第 44 期（2004 年 12 月）

设计模式精解

X 非程序员
-Programmer
软 件 以 用 为 本

联系：think@umlchina.com

http://www.umlchina.com/

DMTF 和 OMG 结盟用 UML 和 XMI 来打造 CIM

[2004/12/13]

DMTF(The Distributed Management Task Force) 是领导面向企业和 Internet 环境的管理标准和集成技术的行业组织。OMG（the Object Management Group(TM)）是跨多种操作系统、编程语言、中间件和网络底层架构以及软件开发环境的标准制定组织。今天两家组织宣布结盟，按照约定，两家组织将使用 OMG 的 UML 和 XMI 对 DMTF 的公共信息模型 CIM（Common Information Model）进行表示和交换，以促进 OMG 的模型驱动架构 MDA 的应用开发（包括分布式管理的应用）的标准化工具集在业内的广泛应用。

两家的结盟首先关注于建立将 CIM 规约映射到 XMI 的机制，以获得在 UML 工具中编辑结果模型的能力，通过流程的简化为开发人员和最终用户节约时间。另外，两家组织还将开发一个 UML profile for CIM，以简化 CIM 开发者在广泛应用的 UML 建模环境中创建 CIM 模型的工作。

DMTF 总裁 Winston Bumpus 指出，“这两家业内重要的标准化组织的合作将给开发人员和最终用户带来巨大的好处。随着分布式管理技术方面开发人员对 UML 标准的广泛应用，基于 UML 的工具将促进 CIM 的广泛被接纳。这些将给行业带来显著的进步，帮助用户在开发管理应用时利用已有的应用开发工具。”

“这个内容涉及面很广的合作协议将加强业界对模型驱动架构 MDA 以及 DMTF 的关键的管理标准的支持。通过使用 OMG 的 UML、XMI 以及相关技术支持 CIM 的表示和交换，这项合作将使得基于 CIM 的模型设计可以利用丰富的 UML 工具，有开源的工具，也有开发商提供的各种工具。”，OMG 的主席和 CEO，Richard Soley，指出，“这将帮助整个行业的最终用户降低在工具上的成本支出，对于和 DMTF 在这项重要工作上的合作机会，我们非常欢迎。”

关于 DMTF

DMTF（the Distributed Management Task Force）有超过 3000 的积极参与者（active participants），是领导面向企业和 Internet 环境的管理标准和集成技术的行业组织。DMTF 的标准集提供公共管理底层架构组件（common management infrastructure component），包括应用（instrumentation）、控制（control）和交流（communication）等方面，以平台无关和技术中立的方式提供。DMTF 技术包括信息模型（CIM）、交流/控制协议（WBEM）以及核心的管理服务。相关信息可以访问 www.dmtf.org。

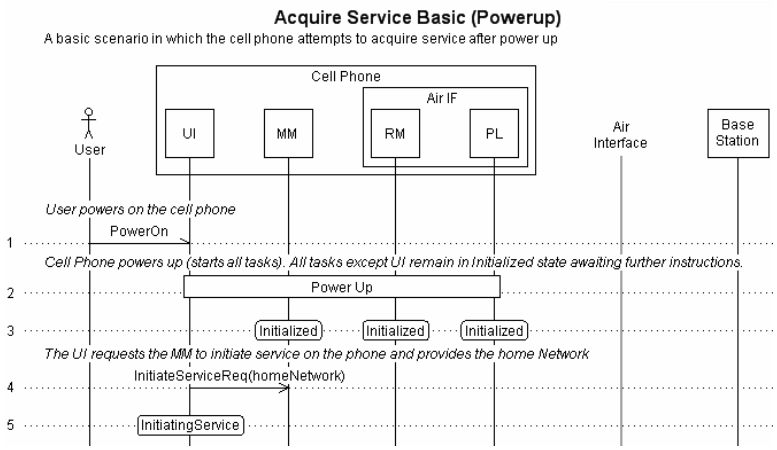
（自 businesswire，UMLChina 袁峰 摘译，不得转载用于商业用途）

新工具简化 UML 顺序图和 call flow 图的绘制

[2004/12/10]

Effexis 软件公司推出面向软件工程师、系统工程师的建模工具：顺序图编辑器，该工具适合于电信/无线软件，网络协议以及复杂系统的设计。

Effexis 软件公司今天宣布建模工具 Sequence Diagram Editor 的正式发售。这是为简化顺序图和 call flow 图的设计和维护而提供的一款建模工具。Effexis 的解决方案将自动处理图形的布局和格式（可以处理多页的情况）。和使用通常的绘图程序相比，开发人员可以用更快地速度开发复杂的顺序图。



Effexis 总裁 Rodger Constandse 说，“对于复杂软件系统的设计，尤其是在电信和无线行业，顺序图和 call flow 图是非常有用的建模工具。在我的实践经历中，过去由于工具的限制，这些图的绘制和维护都非常麻烦耗时，这也导致软件工程师和系统工程师往往没有很好地利用这两种图。因此我们设计了 Sequence Diagram Editor 来自动化处理布局和格式，帮助开发人员更方便地使用这两种图，而不用去关心线条和方框，在发生变化的时候也不用手工去移动。”

Sequence Diagram Editor 支持状态（state）、动作（action）、记时器（timer）和场景（scenario）等元素，以方便系统级设计和 MSC（Message Sequence Chart）中电信/无线的 call flow 图和交互图的绘制。一旦这些图完成，用户就可以按各种尺寸打印出来，或者输出为 RTF 或 PDF 文件格式。

Sequence Diagram Editor 是一个 Windows 应用程序，在 http://www.effexis.com/sde/index.htm 可以下载到 14 天免费使用版本。

（自 emediawire，UMLChina 袁峰 摘译，不得转载用于商业用途）

Ivar Jacobson 中国行，在 CSDN 做交流

[2004/12/8]

12 月 8 日下午 14:00－－16:00，Ivar Jacobson博士作客CSDN的嘉宾聊天室与大家展开面对面的思想和技术交流，参与聊天的网友近千人次，聊天实录下载>>。

Ivar Jacobson 在聊天现场

Ivar Jacobson 拿着笔者送的贺卡

以上图片均来自 CSDN。

（think 文）

IRIS2.0 自动化 RUP 过程

[2004/12/6]

RUP（Rational Unified Process）的顾客现在可以对他们的软件开发过程进行简化的过程裁减和执行了，这将增大他们的投资回报率 ROI（return-on-investment）。

今天，Osellus 公司在 SD 过程会议上宣布了该公司的 IRIS 过程自动化系统。Osellus 是在软件过程自动化系统提供商方面的佼佼者，其 IRIS 系统面向于 IBM 公司的 RUP 过程的裁减和部署。

Osellus 的 COO，Aditya Jha 指出，“RUP 的用户将从这一软件开发方法学蕴含的广泛的最佳实践中长期受益。但有效工具的缺乏妨碍了 RUP 的裁减和执行。IRIS2.0 正是解决这一需要的基于标准的解决方案。”

IRIS2.0 包括了 Osellus 的 RUP 桥接（RUP-Bridge）技术，可以将 RUP 的所有内容导入到 IRIS 中，保存到符合 SPEM（Software Process Engineering Metamodel）的仓库之中。这个解决方案保护所有已经安装的 RUP Plug-in 的内容。Osellus 的 CTO，Payman Hodaie，指出：“RUP-Bridge 使得 RUP 用户可以剪裁、发布和执行他们的 RUP 过程，而不需要重新输入任何数据。我们的解决方案不需要顾客了解那么多的 UML 知识，或者对复杂的软件建模环境的掌握，例如 Rational XDE”。

简化 RUP 的剪裁

IRIS 2.0 允许过程制订者快速地在 RUP 方法论的基础上进行修改，以创建适应组织的不同项目类型的剪裁后的版本。项目主管可以剪裁过程以创建项目特定的 RUP 过程。IRIS2.0 提供的剪裁能力包括对内容以及支持的模型的修改。

使用 IRIS2.0 剪裁 RUP 过程并不需要特别的 UML 或者其它建模技能。剪裁环境是完全基于 Web 的，并完全遵循 OMG 的 SPEM 标准。

过程制订者可以使用 IRIS2.0 将 RUP 和其它商业的、公开的或者内容的方法论相结合。IRIS2.0 系统可以用来实现 RUP 和其它过程改进框架的整合，诸如 CMM、ISO 和六 Sigma。

执行（Streamlined Enactment）

到目前位置，RUP 的实施或者实时执行，都是必须手工来进行的。手工的方式容易产生错误（error prone）、并且需要大量资源，在实践中难以被接受，结果就是大量的 RUP 部署工具以失败告终。

IRIS2.0 通过在每一个剪裁后的 RUP 过程的项目实例中对 RUP 中阶段（phases）、循环（iterations）、活动、角色、工件和指南进行实时管理，显著地提高了 RUP 的投资回报率。

另外，项目主管、项目组以及质量保证（quality assurance）组织都可以用 IRIS2.0 在所有 RUP 的项目中度量一致性，或者微小的分歧。

IRIS 2.0 是业界第一个基于标准的 RUP 执行系统。

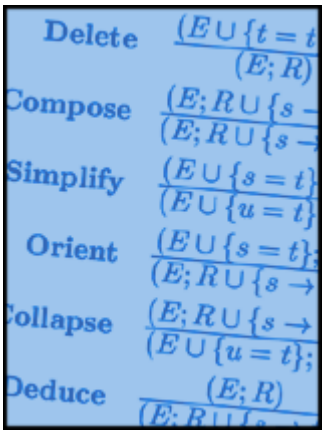
（自 PRNewswire，UMLChina 袁峰 摘译，不得转载用于商业用途）

软件工厂+DSL：微软对 MDA 的回答

[2004/12/1]

两年前，OMG 推出了 MDA，用 UML 来自动化来自应用、中间件或者定制组件的软件集成。顾问公司 META 集团的副总裁 Thomas Murphy 认为，从某种程度上说，软件工厂和与之相关的领域特定语言 DSL（Domain Specific Languages）的概念，是微软对 MDA 的回应。

“MDA 是 OMG 的软件工厂。微软的目标是 DSL 领域，因此很多事情都是建立在观察建模以及事情运转的不同视角（View）的价值之上”



那么，微软认为一个 DSL 是什么呢？微软企业框架和工具组的架构师 Keith Short 指出，许多开发人员已经在使用 DSL 工作了－虽然他们自己还不知道。

“SQL 就是我们用得很好的一种 DSL”，Keith 解释，开发人员使用 SQL 的过程中并不需要了解关系数据库是如何工作的。

好，那么这和软件工厂的概念有什么关系吗？对初学者而言，微软把软件工厂作为自动化一些手工任务和封装领域知识的一种方式。举例来说，领域知识可以被封装为代码或者组件，并可以容易地被复用。Short 认为，这也就是 DSL 的设计目的：从开发人员的角度来看，DSL 可以帮助抽象某个特定领域的复杂度（例如架构、过程、技术标准等）。如 Web services 就很复杂，要求开发人员要了解哪些 service 是可用的，如何连接的细节等。Short 指出，“设想你可以把这些抽象到一个 DSL，这个 DSL 就是关注特定开发环境的工具”。

和 Web services 一样，SQL 是多年来标准化努力的成果。微软和其它开发商都有义务来开发好的工具帮助开发人员创建自己的 DSL。Short 说，“要实现理想的场景，我们要做的事情之一就是要帮助设计人员简单地构建 DSL 并实现它们……”

这个软件巨人在九月的 OOPSL 会议上发出了 DSL 的呼吁，并捧出了一个新的工具集，帮助开发人员在图形化环境下定义、编辑 DSL。Short 指出，藏在该工具集后面的想法就是使得“软件工厂的创建者可以更容易地描述一些事物之间的交互，例如源代码和 schema 之间。”

微软的动作已经引起了开发人员的关注，即使是像 Michael Hudson 这样过去对微软的技术通常不屑一顾的程序员。Hudson 是 Praxis 公司的程序员，J2EE 专家。他认为 DSL 是微软的软件工厂中最有震撼力的思想。但他认为很多思想都是从其它的技术例如 XDE 中借用过来的。XDE 是 Rational 支持 MDA 方法的 RAD 工具。

Hudson 说，“我觉得他们找对了关键的地方，那就是：高层建模必须用领域特定的语言来描述。你可以更快地得到你所面对领域的业务逻辑，开发过程更快更有效。我觉得，软件工厂这个概念如果能够获得成功的话，应该是因为在 DSL 上的创新。”

开发人员是可以将他们的经验用 DSL 的方式进行封装的，这是完全有可能的。这实际上也是他们对自己工作或多或少的抽象。

但 Thomas Murphy 认为这种事情将不会发生，“我们已经推动这个概念很久了：通过抽象来简化软件开发，使得谁都可以进行软件开发。但未来还是会需要大量有创造力的开发人员，在软件开发的最前线去开发组件、进行设计”。

（自 adtmag，UMLChina 袁峰 摘译，不得转载用于商业用途）

征稿

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

Sybase 宣布 PowerDesigner 11

[2004/11/17]

11 月 17 日，PRNewswire：Sybase 公司（NYSE: SY）是企业底层结构和无线软件领域的领袖级开发商。今天，Sybase 宣布了其即将发售的 Sybase PowerDesigner 11，这是更好地协同 IT 和业务目标，并可以根据业务条件灵活变化的企业建模工具。遍及全球的 PowerDesigner 的忠诚用户将会得到新的建模能力和扩展的数据库支持。Sybase PowerDesigner 11 将在 2004 年底正式发布。

Sybase 的全球产品线总裁 Dimitri Volkmann 指出，“新版本进一步巩固了我们对用户的承诺：提供最具有革新性以及完整的数据建模工具。PowerDesigner 11 采取了独特的方法来进行诸如 UML、业务过程分析以及数据建模等不同元数据的整合，从而架起业务终端用户和 IT 部分之间的桥梁，并可以根据业务条件的变化轻松地进行改动”。

Sybase PowerDesigner 11.0 的新功能和优点包括：

-- 业务价值:

* 冲突分析：简单有效地评价业务或者技术变化带来的冲击，从而得到更好的敏捷性和可预测性。

* 需求管理：缩短项目完成时间并通过更有效的需求获取和规约保证更准确的项目结果。并通过建立设计和需求的关联保证更好的可追踪性。

* 业务过程分析：通过理解业务和底层应用、数据的关系来设计更好的业务过程。

-- 技术价值:

* 增强的数据建模：支持最新的 RDBMS 引擎以及数据库中的 Web services 和 XML 等最新功能。

* 增强的 UML：加入了对 J2EE1.4 和 Web service 模式和模板的支持。

* 增强的业务过程模型：为使用该工具设计的过程提供层次视图，并提供更清晰的沟通能力和更好的灵活性。

* 新的信息流动模型：支持复制、数据移动和元数据管理，提供模型驱动的方法设计移动数据库（mobile database）以及和中央数据库（central database）之间的同步定义。

Mount Vernon 数据系统公司的总裁 Michelle Poolet 说，“我们的项目和 Sybase PowerDesiner 休戚相关，我们热切盼望 11 版本的到来。我的团队和我自己都在设计高复杂的数据库。11 版本的新功能将会给已经非常智能的数据建模技术带来更多，包括新的冲击分析、需求、以及其它方面的改进，这些都使得 PowerDesiner 在业务和应用开发领域拓展得更远。”

价格和获得

Sybase PowerDesigner 11.0 将于 2004 年底发售，价格为$995 每开发人员席位。关于功能和价格的更多信息，或索取 demo、订单采购，请垂询 1-800-8-SYBASE 或访问 http://www.sybase.com。

Sybase 开发者网络（Sybase Developer Network，SDN）提供深入支持，帮助开发人员从 PowerDesigner 和其它 Sybase 工具中得到最大收获，提供免费试用软件、技术信息以及和其它 Sybase 用户的合作。SDN 也将继续提供包括 CodeXchange 在内的面向开发人员的服务，开发人员可以参与开源代码的开发、和同事进行项目协作。关于 SDN 的更多信息，请访问 http://www.sybase.com/developer。

（自 PRNewswire，UMLChina 袁峰 摘译，不得转载用于商业用途）

设计模式精解

Design Patterns Explained

看不懂《设计模式》？
不用惭愧，别人也一样苦恼。
先把它供起来吧，看这本！

UMLChina 训练
辅助教材

Alan Shalloway & James R. Trott 著
熊节 译

清华大学出版社

从业务建模到需求说明的转换

Svatopluk Štolfa 著，李玉蓉 译

讨论

摘要

业务建模是软件开发的第一步，也是容易被忽略的第一步。在本文中我们将描述如何利用由 UML 活动图捕捉到的业务流程来创建用例说明。

1 引言

业务建模是大多数软件开发过程中的初始阶段，接下来就是用例建模、分析、设计等。用例建模和软件开发过程中的其它阶段目前已经被定义清楚了，它们之间的转换也已被规范和清晰地定义了。然而，从业务建模到用例建模的过渡却是模糊的，这就是我们常常跳过了这一步的原因。我们常常不清楚业务流程对于软件开发有什么用，因此也不想浪费时间去制造一些对于我们来说效率不高或无用的东西，通常我们通过直接创建用例模型来收集需求。本文的目的是为由业务模型到软件开发过程中的其它阶段的过渡建立规范的方法和有用的技术，这意味着这种新技术可以以下述方式用于软件开发，即通过运用业务建模来将软件开发的下一阶段明确化。

软件开发过程的初始阶段及其作用为：

业务流程是一个或多个相互关联的过程或活动，它们共同实现某种商业目的或达成某种政策目标。业务流程有两方面的用途，即可以把它作为目标或作为工具。在软件开发的过程中，这两方面我们都可以善加利用。通过组织，模拟和实现精确策划的方案，业务流程能给我们有效管理项目的机会。另一方面，我们还可以从运用业务流程来建立用例规划中得益。从这个角度来看，业务建模的目的是为软件工程师和业务工程师的交流提供共同的语言。遗憾的是，目前的技术并没有充分涉及业务流程的这一十分重要的作用。

需求工作流程的目标是通过确定系统的功能来描述系统应该做什么。需求建模让系统开发人员和客户在这方面达成一致。用例模型从系统的参与者和用例的角度来检查系统的功能。参与者指必须与正开发的系统产生互动的人（即用户）或物（即其它系统），用例是指系统展示的一种行为模式，每个用例是由参与者和系统通过对话演示的一系列相互关联的业务。用例模型可用 UML 用例图来描述，我们可以用“包含”、“扩展”和“泛化”关系

来为用例建立结构和用“泛化”关系为参与者建立结构。

下面将介绍我们的研究结果，它主要集中在业务建模方面。接下来的合理步骤是运用业务建模的方法来确定系统需求。这项研究致力于建立从由 UML 活动图捕捉到的业务流程到由 UML 用例图捕捉到的用例模型的过渡方法。

从业务建模到需求说明的转换至今尚无规范的定义，但用我们的方法可以得到一种直观感受。我们的方法包含了以下几步： 首先建立业务流程模型，再由业务流程导出用例模型，然后通过模式匹配来组织用例。

2 由业务活动到用例的映射

业务流程可通过 UML 活动图来描述，我们可以通过业务活动来捕捉业务流程的状态并描述这些状态之间的转变。UML 活动图的目的是描述由执行的内部机理所确定的控制流。

我们的方法可用下述示例来说明，这个例子是关于视频图书馆的一个信息系统。我们应当首先确定这个系统所要提供的活动，剩下的就是由人参与的活动。在图中我们没有将人的活动包括进去，但为了更好的理解仍将这些活动的符号留在了图中。

业务活动包含用例及其组件，我们必须确定这些活动中哪些构成用例的一部份，哪些本身就是一个用例。从活动到用例的映射主要有两种办法，它们分别是“从多个活动到一个用例的映射”和“从一个原子活动到一个用例的映射”。

2.1 从多个活动到一个用例的映射

这种方法是前面提到的映射的第一步（参考图 1），我们确定三个活动（即系统所提供的活动）组成左边一个用例（即借阅）。“借阅”提供了在借阅过程中，系统参与者与系统间的所有相关的互动。在右边有两个活动，它们构成了“归还”用例。“归还”用例提供了与归还过程相关的系统与参与者的互动。

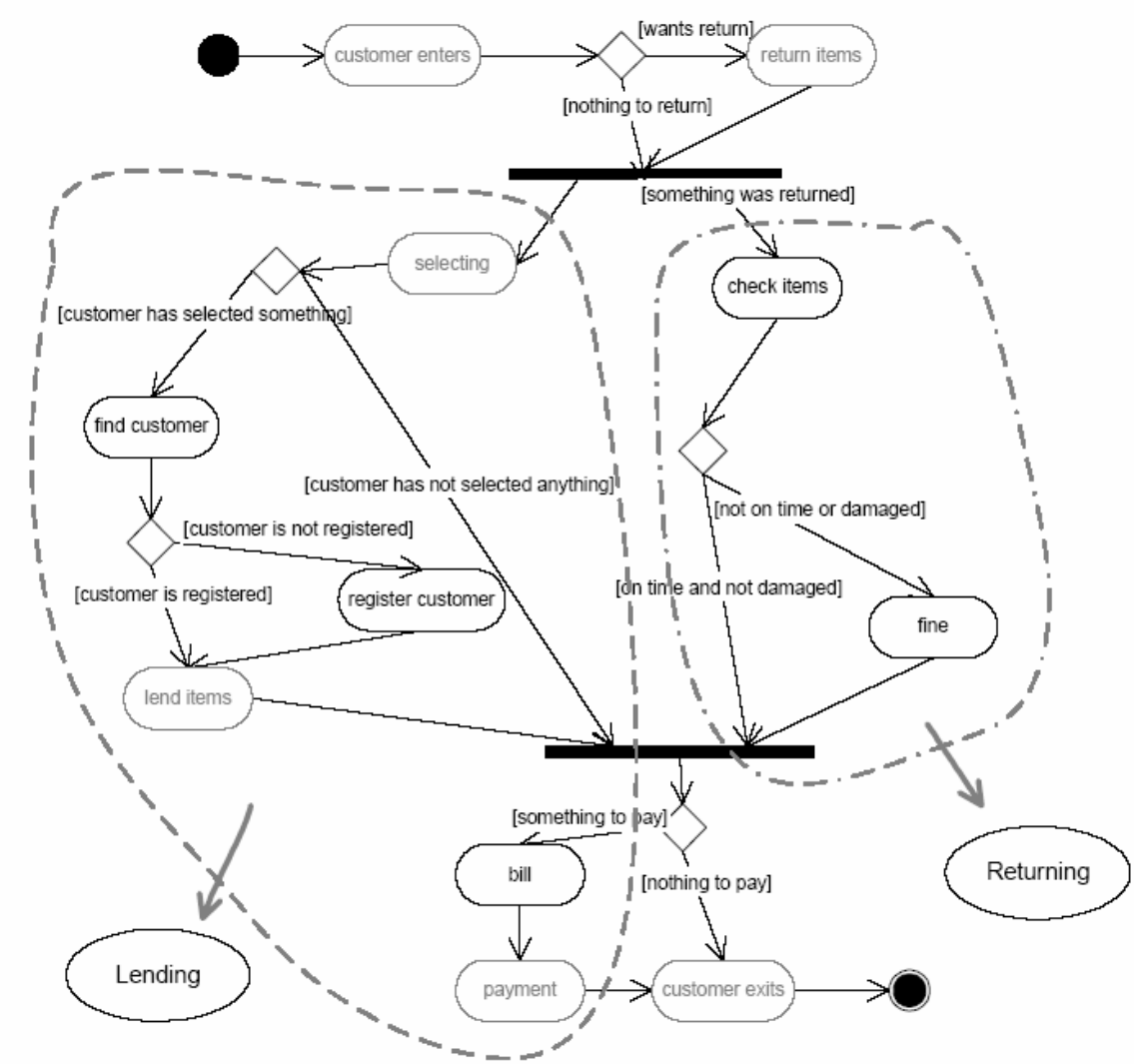


图 1 从多个活动到一个用例的映射

2.2 从一个原子活动到一个用例的映射

我们可以通过这一方法把原子活动（即简单的活动）映射到简单的用例，这种方法通常被称为一一映射（参考图 2）。在我们的例子中共包含 5 个简单活动，即：“查找客户”，“客户登记”，“账单”，“查验项目”和“罚款”，由这种方法构建的用例与活动有相同或相近的名字，使之更清楚。

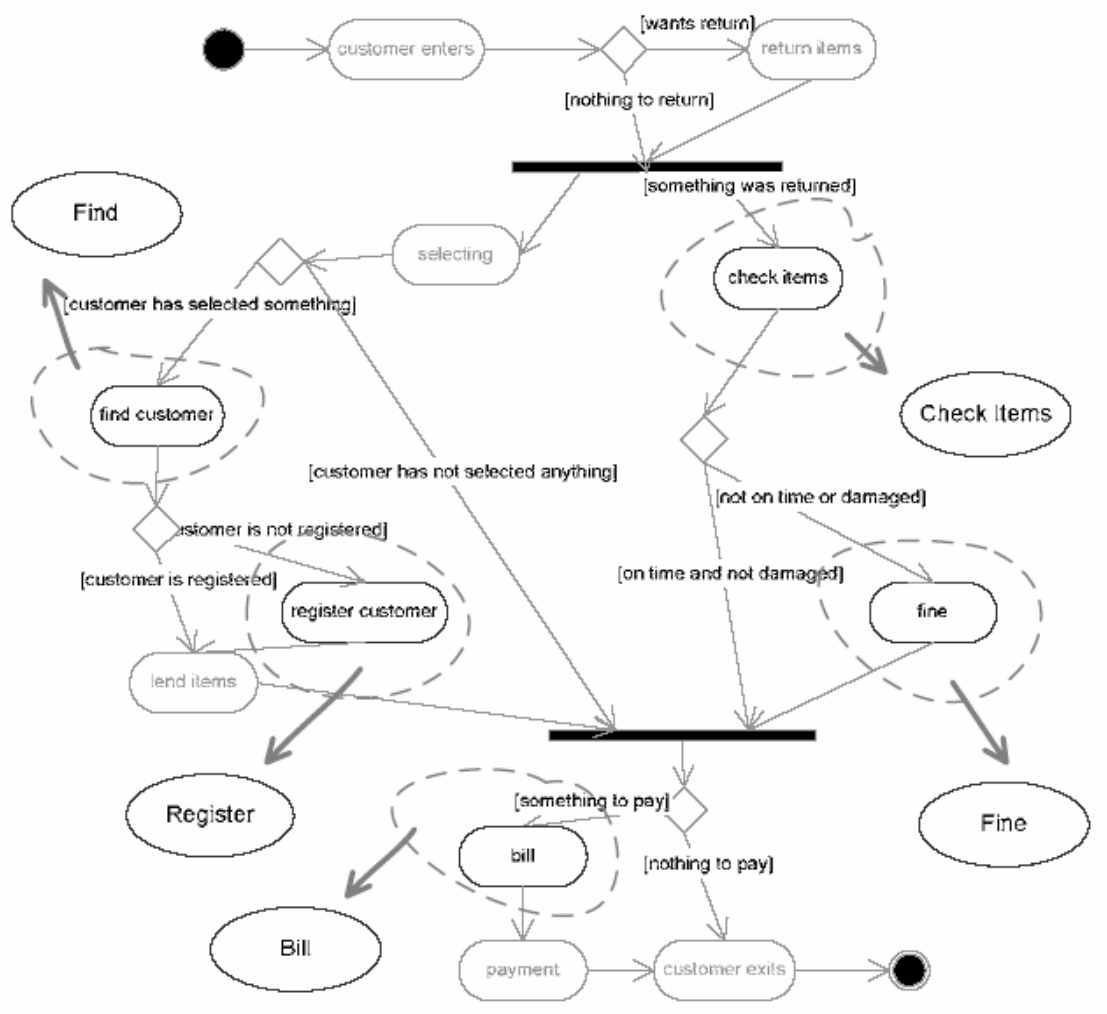


图 2 从一个原子活动到一个用例的映射

3 通过模式组织用例

在上例中我们已有了几个用例，但还没有建立它们相互之间的联系。UML 用例图允许通过“扩展”,“包含”或“泛化”关系来组织用例，通过原始活动的前后关系，我们发展了几种基本映射模式来组织用例。下面介绍其中的两种映射模式，我们称之为“顺序模式”和“可选模式”。

3.1 顺序模式

目的：用“包含”关系组织从顺序活动中导出的用例。

这种模式因系统执行活动的顺序而得名。用“从多个活动到一个用例的映射”的方法我们可以把多个活动映射到一个用例，而顺序活动可通过一一映射的方法映射到多个用例（参见图 3），本模式通过“包含”关系把两个由顺序活动导出的用例关联到主用例，在图 Fig3 中的“包含”关系表明了“UseCasel”使用了“UseCase Activity2”和“UseCase Activity3”。

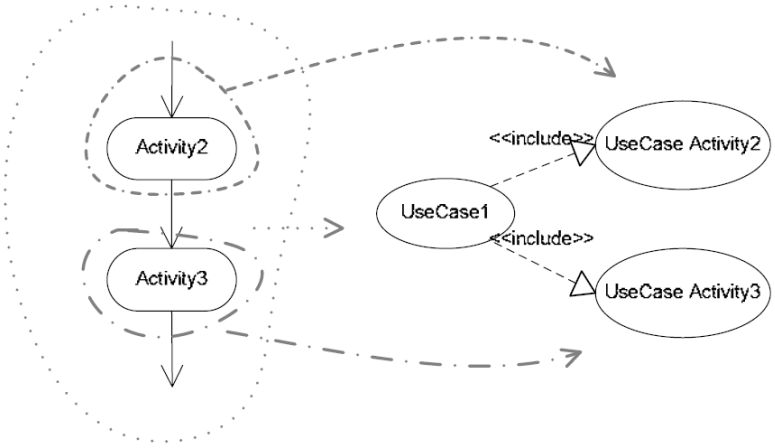


图 3 顺序模式

3.2 可选模式

目的：通过“扩展”关系来组织由可选活动导出的用例。

这种模式表明了如何把可选活动映射和组织到用例图。只有条件为真时，系统才会执行可选活动，否则系统就跳过该活动（参看图 4）。在图 Fig4 中假如某个条件块和附加活动是某个用例“UseCasel”的一部份，则“Activity2”可以映射到“UseCase Activity2”，它扩展了“UseCasel”。“扩展”关系表明“UseCase Activity2”可扩展“UseCasel”。

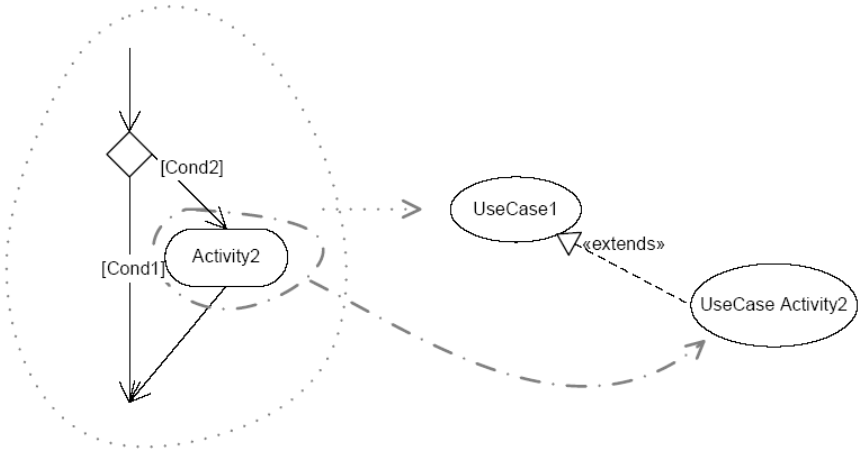


图 4 可选模式

当把可选活动映射到用例，且与它有关的全是这个用例的部分时可使用本模式。

3.3 运用模式到实例中

现在我们来运用上面定义的两种基本模式来组织本文例子中的用例。利用这些映射模式这是非常容易办到的（参见图 5）。我们运用顺序模式组织用例 “查找客户”，“账单”和“查验项目”，用可选模式来组织用例“客户登记”和“罚款”。为更清楚起见，我们排除了“客户未作任何选择”这种情况，因为它可通过扩展“借阅”的另一种用例来实现，“查找客户”和“客户登记”应通过这一被排除的用例来进一步组织。

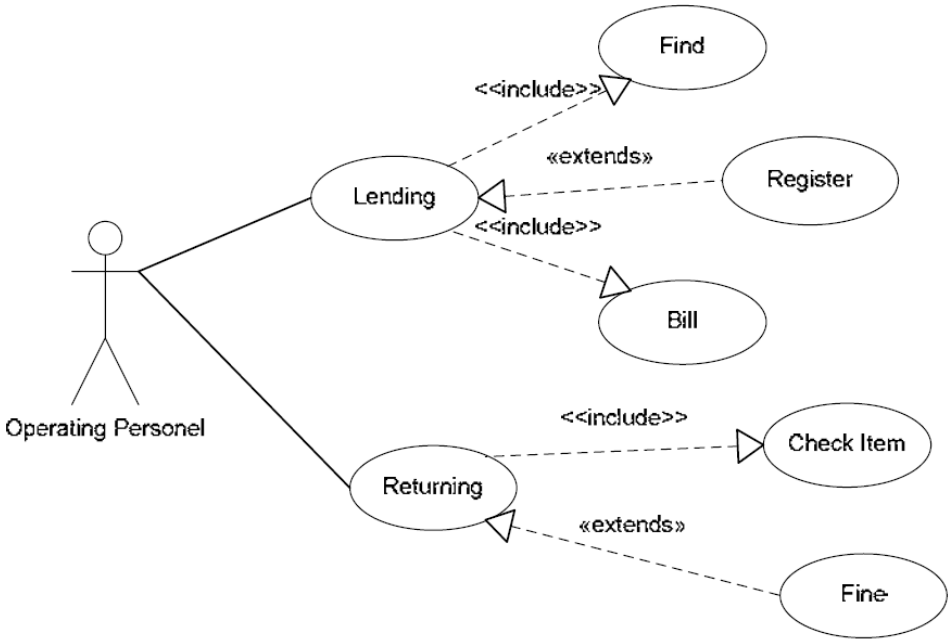


图 5 例子---结果

4 结论

本文介绍了如何运用业务流程来确定用例模型需求的方法，我们引入了两种简单模式并介绍其使用方法。但用例模型开发更为复杂，本文介绍的方法尚不能解决其中遇到的所有问题，因此在将来的研究中我们将着重于该方法的改进和推广。

参考文献

1. Stolfa S., Vondrak I.: ***Using the Business Process for Use Case Specification***, ISIM 2003, Brno, Czech Republic.

2. Schneider G., Winters J. P.: ***Applying Use Cases Second Edition -- A Practical Guide***, Addison--Wesley 2001.

3. Booch, G., Rambough, J., Jacobson, I.: ***The Unified Modeling Language User Guide***. Addison--Wesley 1998.

4. Vondrak I., *Byznys Modelovani*, Objekty2002, Prague, Czech Republic.

5. DeMarco, T.: ***Structured Analysis and System Specification***. Englewood Cliffs, New Jersey, Prentice--Hall 1979.

6. Fowler, M.: Analysis Patterns: ***Reusable Object Models***, Reading, MA: Addison--Wesley 1995.

7. Fernandez, E.B.: ***Building systems using analysis patterns***, Procs. 3rd Int. Soft. Architecture Workshop (ISAW3), Orlando, FL , November 1998 , 37—40.

征稿

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

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

众多问题根源在于
领域模型没有捕获到
业务的深层内涵

Eric Evans

Foreword by Martin Fowler

《领域驱动设计》中译本

即将由清华大学出版社出版，UMLChina 审稿

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

机械工业出版社即将出版

Scott Ambler

UP 实作的一些常见问题（上）

think 文

讨论

说起 UP 或 RUP，一般都会津津乐道其中的用例驱动、架构为中心、迭代和增量的原则。但实践中往往难度不在一个虚的“过程”，而是其中的技能。

用"治病"举例，对于复杂的病情，必须承认这个事实：一次性地检查症状、拟定治疗方案、实施是不可能的。应该分成若干疗程，根据病人的情况随时调整。但不管如何“迭代”地治疗，检查 (拍片、望闻问切) 的技能、拟定治疗方案的技能，实施治疗方案(打针、理疗、手术)的技能必不可少。如果医院团队不具备这些技能，把一个虚的“过程”倒背如流又有什么用？

下面，笔者根据自己的所见所闻，按照科目列出 UP 在实施中的一些典型技能问题。

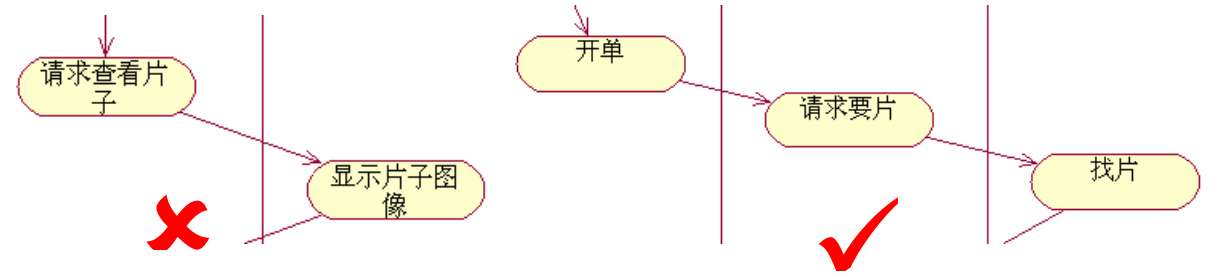
图 1 UP 中的科目

业务建模

业务建模要回答“当前的现实是什么”，通过了解当前现实，帮助理清软件需求。

1. 业务模型不描述当前现实而是掺杂了对系统的想象

用活动图来描述业务用例的实现细节时，本来应该描述当前的业务流程，但开发人员经常会不自觉地把它变成系统实施以后可能的工作流程。



错误的，掺杂了系统实施以后的想像　　正确的现实流程是这样

图 2 业务流程要描述当前现实的流程

需求通过研究业务现实，再结合上愿景而得到。如果描述的当前现实是一个虚假的现实，在上面得到需求就如同在流沙之上构建大厦，让人担心了。

同样，另一个工件业务对象模型里的类，指的也是现实中的对象（业务执行者、业务工人、业务实体），而不是可能要构造的系统里的分析类、设计类。

2. 过多的业务用例

开发人员在建模时，有时会觉得业务用例太少，心虚，不自觉地把业务组织为了提供这种业务用例的价值而产生的内部流程的活动（即业务用例的步骤）当作了业务用例。业务用例代表了一个业务组织对外提供的价值，通过这个价值，来推动业务流程的改进，包括通过引进一个软件系统来改进。所以，业务用例应该是很大的概念，涵盖了许多业务组织中的活动。

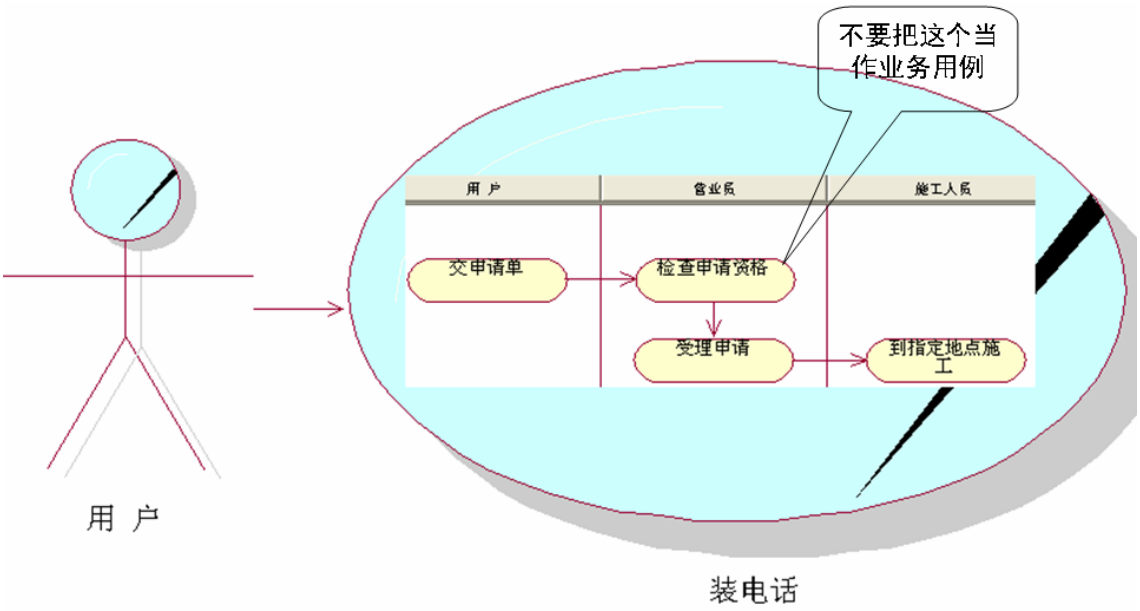


图 3 不要把内部活动当作业务用例

需求

需求回答了“软件开发出来什么样”。使用用例技术，我们把需求组织成：谁用软件？用来做什么？做的细节？

1. 没有愿景

开发人员介绍项目时，洋洋洒洒说一大堆系统有哪些功能，准备采用的技术手段、架构等等，但被问到一个问题时，往往会瞠目结舌，这个问题就是“为什么开发这个系统”。

为什么要开发这个系统？谁出钱“买”这个系统？怎样才会觉得“值”？

愿景应该清晰地写出来，并被参与项目的每一个人共享。缺乏清晰、共享的愿景往往是项目失败的主要原因。缺乏愿景，使得需求的范围无法得到控制，好像系统这个也可以做，那个也可以做。画出了一张漂漂亮亮的用例图，只是达到了形式上的正确。图上的用例都是合理的吗。是不是所有用例都能追溯到愿景目标？

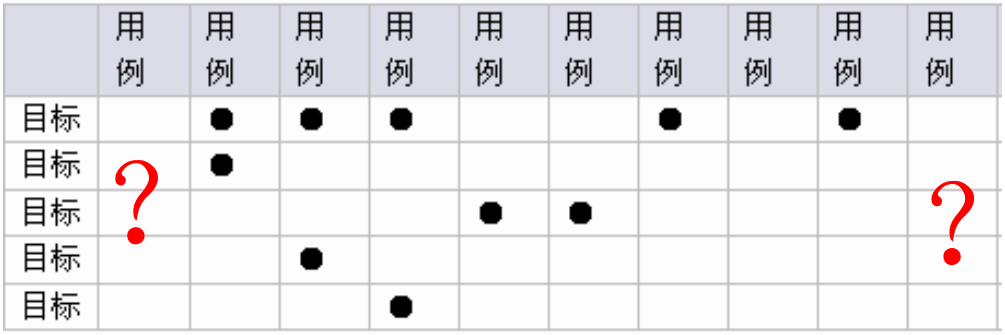


图 4 用例要能追溯到愿景目标

2. 虚假的愿景

即使开发人员回答了愿景，有时候回答的也是一个虚假的愿景。例如，

问：为什么要开发这个系统？

答：要实现无纸化办公。

没有人为了省几张纸就开发一个系统，其实客户所要的目标是“加快公文流转效率”。类似的还有“能够让人们上网买书”，除了在前几年的泡沫时期，难以想象为了让人能上网办一件事就开发一个系统。

愿景目标必须有办法度量。“开发这个系统的目的是为了改善我们的工作”、“开发这个系统的目的是为了帮助我们进行招标工作”这样的目标是没有多大意义的，因为没有明确的度量标准。

3. 不写用例文档

许许多多用例方面的问题，根源在于开发团队不写用例文档。用例的小人圈圈只是概括了目标层级的需求，书写用例文档的过程是使用用例来探索各种需求的过程。“这个算不算一个用例？”这样的问题，很多时候一写文档就知道了。如果你能按照用例该有的内容写出文档来，而且感觉上还比较顺眼，那就 OK；如果不顺眼，就有必要重新审视了。如下图：

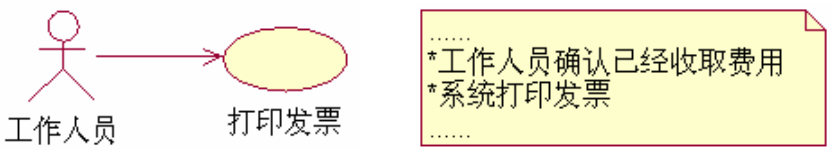


图 5 从步骤看起来，“打印发票”是否工作人员的有价值目标令人怀疑

4. 滥用用例关系

实际上这也是不写文档导致的恶果。用例的关系是整理用例文档使之结构更合理的手段。没有文档，谈论关系是没有多大意义的。

扩展：分离复杂的扩展路径或者未定的、易变的路径到扩展用例。常见的误用是把“做完这个以后可以这样，也可以那样”当作扩展。例如：

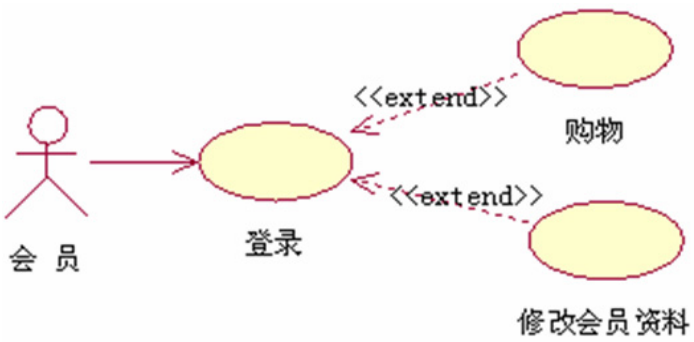


图 6 扩展的误用

包含：提取几个用例中都有的，单独形成价值的公共步骤到一个单独的用例。常见的误用是“用例包含步骤”，例如：

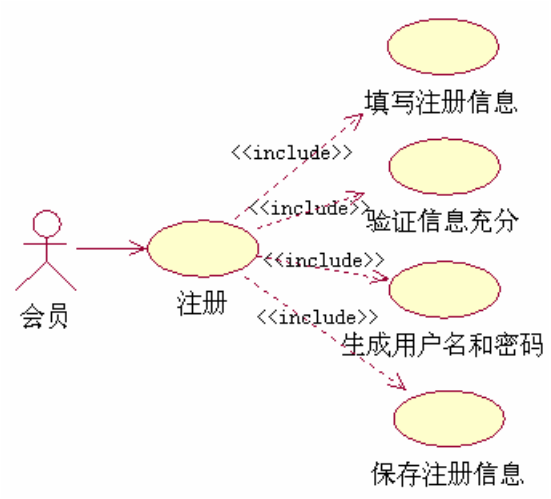


图 6 包含的误用

看到这种“一大带几小”的图形，可以有充分的理由怀疑建模者把步骤当作了用例。带有 Include 关系的用例图往往是“几大带一小”。

5. 只有功能需求，忽略其他需求

用例不止组织了路径、步骤等功能需求，还组织了业务规则、数据需求、性能需求、设计约束等需求。有的时候，开发团队只是简单罗列几个步骤，就认为完成了用例。其实很多产品竞争中，非功能需求（速度、容量）往往是重要的决胜特性，以用例为核心组织需求并不意味着用例目标本身就是这个软件的核心卖点。

分析&设计

分析勾画出软件内部的核心机制，设计则是把这种机制实现在某种架构和平台上。

1. 没有“大图”

打开一些团队的模型，经常会发现 Logicview 下面分了很多个包，每个包里一张类图，里面零零散散的放了几个类。包的名字还经常起名为“××子系统”之类。找不到一张勾勒出整个系统核心概念的“大图”。应该有这样一张图，把所有的核心业务类放在一张图上，使团队对系统的核心概念有整体的把握。

图 7 看不到全局的“大图”

2. 从用例图直接得到类

开发团队有的时候画了用例图，不写用例文档，就直接从用例图去寻找类或者画顺序图。其实只有用例图，并不足以让我们了解系统里面应该有哪些类来实现这个用例。这个时候，开发人员还需要去想用例里面的细节，而很多细节是属于需求的细节，这样就容易把需求（找到问题）和分析（解决问题）混在一起。

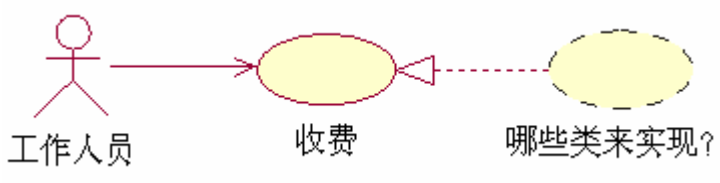


图 8 单从用例名称难以找出类

3. 从用例得到类时演变成功能分解

即使有了用例文档，开发人员有的时候也会因为对象分析技能的缺乏，把寻找类变成一种功能分解——把一个步骤变成类。如下：

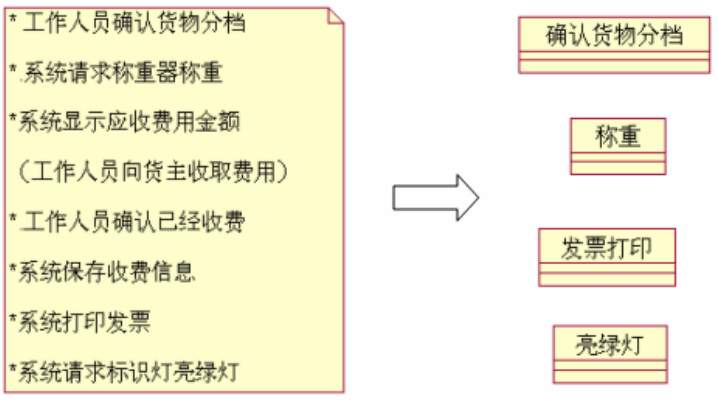


图 9 寻找类变成功能分解

比较正确的类应该是“货物”、“收费”、“称重器”（封装了访问称重器的秘密）、“标识灯”（封装了访问标识灯的秘密）等。

4. 新手直接从交互图着手

从用例文档到类有两条思考路线，如下：

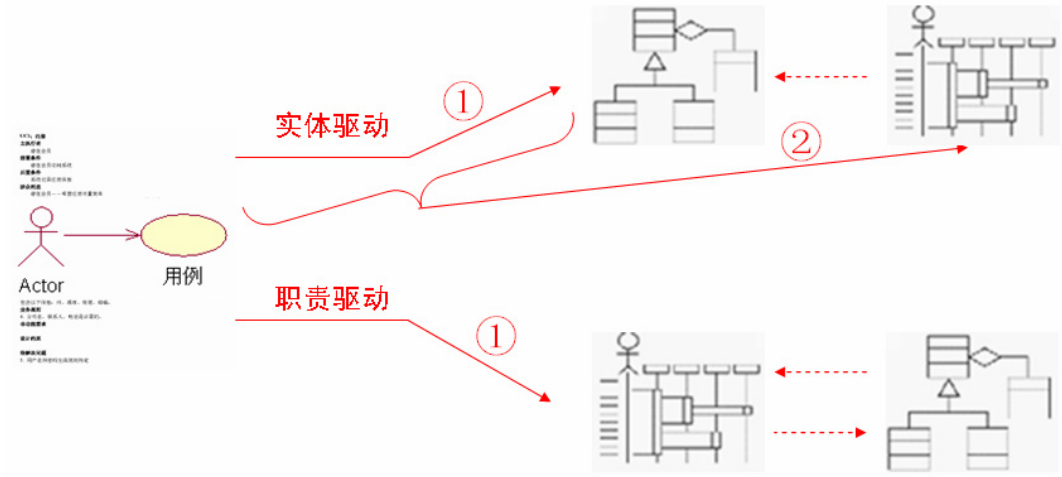


图 9 从用例到类的两条路线

一条路线是先从用例文档抽取出实体类图，然后再把用例文档和实体类图结合起来，通过交互图（顺序图或协作图）把用例文档中的各种责任分配到类上。第二种方式是先从用例文档开始画交互图，在画交互图的过程中，根据责任来发现类、属性、关联。

第一条路线（实体驱动）常被人诟病，认为很容易导致过程＋数据的结果。第二条路线（职责驱动）往往被认为所分配的职责要更合。但对刚刚开始起步的开发人员来说，第二条路线更危险。开发人员容易陷入把步骤当

作类的局面，而且容易产生这样的场面，“我发现好多类啊”，仔细一看，发现了许多边界类、控制类（其实这些类在分析时用不着去发现，有执行者，自然应该有相应的边界类，有用例，我们也可以认为它有一个控制类，基本上是一对一的），而对承担主要业务责任的实体类却是了了带过。其实真正的难度就在于：一个用例需要有那些实体类来协作实现，一个类会参与哪些用例，这是一个多对多的过程，由人脑来决定哪个多对哪个多。所以，初级选手宜采用第一条路线起步，专家级选手可以选择第二条路线。

4. 花哨的分析类图

打开一个分析类图，接口依赖满天飞，还有各种设计模式，看起来恨不得把《UML 参考手册》和《设计模式》里提到的各种知识都用上去。领域类旁边布满了“***Observer”、“***Strategy”…为名的类，看起来十分扎眼。

面向对象的基本复用机制就是下面两种关系：

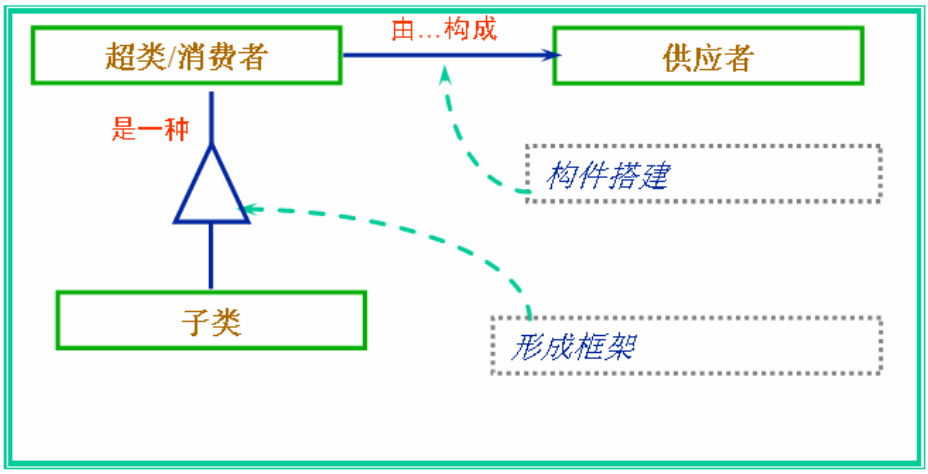


图 10 面向对象的两种基本复用机制

泛化：子类通过继承拥有超类的特征。它表示一种集合关系。“人有男有女”不意味着一个人身上组装了一个男人和女人，而是说人的集合涵盖了男人的集合和女人的集合。

关联：对象通过组装拥有其他对象的特征。它表示一种个体关系。“人有手有脚”不意味着人的集合和手、脚的集合有什么包含关系，而是说一个人的个体组装了一些手和脚的个体。（这是一种组合形式的关联，而“人有车有房”则是相对较弱的连接形式的关联。）

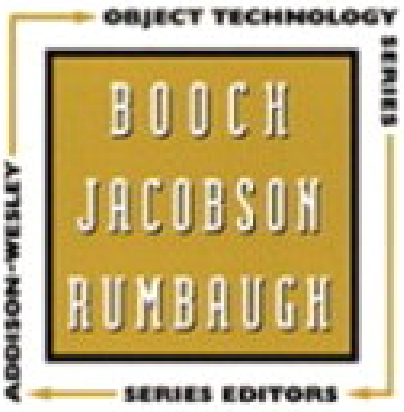
我们应该使用这两种基本的关系来描述我们的领域模型，就像 Coad/Yourdon 那个朴素的时代。某些 GOF 模式如 COMPOSITE（组合）、STRATEGY（策略）中蕴含的经验和思路对领域建模很有益处，但不是所有的 GOF 模式都适合用于领域建模。（待续…）（本文首发于《程序员》2004 年 12 期）

THE UNIFIED MODELING LANGUAGE REFERENCE MANUAL, Second Edition

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

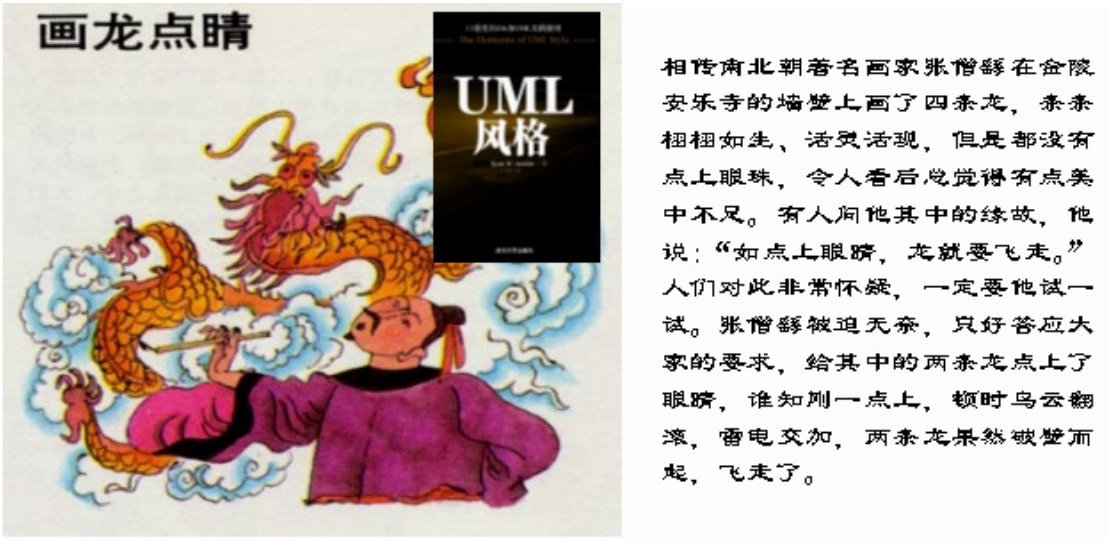
Covers UML 2.0

UMLChina 译
（王海鹏、李嘉兴）



CD-ROM
Included

《UML 参考手册》2.0 版中译本
即将由机械工业出版社出版



它不讲概念，它假设读者已经懂了概念。

它不讲工具，它假设读者已经了解某种工具。

它不讲过程，它假设读者已经了解某种开发过程。

它只是在读者已经了解方法、过程和工具的基础上，提醒读者在绘制 UML 图时需要注意的一些细节。

在这本类似掌上宝小册子中，Ambler 提出了 200 多条准则，帮助读者在画龙的同时，点上龙的眼睛。

掌握需求管理中的需求

Thomas Murphy 著，liujie 译

焦点问题

软件开发过程的平稳运行离不开有效的需求管理。然而，许多组织在不同的团队、项目、业务单元中实施不同的过程，这不但不利于组织从总体上进行项目进度管理，也限制了它们有效把握开发成本和收益的能力。如果开发组织想提高效率、控制预算，就必须对项目文档和初始过程实施标准化。那些没有达到这一层次的组织必须继续改进，以实现复用、敏捷和高效管理。

背景

业界当前的主要问题仍然与资金限制和持续增长的技术复杂度有关。这就要求组织更加有效地实施 IT 项目控制并对其投资回报率做出评估。另外，对于复用的要求也促使组织和过程必须做出改变，以达到预期效益。（复用是由基础设施及通讯协议的标准化过程推动产生的）随着人们对有效控制和期望效益的要求提高，IT 组织必须首先保证实现需求和文档管理的一致性。这种一致性工作大多可以借助于不断改进的工具及集成（2003-05）来实现，但是任何工作都要建立在一致性管理实践之上。

定义需求管理

需求管理过程要实现两个目标，第一，要确保项目所涉及的人员都能够意识到自己有哪些需求，第二，要使他们确定自己的需求全部都已被表达出来。绝大多数情况下，无效的需求收集过程是导致项目失败和软件缺陷的根源。而且，修补缺陷的成本随着时间的推移而成指数增长态势，影响它的时间因素包括，在随后的开发过程中发现这一缺陷的时间，以及组织开始寻求改进开发过程的时间，但是，需求因素却常常被人们忽略。扎实可靠的需求过程是整体过程有效实施的基础，要实现质量和生产力的不断增长，公司就必须从需求过程做起。

SEI 为软件过程定义了以下五个成熟度级别：

- 第一级：不存在任何度量指标来度量当前过程是否有效。计划制定是无效的。
- 第二级：建立了严格的、有章可循的过程，包括软件配置管理、质量保证、项目规划和管理，合同管理 contract management，以及需求管理。Project commitment 以之前的项目为基础，而且这一级别的重点在于通过定义度量指标来建立项目管理规则。在这一过程中，还建立了项目基线 project baselines，作为过程的组成部分，用于 determine project drift。
- 第三级：在这一级别，组织的开发过程都记录在案，包括应用程序的开发及维护。这一级别除了包含了第二级别的控制工作外，还增加了同行评审、组间协调、产品工程、培训、软件管理，以及过程定义与焦点。A specific process（如，统一过程，结构化工程，极限编程）is not specified ，关键在于组织要将它的过程记录下来。目标是改进项目涉及的所有资源的管理。
- 第四级：一旦组织建立了可重复的并且文档化了的过程，还定义了度量指标，之后就可以将重点转移到软件质量和度量指标上，控制过程的有效性。目标在于质量和生产力，度量指标用于 isolate areas for improvement。另外，达到这一级别要求的组织可以使用度量指标和修改进行预测。
- 第五级：在建立了稳定可靠的过程并定义了度量指标之后，组织就可以进行持续的过程改进。第五级组织类似于重视实施六西格玛的公司，它们通常寻求不断优化并 减少变化 seeking areas to optimize and reduce variations。在这一级别，组织仍然能够计划实施技术转变，它们已经将整个开发过程纳入变更管理范围之中。这包括运用根源分析 root-cause analysis 解释缺陷产生的原因而不仅仅只是修补缺陷。

除了上述五个级别，在 IT 组织中还可以发现许多别的管理层次或状态。它们包括第零级，这一级别的软件开发完全是独特的，通常取决于管理开发过程的个人的知识和能力，这种开发过程不是由个人而是由那些有着密切关联的团队协作完成的。还有一些处在第一级别的组织发现，其主要的团队成员反对建立正式的软件工程。我们不能够把这种组织与那些采用敏捷过程的组织，如极限编程组织，混为一谈。虽然敏捷方法及其支持者对于软件工程或多或少持反对态度，但他们仍然建立了可重复的过程，并重视团队协作。最后一个级别表明，重视过程的组织被描述为第二级别的组织。

图 1 SEI 的能力成熟度级别模型

许多公司都借鉴 SEI 的能力成熟度模型（CMM）来推动改进，定义度量指标（见表 1）。CMM2 级（即，可重复的）的实现，需求管理过程是必不可少的。Rational 公司也提出了需求管理成熟度五个级别的简要模型，作为其 Rational Unified Process 的一个组成部分。它给出了一些简短的标题，包括“已记录的”（written）、“有组织的”（organized），“有结构的”（structured）、“可跟踪的”（traced）和“集成的”（integrated）（见图 2）。虽然大多数组织仍然坚持项目中尽量“不立文字”，或者说是“快速出活”，但他们都至少达到了“已记录”级别。这些组织使用标准文本编辑器或是电子制表软件来管理需求，而且不同的团队和项目之间所采用的需求过程大不相同。

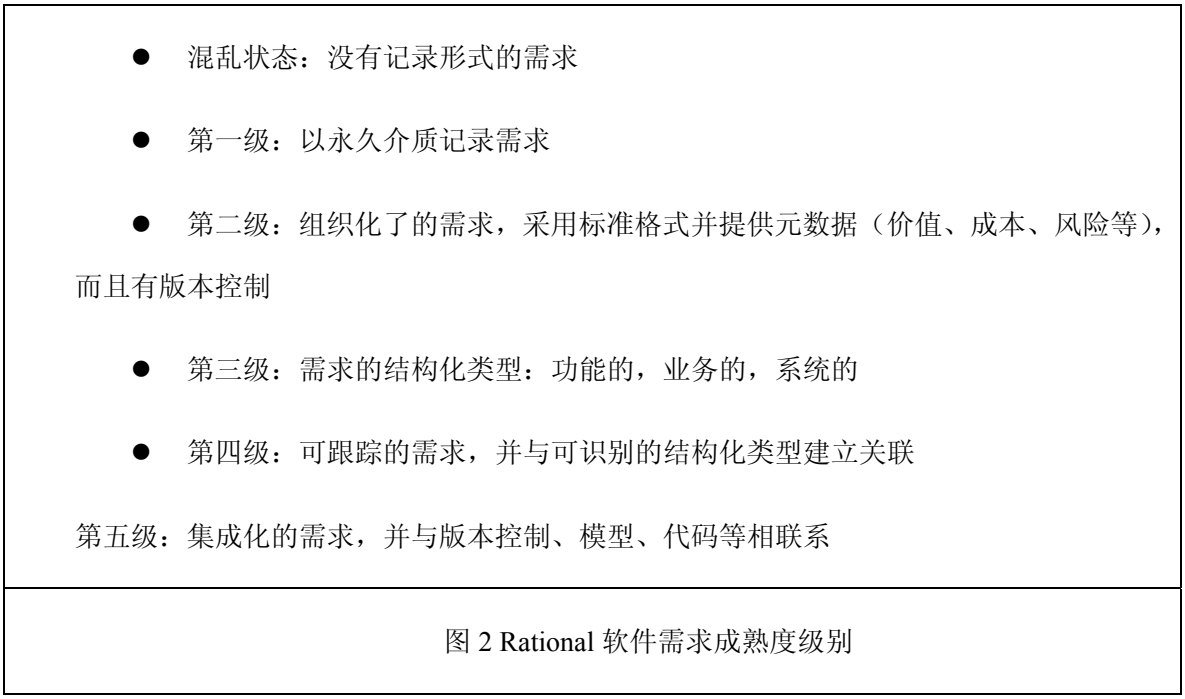


图 2 Rational 软件需求成熟度级别

如果使用了统一的模版，就上升到了第二级别（有组织的）。有组织的文档不但要求所有的文档格式化，而且要求它们要采用统一的格式和编号方案，并进行版本控制，这一点与 CMM 的第二级别是相同的。然而，组织可以不借用任何具体的需求管理工具而达到这一级别，这是因为这一级别关注的是一致性和建立过程，而非可跟踪性和度量指标。这一级别还要求组织进行培训并对原有过程做出相应改进。

上升至有结构级别（即，第三级别）要求对需求进行进一步调整。标准化的需求说明书必须包括一些属性，如优先权，风险，成本和前提条件。这有助于为特性集设定优先权，还可以使 CMM 第三层次的需求服务于团队协作。当组织上升到这一级别时，他们就会选择使用需求工具，或者至少使用数据库，将每一项需求都作为一个独立的属性实体进行管理。

向第四和第五级别迈进一般都需要使用需求管理工具或是需要对开发进行定制。可跟踪性能够帮助人们理解各项需求之间的关系，而且，随着工具的成功集成，它还能够帮助人们理解其他信息模型，如测试用例和统一建模语言（Unified Modeling Language）模型。这种集成促进了生产能力的增长，并降低了维护成本。可跟踪性还会使用户意识到改变要求和资源分配可能产生的成本。这使得 CMM 第四级别的度量指标能够综合起来，评估项目的有效性。

我们可以从文本、表格或是图型模型中获取需求。最高层次的需求管理不但要求具备全方位收集和管理需求的能力，还要求集成各种各样的生命周期管理工具。上述对集成的要求促使人们开发了多种用于构建完整生命周期的产品。主流工具通过驱动分析模型和建立测试用例来激活需求。集成和可跟踪性不仅能够促进开发过程的平稳运行，而且还能确保需求得到成功满足。需求管理是风险管理的关键部分，而且，项目风险的上升也提高了对需求过程稳定性的要求。

贯穿过程的需求

需求的成熟度并不取决于某一具体的过程。一般来说，正式的方法更易于与 CMM 对应起来，因为它们有足够程度的细节表述。然而，敏捷过程仍然离不开需求管理。虽然敏捷方法的实施者经常不使用任何工具，但是工具仍然能够对实现需求管理成熟起到很大的作用。例如，极限编程（见 ADS Delta 864）人员将收集来的需求写到 3×5 的卡片上。这样可以实现开发者和最终用户之间的良好协作，但是对于与 IT 组织其它部门进行协作，或是提供更高层次的 CMM 成熟度所要求的报告和度量指标，它却没有任何作用。使用工具可以抽取卡片上的信息，之后就将这些需求与过程的其它部分进行集成，借助出版工具，产生标准文档。使用需求管理工具还可以实现标准化编号，可跟踪性等，但是它却有可能使开发流程断裂，或是当开发者所依赖的卡片被修改而工具中没有同步反应出这一修改时，就会产生阻碍。灵活性，以及与过程其它部分实现完美集成的工具包，仍然是优秀的需求工具的标志。软件开发市场向敏捷方法方向发展，产生了两大不同类别的工具，一类是以传统的自上而下的工程方法为核心，另一类是以协作和同步开发为核心。

很重要的一点是，必须认识到需求过程应当与项目的规模相适应。大型项目会涉及更多的资源，而且这些资源会对其它软硬件资源产生更大的影响（见图 3）。小型项目则要减少文档的类型，要做到这一点，打破文档的分层是最简单的方法。

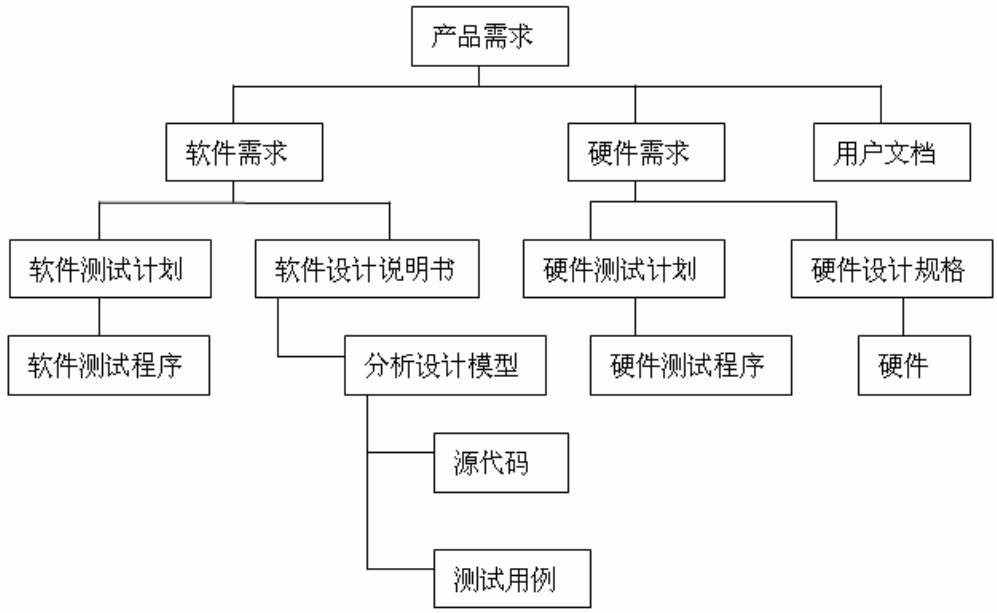


图 3 需求层次图：滴漏效应

对于不同类型的项目，需求管理也会有所不同。软件维护，程序包安装,深度开发都要以现有的状况为基础。通常来说，维护活动是不包含在传统的需求过程之中的，因为需求可以在缺陷管理或是桌面帮助系统中获得。虽然维护活动也可以得到需求数据，但是，它却不能代替使用传统管理工具进行需求改动的过程，也不能代替将需求与某一具体项目联系起来的过程。

需求工具的价值

对于管理工具的价值和为什么要使用它们，人们还有许多疑问。使用办公套件，如电子制表软和文档处理软件，IT 组织可以创建可重复的过程，并对文档和格式进行标准化。然而，这些文档对于整个开发和管理过程来说价值不大，对于实现协作也没有太大效用。在协作环境中（如，Notes/Domino，Groove），人们通过分类、工作流程和讨论/协作加强了文档管理，提高了文档的使用价值。一个更好的方法就是使用需求管理工具，收集有用数据并进行跟踪管理。我们还开始关注需求工具与开发过程其他阶段的集成，这种集成能够使组织更好的综合生命周期各个环节所作努力，并提高工具的效用。团队成员工作地点的分散造成交流不便，从而导致效率低下，而且，虽然开发过程的迭代性和协作性逐渐上升，传统工具仍然只是用于瀑布模型（或是采用功能性划分的模型）。这样，需求的低效获得和流通不畅就造成了开发过程中断层出现和用户满意度的降低。

虽然工具为所收集到的需求提供了记录格式，并帮助人们生成统一的报告，但这仍然不能说明人们有效地获得了需求。因此，组织不但需要选用需求工具或是标准化模板，还要将重点放在需求的获取上，并意识到有效的需求收集并不是一个简单的步骤就可以完成的工作。需求的格式应该多样化：文本描述、图型和原型及法定规则。需求工具为不同类型的需求和数据提供不同程度的支持，而且在设计和开发阶段，对应于加强需求的能力的不同要求，都有不同的工具。不断深入的集成可以推动过程改进，然而集成的一般方式却是移动信息或图像，并在为时已晚的时候进行检查——即在功能和集成测试的时候。

需求工具至少应该具备以下功能：

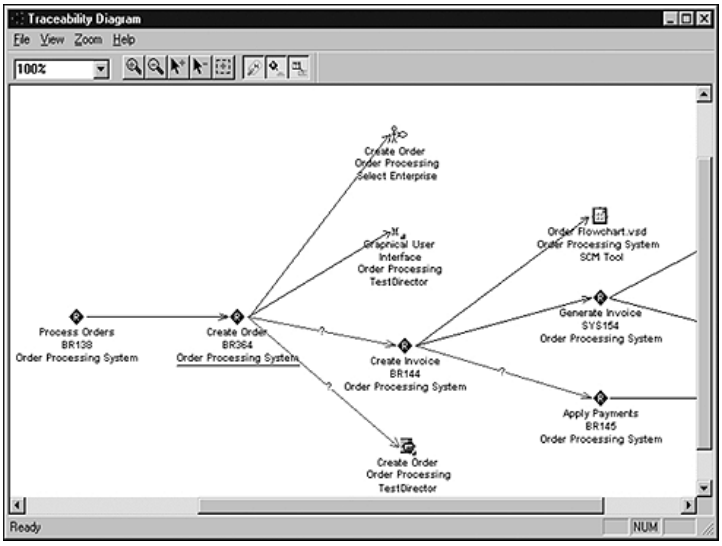
- 对每一项需求均给出标识；
- 需求的分类和委派到目标
- 需求修订标识/基线
- 基本的数据接口

更加完善的工具提供可跟踪性，协作支持，以及建模、测试和开发系统的建模。虽然工具提供商都认同可跟踪性，但是不同的工具提供的服务是不一样的。可跟踪性大多用于不同的需求之间。用户要选择那些提供对所有历史资料进行跟踪的工具，不仅包括对需求的跟踪，还包括对模型、代码和测试的跟踪。

需求工具提供商

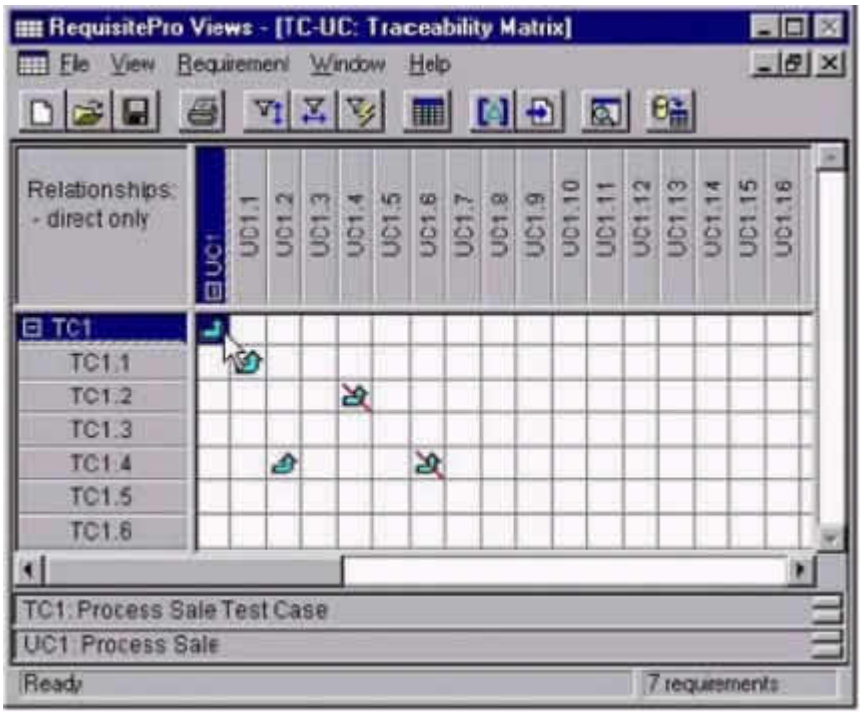
目前有许多公司都提供需求管理解决方案（见图 4）。软件市场上，Borland，Rational（最近被 IBM 收购）和 Telelogic 三家公司拥有大约 75%的安装基础，（同时，在使用独特的解决方案的组织中，也占了多数）从而占据了大部分的市场份额。这些市场领导者相对来说都是刚刚介入需求管理领域，并在过去的两年里开发出了自己的工具产品。而且，它们还都生产了集成式的生命周期管理平台（ILDE——集成生命周期开发环境）。我们期望其他主要的平台和工具供应商（如，IBM, Microsoft, Oracle, Sun, BEA, Borland）通过收购或是自主开发供应其他的生命周期组件，而且我们相信主流环境至少应该能够与其他环境实现触点集成。（如，基于应用程序接口的集成使用 XMI（XML 元数据交换））。虽然基于应用程序接口的集成支持两个应用程序之间的信息共享，但却无法提供完全集成条件下的无缝工作流程。用户需要切换窗口，而且通常情况下，所管理的项目结构都是相同的。而且，这种形式的集成不支持用户随意选择市场上最流行的工具。

Borland CaliberRM



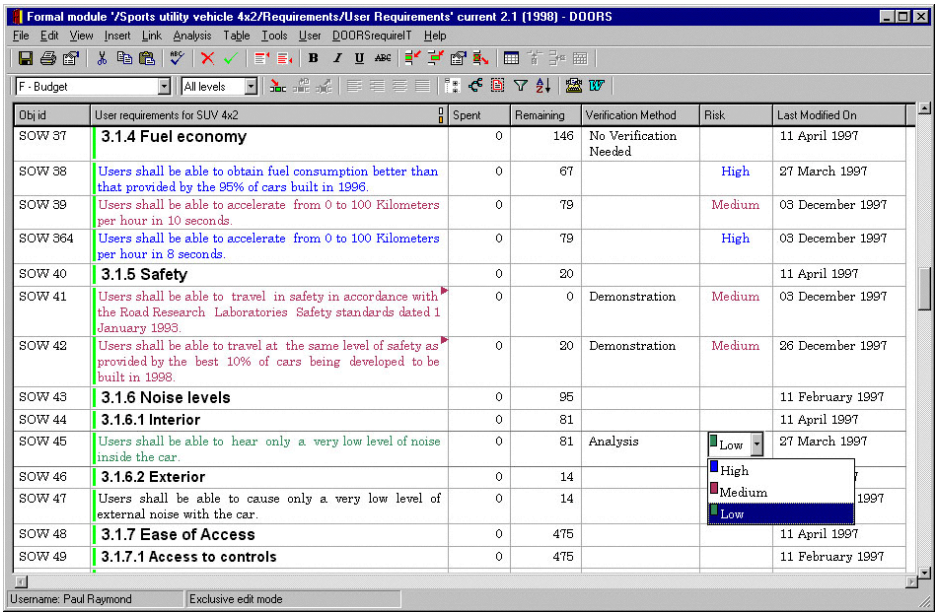
Borland CaliberRM 是一个灵活的，以数据库为中心的需求管理系统。CaliberRM 是 Borland 公司通过收购 Starbase 得到的，而 Starbase 是通过收购 TBI 得到的 CaliberRM。CaliberRM 与 StarTeam 的版本控制和配置管理工具，以及 Borland 的集成建模环境 Together 都能够有效的集成。由于 Borland 的产品是新产品，它多数的集成都是基于触点的，但是我们相信他们在未来一年内会做出改进（Caliber 和 ControlCenter 的集成是一个非常成功的市场迁移集成案例），在 2003 年底以前直接在 Borland 的 IDE 中加入建模和版本控制部分，并在 2004 或 2005 年加入需求和 enforcement 部分。CaliberRM 是十分灵活的系统，它允许用户创建个性化需求收集模板（如极限编程的 Story Card），并提供角色、级别的定制等。系统也有内建的线索型讨论支持，增强了协作。CaliberRM 是一种基于角色的工具，依据角色来分配权力、义务、责任。它支持数据从 Word 中导入，导出至 Microsoft Access，但是当前并不支持 XML。因为它是数据库导向工具，所以必须产生需求文档。CaliberRM 包含一个基于模板的 Microsoft Word 格式文档生成器。Borland 还提供 EstimatePro，使用从需求中抽取出来的数据，形成全面的关于工作量、风险和成本的估算，用于制定需求计划。我们预计 Borland 凭借产品集成的优势和销售渠道的稳固，其总体市场份额会在未来的两年内上升。由于 Borland 明确承诺持续支持 Microsoft 的.Net 平台，同时加上它对 J2EE 和 Linux 的支持，我们认为它的产品线适用于多平台多工具集的组织，尤其是，我们期望看到 Borland 的生命周期工具包能够与 Visual Studio 或是 Eclipse 实现集成。Borland 工具包与 Mercury Interactive 的测试产品能够实现集成，但是与其项目管理工具却不能。

Rational RequisitePro



Rational RequisitoPro 使用底层数据库作为 Microsoft Word 的扩充，提供了一个通用的界面和模板集，以减少转换 Word 文档的复杂度。这也使得在需求文档中使用非文本需求元素（如，模型，图型，图表）变得容易起来。然而，其他工具仅仅提供需求与外部元素之间的链接。同时，因为分解更少而且需求的产生是一个键入文档的过程，所以，对于那些需求收集过程不甚完善的公司来说，这种方法并不那么有效。需求必须包括重要关键的信息，但是在文字处理环境下，需求很容易变得过分冗长。而且，组织要改进需求过程，首先要改进过程和培训，然后使用工具实现自动化和跟踪管理。由于 IBM 即将接管 Rational，投资于 Microsoft 技术的用户应该选择这样一类组合实体，它们承诺持续支持 Visual Studio。公司还致力于对其产品进行再次集成，以支持基于 Eclipse 的 XDE 产品。虽然，由于 XDE 改进的元模型，以及与 Eclipse 开发生命周期其他部分的集成，这一工作的最终结果会更加引人注目，但它同时也是一项相对复杂的工作。Rational 的优势一方面在于其产品线的集成，从而可以推出多种类型的生命周期工具包（如 AnalystStudio），另一方面在于这些工具包与 Rational 统一过程工具的集成。这有助于为各种需求提供最佳的实践和指南（这些需求的收集取决于项目的规模），但其结果却无法支持项目管理集成。

Telelogic DOORS



Telelogic DOORS 是需求管理工具的市场领导者，拥有最完整的生命周期集成，包括与 Telelogic 自己的版本控制和设计开发工具（即 Continuus, Cool）的集成，还包括正在测试的与 Mercury Interactive 的集成，以及与 Microsoft Project 合作的项目管理。与 MSProject 的集成使人们能够看到管理改变对项目日程和整体任务管理产生的影响。当前，Telelogic 的重点在于技术计算市场（如，防御系统，实时和嵌入系统）。

二流供应商

市场上还有许多其他管理工具供应商，我们希望他们大多可以继续以特殊技术（如，工程需求而非软件需求）为重点。当其他工具包供应商（如 CA，Compuware）寻求实现其供应完整性/全面性时，一些二流供应商就可能会被收购。集成的 Chipware RTM 在航空和军事领域有着坚固的应用基础，而且，它不但定位于软件需求，还将工程需求纳入业务范围。这一公司的合作者和集成商日益增多。Holagent 的 RDD-100 提供了使用图形环境获取需求的独立平台，类似于实体关系模型，包括多种基于模型的系统视图。比较来说，主流供应商拥有的是独立的而非集成的建模环境。

Requirements Tools:

- Active!Focus: http://www.xapware.com
- AnalystPro: http://analysttool.com/
- Caliber-RM: http://www.borland.com/products/CaliberRM/
- C.A.R.E. 2.0: http://www.sophist.de
- CORE: http://www.vtcorp.com
- Cradle: http://www.threesl.com
- DOORS: http://www2.telelogic.com/doors/index/cfm
- EasyRM: http://www.easy-rm.com/easyrm.php3
- IRqA (Integral Requisite Analyzer): http://www.irqaonline.com
- Mac A&D and Win A&D: http://www.excelsoftware.com
- Prosareq Requirements Manager: http://www.prosa.fi/eng/pr2Req.htm
- Qualica QFD: qualica.de/english/requirements_mgmt.htm
- RDD-100: http://www.holagent.com/new/products/modules.html
- RDT: igatech.com/rdt/rdtwalkthrough.html
- Reconcile: compuware.com/products/reconcile.html
- Requirement Tracing System: http://www.bandwood.com/cms_exec_summary.htm
- Requirements Manager: http://toolforsystems.com/
- Requisite Pro: http://www.rational.com/products/reqpro/index.jsp
- RMTrak: http://www.rmtrak.com
- RTM Workshop: http://www.chipware.com
- SLATE: http://www.sdrc.com/slate
- SpeedReq: http://www.speedev.com
- SynergyRM: http://www.cmdcorp.com
- Team-Trace: http://www.team-trace.com
- Vital Link: http://www.complianceautomation.com
- XTie-Requirements Tracer: http://tbe.com/products/xtie

Requirements Resource:

- INCOSE (International Council On Systems Engineering) http://www.incose.org/rwg/

Source: META Group

图 4 需求工具和资源

需求和估算

需求的一个作用就是进行软件成本估算，要进行成本估算，就必须明确需求应该达到的细节程度，从而判断项目的成本，判断需求是否必须使用标准的度量指标，如功能点,进行标识。有趣的是，在敏捷过程中，如极限编程，一个关键的因素就是判定项目时间进程的能力。应该明确一点，软件开发是一门非精确性的科学。估算是贯穿项目始终并不断变化的，当开发者对各种各样的问题有了更深刻的认识，当细节从其他阶段被集中起来（建模、开发、测试）时，都要对先前的估算作出相应的修改。对 IT 组织来说，规定具体的有关估算精确度的标准，是十分关键的。主流需求工具要求风险与任务密切相连，不使用具体工具的组织应该识别整体项目中存在的各项风险以及个别特性的特定风险。

软件与系统思想家温伯格精粹译丛

现代需求技术的基石

探索需求

设计前的质量

Donald C. Gause
Gerald M. Weinberg
／著
章柏幸　王媛媛　谢攀／译

Exploring Requirements: Quality Before Design

UMLChina 训练辅助教材

清华大学出版社

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

方面——丢失的链接

Ivar Jacobson 著，summin 译

过去十年来，通过用例清晰地分割源代码的概念，虽然诱人却很难懂；如今，面向方面编程提供了一个实现“关注分离”的途径。

人类通常先理解具体事物再做出抽象。另一条路则是，把抽象变为某种具体的东西，这需要一种严谨的信念——只有当你做出的抽象可信时，你才能鼓吹它。

我做这件事很多年了。当我们使用汇编语言作为开发环境中，要想把构件和接口具体化几乎是不可能的，但我们做成了。现代语言，如 Java 和 C#可以实现这种抽象（构件和接口）的具体化，因而我们不能吹嘘的太多。类似的，扩展机制的具体化却是一个挑战，我们没有现有的编程语言支持它。即使是 UML 也只是在用例建模时支持它，许多作者甚至于坦率的反对这种机制。有了面向方面的编程，我们就有了最终能实现扩展具体化的工具。使用如 PARC 的 AspectJ 或 IBM 的 Hyper/J 的面向方面的编程语言，在到代码的所有途径中支持扩展。

为本序列的上一篇文章《一个方面的案例》做出归纳：为了以需求到可执行代码的所有方式分割和组合用例，即在这期间保持住用例这个逻辑单元，我们必须处理对等用例和扩展用例。扩展用例在 20 年前已经提出了，现在受到了 AOP 的支持。开发人员迄今为止一直通过手动处理重叠行为管理对等用例。如大家所看到的，AOP 的一大特点是可以在多个尺度的分离关注（MDSOC）上工作，HyperJ 还支持对等用例的分割。

自从 1986 年用例被引入到需求规约中，很快，用例就成了这方面的流行工具。在需求获取阶段开始用例，在分析和设计阶段被转化为协作关系，并在测试阶段转化为一种变体——测试用例。因此，我们可以把系统概念化，就如同把一条面包切成片。使用用例，我们能够通过生命周期各个阶段的元素把系统切成“用例片”，几乎所有的阶段都可以做到。为何是“几乎”，而不是全部呢？有两个原因：首先，构造一个构件或类，我们必须合并来自多个用例的代码，最终这些独立的块已无法辨别。其次，UML 中的用例扩展机制（通过<<extend>>关系表现）却无法被分析元素和设计元素（类、构件等）的“协作关系”所支持，更不能被 Java、C#等实现语言所支持。根本的问题是当前实现语言的局限性造成的。

面向方面的编程是一个“丢失的链接”，支持我们对系统进行清晰的“切片”，一个用例一个用例的“切割”系统，这种片段则穿越多种模型（用例模型、分析模型、设计模型、实施模型等等），因此用例在到代码的所有途径上保持着独立，因为我们到达了“分割关注”的目的，“分割关注”是 AOP 的一个原则。事实上，我们通过用例模块对系统进行横切，用例模块则穿越了构件模块，之后，我们再把这些片段重新组装为整体——可部署的系统。这种这种类似于扩展机制的概念就是 AOP 的基础，这就可以部分的支持 UML 中的扩展用例了。因此，感谢 AOP，这些概念现在可以有所收获了。

今天，工作在用例上

为了理解 AOP 能为我们做些什么，我们先要了解当前用例驱动的开发的现状。在最简单的情况下，开发的顺序是从用例到设计再到实施。

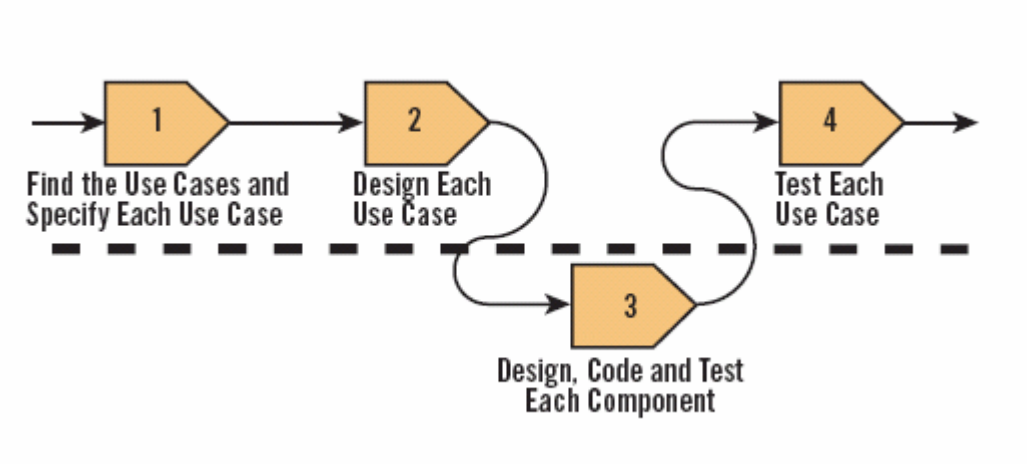


图 1. 工作在用例上的现状。虚线之上是用例的工作，之下则是构件的工作。

在软件生命周期的每次迭代中，开发团队的工作贯穿了一系列的活动，如图 1 所示。

1. 发现用例；详细说明每个用例。
2. 对每个用例做设计
3. 对每个构件进行设计、编码、测试
4. 测试每个用例

（注意：活动 1、2 和 4 都属于用例的工作，而活动 3 属于构件的工作）

从现在开始，我们把“构件”这个术语作为实现元素的统称，如：类、子系统、服务及物理构件等。通常，这四个活动分别表现为开发团队所承担的工作。除了活动 3（对每个构件进行设计、编码、测试）以外，其余活动都以用例为基础，因此这就是术语“用例驱动的开发”的含义。

当然，更多的工作在一次迭代中完成，例如：构架分析和设计，但就这篇文章的目的，我们没有必要走得太远，我们的线索是用例、用例实现和构件。

用例：从经验出发，用例是系统执行的一系列动作，这些动作将生成特定主角可观测的结果值。正式的定义是，用例是一个类似于类（class-like）的结构，描述了特定主角或用户的一组对系统相关的使用。

用例既可以是具体的，也可以是抽象的。具体的用例可以被实例化。例如：电话呼叫的用例是抽象的，而使用某种协议进行电话呼叫的用例则是具体的。当用户进行电话呼叫时，后者的实例就创建了。

用例模型包含了主角、用例和它们间的关系——这是一种需求模型。为了简化用例模型，用例间的关系应该服从一种语言的约束，才是可接受的关系。目标是分离关系，每个用例都表达了一组涉众所关注的利益。因此，用例间的关系，作为与类相似的现象所具有的关系，是唯一可接受的。

眼下，让我们把目光转到两类关系上：一般化，一种具体用例到抽象用例的关系；“扩展”，在基础用例的扩展点上增加行为，而不改变基础用例的一种方式。附加的行为被描述为扩展用例。在基础用例中，扩展点在用例描述中是明确被引用的，可能使用 before 和 after 这类的限定词。在这些扩展点上，当基础用例被解释时，扩展行为就被插入了。

用例实现：用例模型是系统的外部景象，并不表达任何系统内部的构造。系统的内部是在做设计模型时被引入的，用例模型的每个用例都对应一个设计模型的用例实现。用例实现是一个 UML 协作描述（例如：使用序列图），实现用例要求哪些构件参与、构件如何交互、担任什么职责。每个用例都是用例模型中不同的关注点，每个用例实现也是设计模型中不同的关注点，用例实现涉及到许多构件（散射），一个构件则包含多个用例实现的代码片段（混乱）。

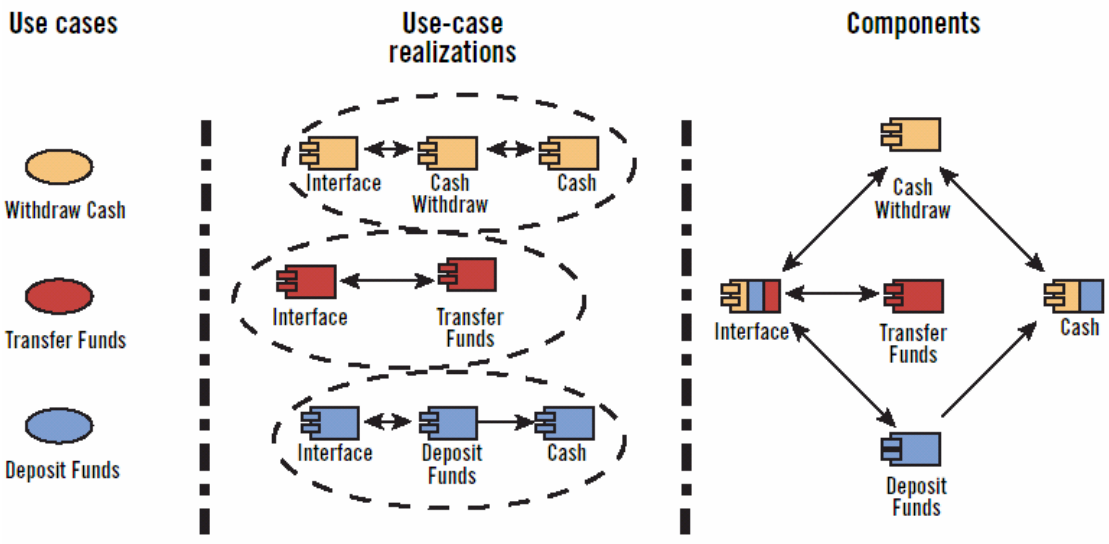


图 2 散射和混乱。用例实现涉及到多个构件，从而产生了“散射”；构件混合了多个用例的代码片段，因此陷入“混乱”。

图 2“混乱和散射”描述了用三个用例（不同的关注点）对 ATM 系统的建模过程。每个用例被单独设计，其结果是一个用例实现。最后，每个参与实现用例的构件都被设计、实现和单元测试。然而，“散射和混乱”并不能显示对等用例的重叠，也不能显示构件 Cash 有分别来自“Cash Withdraw”和“Deposit Funds”的重叠片段，三个用例在接口构件上的重叠也同样不能显示。如果用例不重叠，我们的问题就容易解决了；若重叠存在，就必须分解它。

用例间的一般化机制被传播到 UML 的协作中，因而，一个用例实现可以复用更抽象用例的用例实现。

然而，用例间的扩展机制不能用以上协作来实现，我甚至不能简单地用一个例子来说明这情况，在 AOP 以前，没有程序语言支持实现扩展机制。因而，目前在设计和实施阶段，都不能把扩展用例从基本用例上分离出来。实现扩展用例的代码必须“溶解”到实现基本用例的代码中，基本用例因此不能“忘记”扩展用例。因此，我们没有从用例建模到设计的“无缝”推移技术——扩展用例的实现必须混在基本用例的实现中。

构件：以用例实现为基础，工具能够汇总每个构件在所有用例上的职责，产生构件的规格说明。由于构件参与实现用例，而职责又直接来自用例，因此为构件收集、汇总职责是很容易做到的。

现在，构件的拥有者必须实现构件的所有职责，并安装到统一的环境中。进一步的商讨包括：调和不同用例的需要（例如：调和重叠操作），解决冲突（例如：并发、死锁产生的问题）。最后，每个构件都要进行单元测试，判断是否违反了规格说明。

当前，一个主要的问题是，传统语言不支持分割关注，因此不同用例对一个构件的影响不能被分离出来，不能维持对系统独立的“切片”。

明天，在用例和方面上工作

通过使用方面，我们的开发方法将有实质性的进步。我们可以在到达可执行代码的所有途径上保持用例的独立。在继续之前，让我们重新回顾我们的关键结构，论证它们通向 AOP 的方式。

用例：用例是最直接横切关注的结构，这种具有实效的结构可以派发出方面，但用例不是语言结构，用例对系统的“切片”可以通过多种构件予以实现。用例可以构建在符合某个标准的层上：特殊应用（如：银行的资金管理）、通用应用（可重用的银行系统框架）、中间件和系统件。我们通常在应用上考虑用例，在特殊应用层或通用应用层上构建用例。

然而，在底层的基础结构上，软件同样可以包括用例；这种用例和分布式、持久化、执行监控、审计等有关。在 AOP 中没有任何一种结构与用例相符，也不应该有，因为 AOP 是一种编程技术。然而，AOSD，或者说的更清楚，MDSOC 和 Hyper/J 具有符合用例模块的结构。一个用例模块包含一个用例实现和一组实现该用例的协作构件片段。MSDOC 中的一个 hyperslice 就是一个用代码实现用例的结构。

用例实现：用例的实现是一个典型的把系统分解到构件上的过程，通过 UML 的“用例实现”进行建模——“用例实现”这种协作可被看作设计模型中的某种模块单元，把系统实现切割为构件。在 AOP 的环境中，用例实现既可以是基本程序（即对等用例的实现，对等用例是系统主要的分解）也可以是方面（实现扩展用例）。

方面是实现横切的模块单元，“扩展”可以直接对应到方面（参看表 1“跨年代的转换”）。

表 1. 跨年代的转换——把 OOPSLA'86 上的论文中的结构对应到 AOP

回到我在OOPSLA'86上发表的论文"Language Support for Changeable Large Real Time Systems"中，基础结构在AOP中都有对应。这张表是Karl Lieberherr在一封私人信件中提到的：

OOPSLA'86 的论文	**AOP 的等价物**
存在	基本程序（base program）
扩展	方面（aspect）
扩展不改变基本用例	基本程序"忘记"方面
扩展点	加入点（join point）
扩展点列表	加入点集合

——Ivar Jacobson

UML 现在需要把增加用例实现对"扩展"的支持。扩展点应该可以从用例传播到用例实现之间。

构件：此外，我们还应该在 UML 中增加"扩展"到构件的支持。在我们的编程环境中加入对"扩展"的支持是很容易接受的。扩展点可以通过用例实现传播到参与实现此用例的各个构件中（包括类、子系统、物理构件等）。

一些用例实现可能需要多个构件或单个构件中的多个扩展点，例如：扩展基本用例是调试跟踪的用例意味着实现调试跟踪这个基本用例的构件都需要被扩展，这样，就是多个构件上的多个扩展点的情况。

扩展点："扩展"和扩展点在 AOP 中都有对应物。

1、加入点是程序执行中被精确定义过的点。并非每一个执行点都是一个有效的加入点，好的编程实践总是需要一些约束的。

2、一组用例实现上的及其协作构件上的扩展点就是 AOP 中的加入点集合。加入点集合被定义在加入点模型中，加入点模型也包括选择加入点的机制（例如：使用正规表达式）。这对于用例驱动的方法来说是非常重要的贡献，有些东西是我 1986 年的论文中所缺乏的。

3、扩展行为（一个用例片段）在 AOP 中是一个"通告"（Advice）。"通告"是每个加入点所执行的代码，被定义在加入点模型中。"通告"能够对加入点以三种方式执行：before、after 和 around。扩展行为/"通告"在被扩展的元素/加入点的环境中执行。

为了向将来前进，我们必须把一个极为关键的元素加入到 UML 中：在其他结构中，具有扩展点的扩展关系应该得到传播，从用例到协作，再到类、操作等等，事实上，UML 中所有具有重大意义的语言特性都应该涉及到（例如：子系统）。

我们从方面获得了什么

一种新的模块（MDSOC 中的特殊情况）正在出现：用例模块。按照早先的讨论，我们需要两类模块：用例模块和构件模块。用例模块会分割构件模块。这两类模块承担不同的角色将并肩生存下去。构件模块为系统提供静态结构；用例模块则为这种静态结构提供行为。开发者可以选择不同视角观看系统——用例景象或构件景象，并工作在选中的景象上。编程环境将会把一个景象上的改变传播到另一个景象上。

由于具备了保持用例独立的可能，我们还是可以就像以前一样工作在用例上：1、查找用例，并详细说明每个用例；2、设计每个用例；4、测试每个用例；但活动 3 应替换为：3a、对每个用例编码，3b、组合每个构件的用例片段。（参看图 3）

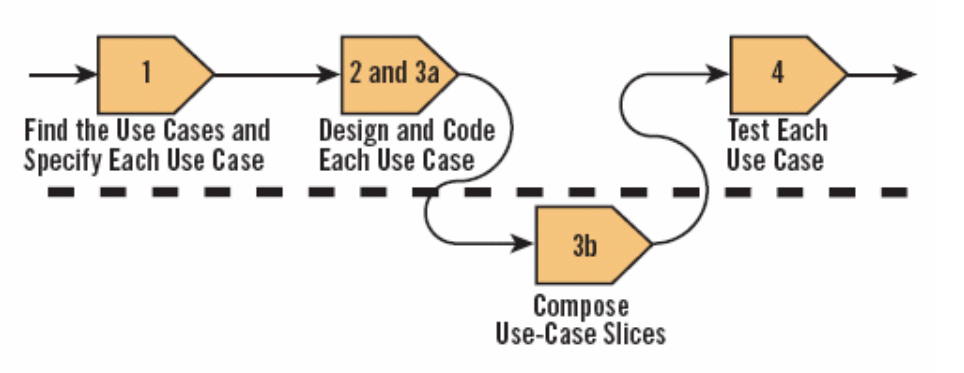


图 3 局部用例模块性

由于活动 2 和活动 3a 都由用例设计者完成，因而活动的次序应该是：

- 查找用例，并详细说明每个用例
- 设计每个用例并对用例编码
- 组合用例片段（一个构件活动）
- 测试每个用例

构件负责人的工作有实质性的改变：他不再对构件编码（这是用例设计者的责任），也不再测试构件（构件测试被集成到了用例测试中），但他必须调和对等用例的不同片段。将来，我希望这部分工作可以用新的工具来执行，只要相关的用例设计者相互协作就能完成。编程环境将可以通过构件把横切系统的用例装配为一个整体；这个整体既可以是预编译构件，也可以是可执行构件，还可以是正在执行的构件——这个整体依赖于装配的时机。此外，构件负责人有责任集成构件的数据（低级类或属性）。

缩减或者潜在地去掉了构件设计、实现和测试这些独立的活动之后，这就是一个流畅的软件开发过程。假如我们能够去掉所有构件的工作，则原来所需的 4 个活动就变成了 2 个活动：1、查找用例，并详细说明每个用例；2、对每个用例进行设计、编码和测试。（参看图 4）

图 3 完整的用例模块性

因而，每次迭代都是一个用例接一个用例地到达代码（甚至于可能是在运行时动态的执行）。此时，我不知道我们能走多远，但我相信使用方面维持用例到代码的独立性将实质性的缩减开支，而开发也将变得流畅起来。

准备前进

用例驱动的开发和 AOP 都不是“银弹”，它们只是两种最佳实践而已。然而，我相信，把它们结合在一起，就可以戏剧性地增强软件的开发方式，支持我们把系统“切割”成用例这种系统片段，并在到达代码的所有途径中保持这种系统片段（用例）。

在这之后，我们可以把这些片段重新组装成整体：正在运行的系统。软件更容易开发，具有高质量，当然更便宜，构造也更加快捷。现在，就是把球转起来的时候了。

致谢

非常感谢Pan-Wei Ng、Karl Lieberherr、Harold Ossher、Mauro Assano、 Kurt Bittner、 Magnus Christerson、Gunnar Overgaard、 Ian Spence及Dean Whampler的支持和建议。最后感谢Alexandra Weber Morales的文字录入。

Domain-Driven DESIGN

Tackling Complexity in the Heart of Software

众多问题根源在于
领域模型没有捕获到
业务的深层内涵

Eric Evans

Foreword by Martin Fowler

《领域驱动设计》中译本

即将由清华大学出版社出版，UMLChina 审稿

从 UML 状态图产生代码

Iftikhar Azim Niaz、Jiro Tanaka 著，赵成 译

摘要

统一建模语言（UML）中的状态图是一种描述反应对象(reactive object)动态行为的有力工具。由于状态图的动态特性（dynamic nature），从状态图产生代码是一项有挑战的工作，而且很多状态图的概念不被 OO 编程语言支持。大多数实现 UML 状态图的方法不是存在维护上的问题，就是只实现了 UML 状态图的一个子集。

本文提出了一种使用设计模式在 OO 语言如 Java 中产生有效和紧凑的可执行代码的新方法。在我们的方法中，状态图中的每一个状态成为一个类，它封装了该状态的所有转换和动作。导致转换的事件被明确的表示，而不需要使用任何 if 或 case 语句来判断，这使得产生的代码是可读的、紧凑的和有效的。产生的代码易于维护。

通过将状态表示为对象，我们用对象聚合和委托概念扩展了状态设计模式，从而实现了顺序子状态和并发子状态。我们也提出一种实现复合转换（分叉/结合）和历史节点的方法。这种方法很好地处理了大多数状态图的特性。

关键词

软件工程、面向对象的分析和设计、状态图、状态模式、对象组合、Java

1 介绍

作为一种系统建模的工业标准，UML 的出现促进了自动化软件工具的使用，这些工具便利了从分析到编码的开发过程。在基于 UML 的面向对象的设计中，行为建模使用状态机描述对象的行为。状态机定义了一个对象在它的生命周期中根据外部事件做出响应同时改变状态的过程.UML 状态图是一个表现状态机的图形[1]。UML 状态图中的语义和符号主要来自 Harel 的状态图[2]，并对它们做了面向对象的扩展。附属于类的状态图指明了该类对象的所有行为。

使用状态图的 OO 方法学充分的叙述了描述对象行为所遵循的步骤，但是未能描述在 OO 语言中状态图的实现，这是因为缺乏对状态图的语法支持。高层次的建模语言和编程语言之间存在一个鸿沟。从类图到 OO 编程语言的转化是简单的，并被多数 CASE 工具支持。为了实现面向对象系统的行为，必须要实现描述类动态行为的状态图。我们的工作是在设计和实现之间架起一座桥梁。通过在 Java 语言和 UML 之间的映射，我们可以直接从状态图产生低层次的 Java 代码。我们的主要目的是在 OO 语言，如 Java 中，展示一个简单和有效的 UML 状态图实现方法。本文所提出的实现技术是有价值的，因为它们提升了抽象层次，并且支持直接将状态图映射到紧凑和有效的代码。

目前存在一些用 OO 语言实现状态图的方法。最通常的实现技术是双重嵌套 switch 语句，外层的变量表示状态，第二层的变量表示事件。此方法对于简单状态图是有效的，但是手工编写 entry/exit 动作和嵌套子状态代码比较麻烦，这主要是因为属于一个状态的代码是分散的并在很多地方有重复，这使得当状态图的拓扑结构改变时，修改和维护代码十分困难。在[4]和[5]中检查了状态和类之间的关系。他们建议通过继承其它的状态机来重用状态机的行为。他们通过为父状态建立一个表格实现了嵌入的状态，但没有考虑并发状态、历史状态和组合状态。在[6]中，状态图的所有状态被作为类的常量属性，类的其它属性被用来跟踪对象的当前状态。状态之间的转换是通过将对象的一个状态属性赋值为另一个新的状态值。事件和动作用方法来实现。在代码中，动作被转换为一次方法调用。这种方法只能处理简单的状态图。

设计模式[7]在 OO 的软件设计中被广泛使用。它指明了可重用的对象或类之间合作和交互机制以解决任何领域内的常见的面向对象问题。有几种设计模式可以用来实现状态图。条件语句（Conditional Statements）模式[3]、 状态表（State Table）模式[3]、 基本状态图（Basic StateChart）模式[8]、HSM 模式[9]、和状态模式[7]。这些模式对于支持子状态、从图形到代码的映射和代码的可读性方面有严重的问题。HSM 模式仅仅支持顺序子状态。

我们的工作关注于实现阶段。我们在寻找一种代码产生工具，它支持从状态图到 OO 语言如 java 的自动转换。这篇文章提出一种新的方法，缩短了 UML 状态图与系统实现之间的鸿沟。此目的是通过直接从状态图产生低层次的 java 代码来实现的。代码的产生是通过创建 UML 状态图和 java 语言之间的映射实现的。

在这篇文章中，我们将描述如何通过对象组合和委托扩展状态模式,从而在 java 语言中实现顺序子状态、并发子状态和复合转换。

2 顺序子状态

我们使用空调系统作为例子来演示我们的方法。

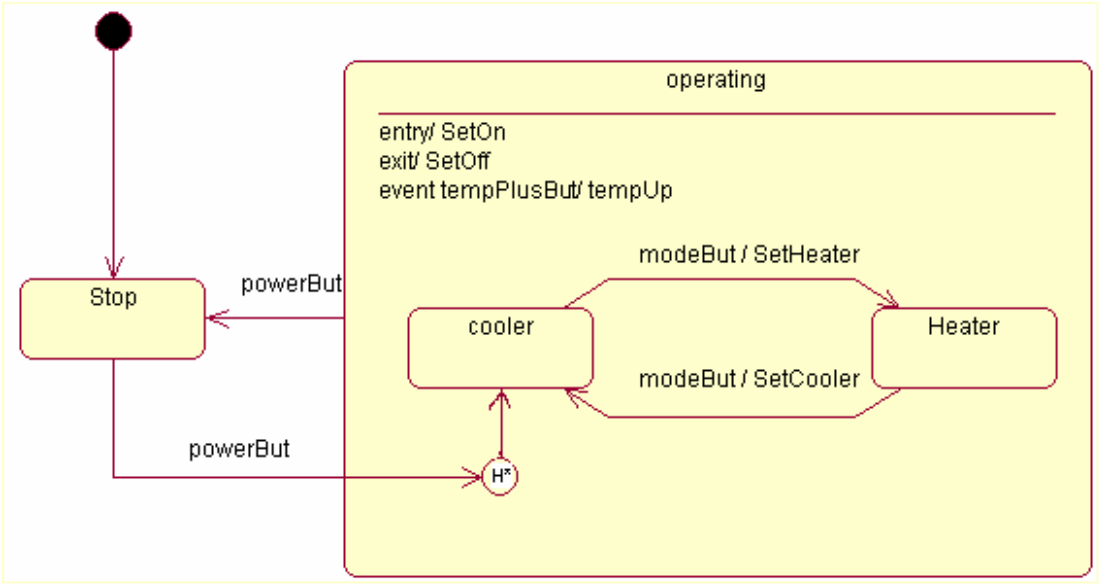


图 1 空调系统的状态图

空调的基本行为在状态图 1 中表明。它包含两个顶层（top-level）状态 Stop 和 Operating。开始时，空调处于 Stop 状态，在这个状态它接收 powerBut 事件。收到此事件后，空调做出反应，将状态从 Stop 转换到 Operating。Operating 状态是一个组合状态，Cooler 是它的初始子状态。Heater 状态是另一个顺序子状态。处于 Operating 状态时，收到 modeBut 事件后，空调将切换到下一个子状态。收到 powerBut 事件，空调将切换到 Stop 状态。再发送一个 powerBut 事件，将再次激活空调。当空调被再次激活后，它将切换到 Operating 中的历史子状态。历史子状态记录了空调从 Operating 切换到 Stop 状态前的最后活动子状态，并且当空调再次被激活时，转换到这个子状态。作用在父状态上的转换对于父状态内的所有嵌套状态都是有效的。

2.1 实现策略

我们实现 UML 状态图的方法是基于[10]、[11]、[12]和状态模式[7]。状态模式中将与一个特定状态相关的所有行为放入一个类中。在我们的方法中，状态图中的每一个状态成为一个类。该状态上的每一个转换，成为对应类上的一个方法，并且每一个动作成为环境类（Context Class）的一个方法。状态图描述了环境类的行为。一个环境类将所有的事件委托给当前状态对象来处理。状态转换是通过改变当前状态对象来完成的。状态模式没有处理顺序子状态和并发子状态。因此我们需要某种机制来实现状态的层次结构。对象的组合是在运行时刻通过一个对象获取其它对象的引用来动态定义的。对象的组合保持了每个类的封装性，使类之间的依赖性较少。委托的主要优点是，它使得在运行时刻组合行为和改变组合的方式变得容易。

当一个组合状态活动时，有且只有一个它的顺序子状态是活动的。顺序子状态显示了组合状态中的一个嵌套的状态图。这使得我们可以通过对象组合和委托扩展状态设计模式，从而实现组合状态。这样，组合状态成为嵌套状态图的环境。一个抽象状态类定义组合状态中顺序子状态的行为接口。顺序子状态将成为具体状态类。大多数时候复合状态类具有控制权,在涉及子状态的转换的时候将请求委托给子状态处理。图 2 显示了我们实现方法的类图。

AirCon类是图 1 中的环境类。每个活动作为环境类上的一个方法。State对象是当前活动状态的引用。通过在AirCon对象中提供一个opHistory引用实现历史节点。开始时，环境类设置opHistory指向*Cooler*状态，随后在组合状态*Operating*的exit()方法中设置opHistory指向当前活动子状态。当*Operating*状态成为活动时，它最近一次的活动子状态也被设置为活动。AirConState和AbsOpState类是抽象状态类，它们为具体状态类提供了通用接口。所有的具体状态类拥有环境对象的引用。AirCon（环境）对象将所有的到达事件委托给它的当前状态对象（state）处理。在处理转换时，具体状态对象首先执行当前状态的离开动作，然后调用Aircon对象的setState()方法设置新的状态。在setState()，新状态的进入动作也被执行。Operating类成为嵌套状态的环境。Operationg对象属性中的subState对象是当前活动子状态的引用。Cooler和heater对象拥有Operating和AirCon两个环境的引用。如果转换的目标是组合状态，转换将被组合状态执行，但是如果目标是子状态，组合对象将把请求委托给当前活动子状态处理。活动子状态将执行转换上的动作，接着执行离开动作，最后调用Operating对象的setSub()方法设置下一个子状态。内部转换*tempPlusbut*被Operating类实现。tempup动作作为环境类AirCon的一个方法。

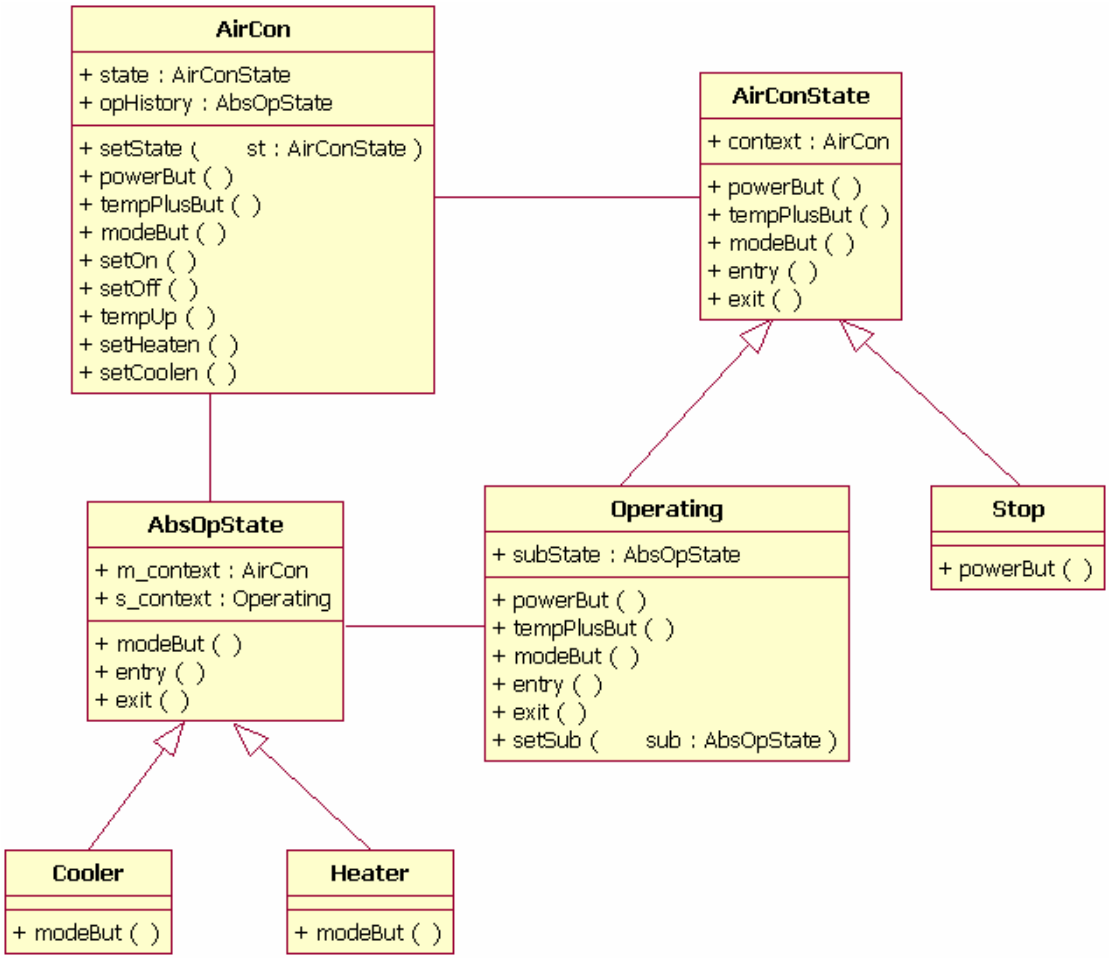


图 2 实现顺序子状态的类图

3 并发子状态

在这一节将描述用我们的方法来实现并发子状态。如果组合状态包含并发（正交或 AND）子状态，那么当组合状态变为活动时，它的子状态同时变为活动。这些子状态指定了两个或多个在父状态中并行执行的状态机。图 3 显示了一个更加详细的包含空调并发子状态的状态图。*Mode* 和 *Speed* 是 *Operating* 组合状态的两个并发区域。

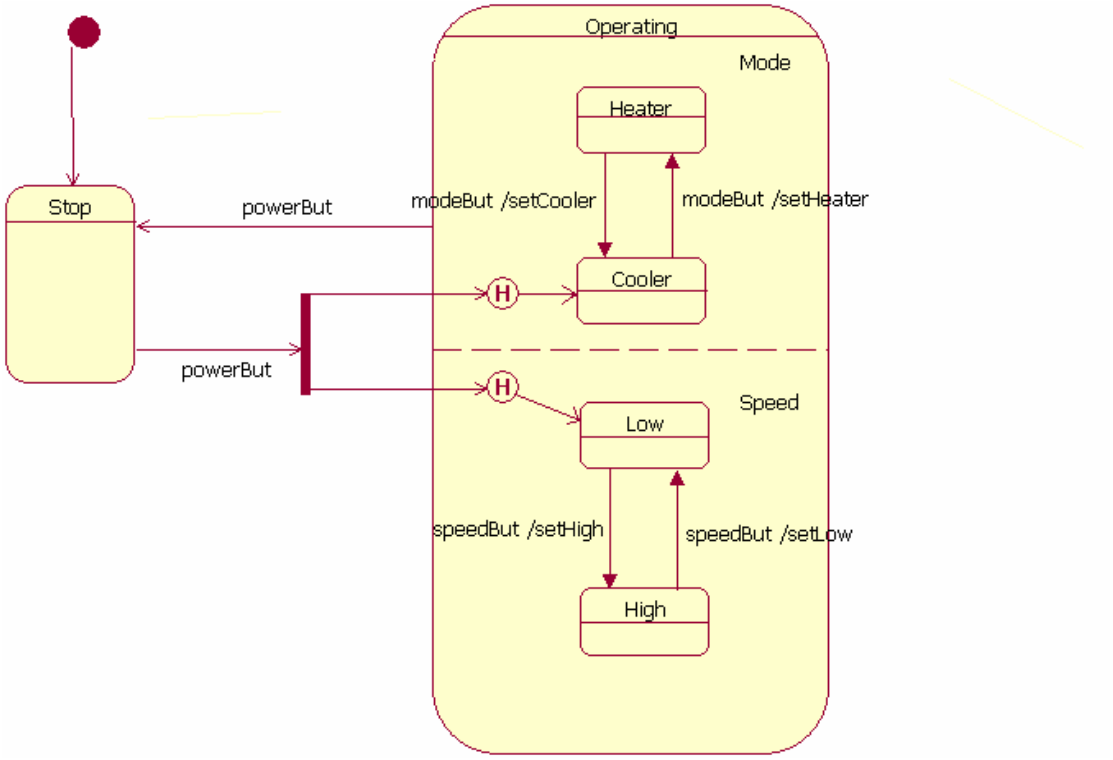


图 3 有并发子状态的状态图

3.1 实现策略

并发子状态显示了组合状态中的一个嵌套状态图。

这使我们可以通过用对象组合和委托扩展状态模式，从而实现并发子状态。*Operating* 状态将成为并发区域 *Mode* 和 *Speed* 的环境。*Mode* 和 *Speed* 将成为抽象类，并将定义在各自并发区域中的嵌套顺序子状态的行为接口。

图 4 显示了该实现方法的类图。

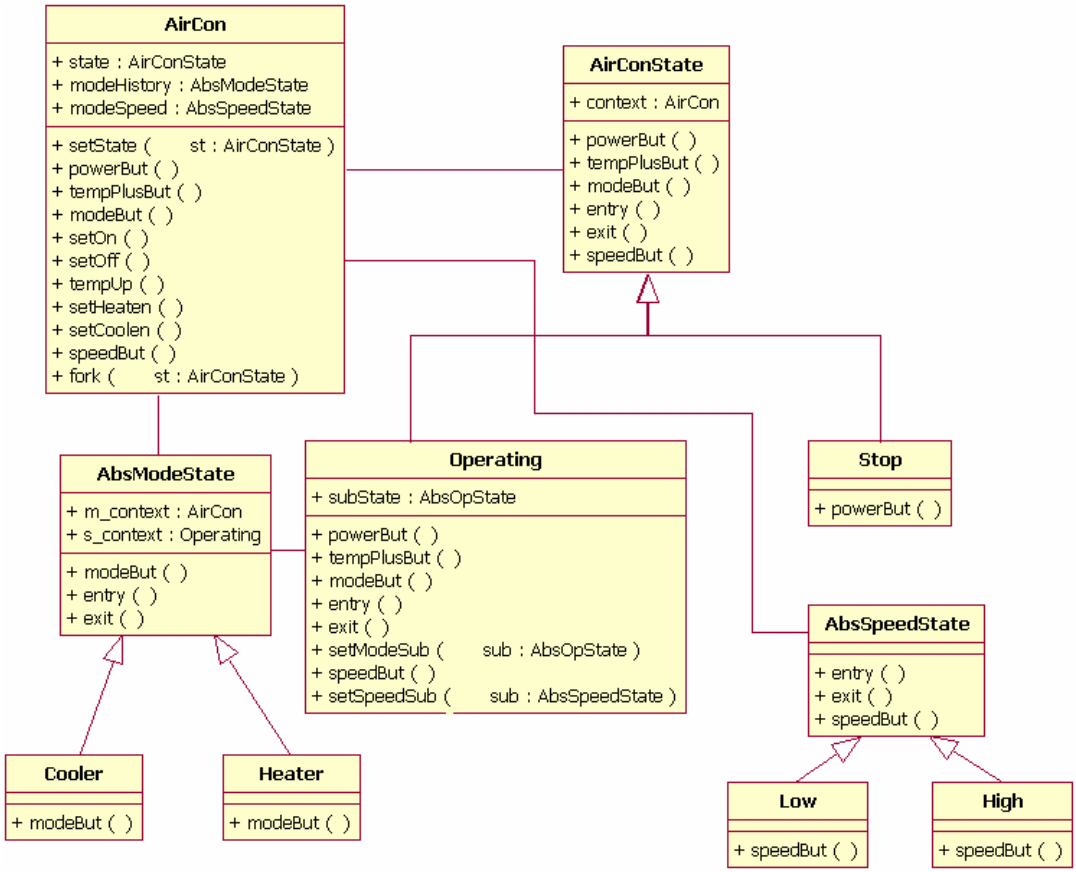


图 4 实现并发子状态的类图

如图 4 所示，*Operating*类成为*Mode*和*Speed*这两个并发区域的环境。AbsModeState和AbsSpeedState类定义了两个并发区域的接口。Cooler、Heater、Low和High成为Operating组合状态中的具体子类。Operating对象的属性modeState和speedState是各自并发区域的当前活动子状态的引用。具体类对象将保持两个环境的引用Operation和AirCon。通过在AirCon对象中提供modeHistory和speedHistory引用实现历史节点。处在*Stop*状态时若收到*powerBut*事件将分叉进入*Operating*状态的两个并发子区域，因此*Stop*状态负责激活两个并发区域中的合适的子状态。*Stop*状态调用环境的fork()函数，使得所有的，而不是一个，目标状态成为活动。当处于*Operating*状态时，若收到modeBut或speedBut请求，Operating将请求委托给当前活动的子状态处理。活动子状态将执行转换对应的活动，接着调用Operating对象上对应的子状态设置方法改变子状态。

4 复合转换

复合转换是由一个或多个转换组成的路径，它有一组源状态和一组目标状态。当所有的源状态成为活动时，复合转换被触发。当复合转换被激发后，它的所有目标状态都成为活动的。复合转换的图形用短条表示。

分叉(Fork)

分叉是指有一个源状态和两个或多个目标状态的转换。如果在源状态活动时出现触发事件，则转换动作被执行并且所有的目标状态都成为活动的。

结合(Join)

结合是指有两个或多个源状态和一个目标状态的转换。如果所有的源状态活动时出现触发事件，则转换动作被执行并且目标状态成为活动的。

图 5 显示一个带有复合转换的状态图。分叉的触发事件是 t1。这意味若状态 A 活动且事件 t1 发生，则在两个并发区域中的 C 和 E 状态将同时成为活动。图 5 也包含一个从状态 D 和 F 到状态 B 的结合。仅当 D 和 F 是活动的，转换 t4 才被激发。如果状态 D 和状态 F 不是活动的，那么事件被忽略。

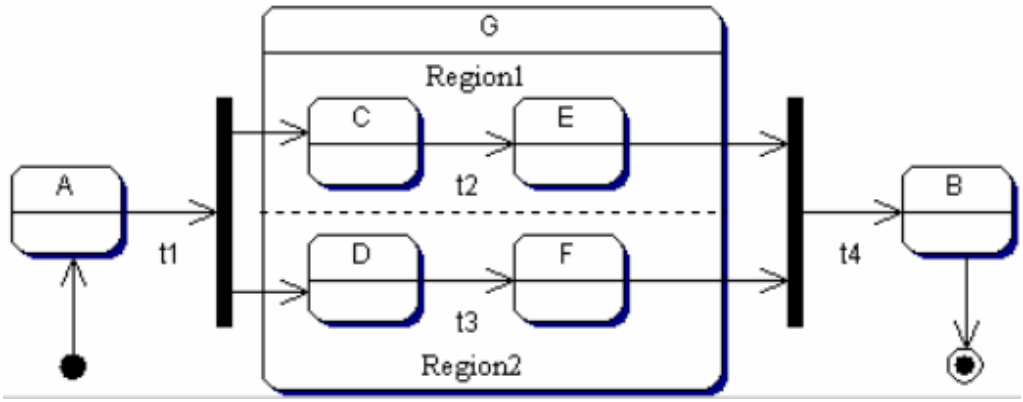


图 5 带有复合转换的状态图

4.1 实现策略

分叉是容易实现的。源状态使得多个，而不是一个，目标状态活动。为了实现结合，我们必须在转换激发之前确信所有的源状态是活动的。图 5 的一种 Java 代码实现如下所示。我们假定环境类的名字是 Test。

```
class Test

{

TestState state; // state object

// References for all the state objects

A aState; B bstate; G gState;    C cState;
```

```
D dState; E eState; F fState;
void setState(TestState st)
{
    state = st;     // setting new state
    state.entry();// executes the entry action
}
void fork(TestState st)
{
    // setting the concurrent states
    if (st.equals(gstate))
    {
        gstate.region1State = cstate;
        gstate.region2State = estate;
        ...
    }
} // End of Test Class
}


class TestState
{
Test ac; // Reference to the context object
// declaring abstract methods
```

```
…..
}

class A extends TestState
{
void t1()
{   // implementing Fork
    exit(); // executes the exit action of current state
    ac.fork(ac.gState);// setting substates to be active
    ac.setState(ac.gState); // change to G composite state
}
…..
}

class B extends TestState {….}

class G extends TestState
{
AbsRegion1State region1State;
AbsRegion2State region2State;
void entry()
{
```

```
        region1State.entry();
        region2State.entry();
    }
    void t4()
    {
        region1State.exit();
        region2State.exit();
        exit();
        ac.setState(ac.bState); // sets the new state
    }
    void t2() { region1State.t2(); }
    void t3() { region2State.t3(); }
    void setRegion1(AbsRegion1State subRegion1)
    {
        region1State = subRegion1;
        region1State.entry();
    }
    void setRegion2(AbsRegion2State subRegion2)
    {
        region2State = subRegion2;
        region2State.entry();
    }
```

```
…..
}

class AbsRegion1State
{
Test m_context; // super Context object
G s_context;    // sub Context Object
// defining abstract methods
}

class C extends AbsRegion1State
{
void t2()
{
    exit();
    // changes to next substate
    s_context.setRegion1(m_context.dState);
}
….
}
```

```
class D extends AbsRegion1State
{
void entry()
{
    // implementing join
    if (s_context.region2State.equals(m_context.fState)
    {
        m_context.t4(); // triggers transition
    }
}
…
}

class AbsRegion2State
{
Test m_context;
G s_context;
// defining abstract methods
}
```

```
class E extends AbsRegion2State
{
void t3()
{
    exit();
    // changes to next state
    s_context.setRegion2(m_context.fState);
}
…
}

class F extends AbsRegion2State
{
void entry()
{
    // Implementing join
    if (s_context.region1State.equals(m_context.dState)
    {
        m_context.t4(); // triggers the transition
    }
}
…
```

```
};
```

在具体类A的t1()方法中实现了分叉。事件t1 触发后，首先状态*A*的离开动作被执行，接着调用环境类Test的fork方法设置两个并发状态*C*和*E*，接着状态从*A*转换到组合状态*G*并且*G*的entry动作被执行，在此动作中依次执行了并发状态*C*和*E*的entry动作。

为了实现结合，我们必须在转换激活之前确信所有的源状态是活动的。这些可在源状态*D*和*F*类的entry方法中完成。在entry方法中如果检测到另一个状态是活动的，那么将触发外层环境类Test上的对应事件处理方法，该方法将该事件委托给当前活动的事件G处理。t4()首先执行子状态的exit动作，接着执行自身的exit动作，最后将当前状态设置为B并执行该状态的entry动作。

我们也是使用 J[14]中的 Rhapsody 产生了图 5 的代码。代码的片断如下所示：

```
class Test implements RiJStateConcept
{
Reactive reactive;
static final int RiJNonState=0; static final int G=1;
static final int Region2=2;     static final int F=3;
static final int E=4;           static final int Region1=5;
static final int D=6;           static final int C=7;
static final int B=8;           static final int A=9;
protected int rootState_substate;   protected int rootState_active;
protected int Region1_substate;     protected int Region1_active;
protected int Region2_substate;     protected int Region2_active;
```

```
Test(RiJThread p_thread)
{//constructor
    reactive = new Reactive(p_thread);
}
boolean startBehavior()
{
    boolean done = false; done = reactive.startBehavior();
    return done
}
class Reactive extends RiJStateReactive
{ // inner class
    Reactive(RiJThread p_thread)
    { // constructor
        super(p_thread);
        initStatechart();
    }
    boolean isIn(int state)
    {
        if (Region1 == state) return isIn(G);
        if (Region1_substate == state) return true;
        if (Region2 == state) return isIn(G);
        if (Region2 substate == state) return true;
```

```
        if (rootState_subState == state) return true;
        return true;
    }
    int rootState_dispatchEvent(short_id)
    {
        int res =RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
        switch (rootState_active)
        {
            case A : { res =A_takeEvent(id); break; };
            case G : { res =G_dispatchEvent(id); break; };
            case A : { res =B_takeEvent(id); break; };
            default : break;
        };
        return res;
    }

    int G_dispatchEvent(short_id) { …. }
    int Region1_dispatchEvent(short_id) {….}
    int Region2_dispatchEvent(short_id) {….}
```

```
protected void initStatechart()
{
    rootState_substate = RiJNonState;
    rootState_active = RijNonState;
    Region1_substate = RiJNonState;
    Region1_active = RijNonState;
    Region2_substate = RiJNonState;
    Region2_active = RijNonState;
}
……….
int DTaket4()
{
    int res =RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
    if (isIn(F))
    {
        G_Exit(); B=entDef();
        res = RiJStateReactive.TAKE_EVENT_COMPLETE;
    }
        return res;
}
```

```
int D_takeEvent(short id)
{
    int res =RiJStateReactive.TAKE_EVENT_NOT_CONSUMED;
    if (event.isTypeOf(t4.t4_Default_id))
    {
        res = DTaket4();
    }
    if(res==RiJStateReactive.TAKE_EVENT_NOT_CONSUMED)
    {
        res = Region1_takeEvent(id);
    }
    return res;
}
void D_entDef() { D_enter(); }
void D_enter()
{
    Region1_subState = D;
    Region_active = D;
    DEnter():
}
.....
```

```
}
...
} // end of class Reactive and class Test
```

5 比较

Rhapsody [14]是一个 Case 工具。它可以为应用建立 UML 模型，并产生对应的 C、 C++或 Java 代码。R 中的代码产生是基于 Object Execution Framework(OXF)[14]。动态模型是用框架类定义的并被代码产生器写死(hard-coded)。

在 Rhapsody 的模型中，用类来实现事件并且通过执行 switch 语句完成转换搜索（transition-searching)。然而在我们的方法中，事件成为一个方法；类的继承结构通过对象的组成实现。使用多态概念自动执行状态搜索。我们比较了图 5 的两种方法产生的代码。表 1 显示了两者的对比。Rhapsody 的数字没有包括 OXF 框架附加的代码。

	Rhapsody*	我们的方法
源代码的行数（Source Code: No. of lines）	675	250
源代码的字节数（Source Code: No. of bytes）	24270	6420
类的数目（No. of classes）	7	11

表 1 对代码的紧凑性比较

1. 我们方法产生的代码更紧凑。如表 1 所示，不包括 OXF 的代码，Rhapsody 产生的源代码仍然将近 3 倍长于我们方法产生的代码。

2. Rhapsody产生的代码难于理解。它使用数据值定义状态，并且明确使用Reactive类的操作检查数据。状态的转移由对变量的赋值完成，且没有被明确的表示。它将状态选择代码放到Reactive类*takeEvent（short id）*方法的*switch*语句中。这使得代码难于理解。我们的代码将每一个事件转化为一个操作调用。根据多态原理，相应的方法被调用。转换的代码被放入相应类的不同方法中。因此所有的状态和转换都是明确的，不需要使用任何条件语句。这增加了代码的可读性。

3. 我们的代码易于维护。我们将与特定状态相关的所有行为放入一个对象中。因为所有与状态相关的代码处在单独的类中，所以可以通过定义新的类和操作添加新的状态和转换。在 Rhapsody 中，由一组状态图描述的系统实际行为被产生的代码和 OXF 框架掩盖。

4. 看上去我们的方法引入了太多的小类，这是因为不同状态的行为被分散到几个类之间。这增加了类的数量。然而这样的分布减少了大量的条件语句。大量的条件语句是不受欢迎的，因为它们使得代码难于理解、修改和扩展。

这些机制上的不同使得我们方法产生的代码紧凑、易于阅读和维护。

6 相关工作

最相关的工作是 Harel 和 Gery[13]，它们工具 Rhapsody 可以从 UML 模型产生 C＋＋和 java 代码。如前文所述，我们的代码比 Rhapsody 的更加可读和易于维护。

Kohler et al.[15]提出了一个从状态图转换代码的方法。他们的方法采用了基于状态表的通用数组，但却使用对象结构表达运行时的状态表。他们使用对象表达状态图中的状态，使用属性记录 entry 和 exit 动作。库函数*handleEvent()*用来解释状态表、响应事件和调用合适动作方法。状态表的设置比较复杂和费时。在我们的方法中状态作为对象，entry 和 exit 动作方法。我们方法产生的代码更加简单和有效。

Tomura et al.[16]提出了状态图设计模式，该模式定义了类和状态转换执行机制，该机制用于实现开放分布式控制系统的设备组件模型的动态行为。环境类是被状态图指定动态行为的类。该类只有一个 StateMachine 类实例。StateMachine 用来描述状态图，该类对象包含了状态和转换的两个集合。每个对象对应于状态图、顺序子状态或并发子状态。事件监护和动作也用类来实现。

Knapp 和 Merz[17]描述了名为 Hugo 的一组工具。一组通用 java 类为状态图提供了标准运行时组件状态。状态作为对象。*run* 方法用于设置和初始化相关状态图。Hugo 的代码本质上是解释性的并且没有提供代码优化。

Gurp 和 Bosch[18]提出了实现状态图的 FSM 框架。状态、转换和动作都用对象表示。与状态模式相似，环境组件有一个当前状态的引用。用一个状态对象而不是状态子类来表示当前状态。转换对象有到目的状态和动作状态的引用。对于简单转换，FSM 框架实现转换的代价比状态模式高两倍。通过在 hash 表对象中查找来实现转换搜索。Hash 表对象将事件映射到转换。

7 总结和进一步的工作

本文描述了从 UML 状态图产生 java 代码的一种实现方式。通过将状态表示为对象并用对象组合和委托扩展状态模式，从而实现了层次状态和并发状态。该方法成功的处理了大多数状态图概念，如子状态、历史节点和复合转换（fork/join）。状态图的一些其它概念，如监护和分支、时间和信号事件将在以后实现。该方法可用于从 UML 状态图到代码转换的基础。当前我们正在实现此方法。

我们的方法是一种面向对象的方法，因此它能用其它 OO 语言，如 C++等,产生低层次的代码。代码产生引擎将被裁减来适应目标语言，因为在不同的 OO 语言中一些特性是用不同的方法实现的。

参考文献

[1] Object Management Group, OMG Unified Modeling Language Specification Version 1.4, OMG, 2001.

[2] D. Harel, Statecharts: A visual formalism for complex systems, Science of Computer Programming, No.8, 1987,231-274.

[3] B.P. Douglass, Real Time UML - Developing efficient objects for embedded systems (Massachusetts: Addison-Wesley, 1998).

[4] A. S. Ran, Modeling states as classes, Proc.Technology of Object-Oriented Languages and Systems Conference, 1994.

[5] A. Sane, R. Campbell, Object-Oriented state machines: subclassing, composition, delegation, and genericity, ACM SIGPLAN Notices, OOPSLA'95, vol.30, Austin, Texas, USA, 1995, 17-32.

[6] K. O. Chow, W. Jia, V. C. P. Chan and J. Cao, Modelbased generation of Java code, Proc. International Conf. on Parallel and Distributed Processing Techniques and Applications (PDPTA 2000), Las Vegas, USA, 2000.

[7] E. Gamma, R. Helm, R. Johnson and J. Vlissides, Design patterns: elements of reusable object-oriented software (Massachusetts: Addison Wesley, 1995).

[8] S. M. Yacoub and H. H. Ammar, A pattern language of statecharts, Proc. Fifth Annual Conf. on the Pattern Languages of Program (PLoP ' 98), Monticello, IL, USA, 1998, TR #WUCS-98-29.

[9] M. Samek and P. Montgomery, State-oriented programming, Embedded Systems Programming, August 2000, 22-43.

[10] J. Ali and J. Tanaka, Converting statecharts into Java code, Proc. Fourth World Conf. on Integrated Design and Process Technology (IDPT'99), Dallas, Texas, USA, 2000 (CD-ROM).

[11] J. Ali and J. Tanaka, Implementing the dynamic behavior represented as multiple state diagrams and activity diagrams, Journal of Computer Science & Information Management (JCSIM), Vol. 2, No. 1, 2001, 24-34.

[12] J. Ali and J. Tanaka, An object oriented approach to generate executable code from OMT-based dynamic model, Journal of Integrated Design and Process Science, Vol. 2, No. 4, 1998, 65-77.

[13] D. Harel and E. Gery, Executable object modeling with statecharts, Computer, Vol. 30, No. 7, 1997, 31-42.

[14] Rhapsody case tool reference manual, I-Logix Inc. http://www.ilogix.com.

[15] H.J. K.hler, U. Nickel, J. Niere, and A. Zündorf, Integrating UML diagrams for production control systems,Proc. 22nd International Conf. on Software Engineering (ICSE 2000), Limerick, Ireland, 2000, 241-251.

[16] T. Tomura, S. Kanai, K. Uehiro and S. Yamamoto, Object-oriented design pattern approach for modeling and simulating open distributed control system, Proc. IEEE International Conf. on Robotics and Automation (ICRA 2001), Seoul, Korea, 2001, 211-216.

[17] A. Knapp and S. Merz, Model checking and code generation for UML state machines and collaborations, Proc. 5th Workshop on Tools for System Design and Verification, Reisenburg, Germany, 2002, 59-64.

[18] J. V. Gurp and J. Bosch, On the implementation of finite state machines, Proc. IASTED International Conf. on Software Engineering and Applications, (SEA' 99), Scottsdale, AZ, USA, 1999, 172-178.

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

机械工业出版社即将出版

Scott Ambler

Web 应用设计模式

Michael Weiss 著，wh89954 译

讨论

介绍

这篇文章讨论了对应用于“小型”web应用的模式语言的研究，它的目的是描述出开发web应用的原理性框架。为什么仅仅是“概念性的框架”？因为许多web应用可能不需要一个复杂的应用服务器框架。只需要用于特殊任务如模板处理的微型框架。主要的应用可以在一个标准的开放源码平台如LAMP（(Linux, Apache, MySQL, and Perl/PHP/Python）的顶端来开发[11]。

在一个全面框架的复杂性和你的应用的需求之间存在着平衡。框架通常具有很大的特征集，因而也有不合理的学习曲线，开发也比较困难。例如，当一些web应用可能需要一个内容管理系统时，大多数情况下，一个模板处理程序可能就可以满足要求。

在另一方面，一个概念上的构架可以为你提供有关如何建立一个定制的应用服务器模式，并且告诉你何时应用处理特别任务的微型框架。在本文中的概念性框架是作为一种模式语言进行论述的。这些模式按照非简单、小型到中型的大小被连接，我们的目的是论述包含有典型设计的通用实践。

在这些设计的论述中包含有应用程序与用户之间相互作用的流程，处理用户请求，在应用程序的调用之间保存应用程序的状态。另外涵盖的普遍问题是如何为不同类型的浏览器生成输出，与外部数据源(包括网站)相互作用，并且使得应用程序的互动性更好。

在另一方面，web 站点和导航设计是在模式语言之外的范畴。例如，在Welie[22]，van Duyne et al [20]，Lyardet and Rossi [17]，Rossi et al [18]，and Fernandez et al [12]的模式文档[22]中对这些问题也有所涉及。按照Welie的观点，这些模式能够被分为站点、经验、任务以及基本的互动模式。

站点模式的一个例子是商业站点，任务模式的一个例子是购买/销售模式，支持购买/销售任务的一个模式是购物车（SHOPPING CART）。管理一个购物车是另一个任务模式的一个特别实例：即收藏夹（FAVORITE），它同样也是基本互动模式LIST BUILDER的一个特殊案例。另一个基本模式是WIZARD，一步步把用户引向结账。

有关这些模式的全部背景是网站和导航设计已经全部完成。假设我们正在建设一个电子商务网站，并且已经选择使用 SHOPPING CART 和WIZARD 模式来构建基本的用户互动，我们该如何在程序级别处理这些设计？用这种方式来划分模式设计语言只是反映了在网站设计者与程序员之间的典型分界。

模式语言综述

在这一节中，通过一个带有注释的路线图对模式语言进行总结（见图1），箭头指示了模式之间是如何改善和理解的，在箭头上的注释总结了在模式的上下文中或者在先前使用的模式中选择应用一个特别模式的基本原理。这个基本原理是作为模式解决问题的一个主要因素给出的。

可维护性是许多模式的一个推进力量：TRANSITION TABLE (5), CENTRAL DISPATCHER (10), SEPARATE CONTENT FROM PRESENTATION (19), DATA SOURCE ADAPTER (33), CHAIN OF APPLICATIONS (33).其他的模式是由对安全性、隐私性的关注而推动的：ACCESS CONTROLLER (33), SESSION (24), UNIQUE ID (29).还有一些模式关注的是封装性，如： CONTINUATION (14), WRAPPER (33), EXPOSE APPLICATION AS SERVICE (33).最后，性能是由OFFLOAD WORK TO CLIENT (30).来确定的。

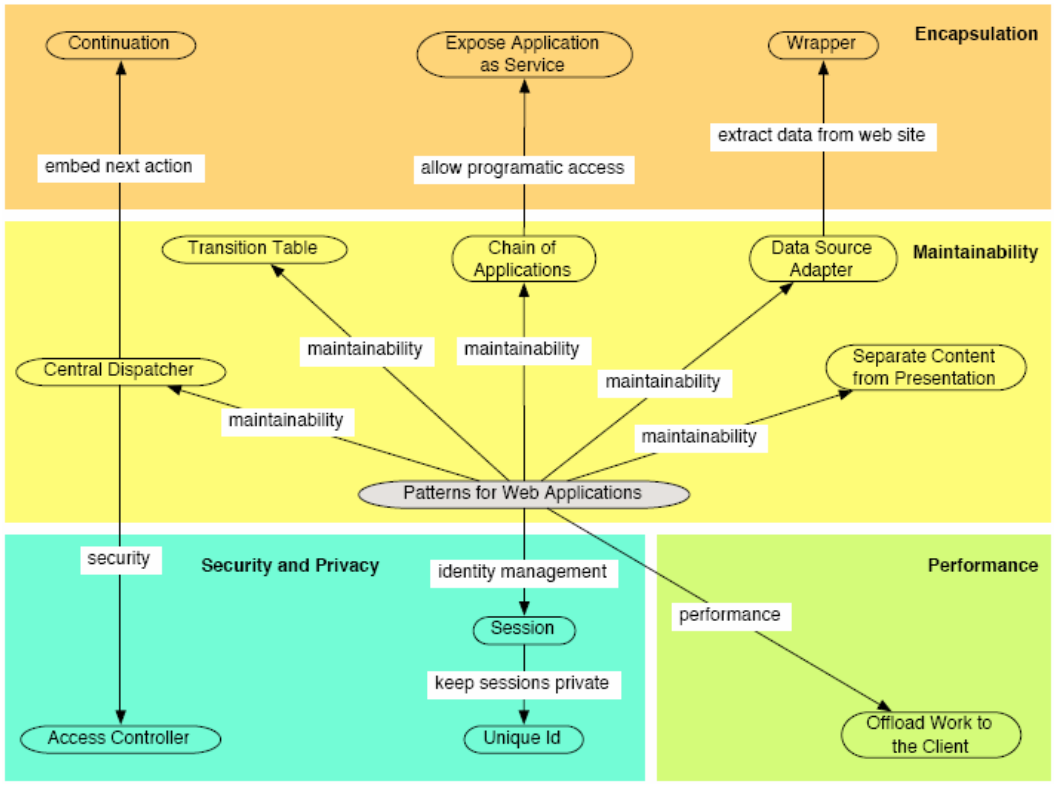


图1 模式路线图

这里是应用这些模式的建议顺序。你的网站应用设计是由通过使用TRANSITION TABLE (5) 定义互动流来开始的。互动流表达了表达并呈现给用户的页面形式以及用户可以执行的动作。你接着在CENTRAL DISPATCHER (10)中定义应用程序如何对用户的动作进行反应。如果你需要支持多个输出格式，可以使用SEPARATE CONTENT FROM PRESENTATION (19)。

多数的web应用将会需要跟踪用户的配置、会话信息—session（24），DATA SOURCE ADAPTER (33)描述了你的应用如何于外部数据源（数据库和web 站点）相互作用，高级的安全技术不在本文的论述范围之内，基本的web站点保护，连同用户隐私的保护可以通过 ACCESS CONTROLLER (33)，UNIQUE ID (29)获得。

甚至，你将发现将你的web 应用分离成可以组装成一个新应用的多个子应用是很方便的。CHAIN OF APPLICATIONS (33) 描述了这些方法，EXPOSE APPLICATION AS SERVICE (33)描述了一个使用web service 的特别实例。对于性能和响应性方面的原因，你可能也会需要 OFFLOAD WORK TO CLIENT (30)。

我们将会通过运行代码示例来说明这些模式，这个代码示例是一个在网上的在线书店中比较书目价格的应用。这个例子将会对每一个模式逐一介绍。我们也会摘录使用PREL执行的代码。但是我们将努力摒弃PERL语言中使得移植到其他环境中比较困难的特性。

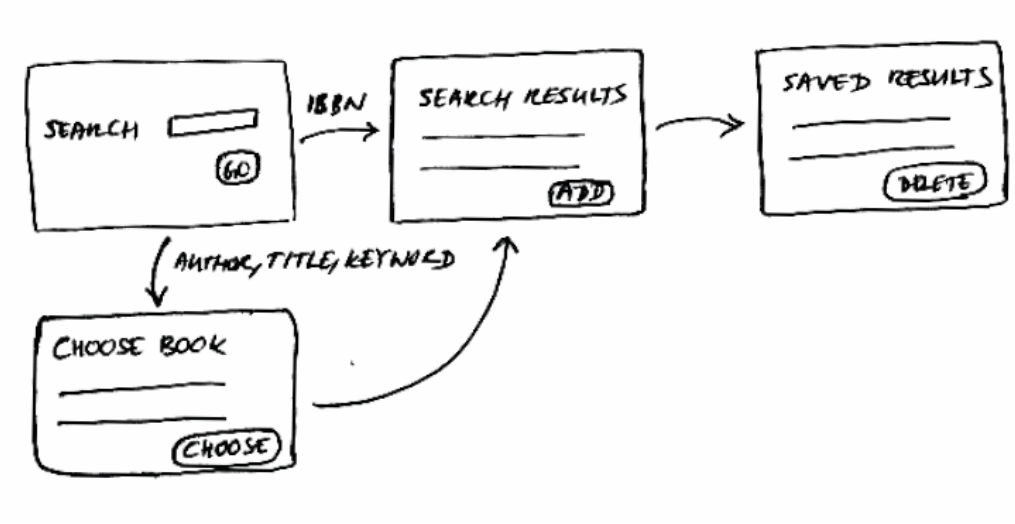
在对模式的论述格式中需要说明的最后一点，模式是用Alexandrian form [1] 进行描述的，也就是说，对于每一个模式，我们论述它的上下文、问题、问题的详细内容、解决方案以及结果等，问题和解决方案使用斜体进行强调，对于每一个模式，使用一个对话框对解决方案进行总结，对话框使用use case 图示符号来指明解决方案中的结构和行为。

因为模式语言是很长的，它将会以不同的部分出版，在本文中将会涉及以下的模式：TRANSITION TABLE (5), CENTRAL DISPATCHER (10), CONTINUATION(14), SEPARATE CONTENT FROM PRESENTATION (19), SESSION(24), UNIQUE ID (29), OFFLOAD WORK TO CLIENT (30)。

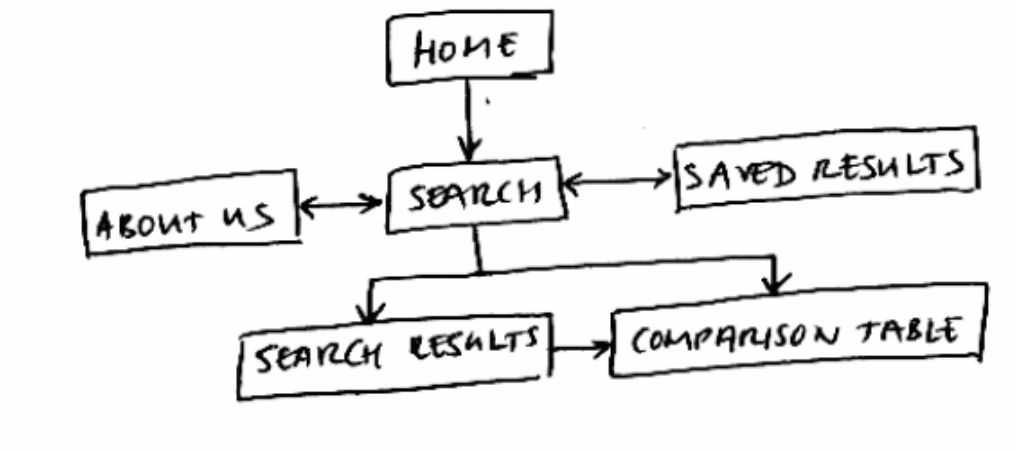
模式其余部分的个性特征将会在附录中描述。

转换表（Transition Table）

假设你正在设计一个web 应用，你的web 站点的导航设计已经通过脚本、站点图、以及单个页面的一些线框（wireframes）进行了描述，脚本就像一个与你交互的剧本作家一样，使你对应用程序是如何运行的有一个整体的感觉[23]。然而，它们只是用户与网站将会如何进行交互的一个粗略的草图。



站点图说明了你的网站是如何组织的，并且可能会指出例如动态生成的页面以及站点中受保护的访问区等其他重要的信息。然而，一个站点地图并不指明在页面之间的每一个单独链接。很显然，诸如快捷键等破坏了站点地图层次结构的链接环（hat）也被排除在外。



Wireframes说明了搜索框和特殊页面的表现等重要页面元素。Wireframes也可以用于指明在站点地图中未被详细说明的页面间的链接，例如，与相关页面以及返回的链接。在示例中，Wireframes显示了一个价格比较网站中搜索页面的详细情况。

需要注意的是，你必须指定详细的逻辑导向，而不能在众多的链接中迷路，也不能遗忘重要的导航路径。

一个平稳的交互流使得一个良好的用户使用体验得以建立，在你设计你的逻辑应用的详细细节之前，你需要去规划你的应用程序页面应当以一个什么样的顺序排列。

一个站点地图说明你的网站的组织方式，它只是显示出在页面之间存在的关键联系。一个完整的站点地图可能不会达到说明整个网站中*典型*导航路径的目的。在向你的web应用转换时，你需要明确指出你的导航逻辑。

例如，在设计对不同在线商店中的书目价格进行比较的应用时，用户可能需要通过作者、题目、关键字、或ISBN号来进行检索。检索结果是由每一个商店中的价格、是否有货构成的表格。用户可以将在该网站上不同的检索结果进行保存。关键的功能（如about us, search, saved results）在每一个页面中都应该具备。站点的导航设计产生了上述的站点地图。

这样的站点地图还不是很精确。它没有包含两个重要的转变：

一个是为了增加一个搜索结果从比较结果表到已保存结果表的转变，另一个是为了删除一个搜索结果已保存结果表本身的转变。要注意到，这些情况不会被遗忘，但是可能不会被考虑到，因为它们打破了继承关系。诸如这样的链接应该在类似前面例子中的wireframe中被说明，但它们通常不会在站点地图中得到显示。

在前述的站点澄清模糊的问题对于避免在导航逻辑执行中的矛盾是非常重要的。设计决策的细节应该不遗留在处理用户请求的代码中，这是因为如果一个站点图没有被完全说明，它的相同部分可能被不同的开发者作不同的解释。

如果没有对导航逻辑的详细细节做说明，论述如何从站点地图得到执行导航逻辑的唯一文件将会是在代码中，任何一个执行扩展操作的用户将不得不在自己在代码中寻找导航设计决策，这将会是一个潜在的时间开销大、有错误倾向的过程。在另一方面，如果所作的说明没有与实际执行过程同步，结构良好的代码将比说明更加精确。

这个模式建议按照一系列转换规则来定义可接受的页面继承关系。每一个页面响应一个用户action，在每一个转换规则的左部是用户action的名称，右部包含了用户下一步可以进行合法action一系列的action集。

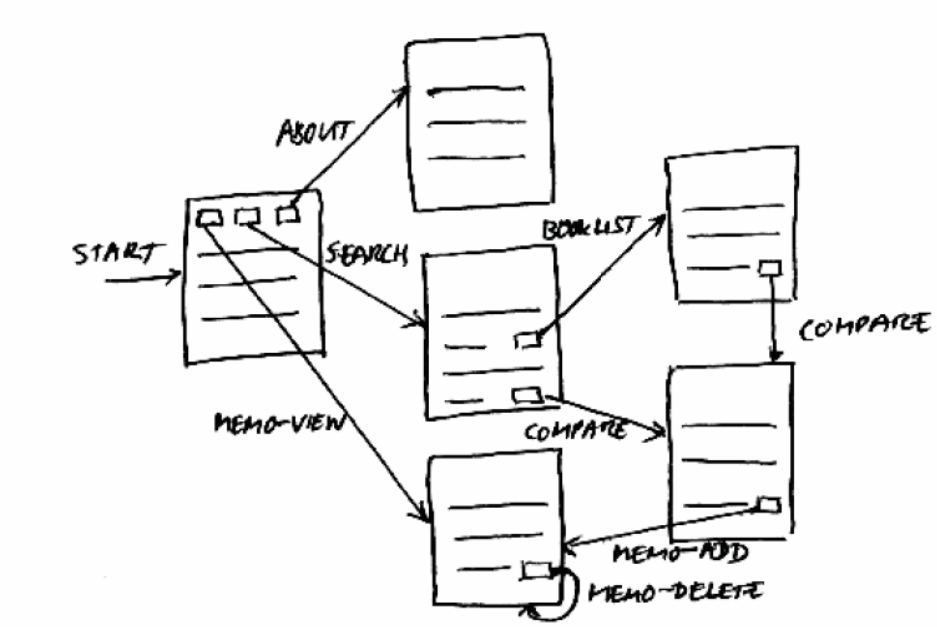
```
<action> => <next action 1> | <next action 2> | ...
```

例如，下面的转换规则提供了这个价格比较站点的一个详细导航设计。从start 页面开始，用户可以执行以下的action：about,search,以及memo-view,about action产生该应用程序将要被显示的信息，search action将会产生一系列匹配的书目（booklist）和结果比较表（compare）,用户可以在其中通过除ISBN号以外的条件对匹配书目进行查询和选择。memo-add 和 memo-delete action可以更新所保存的结果。

```
start    => menu
search   => booklist? | compare? | menu
booklist => compare? | menu
compare  => memo-add! | menu
memo     => memo-delete! | menu
about    => menu

menu == about? | search? | memo-view?
```

我们同样要指明一个 action是否会产生边界影响。没有边界影响的 action以？来标识，会产生更改影响的action用！来标识。因此有：

在页面之间定义合法的转换规则。每一个页面响应用户的一个 action，一个转换规则定义了下一步可以执行什么样的 action。



在前面所定义的转换规则保证了对信息框架开放的导航逻辑细节不会直到后来的设计中才被解释，这有助于保证导航逻辑被持续地执行。这种解决方案的另外一个好处是所定义的转换规则能够被用于导出web应用的测试用例。

转换规则为维护和扩展代码提供了参考，因为导航规则不再只是在代码中说明。然而，这就要求对导航逻辑所作的修改必须在说明中得到反映。记录这些问题的一个好方法是使用转换规则来进行测试，就像建议的那样。

同样需要注意到，应用程序同样需要执行合适的错误处理程序来处理规则之外的请求。请求可以以任何序列被接受，例如，如果请求是从书签而不是文本中提交的，可以使用浏览器的回退按钮或者使用缓存来处理。

对用户action的处理应该以CENTRAL DISPATCHER (10) 的逻辑被编码。下一步的action应当使用CONTINUATION (14)进行压缩。这种模式产生于Tennis Shoppe 案例的研究[14]，有关它的设计的讨论在session 管理中被讨论。

Central Dispatcher（中央调度器）

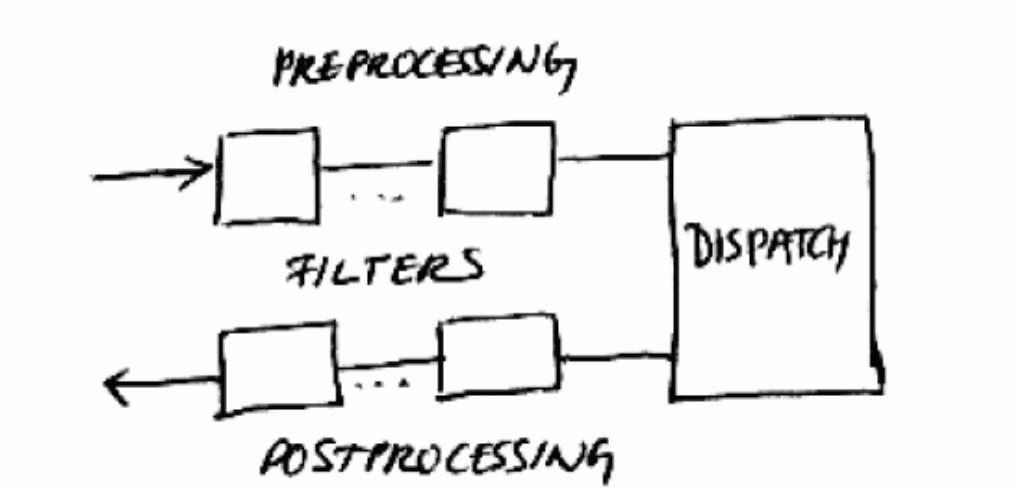
你已经使用TRANSITION TABLE 定义了导航逻辑的细节，你现在需要定义应用程序如何处理用户的请求。一些页面可能也需要特别的预处理和后处理（例如登陆、授权、和输出变化）。

应用程序非结构化的演变，例如加入处理新用户action的代码，经常会导致应用程序的逻辑非常难以理解和扩展。特别地，它对于保证预处理和后处理的一致性表现是一个挑战。

一个处理用户请求的方法是为每一种类型的请求编写单独的脚本，依靠web server 根据脚本的名称向适当的脚本发送请求。因为这种方法易于执行，它会导致对普通的功能（如登陆、授权和输出变化）产生重复的代码以及导航逻辑执行的不一致性。原因是由于脚本同时执行了导航和表现逻辑。

另外一个观点是集中处理请求，并将导航逻辑从表现逻辑中分离出来。在这种方法中，请求被一个单独脚本处理，它将请求发送到合适的请求处理器。请求处理器能够与以发送器同样的脚本执行，也可以使用单独的脚本执行，也就是被发送器通过重定向从web server 调用的那些东西。相同的脚本在执行中的差异在于将处理请求的代码分离到命令目标中（这些命令目标可以作为PERL中单独的包或文件被执行）。

在集中发送中，“进入”脚本起到像中央调度器一样的功能。除了正确地发送请求，它还是将预处理和后处理进行统一的地方。鉴别是预处理过程的一个例子。在这里，一个中央调度器能够超越一个web server 所提供的简单功能。例如，web server 提供了一种用于鉴别的简单表格来限制对你的网站中特定区域的访问，然而，用户界面却非常粗糙（一个弹出对话框），并且登陆信息和口令都是用明文提交的。在这里，除了一个单一的客户端session 外，没有别的方法来记住用户的口令。



一个基本的思想是测试用户请求的action类型，并对应用程序的适当部分进行分支。然而，如果处理代码与发送代码相互混合，就会导致回旋的应用逻辑。这就需要把确定所需要处理的action的应用程序部分和对请求的处理部分明确分离，这种解决方式是将发送模块从处理请求的代码结构中分离出来而实现的。

例如，假设用户的操作请求是通过form中的action 参数传递的，下面所摘录的这段代码将把action发送到价格比较网站中正确的请求处理器。注意，这个例子同样说明了预处理过程（登陆）的一个简单案例。

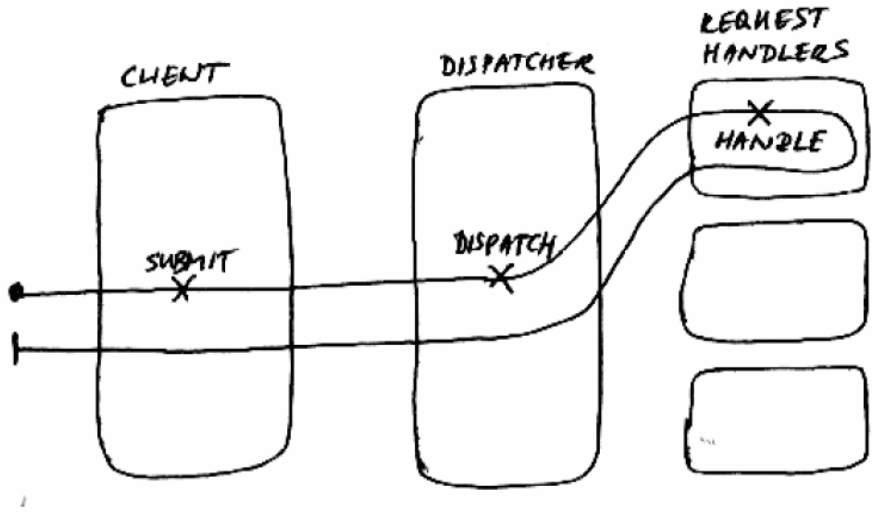
```
my $q = new CGI;
my $action = $q->param("action") || "start";

# pre-processing
log($q);

# dispatching block
if ($action eq "start") {
  start($q, ...);
} elsif ($action eq "about") {
  about($q, ...);
```

```
} elsif ...
  .
  .
} else {
  error("don't understand");
}
```

所有的请求处理器都向调用的文本传递一个引用 $q。当我们将这种模式与其他模式合并时，其他的数据（例如一个session 的 id）可能也需要向处理器传递。调用 log() 只仅仅是集中预处理和后处理的一个简单例子。更多复杂的例子分别在ACCESS CONTROLLER、SEPARATE CONTENT FROM PRESENTATION (19)中讨论，

因而有：

集中用户请求的处理过程，在一个单一的脚本中处理所有的用户请求，这个脚本会将请求发送到适当的处理器。为预处理和后处理过程（如登陆、鉴别和输出变化）在这个中央调度器中定位代码。



使用一个中央调度器比为每一个用户action使用单独的脚本更加复杂。虽然单独的脚本方法对开发者来说更加熟悉，但是使用一个中央调度器可以得到切实的好处。集中的请求处理使得具有比较复杂的导航逻辑的应用程序易于管理。它在表现逻辑中明显分离了导航逻辑。中央调度器接受请求，并把请求发送到适当的请求处理器。

在中央调度器中对预处理和后处理定位代码可以保证在你的应用程序不同部分的通常行为具有一致性。例如，当与ACCESS CONTROLLER(33) 联合使用时，这个模式加强了在一个单独节点上对网站中的限制访问区域的所有访问请求进行的资格检查。

这个模式通常与CONTINUATION(14)和 SEPARATE CONTENT FROM PRESENTATION (19) 一起使用请求处理器使用CONTINUATION (14)创建它的回复页面，并且嵌入需要的文本信息对下一步能够进行的每一个action进行处理。它同时也负责页面的生成。实际上，如果需要支持多种类型的浏览器，SEPARATE CONTENT FROM PRESENTATION (19)将帮助按照需要的输出格式生成输出页面。

这个模式在web应用的设计中得到了广泛的使用。有关它的使用的具体例子在[14]的Tennis Shoppe case study和原始Wiki的设计[9]中有描述。FRONT CONTROLLER (344)模式也有同样的作用，一个单独的对象处理所有的用户请求，并把它们委派到适当的处理器。有关登陆、鉴别和输出变化等的普通代码能够通过筛选加入进来。

Continuation（延续）

使用CENTRAL DISPATCHER (10)的建议，使用请求处理器来生成响应用户action的下一个页面。

处理用户请求的文本信息（一个action，加上任何的请求参数）需要从一个页面传递到下一个页面。

当在CENTRAL DISPATCHER (10)中的一个请求处理器生成了一个页面来响应用户的action，这个页面就需要包含执行下一步(或一系列)action时这个action的TRANSITION TABLE (5)所指明需要的所有信息。这样做的原因是由于位于在WEB 客户端和应用程序之间通讯底层的HTTP协议是无状态的（STATELESS)。它不能保存在请求之间的应用程序文本，这是需要你自己做的事情。

在设计对这个问题的解决方案时，你需要考虑大量的约定。一方面你希望这个解决方案是简单的。通常你能够假设在一个短暂的事务处理期间（例如登记新用户）你的应用程序页面将会被连续访问，也就是说，它是和已经设计好的导航流同步的。这样的一种解决方案不需要使用可以被用户禁用的 COOKIES，它可以工作在所有浏览器的配置下。你同样需要对文本进行压缩，以便为一个新 action 向文本中嵌入增加的代码时不会影响已存在 action 的代码。

在另一方面，接收一系列连续页面请求时的依赖限制了对CONTINUATION (14)的使用来缩短交互的频度，这种依赖对导航流中的任何中断是脆弱的，例如，由用户点击“回退”按钮导致的中断。以无阻碍格式传递的文本信息能够在客户端阅读，也能够被操纵。（允许用户读数据潜在地与隐私相关，所以允许它们来操纵数据增加了与隐私的关系）。

这个模式建议为每一个下一步action编码，action的参数包含在FORM的参数中。FORM参数通常与<form>相关，但是并不是必须这样。例如，它们可以嵌入到URL中。在FORM中，它们可以被存储在隐藏域，或嵌入在FROM的ACTION属性中。

建议有三种方法类压缩文本信息，每一个都有优点和缺点。

首先，文本能够在 url 的参数中被编码。特别地，固定参数（如所用的语言）通常是用这种方式嵌入到页面的。下面 url 将 about action 进行了编码：

```
<a href="bookfinder.cgi?action=about">About Us</a>
```

这是一种在页面中编码静态文本的建议方法。(至少在你建立一个 html 页面时，你可以使用 wml 页面来在 url 中嵌入动态信息，它同样可以包含变量)，然而，你应当避免用这种方式建立复杂的 url，因为它们难以阅读。同样，这种方式对查询 action 是受限的，其次，如果可能采取多于一种的 action，每一个 action 的文本可以在同一个页面的不同 form 中被编码，下面的 form 中包含了处理一个 memo-add action 的请求，这个 action 被嵌入到 form 的 action 属性中，action 的参数（isbn,bookstore,等）被存储在隐藏域中。

```
<form action="bookfinder.cgi?action=memo-add" method="POST">
  <input type="hidden" name="isbn" value="0802117244">
  <input type="hidden" name="bookstore" value="Chapters.ca">
  <input type="hidden" name="price" value="39.95">
  <input type="hidden" name="availability" value="24 hours">
  <input type="submit" value="Save">
</form>
```

使用隐藏域比将所有的 form 参数嵌入到 url 中具有更好的可读性，然而一些文本信息的片断（例如，action 的名称）仍然可以在 form 的 action 属性中被编码到 url 中。这将会明确指出哪一个文本参数具有静态值，并且哪些值是被应用程序动态生成的。既然“隐藏”域当然对于客户端是完全可见的，具有敏感信息的域应当被加密以保证它们的文本信息不被阅读和操作。

第三，当使用一个单独的 form 时，不同的 action 和它们的关联数据能够通过将输入域名字赋予一个唯一的标识符来识别，下面的进行 memo—delete action 的 form 说明了这个操作过程。在 saved result 页面中的每一个搜索结果都被给予了一个唯一的名字，它是以相应的提交按钮的名字来反映的。

```
<form action="bookfinder.cgi" method="POST">
  <table border="0">
    <tr> ...
      <td><input type="submit" name="memo-delete.0802117244"/></td>
    </tr>
    <tr> ...
      <td><input type="submit" name="memo-delete.0393041530"/></td>
    </tr>
  </table>
</form>
```

当单独 action 可以做一系列事情时，通常使用这种方法（在这个例子中，是 memo-delete）。因为 form 的导言（action 和方法）不会被多次重复，它将会导致更加紧凑的页面表现形式。在另一方面，对隐藏输入域（包括提交按钮）使用唯一的标识符进行修饰比在多个 FORM 中使用一个参数来指明每个 FORM 所用的项目要复杂得多。如果你想要保证项目的标识不被操作，它也要按照前面建议的方法进行加密。

例如，在响应 COMPARE action 时，价格比较站点从在线书店中提取出指定书目的价格和存货信息。从这开始，它生成了一个价格比较表，它可以表示为下面的网状关联数组。

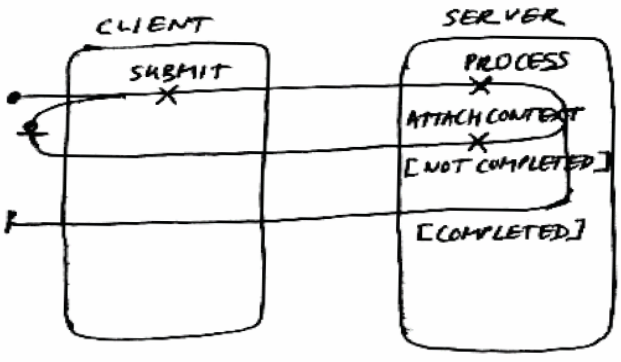
```
my $searchResult = {
  title => "Lucky Pierre",
  authors => [
    "Robert Coover"
  ],
  isbn => "0802117244",
  quotes => [
    {
      bookstore => "Amazon.ca",
      price => "31.96",
      availability => "2-4 weeks"
    },
    {
      bookstore => "Chapters.ca",
      price => "39.95",
      availability => "24 hours"
    }
  ]
};
```

这段代码期望显示请求处理器如何为每一个可以调用 memo-add action 的文本生成一个单独的 form：

```
sub compare {
  ...
  foreach $quote (@{$searchResult->{quotes}}) {
  print <<END_OF_HTML;
<form method="POST" action="bookfinder.cgi?action=memo-add">
  <input type="hidden" name="isbn" value="$searchResult->{isbn}">
  <input type="hidden" name="bookstore" value="$quote->{bookstore}">
  <input type="hidden" name="price" value="$quote->{price}">
  <input type="hidden" name="availability" value="$quote->{availability}">
  <input type="submit" value="Save">
</form>
END_OF_HTML
  }
  ...
}
```

因而有：

在处理一个用户请求时，可以通过在 form 中将每一个下一步将要进行的 action 和它的参数在 form 参数中进行编码，它们可以存储在 url、隐藏域或提交按钮中。



由于压缩文本的方式，将一个 action 的整个文本在返回页面中传递的解决方案比理解在 server 端管理的 session 信息更为容易。在另一个方面，这样的一个解决方案可能会违反有关安全性和隐私的条文，因为文本信息会被以明文的方式传递，并且应用程序不能够保证所传送的信息没有被修改过。

对于长期的转换，只依靠CONTINUATION (14)也会使一个应用在处理流程中的中断时是脆弱的。这时，使用SESSION (24)在server 端存储文本信息。只有session 的id 需要使用 CONTINUATION (14)来编码，或者，选用在client端的cookies。通常这两种对session编码的form是同时使用的，因为我们不能总是假定client端可以使用cookies。

这种方法已经在[14]的Tennis Shoppe案例中得到了使用。ASYNCHRONOUS COMPLETION TOKEN (65)模式在目的上是与此类似的。

一个异步完成的令牌随着一个client 的请求被发出，指明了来自于server的响应如何被处理。同样的，一个附加部分提供了处理下一个用户action的文本信息，但是只是作为server端的提示，而不是针对client端。

从表现分离内容

在 CENTRAL DISPATCHER (10)中的请求处理器负责生成返回页面。这个任务更为困难的是你需要支持具有极大表现差别的多种web client类型（例如，高端LCD显示器和网络电视的不同屏幕解决方案），或者适应独立于你的网站功能的表现形式的频繁变化。

如果你在应用程序逻辑中写下应用的输出结果，那么在改变它的表现形式或为支持新的web client 端而加入重复性代码的时候，你就会面临对应用程序的大部分进行重写的问题。你需要减少为适应目标输出格式所需的努力。

将用户界面从你的应用中分离出来的做法已经被通常的应用程序明智地采纳，但是当遇到需要写下web 应用时，我们似乎还要从头学习。这样做有很多理由。理由之一是web应用的初始版本通常是为一个单独平台的特殊浏览器而编写的（例如在windows 2000平台上的IE5）。支持其他类型web client 端是出于事后考虑，并且通常由于为每一个输出格式建立的重复的应用程序代码会导致代码的泛滥。

另外的原因是在web 设计者和应用程序员之间角色划分的模糊性。从功能性上独立地网站的表现形式应该是可能的。然而，在实际上，应用和表现逻辑经常纠缠在一起，对表现方式的改变可能会影响到应用程序逻辑中的很多部分。

可以使用不同的方式在应用程序和表现逻辑之间进行分离，最简单的方法可能是使用Cascading Stylesheets (CSS)。某些表现定义如在特定html标识中使用的字体、文字颜色、或者页面空白的设置可以通过定义样式表规则从页面内容中分离出来。这样，页面内容现在就由与格式无关的html子集标识出来。

然而，样式表规则在定义你的内容应该如何进行格式化方面是有限度的，并且不能使用它来对某些标准进行识别和过滤（例如只显示低于某一个价位的报价，并将它们以升序方式存储起来），或者将它映射到其他类型的报价。

下面的两种方法可以解决在使用CSS中的这些限制。在模板和XSLY样式表中都包含了对一个web页面内容的中间表现的定义，从这个定义中可以产生期望的输出格式。模板是按照所需的输出格式书写的，其中包含有占位符，来自于应用程序的值能够通过模板处理器插入进来。对模板的改变能够不影响到将应用程序数据组装到模板的代码。

然而，模板仍然与应用程序代码有轻微的联系，即脚本语言。在应用程序和模板处理器之间传送值通常只限于键值组合。交叉数据结构只能用比脚本语言更为特殊的方式来表达。例如，HTML::Template支持hash 列表、hash列表数组等。

一个更为通用的方法是由应用程序逻辑依照XML来产生自己的结果，并使用XSLT样式表来将数据转换到期望的输出格式。XSLT样式表在server 端和client 端都能够被处理。更倾向于使用在server 端的样式表，因为它们允许更有力的转化而不会将隐藏的代码暴露给客户端。当使用在client端的样式表时，数据是发送到客户端的。

通过将文本内容封装成XML，我们使用XML作为表现形式和应用逻辑之间的一种接口。在涨价时，XML表现保持了内容的含义。标签用于将内容组织起来反映潜在的商业对象和属性、数据库的表和列。表现逻辑现在存在于样式表中，并能够被改变而无需触及应用逻辑。

然而，因为XSLT样式表提供了一个将内容映射到目标的输出格式更为通用的方法，处理样式表比使用模板更加关注于时间开销。然而，因为XSLT的处理技术变得更加高效，XSLT处理器可能很快在性能上与模板处理器具有可比性[10]。在另一方面，内容的一个XML表现具有另外的优点，即它独立于脚本语言，并且可以与其他的应用程序进行交互---参见CHAIN OF APPLICATIONS (33).

例如：价格比较表的内部表现以讨论的方式对CONTINUATION (14)模式能够被以直接的方式映射到XML作了介绍。交叉关系数组的继承结构被一一对应地转换为XML。

```
<searchResult>
  <title>The Adventure of Lucky Pierre</title>
  <authors>
    <author>Robert Coover</author>
  </authors>
  <isbn>0802117244</isbn>
  <quotes>
    <quote>
      <bookstore>Amazon.ca</bookstore>
      <price>31.96</price>
      <availability>2-4 weeks</availability>
    </quote>
    <quote>
      <bookstore>Chapters.ca</bookstore>
      <price>39.95</price>
      <availability>24 hours</availability>
    </quote>
  </quotes>
</searchResult>
```

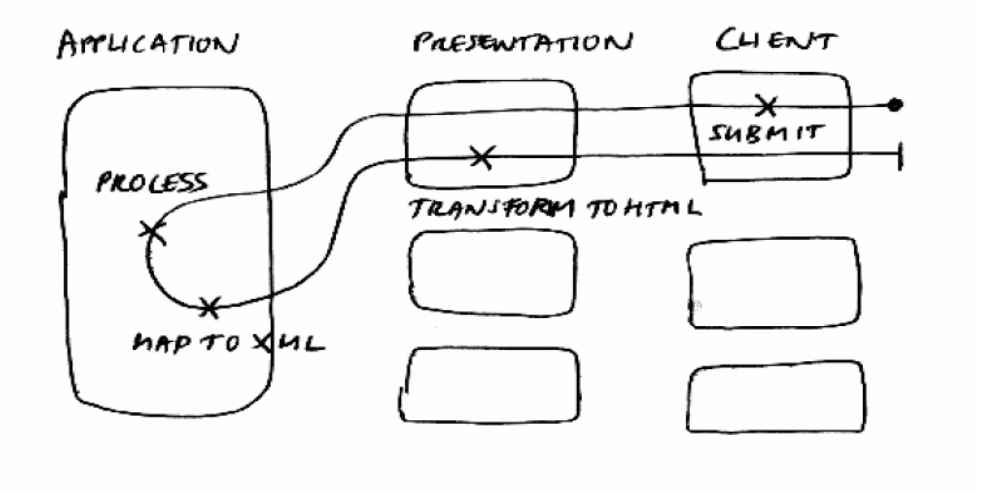
一个XSLT样式表是由与输入文档的不同部分相匹配的模板组成的。例如，下面的模板定义了对特定商店中代表了报价和存货情况的

报价要以表格中的列来显示。这种样式表生成了同样的输出，这是有关CONTINUATION (14)的讨论的核心。

```
<xsl:template match="quote">
  <tr>
    <td><xsl:value-of select="store"/></td>


    <td>$<xsl:value-of select="price"/></td>
    <td><xsl:value-of select="availability"/></td>
    <td>
      <form action="bookfinder.cgi?action=memo-add" method="POST">
        <input type="hidden" name="isbn" value="{../../isbn}"/>
        <input type="hidden" name="bookstore" value="{store}"/>
        <input type="hidden" name="price" value="{price}"/>
        <input type="hidden" name="availability" value="{availability}"/>
        <input type="submit" value="Save"/>
      </form>
    </td>
  </tr>
</xsl:template>
```

因此有：

把中间内容表现定义为你的应用中表现和应用逻辑之间的一个接口。使用CSS、模板、或者XSLT来把内容转换为目标的输出格式。



这种模式建议采用三种方式来分离应用和表现逻辑，每一个都具有自身的优点和缺点。总的来说，CSS支持对内容和格式的分离，但是没有对内容的结构进行改变。特别的，使用CSS不可能把内容从一个标记格式映射为其他（如从XML到WML）。模板和XSLT支持任何目标输出格式的生成。

内容的XML表现能够被映射到浏览器所需的、甚至在结构上存在差异的格式，如用于移动浏览的WML、用于声音浏览的VoiceXML。尽管这些标记语言的映射细节是不同的，基本方法与生成HTML输出是相同的。主要的区别是移动和声音浏览器每次需要的是其他的对话模型，而不是桌面的web浏览器。

定义一个中间表示增加了一些初始负担。选择一个模板处理器使用的内部数据结构使得解决方案有别于脚本语言和模板处理器。另一方面，一个基于XML的表示是独立于脚本语言的，并能够被CSS和XSLT处理。

使用一个中间表示同样增加了一些性能开销，它在为新的web客户端增加支持时，它和灵活性之间会进行权衡。在模板和XSLT之间，模板被认为处理速度更快。然而，XSLT处理器更有效的执行已经接近了与模板处理器之间的性能差距。

有关这种模式的使用在文档[7]中描述为一个体系结构入口的基础，在[5]中描述为一个支持在web应用中多种设备的策略。它在降低

应用程序开发和web设计角色方面耦合的应用在[21]中作了扩展探索。内容管理框架诸如Cocoon[2]和AxKit[3]直接支持模式。

Session

你需要在用户的请求之间保存web应用的状态。然而，就像建议的那样在页面之间使用CONTINUATION (14) 传送文本信息并不总是适当的，用户可能已经使用回退按钮到达了一个页面，或者一个用书签标识的页面，这样他们的下一步action就不再与设计的导航流同步。

在为每一个请求在server 和 client 端之间传送内容信息引发了许多的讨论。当设计的导航流被打断时，文本会被丢失。客户端阅读文本数据的能力同样引起大量的隐私和安全性的风险。

通过在server 和 client 端之间传送文本信息来保存应用程序状态的设计在发生导航流中断的问题时并不是很健壮的。这同用户在使用你的网站时转向其他的站点、结束他们的浏览器session 、或者由于使用回退按钮导致的结果是同样简单的。在这种情况下，处理用户下一步action所需要的文本信息被丢失了。

作为一个例子，当用户在线定货时，他们通常不会立刻去结帐，他们希望购物车的内容能够被储存一段时间，甚至于他们可以临时中断他们的购物过程去执行其他action。（Amazon.com 在广告上说他们可以把购物信息保留90天）。

你可能希望去收集持续地多次使用同样一个应用的客户信息。这可以是你通过客户与应用程序的交互暗中收集的（如一个历史或最近访问的产品），或者是客户在定制应用程序时输入的数据（如所要监视的一系列库存符号）。

正像这些例子所显示的，应用程序的状态信息就是我们希望以不同长度的时间间隔来保存的东西。一些信息只需要短期保存，例如在登记表中输入的用户名。这些信息不必持续在整个浏览器session期间，它只需在连续的form之间传送。这可以通过使用CONTINUATION (14)获得。

一些信息我们希望保存中等时间跨度（例如购物车的内容），另一些需要保留很长时间跨度（如用户的配置文件）。中等时间跨度与单独的、可能被中断的事务（只要用户结束了结帐阶段，购物车的内容是不再相关的）是一致的。而长期的暗示了信息通过与应用程序的重复交互被聚集。

持续跨越浏览器session的中期或者长期的应用程序状态只能使用客户端的cookies来存储。如果状态信息是小型的（如用户最近一次访问应用程序的时间），它能够被直接存储在cookie中。然而，在通常情况下，只有一个唯一地与用户的session 相连的标识符（session id）能够被存储在cookie中，实际的应用程序状态（session 信息）应该被存储在server 上，或者是在数据库或文件中。

存储文本信息引发了大量的平衡问题。在每一个请求和回复中传输完整的文本简化了设计，但是导致了隐私和安全性的问题。敏感的信息如口令、信用卡号码不应该在文本中传送，因为文本对用户是可见的。非授权的用户同样可以通过浏览器的缓存获得对信息的访问。最后，一些监视网络的用户可能会捕获HTTP请求和回复，并分析出需要的信息。

你同样需要关注用户对文本的操作。一个非常有名的例子是一个用户通过在付款后在购物车上添加了很多商品并提交了伪造请求。同样的，对导航流的中断导致文本信息的丢失。虽然在某些情况下不必在意（你不必记住用户上次在某个新站点读了哪些书），但是在其他一些场合就很重要（你必须记住在浏览器session期间用户购物车的内容）。

这个模式建议将文本分离为一个可以与客户端共享的session id 和 需要在server上保留的session 数据。有关session的信息被存储在数据库中，是用session id 作为数据库的键值。当用户第一次访问应用程序时，session id 没有和请求一起发送，接下来server 创建一个新的session id 和数据库中一个相应的记录。在后续的请求中，客户端包含了session id ，server 从数据库中重新得到session的信息。

存储session id 以及将它发送到server 的方法有很多种。一个是在URL中嵌入session id 。这种方法需要session id 被存储在内容或回复的所有URL中。尽管有一些web server 已经提供了重写机制来自动处理这项任务，但是它们使用起来比较复杂。这种方法另外的局限性在于它只能处理一系列连续的页面。使用隐藏域来存储session id 具有同样的缺点。

另一种方法是在cookie中存储session id 。Cookies 是由浏览器自动发送到设置它们的服务器上，这样它们就是我们能够执行持续在整个浏览器session期间的文本唯一的办法。然而，cookies 也有它们自身的缺点，用户可以因为隐私的原因选择禁用cookies ，并且cookies也会偶然被删除。为了使得应用程序尽可能地健壮，你需要选择通过在form 参数中把session id 进行编码，以及在cookie 中存储session id 来合并两种解决方案。

在两种解决方案中，session id 必须被加密以防止它被操纵。通过操纵的方法，一个用户可以通过猜测session id 来“hijack”另外用户的session。这包括计算session id 的一个防止修改的hash列表——参见UNIQUE ID (29) 中的更多细节。

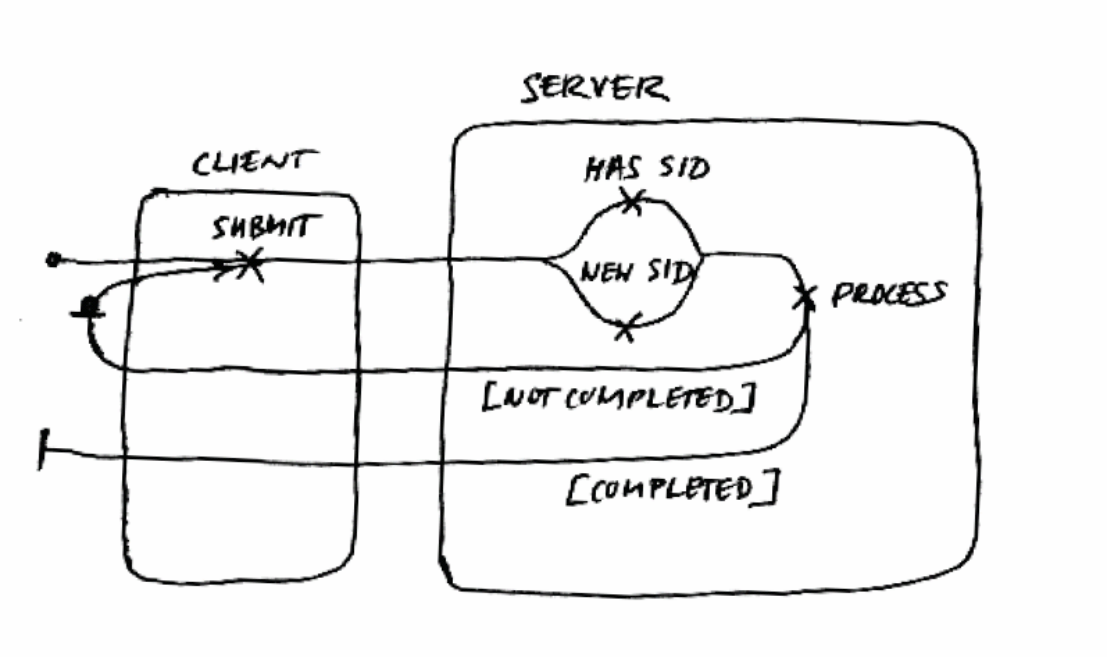
另外的一个考虑是当一个session 已经静止了很长时间后，它应当被结束，并且session id 也应该变成非法的。这就避免了当一个连接过早关闭时一个session 被无限期地被打开。如果这个站点是受控访问的，这同样保证了应用程序仍然被授权用户使用，而不是别的人。

例如，价格比较站点允许用户保存多个搜索的比较结果，这种信息需要在整个浏览器session期间持续有效。因而我们决定使用session,并把已经保存的结果存储在session 的信息中。在下面的代码中，函数getSid()检查cookie的名字sid，并在需要的时候生成一个新的session id 。我们接下来向client 端返回一个包含了已经更新的从当前时间算起30天为止的过期时间。

```
my $q = new CGI;
my $action = $q->param("action") || "start";

my $sid = getSid($q);
my $cookie = $q->cookie(-name => "sid", -value => $sid,
  -expires => "+30d");
print $q->header(-cookie => $cookie, ...);

if ($action eq "start") {
  start($q, $sid, ...);
} elsif ($action eq "about") {
  about($q, $sid, ...);
} elsif ...
  .
  .
} else {
  error("don't understand");
}

sub getSid {
  my $q = shift;
  my $sid = $q->cookie("sid");
  unless ($sid) { $sid = newSid(); }
  return $sid;
}
```

因而有：

为了维持中期或长期的文本，或者如果你的网站用户使用session 在客户端限制访问，可以把session id 存储在客户端，把相关的session 信息存储在服务器端。session的信息包含在数据库中，使用session id 作为数据库的键值。



把session 信息存储在服务器上而不是嵌入到web页面中具有很多的好处。一个是这种解决方案对于导航流的中断更加健壮。然而，无序的请求仍然需要被适当处理。使用 DIRECTED SESSION [16] 你可以对用户以一个特定的顺序访问你的网站页面有所加强。

同样，将session 的信息保存在服务器上保证了隐私和敏感的信息。然而由客户端提供的输入仍然需要验证，否则输入的信息中需要嵌入脚本以终止欺骗其他应用程序用户释放出潜在敏感的信息（跨网站攻击）。用于从一个不信任的客户端校验输入信息的一个模式是CLIENT INPUT FILTER [16]。

对session id 的加密在UNIQUE ID (29)中作讨论。 使用session 来控制对web站点的访问在ACCESS CONTROLLER (33) 中作深入讨论。 这种模式的使用在TennisShoppe case study [14]和Karkkhainen’s tutorial [15]已经作了讨论，讨论HTTP的session 处理的相关模式是AUTHENTICATED SESSION [16] 和FRONT DOOR (17) [19] 。

Unique Id

当你正在使用session(24)来在用户的请求之间维持你的web应用的状态时，你需要对session id进行加密以保证它们不被操纵，阻止用户对其他的session 进行“hjjacking”。

如果session id 是在未经加密的form中发送的，用户可以得到其他用户session 的访问权（即窃取session ）.你需要一种方法来保证session id 没有被改变。

因此有：

使用加密的hash（例如MD5）创建大量唯一的session id来保证session id 不能被轻易猜到。这个hash 表应该包含请求源的一些标识来确认session id（如源地址、源端口或者用户代理）。

将任务下载到客户端

你希望立即向用户提供有关某一个特定功能的反馈，而没有由于从服务器上请求一个新的页面导致的延迟。

在标准的廋客户端方式下，form只是接收输入并将它传送到服务器上进行处理。确认需要一个对服务器的往返交互。你需要使得应用程序具有更好的响应。

下载功能可以确认输入、或将传播改变到客户端的脚本，你还需要确认在服务器上的输入，但是确认逻辑不再需要提供用户的反馈。

这个模式所提供的解决方案是提供了对客户端的反馈，而不需真正将数据发送到服务器上。然而，服务器端的应用程序仍然需要确认数据是正确的而不必考虑客户端代码的作用。在另外一个方面，应用逻辑将会更简单，因为它不需要给出深奥的错误信息。

Conallen [8]描述了一个更简单的模式：THICK CLIENT。

参考文献

[1] Alexander, C., A Pattern Language, Oxford University Press, 1977

[2] Apache Software Foundation, Cocoon, *cocoon.apache.org/2.0*, last accessed in May 2003

[3] Apache Software Foundation, AxKit, *axkit.org*, last accessed in May 2003

[4] Buhr, R., Use Case Maps as Architectural Entities for Complex Systems, IEEE Transactions on Software Engineering, 1131-1155, 1998

[5] Burke, E., Java and XSLT, O'Reilly, 2001

[6] Buschmann, F., and Henney, K., A Distributed Computing Pattern Language, European Conference on Pattern Languages of Programming (EuroPLoP), 2002

[7] Carlson, D., Modeling XML Applications Using UML: Practical e-Business Applications, Addison-Wesley, 2001

[8] Conallen, J., Building Web Applications with UML, Addison-Wesley, 2003

[9] Cunningham, W., Wiki in HyperPerl, *c2.com/cgi/hp?WikiInHyper Perl*, 1995

[10] Daum, B., and Mertens, U., System Architecture with XML, Morgan Kaufmann, 2003

[11] Dougherty, D., LAMP: The Open Source Platform, O'Reilly, www.onlamp.com, 2001

[12] Fernandez, E., Liu, Y., Pan, R., Patterns for Internet Shops, Conference on Pattern Languages of Programming (PLoP), 2001

[13] Fowler, M., Patterns of Enterprise Application Architecture, Addison-Wesley, 2003

[14] Guelich, S., Gundavaram, S., and Birznicks, G., CGI Programming with Perl, O'Reilly, 2000

[15] K¨arkk¨ainen, S., Unix Web Application Architectures, *http://webapparch.sourceforge.net/#23*, 2000

[16] Kienzle, D., Elder, M., et al, Security Patterns Repository, Networks Associate Technologi, *http://www.securitypatterns.com*, 2002

[17] Lyardet, F., and Rossi, G., Patterns for Dynamic Websites, Conference on Pattern Languages of Programming (PLoP), 1998

[18] Rossi, G., Lyardet, F., and Schwabe, D., Patterns for e-Commerce Applications, European Conference on Pattern Languages of Programming (EuroPLoP), 2000

[19] Sommerlad, P., Reverse Proxy Patterns, European Conference on Pattern Languages of Programming (EuroPLoP), 2003

[20] van Duyne, D., Landay, J., and Hong, J., The Design of Sites: Patterns, Principles, and Processes for Crafting a Customer-Centered Web Experience, Addison-Wesley. 2003

[21] Wallace, D., Ragget, I., and Aufgang, J., Extreme Programming for Web Projects, The XP Series, Addison-Wesley, 2003

[22] Welie, M., Interaction Design Patterns, *www.welie.com/patterns/index.html*, last accessed June 2003

[23] Wodtke, C., Information Architecture: Blueprints for the Web, New Riders, 2003

附录：模式摘要

数据源适配器 DATA SOURCE ADAPTER

创建一个统一的接口来访问同一个类型的数据源。只通过这个接口访问数据源，对数据源如何执行接口的改变不会影响到访问它的代码。

包装器 WRAPPER

当从一个web站点提取数据时，用元数据填充数据，元数据不仅指明了每一个数据元素的类型，它同样指明了它与其它元素的关系。增加的冗余导致了处理过程的开支，但是引入了一个更具有可重用性和扩展性的数据表现。

访问控制器 ACCESS CONTROLLER

如果你需要限制对你的站点中某些区域的访问，在中央调度器中对用户授权。这保证了授权执行的持续性，消除潜在的“漏洞”，它可以允许其他非授权的用户访问你的应用。

应用链 CHAIN OF APPLICATIONS

应该考虑创建更小的、更有可管理性不复杂的的应用程序，它可以被用pipe-and-filter 的样式连接在一起，用更灵活的方式提供同样的功能。

将应用表现为服务 EXPOSE APPLICATION AS SERVICE

将可重用的功能表示为一个web 服务，这允许了与其他应用程序的直接集成。使用这个web服务作为你自己的应用程序的输入，你可以继续提供一个基于浏览器的接口。