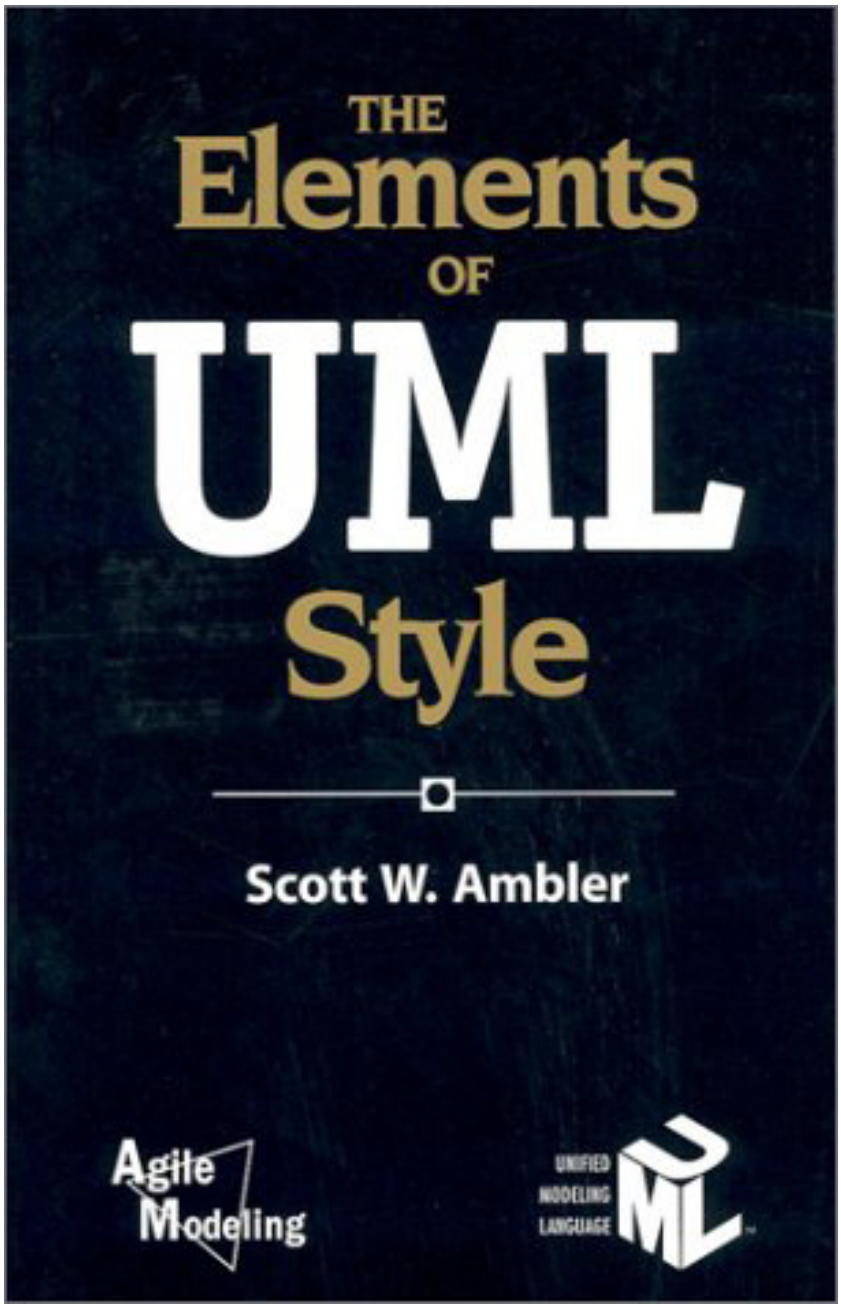
清华大学出版社

《UML 风格》中译本节选

Scott W. Ambler 著，王少锋 译

查看评论

本书中译本即将发行，由清华大学出版社闻洁编辑室制作，闻洁编辑室曾经为中国读者引进了《人月神话》、《人件》、《最后期限》、《探索需求》、《与熊共舞》等书。



点睛——《UML 风格》推荐序

Think

这是一本特别的小册子。

它不讲概念，它假设您已经懂了概念。它简单地告诉您“在递归关联上指明角色名”，不告诉您什么叫“递归”、“关联”和“角色”。这方面的知识，可以去看“UML Distilled”。

它不讲过程，它假设您已经了解某种开发过程。UML 提供了 9 种图，Ambler 根据自己的实践为各种图提出了一些准则。这些准则的使用，不受某种开发过程的约束。想了解更多这方面的知识，可以去看 Ambler 的“The Object Primer”和“Agile Modeling”。

它不讲工具，它假设您已经了解某种工具。Ambler 提出的这些准则，适用于各种 UML 工具，包括白纸和铅笔。如果想学习某种主流工具，可以去看“……从入门到精通”之类的书。

由此可见，这本书有别于以前出版过的 UML 书籍。它在您已经了解方法、过程和工具的基础上，告诉您在绘制 UML 图时需要注意的一些细节。

UML 实现了模型可视化表示的统一，大大有助于开发团队内部、开发团队与客户之间的交流。本书则又向前迈出了一步，致力于帮助开发团队寻找并遵循一种良好的 UML 绘图实践，实现 UML 绘图风格的统一——整齐漂亮的图纸和杂乱肮脏的图纸在团队的内部交流和给客户的印象上将会有天壤之别。在这本小册子中，Ambler 提出了 200 多条准则，它们将会在您画龙的同时，帮助您点上龙的眼睛。

（http://www.umlchina.com）向您推荐本书，并将之作为 UMLChina 训练的辅助教材。

第 6 章 UML 顺序图

和协作图一样，UML 顺序图是一种动态建模技术。UML 顺序图一般用于以下场合：

- 验证和充实使用场景（usage scenario）的逻辑和完整性。使用场景正如其名字所说明的，是对系统可能被使用的方式的描述。使用场景的逻辑可能是用例的一部分（也可能是备选的路径）；或者是完整的贯穿用例的路径，例如以动作的基本路径形式、以动作的部分基本路径加上一个或多个备选的场景等等形式所描述的逻辑；或者是贯穿于几个用例的逻辑，例如，“当一个学生注册入一所大学后立刻注册 3 个研讨班”这样的逻辑。

- 探究设计问题，因为顺序图提供了一种可视化方式来逐步调用类中所定义的操作。

- 让建模人员感觉到系统中哪些类将会是复杂的，这是个迹象，表明建模人员可能需要为这些类画状态图。

- 检测面向对象设计中的瓶颈。通过查看哪些消息被发送到一个对象，以及被调用的方法大概需要多长时间运行，建模人员很快就会了解哪里的设计需要改变，以便在系统中均衡分布负载。一般地，建模人员仍然需要用性能评测工具（profiling tool）来收集遥测数据（telemetry data）（译者注——遥测数据（telemetry data），这是一种直译，其实就是运行性能评测工具后，由计算机自动记录下来的数据。），以便发现瓶颈的准确位置。

6.1 一般性准则

125. 尽量从左至右排列消息

顺序图中的消息流从左上角开始，出现在图中较低位置的消息要比出现在较高位置的消息迟发送。因为在西方文化中，人们是从左至右阅读的，因此应该尽量在图的顶部从左至右排列类元（classifier）（如参与者、类、对象和用例）（译者注——这里是举例说明类元的例子，如参与者、类、对象和用例等。原文这里及后面多处认为在顺序图的顶部放置的是类元（classifier），但根据 UML 的规范说明，在顺序图的顶部放置的是类元角色（classifier role）而不是类元。），以便可以从左至右描绘消息流。在图 21 中，可以看到类元正是按这种方式排列的，如果 Seminar 对象被放在控制器（controller）的左边，那么就不能从左至右描绘消息流向了。有时候不可能让所有的消息流向都是从左至右，例如当两个对象之间彼此调用操作的时候，这种情况很常见。

126．对类元分层

分层是常见的面向对象设计方法。把系统按用户界面层、处理/控制（process/controller）层、业务层、持久（persistence）层和系统层进行组织非常常见（Ambler 2001）。当系统按这种方式设计时，通常每层中的类元相互之间会紧密协作，而和其他层的类元相对会低耦合。按类似的方式对顺序图进行分层是有意义的，一种分层的方法就是从最上层（比如用户界面层）开始，把最上层放在左边，越向右边越接近最底层。按这种方式对顺序图分层通常会使它们更易于阅读，也更容易发现分层的逻辑问题。图 21 就采用了这种分层方法。

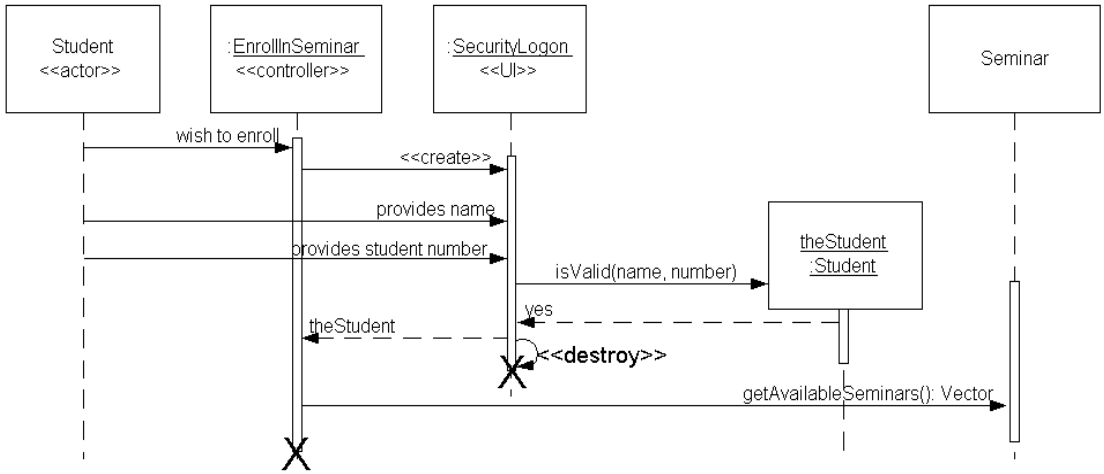


图 21 学生注册研讨班课程（译者注—— 原著中图 21 中的 Student、Seminar 都是类，应改为匿名对象较好。）

127．如果必要的话，可以用和类名一样的名字命名参与者

在图 21 中，可以看到有一个参与者和一个类的名字都是 Student。这是完全可以的，因为这两个类元表示的是两个不同的概念：参与者表示的是现实世界中的学生，而类表示的是要构造的业务应用中的学生。

128．包含逻辑的平铺直叙式的描述

图 21 可能很难理解，因为它很接近于实际的源代码，特别是对那些不熟悉阅读顺序图的人来说。在建模时在页左边的空白中包含逻辑的业务描述非常常见，就像在图 22 中所看到的，特别是当顺序图是描绘使用场景（usage scenario）时更是这样。这样增加了图的可理解性，就像 Rosenberg 和 Scott（1999）所指出的，这样也提供了追踪用例和顺序图之间关系的有价值信息。

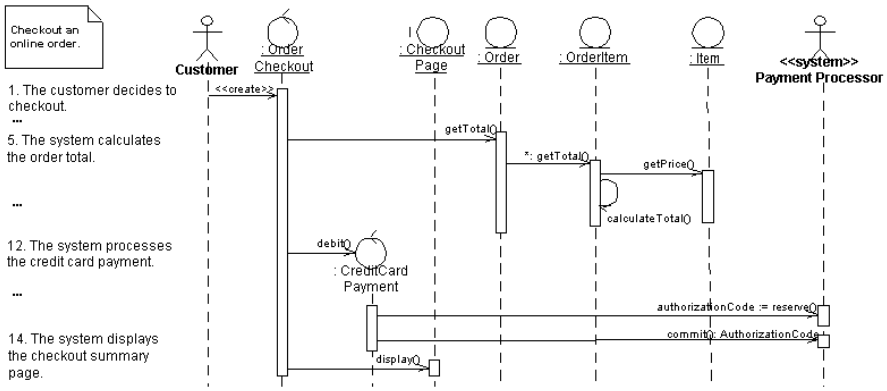


图 22 在线订单结帐 （？？？这个图中少个类，用书上的图，书上图中的 OrderCheckout 类是控制类，符号有错）

129．把先导型系统参与者放在图的最左边

先导型（proactive）系统参与者——即发起交互过程的参与者——经常是所要建模的焦点。对于业务应用来说，大部分使用场景的主要参与者是人或机构，他们是所要建模的场景的发起者。

130．把反应型系统参与者放在图的最右边

反应型（reactive）系统参与者——需要与之交互的外部系统——应该放在图的最右边，因为对于大部分的业务应用来说，这些参与者是作为后端实体（back-end entities）对待的，即系统是通过像 C 语言的应用程序接口（API）、CORBA 的接口定义语言（IDL）、消息队列或 Web 服务等这些存取技术来和反应型参与者交互的。换句话说，把这些后端系统放在图的后端。

131．避免建模对象的撤销

尽管内存管理是很重要的问题，特别是像在适当的时候从内存中删除对象这样的问题，但很多建模人员选择不在顺序图中对对象的撤销（destruction）建模（对象的撤销是用激活期竖条的底部加“X”符号或用消息的<<destroy>>版型表示）。比较图 21 和图 22，注意到图 21 中引入了对象的撤销，结果使图显得很混乱，但又没有带来明显的好处，而图 22 没有指明对象的撤销，图就显得比较融洽。记住要遵循敏捷建模（Agile Modeling，AM）（第 12 章）的实践——简单地描绘模型。

注意，在实时系统中，内存管理经常是一个很关键的问题，以致建模人员可能需要对对象的撤销建模。

132．不必过度考虑激活期竖条的问题

激活期（activation）竖条是 UML 顺序图中从类元上悬挂下来的虚线上的小长方形。激活期竖条是可选的，它是用来指明控制焦点（focus of control）的，隐含说明处理是在哪里出现以及处理所需的时间。然而，激活期竖条几乎就是视觉上的噪音（noise），因为内存管理问题应该留给程序员处理更好。一些建模人员更愿意采用如图 21 所示的“连续型风格”（continuous style）的激活期竖条，这种风格的激活期竖条一直保持到处理的结束；而另一些建模人员更愿意采用如图 22 所示的“中断型风格”（broken style）的激活期竖条。这两种风格都是好的，选择其中的一种然后就接着做下去。（不必再考虑是用“连续型风格”还是用“中断型风格”。）

6.2 类元准则

注意，类元的命名规范是在其他章节中描述：用例是在第 3 章，类和接口是在第 4 章，构件是在第 10 章。

133．当对象会在消息中被引用时，命名这些对象

顺序图中的对象用标准的 UML 格式“name: ClassName”标记，其中 name 是可选的（有 name 的对象称作已命名对象，而没有 name 的对象称作匿名对象）。在图 21 中，Student 的实例被命名为 theStudent，因为这个实例作为一个消息的返回值被引用，而图中的任何地方都不需要引用 SecurityLogon 的实例，因此这个实例可以是匿名的。

134．当多个对象具有相同类型时，命名这些对象

当一个顺序图包含同一类型的几个对象时——例如，在图 23 中，可以看到类 Account 有两个实例——则应该为该类型的所有对象命名，以使图没有二义性。

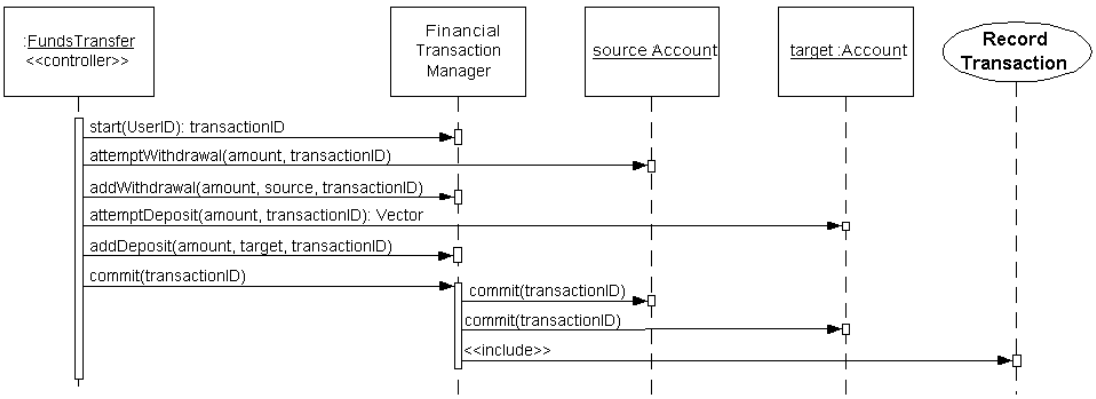


图 23 在帐户间转移资金

135．在类元上应用文本形式的版型要一致

表 5 总结了一些通用版型，建模人员可能会考虑在顺序图的类元上使用这些版型。不要花很多时间为应该使用哪个版型做极度痛苦的选择——例如，<<JSP>>和<<java server page>>都是可以的——只需要选择其中的一个，然后使用时保持一致即可。

版型	用法
<<ASP>>	设计时用于表示微软的 ASP（Active Server Page）页面。
<<component>>	设计时用于指明构件。
<<controller>>	用于指明控制器类，这些控制器类实现了使用场景（usage scenario）和（或）涉及多个业务类的逻辑。
<<GUI>>	设计时用于表示图形用户界面的屏幕。
<<HTML>>	设计时用于表示 HTML 页面。
<<interface>>	设计时用于表示 Java 接口。
<<JSP>>	设计时用于表示 JSP（Java Server Page）页面。
<<report>>	设计时用于表示打印的或电子的报表。
<<system>>	用于指明系统参与者。
<<user interface>>	表示普通的用户界面类。一般是用于分析层的图，在这些图中建模人员还没有决定要使用何种的实现平台。

表 5 通用版型

136．慎重地使用可视化版型

建模人员可以在顺序图中使用可视化版型，如图 22 和图 23 所示，但这不是很常见的做法，因为这样可能会减少图的可理解性。在图 22 中，Customer 是一个参与者（使用了和用例图中相同的符号），OrderCheckout 是一个控制器类，CheckoutPage 是一个用户界面类，Order 是一个业务实体类。

需要注意的是，可视化版型通常由团队使用，用于开发鲁棒图（Rosenberg and Scott 1999；Ambler 2002）。因为使用了像图 22 中所描绘的可视化版型，因此涉及项目的每个人都习惯于这些符号。

137．把注意力集中于关键的交互

敏捷建模（AM）的实践——创建简单的内容——建议在创建模型时要把注意力集中于系统的关键方面，而不要包括无关的细节。因此，如果顺序图是用于探究业务逻辑的，就没必要包括对象和数据库之间的详细交互，像 save()和 delete()这样的消息可能就足够了，或者更好地，可以简单地假定持久性（persistence）会适当地被处理，而不用考虑那个层次的细节。在图 22 中，没有关于从数据库或对象缓冲区中读取 orders 和 order items 的任何逻辑，也看不到关于 CreditCardPayment 类连接到 Payment Processor（译者注——原文为 payment processor，为了和图 22 一致，改为 Payment Processor。）的逻辑，但这些逻辑肯定是要在后台发生的。通过把注意力只集中于关键的交互——即那些和所要建模的东西有关的——可以使图尽可能简单，但仍然可以达到建模目的，因而增加了建模人员的生产率，也很可能增加了图的可读性。

6.3 消息准则

注意，操作特征标记（signature）的命名规范——和消息、参数、返回值的命名相关的准则——在第 4 章中有详细的描绘。

138．把消息名放在箭头旁

大多数的建模人员会调整消息名的位置，如图 22 中的 calculateTotal()，使得消息名和箭头对齐。一般地，消息的接收者会实现相应的操作，因此把消息名放在类元的附近是有意义的。

注意，图 23 并没有遵循这条准则。所有的消息名是在箭头的尾端对齐，它们是在发送者的附近，采用这种方法的优点是很容易看清楚所要建模的场景的逻辑，缺点是难以确定被图右边的类元所调用的操作是哪一个，因为需要沿着线找到另一头的调用符号。和通常做法一样，选择一种方法并一致地应用它。

139．直接创建对象

通常在顺序图中有两种方法指明对象的创建：发送<<create>>版型的消息，如图 22 所示的 OrderCheckout 对象；或者直接显示对象的创建，即把类元在图中向下移动一些，并从类元的侧面调用消息，就像图 21 所示的 theStudent 对象和图 22 所示的 CreditCardPayment 对象。采用直接创建对象这种方法的主要优点就是可以形象地传达出这样的意思，即对象是在所要建模的逻辑的中间某部分才开始存在。

140．在软件消息上应用操作特征标记

不管什么时候，当一个消息被发送到基于软件（software-based）的类元——如类、接口、或者构件——时，通常的惯例是用实现语言的语法描述消息名。例如，在图 23 中，发送给源帐号对象的消息是 commit(transactionID)。

141．对涉及人和机构参与者的消息使用描述性的文字

不管什么时候，当消息的源或目的是表示人或机构的参与者时，则用简短的描述性文字标记消息，以描述要传达的信息。例如，在图 21 中，由学生参与者发送的“消息”是 provides name 和 provides student number，是对现实中人要做什么的描述。

142．对于参数，优先考虑使用参数名而不是参数类型

注意，在图 23 中，对于大多数的消息参数，显示的是参数的名字而不是参数的类型，唯一的例外是 start()消息中传递的 UserID 参数，这个参数可以使读者准确地确定消息中传递的是什么值。有时候类型信息是不够的，例如，消息 addDeposit(amount, target, transactionID)比 addDeposit(Currency, Account, int)传达了更多的信息。操作的类型信息用 UML 类图捕获更好。

143．指明参数类型作为参数的占位符

有时候，参数本身所要传递的信息和建模人员所要建模的东西并不相关，相关的是“有某些东西要传递”这个事实。在这种情况下，指明参数的类型，就像图 23 所示的 start(UserID)中的参数。

144．把<<include>>版型用于用例的调用

图 23 显示了一个用例在顺序图中是如何通过一个<<include>>版型的消息被调用的，当所要建模的使用场景包含直接调用用例这样一个步骤时，这是一个方便的窍门。

6.4 返回值准则

145．不要对明显的返回值建模

返回值用带虚线的箭头及指明返回值的标记表示，返回值是可选的。例如，在图 21 中，指明返回值 theStudent 是作为消息调用的结果从 SecurityLogon 类返回的，而在图 22 中，没有指明发送给订单（order）的 getTotal()消息的返回值。在第一个例子中，创建一个安全登录（security logon）对象的行为会导致生成一个学生（student）对象，这个是不明显的，而向订单对象发送请求其总数的消息，其返回值是明显的。

146．只有当需要在图中别处引用返回值时，才对返回值建模

如果需要在顺序图的别处引用返回值（经常是作为另一个消息的参数被传递），则在图中明确指明返回值是从哪里来的。

147．把返回值放在箭头旁

大多数的建模人员会调整返回值的位置，使它们和箭头对齐，如图 21 中的 yes 和 theStudent。一般地，返回值的接收者会用它做某些事情，因此把返回值放在接收者的附近是有意义的。

148．把返回值建模为方法调用的一部分

为了不把图搞混乱，考虑在消息名中指明返回值以代替虚线，可以用 returnValue := message(parameters)这样的格式，图 22 中的 authorizationCode := reserve()消息用的就是这种格式。用这种方法，建模人员只需单条的消息线，而不是一条消息线加一条返回值线。

149．指明返回值类型作为返回值的占位符

有时候，所要返回的信息和所要建模的东西并不相关，重要的是“有某些东西要返回”这个事实。在这种情况下，指明返回值的类型，就像图 22 所示的 commit(): AuthorizationCode 那样。

150．为简单的返回值指明实际值

在图 21 中，yes 这个值是消息 isValid()的返回值，这非常清楚地表明了学生名和学号的组合是有效的。如果返回值被命名为 Boolean，以此指明响应的类型，或者命名为 eligibilityIndicator，以此指明返回值的名字，则都不如用 yes 这个返回值清楚。（本译稿经清华大学出版社同意刊登）

斐力庇第斯从马拉松跑回雅典报信，虽然已是满身血迹、精疲力尽，但他知道：没有出现在雅典人民面前，前面的路程都是白费。

学到的知识如果不能最终【用】于您自己的项目之中，也同样是极大的浪费。而这最后一段路最是艰难。

UMLChina 聚焦最后一公里，所提供服务全部与您自己的项目密切结合，帮您走完最艰难的一段路。

糟糕界面集锦—界面设计考虑要点

pixelcentric 著，雷立辉 译

查看评论

颜色的使用

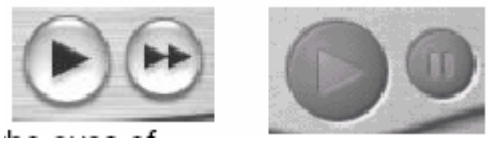
颜色选取和使用主要是针对于界面的图标来说的，其它地方几乎不会碰到这类问题。用户通常习惯于白底黑字（或相反，某些视力有障碍的人使用黑底白字），如果选用其它配色方案可能就会造成麻烦。

1. 考虑可读性

使用彩色用户界面元素时，要考虑紧接这个界面元素的相邻界面元素是什么。特别是要注意文本所在的背景底色。Apple 就在 Mac OS X 10.2 版中的强制退出管理小程序中犯了这个错误。

这样设计的最初理由是为了表明名称显示为红色的应用程序在此时不能响应。理论上这似乎是一个很好的特征，因为它能很快让人找到出错的程序。但实际上由于被选择的项目背景是蓝色的（就不说用户可能会改变系统颜色设置了），这种蓝底红字实在是难以阅读。

另外一个问题与对比度相关的便是用户界面中的关键元素是否醒目。就以 QuickTime4，5 及 6 各自的电影播放控制按钮来说吧。灰色的，低对比度的 QuickTime 4 按钮对于有些人是非常难以辨认的一特别是眼睛没有年轻人亮行动没有年轻人快的那些中老年人来说一而这些在 QuickTime 5，6 中“Aqua”化的按钮则变得对比度更强从而好用多了。



2．考虑文化差异

许多文化中，红色都表示一种否定色彩的回答或命令，而绿色则表示一种积极色彩的回答或命令。即红代表“否”而绿代表“是”。如果忽略这点则可能造成问题。

这个糟糕的以是否回答的 Windows 风格的对话框竟然在 Mac 操作系统 X 中也出现了。其错误不仅仅在于它非动词性质的按钮而且更重要的是其使用的颜色也有问题。“否“用红叉表示，似乎表明了它是一个否定的，不应被选择的按钮－但实际上，当你一看提问便知选择“否”并不会使数据丢失而看起来似乎安全的绿色标志实际上是具有破坏性的选择。

使用颜色的失误正反两种情况都有：一些应用程序本来是安全的选择却做了危险的显示寓意。比如说一个提示性质的对话框使用红色符号来无谓地吓人；相反的，上面截图中的白色提示图标理应改为黄色的警告图标。我们一定要根据上下文使用正确的颜色设置。

硬性使用红－绿颜色对来表示这些与是/否选择无关的问题回答大体说来是有害的。比如说，一个计划程序用绿色表示人员是合适的而红色表示人员不太合适这样的方案绝对是有问题的，因为这种颜色的寓意与具体使用的情形关系并不是太明显。而在这种情况下，用一个亮色/有阴影的灰色配色方案会更好一些。因为灰色通常表示一种无效状态。

3．不要硬性规定颜色

你的程序如果使用自定义控件或自定义高亮颜色（比如文字的颜色），不要硬性规定这些元素的颜色。如果你规定了颜色，如果用户使用的可视化主题和窗口管理器所设定的缺省颜色不一样时，你程序就有可能完蛋：要么因为难看，要么因为不能阅读，要么因为既难看又不能阅读。不要硬性使用颜色而要使用系统提供的颜色设定。如 Mac 操作系统的字符颜色（accent colors）和控件颜色就是由显示管理器来设置的。

如果使用系统颜色不可行时（比如说你程序所运行的平台并没有这项功能），那么就应该将配色方案写成用户自定义的。如可以通过程序中的参数设定窗口中的颜色拣取器来让用户改变文本颜色。

总是要保证测试你的程序：使用不同的窗口管理主题来确保在不同的颜色配置条件之下你的程序都是易于使用的。

4．不要过度使用

就像界面中的声音一样，如果使用得当，颜色也是非常有效的。但是太多的颜色也会让你的界面主题分散而且华而不实。当一个界面中太多的元素都是以亮色显示时，这些颜色效果就会相互抵消从而造成每一个元素都不会突出，从而看起来个个都好像是沉闷的灰色元素一样。这一点对于图标的设计尤为重要：大片的，明亮的颜色区域会让图标的其它部分黯然无光。用颜色就要用得有意义。

声音的使用

如果使用得当，声音就像颜色一样会是一种具有潜力的强大的和高信息量的界面元素。但还是应该少而精地使用声音元素。

1．不要过度使用

声音或许会造成非常吵闹的效果，所以一定要在真正需要的地方使用它。在作为可视的提示或作为视力不佳的通用性访问特征时，声音使用最为有效。当然，在一个程序中的各种操作都有声音提示将会造成听觉过载甚至于困扰以及惹恼用户。所以要避免在相当短的时间内产生过多的声音信号，从而让用户将声音和一个错误消息或者操作完成相联系。

2．不要单独使用声音

相比需要用户干预的对话框来说，声音是一种过而不留痕的信号。所以不要将声音作为传递信息的唯一形式，因为用户有可能没有听到声音；用户可能会听力不佳；可能会在程序播放声音时不在现场；也可能就因为你过度使用声音嫌烦而直接关掉音响！

图标的使用

在一个图形界面环境中，图标是文档、目录等信息的载体而且用户主要通过图标来操作这些对象。请记住：对于大多数用户来说，图标并不是在文件系统中某个位置存放的文件的一个指针——用户认为图标本身即是文件。

注意：现在只有 Mac 操作系统图标设计指南。而对于 Windows 以及 KDE 窗口界面系统的图标设计指南将在以后发布。不过，现在这个指南的某些章节是与平台无关的。

1．桌面的革命

Mac 操作系统 X（Mac10）的发布带来了全新的桌面图标设计方案。Mac 7 已经给大家提供了易于设定的 256 色图标；Mac8.5 给我们提供了 8 位通道的真彩图标；但是 Mac X 的 128x128 大小的缩略图式图标则是一个更大的飞跃。

在某些方面，这个改变又是不太让人容易接受的。因为使用 ResEdit 软件来一个一个象素绘制图标的年代已经过去，而变成了使用诸如 Illustrator 或 Photoshop 这类的大型工具来绘制它们；有些人也可能说那些友好的，让人感到温暖的百分之百的手绘图标的年代已经逝去。但是另一方面，Aqua 方案又使得图标变得水晶般透亮而且在大显示器上显示得足够大。谢谢苹果在 Aqua 人机界面设计方案上所做的努力。相比 Mac7 或 Mac8/9 所创建的图标，在软件中使用 Aqua 图标总的来说会与整个 Mac10 协调和一致。这最终达到视觉统一的效果。

相比昔日系统中的 32x32 象素图标的设计来说，创建一个 Mac10 图标会更难一些。首先图标的面积比以前大了很多，其次表现的细节也比以前要求更高，而且还要确保在低分辨率情况下（甚至到 16x16 大小条件下）依然显示良好。从而因这些要求使得图标设计更有难度。由于 Aqua 图标提供了从 16x16 到 128x128 范围之间的任意大小的图标，所以图标设计师不能只设计一款 128x128，或 16x16，或在这范围之内的一个尺寸的图标；图标设计师必须为这个范围内的任意一个大小的方形设计一个图标。为了设计出一款干净、醒目而且易懂的 Mac X 的 Aqua 图标，我们将会在这篇文章里介绍设计师必须要了解的设计哲学和设计概念。

2．基础知识

图 1 应用程序图标（左）和文档图标

任意 Aqua 图标的基础是形状，这是首当其冲的特征。而颜色及图案的差异则是紧随其后的。应用程序图标（图 1 左）通常是稍微反时针旋转的矩形图片。它们经常含有一些具有标志意义的元素（通常是一个工具，如一支笔或是一把刷子）使得用户知道这个程序大致是干什么的。应用程序图标应该具备一种传递当它们被打开时会发生特定事件的含义。而把产品标题作为图标则是一种不好的选择。

文档图标（图 1 右）通常与其相关的应用程序（host application）图标在色彩和图案保持类似，但是这些内容应该被一个右上卷边的纸状图案所包围。如果相关的应用程序包含一个工具或标志（如 TextEdit 的笔或预览用的放大镜），那么文档图标中就应该略去这个工具或标志，以免用户将这两类图标混淆。如果一个文档图标包含有一个徽标例如文件类型的标志符，那么这个徽标就应该完全包含在这个文档图标纸型图案中（如图 1 是一个好的例子而图 3. A 是一个不好的例子）。

图 2A 一组设计良好的图标

图 2B 设计较差的图标

工具条图标设计图案甚至应该更与原始形状接近。如果一行工具图标大体相似则会造成用户使用时混淆。这就是在浏览器工具条中做一个圆形停止浏览图标并且还有一个圆形的刷新箭头图标不好的原因了。就是这个原因，苹果在 Mac10 中重新设计了一组浏览器工具条图标，它们形状各自相异：计算机图标是矩形的；应用程序图标是三角形的；删除图标是圆形的等等。即是这些图标的颜色属于同一个颜色系列，它们仍然容易区别——相反，在 Mac10 中的浏览器 Chimera 中的工具条（图 2. B）则缺乏与上述工具条相匹敌的设计。

3. 风格

图 3A Mac9 风格，不要这样做

保持 Aqua 图标的统一风格也是非常重要的。Finder 图标（finder icons）应该是具有真实照片感的，但不完全是。在任何情况下图标也不应该是一副照片的缩小版。而且，尽管 Aqua 图标是 3D 的，但也不完全是 3D 的，毕竟我们还要保留图标的基本形状通透（shine through）。Aqua 图标中的 3D 效果应该以我们面对前方稍上的角度来渲染，就像观察者在看一个放置在他（她）前面桌子上的物体一样。

另一方面，工具条图标应该是较简单而且较平面的东西。设计工具条图标时应该较少使用阴影，3D 效果。所有工具条图标都应该是正（面直对）视图（或如果可行的话，俯（首向下的）视图）。应用程序图标（如终端或网络工具）也应该是正视图。

值得注意的是 Mac10 的图标阴影风格也和以往的版本大不一样。与 Mac9 图标阴影是由光源以左上角度打过来不同的是，Mac10 采用正上方光源（即垂直 90 度）产生的阴影。阴影应该在物体的下方而非侧面（如图 3.A 就是错误的阴影）。

图 3B 无可救药。不要这样做

通常，模仿 Aqua 窗体控件和按钮等做一个类似风格的图标是错误的想法。不仅此种风格无可救药，而且由于这种样式的窗体控件充斥了系统的每个角落。使用与 Aqua 界面元素类似的图样也会让用户不能分清楚哪是图标哪是控件。

当我们使用得当的时候，这些珠宝般晶亮的控件会将这些图标衬托得华丽多彩而且过目不忘。不过，如果使用不当，你的图标将被迷失在早已存在的这些 Aqua 珠宝的汪洋大海之中。特别的，除非是要使用透明和光泽特性表达一种特殊的意义，否则这两个特性总会被排除在图标的设计之外（举例来说如图 1. A 中预览程序图标中的放大镜）。

4. 优雅的减少细节

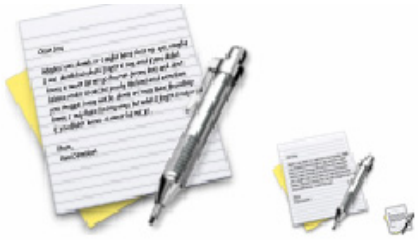


图4A 细节逐渐消失

创建Mac10其中一个关键概念便是当创建文件浏览器中使用的图标要让这个图标随着它尺寸的变小让细节逐步消失。说着容易做着难，因为你不知道图标变小后是什么样子，也不知道某些细节在变小后是什么样子，因为图标缩小时某些细节就会被其它细节淹没。在这种情况下，图标总体的形状和颜色就被那些在128x128图标中的局部细节来的更为重要了。创建这种效果的时候，有时也把这个过程叫做“退化细节（receding detail）”，也需要很多尝试，但这最终还是值得的，因为这保证了你的图标在尺寸很小的时候依然可以“容颜依旧”。典型的举例来自程序TextEdit的图标，这个图标包含有一段文字。尽管少量细节丢失了，这个图标的文字块依然在48x48大小的图标中依然存在。不过，图标上页面上的横格细线看不见了。即使在16x16大小（图4. A 最右的图标）图标尺寸最小的情况下这个文字块还是有一个墨色的轮廓依然保留，而最基本的元素信纸及笔依然可以辨认。

5. 采用小图标资源隐藏技术（Take A Hint）

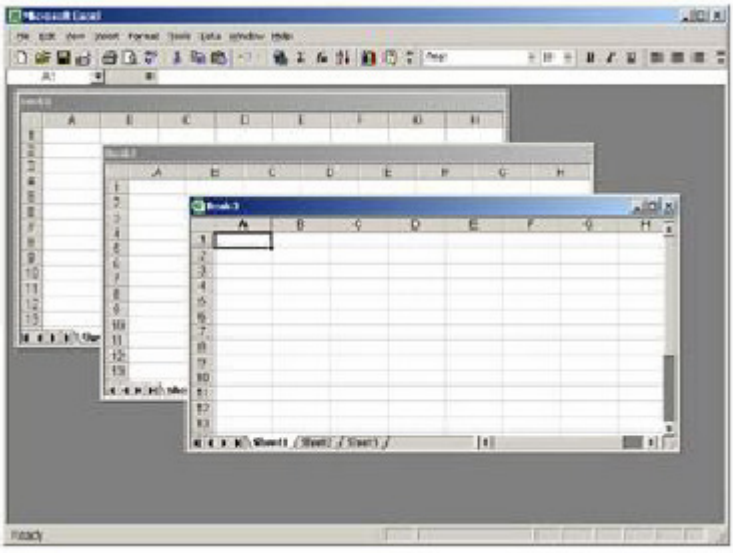
每一个 Mac10 Finder 图标重要的一点便是除了 128x128 资源之外还隐藏有一些其它尺寸的图标资源。隐藏图标尺寸有 3 种：16x16，32x32 及 48x48。如果图标资源里含有这三种大小的图标并且要显示这三种里某一种大小或近似大小的话，Finder 将会使用最接近目标尺寸大小的图标来缩放至目标大小。这种技术加上优雅减少细节（Graceful Degradation）技术，是一种解决细致入微的 128x128 图标在更小尺寸下显示仍然光彩依旧的好方法。

文档窗口

所有的主流操作环境都是为了一个目的，也就是处理数据即文档。典型的文档窗口的显示方式及操作方式都与其它如状态窗口或警告窗口在内的窗口明显不同。下面的文字就说明了在设计文档处理的过程中要记住的事。

1. 不要使用多文档界面

多文档界面（MDI）的使用在 Windows 和一些 Unix 桌面管理系统中可以见到。由于本身就是有缺陷的窗口管理系统，为了解决这些缺陷反而导致了更多的窗口管理问题，所以让很多人吃尽了苦头。MDI 其中一个致命缺点便是这个系统限制了窗口的显示区域。而且，MDI 造成一个程序管理多个窗口的难题并且剥夺了用户同时在几个应用程序间操作的能力。Mac 操作系统 X 就没有采用 MDI，所以它就赋予了用户在程序窗口随意切换的能力，即使是几个不同程序的窗口。你可以在你的系统屏幕的底部开一个 IE 窗口，在其上再打开一个 Word 文件。而在这个窗口上面，你还可以打开另一个 IE 窗口等等。这样用户就可以不用切换到那个程序去看那个程序窗口的内容。这个特性在你将一个网页里的文字拖放到另一个文本文件中感觉尤为顺手。



另一个有趣的事儿便是微软自己的程序就对 MDI 的使用方式不一致。最新版本的 Excel 使用了 MDI 而最新版本的 Word 却不是这样。这让操作 Excel 没有操作 Word 顺手。IE6 也没有使用 MDI。看起来微软似乎有点在行进中开火的意思（fire and motion），一面规劝第三方使用 MDI，而另一方面自己又在逐渐抛弃 MDI。

2．提倡添加拖放支持

拖放操作是一种相对简单的手工操作，并且是执行如挪动文本或文件等各种命令的自然方式。添加文档间和程序间的拖放支持是重要的。Windows 在这方面却做得不太好。我们本来想通过拖放一个项目来重新安排一下这个项目的位置没想到它却为我们创建了一个快捷方式；本来将这个文档中的文本通过拖放到另一个文档中去应该产生拷贝操作，没想到 Windows 却弄成剪切操作。当然，文档内的拖放文本的操作应该就是简单的挪动位置而不产生其它操作。

当拖放操作时鼠标按键松开时有一个明确提示是非常重要的。一个好的办法便是在鼠标箭头旁边显示一个状态图标，因为这时候用户正在看着鼠标箭头。同时，在拖放时显示一个虚化的图像也是一个好办法。

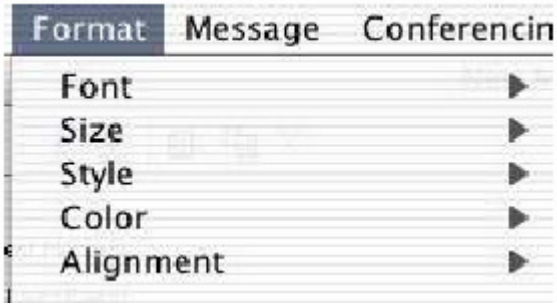
请不要对本来没有拖放意义的标准控件添加拖放支持，同时也要禁止用户将数据拖放至没有拖放意义的控件中去。

3．使用标准命令和控件

用户通常习惯于原来操作系统的工作方式，所以发明一个新控件或改变原来控件的工作方式的想法是错误的。不要重新发明轮子。如果你真的需要一个自定义控件来完成操作，请尽力将其操作方式和整个用户界面的操作方式保持一致。大体来说，你所设计的界面元素只可能是复杂的，很有可能这个操作是太复杂了。

4．避免多级子菜单

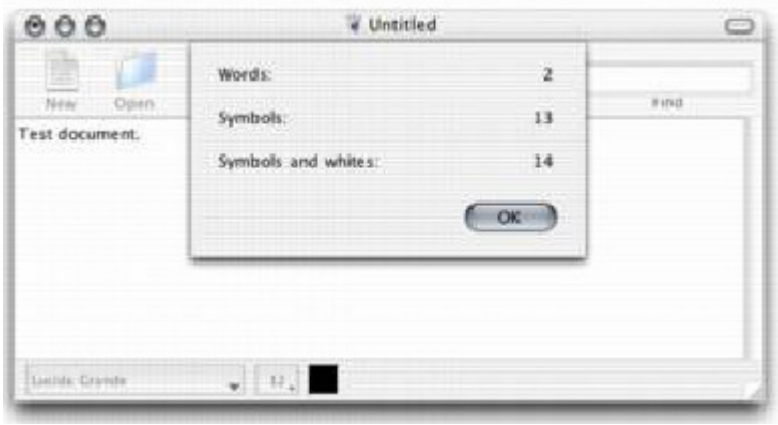
子菜单或许是唯一一种难以手工操作的界面元素了，所以在界面设计时如果可能的话一定要尽量避免使用它。多级子菜单是最有问题的，因为鼠标箭头稍一脱离菜单就会造成整个菜单全部消失，所以强迫用户重新在这些菜单中战战兢兢的挪动箭头去找那个隐藏很深的命令。



FirstClass 软件（如上图）将非常常用的编辑命令做的如此之难用以致于他们不得不创建一个新的替代方法（如这个图片底端的工具条）（译者注：原图可能有失误）去做这件事。如果可能的话，避免使用子菜单，特别是像上图那些重要的命令更是不要把它们放进子菜单。如果是真的需要的话，也要让它们只有一级。

5．不要劫持计算机

任何一个应用程序相关的操作都不应该让应用程序阻止整个系统的运行；同样，一个文档相关的操作也不应该阻止整个应用程序的运行。这是 Mac 操作系统 X 发起的两个标准，这也应该被其它操作系统所遵守。用户不喜欢不停打断他操作的操作系统。诸如与文档相关的警告，信息窗口和保存命令都应该显示在那个文档窗口里面，而不是阻止整个应用程序以至整个系统。



举例来说，既然如一个文字处理系统或文本编辑工具的字数统计功能是与当前打开的这个文档有关，所以这个窗口显示的信息也就只与这个文档有关，所以用户就可以不用关闭它就可以在另外一个文档或程序继续工作。（译者注：目前 Microsoft Office Word 2003 还没有达到这个要求）

6．鼓励用户探索

不要让用户在使用你的程序时产生对某些命令或特征的使用恐惧心理。尽力提供多级的“撤消”和“重复”命令支持，并且让用户明确知道你对这个操作提供了支持（如果有工具条，将“撤消”和“恢复”放入工具条，并且把这两个命令放进编辑菜单里）。让你的程序能容忍用户的各种操作。

如果一个操作不可恢复，在执行这个命令前要提示用户。允许用户取消这个命令，就像上面这个截图例子一样。

对话框

对话框通常是用来请求用户输入或显示信息的。它们应该尽量少用——仅在这个对话框有充足理由存在的情况下使用。下面是几条关于对话框使用的指导性意见。

1．抛弃“是”“否”回答

经典对话框的缺点是使用“是”和“否”作为对话框按钮的内容。这会强迫用户必须仔细阅读对话框内的内容从而得知他（她）究竟同意还是不同意。请考虑以下的例子，它们分别来自于 Windows 和 Mac 操作系统 X：

第一个例子不会让用户知道这个对话框是何用途。没有对话框里面文字的提示，用户不可能知道下一步该怎么办。相反，第二个例子用意则非常的明显以致于不需要解释文字用户也知道该怎么做。这两个例子都是来自于标准的系统保存对话框。Mac 操作系统使用动词来作为按钮标题，所以它能明确提示用户这个按钮的用途，所以用户甚至不用阅读对话框提示就可以做出选择。

当这个对话框命令不能用动词来表示时，也不要用“是”“否”，而要用“确定（OK）”和“取消（Cancel）”并且让“OK”作为指令执行的动作按钮。如果这个命令或操作会造成数据丢失或破坏，则一定要把“取消（Cancel）作为缺省按钮。

2．要有实质信息，也不要信息过量

如果用户不能理解对话框里的内容，这就意味着是这个对话框设计者的失败。通常情况下，就像标准保存对话框或打开文件对话框那样，所有对话框都应该含有一个用来提问或警告的消息（图标），另外还含有一段附加的描述性文字来提供额外提示譬如“这个操作不可恢复”。你千万不要像下面这个对话框这样做：

这个问题对话框的标题与所提示的信息不符，更糟糕的是没有任何信息告诉用户究竟要“确信”什么（译者注：对话框显示“您确信吗？”的问题，而窗口标题却是“错误”）。

另一方面，不要信息提示过量。不告诉用户要做什么是错误的，而告诉了过多的信息也是错误的。长信息阅读费时不如短信息简明扼要，长信息会有更多的可能性让用户感到厌烦。

3．使用有意义的留空

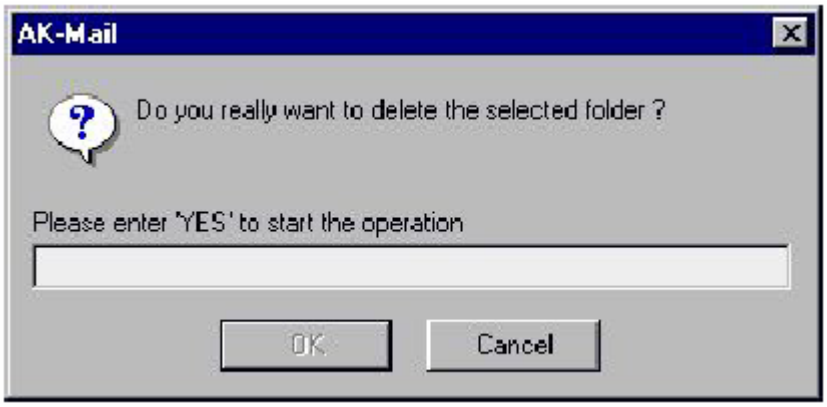
如果你设计的对话框需要两个以上的按钮或者其它控件，请使用空白来传递特殊的意义，再以本小节 1 中的例子来说明：

第一个例子完全没有使用空白，按钮之间的距离都是一样的，尽管我们知道这三个按钮点下去造成的后果大不相同。有意思的是，这个会造成数据损失的按钮竟然相当不方便地放在中间。所以，这就造成一种被错误选择的可能性：如果你要点“是（Yes）”的右部，或者“取消（Cancel)”的左边，可手不小心一抖错过了几个象素就点到“否”按钮上了！数据因此而丢失。

而下面的这组按钮就有效的利用了空白。既然对话框中的提示是“您想在关闭文档前保存内容吗？”，所以这些按钮的动作结果和上面的一组一样：“取消”使关闭文档的命令取消；“保存”将保存文档并且关闭；因此“不要保存”按钮是唯一可以造成未保存数据丢失的按钮，所以它和其它两个按钮“取消”和“保存”（它们都不会未保存数据丢失）之间有一个很大的空白。这样的分布保证了让用户偶然错误选择“不要保存”的可能性非常小。

4．用户不是傻瓜

不要剥夺用户保护自己的权利。一些应用程序假设用户存在危险性而让用户在对话框中输入一个消息诸如“Yes”之类的东西才能向下执行从而来保证数据安全。这减慢了用户的操作速度并且让用户感觉这个程序不太好用。每个人都不喜欢别人把自己当作傻瓜。需要键盘输入的需求同样没有考虑这些对键盘操作不太灵便的需要，尽管他们对键盘不太适应可对于鼠标却可以操作。



隐喻

界面通常都需要使用一些隐喻来帮助用户理解和学习如何使用程序。或许最著名的应该是施乐公司帕罗阿尔托研究中心（Xerox PARC）开发的桌面隐喻——苹果使用自己的一些股票换取了在麦金塔什计算机上使用该隐喻的使用权。隐喻可以让用户更加简单地操作系统，但隐喻不当则会削弱程序的易用性。

1．足够真实

设计界面元素时要基于真实世界里的物体，不要使界面元素完全与真实物体无关。直到 Mac 操作系统 8 和 9，苹果才从一个非常不恰当的隐喻中脱离出来：用户通过将磁盘图标和 CD 图标拖到垃圾回收站中来弹出磁盘和 CD。对于偶然使用苹果系统的人来说很难知道这种方法，并且用户总以为把磁盘图标或 CD 图标拖到垃圾回收站中去会删去它们的数据。这个隐喻的使用是因为程序员的思维方法：删除文件在某种意义上是将东西扔掉而弹出磁盘则是完全不同。这种意义根本不明显。Mac 操作系统后来的版本就将弹出命令变成弹出磁盘的首要命令。Mac 操作系统 X 又前进了一大步：当拖动磁盘图标的时候，垃圾回收站的图标就会变成一个弹出图标。将磁盘拖到弹出磁盘图标上就十分的形象生动了。

2．也不要太真实

如果将一个界面设计得和真实世界完全一致也是不太可取的。应该采用真实世界里简便的使用方法而抛弃那些难用的东西。比如说，真的没有必要像真正的橡皮擦那样痛苦而缓慢的在文本编辑工具中“擦”去文本。相似的，也没有必要像倒垃圾桶那样来清空 MaC 或 Windows 里面的垃圾箱。

国际化

国际化的问题的要点不多,最重要的是你不能假设用户所在的国家，这包括他们使用的地址格式、度量衡、小数（十进制数字）分隔符（decimal separators）及甚至图标风格。

1．不要假设用户所处的地方

永远不要强迫用户使用美国使用的地址格式。许多国家并没有“州”的概念。如果强迫用户必须在州的域段里输入一些字符才能允许用户使用这个软件的话，这可能使你的用户放弃使用这个软件。如果你不能支持国际地址格式的话，那么最好提供一个传统的文本框来让用户输入地址。你的用户绝对足够聪明且知道他们自己的地址及其书写格式。

同样的原则使用于小数（十进制数字）分隔符、度量衡、货币符号等等。让这些值变成用户自定义的。当然，更好的办法应该是检测用户使用的操作系统的区域设置，让程序根据它来显示不同的表示方法。即使这样，这些输入值应该还是用户可设置的，因为一个芬兰人可能暂时呆在英国。

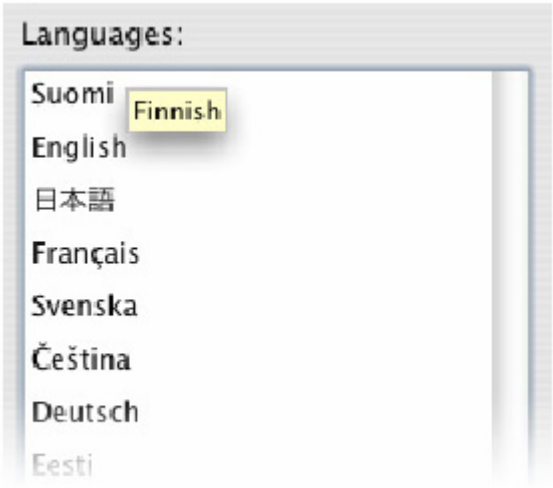
2．考虑文化差异

当你的软件要国际化时，请仔细考虑软件的界面元素可能因为文化的差异而被用户错误理解。文化差异范围非常广泛，涉及从不同颜色具有不同的含义到不同的手势在不同文化里表示的意义完全相反等诸如此类的问题。比如说，在西方世界点头表示同意而在某些亚洲国家点头则表示不同意。

在软件的本地化过程中，开发者总是除了对字符串进行翻译而不考虑其它要本地化的东西。比如让图像、动画等等都要顺应当地的风俗习惯。比如说，一个瑞士籍的信件读者可能不习惯使用一个美国风格的邮箱图标作为自己的邮箱图标。同时，一些符号或显示元素甚至被视为文化挑衅。一个典型的例子便是那些需要磁盘引导（disk first utilities）的软件，它们使用十字架符号。对于开发者来说，十字架表示修复和还原，但有些用户则认为这个是耶稣受难的表示（pro-Christian statement）。

3．使用正确的语言名字

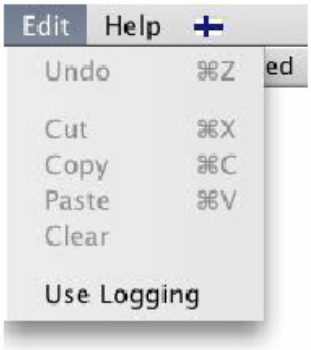
如果可能的话，使用它们本来语言里使用的名字而不是英文名称，无论这是让用户选择界面的语言还是其它你需要知道用户或顾客的母语的地方。这对于用户从语言列表中通过用自己母语表示的母语名字特别有意义。这不仅让用户更容易的发现自己的母语名字而且特别是在那些不是以拉丁字母排序的语言如日语和中文的列表中能加快选择速度。或许你还想在这些以母语表示各自的语言名字旁边还显示一列英语名字，比如说，苹果就在 Mac 操作系统 X 中的语言选择列表中使用标签来显示其对应的英语名字（如下图）。



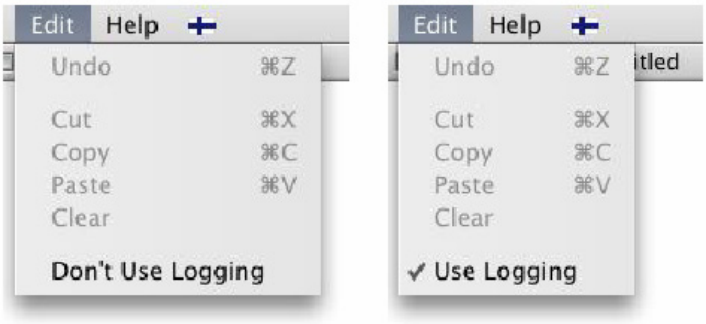
二义性

在使用计算机的时候，用户会随时改变软件设置。所以软件的命令和文档不能有二义性表述。

总是使用肯定的语言表达方式

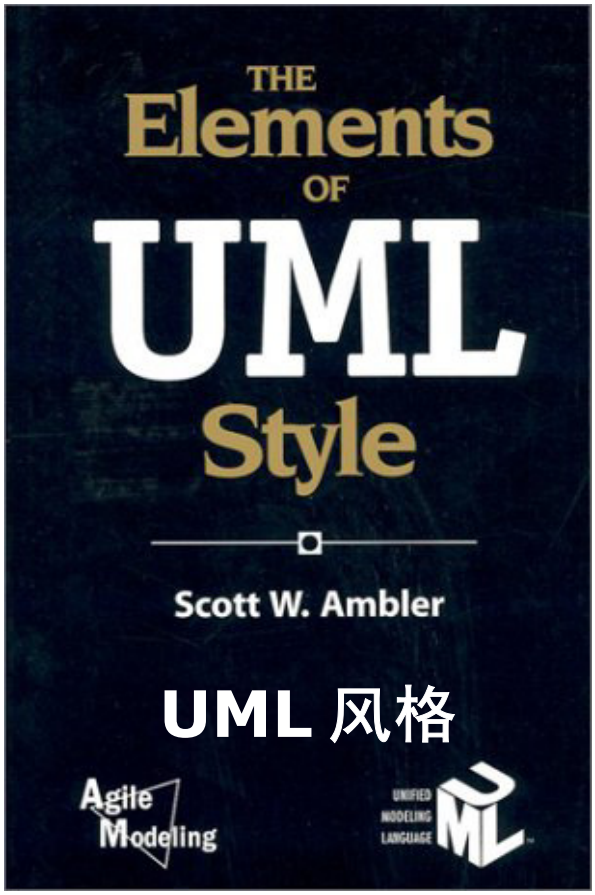


如上图所示，菜单项“Use Logging（写日志）”命令的结果可能有两种方案：一种使菜单项打勾，一种让菜单项改成否定的描述（如下图所示）。而对于程序实现者或者用户界面设计者来说，如果实践经验不足，则这两种方案似乎都是可行的。这样，采取哪一种方案就随当时的心情了。



而问题主要出在这儿：如果脱离上下文环境，命令“Don’t Use Logging（不要产生日志）”既可以认为是当前状态的表示，也可以认为是当这个命令被执行后产生的结果。所以这个菜单标题不能说明当前是否在写日志的状态。因而只有你执行了这个菜单之后你才有可能搞明白这个菜单究竟是什么意思。当然，这有可能导致不可预知的问题甚至造成破坏性的结果。

否定性质的标题是产生二义性的根本原因。这会让用户对你以及程序失去信任。所以一定要使用肯定性的命令和状态描述。不要说“不要产生日志”而要说“关闭日志功能”；不要说“不要使用小写字母”而要说“使用大写字母”。要让你的用户习惯于查看菜单项是否打勾。使用复选框控件而不要使用按钮按下去的状态来表示否定。



UML 风格

口袋里的 UML

批判索尼爱立信的手机设计

dozer 著

查看评论

We suffer from the bad design.

设计充满我们生活的每一个角落。从摇篮到坟墓，我们所经历的生活的历程中，所使用的物品和服务，从计算机、软件、电子产品、汽车到服装、建筑..几乎是所有的一切，都是设计的产物。似乎每个人都会用笔画上几笔，但大师和俗人的作品是天差地别。大师总是少数，平庸总是压倒性的多数，在平庸中充斥着大量糟糕和拙劣。在设计中，同样有好的设计和差的设计，只是不幸的是，糟糕的设计并不是生活中的偶然错误，它是如此广泛的存在，以至于我们的日常生活中经常能感受到糟糕设计所导致的糟糕体验。一般而言，我们都倾向于觉得大公司的产品是设计精良的产物。的确，大公司往往有更强的资源、更好的技术、更多的经验，但这并不意味着大公司就没有糟糕的产品。伟大的 GE 够强大了吧，但是 GE 产品也有很糟糕的设计。Apple 的发明人之一 Tog 曾经在一篇 case 分析中针对 GE 的一款洗碗机的糟糕设计提出这样的感慨：“so every company makes a bad design once in a while. But this dishwasher is a GE, the same people who engineer components for our nation’s nuclear arsenal. Should we be concerned?.”

在很多情况下，我们遇到不好的设计，我们除了忍耐和承受，没有别的办法，甚至很少有人公开地表达出来，那些错误却一直存在下去，直到产品生命期的终止。有时大家虽然感受到不爽，但未必真的去细想到这是设计导致的问题。当我在上海徐家汇地铁站中，发现其出口有一边有路牌指示，但另外一边竟然没有，这在有时真的会导致走错方向，我就因此好几次走错方向，而对于那些不熟悉地形的外地游客，这个设计上的缺陷会导致困惑和错误的发生。相信我不会是唯一的一个受害者，而且这个错误依然还在继续下去。

对于设计上的缺陷，只要留心观察，有太多的产品中可以找到。作为 product designer,我使用新手机 Sony Ericsson 的 T618 时，有意观察了一下它的设计特点。明显感觉到它所导致的不愉快体验。尤其在开始使用的那段时间，我有一种感觉，那就是 Sony Ericsson Sucks.我甚至感到有点恼怒

只是到后来，这种感觉慢慢变淡了，甚至觉得整体上其还算是 OK，过得去，时间长了，觉得它还不算是很糟糕，它甚至还是有一些聪明的设计之处，但它算不是上一款伟大的好产品，其设计上的缺陷使得它不能成为我心目中的伟大产品（我觉得 Nokia 8250 算是一款经典产品)。

Sony Ericsson T618 (以下简称 T618)，其外形比较吸引人，听说是去年最畅销的机型，现在上海满大街的都是拿着 T618 的群众。消费者总是倾向于被外在的东西所吸引，产品内在的特点，往往只有在实际使用过程中才有更明确的体验，它的问题只有在实际使用中才会被发现。现在的产品往往淹没在 feature 的战争中，可是众多的功能中，用户又真正用到哪些？又有多少功能是因为用户不会使用，而沉睡在产品中？消费者经常为自己不需要用到，或者不了解的功能，支付额外的金钱。

在更高层次上，产品的竞争，不只是 feature 的竞争，还包括 usability 方面的竞争。以用户为中心进行设计是一种态度，也是一句响亮的口号，但 slogan doesn't work。要真正实现以人为本的设计，并不是容易的事情。一些开发商至少对此有高度的认识，但还有大量的开发厂商、开发人员对此很漠视，或者缺乏足够的意识。

well.不再废话，让我具体说说 sony ericssson 的问题：

我一开始使用 T618 就觉得不舒服，刚刚从熟悉的 Nokia 8250 的操作习惯转换到新的模式下，觉得很不顺手，这往往是正常现象，但是索尼爱立信的问题在于，不仅仅是习惯改变导致的不适应，而是它的设计缺陷导致的问题。虽然现在时间长了，慢慢也习惯了，似乎一些不便都不是很明显的感觉了，但是其中一些设计上的缺陷/不足依然存在的。最明显的感觉是操作步骤烦琐，操作效率不高，特别是其中文输入法有明显的设计缺陷。然而，人就是一种适应性很强的动物，面对很多糟糕的设计或者糟糕的环境，一旦习惯了以后似乎很多东西都是可以容忍的了。Apple Macintosh 的发明人 Jef Raskin 曾经提到 Formation of Habits 对于易用性是很重要的，一些设计不良的产品一旦用户养成使用习惯，似乎都可以变得易用了，变得不那么敏感了。然而，如果更好、更聪明的产品设计可以有效地提高用户的生产率，提高用户的满意度。

下面描述一下对 T618 的使用体验实际场景：

新机器拿回家后，把卡装进去，开机竟然没有任何反应,心想肯定电池没有电，于是拿出随机附带的充电器准备充电，发现插头插不进去，又不敢用力插，生怕插坏了，但不用力插又插不进去，到底怎么了，一个简单的充电，我竟然不会。昏倒!!! 赶快拿出说明书，原来这个特别设计的插头需要倾斜以某种角度才能“咔嗒”一下插进去，而且还必需让有“闪电 flash symbol”标志的面朝上。我倒!! 这么简单的一个任务，竟需要如此麻烦的操作。也不知道是哪个“聪明”的设计师想出了这么一个貌似精巧的设计，可是他/她是否考虑到以下情况？

如果在黑暗的环境里，用户如何方便地充电？完全不方便，用户的最好选择是打开灯，仔细操作。

如果用户不仔细阅读说明书的情况下，强行用力插进去，怎么办？结果便是损坏充电插头。当然大部分用户可能都是小心谨慎的，但不能排除少数心急鲁莽的用户。

插头拔出来也是需要小心以倾斜角度拔出来，如果用户随手拔掉，有可能导致损坏。

事实上，这个非常简单的充电任务，根本不需要搞得如此复杂。让我们看看 Nokia 8250 的充电操作是如何进行的－－－只要把圆形的插头对照圆形的插口插进去就 ok 了，非常简单，非常直观，连傻瓜都会。我实在想不通为什么 SONY ERICSSON 的设计人员要把一个简单的功能复杂化，难道是要卖弄他们巧妙的构思或者要证明他们的独特技术？技术是让用户获得更好的体验，而不应该是用来卖弄或炫耀的东西，伟大的技术并不能保证制造出伟大的产品。作为设计人员和开发人员，首先应该明白技术是用来做什么的？大量的实践证明，那种以技术为主导的开发方式，“经常导致用户体验（user experience）将是一个糟糕的妥协，产品将不是被设计为用户使用的产品，而是被设计为一堆技术的载体，设计/开发人员的所作所为显得好像技术比产品更重要。”[Soctt Berkun,微软公司]。我从来就不怀疑移动通讯的重量级厂商索尼爱立信的技术开发能力，但是这种技术能力如果不是围绕用户为

中心，这并不能必然确保他们能制造出伟大的产品。对于用户而言，对手机充电这个常用任务，应该让用户感到简单而不是复杂，要做到这一点并不困难，甚至很简单，一个良好的设计能很容易做到这一点。Scott Berkun[微软公司] 曾经说道：“Why Simple Products Are Great Products 为什么简单的产品是伟大的产品？ 最强有力的工程技艺是那种我们不会留意到的技术。工程师和开发人员的真正力量体现在把非常复杂的东西转换为难以置信的简单事物....汽车工业、城市建筑、消费类电子产品中的最佳作品都体现了伟大的工程是如何聚焦于隐藏复杂性而不是显现它们。 增加产品价值的最佳方式是增加它的功能却没有增加复杂性,....使复杂的特性让用户看起来很简单——制造出了伟大的产品。”

当我以正确的方式接好充电器进行充电以后，发现没有任何反应。说明书上说首次充电可能需要 30 分钟后才会显示充电指示，然而在几个小时以后，手机还是没有任何反应，也就是说手机还是没有电。重新充电，还是如此。难道是手机有毛病？我只好把手机拿到索尼爱立信上海客服中心，一去那里，我 kao.人还挺多，等了好一阵，当我把手机交给客服中心的小姐，她微笑地对我讲：“先生，你电池装反了”。彻底昏倒，真是好没有面子。是谁的错？是的，是我粗心大意，犯了错误。然而，真的是我的错吗？不，这是设计人员的错，因为他们没有正确地设计，使得我可以反装电池。同样是手机，Nokia8250 就设计得让用户不可能装反电池。一个好的设计是不会让用户出现这种情况的，而且要做到这一点，并不困难，防止用户弄错方向，有很多很多产品的设计都考虑到这个问题(如 Intel 的 CPU)。面对用户使用产品所犯的错误或者当用户不知道如何使用产品，来自很多设计师/开发人员的一个心照不宣的经典反应是“we need smarter users.我们需要更聪明的用户 ，似乎是用户应该为这一切承担责任，但这一切是设计人员的错，是糟糕、笨拙的设计让购买产品的用户感到困惑或者遭遇挫折。如何让产品更好用、更易用是设计/开发人员的责任。用户是人，而人是会犯错误的，这是很正常的现象。作为设计人员，我们不能要求用户不犯错误，而是要努力避免用户犯错误。研究 human factor 的大师 Jakob Nielsen 早已经提出了设计中应该遵循的基本原则 Error prevention（防止用户出错）。大量的易用性问题，就是因为设计时没有充分遵循 Prevention of Errors “原则。相信 Sony Ericsson 的设计者应该知道这个基本原则，然而他们没有做好。

而在实际使用 sony ericssson 完成日常功能时，它在交互操作方面的问题，在此我列举部分：

Sony Ericsson 最让我痛苦的是发手机短消息，因为设计上的缺陷，如果要输入中英文、数字、大小写等混合存在的情况，麻烦死了，假如要输入“我的电子邮件地址是 dozing_runner@msn.com,我的 yahoo 账号密码是 24YnBi24.7aCp3e 这么一段短消息，我保证你会很痛苦。实际案例：我有个同事也是使用 T618,有一次他发短消息的内容中需要输入一些密码（数字加大小写英文），足足输了 10 分钟，最后还输错了，都是这个所谓的专利 T9 拼音输入法惹的祸。

或者：不改变现在的界面模式，只是改变*键的用途，*键不再是用于切换拼音输入/笔画输入，而是用来作为中文输入/数字、字母输入的切换。当用户需要临时输入字母、数字的时候，只要按*号键，就可以输入。例如：按 2,则在屏幕上显示 2.a.b.c.A.B.C，用户可以选择需要的对象。而需要输入汉字时，再按*切换。至于并不经常使用的“拼音输入/笔画输入”则通过菜单进行。

发短消息，操作步骤烦琐，我需要按好几次才能发出一条消息，跟 Nokia8250 的简便操作形成鲜明对比。Sony Ericsson 的设计违背易用性的重要设计原则之一 Efficacy.

它的流程是这样的：按操作杆往右，调出“编写新消息”。。

Note:以下的界面全部是模拟界面，跟实际界面并不完全一致，只是为了表达概念。

为什么会出现如此麻烦的操作步骤呢？是因为 Sony Ericsson 提供更多的选择功能所导致的。那么有没有可能简化呢？也许。让我也提一种可能的解决方式。

说明如下：

点“继续”后，直接弹出通讯录。

如果选择的人员只有一个号码，则不需要弹出“选择号码”的界面。现在 sony ericsson 不管联系人是否只有一个号码，也要出现选择号码的界面，真是多余。实际上程序可以判断该联系人是否只有一个号码，需要卖弄的技术应该用在这些地方。

sony ericsson 的做法是直接弹出查找界面，如果是英文界面，是不错的做法，可是在中文界面下，真的要输入名字去查找，很慢。对于通讯录比较少的人，还不如直接到里面去寻找。那么对于通讯录中有几百个人的情况下，怎么办呢？所以，我的做法是：把“查找”功能合并到通讯录中，进去后默认选中它。如果用户要查找，则按一下，再弹出查找界面。如果用户不查找，直接在通讯录中上下选择，也可以。

可是有人会争辩道，我这种做法是强迫用户从通讯录中选择，可是有的人不是使用通讯录，而是直接输入电话号码，或者通过其他方式。是的，这种情况是完全存在的。那么请注意在通讯录上有一个“其他”，点击后，会弹出：

用户可以选择“电话号码”，也可以选择“未存的号码”等方式发送短消息.因为默认是选择“通讯录”，那么用户可以通过“配置”来设定以后默认首先“通讯录”还是别的。

Sony Ericsson T618 的“收件人”确认界面看上去似乎的多余，但仔细研究一下，发现原来它是为了实现同一消息发送多人的需求。

可是这种情况，可以采取别的方式。

1.最简单的模式是 Nokia 8250 的方式，即发送消息后，“编写新消息”界面仍然存在，用户可以继续发送给别的人。用户也可以重新写消息，发送给别人，我觉得这种方式蛮方便的。

2.即便 T618 不采取 Nokia 的模式，因为 T618 已经提供了群组发送功能，所以可以满足部分需求。如果是欲发送的用户不在群组中，用户也可以通过“outbox 中转发。实际上 Sony Ericsson T618 提供的这个增添其他收件人界面，操作起来也不方便。最重要的是，多数情况下，用户往往是只发一个人。为了满足少数情况的需求，而强迫用户每次都这样去操作，是不友好的。

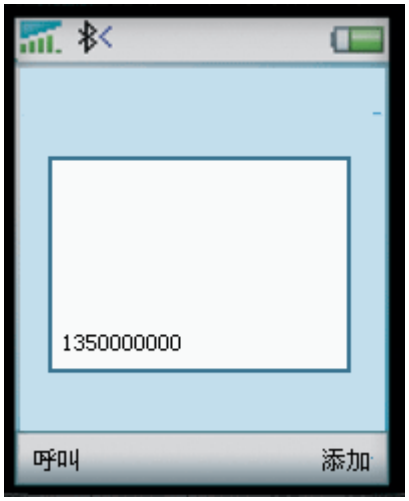
通话清单中，显示的通话纪录，要查看不方便。它的模式是“联系人名＋号码＋通话时间”以一条纪录显示，在一定时间间隔后，交替显示。这样的方式表面上巧妙，但查看不方便。如果要删除整个通话清单，竟然不可以。只能一条一条的删除，真麻烦。其实要删除全部通话纪录还是有地方的，先调出功能菜单,再选择“通话信息”： 再从“通话信息”－“选项”中选择“清除通话清单”，不是“通话信息”－“通话清单”中，把这个功能放在选项中，这种做法太隐蔽，属于菜单分类不合理。更重要的是，对于要“清除全部通话纪录 ”，合理的做法是：按“通话”调出“通话清单”，按“更多” ，在出现的 contextual menu 中，列出“清除全部通话纪录”。T618 清除“短信”就是类似的做法，但对于清除通话纪录，却不采用一致的操作模式。

通话清单中，如果是同一个号码多次拨打，例如一个未接来电拨打了四次，却只显示了最后一次拨打时间，这样的做法隐藏了重要的信息。因为用户有可能需要了解每次拨打的具体时间。一般而言，最新拨入的电话号码应该显示在通话清单的最上面。但是因为 T618 是同一号码多次拨入是记录在同一个号码上，即便它是最新的拨入号码，也可能不是显示在最顶端，这样的做法不合理，削弱了用户的对未接来电的关注。

Sony Ericsson 的另外一个设计特点是，以所谓层次化的方式，给用户一个结构化的 mental map,这是很好的做法。. 然而对于 Power user 来说，这样的做法，有时会显得不方便。Good interface do NOT punish power users.好的设计界面不应该惩罚高级用户。In many case,the very design practices that make things easiest for beginners make things harder for everyone else.Designers will find it very challenging to serve both the user who is just leaning a system and the user who has alrady learned it well.[Larry L. onstaintine].还记得有一次，大家在议论使用向导方式是一种很简单友好的方式，的确有时是这样的，然而如果当用户很熟悉操作的情况下，总是以向导的模式是在惩罚用户。一个用户在一开始可能是初级用户，但当他/她经常使用，逐渐熟悉系统之后，他/她会变成高级用户，这就是 evolve。

Scenario1: 一个人告诉我一个新的电话号码，我想把它存储在 Sony Ericsson 中，要添加一个新的号码到手机中，我该怎么办？常规方式下，Sony Ericsson 只能通过“功能菜单”，选择“通讯录”，再选择“添加联系人”。是的，这种方式是不错的，不过有没有别的方式呢？我设想了一种设计：

当那个人告诉我号码的时候，我不是去寻找该从哪里才能调出添加号码的菜单，而是我直接按号码，当然我不是呼叫，而是点“添加”,然后调出“添加联系人”



Senario2：当有一条短消息来的时候，我当时很忙，没有选择查看。那么后来我要去看的时候，怎么办？我需要点好几次，才能从“短信”的“收件箱”中找到查看。

Senario3：我想写彩信，或者我想写电子邮件，或者添加一个日历中的约会、任务，或者设定闹钟等，我都必需每次通过功能菜单，一级一级地进入后才能操作，操作比较烦琐。

那么针对上述的情况，有没有更简便的操作呢？是否可以通过自定义快捷功能菜单呢？Sony Ericsson 提供了一种添加快捷方式的功能，但不好用，因为我要调出这个所谓的快捷方式，需要按好几次操作杆。

那么可以扩展它现在的“更多菜单”，在里面添加更多常用功能，当然用户可以自己定义这个菜单。那么这样的操作效率会提高。我修改了原先的功能菜单项目，添加了新的项目，如下图：

索尼爱立信的问题还不仅仅只是这些...我也写累了，且住手于此。

本文最初发表于 www.ChinaUI.com，《非程序员》获得授权刊载，版权为 www.ChinaUI.com 所有，谢绝转载。

UMLChina 辅助教材

EXPLORING REQUIREMENTS

QUALITY BEFORE DESIGN

BY DONALD C. GAUSE & GERALD M. WEINBERG

发现什么都不是

而发现的过程就是一切

——《探索需求》

中译本预订>>

为什么伟大的技术不能制造出伟大的产品？

Scott Berkun 著，梁劲 译

查看评论

我们都热爱技术，这正是我们从事这个行业的原因。我们有一个心照不宣的信念，认为技术可以拯救世界。我们精于创造技术并把它们包装成窗口或网站，但我们经常失败于把它们以让我们的顾客容易使用并欣赏的方式集成起来。有时我们会对很难实施或很难创建的东西感到敬畏，却没有对用它们来实现的目标给予尊重。

多年来，我注意到我们对技术的热爱并不总是带领我们处在正确的方向上，在这个专栏力我将尝试去描述这个问题。

玩乐雇佣条款

你的经理可能已经对你表达了一个愿望，做很酷的项目并玩得开心。这不是故事的全部，隐含的真相是所谓的“玩乐雇佣条款”：如果(而且只有如果)你能确保用户对产品有一个愉快的体验，你是被雇佣来玩的。这是因为最终用户支付你的工资——他们支付我们所有的工资，这意味着你所做的一切应该最终让用户获得好处，你的公司也因此受益。你想要去扩大你公司的目标与用户的目标之间的交集。

你写的每一行代码，你发现的每一个 bug，你所做的任何市场调研都应该以某种形式帮助用户。无论你的工作多么不明显或间接，你还是被“玩乐雇佣条款”所约束。例如，你修补一个设备驱动程序，改进可靠性，或优化服务器性能将使产品变得让用户感到更好。做市场研究能够帮助产品定位于合适的对象，从而可以满足他们的需求。如果你没有能把你的工作与它是怎样帮助客户联系起来，把你的时间花在别的地方吧。你越是经常从最终用户的角度来思考你的工作，思考你的团队的工作，你越有可能帮助去创造一个伟大的产品。

我们不是我们的用户

在这个行业中我们形成了类似的思维方式。在大部分时间我们是和那些在数学测试中获得高分的人一起渡过，我们认识那些有股票期权的人，我们和那些能从拆卸计算机中获得乐趣的人在一起工作。我们忘记了在我们这个行业中的人跟世界上的其他人是非常不一样的。这正是为什么进入易用性实验室似乎就象一次进入模糊地带的旅行， 似乎那些用户是少数，是从某个扭曲和更缓慢的宇宙来参观我们。现实是：我们是压倒性的少数，那些在易用性实验室的参观者才是多数派，他们是使用我们产品并支付我们薪水的人。

并没有其他方式可以替代观看某个人使用你创造的东西，这是唯一的方式可以验证你设想的目标是否和现实相匹配。你愿意一个外科医生在手术之前和手术之后都不对你进行检查吗？你希望一个建筑合同商在没有和你讨论计划且确保你能得到你所要的情况下就重新装修你的厨房吗？好的工匠在制造产品之前愿意首先考虑产品将如何被使用。我们具有惊人的创造能力，也容易掉入陷阱，掉入所创造的产品只吸引我们自己而不是吸引客户这种陷阱。除了通过易用性测试和其他形式的客户反馈，并没有其他方式可以了解你有多么偏离方向。你必须在整个产品周期花时间与用户沟通，反复更新开发团队对于他们正在做什么以及是在为谁做的看法。

什么为主导？用户还是技术？

技术专家和真正的设计人员在对于怎样做产品的方式上有根本的差异。技术人员趋向于以技术开始。我们让开发团队开发一项技术，然后硬塞进一个用户界面，对整个框架的用户体验（user experience）完全被技术所支配。这必然导致用户体验将是一个糟糕的妥协，产品将不是被设计为用户使用的产品，而是被设计为一堆技术的载体。我们的所作所为显得好像技术比产品更重要。伟大的产品是以这种方式设计的吗？ 大师级的厨师直到最后一分钟才弄清食物对顾客是什么一种味道吗？裁缝是在衣服都被缝好以后才去量顾客的尺寸吗？

任何行业中的好工匠都明白用户将会使用他们的作品，他们将考虑用户之所想来做出每一个决定。软件或 Web 开发并无不同。去餐厅吃饭或去电影院看电影的人和使用我们产品的人是同样的人。我们需要培养一种兴趣，即去了解其他领域中的产品是如何开发的，他们是如何实现所需要的效果。汽车、CD 播放器、器具等的制造者，除了他们做那个行业比我们做软件的时间更长外，都会面临和我们同样的挑战，需要在工程、商务、易用性之间实现平衡。通过了解他们开发方式的差异，我们可以从他们的成功和失败中学到很多。

为什么简单的产品是伟大的产品？

最强有力的工程技艺是那种我们不会留意到的技术。工程师和开发人员的真正力量体现在把非常复杂的东西转换为难以置信的简单事物。汽车中的自动传动系统显著地具有比手动传动系统更多的工程技术。汽车工业、城市建筑、消费类电子产品中的最佳作品都体现了伟大的工程是如何聚焦于隐藏复杂性而不是显现它们。

增加产品价值的最佳方式是增加它的功能却没有增加复杂性。当你想增加一个新的 feature，是否有办法增加了它却没有为之增加一个用户界面？它能被可靠地自动化吗？或者，为了包含新的功能，是否可以通过新的、改进过的方式替代旧的，来调整、甚至去掉原先的功能？想一下汽车工业是怎么做的，想一下他们怎样以对用户影响最小的方式来增加重要的 feature。就象动力方向盘是对于普通方向盘的一个新添加功能一样，防锁死刹车系统是对标准刹车踏板 UI 的补充。要获得新特性的好处，司机不需要训练或重新学习。这种设计努力——使复杂的特性让用户看起来很简单——制造出了伟大的产品。

软件作为服务的真正含义

当你走进一家运动用品商店，询问一个关于背包的问题，你期待得到尊重，你希望售货员按你的水平和你交谈，处理你的问题，以礼貌和公平的方式解决你的问题。软件用户或 web 用户也是一样，他们期待被尊重地对待并得到优质服务。客户或顾客总是正确的。

当我们把出现的一个错误信息认为是用户的错误而不是开发人员的错误时，我们犯了一个严重的错误。如果一个用户试图从一个网站上购物，出现了服务器数据库问题，这是谁的错？是我们的错误！我们没有足够聪明到去确保用户从不遇到这样的问题。要么是因为项目经理和设计师没能创建适当的界面设计，要么是因为开发、测试组没有能发现重大的缺陷。网站上的错误信息比用户在软件中遇到的错误信息更糟。当我们的产品中出现错误的时候，我们提供了礼貌和有帮助的支持了吗？我们是否以顾客至上的方式对待客户了？几乎从来没有。通常我们是以这样的错误信息做出回应：*“Server error 152432. Scripting service failure”*。 每一条错误信息意味着一个陷入麻烦中的用户。想像你的用户，坐在那里，会议迟到，感到沮丧，只是因为他们没有完成他们非常想完成的事情。你希望让你的用户在那一刻看到什么？他们应该得到哪种服务？你放到产品中的每一个错误信息是一个提供良好服务的机会。如果你想把提供优质服务作为你的网站或产品的一部分的话，你必须把错误信息和错误处理规划在你的 schedule 中。项目经理应该总是把错误信息处理作为一个任务添加到开发、测试团队的 schedule 中，但是记住：没有什么伟大的错误信息。一个伟大的错误是一个通过一流的产品设计和错误处理代码所消除的错误。

超越只提供错误信息的服务不是去责备用户，而是向用户提供很大的帮助。在用户的体验过程中，可以有很多机会提供给用户好的服务。观看某人使用你的网站的主要功能，自问它能否达到在一家好的商店或餐厅中得到的服务水平，一个好的服务员知道何时中断你，何时让你一个人呆在哪里，知道怎样以彬彬有礼的方式服务你。你的网站或软件质量越是能接近用户日常生活中得到的良好服务水平，你将越是接近于拥有一个伟大的产品。

投入时间去创造伟大的产品

把什么都做好是困难的，写好代码比写差代码花更多的时间。如果你团队的管理部门是致力于制造伟大的产品，那么他们应该采取措施去调整团队的目标和 schedule 以便符合能做出伟大产品的现实。这是他们的工作，如果他们没有能那么做，让他们知道这些就是你的工作。如果问题是在于你想了解怎样把 UI 交互性设计或易用性集成到开发流程中，那么去询问团队中从事设计和易用性测试工作的人员。好的产品设计来自好的流程，很多团队还没有认识到这点。在很多情况下，在易用性工程（usability engineering）上投资最终能节约时间和金钱，因为你开始就设计得很好，而不是在产品发布之后试图去打补丁。

领导得很好的团队会让每个人都明白他们个人的所作所为作是怎样影响客户的。针对用户所面临的问题，针对功能和技术怎么去解决用户的问题，在产品开发流程开始之前，应该把产品管理、产品计划和易用性测试等包含在一个构架里。没有一个构架，你只能是在构建一种不解决任何问题的技术。你应该每天问自己你正在解决什么样的问题。它是谁的问题，投资你的时间在里面是否合理。

如果你的目标是让产品易用而且你知道怎么才能做到，那么你应该安排你的项目的进度日程以确保它的实现。说“我没有时间”去开发产品的一个关键性方面总是一种逃避，它真的意味着你的团队没有很好地计划或者目标和计划没有很好地匹配。如果你把用户对产品的体验作为一个低优先级的事项，那么也许你应该重新评估你的项目优先级。如果你相信什么事是重要的，你可以为其制定计划日程。

伟大来自何方？

通过变成一个用户至上的鼓吹者，可以你帮助做出一个伟大的产品。我坚信是整个团队都重视并致力于用户才是产生重大差别的原因。把这篇文章发给你团队中的同事，去进行易用性测试，如果团队中没有人了解该怎么做的话，学习怎么去做。和产品设计师讨论什么地方出错了，询问你的易用性工程师他们是怎么从事工作的，询问你怎样才可以支援他们。你的正式工作头衔并不重要，用户支付你的薪水跟你做什么无关。如果你从事 UI 方面的工作.帮助你团队中的其他人拓展他们的视野，邀请他们参与。当你读到关于设计方面的好文章或好书，把它传递给那些最需要他们的人。

WILEY

TIMELY. PRACTICAL. RELIABLE.

UMLChina 指定教材

Agile Database Techniques

Effective Strategies for the Agile Software Developer

《敏捷数据》

UMLChina 李巍 译

Scott Ambler

用 UML 进行超媒体建模

Peter Dolog、Mária Bieliková 著，车皓阳 译

查看评论

摘要

这篇文章讨论了使用 UML 进行超媒体建模的一种方法。我们构建了超媒体系统的四种视图：应用领域模型、导航模型、外观模型和用户模型，然后在它们之上对超媒体模型进行了描述。这种方法强调使用 UML 状态图作为主要技术，用于导航建模。我们把这种方法和其它几类模型联系起来。同时针对用户模型，也给出了适应性建模的方法。

关键词

超媒体建模 行为模型 UML 个性化超媒体

1 简介

本文给出了使用 UML 对超媒体应用程序建模（Hypermedia application modelling）的一种方法。超媒体应用程序可以看成为特定领域提供信息的一个系统。而外观信息可以划分成由活动链接互连的块。为了提供超媒体程序的综合模型，现在已经提出了好几种建模观点[4]。主流观点认为它与解决环境的特征相关，也即，如何解决“超媒体应用程序中要对什么进行建模？”的问题。根据这种观点，我提出了以下的分类：

- 应用领域模型：利用特定领域中概念之间的相互关系进行合并。这种模型确定了超媒体应用程序提供的信息。
- 导航模型：定义了应用程序在信息网络中导航用户的方式。导航模型把信息块分组为上下文，并以链接相连。
- 外观模型：与为用户提供的信息块的外观相关。它也定义了空间布局以及与用户界面相关的信息块的内容。外观模型定义了外观类或对象，它们之间的空间关系，以及相关的内容。

为了支持个性化，我又加入了一个用户模型。用户模型确定用户的角色和功能，对于适应性和个性化来说这是很重要的。

上面提到的视图要从需求到实现逐步求精。

文献中已经出现过好几种超媒体建模的方法。大部分在建模过程中，考虑的是上面提到的视图。区别主要在建模技术和在超媒体应用建模过程中使用的相应标记上（结构的或行为的）。

超媒体应用程序是一种交互系统，其中应用程序的反应行为（reactive behaviour）在导航和外观视图中尤为重要。现有的方法大量采用结构模型，使用 E-R 图（例如，超媒体设计方法（HDM）[11]，关系管理方法（RMM）[14]，Web 建模语言（WebML）[2]），或类图（例如，UHML（Nora Koch 做的扩展 UML）[12]，W2000（HDM 的后继）[1]，面向对象的 HDM（OOHDM）[15]）。象 Petri 网和状态图等行为技术的作用都被低估了，没有广泛使用。例如[10]中定义的基于 Petri 网的模型。但是，我们并没有发现任何设计方法包含它。OOHDM[15]提到了导航转换模型，它由 Harel 状态图表示。这与基于状态图方法的超媒体模型（HMBS/M）[3]很相近，在其中，对内容和导航实例模式的建模是通过状态图完成的。

我们提出的 UML 状态图基于[5]中的导航建模方法和[6]中的适应性导航建模方法。我们相信，行为模型在所有方面（而不仅仅是导航）对超媒体建模都很重要。另一方面，结构模型也是一种有效的手段，可以表示超媒体系统上的特定视图。本文的目标是讲述这种建模方法，基于导航模型的行为状态图在结构模型之外也起着重要的作用。

本文后续部分是这样组织的：第二节描述应用领域建模，第三节描述导航建模。在第四节中，我们讨论了如何使用 UML 表达用户模型。后面，在第五节里讨论了外观模型。最后是结论和未来要进行的工作。

2 应用领域建模

在超媒体建模中，所有的已知方法都从某种应用领域建模开始。应用领域建模的主要元素是一个概念。概念通过关联、泛化和聚合等关系建立起联系。概念可以拥有自己的特征。

OOHDM、HMBS/M、W2000 和 UHML 都把应用领域建模看成概念建模，并通过类图表示领域模型。W2000 使用 UML 类图，并把应用领域建模命名为信息结构设计。HDM 和 WebML 使用 E-R 图进行领域建模。RMM 把应用领域模型看成关系管理数据模型，并使用扩展实体关系图来建模。

我们采用 UML 图在应用领域建模中进行概念建模。我们也推荐使用活动图，在应用领域中针对活动建模。在 UML 中使用构造型类元素 *Concept* 表示概念。功能被表示成属性或方法、算符和过程。基本结构关系包括泛化/特化，包容或聚合以及普通的关联。包是表示包容的另一种机制。

图 1 给出了应用领域模型的一个简单示例。要建模的应用是从软件工程的子领域——需求工程提供信息。为了限制空间，我们把概念限定在高层抽象上，仅关注部分领域。关系的基数在这种上下文中并不重要。需求工程领域是从需求工程过程的观点来建模的。需求工程过程（Requirements Engineering Process）聚合那些表示需求工程活动的概念。需求获取（Requirement Elicitation）的概念与需求源（Requirement Sources）和获取技术（Elicitation Techniques）相关。需求工程过程可以由不同的过程模型（Process Models）表示。每种过程模型按照不同的方式管理和支持，它们不同程度地影响需求工程的质量。这些模型的活动由不同的过程使用者执行。

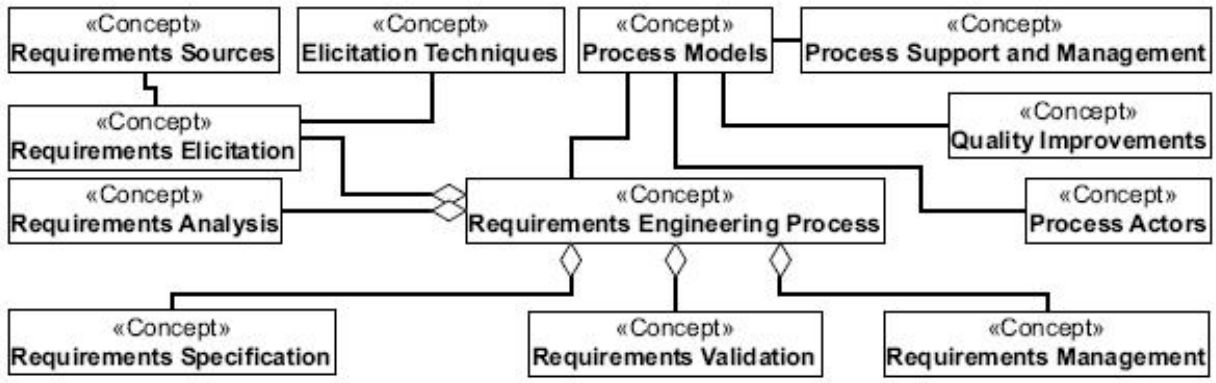


图 1 基于[13]的需求工程的概念模型

对应用领域模型求精会生成超媒体应用程序的数据库方案。然后，这些内容就被直接分配给数据库方案。

应用领域的行为模型也可以建立起来。这个模型由领域相关的活动组成。UML 活动图的用途就在于此。当超媒体应用程序需要表示某些活动的信息时，活动模型就会很起作用。这个模型是导航的基础，比如，这些信息是按照活动的顺序表达的。活动可以从概念视图中取得，这时它们会被表示为概念。

3 导航建模

在超媒体应用程序中，导航反应了在大量信息之中用户定位（局部或全局）问题。合理地把信息块组合为信息组件，并在它们之间建立起充分的联系，就可以减少导航问题（在超空间中迷失）。创建导航模型，也即把相关信息编组为节点，通过链接建立节点间的联系，这将会大大支持超媒体应用软件的开发过程。相关信息的组合可以通过在概念模型上执行查询来完成。这需要概念模型上某种查询语言的支持。另一种可能性是把部分概念模型映射成导航模型。一种特殊类型的映射是实例化。

在简介中提到的超媒体建模方法采用了不同的导航建模技术。例如，OOHDM 区分两种导航模型：导航类模式和导航上下文方案（context schema）。两者都采用结构技术进行建模。在导航类模式中，节点和链接被当做类的构造型和关联。节点从概念模式的角度将相关的类进行分组，并使用链接将它们互连起来。导航上下文模式使用特殊标记表达，它们定义了导航类的访问原语和集合。UHML 和 W2000 执行相似的导航建模，但它们使用的是 UML 或是自己的标记。WebML 把导航视图当成超文本模型，由组合模型（composition model）与导航模型组成。组合模型是上下文模式的对等体。导航模型定义了组件之间的相互连接关系。RMM 和 HDM 使用分片定义上下文和实体观察。它们提供了好几种访问结构建模原语，例如索引，导游（guided tour）等等。

我们的方法是把领域模型类图映射为导航模型。导航模型由 UML 状态图表示。目的是想通过概念模型中出现的概念对导航进行建模。概念以及它们的相互关系被转换或映射为状态和转换。这里，状态图的用法和以前 UML 状态图的语义稍微有些不同，在以前的 UML 中，有类、用例、操作、子系统、执行者或方法。映射的详细内容可以在[6]中找到。事件触发状态转换，大多数事件是作为用户与超媒体系统交互的结果出现的。相应地，创建导航模型的第一步是分析高层交互机制。这是由顺序图建模的（在[5，6]中可以找到示例）。

图 2 是一个状态图的示例。这张图由三部分组成：需求工程概观（Requirement engineering overview）、获取技术（Elicitation techniques）和支持功能（Supporting Features）。需求工程概括起来包括四个主题：简介（Introduction）、主要活动组合（Compositions of main activities）、用户以及由状态表示的需求工程。导航（也称作浏览语义[5]，browsing semantics）是由下一链接确定的。我们使用历史标记（H）表示用户在退出系统之前进入同一状态。

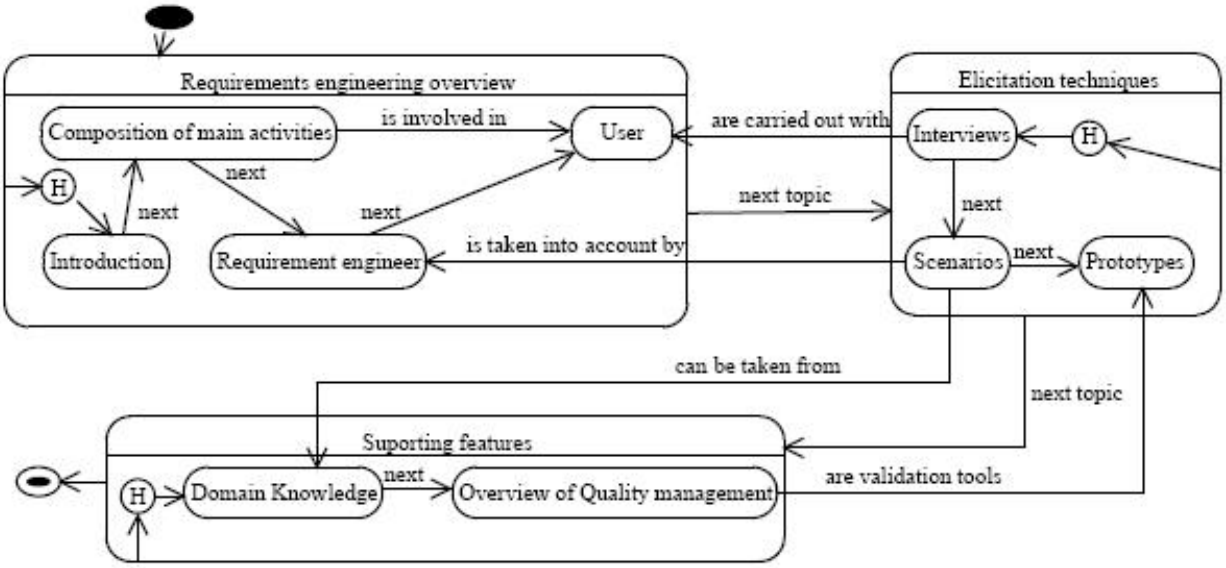


图 2 需求工程导航模型的一个示例

4 用户建模

信息要呈现给不同的用户使用。用户有不同的目标、偏好、相应概念或相应概念集合的知识水平。现有的用户建模布署方法仅有几种。大致是因为个性化或适应性超媒体是个相对较新的领域。比方说，适应性规则可以用 WebML 或扩展 RMM 定义。用户模型可以用类图在 UHML 中布署。

用户模型包括用户的不同特征：用户的目标/任务、兴趣、知识、背景、超空间使用经验和偏好。如何表达这些特征，有好几种可能的情况。我们把 De Bra 等人[16]开发的适应性超媒体建模中的用户模型作为一个例子。中间使用类图（见图 3）表示用户模型的结构特征。在每个概念中，最详细的知识描述是针对每个用户度量的。但是，并不总是会有这么精细的度量。在这种情况下，知识会在较高层次上进行度量。

有时候，结构视图并不足以建模跟踪用户的知识水平、目标和偏好。我们建议在一个有着多种关系的复杂领域内，采用行为模型表达用户模型记录。表达知识状态、水平和它们之间的转换（例如，当偏好或目标发生变化时），我们用的是状态机模型。

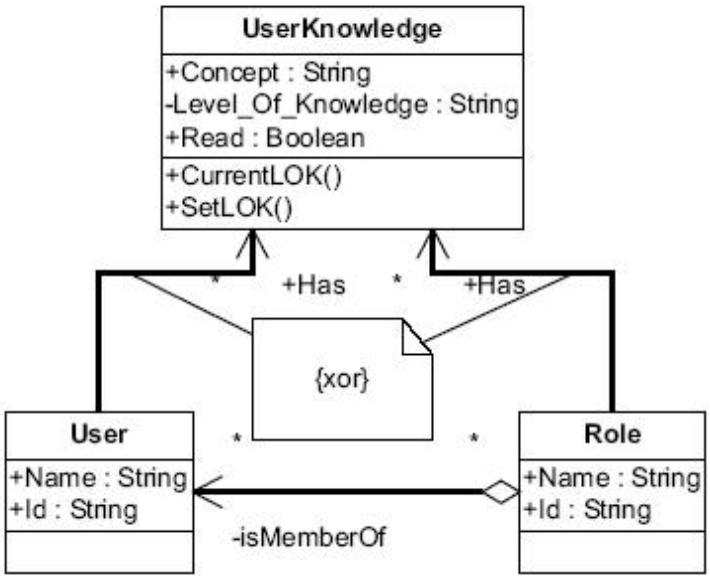


图 3 电子授课用户模型的示例

5 外观建模

外观模型应该反映下面的特性：信息呈现形式（字体风格，颜色，背景色，等等）；在系统界面内整理信息块；用户与系统交互带来的界面变化方式。与用户模型或隐含模型相关，适应性外观建模基于适应规则，但它们并不依赖用户模型。

外观模型大多都是由结构技术表示的。例如，OOHDM 布署高级数据视图；WebML 使用样式表来进行外观建模；UHML 定义了一些类的特殊构造型。在[9]中，RMM 由外观图扩展。这个图把超空间表示为节点，由三种链接组成：空间（spatial）、时间（temporal）和导航（navigational）。

为了进行外观建模，在导航模型中我们定义了状态机，它是通过合并多余（可选的或并行的）状态而生成的，它表示不同形式的外观信息。规则根据用户知识水平表示适应性，规则由监视哨（guards）、条件、行为状态和转换来建模。例如，在图 2 的导航模型中，外观模型采用两种可能的呈现状态扩展了领域知识状态（Domain Knowledge state）：定义（definition）和细节描述（detail description），它们根据用户对概念的掌握知识水平来显示（参见图 4）。

到细节描述状态的转换由状态和相应行为标记：

[*UserModel:CurrentConcept:CurrentLOK*() *<>0 low0*]=*Display*()

下一状态在离开组合状态后，设置用户知识水平。

nexttopic[*UserModel:CurrentConcept:CurrentLOK*() <>*0 low0*]/*Display*();
UserModel:CurrentConcept:SetLOK()

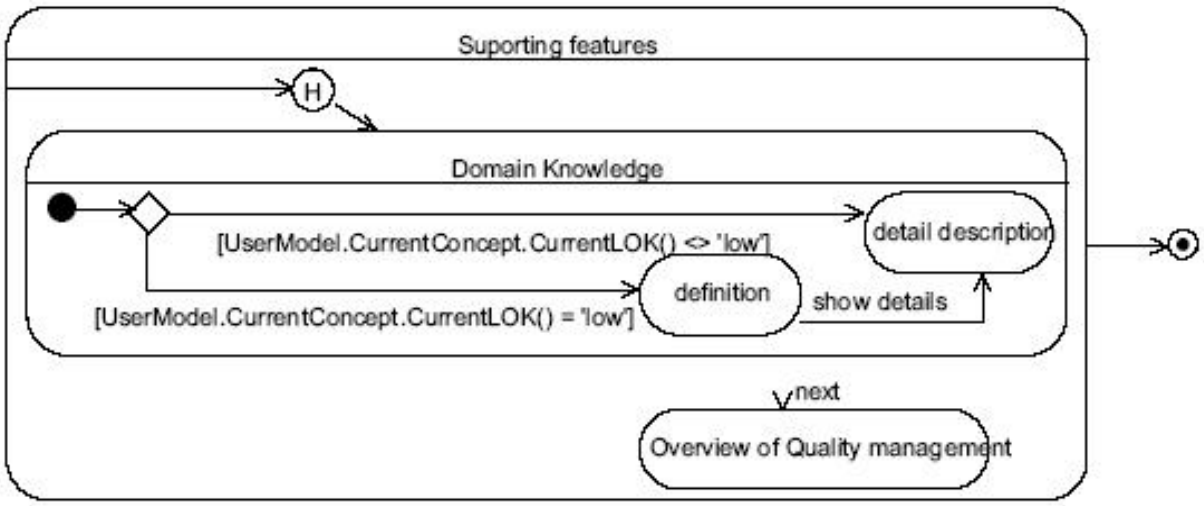


图 4 电子授课中适应性导航模型的一部分

这个转换由三个参数标记：接收事件（received event）、监视哨和效果（effect，行为表示）。第一部分是下一主题事件。与用户当前知识水平相关的监视哨状态后面后跟着两个事件序列，以显示相关知识，并设置新的用户知识水平。外观模型使用类 UserKnowledge 的行为功能 CurrentLOK()和 SetLOK()，与用户模型相互连接。

超媒体系统涉及到不同的媒体类型，这是它的一个特殊性质。当并行呈现的媒体类型显示出相关信息时，必须保持同步。在我们的方法中，我们通过组合状态对通道建模[7]。并行同步媒体通道由并行状态建模，为了同步通道要使用同步标志（同步或创建并加入伪元类）。所有提到的建模元素都要包含在 UML 状态机包中。

类或对象的静态外观特征可以通过增加外观结构功能来建模。另一种可能是为外观模板创建新的类构造型。然后这些模板可以映射成特殊状态。我们在系统中提出并实现了这样的方法，用于在网上创建有结构的文档和它们的注解。

6 结论

我们描述了使用 UML 对超媒体系统建模的可能性。我们相信状态机与结构建模技术一样，也会更好地适合个性化或适应性超媒体应用程序建模。状态转换在语义上与用户界面信息变化之上的交互更为贴近。用于导航建模的结构技术使用关联或构造型。这种关系类型可以说是用以系统静态因素的建模。而且，适应性会给导航带来更好的动态特性。

尽管如此，导航建模的静态因素还是很有用的。正如我们提到的那样，状态图仅在 HMBS/M 和 OOHDM 之中得到应用，但也是在实例层次上。我们允许把类映射为状态，关系映射为转换或转换的分解。除此之外，OOHDM 和基于它的一些方针也推荐仅使用结构技术，这样仅会导出一种“缺省浏览语义”。但是，这对于交互式适应化超媒体应用建模来说并不够，在这里对于不同的用户要考虑到不同浏览方式的可能性。基于状态图，对导航模型的直接求精，可以作为外观模型（内容的另一种选择）使用。

进一步给出的建模技术也被用在不同的现有超媒体开发方法之中。本文提出的导航建模方法需要技术和导航模型之间不同的映射规则。图 5 描述了模型之间的高层依赖。这些依赖关系可以看成映射的基础。我们现在的工作是，在从分析到实现过程中，对模型转换进行准确的定义。

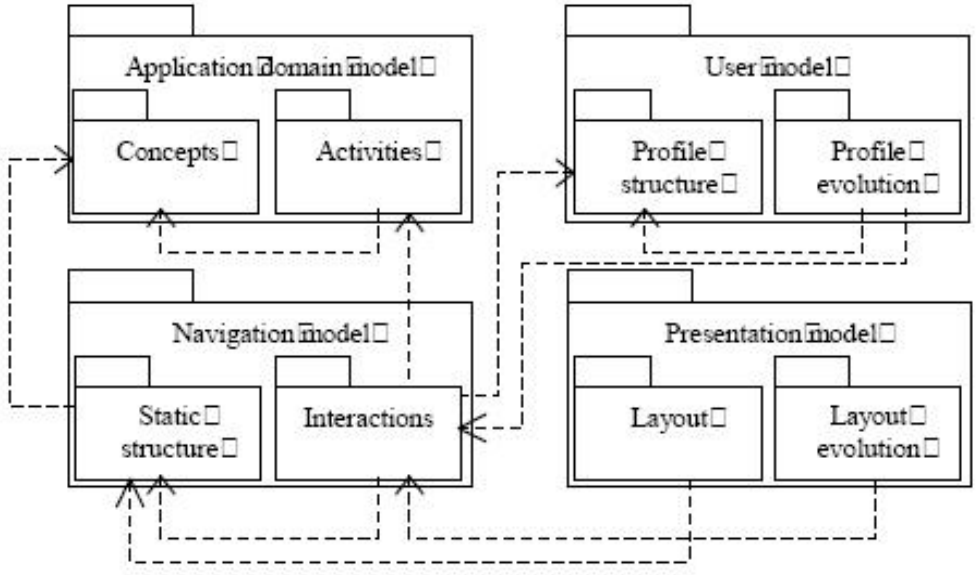


图 5 超媒体模型之间的高层依赖

参考文献

1. L. Baresi, F. Garzotto, and P. Paolini. Extending UML for modeling web applications. In *Proc. of 34th Anual Hawaii International Conference on System Sciences (HICSS'34)*, Maui, Hawai, Jan. 2001. IEEE Press.

2. S. Ceri, P. Fraternali, and A. Bongio. Web Modeling Language (WebML): a modeling language for designing web sites. *Computer Networks and ISDN Systems*, 33(1–6):137–157, June 2000.

3. M. R. de Carvalho, M. C. F. de Oliveira, and P. C. Masiero. HMBS/M - an object oriented method for hypermedia design. In *Proc. of V Brazilian Symposium on Multimedia and Hypermedia Systems - SBMIDIA'99*, pages 43–62, Goiˆania, June 1999.

4. P. Dolog. Modelling in hypermedia development, Aug. 2001. Technical Report (A Written Part of PhD Examination). Department of Computer Science and Engineering, Slovak University of Technology.

5. P. Dolog and M. Bielikov´a. Modelling browsing semantics in hypertexts using UML. In J. Zendulka, editor, *Proc. of ISM'2001 - Information Systems Modelling*, pages 181–188, Hradec nad Moravic´ı, Czech Republic, May 2001.

6. P. Dolog and M. Bielikov´a. Navigation modelling in adaptive hypermedia. In *Proc. of 2nd Conference on Adaptive Hypermedia and Adaptive Web-based Systems*, Malaga, Spain, May 2002. Springer LNCS.

7. R. ˇCervenka. Modelling of hypermedia applications. In B. Rovan, editor, *Proc. of SOFSEM' 98 — 25th Conference on Current Trends in Theory and Practise of Informatics*, pages 294–303, Czech Republic, Nov. 1998. Springer LNCS 1251.

8. P. Faˇsung. Annotating structured documents on the web, Dec. 2001. Diploma Project supervised by Peter Dolog. Dept. of Computer Sci. and Eng., Slovak University of Technology.

9. F. Frasincar, G. J. Houben, and R. Vdovjak. A RMM-based methodology for hypermedia presentation design. In A. Caplinskas and J. Eder, editors, *Proc. of ADBIS 2001 -Advances in Databases and Information Systems*, pages 323–337, Vilnius, Lithuania, Sept. 2001. Springer, LNCS 2151.

10. R. Furuta and P. D. Stotts. A formally-defined hypertextual basis for integrating task and information, 1994. Tech. Report TAMU-HRL 94-007.

11. F. Garzotto and P. Paolini. HDM — a model-based approach to hypertext application design. *ACM Transactions on Information Systems*, 11(1):1–26, Jan. 1993.

12. R. Hennicker and N. Koch. A UML-based methodology for hypermedia design. In S. S. A. Evans and B. Selic, editors, *Proc. of UML 2000 Conference*, York, England, Oct. 2000. Springer LNCS 1939.

13. IEEE. Guide to software engineering body of knowledge v. 0.95, 2001.

14. T. Isakowitz, A. Kamis, and M. Koufaris. Extending the capabilities of RMM: Russians dolls and hypertext. In *Proceedings of the 30th Annual Hawaii International Conference on System Sciences*, Jan. 1997.

15. D. Schwabe and G. Rossi. An object-oriented approach to web-based application design. *Theory and Practise of Object Systems (TAPOS), Special Issue on the Internet*, 4(4):207–225, Oct. 1998.

16. H. Wu, G. J. Houben, and P. D. Bra. User modeling in adaptive hypermedia applications. In *Proc. of the Interdisciplinaire Conferentie Informatiewetenschap*, pages 10–21, 1999.

Smiling小组
名称：UMLCHINA
E-mail: umlchina@smiling.com.cn
描述：专门讨论UML/OO应用相关细节
组长：umlchina mouri sealw
成员：40581人
记数：3794085次　　小组积分：919078

The Addison Wesley Signature Series

A MARTIN FOWLER SIGNATURE BOOK

PATTERNS OF ENTERPRISE APPLICATION ARCHITECTURE

中译本即将上市

MARTIN FOWLER

WITH CONTRIBUTIONS BY
DAVID RICE,
MATTHEW FOEMMEL,
EDWARD HIEATT,
ROBERT MEE, AND
RANDY STAFFORD

UMLChina 负责审校，并指定为训练用教材

Agile软件开发丛书

PEARSON Addison Wesley

有效用例模式

Patterns for Effective Use Cases

Foreword by Craig Larman

[美] Steve Adolph
Paul Bramble 著

车立红 译

UMLChina 审

UMLChina 指定教材

清华大学出版社