第 48 期（2005 年 4 月）

骑墙

X 非程序员 -Programmer

软 件 以 用 为 本

联系：think@umlchina.com

http://www.umlchina.com/

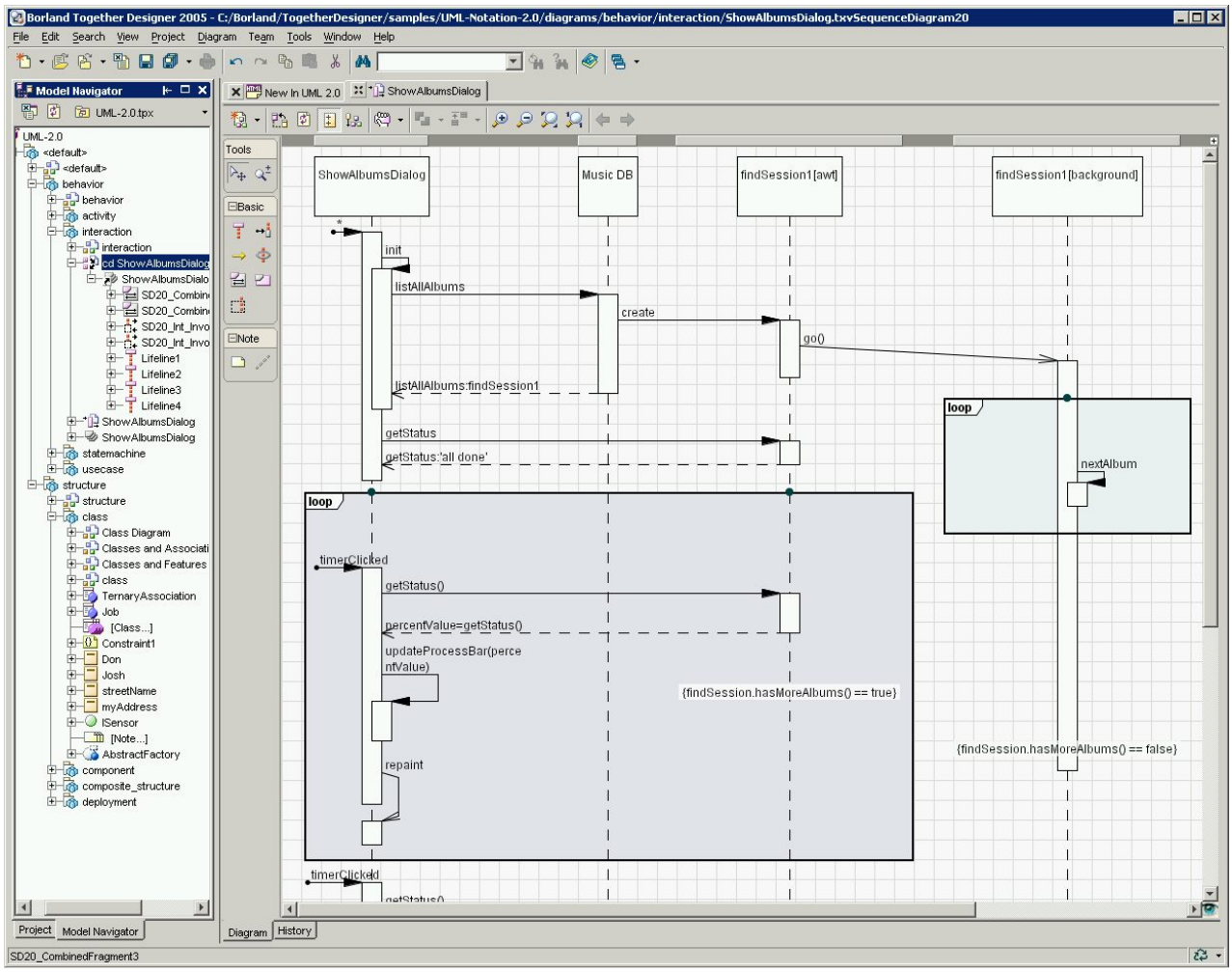
Borland 再次携手.NET

[2005/4/18]

据官方报道，软件开发工具商 Borland 更新了面向 Visual Studio .NET 开发者的建模工具。

Together 2005 for Microsoft Visual .NET 是该建模工具近一年来的首次更新，该工具的特点是基于角色的建模以及对 UML2.0 的支持。

Borland 的 Together 工具是该公司即将推出的代号为 Themis 的软件开发优化框架（software delivery optimization framework）的建模基础。该框架目标是提供完整的生命周期管理平台，被分拆为三个主要功能模块：architect、designer 和 developer。

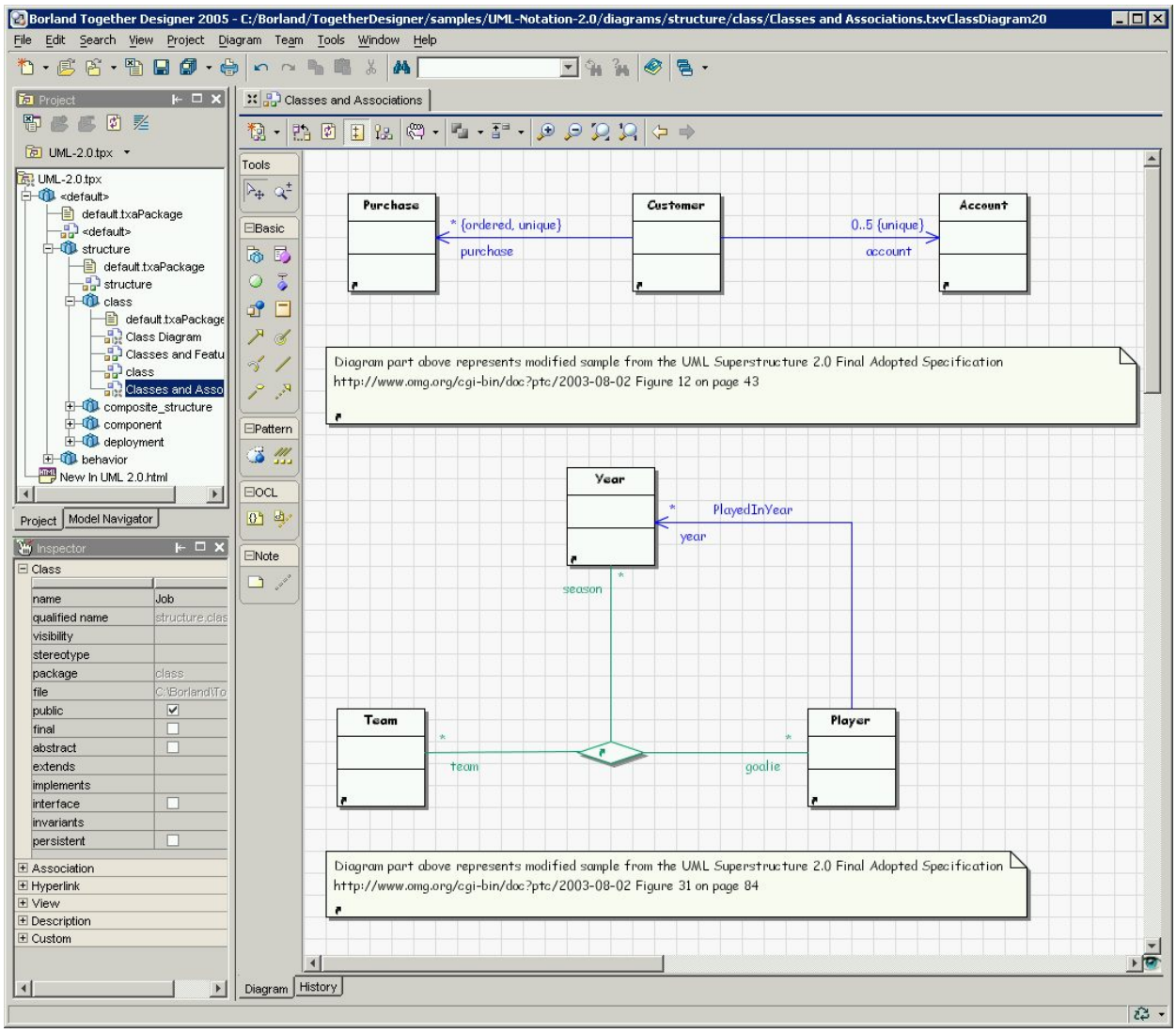


这个新软件是 Borland 包含了基于角色的软件开发的第一个.NET 建模产品，其中将软件活动分拆给设计人员和开发人员。

官方声称，Together 的这次更新架起了微软建模环境、DSL 和 UML Schema 之间的桥梁。

Borland 的 Together 产品主管 Tom Gullion 认为，“开发人员可能想用 UML 来进行高级别的企业架构，用 DSL 来描述服务、组件等。

Borland 具有充分的灵活性，既可以提供今天的市场所需要的（译者注：UML），也可以提供一些我认为微软可以用到的功能，因为他们没有过多支持 UML，而是关注 DSL 这边。”



Together Designer 是提供给架构师和分析人员来处理软件对象设计和建模需求的。该软件在 Visual Studio 2003 集成了 OCL 支持的增强的 UML 建模能力以及图形搜索的新功能。

Designers 现在可以从 IBM 的可视化建模和开发工具 Rational Rose 中导入.mdl 文件。

“Rational 还没有在.NET 上做太多工作，似乎也并不想支持 Microsoft Visual Studio 2003/2005 平台，因此我们想进一步帮助这部分开发人员简化在 UML 建模上的工作，并可以较轻松地移植到 Borland 软件上来”。

Together Developer 2005 构建与上一个版本的核心功能，并增强了对 Visual Basic 编程人员的支持。开发者现在可以重构并审核（audit）VB.NET 代码，另外更新的部分还包括对 VB.NET 和 C#编程人员的新的度量。

Borland 同样在寻找从模型驱动开发角度出发对软件开发的角色进行扩展的方法。模型驱动开发是在编写代码之前创建可视化的模型，并将业务和应用逻辑与开发平台分离开来。Borland 官方声称，他们已经在对 MDA 标准 OCL 和 XMI 的支持上有了起步。

去年 10 月以来 Borland 和微软合作，为即将到来的 Visual Studio 2005 Team System 提供类似 UML2.0 的支持，并希望在后者发布后迅速推出工具。目前 Visual Studio 的第二个 beta 版已经可用了。

（自 internetnews，袁峰 摘译，不得转载用于商业用途）

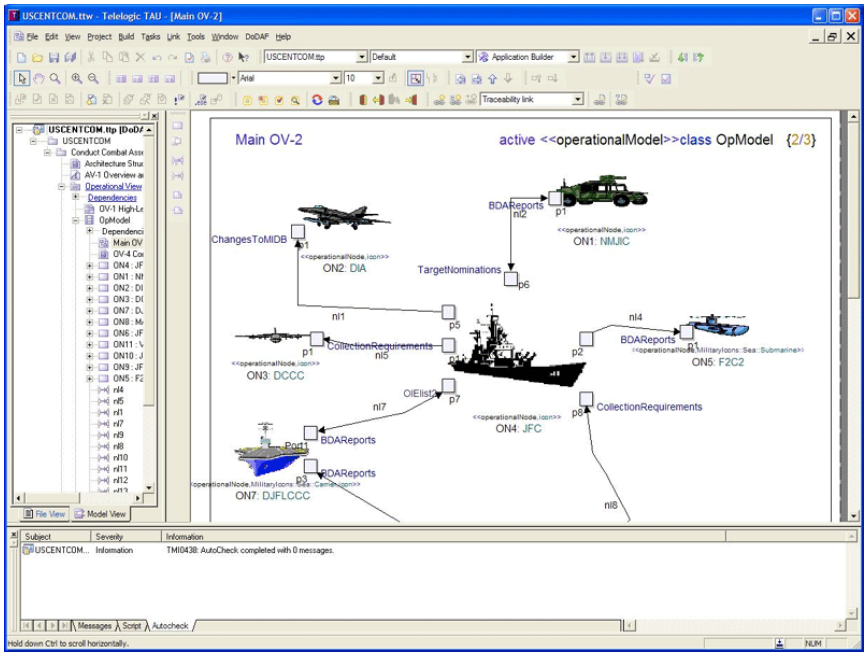
征稿

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

SI 的军队系统 UML 应用

[2005/4/8]

SI 国际在 4 月 4 日声称其基于 UML 的方法解决了军队的命令、控制、交流、计算和智能需求。他们从 1997 开始就在研究面向对象技术了。



UML，一种面向对象的软件设计语言，可以同时应用于业务过程建模和企业架构开发领域。在一个示例中，SI 国际公司使用 UML 完成了战地指挥官集成命令和控制系统。

SI 国际现在面向将它的 UML 经验应用在更广泛的领域。该公司电子政务和企业架构部门副总裁 Earl Pedersen 指出，公司已经为核心的命令和控制模型开发了 81 个不同的过程，这些将应用在所有的服务上。

Pedersen 指出，面向对象技术比过去的方法如功能分解等“提供了更大的灵活性”。他认为过去的方法缺乏处理今天部队需要的变化命令和组织合成的能力。

SI 国际的面向对象方法可能用到大量的螺旋软件开发方法学，但通常和 IBM 的 Rational 统一过程（RUP）一起使用，作为系统工程的方法论。

（自 scw，袁峰 摘译，不得转载用于商业用途）

UML 工具 SDE 获得 Jolt 奖

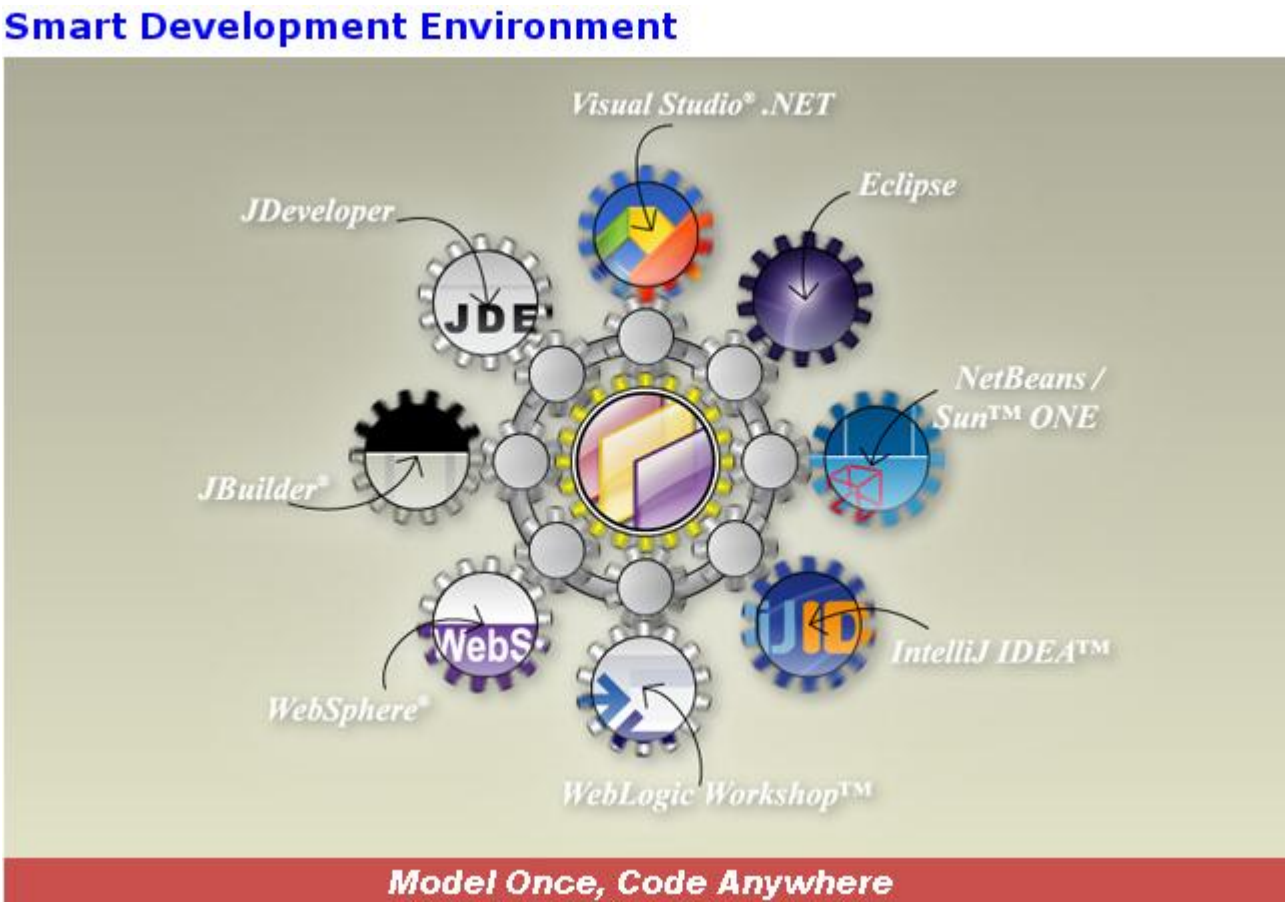
[2005/3/22]

Software Development 杂志颁发的奖项“The Jolt Product Excellence and Productivity Awards”是企业的骄傲，它颁发的对象是那些“联合（jolted）”业界并提高软件开发生产率的产品、书籍和网站。

Smart Development Environment 最终在设计工具类总共 8 个提名中胜出。其竞争对手包括 BM Rational Software Architect, Borland Together Designer 2005, MagicDraw UML 9.0 和 SmartDraw 2.0。

关于 Smart Development Environment

Smart Development Environment (SDE)，是赢得该荣誉的 UML 建模工具，它是可以应用于各种 IDE 如 Visual Studio .Net, Eclipse/IBM WebSphere, Borland JBuilder, NetBeans/Sun One Studio, IntelliJ IDEA, Oracle JDeveloper and BEA WebLogic Workshop 上的方便易用的插件，可视化地提供统一的建模和开发环境。



抛去选择的 IDE 不论，SDE 具有如下优点：

- 全面的 UML 2.0 符号支持

- 代码和可视化模型之间的直觉导航（Intuitive navigation）

- 实时或按需的，双向递增的 UML 模型和源代码同步

- 出众的可视化建模环境

- 诸如用例建模、文本分析和 CRC 卡支持的多种需求管理工具

- 强大的 UML 报告生成器（PDF, HTML, MS Office and Open Office）

- 反向工程支持(C++ Source, XML, XML Schema, CORBA IDL Source, database with JDBC, .NET dll 或者 exe files, Hibernate mapping file, Ada 9x Source)

- 对复杂图形的自动化排列

- Object Relational Mapping (ORM) 支持

（自 i-Newswire，袁峰 摘译，不得转载用于商业用途）

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

OBJECT-ORIENTED SOFTWARE CONSTRUCTION

SECOND EDITION

CD-ROM INCLUDED

- *The* Most Comprehensive, Definitive O-O Reference Ever Published
- An O-O *Tour de Force* by a Pioneer in the Field
- CD-ROM Includes Complete Hypertext Version of Book AND Object-Oriented Development Environment

BERTRAND MEYER

《面向对象软件构造》中译本，UMLChina 辅助教材

即将由 机械工业出版社 出版

Rumbaugh 嘲笑微软在 UML 上的姿态

袁峰 译

讨论

Jim Rumbaugh 是 IBM 杰出的工程师，如今他正领导 IBM Rational 分部的软件建模工作。他和 Grady Booch、Ivar Jacobson 并称为发明 UML 的“三友”，UML 在 1997 年被国际对象组织接收为建模标准。他也参与了 RUP 的开发并且曾经是面向对象分析与设计方面的 OMT 的主要开发者。上周，InfoWorld 编辑在 Santa Clara 召开的 SD West2005 会议上对 Rumbaugh 进行了访谈，讨论了 UML、SOA (service-oriented architectures) 和 ESB (enterprise service bus)技术。Rumbaugh 对微软及其在 UML 上的骑墙姿势表示了不屑。

IW: 可以谈一下 UML 的背景以及当年你是如何参与进去的吗？

JR: 当然可以。当年有大把的方法论，每个都有自己的建模方法。人们不断地询问我们，为什么不统一？但所有这些人都来自不同的公司，因此他们没有动机去做这个事情。UML 的出现是在 Rational 的 CEO，Mike Devlin，聘用了我、Ivar Jacobson 和 Grady Booch 之后。我们三个到了一个公司，Grady 和我先合并得到了我们称为的 Unified Method 方法，而实际上 Ivar 建议把它称为 Unified Modeling Language。在此基础上不断有其他人加入进来，这就是一切的开始。

IW: 目标是什么呢？

JR: 嗯，最初的目的是从不同的建模方法中提取一些共同之处，统一起来以便大家都可以使用，结果最后得到了 UML。现在，除了我们仨之外以及 Rational 之外的很多人都加入了进来，这也是 UML 成功的原因。你知道，UML 现在已经淘汰了很多其他的方法，这些方法的使用者最终放弃了使用这些方法而开始使用 UML。因此我想 UML 已经成功了。

IW: 微软并没有真正支持 UML。他们曾经告诉我，他们并没有发现使用 UML 的太多必要，这说明了什么问题呢？

JR: 我们会看他们是否真正支持 UML。他们现在也在用其他方式支持 UML。他们似乎在骑墙，我打赌如果他们发现有足够的人想用 UML，如果他们不能说服人们使用他们的方法，他们最终会回到 UML 上的。但再重复一次，很多的人都认为 UML 是有用的。只是微软说了一些其他的话，这代表不了任何事情。他们并没有在所有可能的领域获得成功，这是肯定的。[编者注：微软曾为 UML1.4 提供支持，但并没有为新版本 UML2 提供支持。但 Borland 为微软的 Visual Studio 提供了 UML2 的支持。

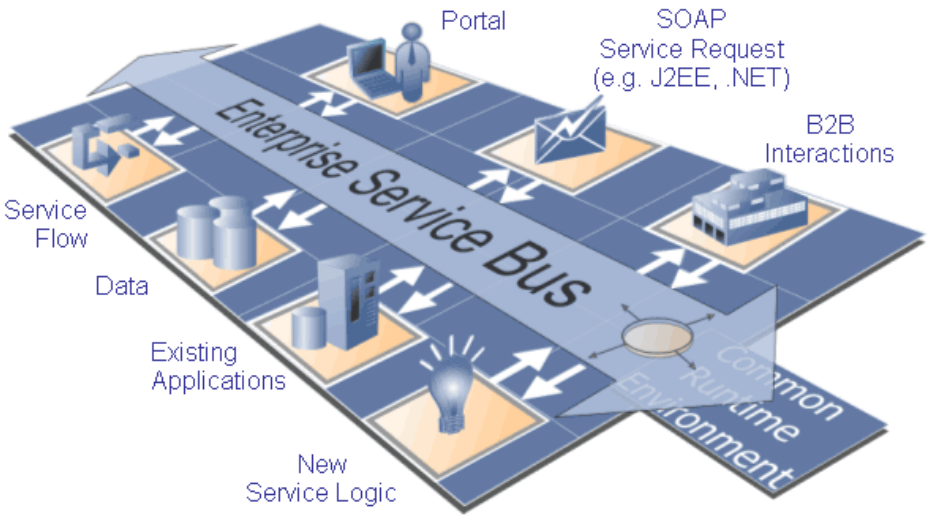
IW: 你曾经谈到过模型驱动架构的重要性。UML 是它的一种表现形式吗？基于 MDA 的技术有哪些？

JR: UML 是一种建模语言，因此它是表达模型的一种方式。MDA 是一种构建模型的方法。因此这两者是正交的。你需要表达模型的方式，也需要开发模型的方法。因此这两个一个是一种开发过程，而另一个是语言、格式、表达—嗯，你怎么叫它都可以。

IW: 你在会议上的发言中谈到了 SOA 和各种标准。但从来没提到 Web services，为什么把它忽略掉了？

JR: 哦，Web services 是 SOA 的一种。实际上它位于 SOA 之下。Web services 曾经是 SOA 的第一个示例，或者说是一个好的例子。

IW: 你提到过 ESB。是否 IBM 已经有了一个企业服务总线 （Enterprise Service Bus）产品，还是你们将拥有一个？



JR: 实际上它已经嵌入到我们的中间件产品里面了。它当然不是一个单独的部分。从某种意义上说 ESB 是产品需要支持的一个概念，因此它不是一个可以单独分离出来的东西，而是你的中间件必须支持的一个属性，这个属性指的是你需要轻松地从任意一处调用另外其他地方，而不让诸如机器码、语言和平台之内的东西来妨碍你。

IW: 那么 ESB 的基础是什么呢？它更多的是一个消息总线（messaging bus），你同意这种说法码？

JR: 哦，它基于在不同事物之间交换清晰的信息的想法。因此基本上它就是在应用程序之间来回地发送消息。

IW: SOA 现在有多流行？

JR: 对，正如我们曾经指出的，Web services 是一种 SOA。因此用户们正在使用它，而且我们希望使用得会越来越多。我们的部分工作就是让它更容易发生。现在的一些困难比如在平台之间的连接困难。因此 IBM 正在开发中间件。中间件的任务就是帮助人们构建可以交互、可以完成任务的应用程序。现在已经有了很多相关的产品，我希望以后会有更多。

IW: 因此 SOA 目前很大程度上基于 Web services?

JR: 嗯，这当然是一个主要的部分。但还有大量 SOA 没必要用 Web service 的。我们认为 SOA 的未来就是把那些庞大的系统分拆为小块，可以容易地进行重配置、替换。

IW: 如果一个公司从过去庞大(monolithic)的架构转移到 SOA，是不是要花很多钱？比如我这里有一个来自 Sun 的人，他可能会告诉我如果要部署 SOA，IBM 会卖 IBM 的全球服务(Global Service)给我。是不是要转移到 SOA 上就要花大笔的钱？

JR: 嗯，你没必要一口吃成胖子，可以一步步来。当头头们说，“我们马上就要把所有东西都转移到新的 Idea X 上拉，我们要彻夜执行，每个人都要”，我们和公司都会被毁掉。这种做法是毫无效果的大灾难，总的来说，你需要一次一点地来，可以现在可以得到回报的地方开始。

首先要做的是态度上的转变。大家过去在 It 方面是紧凑、封闭的想法，现在需要的是打开，把一些服务外包出去，这是真正的文化上的障碍。这也是实际上的障碍所在，CEO 的工作、CFO 和 CTO 的工作就是要消除这些障碍。

IW: 你提到了服务编排(（choreography）。你对 BPEL 的印象是什么?

JR: 嗯，我们正在用这个。它是相当低水平的 choreography。但是在这些标准的基础上进行工作是很重要的，因此我们有一些产品是基于这些标准的。

IW: Java vs. .Net 之战，谁会是胜利者？

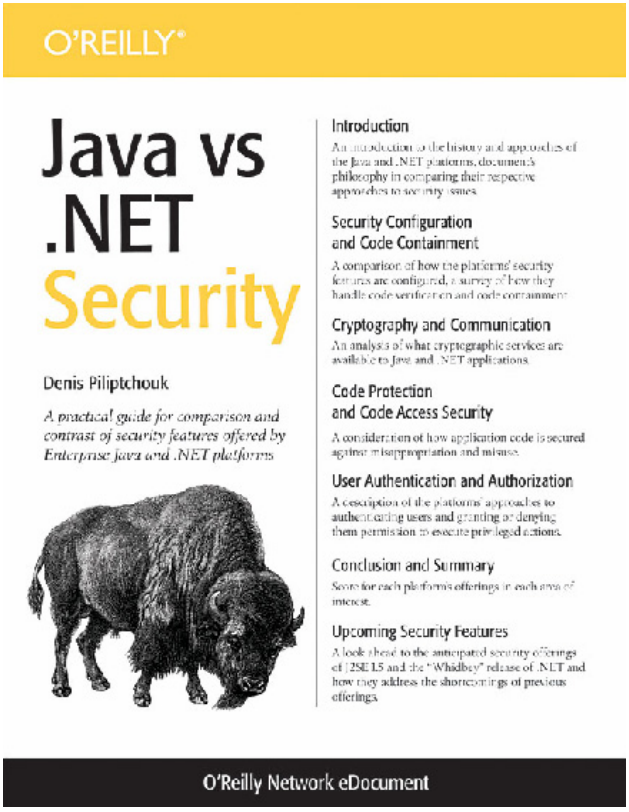
JR: 如果一定要一方放弃才算胜利的话，我怀疑不会有一个绝对的胜利者。我的意思是，我认为他们都会存在下去。

IW: 和 Sun 在 Java 上相比，微软似乎找到了从.Net 上挣钱的方法.

JR: IBM 也在支持 Java，我们将致力于：我认为微软会有一个强大的对手。

IW: 你认为 IBM 在领军 Java 技术吗？甚至比 Sun 更靠前？

JR: 嗯，我认为 IBM 大力推动着 Java，但我并不认为我们希望 Sun 放弃支持它，当然 IBM 是主要领导者中的一员。



IW: IBM 和 Rational 的合并现在看来效果如何？是不是 Rational 觉到了某些方面的限制？

JR: 实际上我觉得和过去相比，大多数的 Rational 员工觉得现在产品的范围更广了，因为 IBM 覆盖的范围比 Rational 要广泛。而且我们在过去从未涉猎的领域也开始销售产品了。因此实际上我们的范围增大了。

软件工程技术丛书

设计系列

企业应用架构模式

Patterns of Enterprise Application Architecture

(美) Martin Fowler 著

王怀民 周斌 译

UMLChina 审校

UMLChina 训练辅助教材

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

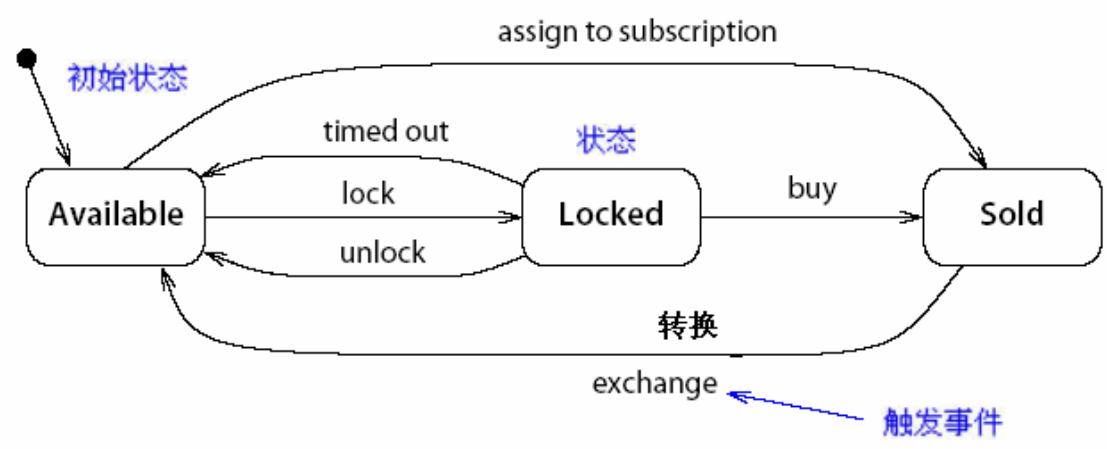
Scott Ambler

字斟句酌看 UML 变迁（下）

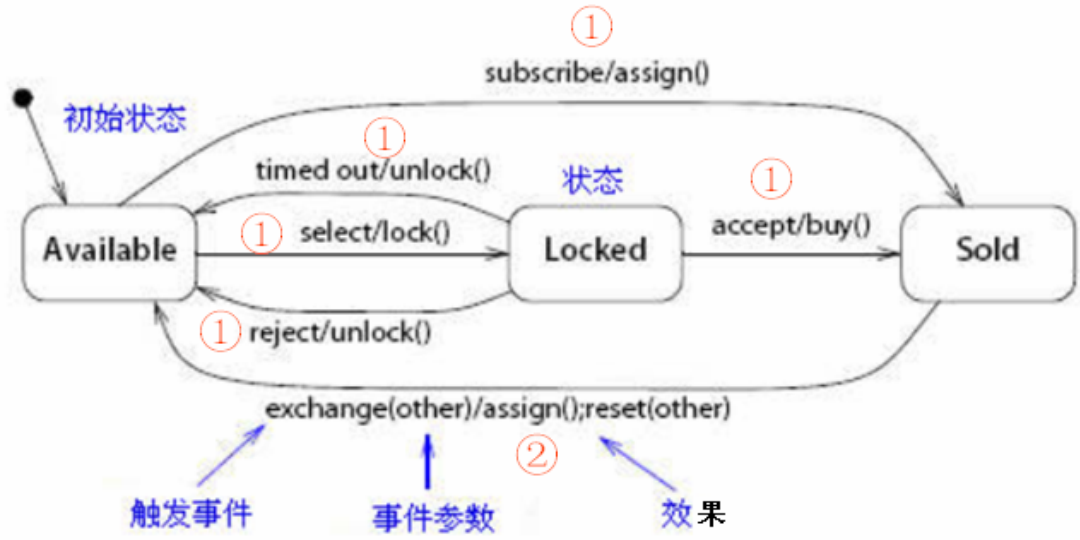
潘加宇、李嘉兴 著

讨论

状态机图



第一版状态机图



第二版状态机图

这是Ticket（票）类的状态机图。

①assign to subscription变成了subscribe/assign(),timed out变成了timed out/unlock……。作者可能是为了更丰富地说明状态机图中转换的用法。状态机图描述对象的一组属性值，当发生指定事件时，对该事件做出的相同性质的反应，它既不是从系统的全局结构研究对象，也不反映系统对外提供的某种价值（用例），而是一个对象随时间变化穿行各个状态之间的轨迹，所以常用来精细刻画对象的行为。subscribe/assign()的意思是，当发生subscribe（预订）事件时，该对象执行assign(分配)动作，并进入Sold（已售）状态。对应于类似以下的代码：

```
public void subscribe ()
  {
  switch (state.value)
    {
    case State.Available:
      assign();
      state.value=State.Sold;
      break;
……
```

上面的代码虽然直接，但会有大量switch…case，显然结构不良（Switch Statements），可以重构（Replace Conditional with Polymorphism）为State模式，转换的代码被放入相应类的不同方法中。根据多态原理，相应的方法被调用。这样所有的状态和转换都是明确的，不需要使用任何条件语句，增加了代码的可读性。

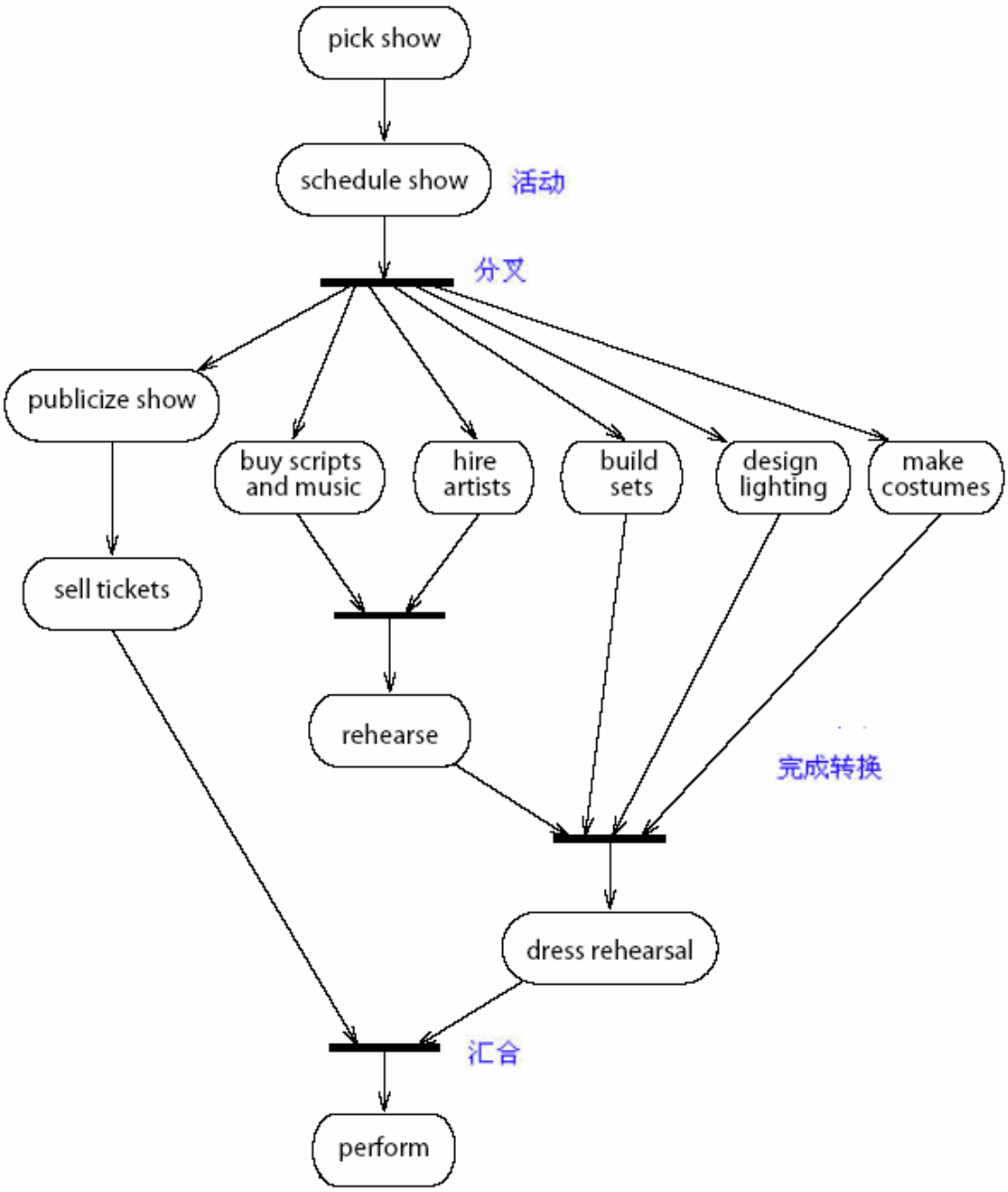
此处改变不是UML2的新特性。

②效果。在UML1中，事件后发生的动作叫动作（action）列表，而在UML2中，叫做效果（effect）列表。效果的定义是：当转换引发时执行的动作（action）或活动（activity）。效果的执行具有执行到完成（run-to-completion）的语义，在执行过程中，不会处理其他的事件。

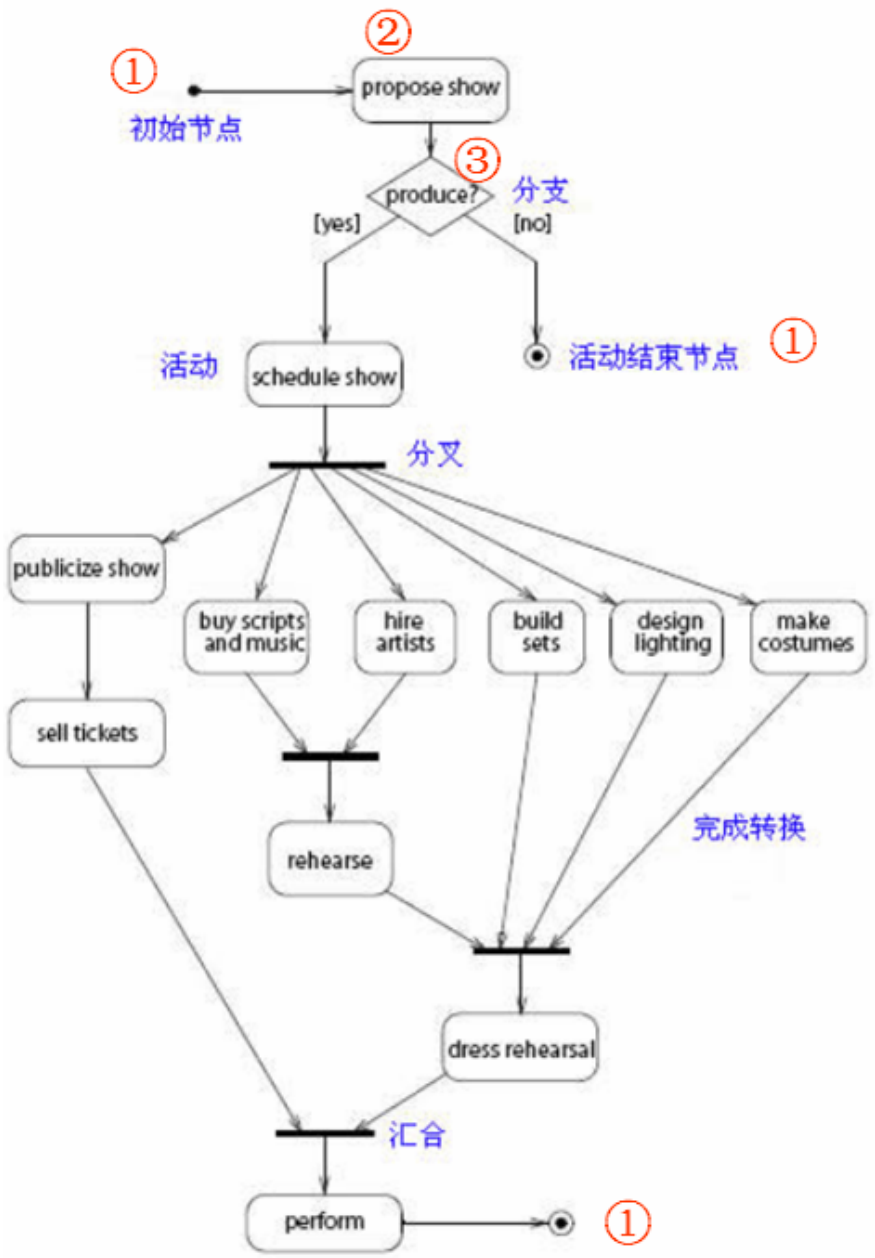
其他

状态机图在UML2中有很多变化。状态的种类增加了选择（choice）、入口点（entry point）、出口点（exit point）等概念，增强了表达力；还添加了一种状态机图的变种：协议状态机（protocol state machine）图。但本文只讨论《UML参考手册》不同版本中这几个例图的区别，所以就不多涉及了。

活动图


pick show
schedule show
活动
分叉
publicize show
buy scripts and music
hire artists
build sets
design lighting
make costumes
sell tickets
rehearse
完成转换
dress rehearsal
汇合
perform

第一版活动图



第二版活动图

①添加了初始节点和活动结束节点。初始节点（initial node）指明了当活动被调用时，从哪里开始执行。当包含初始节点的活动被调用时，一个控制令牌被放置在初始节点中，活动可以开始执行。活动结束节点(activity final node)代表活动执行完毕，执行活动结束节点会强制终止活动中所有的流以及活动的执行。UML1中也有这么两个点，但不叫初始节点和结束节点，叫初始状态（initial state）和结束状态（final state）。在UML1中，活动被认为是状态机的变体，这确保了模型的良构，但是却给模型的形式增加了很大的约束。在UML2中，状态机和活动的元模型被分离开，并且活动的语义主要（松散地）基于Petri网的语义（把活动图理解成令牌沿活动边流动而不是状态

转换），添加了许多新的流特征。节点、令牌等概念就来自Petri网。

相对于“初始节点”，“活动结束节点”前面多了一个词“活动”。因为UML2活动图增加了另外一种结束节点：流结束节点（flow final node）。活动结束节点终止活动图中所有的活动；而流结束节点终止包含该节点的活动的线程，但不影响并发的活动。这样，“结束节点”代表一种抽象类型的节点，使用时需要指明是哪一种结束节点。

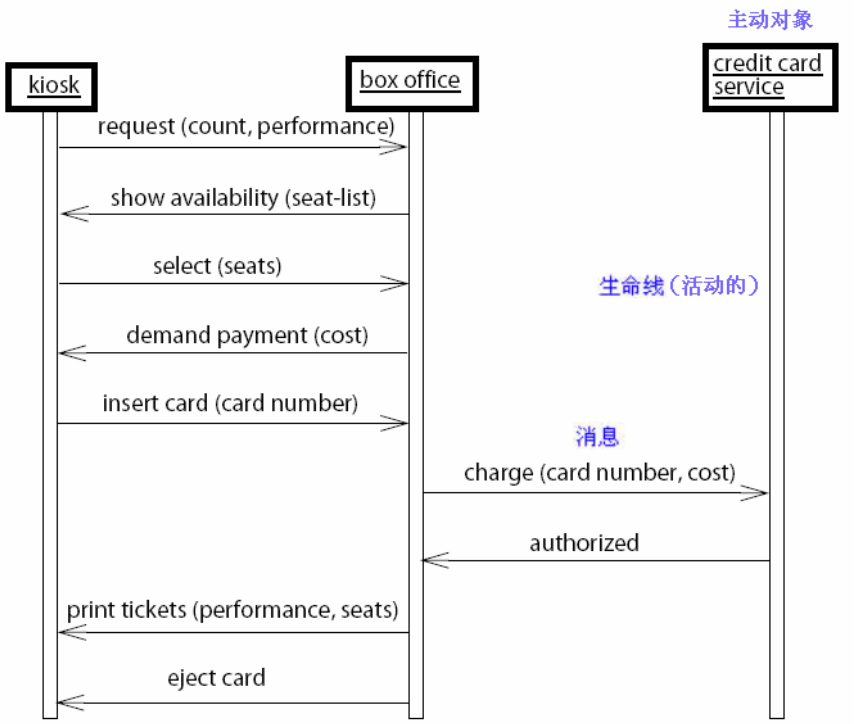
②Pick show变成了Propose show。这应该是一个用词上的改正。Pick是从已经有的剧目里挑一部，Propose（策划）一部新剧才会引起下面的一系列活动（招演员等）。此处改动和UML2无关。

③添加了分支结构，作者可能想使描述更符合现实，也可能是为了向读者展示一下分支的使用。把produce放在菱形里面这种简写的监护条件表示法是UML2新增的，相当于[produce=yes]和[produce=no]两个条件值。

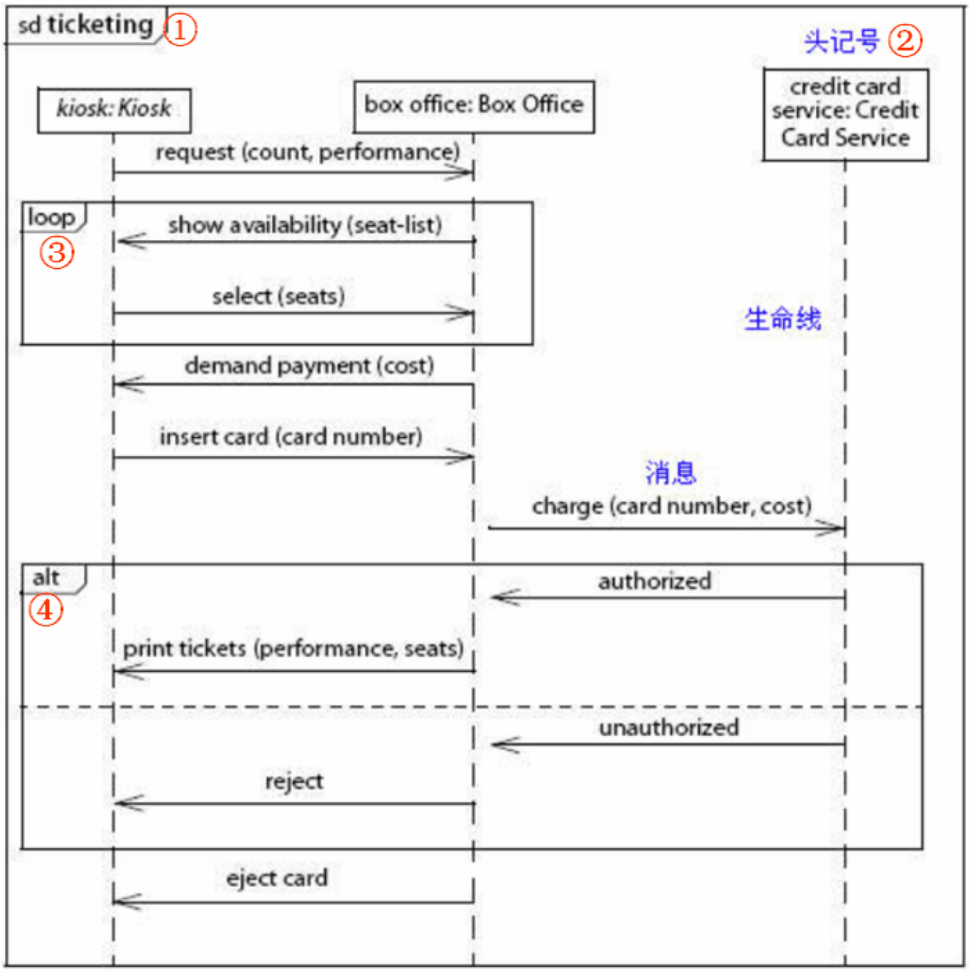
其他

另外的变迁包括：多维分区（泳道）、时间、接收信号、参数、连接说明、针脚、扩展区……活动图在UML2中改动十分之大。

序列图



第一版序列图



第二版序列图

①左上角的小五边形标签“sd ticketing”用来标识这张图，sd表明这是一张序列图（sequence diagram），class表明是类图，comm表明是通信图……。这样就相当于给这张图加了一个“框”（frame），以便整个嵌到更大的“框”里，实现交互的复用。我们在为用例的多条路径绘制序列图时，经常会发现图与图之间相当一部分交互是重复的，但这点交互又未必够得上子用例的级别。这时用一个框把它框起来，在别的地方就可以通过ref关键字迅速复用，如：ref Login。当然，如果没有复用需要，就没有必要加这个框，一般来说稍有UML知识，就能通过我们的肉眼辨别出来这是一张序列图。

②“主动对象”（active object）变成了“头记号”（head symbol）。序列图中的对象已经改成不是主动对象，还用了“对象名：类名”的完整方式来表达。此处应是一个更正，与UML2新特性无关。

③loop。UML2中序列图的表示法很大程度上基于ITU制定的消息序列图（Message Sequence Charts，MSC）标准。UML1中的交互不适合用来构建更复杂的元素，比如条件、循环和并发线程。这一点为许多人所诟病。UML2作

了很大的改进。UML2交互中的复合片断元素能够以简洁而有效的方式处理结构化元素。一个复合片断包含一个说明它是哪一种构造的关键字，例如：loop构造表示循环。

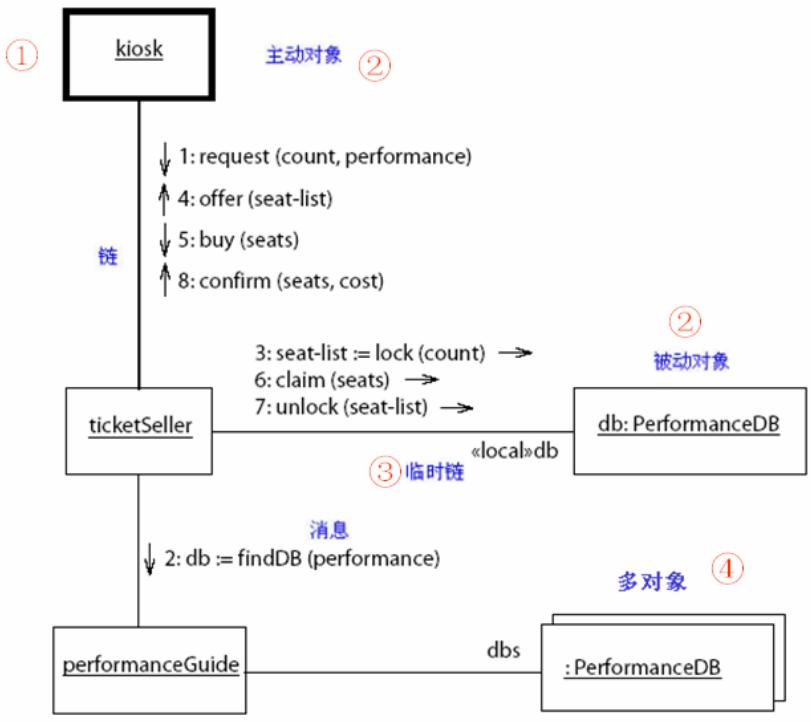
在UML1中表示循环，做法是在图上消息旁加上一个“*”，例如：*[for each seat]。

④alt。同上，alt也是一个构造关键字，表示分支。用水平虚线隔开不同条件下的消息组，根据监护条件决定走哪个分区。

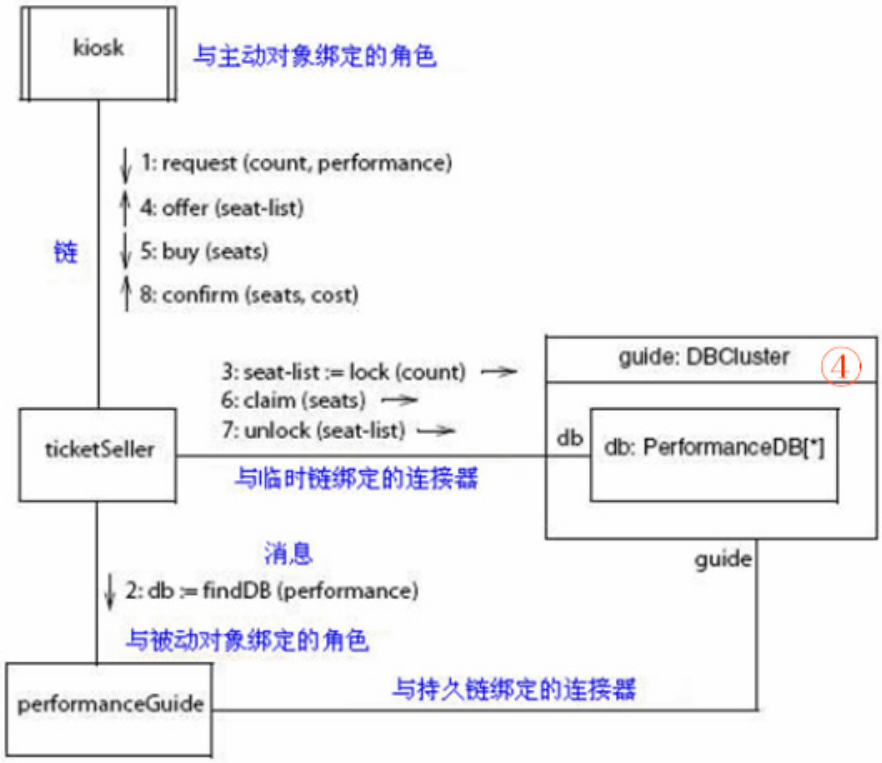
如果用UML1来表示分支，可以在图上用中括号把条件放在消息旁边，如果工具允许，可以把条件和消息绑在一起。

总体来说，UML1序列图的表达能力是很弱的（第一版的例图中作者就干脆没画循环和分支），比如并行的消息，由于没有像UML2中par这样的关键字，因此在UML1中是没法表达的。基于MSC的UML2序列图表示法，比起UML1，在清晰性上有很大改善。各种构造一个套一个结合使用，能精细地表达各种算法，和代码的映射更加直接了。不过又会引出争议：序列图能表达职责分配就够了，真的有必要那么细吗，具体实现细节用代码（字符形式）表达岂不是更方便？这个问题的尺度如何把握就见仁见智了。

通信图



第一版协作图



第二版通信图

UML1中的协作图在UML2有了新的名字：通信图。

①主动对象从大黑框变成了双重边。UML1用粗边线的矩形表示主动对象，UML2改进了主动对象的表示法，用左右边为双线的矩形表示。比起区分粗和细来，区分单和双容易多了。

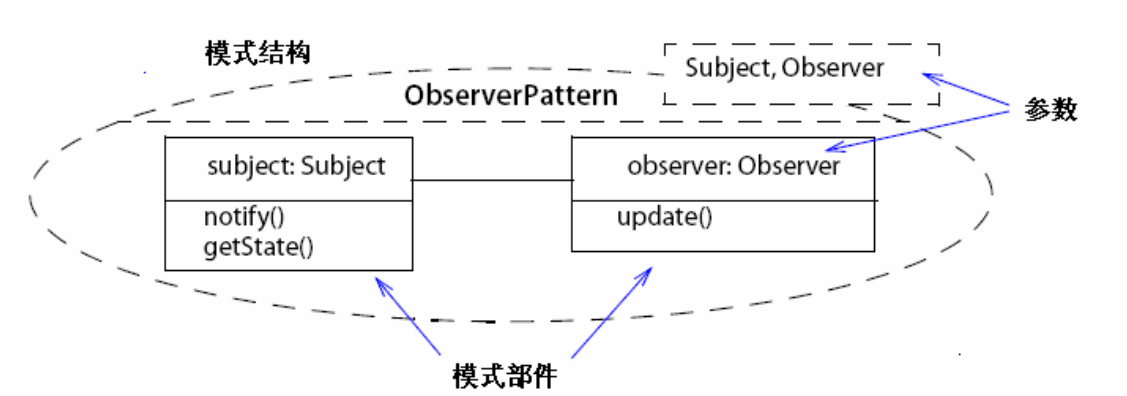
②“主动对象”变成了“与主动对象绑定的角色”，“被动对象”变成了“与被动对象绑定的角色”。通信图更着重描述对象在交互中承担的角色，一个对象能够扮演（绑定）多个角色。名称下面也没有下划线了。

③“临时链”变成了“与临时链绑定的连接器”。就像对象和角色绑定一样，链（link）也和连接器绑定。连接器（connector）是结构化类元中的或者协作中的两个结构化部件（structured part）之间的连接。它描述的是只在特定上下文中适用的上下文关联，比如类元中的对象或者参与协作的对象。

④多对象（multiobject）是UML1中的概念，在UML2中已删除。多对象是为了让建模者以两种互补的方式对一个集合建模：一种方式是作为单个对象，具有对整个集合的操作，另一种方式是作为多个对象的集合，每个对象有自己的操作。这一概念在UML2中可以通过使用结构化类来建模，而不必使用多对象。结构化类包含一组对象。其它类既可以与该结构化类关联，也可与其部件关联。

其他

“协作”在UML2中已经不是一种正式的图，变成了组合结构（Composite Structure）的一种，可以用来描述模式。模式就是一种参数化的协作（collaboration），例如可以把GOF的Observer模式表现为：



结语

除以上提到的这些图的种类变化外，UML2还新增了交互概述（Interaction Overview）图，综合了活动图和序列图的特点；新增了时间（Timing）图，为序列图加上了时间的量度，以适用于实时开发。

我们再来总结一下所有图的变化程度：

基本没变：用例图

有些变化：类图、对象图、包图、部署图、状态图、通信图

变化很大：活动图、组件图、序列图

新增：组合结构图、交互概述图、时间图、协议状态机图

从词条上看，《UML参考手册》第一版有385个词条，第二版增加到了625个词条。当然，实际上的增加没有那么夸张，因为第一版出现的词条即使在UML2中不再存在，第二版也会列出来说明“UML2已经废弃此条”。

本文只是重点对《UML参考手册》第一版和第二版例图进行比较，尽力分析和解释其中的变化。因为笔者水平有限和对UML2的认识不足，某处为什么要这样改，笔者的说法可能是错误的。发信去问三位作者能保证得到更准确的答案，但由于时间精力原因笔者没有做到。

（本文首发于2005年4期《程序员》）

The Unified Modeling Language Reference Manual, Second Edition

JAMES RUMBAUGH
IVAR JACOBSON
GRADY BOOCH

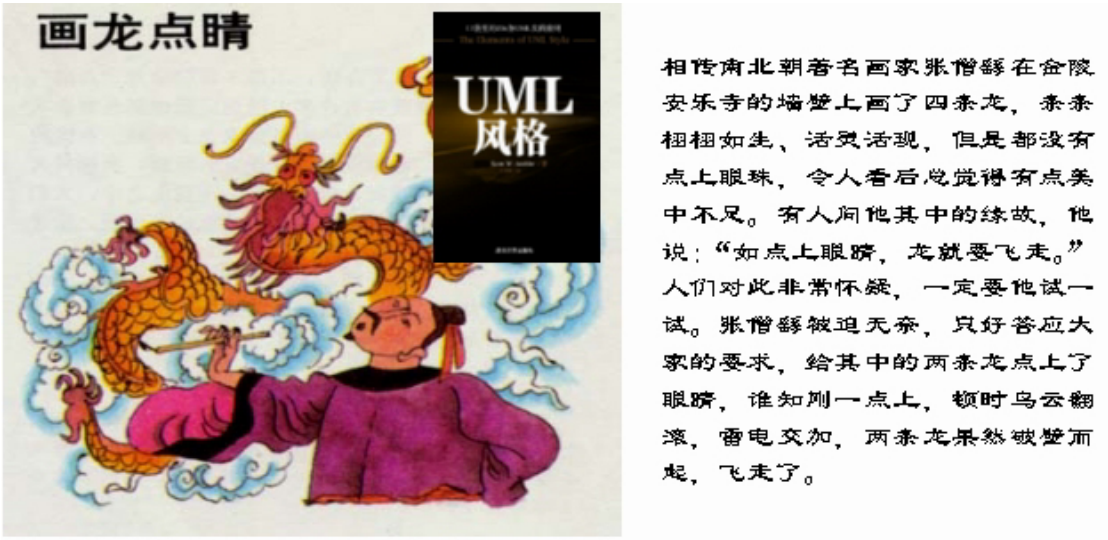
Covers UML 2.0

UMLChina 译
（王海鹏、李嘉兴）

OBJECT TECHNOLOGY SERIES
BOOCH
JACOBSON
RUMBAUGH
ADDISON-WESLEY
SERIES EDITORS

CD-ROM included

《UML 参考手册》2.0 版中译本
即将由机械工业出版社华章分社出版



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。

敏捷软件开发中的需求工程

Frauke Paetsch 著，Anyion 译

讨论

4 用于敏捷方法的需求工程技术

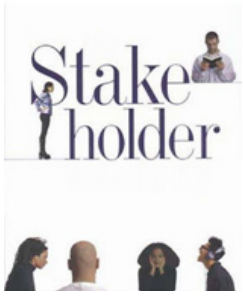
4.1 概览

在不同的敏捷方法中几乎可以找到在第二章里描述的所有需求工程技术，虽然它们可能被用在不同区域或不同范围内。本章描述了敏捷方法是如何与这些技术相结合的，并且给出一个分析摘要。

4.2 客户介入

所有敏捷开发技术的一个关键是客户必须是“可访问”的或在“本地” (XP) 。这是快速反馈和沟通的基础。这两种方法都可增进对需求(对开发人员而言)和开发过程（对客户而言）的理解。客户介入是敏捷方法的第一个目标（值得一提的是，在这里的“客户” 与“用户” 是同一概念。传统上，在 RE 中，所谓客户是指支付软件系统的人，而用户是指使用这个软件系统的人。） 。例如，XP,假设存在一个"理想的" 客户代表: 该代表能正确地回答开发人员可能提出的所有问题，他被授权做出约束决定并能做出正确的决定。

涉众介入也是当前需求工程的一个关键要素。涉众拥有所要开发的系统的相关领域知识。不同的 RE（需求工程）捕获技术目的在于从所有介入者(用户、领域专家等等)手中获得尽可能多的系统相关信息。1995 年，来自 Standish 小组的 CHAOS 报告 [Gro95] 显示了客户介入的重要性。客户介入被认为是项目成功的首要条件，缺乏客户介入是项目出现问题的主要原因。因而，传统的 RE 和敏捷方法一致认同涉众介入的重要性。传统方法和敏捷方法之间的区别是，在传统方法中客户主要介入 RE 阶段，而不介入实际的软件开发阶段；而在敏捷方法中客户介入整个开发过程。



虽然敏捷方法基于用户介入整个开发过程，问题仍然存在。在许多情况下，该方法需要有几名不同背景的客户，但有时只有一名客户有时间与开发小组工作。这意味着，不是出现的所有问题都能得到足够详细的答案。这也许会导致开发进程的延迟或开发人员可能做出不正确的决定。此外，客户也许对一些业务领域没有充分理解，因此他无法告诉开发小组相关的需求。即使需求能在小组会议(DSDM，Scrum)上被捕获，它不能保证有所需背景的用户或客户会出席。但在传统需求捕获中也会出现同样问题。虽然有许多不同的技术可以获取需求，但没人能保证所有问题域都覆盖到。其它技术例如评审或测试是必需的，以确保能发现这些问题域。

4.3 访谈

因为客户介入是敏捷软件开发的基本目标，最普遍的 RE 相关技术是访谈。访谈提供对所需知识的直接和“未经过滤”的访问。我们知道通常知识传递链会导致误差和错误信息。能直接与客户联系的开发人员获得的信息错误较少。面对面交流在客户和开发人员之间建立了关系，这增加了自信和他们彼此的信任。所有敏捷方法强调：与客户谈话是获取所需信息的最佳方式，并可避免误会。如果任何问题没有被弄清楚或被隐晦地定义，客户或开发人员应该设法与相关负责人沟通以明确避免知识的间接传递。

4.4 划分优先级

划分优先级存在于所有的敏捷方法中。普遍做法是先实现优先级高的功能，以便能为客户交付价值最高的软件。在开发期间知识和对项目的理解会不断增加，新的需求也不断增加。所以,在整个开发过程中优先级划分被反复进行，以保证优先级被及时更新。划分优先级帮助客户了解在需求和他们能影响系统的方式之间的区别。

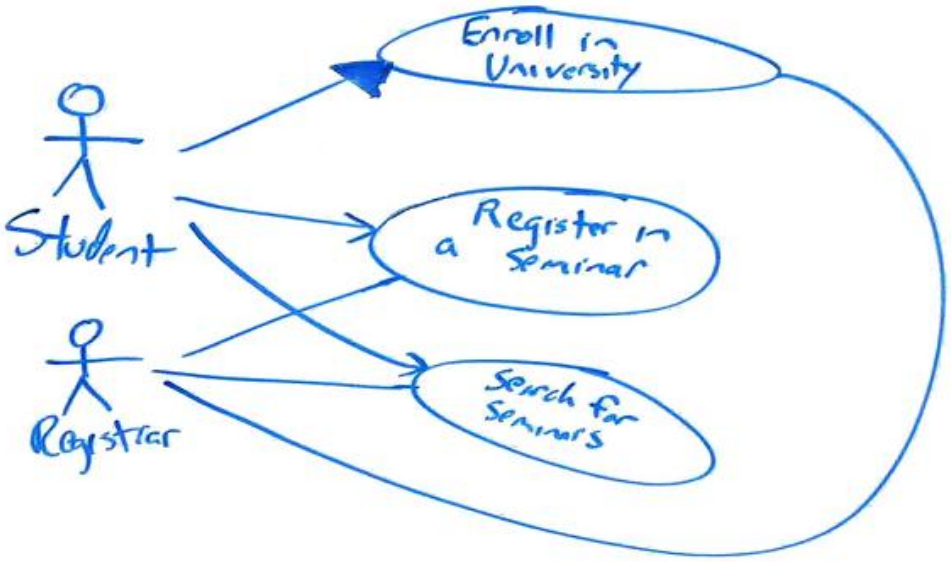
4.5 JAD

在 ASD 中使用 JAD 会议来加强客户介入。在 JAD 会议期间开发人员和客户讨论所期望的产品特性。这种讨论可能是非常有生产力的，因为会议领导可以防止参加者“逃跑”。另外，短小的 JAD 会议侧重交互，这不象 XP 中的本地客户策略那样束缚客户代表。会议的结果通常被记录下来，如果将来出现问题会有据可查。在敏捷开发中可以放松对文档的要求。为了持续获得反馈，在开发过程中应频繁举行 JAD 会议。

在 DSDM 中，JAD 会议用于在项目的开始时获取对新系统的理解。JAD 促进了在不同参加者小组之间的合作、理解和配合。由于参加者会具有不同背景，关于新系统的不同部份的信息可能被收集并为进一步需求捕获提供基础。它们也鼓励客户介入和信任。这显然与敏捷原则是一致的。

4.6 建模

尽管在 AM 中使用建模的目的是不同的。例如在 AM 中，模型被用来在开发中交流对系统的一小部份的理解。大部分模型用过就丢掉了。它们被画在白板上或纸上，实现了它们的目的后就会被擦掉。大多数模型不会成为永久文档或系统最终模型的一部分。当在 RE 中使用建模时，有多种不同模型。开始是整体系统的概要模型，然后是具有更多细节的其它模型。所有这些模型会成为系统文档的一部分并需要不断更新。



在 FDD 中建模的目的与在 RE 中相同: 被开发的模型体现了整体系统，而且开发应基于这个模型。但由于设计和构建阶段是迭代进行的，需求不断变化，因此模型也可能在开发过程中晚些时候被建立。模型代表开发开始的第一步，但没有全面地描述最终系统。

4.7 文档

在敏捷软件开发中极少使用全面和一致的需求文档。一些敏捷方法包括一种文档或建议使用需求文档(DSDM，Scrum，Crystal)，但主要内容和如何扩展部分并没有被详细描述，而是留给开发组去决定。

在敏捷开发中，缺乏文档也许会造成长期问题，但加快了开发速度。使用文档是为了在成员之间分享知识。一名新团队成员将有许多关于项目的问题; 这些可以由阅读和理解文档得到答案。询问其他团队成员可能会减慢工作进度，因为解释一个复杂的项目是需要时间的。当需要对一个已完成的项目做改动时会发生同样的情况。最初开发该项目的开发人员(他去了其它公司或正在开发一个新项目) 很有可能不能承担该任务，因此需要一个新团队来完成修改。由于敏捷方法经常产生干净和紧凑的代码，长期问题可以被减少。另外，当系统进展到一个纯生产步骤时，客户经常会要求敏捷团队在团队解散前编写设计文档。文档的范围常常是非常局限和集中于系统的核心方面。这增加了修改软件时及时更新文档的可能性。

另一方面，根据代码而言，敏捷方法应该比传统方法更有生产力，因为与传统的 RE 方法相比它们花费了较少的时间来编写文档。当传统方法趋向于编写过多的文档时，敏捷方法倾向于在不充分的文档前提下允许犯错。

传统方法试图设法编写足够多的文档去回答那些大多会在将来被问到的问题。为了做到这一点，他们需要(1) 预测未来问题，并且(2) 以简明和可理解的方式回答它们。这两点都很难做到 — 这就是为什么编写好的需求文档是很困难的。通常，为了能够很好地覆盖未来的问题，传统方法也许编写了太多文档(例如，回答那些从未被问到的问题实际上是浪费金钱)。这导致了新的问题：无法保证及时更新所有的文档，和读者无法轻易地为他们当前的问题找到相关信息。

文档的利弊和团队大小有关。对一个大的团队，也许使用文档来代替多次向不同的人解释同样的事更好。当试图调整敏捷方法以用于更大的团队时，与小项目相比它们经常要增加文档数目[Coc02] 。

4.8 验证

需求验证是所有敏捷软件开发方法中的一个重要角色。软件可以通过评审会议和验收测试来验证。这会增加客户对程序和开发小组的信心，因为被展示的系统象预计的那样运行。评审会议还表明项目在范围和日程计划内。

所有方法确认需求的方式都是类似的。它们使用不同类型的评审会议来展示所开发的软件。客户可以使用该软件，体验系统是如何具体运作的，以及哪些功能已经实现。如果有问题，在每次评审会议上他们可以从开发人员那里立刻得到答案。客户也可以和开发人员讨论功能的实现方式并要求修改设计。在评审会议期间他们可以了解设计和技术的优势、劣势、好处和限制。客户能了解已实现的需求是否被理解正确，以及它们是否正确地实现了。客户还可以进行验收测试去确认系统是否按所期望的方式运行，如果没有，及时澄清问题。问题和变动请求会影响未来的开发。根据方法和项目情况，在每次评审后软件被投入生产。这种快速部署通过提供更加快速的投资回报来改变软件项目的经济状况。

特别是在 XP 中,客户应该总是参与开发，能提出问题的可能性和看到测试被通过增加了开发小组对软件的信心。

由于所有敏捷方法都基于小发行版本，验证会被严格强制执行。至少对每个新的发行版本，开发人员能从客户那里得到反馈。这种开发方式可与原型进化法（evolutionary prototyping）相比，这二者都基于同样的经常交付概念，交付哪怕极少的可运行代码以提高反馈。二者的区别在于敏捷方法更强调测试。

4.9 管理

有限的需求管理是敏捷软件开发方法的一个共同“属性”。从 RE 的角度看它应该是能够跟踪需求、设计变动、或文档以了解为什么做这些变动。但跟踪改变需要做更多工作和编写额外的文档。另外，没有证据表明跟踪变动能为项目提供任何金钱利益。

如果需求体现为两个组织之间的法律合同，那么缺乏文档是个重要问题。这是为什么敏捷项目合同经常根据时间和费用而不是固定价格/固定范围的原因。固定价格/固定范围的合同通过定义客户将得到什么，以什么价格，在什么时间来建立信任。敏捷方法设法与客户紧密合作以创造信任感。因为客户看见开发小组正在开发软件而不是做其它事，他/她相信团队将交付能满足他/她所需要的可运行的软件。

不过，敏捷方法为需求管理提供了一个良好基础。所有需求都被记录在索引卡片(用户故事) 或被维护在产品日志上。区别是需求/用户故事/功能被描述的详细程度: 敏捷实践通常忽略细节，当需求真正在软件开发期间开始实现时才去澄清它们: 他们将收集细节所需工作拖延直到需求要在下一次迭代中实现时才进行。这被称为惰性需求捕获 。

DSDM 提供跟踪变动的可能性。当不得不重复业务研究的某部分时，这些变动可能被记录在一个分开的文件上。在 Scrum 中用产品日志来标记需求所导致的变动。他们能继续提出新需求或记录为什么他们要删除/变动，而不需删除老版本。

4.10 观察和社会分析，头脑风暴

这些技术没有明确地在任何敏捷软件开发方法中提及。但因为它们是需求捕获过程中的基本技术，它们可以与任何敏捷方法一起使用。观察方法在需求捕获中特别有用，因为一名高级“用户”并不总是谈论他的工作过程的最佳人选。这些人有时会忘记重要细节因为工作已经成为例行公事。这些细节可以在观察期间被发现。

4.11 非功能性需求

在敏捷方法中处理非功能性需求没有被很好定义。客户或用户谈论他们希望系统能做什么时通常做不考虑资源、可维护性、可移植性、安全或性能。一些关于用户界面或安全的需求可能在开发过程中得出并仍然集成进去。非功能性需求常常与功能需求相关并难于表达。但多数非功能性需求应该在开发中被了解，因为它们能影响对数据库、编程语言或操作系统的选择。敏捷方法应该更明确地包括对非功能性需求的处理。而且需求工程没有明确定义对决定非功能性需求的处理。

4.12 结论

在所有敏捷过程中，RE 处理阶段都包括捕获、分析、和验证。在不同的方法中可能使用不同技术。因为在 RE 过程中该阶段没有被清楚地分离出来，它们也以其它方法体现出来(如 XP 中的计划游戏也是一种捕获和分析技术) 。它们在每次迭代中都被重复，这使得在阶段中区分它和将它与 RE 处理阶段进行比较变得更困难。在敏捷开发过程中所使用的技术有时被隐晦地以不同方式描述，实际实现留给开发人员去完成。这是在敏捷方法中强调高度熟练的人才的结果: “优秀”的开发人员将做"正确的事" 。另一方面，传统方法规定什么细节需要完成并提供给每位开发人员更多的指导去做“正确的”事。不幸地是，确定在项目的未来阶段什么是正确的是非常困难的。

由于 RE 管理基于文档，在敏捷软件开发方法中无法很好地体现它。文档是敏捷软件开发的一部份，但与在 RE 中的重视程度不同。因为所有敏捷方法会包括必须的尽量少的文档，为未来的开发编写足够的文档是开发小组的责任。

总之，在关键部分(象涉众介入) 敏捷方法和 RE 追求的目标相似。主要区别是在一个所实施的项目中对所需文档数量的强调。这是由于对需求稳定性的核心假设不同。因而，这两种方法之间的关系更象是威士忌酒和冰块，而不是火和冰。

5 开发目标

5.1 概要

文献调查显示了在敏捷软件开发中 RE 技术如何被使用。其它可能性是在一个被定义的过程中使用部分 RE 技术，将其作为软件开发的一种管理工具。几种 RE 技术可以被集成在这样的过程中。例如，在计算机程序的帮助下，不同类型的模型被开发，并能很容易地被包含。但不同的模型并不总是以相似的方法被使用和使用在所有的敏捷方法中。例如，划分优先级是所有敏捷方法中的一个基本技术，并且总被以同样的方式使用，它为所给任务或需求分出优先顺序。因为划分优先级被认为对发展过程有实质影响，在一个敏捷软件项目中我们决定提供在一个被定义的过程之内划分任务优先级的可能性。如今团队可能是分布式的，划分优先级在分布式环境里也应该是可能的。这意味着，团队不必见面就能讨论优先级划分问题，这可以节省时间。划分优先级特性应该允许每个团队成员为自己划分任务/需求的优先顺序以作为建立“团队”一优先级的依据。最初的团队优先级可以依据团队成员的优先级来计算。使用这个计算出的团队优先级作为起点，整个团队最终达成一致意见得出最后的优先级。

为敏捷软件开发项目的一个现有管理工具是 MILOS ASE 系统，由卡尔加里大学开发。由于它可以通过互联网来访问，它特别适用于分布式的开发过程。

5.2 开发环境

MILOS ASE系统是为支持在分布式环境中的敏捷软件工程实践而开发的。它基于卡尔加里大学和凯泽斯劳滕大学的MILOS[3] 项目。MILOS的目标是支持软件项目的计划、处理、施行、和提供能在计划和施行期间访问知识和经验的机制。在MILOS和M-ASE之间的主要区别是，MILOS是为传统软件工程开发的，因此侧重对输入输出的处理，而 M-ASE 侧重于迭代、任务，以及对它们的计划和管理。M-ASE 的在线演示可从’http://sern.ucalgary.ca/~milos’找到。M-ASE 的主要功能是迭代计划、用户故事管理、进展跟踪、和度量标准的收集（metrics collection）。

一次迭代是一个固定时间周期，在此期间开发人员将为项目工作。在开发人员开始工作之前，迭代必须经过创建和规划。创建迭代时将给它一个用给定日期生成的缺省名称或一个供选择命名。一个关于迭代的概要描述，例如目标或任务类型，估计工作时间需要被描述，以及在迭代中可能会使用的技术列表。工作成果根据上一次迭代的工作时间和速度来估算。迭代成果根据任务成果来计算。各个成员计划花在项目上的时间也被列出(通常开发人员的报酬按天计算，再乘以迭代所需天数)。每次迭代由用户故事和分配到迭代中的其它任务构成。

用户故事(或故事卡片) 是对将在软件产品中实现的功能的描述。各个用户故事应该有一个短的名字来描述其功能和估计的“最初工时”，以小时计。为了跟踪用户故事它应该被分配到一个项目或一次迭代中。更多细节如活动类型或风险可以加到用户故事中，以改进迭代计划和性能。用户故事包含一个特别叙述栏，它是用户故事中最重要的一个字段。这个字段用来记录所有关于功能的有用信息。

在 M-ASE 中使用“计划白板”来管理用户故事。管理内容包括在迭代开始前分配用户故事给团队成员，提供估计完成时间和创建完成用户故事所需的子任务。当一个新用户故事被创建，它自动被分配给项目经理。由于通常会有不同的人在项目中工作，每个人都负责几个任务，在迭代开始前，缺省任务分配可以在白板上更改。白板允许为用户故事增加子任务，这将用户故事拆分为多个步骤。这使开发人员更容易估计和理解用户故事。在迭代开始之前，所有当故事卡片被创建时输入的最初估计可以在白板上更新。设置这些初始估计值最终表明迭代的开始。当迭代已完成但还有未完成的任务时，这些任务可能被移至下一次迭代。

5.3 目标

如章节 4.4 所描述的，划分优先级是需求工程和敏捷软件开发中的一项重要技术。划分优先级在 M-ASE 仍然体现得不好。有时可以判断一项任务重要或是不重要，却无法在迭代中对其划分等级。能在一个分布式团队中进行优先级划分将是 M-ASE 的一个重要功能。

敏捷开发的基本原则是经常交付有价值的软件给客户。但在许多情况下开发人员无法决定什么对客户最有价值，因为客户才是拥有业务知识的人。在 M-ASE 中的划分优先级功能使用户能够在迭代中排列任务优先次序。由于客户和开发人员都可以访问系统，划分优先级将同时受到业务和技术的影响。这将导致所提供的次序不仅对客户是最有价值的，而且是技术分析的结果。团队优先级的结果帮助开发人员在合理时间内为客户实现最有价值的系统，完成所有可实现功能。随着项目继续，那些不太有价值的功能或难于实现的功能将会被标为高优先级，因为所有其它功能已经实现了。

划分优先级功能将允许客户在任何时候取消项目，但仍能得到有价值和实现了部分功能的软件系统。最后的优先级划分允许所有用户讨论任务和它们的优先级。这有助于增加对问题域的理解。

6 设计和实现

6.1 概要

本章描述划分优先级功能是如何集成到 M-ASE 系统中的。首先从用户角度解释用户界面和功能。实现描述部分讨论了更多细节并解释了技术实现。

6.2 功能性

划分优先级功能可通过显示了迭代细节的站点来访问(参见图 11)。该站点包含一个对划分优先级站点的链接和概要表。概要表显示所选迭代的所有任务和它们的团队优先级。表中还包含一个信息字段。这个信息字段显示一则短消息，通知用户是否所有团队成员在本次迭代中都有标记了优先级的任务。

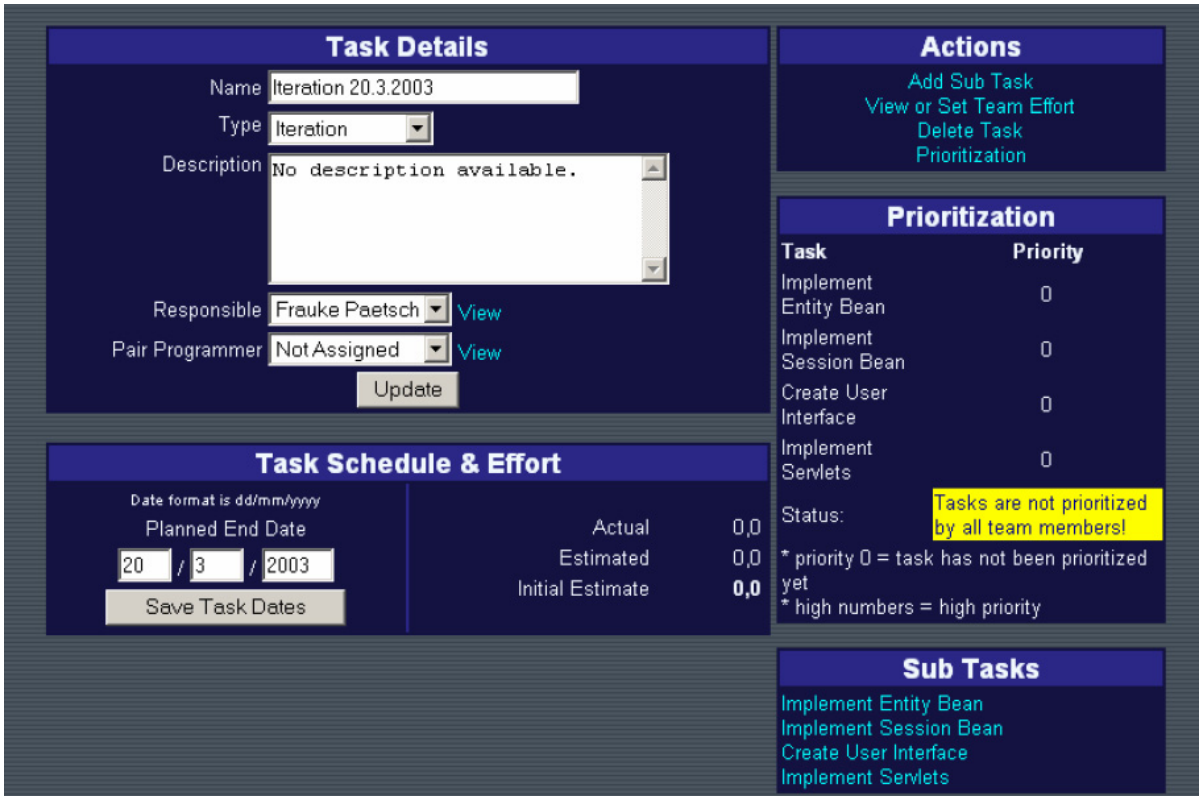


图 11 迭代细节站点

划分优先级站点只有在输入了任务以后才是可访问的。它显示了一张任务和团队优先级的表格以及包含所有团队成员的表格(参见图 12)。所显示的优先级是根据团队成员优先级所计算的团队优先级。在团队一致同意最终的团队优先级之后,最终优先级会在表中显示。如果任务没有被划分优先级或在迭代中增加了新任务，任务优先级为’ 0 ’ 。该表格还包含一个信息字段 用来显示消息。在划分优先级以后，这些消息将通知用户某些特殊事件，例如，如果两项任务具有同样的优先级，或如果最终团队优先级包含相等的值。如果一名团队成员改变优先级或在团队已经对最终优先级达成一致意见之后增加了新任务，项目经理将收到一则显示在信息字段的消息。项目经理将决定是否要重新划分优先级。

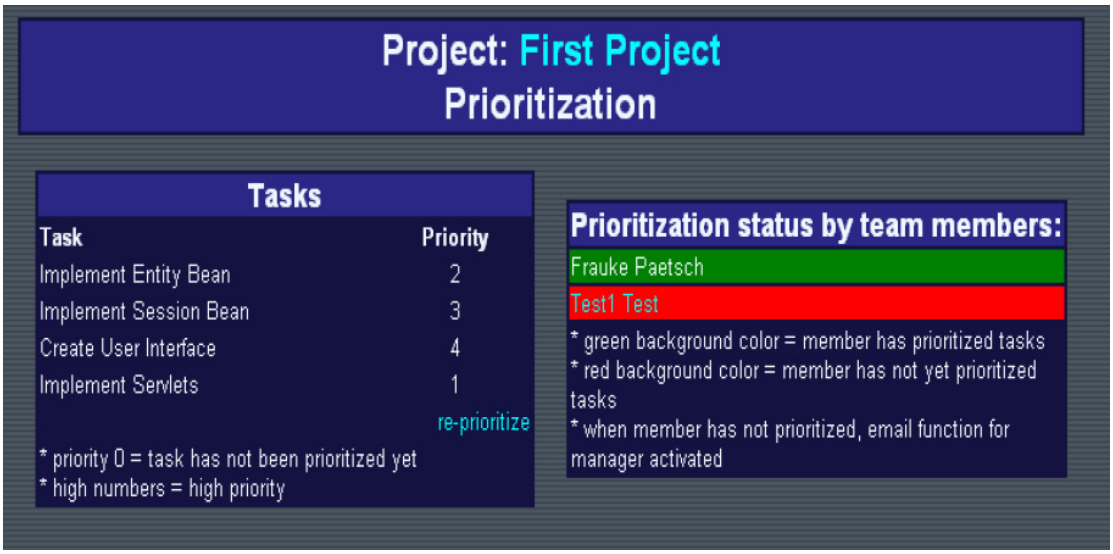


图 12 划分优先级主页

在站点中的第二张表格显示所有团队成员。单元格背景是绿色的，表明这名团队成员已经划分了迭代中任务的优先级，或红色，表明团队成员还未划分迭代中任务的优先级。如果当前用户是团队经理，那些未划分优先级的团队成员的名字会被链接在“提示邮件”中。

划分优先级的第一步是，团队成员先对每次迭代中分配给自己的任务划分优先级（参见图 13）。划分优先级可以在任何时候重复进行，因为如果增加了新任务、或删除任务、或有证据表明应提高或降低一个任务的优先级，更改也许是必要的。这些变动会影响团队优先级，但当最终团队优先级已经被定案，它们不会影响最终团队优先级。对优先级的变动，包括最初的优先级划分，通过划分优先级主页来进行。如果团队成员希望改变优先级，可以通过显示在表中的“重设优先级”连接来进行。

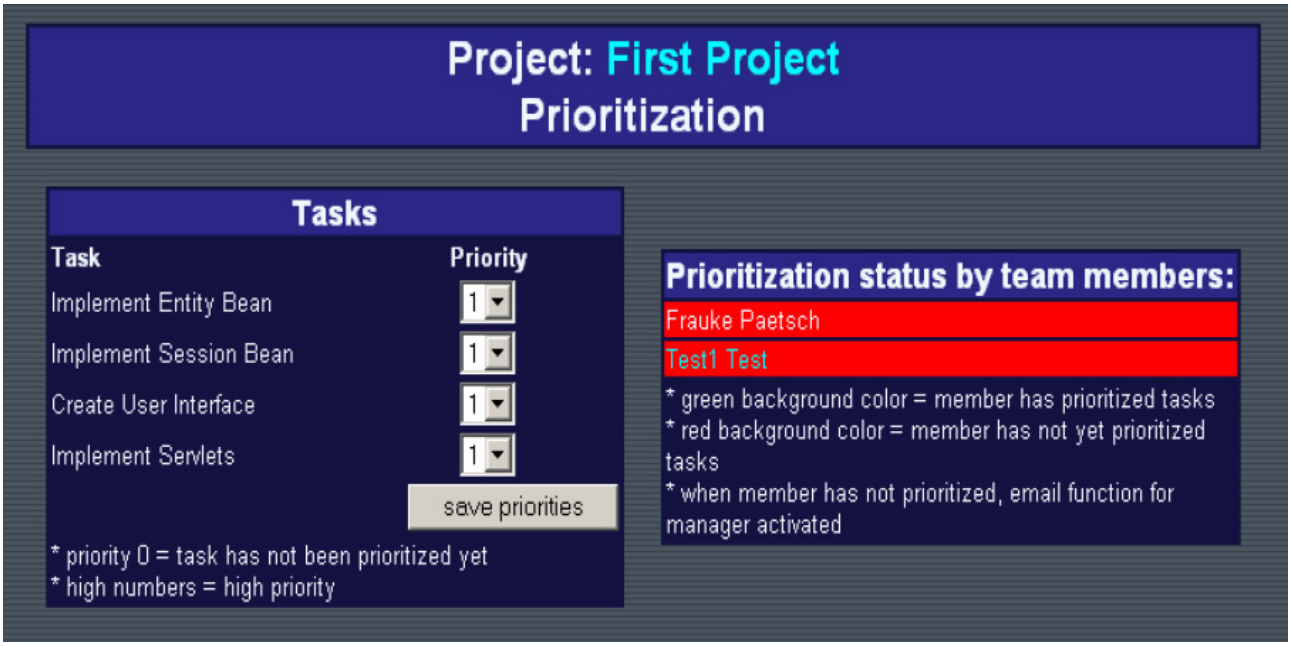


图 13 团队成员的优先级划分

（重新）设置优先级站点和前一站点显示同样的两张表，但用下拉列表来显示优先级。如果还没开始划分优先级，所有任务的优先级将被设为’1’。如果已完成优先级设置，所显示的优先级是团队优先级。下拉列表中的数字从 1 开始到迭代中的任务总数。数字越大，优先级越高。用户为每个任务选择一个优先级，然后点击“优先级”按钮来保存数据。

最终等级应该在团队会议（例如在NetMeeting 会议期间)上完成,这样所有团队成员能讨论优先级是否合理。等级在计划白板（参见图 14）上列出。只有任务按优先顺序排列，等级才能被划分。任务按团队优先级的顺序显示。每项任务可以使用显示在任务优先级旁边的箭头上移一格或下移一格，或置顶/底。当任务的优先级被改变，站点将被重新加载并显示新的优先级顺序。如果所显示的任务按字母顺序排列，将只显示优先级，不显示用来移动任务的箭头。当团队一致同意最终的等级后，项目经理将通过选择迭代旁边的复选框来保存数据，此时点击“确认优先级”按钮，优先级被最终确认。只有项目经理能执行该操作。

图 14 计划白板 （planning whiteboard）

6.3 实现

第一步是实现一个 session bean 和一个 entity bean 提供对数据库的存取，计算团队优先级的算法和其它用于白板或优先级划分站点的函数（参见在表 15 中的类图）。 entity bean 提供对数据库的访问，session bean 提供事务逻辑。由 session bean 提供的功能有：

检查数据是否在数据库中，如果不在，则向数据库表中增加新记录

更新现有记录.

检查是否有相同优先级的任务

计算团队优先级

更新团队优先级

改变优先级(当任务在白板上上移或下移时)

保存最终团队优先级

团队优先级根据每个团队成员的优先级来计算。对每项任务，所有团队成员的优先级都要被总结。结果团队优先级是被总结的优先级的新顺序，用最低值表明最低优先级(参见列表 2)。当每次团队成员重新设定任务优先级时团队优先级会更新。只要团队优先级未被定案，最终团队优先级就可以被更新。概要表和白板总显示最后的团队优先级。

当任务在白板上上下移动时，最后的团队优先级会改变。那意味着所有已登录的团队成员能在重新加载网页后看到变动。移动任务功能使用 Java 的 LinkedList 来改变条目顺序(参见列表 1) 。所有任务按优先级有序存贮在 LinkedList 中。被移动任务的索引被保存在一个变量中，而任务被存放在 LinkedList 中。当所有任务都被保存后,从列表中删除被移动的任务然后将它放在新位置上。然后根据它们在 LinkedList 中的新次序对所有任务重新设置优先级。

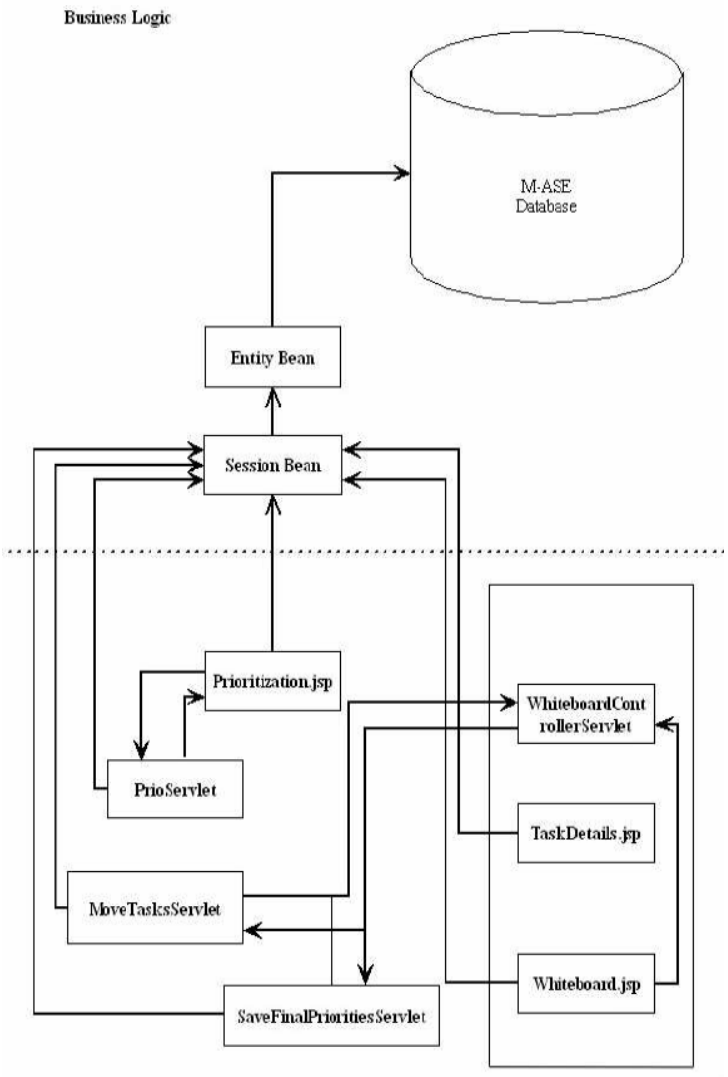


图 15 类图

当最后的团队优先级被保存后,数据库(被完成)中的一个布尔字段会被设置为“真”，表明最后的团队优先级不会再有改动。如果团队成员为任务重新划分优先级，数据库(改变后)中的另一布尔字段将被设置为“真”，项目经理必须决定是否重新修改最后的优先级。如果他决定修改优先级，布尔字段“被完成”和布尔字段“被修改”将被设置为“假”并且最后的优先级可能被重新划分。如果项目经理决定不修改优先级，只将“被改变”字段设置为“假”。每次加载 jsp 站点时这些布尔字段将被检查，站点将根据这些检查结果显示。

增加了一个额外的 jsp 站点并且必须修改两个 jsp 站点。新 jsp 站点是划分优先级站点。站点包含两张表，一张显示任务和优先级，另一张显示团队成员。任务和团队成员按存放在数据库里的顺序来显示。显示在任务表中的消息和站点外观取决于数据库中的布尔字段和 URL 参数。如果任务未被设置优先级，布尔字段“已划分优先级”被设置为“假”。当站点被加载时会检查该字段并根据其状态(真/假)来决定是否显示概要表或包含下拉列表的表。当 URL 参数“重新划分优先级”被设置为“真”时，包含下拉列表的表也会被显示。如果团队成员两次选择同一优先级或计算出的团队优先级包含相同优先级，一个 URL 参数会被设置并且显示一则对应消息。URL 参数将由 PrioServlet 设置。当用户保存优先级时，数据被发送到 PrioServlet，以检查是否有重复优先级并计算团队优先级。根据该函数的返回值，PrioServlet 重定向到不同的 URL。

在 jsp 站点显示了迭代的细节，增加了一张显示迭代中的所有任务和它们的优先级的表。由于 jsp 站点也被用来显示任务细节，它必须检查当前的过程是任务还是迭代。当从数据库中读出表数据时，它同时检查是否所有团队成员和任务在优先级表中(参见列表 3)。

在白板站点，上下移动任务的功能在团队一致同意最后的团队优先级以后被自动增加。该功能使用 java-script 函数来实现，它将迭代 id、任务 id 和一个参数和函数名转发给 WhiteboardControllerServlet。根据函数名，controller servlet 转发数据给不同的更具体的 servlet 。增加了两个 servlet 来处理移动任务的功能和保存最后的团队优先级。两个 servlet 都使用了 session bean 中的函数并重定向回白板站点(参见列表 4)。它们不包括业务逻辑并只用来转发数据。

此外对于用户界面和 Java bean，一个供 session bean 使用的新测试类被包含在 M-ASE 系统中。测试类检查 session bean 的基本函数，例如向数据库中增加任务和删除任务 (参见列表 5)。如果测试类无法增加或删除任务，它会抛出异常，以便运行测试的开发人员能检查哪里出了问题。

7 总结

对最常见的需求工程技术的调查和它们的描述为研究不同的敏捷软件开发方法建立了基础。对敏捷方法的研究主要关于哪些 RE 技术已经被使用并且哪些能额外被使用。在需求工程和敏捷软件开发方法之间的主要区别是文档的数量和两种技术中的基本思想。在需求工程中需求被认为是稳定的，因此可以在软件开发之前文档化。在敏捷软件开发中需求在整个开发过程中不断变化，这导致要求不断修改需求文档，因此文档数量被尽可能地减少。

这两种方法都基于客户介入和对需求进行优先级划分。使用划分优先级能帮助开发人员为客户首先实现最有价值的功能。由于这个技术提供很多好处，一个称为 M-ASE 的网上管理系统被开发，为敏捷软件开发提供该功能。该功能允许团队成员为每次迭代中的任务划分优先级。团队优先级基于个人优先级的平均值。团队最终应就优先级划分问题达成一致意见。

因为团队优先级基于个人优先级的平均值，M-ASE 的未来目标可能包括提供新的计算方法和使用譬如 AHP 的数学算法。这可能要求更多的用户输入，但使决定等级更加容易。对划分优先级功能的一个可能扩展是“讨论板”，团队成员能用它来评论任务并给出选择优先级的原因。如果团队成员不能通过 NetMeeting 或类似的工具沟通，这可能是有用的。

征稿

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

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

一个生物化学领域的面向对象案例研究

Jacqueline Signorini、Patrick Greussay 著，曹想华 译

摘要

我们提出的这个案例研究的对象已为人所熟知却又非常复杂，并且本质上它是网状生物计算的系统-凝血级联系统-我们采用的是来自软件上被称作面向对象设计方法。这份说明书包含了对类和方法的定义与继承，以及使用最广泛应用的OOD语言：统一建模语言UML和它的扩展-实时UML。

首先，我们强调对于统一方法论必须指定足够复杂的生物和生化过程的必要性。然后，以凝血级联系统为例，我们定义类图来展示酶原-酶之间转化的促凝因子的静态结构。最后，我们给出一个包含事件，通信，同步和次序的动态模型。

最终，我们可以证明OOD可以用在许多领域，而不仅仅是软件设计上。还可以总结出统一且可共享的描述方式的好处。当然，作为一个附加结果，我们自动产生了一套模拟软件。

关键词

面向对象设计，UML规范，凝血级联系统，蛋白酶解网

1. 简介

针对非软件领域复杂系统的 OOD

OOP现已是主要的软件设计范型。为了搞清楚且设计出一些复杂的程序，OOP要求建立在描述性语言的新方法。在程序编写之前，这些语言为系统的状态以及过程产生了一个完整详细的描述，而这些状态和过程构成了整个计划的任务。使得描述性语言(UML，实时UML)成功的另一个原因就是：它使得各个独立的松散合作的团队间能够共享对一个原型的描述。

这已经是个标准的结论:是OOP产生了一大堆面向对象的设计方法。而这些方法已经成为公认的技术方法上的工艺。

这些方法形成了一些规范,而这些规范独立于目标程序,程序是随后参照OOD的描述而手工编码或由计算机产生的。

现在该考虑这个问题:OOD的能力不仅仅体现在程序设计上,也为众多的复杂领域所需要。它们到现在仍依靠:

1. 统一而稳定的描述方法;

2. 使用诸如封装,类共享,嵌入,静态属性,多态这些概念的方法

3. 可以共享并重新整合由不同研究团队建立的描述的可能性

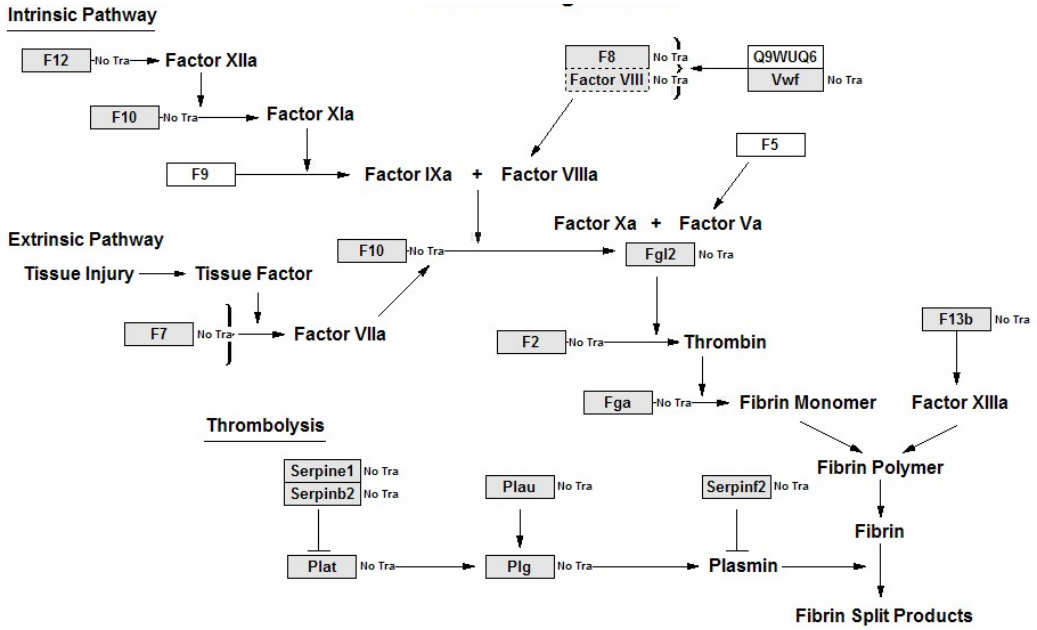
4. 一种能力:对于特定的情景,能自动生成与之对应的模拟软件的代码。

我们因此提议使用生物计算界的一个典型例子-凝血级联系统中的系列过程-一个有望证明统一描述方法的必要性的案例研究。

为了这个目的,我们的工具是当今OOD中使用最广泛的标准化语言:统一建模语言(UML)。

什么是凝血级联系统?

凝血和随后的受伤组织的修复接着凝块的分解的过程被称作止血。它建立在一系列酶原和酶之间的转化之上,也被称作蛋白酶解迭代[5]。解蛋白酶是一种蛋白质,它能把别的蛋白质分解成更小的部分。由于它们自身的极度危险,它们通常在血浆中作为酶原形成和传输,处于非活性形式的酶通过活化作用经过蛋白酶解来从先驱分子中释放活性因子。凝结的经过作为活化过程中调节一系列积极因素与消极因素相互反馈。它能确保交错的蛋白纤维凝块的形成以便堵住受伤的血管,通过凝血酶的作用来防止血液的流失。凝块中所有的血浆蛋白质基本都是由肝脏产生,它们是可以用罗马数字表示的凝结因子。这些数字只反应它们被发现的先后顺序,而不表示它们在凝血过程中作用的先后顺序。凝血素因子XII,XI,X,IX,VII是在凝血过程中分别被转化成活性酶的酶原。相反,因子V和VIII都是反因子。当一个这样的酶被激化时,他的编号就要相应加10。



为什么要使用UML来设计这样一个系统呢?首先,它提供了一种结构分析和事实上很适合该信息系统的面向对象方法,因为系统中的对象都是很强联系的特殊实体。第二,蛋白酶解的迭代过程直接表现为酶原-酶之间的转化,实际上可以被映射为一个状态图,在图中,对象的行为都会被标识:进入系统,退出系统,存在于某个状态中或该状态下的可能发生的事件中。第三,通过活动图和序列图定时工具被用作同步条,支路,连接点或合并点,以便通过消息传递和方法触发来勾画出凝结机制。

2. OOD 凝血系统:类

在我们的模型中,定义四个类来适应指定的止血阶段。每个类都是对象的集合,而对象(相应的血液因子,负载蛋白,反因子)都是类的实例。我们把类描绘成一个三段的盒子,顶部包含类名,中部包含类的属性列表而底部包含操作或方法列表。通过定义类间或对象间的关系使得类之间能够互相协作并交换信息。

有五种类型的关系:关联,聚合,组合,泛化,依赖。这四个类设计了复杂的凝血过程(图1),共用了三种关系。接触阶段的类,相应于凝块形成的初期(受损的血管壁中血浆形成胶原质),因为其对内源通路的类有着逻辑上和物理上的依赖,所以使用一个聚合的连线把它们联系起来。这个连线在关系对象的一方有菱形箭头。

接下来的两个类,内源通路类和外源通路类使用紧闭的箭头来标识一种与公共通路子类之间的泛化或者继承的关系。在UML中,泛化意味着两件事情:继承和子类化。继承意味着一个子类拥有其父类的所有属性,方法,相关性和依赖性。子类可以扩展或者特化它继承来的属性。通过特化,子类可以多态地重新定义一个操作(或状态图)使得它们更加适合这个类,甚至对于一个相关的类它可以替代这个继承来的行为[11]。

连接类型的定义反应了公共过程中产生凝血酶的事实。首先,通过积极的反馈,无论内源通路或者外源通路,凝血酶迅速扩大了一些促凝血因子的止血活性:因子XI,VIII,V,XIII,X,IX,I(纤维蛋白原)。然后在消极反馈的层面上,凝血酶突变成凝血酶抑制素(反凝血酶III,肝磷脂反因子II,a2-巨球蛋白,a1-抗胰蛋白酵素)从而激活血浆蛋白质,而这种蛋白质可以抑制并使凝血因子退化。这种内在的规则限制了凝血层级的发展。

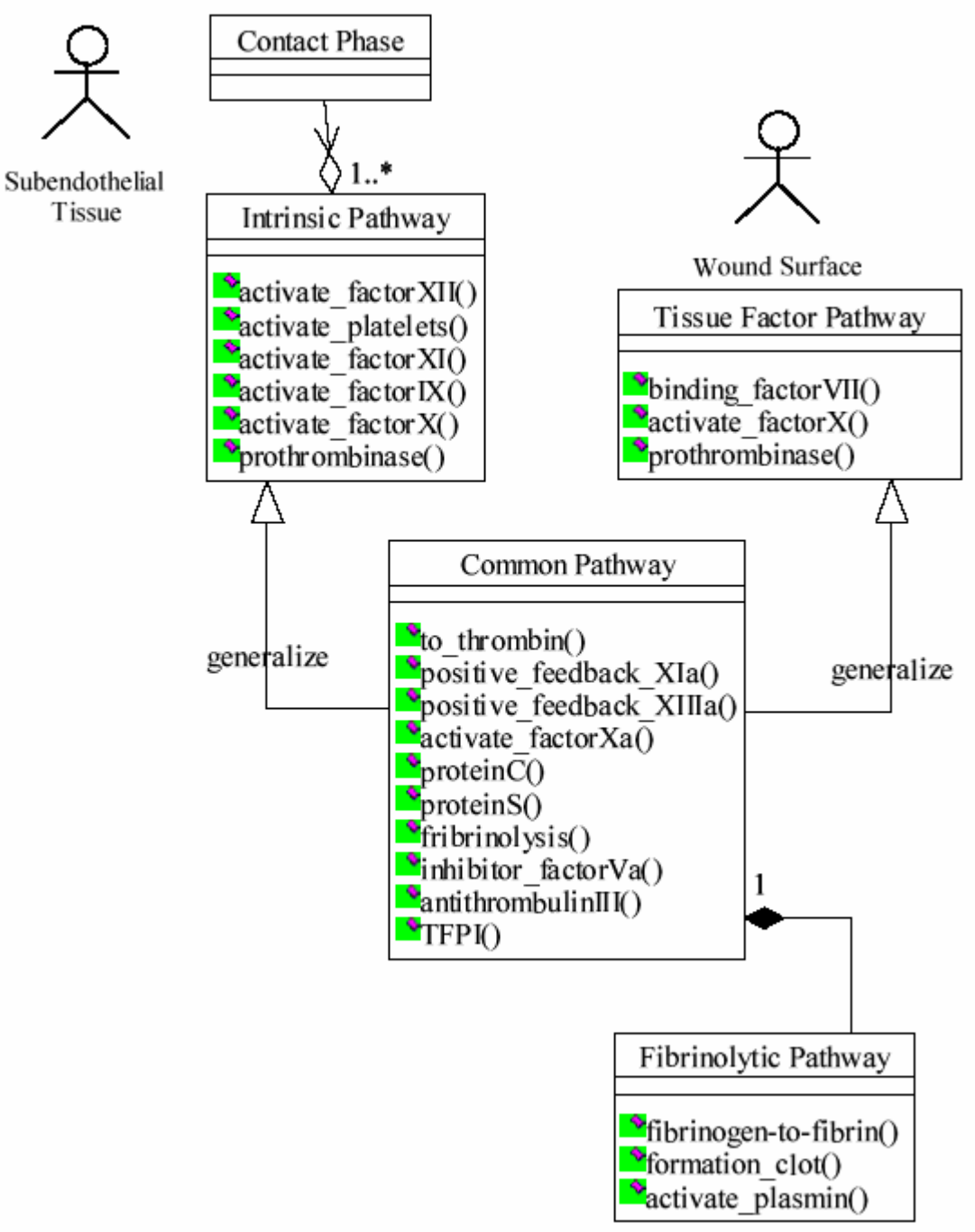


图1 凝血过程中的四个类

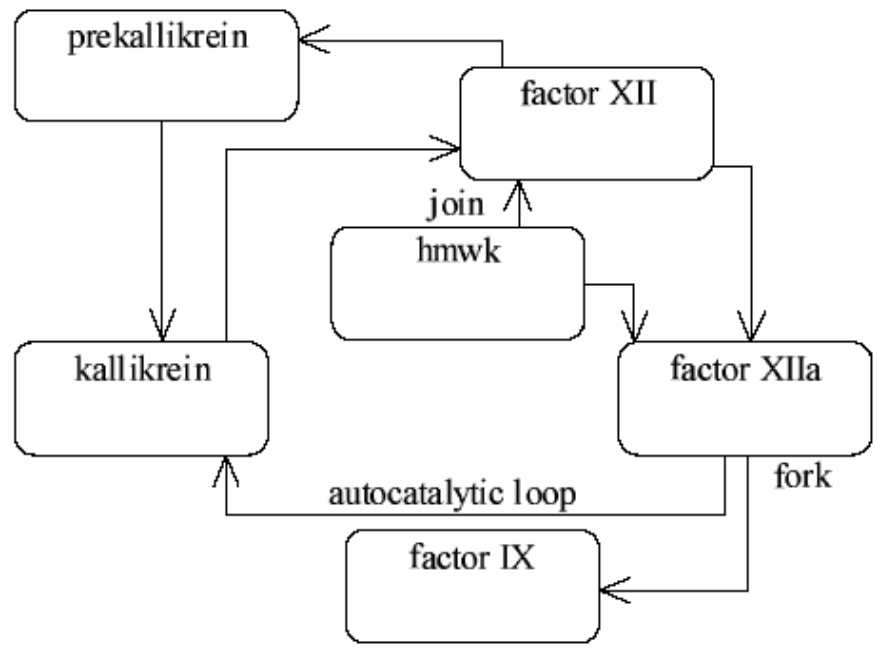


图2a. 接触阶段状态图

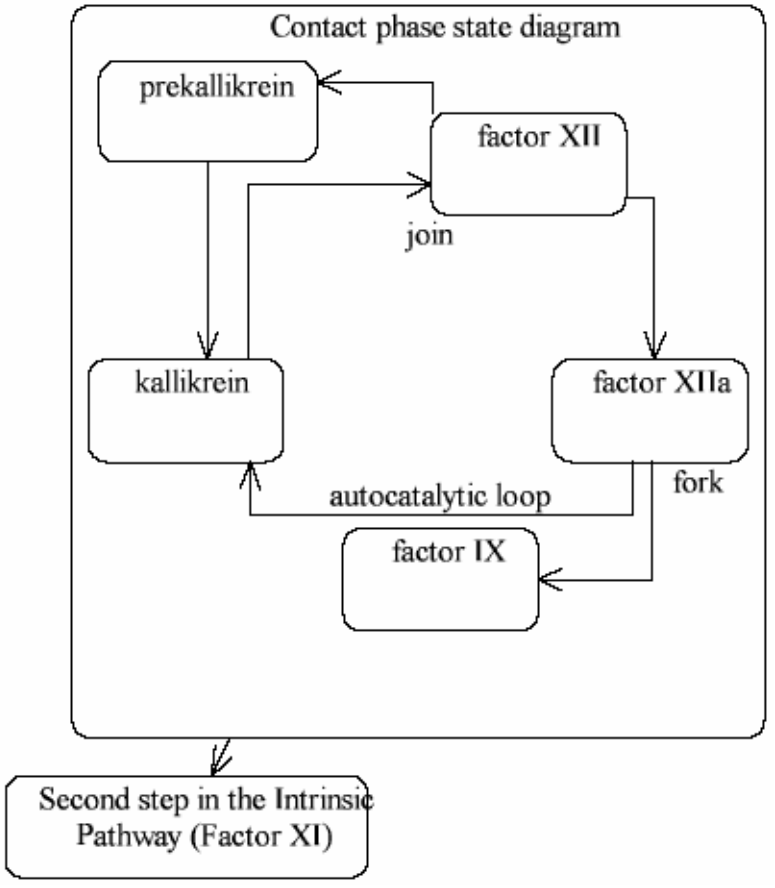


图2b. 嵌套状态

第三种关系发生在纤维素溶解酶通路类和公共通路类之间,它们是一种组合关系,具有很强的聚合性而相互适应。纤维素溶解酶通路类作为一个组件来构成公共通路类这个组合类。所有组件都包含在组合类中并且仅被包含者共享。这种关系表示为在组合类的一端有一个实心的菱形。实际上,在凝血的过程中,纤维素溶解酶过程强烈依赖于凝血酶的激活,就是公共通路类的第一个步骤,它完成纤维素的溶解酶的蛋白酶解过程[10,12]。

凝血酶生成了许多纤维蛋白单体,这些单体随后通过因子XIII的激活聚合而成为纤维蛋白带和非溶解性纤维蛋白,凝血过程的最后一步形成了交错的纤维蛋白聚合体[4,10]。

出现在靠近的两张图中的两种角色被画在系统之外,它们勾画出与系统的静态设计之间的交互接口。关系线末端的数字表示分别参与这种关联的对象个数。如果是"*"那就意味着多个对象的集合,称作多重角色。而如果聚合体的一端是"1"表示单独负责创建它的组成组件。

3. 接触阶段状态图

对象和对象改变的识别和对象分析相关,分析可以分离出一组对象和它们之间的关系。逻辑对象的动态行为的说明来自于对象动态行为的状态图表或状态图设计。图2的状态图提供了接触阶段模型的有限状态机。

直接转换的有六个状态。当一个事件发生时-如消极的血浆生膜的暴露-对象一旦接受了这个事件或因该事件而采取一种措施时就转换成另外一个对象或另外一个状态。不久就进入了这个新的状态,一个隐式的定时器调节着转换退出的时间或通过自己的触发的方式来加以放大过程。转换本身可以用参数,触发器,布尔表达式或动作列表表示。最引人注目的UML状态图特征是状态与状态的嵌套。在图2b中显示的,第一阶段血小板聚合的接触阶段状态图就嵌套在内源通路状态图中。

状态图是整个系统的所有状态的动态视图。在本文的剩余部分,我们描述一下血凝机制的三种模型,每一个模型都对应一种状态视图并作为它的完整动态行为设计。

4. 血凝协作图

第一个模型叫做协作图,它描绘了参与给定的一组消息的所有对象的结构组成形式。在这种图中,它显示了暗含的凝血过程的结构上下文,标示出了先前定义的四个类之间的关系。先前的分析中,结构上下文是通过用例图表示的。用例图只是粗略表现角色和系统之间的对象协作关系。然后,分析捕捉到设计的细节并把系统分解成一个个对象,增加了新的理解来提炼用例场景。图3构造了一组对象协同工作的血凝形成场景。它展现了一个有序的血凝

系统行为过程。

在这个场景中，外源通路（Extrinsic Pathway）的层级系统中（右边）画了一条转化成因子Xa的可选路径。当血液呈现在伤口处,组织因子(因子III)被释放。由于钙和磷脂的存在,组织因子和因子VII形成了复杂的TF-VIIIA物质,这种物质使得因子X和IX得以激活。因子Xa和反因子V在凝血素的作用下发生反应生成凝血酶。然后,剩下的过程就和内部过程很相似了。TF-VIIIA化合物因为组织因子抑制素的作用而很快停止活动。循环过程迂回圈从凝血酶的定义处开始,用自下而上的箭头表示凝血过程的两大功能机制:1.限制纤维蛋白的数量从而阻止肌肉萎缩症的发生；2.血凝过程局部化并阻止广泛的血栓症发生。

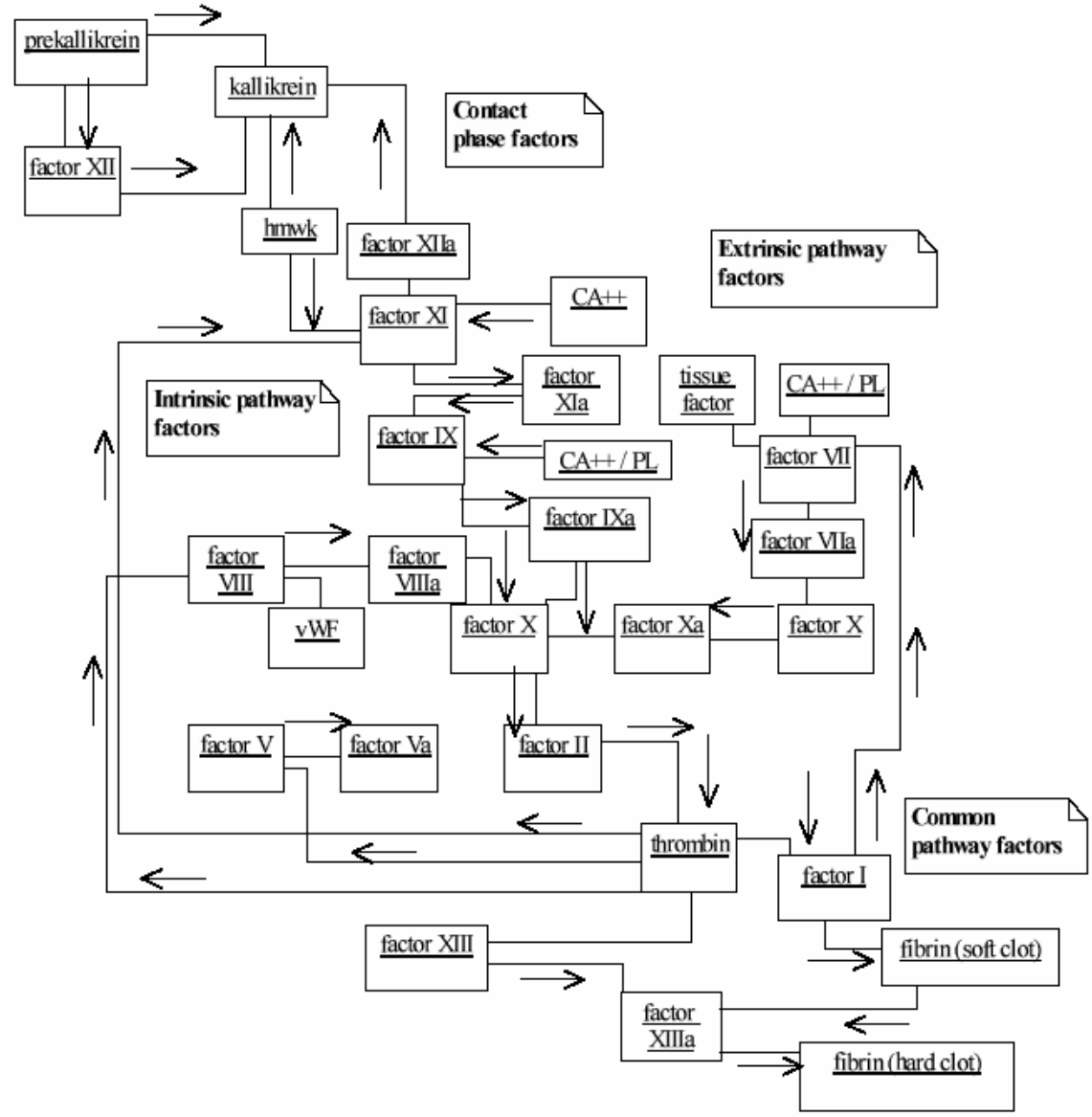


图3 血凝协作图

5. 外源通路活动图

下面的图叫做活动图，它给出了对外源通路的比前些章节（图4）阐述更加精炼的理解。

它展示了对象间活动的流程, 这些对象包括转换, 分支, 合并, 产生和联合。可以用一种算法或者详细的计算来表示。

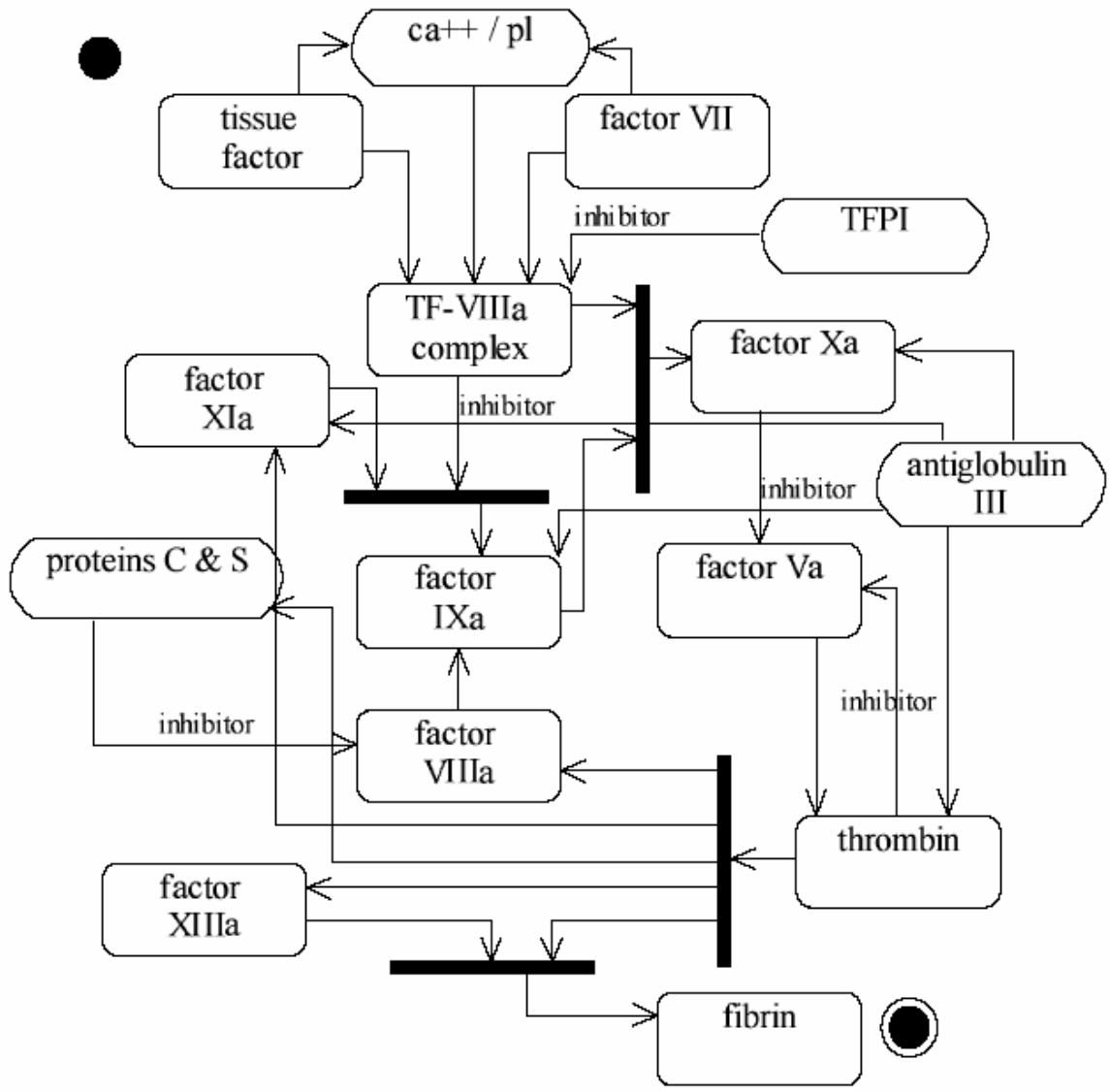


图4 外源通路活动图

在图4中,四个同步条用黑长方形刻画。动作在相同的顺序下执行且状态的转换需要同时发生。实际上,一些连续的状态在传递的事件中要求同步。这个状态下,对于凝血酶来说,产生条活动了,它通过连续状态发送激励信号,同时,从不同地方出发的联合条中止了它的活动并阻止流血的紊乱。例如血友病,就是一种因为缺乏因子VIII的X-linked类型混乱因素,这种因子必须在凝血酶和因子Xa发生蛋白酶解的时候激活。

6. 凝血过程序列图

协作图强调对象间的关系。序列图强调它们之间消息传递的序列图的顶端,矩形给出了对象的标识。为把它们和类区别开,名字下面带有下划线。垂直的虚线是生命线:图的时间轴是笔直向下延伸的。生命线之间的箭头表示对象间传递的消息。白色的矩形被称作活动期,它表示为响应一个消息而执行的方法的时间段。

图5显示了我们对对象间传递消息和执行方法的顺序的理解。可以看出,VII-TF复合物对于外源通路来说是反因子,V-X复合物对于公共过程来说是反因子,VIII-IX对于内源通路是反因子。

图的上面部分展示了促凝血因子的激活。开始在VII-TF复合物中激活执行factorX()方法,消息传递触发新的方法,在这点上,对象激活状态的改变通过积极反馈循环加速了凝血过程。下面部分主要靠凝血抑制消息的驱动限制促凝血因子的发展相反却激活血纤维蛋白酶。

很明显,细节的时间分析应该把时间的权值添加到图中作为限制条件或者同步模式。但是,那些时间值只能通过体外的凝血过程来获得,而它们本身就是理解模型的基础。

7. 总结

面向对象设计对于计划和原型都很复杂的软件来说是很有价值的技术,但也同样适合建构概念上的域模型。在这篇论文中,选择一个非常复杂的生物计算过程作为案例研究-血凝级联系统-我们证明了面向对象技术在专业领域的有效力量,从该领域建设性的角度出发,通过描述性的图表进行研究。从OOD中我们已经使用了标准的UML核心技术。每一种图都提供了对血凝过程的独特描述。

类图给出了一个对凝血因子的结构组织和它们的关系。状态图和协作图构建了该过程的有限状态机。最后,通过活动图和序列图,我们描述有时间顺序的酶原-酶之间的激活以及由此可能带来的有条件的或平行的行为。商业版本的UML都为人熟知,例如Rational Rose或是它的自由软件替代品Poseidon,能够产生Java代码,和UML图严格相关。

在这种意义上,我们能够为上述描述的过程自动产生Java模拟器。当然,并非如此:这样的产生器会提供某个Java版本的类描述,继承的机构,本地状态变量,也不能给出方法定义的头部:参数和类型。很显然,方法体的算法部分(函数体)还必须由有一定经验的程序员来完成。幸运的是,要求大部分的算法结构都是一些一次性的或连续的对参数的触发,信号处理,或者简单的重写规则。我们将考虑一个更深入的研究,包含指定的子语言到UML中的可能性,这样可以为完整的描述提供一个完整的产生模拟器的功能。用计算的术语讲,这样可以增加UML宏扩展的能力。

现在,如果一个设计问题发生在生化网上,它的自动检测,不要提及它的自动纠错,是一项令人振奋又很难的人工智能问题。一旦一个生化网的标准描述被建立,这样的问题就可以让研究员同生物学家一样能够研究了。

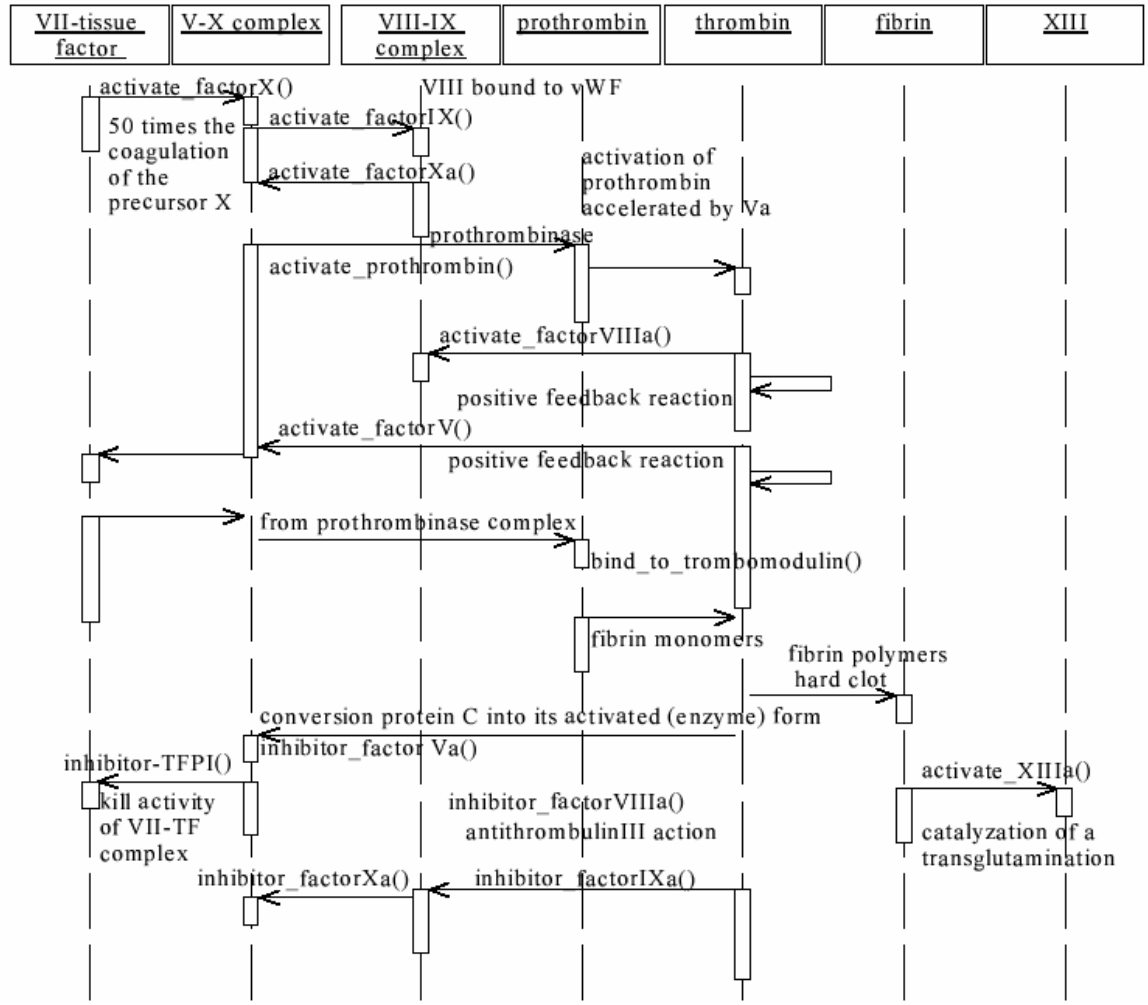


图5 血凝序列图

怎么样比较我们当前的方法所建立的通用通路图和生物学家已经在使用的通路图呢?一个软件工程上很明显的答案就是这么一个事实:一旦我们拥有了一些UML规范可用,那将相当自然而然的想到把它们合并到一个统一描述中,这样可以更加接近标准的生化图。一个不大明显的答案是考虑类的层次作为所描述的生化过程以及演化过程的内在性质,并且不仅仅是一份制造规格的装置而已了。当然,这项很诱人又投机的话题已经超出本文的范围了。

参考文献

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

[2] Broze, G.J., Tissue factor pathway inhibitor and the revised theory of coagulation, Annual Review of Medicine, 46:103-122, 1995.

[3] Broze, G.J. and Tollefsen, D.M., Regulation of blood coagulation by protease inhibitors, In The Molecular Basis of Blood Diseases, 3rd Edition, 1W.B. Saunders Co., Philadelphia, 657-679,22 Signorini and Greussay 2001.

[4] Chambers, R.C. and Laurent, G.J., Coagulation cascade proteases and tissue brosis, Biochemical Society Transactions, 30(2):194-200, 2002.

[5] Davie, E.W., Fujikawa, K., and Kisiel, W., The coagulation cascade, Biochemistry, 30(43):10363-10370, 1991.

[6] Douglass, B.P., Real-Time UML, Developping E_cient Objects for Embedded Systems, Addison-Wesley, 2000.

[7] Fowler, M. and Scott, K., UML Distilled, a Brief Guide to the Standard Object Modeling Language, Addison-Wesley, 2000.

[8] Gilles, J-G., Anti-factor VIII antibodies of hemophiliac patients are frequently directed towards nonfunctional determinants and do not exhibit isotypic restriction, Blood, 82(8):2453{2461, 1993.

[9] Halkier, T., Mechanisms in Blood Coagulation, Fibrinolysis, and the Complement System, Cambridge University Press, Cambridge (England), 1991.

[10] Kimball, S.D., Thrombin active site inhibitors, Current Pharmaceutical Design, 1:441-468, 1995.

[11] Liskov, B., Data abstraction and hierarchy, SIGPLAN Notices, 23(5), 1988.

[12] Wang, W., Boa, M.B., Bajzar, L.,Walker, J.B., and Nesheim, M.E., A study of the mechanism of inhibition of brinolysis by activated thrombin-activable brinolysis inhibitor, Journal of Biology Chemistry, 273:27176-27181, 1998.

"*About Face 2.0* could completely redefine how software applications are created!"
— Pete McBreen, author, Software Craftsmanship

ABOUT FACE 2.0

THE ESSENTIALS OF INTERACTION DESIGN

ALAN COOPER

& ROBERT REIMANN

当木匠看到锤子时，他不想和锤子讨论钉子的问题。他会直接用锤子钉钉子。在车里，如果司机想改变方向，他转动方向盘。司机喜欢通过合适的设备从车子和外部环境直接获得反馈：挡风玻璃外面的视野；仪表板的读数；疾驰而过的风声；轮胎压在道路上的声音；对侧向重力的感觉以及路面传来的的振动。木匠也希望有类似的反馈：钉子下沉的感觉，铁互相击打的声音以及举起锤子的感觉。

International Bestseller
Completely revised and updated

UMLChina 辅助教材。中译本即将由电子工业出版社出版，詹剑峰 译

审稿：国内首个交互设计网站 De Dream'

基于 MDA 的 Web 信息系统开发方法

Paloma Cáceres、Esperanza Marcos、Belén Vela 著，徐异婕 译

讨论

摘要

如今，新技术和新平台推陈出新、日新月异，这为软件产品的开发带来了不小的困难。这种状况也导致了一些问题，例如可移植性、集成和互操作能力等。对象管理组织（Object Management Group，OMG）提出了模型驱动架构（Model Driven Architecture，MDA），通过让不同平台实现相同的模型，来改进应用的可移植性。MDA 定义了一种基于平台独立模型(platform independent model，PIM)和平台相关模型(platform specific model，PSM)的架构。本文介绍 MIDAS（Multi-tier Distributed Application Services Suite，多层分布式应用程序服务包），它是一种基于 MDA 的模型驱动方法论，用于 Web 信息系统的开发；本文采用 XML 和对象－关系技术，将 MDA 元模型应用于 Web 平台；本文介绍的 MIDAS 提出了不同的 PIM 和 PSM，并且定义了这些模型之间的映射规则；为了更好地说明 MIDAS，本文还介绍了案例研究的一部分。

关键词

模型驱动架构、模型驱动方法论、映射规则、Web 信息系统建模

1 介绍

新技术不断出现并流行，例如 Java、HTML、XML、J2EE、和 UML 等等。企业通常使用这些现代技术来开发信息系统，于是，信息系统中出现了一些与互操作性、可移植性、可集成性相关的新问题。

为了帮助计算机用户解决上述集成问题，OMG 开发了 MDA[12,13,15]。MDA 是一个为软件开发而建立的模型驱动的框架，它提出：为了开发出能在不同平台之上运行的软件系统，应该用平台独立模型（PIM）对整个系统进行详细说明。MDA 提出了模型的分类方法、以及模型之间的关系和映射规则，如下所示：

• **PIM**——平台独立模型

PIM是一种高度抽象的模型，独立于任何实现技术。

• **PSM**——平台相关模型

PSM是一种针对特定平台的模型。一个PIM可以被转换为一个或多个PSM，每个PSM针对一个特定的执行技术。

MDA也提供模型间的映射。映射是一组用于模型转换的规则和技术，映射有如下四种：

• **PIM to PIM**

这个转换用于将分析模型映射为设计模型。通常，在开发生命周期中，模型的开发独立于任何平台。

• **PIM to PSM**

当PIM需要被精化（refine）为PSM时，将使用这个转换。

• **PSM to PSM**

这个转换通常用于PSM的精化。

• **PSM to PIM**

这个转换用于根据使用特定技术的PSM来获得PIM。

本文介绍MIDAS [9]，它是一种基于MDA的模型驱动方法论，用于Web信息系统的开发。我们使用XML[1,19,20]和对象－关系技术[3]，将MDA元模型应用于Web平台。MIDAS提出了不同的PIM和PSM，并定义了这些模型之间的映射规则，在下文中我们会讨论。

文章的余下部分按如下结构组织：第2节是MIDAS方法论的总体介绍，第3节介绍一个使用MIDAS技术的WIS案例——马德里电影协会WIS系统——的部分研究。最后，第4节，总结主要结论和未来的工作。

2 MIDAS：一种模型驱动的方法论

MIDAS [9] 是一种基于MDA的模型驱动的方法论，用于Web信息系统的开发。我们使用XML[1,19,20]和对象－关系技术[3]，将MDA元模型应用于Web平台中。该方法论提出了一些PIM、PSM、以及它们之间的映射规则。因为MIDAS要求使用统一的符号来为整个系统建模，所以我们采用UML表达PIM和PSM。不幸的是，用UML来表示整个WIS系统还存在缺陷。但最近一些针对Web建模[2,5]的UML扩展已经出现。MIDAS提出可以使用这些扩展中的内容，例如Web服务建模[6,7]方面的、对象关系数据库设计方面的、以及展示XML Schema [16]或XLink [17]方面的UML扩展。当然，如果必要也可以定义一些新的扩展。

根据文献[14]，WIS建模的要求可以根据三个正交的维度来分类：层次（level）、阶段（phase）和方面（aspect）。第一维由三个层次组成：内容层、超文本层、表现层。第二维由软件生命周期的不同阶段组成，范围包括从分析到执行。第三维由结构和行为两个方面组成。我们在描述MIDAS时，将采用依据方面进行分类的方式：一方面，会考虑到结构模型（structural models）；另一方面，也会顾及到行为模型（behavior models），见图1。

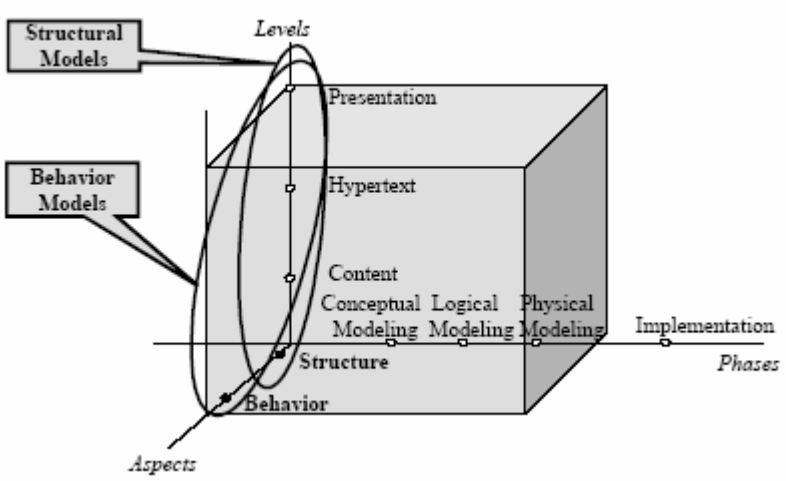


图1 MIDAS的结构维和行为维（根据文献[14]）

MIDAS采用模型驱动架构。它的系统核心是领域和商业模型。在这个核心之外，我们定义了一个环，包含系统的结构维和行为维。核心和环表示了技术和平台的依赖关系模型。外环则是不同的平台和被支持的技术。基于这个架构（见图2），同样也基于MDA，MIDAS定义了一个框架专门为信息系统制定规约，其中区分了“系统功能”规约和“系统功能实现”规约。

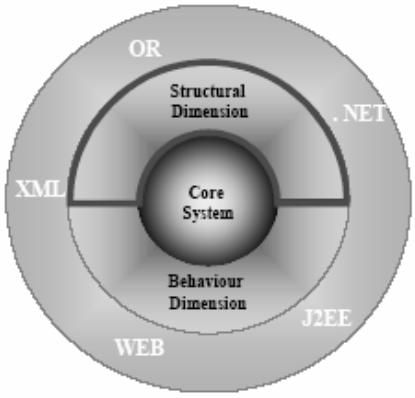


图2 MIDAS架构

MIDAS依据结构维和行为维定义PIM和PSM，并提出了一些不同模型之间的映射规则，即PIM之间、PSM之间、以及PIM和PSM之间的映射规则。

本文介绍一个具体的MDA应用，其中涉及到Web平台、对象－关系技术、以及XML技术。我们将关注图2所示的MIDAS结构维，并描述具体的PIM、PSM、以及它们之间的映射。

2.1 结构维模型

MIDAS的结构维模型被分为PIM和PSM，即平台独立模型和平台相关模型。

- **PIM的结构方面**

如图1所示，在概念建模阶段，所提议的平台独立模型包括内容层、超文本层和展现层。

在内容层，建议使用UML类图来完成数据概念模型。在超文本层，MIDAS建议采用两个技术来实现超文本：片段（slice）和导航（navigation）概念模型。这两项技术是RMM[4]中提出的，RMM在扩展UML中分别使用了片段和导航图。MIDAS建议使用的符号是以“基于UML的Web工程方法（UWE）[5]”为基础的。在展现层，我们建议使用在UWE [5]中定义的展示图来完成描述概念模型。

- **PSM的结构方面**

平台特定模型以对象－关系技术和XML技术为基础，与逻辑建模和物理建模以及实现阶段相关。对象－关系技术常用于描述内容层，而XML技术常用于描述超文本和展现层。

在内容层，根据文献[11]中给出的建议，采用对象－关系模型来完成逻辑数据建模。在超文本层，MIDAS建议实现超文本的逻辑模型（片段和导航模型）。片段（slice）的逻辑描述采用XML Schemas [19]，而片段和导航

逻辑模型采用文献[17]所提及的UML扩展中的XLink [20]描述。至于逻辑模型的展现，我们则使用XSL [21]。

2.2 模型的结构维之间的映射

MIDAS方法论也提供前面描述过的模型之间的映射：

- **从PIM到PIM的映射**

MIDAS提议采用基于RMM[4]的指导方针，从数据概念模型变换为片段和导航模型。这个方法论也提供了一些从数据和超文本概念模型变为展现概念模型的指导方针。

- **从PIM到PSM的映射**

从“PIM的结构方面”到“PSM的结构方面”的映射规则是从概念模型（在内容层、超文本层和展现层）为出发点到逻辑模型的。

我们将依据在[10,11,18]中为对象－关系数据库设计而定义的转换规则，把概念数据模型转换为数据逻辑模型。使用XML、以及用于片段和导航图的“片段 y XLink”的XML Schema技术，将概念超文本模型转换为逻辑超文本模型。依靠在XSL [21]中的样式表（style sheet）定义，概念展现模型直接被映射为逻辑层。

- **从PSM到PSM的映射**

“PSM的结构方面”之间的映射规则支持不同PSM之间的转换，也就是说，在逻辑模型之间、以及逻辑模型和实现模式之间可用相互转换。

在内容层，MIDAS对如何从某具体产品（例如Oracle9i）[11]的SQL中获取数据库执行，提出了一些转换指导方针。在超文本和展现层，MIDAS提供了一些把逻辑模型映射为执行模型的映射规则。这些规则描述了如何通过从在内容层实现的数据库中抽取这些信息，获取用XML表示的最终的动态Web页面。MIDAS也提供了一些指导方针，来定义如何使用象ASP、JSP等技术将数据库和XML页面集成。

3 案例研究

在本节中，我们将展示研究案例的一部分。这是一个马德里电影协会的Web信息系统，电影协会由几条电影院线组成。这个WIS可以在http://kybele.escet.urjc.es/ejemplos/cine_entradas/访问到。它提供协会方方面面的信息，以及在线购买电影票的服务。本文只集中讨论这个WIS的结构维。

之前，我们提到过，MIDAS的结构维模型被分解为PIM和PSM，即平台独立模型和平台特定模型。首先，我们将展示已经获取的概念模型：包括数据、片段和导航模型（PIM的结构方面）。接下来，我们将展示逻辑数据、超文本模型和它们的执行（PSM的结构方面）。

- PIM的结构方面

这个案例研究的平台独立模型，在概念模型阶段，包括内容层、超文本层。我们已经为了简洁省去了展现层。

在内容层，我们建议使用UML类图来获取数据概念模型。我们将仅展示一个已经简化的数据概念模型。从一开始，这个部分的数据概念模型就包括几条电影院线，由一个以上的电影院组成。在每条电影院线中，有一些播放的影片，同时，每部影片会在多个影院播放。一部影片会被播放多个不同的场次，但在同一场次只能播放一部影片。一部影片只有一个制片人，制片人可以生产多部影片。（见图3）

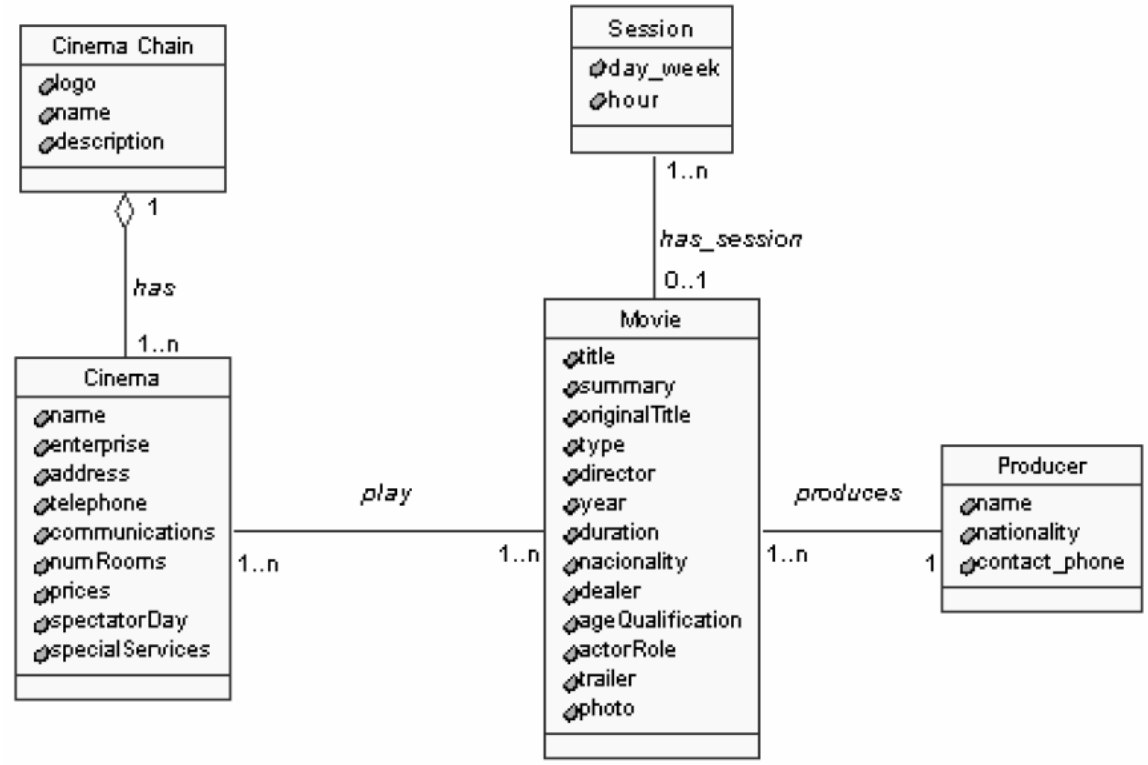


图3 部分数据概念模型(PIM)

在超文本层，MIDAS建议采用两项技术来实现超文本：片段和导航概念模型。从数据概念模型，我们获取了片段和片段图，我们根据UWE的建议采用扩展UML符号来表示，见图4。每个片段用一个导航类来表示，导航类的命名方式和数据概念模型中的类的命名方式相同，这些数据概念模型来源于已经获得的主要信息中。这些信息被呈现在每个片段中，它们与另外一些信息是一致的，这些信息包括：1、即将展示在页面中的信息，2、能被归

属为一个多个数据概念模型的类的信息。也可能会有一些数据概念模型中的类，它们不被作为导航类包含。在我们的数据概念模型中存在一个制片人（Producer）类，它不在片段概念模型中出现。这个类显示的仅有信息是：1、制片人的名字 2、在导航类Movie中包含的属性producer。

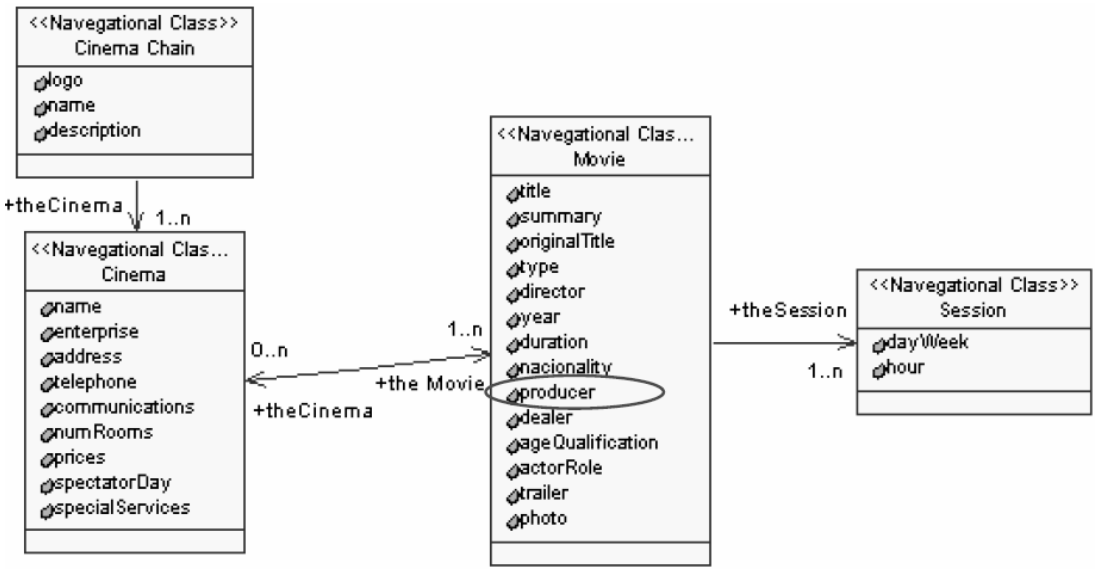


图4 部分片段概念模型(PIM)

在导航概念模型中，访问元素被增加到片段概念模型中。因此在我们的案例研究中，在导航类之间包含了四个导航的索引：院线、影院、影片和场次（见图5）。

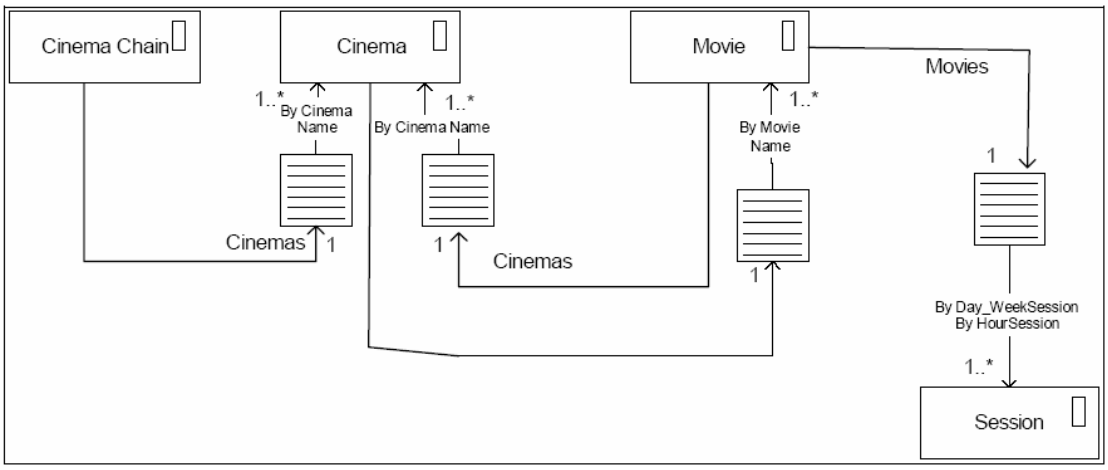


图5 部分导航概念模型(PIM)

- PSM的结构方面

平台特定模型是基于对象－关系技术和XML技术的，这些技术关于逻辑建模和物理建模以及执行阶段。对象

－关系技术多用于表现内容层，而XML技术多用于表现超文本层。

在内容层，与超文本设计平行的，用对象－关系模型来完成逻辑数据模型；我们将用标准的SQL:1999,来完成数据库设计，应用了在[11]中定义的转换规则。

在这个案例研究中，数据库与超文本的设计无关，并且可能有其它的作用。精确概念模型中的每个类（包括院线、影院、影片、制片人以及场次）都被转换为对象类型。根据REF引用或ARRAY引用，将这些类之间的关系转换为与它们相关的类。这种类型(REF或ARRAY)通过关系的多样性来展示这些关系。

因此，我们将描述一个Cinema类和Movie类之间的多对多关系的例子，它引入对象类型Cinema的属性play，对在影院中播放的影片来说，它是一种ARRAY类型的引用。为了描述双向关系，我们引入Movie对象类型，同时属性也是played，对播放影片的对象Cinemas来说，它也是一个ARRAY引用。用<<udt>> stereotyped类来描述用户定义的类型，例如：Type_Price 和 Type_Address。常用于描述数据库设计的图形符号是扩展UML[11]。图6中显示了采用UML描述的标准SQL1999的数据逻辑设计。

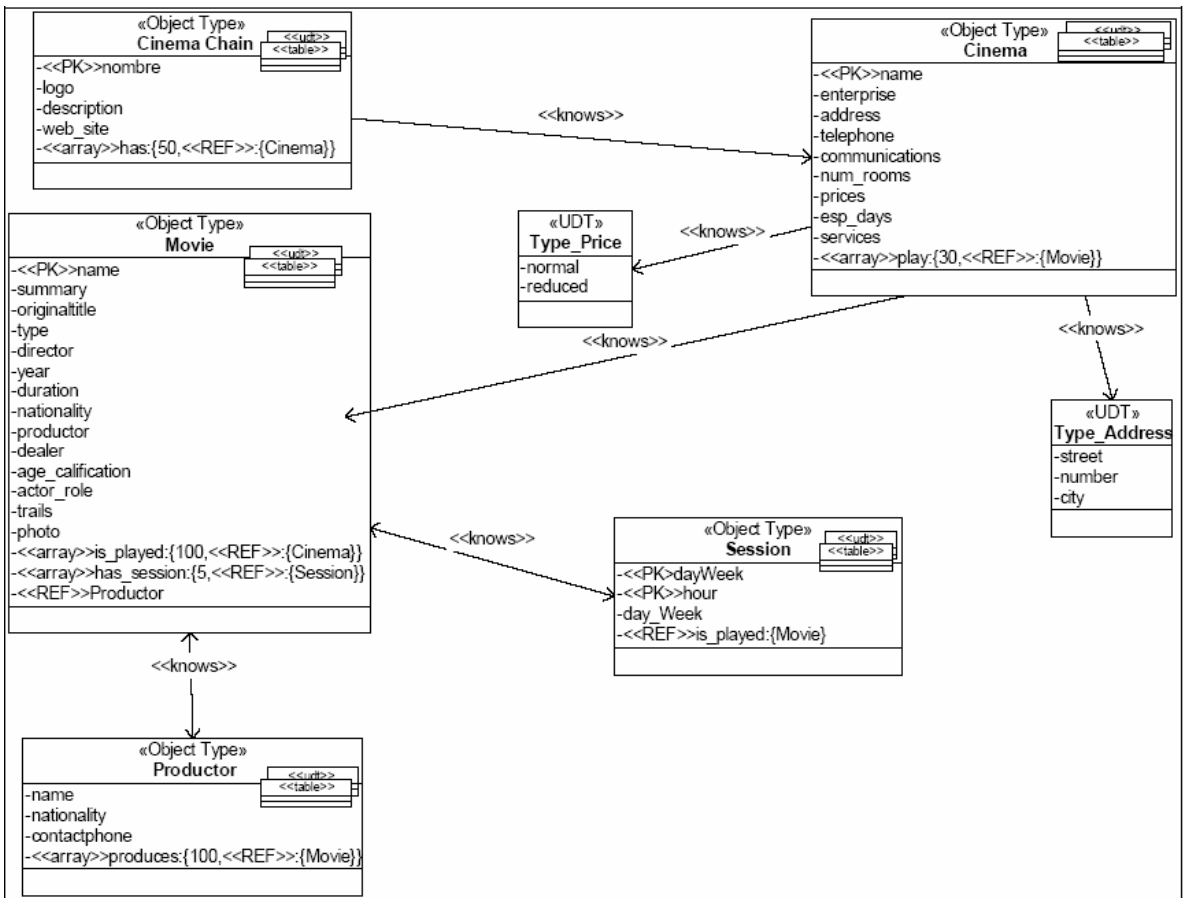


图6 使用SQL:1999逻辑数据库设计(PSM)

在超文本层，我们只能完成逻辑片段和导航模型。为了获取逻辑片段模型，在概念片段模型中，每个获得的片段都被转换为一个XML Schema，这个XML Schema采用文献[16]建议的UML符号。图7中我们介绍的从院线片段中得到的XML Schema UML图与图8中的 XML Schema 代码是相对应的。

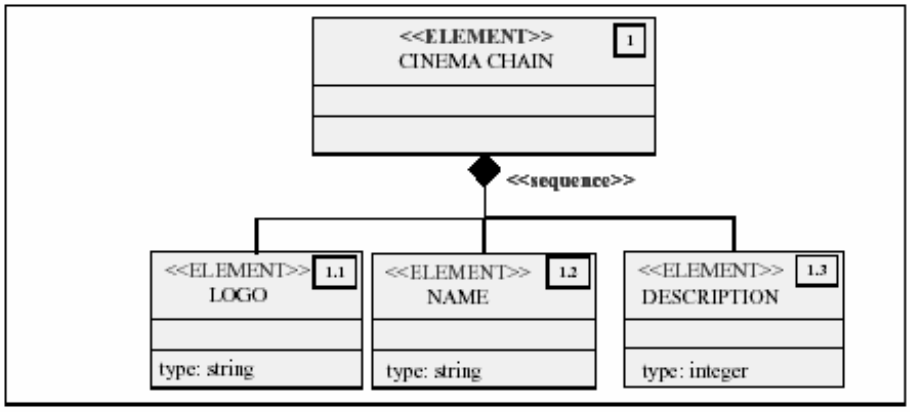


图7 使用UML符号来表达XML Schema对片段的描述(PSM)

```
<?xml version="1.0"?>
<!-- File Name: CinemaChain.xsd -->
<xsd: schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:element name="Presentation">
  <xsd:complexType>
   <xsd:sequence>
        <xsd:element name="LOGO" type="xsd:string"/>
        <xsd:element name="NAME" type="xsd:string"/>
        <xsd:element name="DESCRIPTION" type="xsd:string"/>
   </xsd:sequence>
  </xsd:complexType>
 </xsd:element>
</xsd:schema>
```

图8 XML Schema对片段的描述的相应代码

为每个概念片段模型中的片段获取XML Schema之后，我们必须构建逻辑导航模型。这个模型描述了如何通过页面以及到达页面的访问元素来导航。通过在模型之间运用映射规则，概念导航模型被转换为逻辑层。因此，我们必需按照[17]中定义的规则，将导航概念模型转换为XLink。

在图9中，我们可以看到：在逻辑层，完整的导航概念图被转换为一个<<Extended Link>>，并关联了任意数量的资源。这个扩展关联将用XLink方式来表示，包含一个名为“逻辑导航模型”的属性类。这个类与所有组成逻辑导航模型的片段相关联，它们是依靠组成关联<<is_composed>>完成组合的。

在逻辑层，每个片段概念模型中的片段将被转换为本地资源，通过一个<<XLink Locator>>元素来表现，同时用相关的片段名来命名。在这种方式下，我们得到了一个片段资源，包括院线、影院、影片和场次。导航模型的访问元素，诸如：索引、指导教程、查询以及菜单，也可以在逻辑层被转换成通过<<XLink Locator>>元素来表现的一个资源。

两个片段之间的导航或者片段和访问元素（在此案例中为四个索引）之间的导航都通过一个<<arc>>关联来表示。如何横贯的信息，被包含在详细说明了标签值的arc关联中。相应的XLink代码在图10中显示。

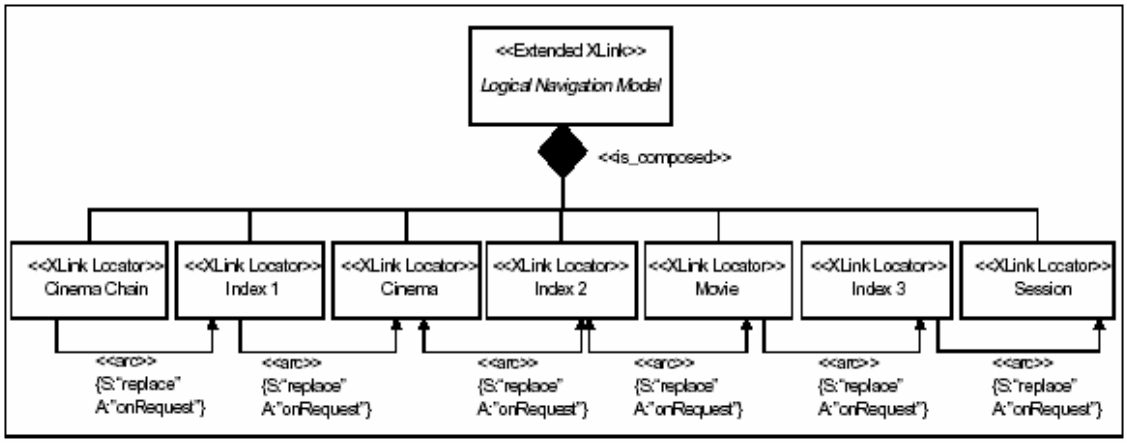


图9 Xlink对逻辑片段模型的描述(PSM)

```
<Links xmlns:xlink=http://www.w3.org/1999/xlink>
<logicalNavigationModel xlink:type="extended"
title="Cinema Consortium"
<!--definition of all local resources-->
<cinemaChain xlink:type "resource"
Xlink:label="sliceCinemaChain"
</cinemaChain>
<index1 xlink:type "resource"
Xlink:label="sliceIndex1"
</index1>
<cinema xlink:type "resource"
Xlink:label="sliceCinema"
</cinema>
<index2 xlink:type "resource"
Xlink:label="sliceIndex2"
</index2>
<film xlink:type "resource"
Xlink:label="sliceFilm"
</film>
<index3 xlink:type "resource"
Xlink:label="sliceIndex3"
</index3>
<session xlink:type "resource"
Xlink:label="sliceSession"
</session>
<!-- definition of the links
between slices-->
<link1 xlink:type="arc"
xlink:from ="sliceCinemaChain"
xlink:to="sliceIndex1"
xlink:show="replace"
xlink:actuate="onRequest" />
<link2 xlink:type="arc"
xlink:from ="sliceIndex1"
xlink:to="sliceCinema"
xlink:show="replace"
xlink:actuate="onRequest" />
<link3 xlink:type="arc"
xlink:from ="sliceCinema"
xlink:to="sliceIndex2"
xlink:show="replace"
xlink:actuate="onRequest" />
<link4 xlink:type="arc"
xlink:from ="sliceIndex2"
xlink:to="sliceMovie"
xlink:show="replace"
xlink:actuate="onRequest" />
<link5 xlink:type="arc"
xlink:from ="sliceMovie"
xlink:to="sliceIndex3"
xlink:show="replace"
xlink:actuate="onRequest" />
<link6 xlink:type="arc"
xlink:from ="sliceIndex3"
xlink:to="sliceSession"
xlink:show="replace"
xlink:actuate="onRequest" />
<link7 xlink:type="arc"
xlink:from ="sliceMovie"
xlink:to="sliceIndex2"
xlink:show="replace"
xlink:actuate="onRequest" />
<link8 xlink:type="arc"
xlink:from ="sliceIndex2"
xlink:to="sliceCinema"
xlink:show="replace"
xlink:actuate="onRequest" />
</logicalNavigationModel>
</Links>
```

图10 相应的Xlink代码

4 结论和进一步研究主题

在本文中，我们描述了MIDAS，一个关于WIS开发的模型驱动方法论。这个方法论是一个特定的关于Web平台的MDA应用，采用了XML和(对象-)关系技术。MIDAS提出了一些平台独立模型（PIM）和平台特定模型（PSM）。同时，它也定义了上述模型之间的映射。我们依据MIDAS中定义的结构维和行为维为这些模型做了一个初步的分类。

在本文中，我们集中在一个WIS的结构维上，首先描述了内容层、超文本层和展现层的PIM；然后，总结了这些层的PSM。接下来，我们展示了这些模型间的映射；最后，我们使用一个案例研究来验证这些建议，这是一个马德里电影院线WIS，它采用了扩展UML和MIDAS技术。

现今，我们也正在确定行为维的PIM和PSM；用同样的方式，我们也确定了结构维的PIM和PSM。我们也正在继续将MIDAS扩展到诸如J2EE 或者.NET的其它平台和技术，并且继续实现模型之间的未定义的映射，用于半自动地生成信息系统。

致谢

这项研究得到了西班牙科学与技术部的支持。

参考文献

1. Bray, T., Paoli, J, Sperberg-McQu4een, C. M. and Maler, E., Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation. Retrieved from: http://www.w3.org/TR/2000/REC-xml-20001006/, 2000.

2. Conallen, J., *Building Web Applications with UML.* Addison Wesley, 2000.

3. Eisenberg A. and Melton J., *SQL:1999, formerly known as SQL3.* ACM SIGMOD Record, Vol. 28, No. 1, pp. 131-138, March, 1999.

4. Isakowitz, T., Kamis, A. and Koufaris, M., *The Extended RMM Methodology for Web Publishing.* Working Paper IS-98-18, Center for Research on Information System. Retrieved from: http://rmm-java.stern.nyu.edu/rmm/, 1998.

5. Koch, N., Baumeister, H. and Mandel, L., *Extending UML to Model Navigation and Presentation in Web Applications.* In Modeling Web Applications, Workshop of the UML´2000. Ed. Geri Winters and Jason Winters, York, England, october, 2000.

6. Marcos, E., De Castro, V. and Vela, B. *Modelado de Servicios Web con UML: Un caso de estudio*. Workshop on Advances in Database and Information Retrival. Apizaco, Mexico, september, 2003. Accepted.

7. Marcos, E., De Castro, V. and Vela, B. *Representing Web Services with UML: A Case Study.* The First International Conference on Service Oriented Computing (ICSOC03). Submitted.

8. Marcos E., Vela B. and Cavero J. M., *Extending UML for Object-Relational Database Design*. Fourth Int. Conference on the Unified Modelling Language, UML 2001, Toronto (Canadá), LNCS 2185, Springer Verlag, pp. 225-239, 2001.

9. Marcos, E. Cáceres, P., Vela, B. and Cavero, J.M., *MIDAS/DB: a Methodological Framework for Web Database Design.* DASWIS 2001. Yokohama (Japan), November, 2001. LNCS-2465. Springer Verlag. ISBN 3-540-44122-0. September, 2002.

10. Marcos, E., Vela, B., Cavero J.M., Cáceres, P. *Aggregation and Composition in Object-Relational Database Design*. 5 th. Conference on Advances in Databases and Information Systems. Ed.: Albertas Caplinskas and Johann Eder, pp.195-209, 2001.

11. Marcos, E. , Vela, B. and Cavero J.M., Methodological Approach for Object-Relational Database Design using UML. Journal on Software and Systems Modeling (SoSyM). Springer-Verlag. Ed.: R. France y B. Rumpe. ISSN: 1619-1366. Volumen SoSyM 2, pp.59-72, 2003..

12. Miller, J. and Mukerji, J. (Eds). (2001) Model Driven Architecture. Document number ormsc/2001-07-01. Retrieved from: http://www.omg.com/mda, 2003.

13. Miller, J. and Mukerji, J. (Eds). (2001) MDA Guide Version 1.0. Document number omg/2003-05-01. Retrieved from: http://www.omg.com/mda, 2003.

14. Retschitzegger, W. y Schwinger W. Towards Modeling of Data Web Applications- A Requirement's Perspective. In Proceedings of the America's Conference on Information Systems. V.I, pp. 149-155, 2000.

15. Soley, R. And the OMG Staff Strategy Group. *Model driven Architecture.* Object Management Group, White Paper. November 27, 2000.

16. Vela, B. and Marcos E., *Extending UML to represent XML Schemas*. The 15th Conference On Advanced Information Systems Engineering (CAISE '03). CAISE'03 FORUM. Klagenfurt/Velden (Austria). 16-20 June 2003. Ed. J. Eder, T. Welzar. Short Paper Proceedings. ISBN: 86-435-0549-8. 2003.

17. Vela, B. and Marcos E., *Hypertext Design in UML: From Conceptual to Logical Slice and Navigation Models in XLink*. 20th Int. Conference on Data Engineering (ICDE 2004), Submitted.

18. Vela, B., Cavero, J. M. andMarcos, E. *Diseño de Bases de Datos Objeto-Relacionales con UML.* 1ª Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento (JIISIC'2001). Anales de las Jornadas Iberoamericanas de Ingeniería del Software e Ingeniería del Conocimiento. Ed. Silvia Teresita Acuña y Cecilia María Lasserre. Editorial Universidad Nacional del Jujuy, 2001, pp. 59-68. Buenos Aires (Argentina). 13-15 June 2001

19. W3C XML Schema Working Group. *XML Schema Parts 0-2:[Primer, Structures, Datatypes]*. W3C Recommendation. Retrieved from: http://www.w3.org/TR/xmlschema-0/, http://www.w3.org/TR/xmlschema-1/ and http://www.w3.org/TR/xmlschema-2/, 2001.

20. W3C XML Linking Language (XLink) Version 1.0. W3C Recommendation. Retrieved from: http://www.w3.org/TR/xlink/, 2001.

21. W3C Extensible Stylesheet Language (XSL) Version 1.0. W3C Recommendation. Retrieved from: http://www.w3.org/TR/xsl/, 2003.

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！

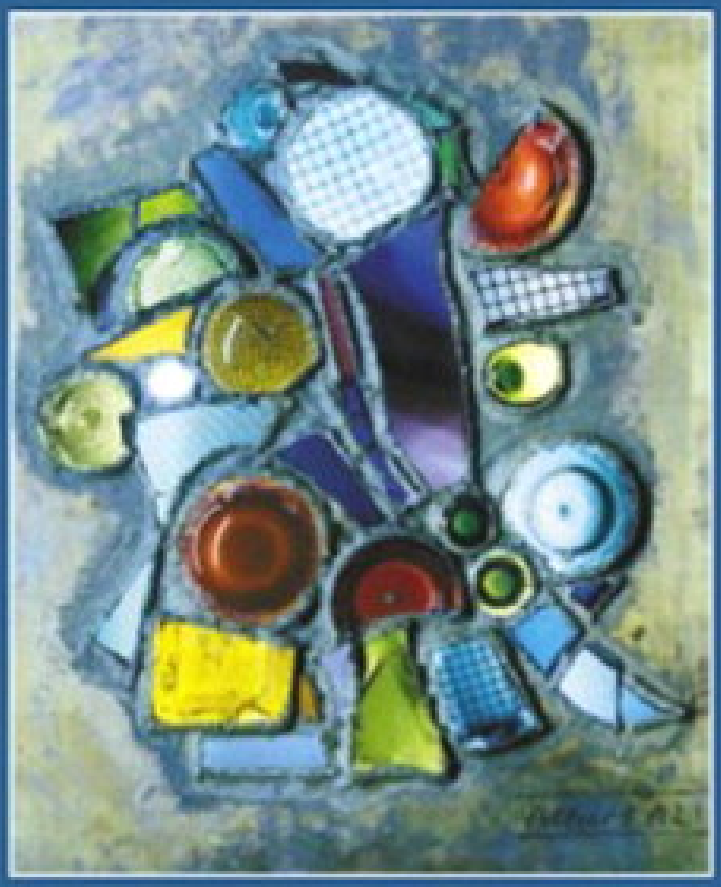
UMLChina 训练
辅助教材

Alan Shalloway & James R. Trott 著
熊节 译

清华大学出版社

Object Design

Roles, Responsibilities, and Collaborations



一说起“面向对象”就是“设计模式”？ ——对象的回归！

Rebecca Wirfs-Brock and Alan McKean

Forewords by Ivar Jacobson and John Vlissides

《对象设计》中译本即将由人民邮电出版社出版

UMLChina 辅助教材

应用程序的安全架构模式

Joseph Yoder、Jeffrey Barcalow 著，朱小平 译

讨论

摘要

编写安全的应用程序要比编写一个用密码保护的登陆界面难得多。这篇文章介绍了一些可以在处理应用程序安全问题时使用的模式。安全访问层（Secure Access Layer）给应用程序提供了一个访问系统安全特性的界面。单访问点（Single Access Point）限制主体通过单访问点访问应用程序。检查点（Check Point）使开发者可以处理未知或变化的安全策略。用户组有不同的角色（Roles），它定义了组成员可以做什么和不能做什么。有关用户的全局信息通过会话（Session）在整个应用程序中使用。最后，用户可以得到一个只包含合法参数的受限视图（Limited View）或者一个带错的完整视图（Full View With Errors）。这七个模式组合起来提供了创建安全应用程序的框架结构。

简介

开发应用系统时经常忽视安全问题。这种忽视，主要是因为开发人员过于关注领域知识而对如何保护系统却没有太多的考虑。开发人员建立模型，并通过它研究如何满足用户的需求。在这种情况下，安全通常是她或他最后才考虑或担心的问题。但是，到最后部署系统时，才发现此时往系统中增加安全模块要比增加一个用密码保护的登录界面难得多。这篇文章描述了如何设计一个安全系统，使得它的安全细节可以在后续的开发中实现。

在一个关注安全的企业环境中，一般都有关于物理设备，操作系统，网络和应用程序安全需求的详细文档。这些文档描述了如下主题，比如：用户权限；采用多么安全的密码和多长时间更换密码；数据是否需要加密；需要多安全的通信层等。

安全经常被忽视的一个原因是缺少安全策略，或者是安全问题看起来可以放在后面解决。忽视安全因素是非常危险的，因为在一个已经完成的应用程序里加入安全模块很困难。在应用程序设计时就考虑安全因素可能会使设计变得复杂，但是它要比在开发阶段再加入安全设计更清晰。同时，大规模的代码重写可以被避免，因为企业的安全策略可以被继承到开发的各个阶段。一个考虑周到的安全设计更容易适应变化的安全需求。

本文中所描述的七个模式可以在设计安全应用程序时使用。但它们并不代表安全模式的全部。相反，它们是一个起点，由此可以收集那些能够帮助开发人员解决应用程序安全问题的模式。两个作者都具有重构一个已开发好的系统，以使之能够满足企业的安全需求的经验（The Caterpillar/NCSA 金融模型框架 [Yoder]）。从上述经验中，我们发现下面的模式可以应用于开发安全程序。一般来说，在设计应用程序的架构时能够意识到这些模式会非常有帮助，即使你在后面的开发中并不需要这些安全特性。

安全的应用程序不允许用户通过后门查看或编辑敏感数据。单访问点（Single Access Point）可以通过将访问应用程序的入口限制为一个来解决这个问题。同样，只有应用程序的组件以及组件之间的交互是安全的，应用程序才是安全的。因此，任何一个应用程序都应该有个安全访问层（Secure Access Layer）来与外部系统安全通信，并且应用程序的各个组件都应该以安全的方式与之交互。这个层使得应用程序的代码与它的外部接口相互独立。

在处理违反安全策略的行为时，能够提供一个可以验证用户并做出正确决定的地方非常重要。检查点（Check Point）封装了处理不同安全策略以处理不同的安全入侵，并能够对不同的安全破坏处以合适的惩罚。

如果有多个用户使用系统，用户权限可以以用户的职位分类而不是以它们的名字分类。在这种情况下，用户可以被分配某种角色（Roles），此角色描述了此类用户可以做的动作。一组用户可以有不同角色，这些角色描述了此组用户可以做什么，不能做什么等。例如，应用程序可能提供管理应用程序，创建并维护应用数据，查看数据的角色。应用程序的所有用户都被分配某种角色以描述他们具有何种权限。

有些数据可能在整个安全系统中使用，比如用户是谁，他们有何权限，系统状态如何等。例如，当访问一个数据库时，应用程序需要知道用户名和数据库的位置。多个用户可能同时登录系统，他们的信息需要单独维护。关于用户的全局信息通过会话（Session）在整个应用程序中被使用。

用户的系统视图依赖于她或他当前的角色。这些视图可以表现为一个只有合法选项的受限视图（Limited View），此视图只包括她或他有权查看或运行的部分。

另一个选择是给用户显示一个带错的完整视图（Full View With Errors），允许他们查看所有的部分。用户当前没有权限的部分可以被置为不可用，或者当用户执行非法操作时抛出异常。

表 1 列出了本文所讨论的应用程序安全模式。它同时列出了每个模式所解决的问题。这些模式组合在一起提供了解决应用程序安全问题的方案。在本文的最后一节“一起工作”中讨论了如何综合使用这些模式。

模式名	意图	页码
单访问点（Single Access Point）	提供一个登录系统的模块和一种登录系统的方式	2
检查点（Check Point）	组织安全检查以及检查后采取的动作	4
角色（Roles）	组织具有相同权限的用户	7
会话（Session）	在多用户环境中把全局信息局部化	10
带错的完整视图（Full View With Errors）	提供给用户一个完整视图，并在必要的时候显示错误	13
受限视图（Limited View）	只允许用户看他们有权查看的部分	14
安全访问层（Secure Access Layer）	把底层安全集成到应用程序的安全中	18

表 1 模式列表

这篇文章没有讨论底层的安全模式或主题，它是安全访问层（Secure Access Layer）的基础。这些底层安全模式处理如加密，防火墙，Kerberos 和 AFS 等这样的问题。可以找到很多关于底层安全的主题和技术的资料。国际密码软件网[ICSP]和应用密码学[Schneier95]是探索这些主题深层内容的很好的参考资料。

单访问点

别名

登录窗口（Login Window）

一条路进入（One Way In）

监视门（Guard Door）

验证界面（Validation Screen）

动机

军事基地是代表安全场所的一个基本例子。军事人员必须被允许进入而间谍，破坏分子，记者必须被拒之门外。如果军事基地有多个出入口，那么检查每一个出入口就显得困难而且花费不少。如果每个人都必须从一个检查站出入，那么安全性就比较容易保证。

为一个与网络，操作系统，数据库和其他基础系统通信的应用程序提供安全性比较困难。应用程序需要提供一个途径，使得用户可以登录系统，并为之创建权限信息；而且它需要和与之交互的系统的安全模块集成。有些时候，用户可能需要在几个系统上被验证。而且，一些用户提供的信息可能需要保留以供将来处理。单访问点（Single Access Point）通过提供单一的进入点以验证用户并收集与用户相关的全局信息来解决这个问题。

问题

如果一个安全模型有许多“前门”，“后门”，“侧门”，验证用户就比较困难。

条件

- 有多种方式打开应用程序使得它在不同环境中容易使用
- 应用程序可能由多个需要安全模块的应用程序组成
- 不同的登录窗口或过程可以有重复的代码
- 一个单一入口点可能需要收集所有与用户相关的信息供整个程序使用
- 多个入口点的程序可以只收集与入口相关的用户信息，这样，用户可以不输入无关信息

解决方案

只提供一个进入系统的途径，如果需要的话，可以创建一个机制以决定启动哪个子程序。

一个典型的解决方案是创建一个登录界面用于收集与用户相关的信息，如用户名，密码，或者是一些配置参数等。这些信息可以传递给检查点（Check Point）去验证。然后一个关于配置参数和用户权限的会话（Session）被创建。这个会话用于跟踪处理用户与应用程序当前交互的全局信息。当打开新的子程序时，会话被传递给检查点（Check Point）以验证信息并处理错误。

UNIX 是一个有多个访问点的系统的例子。每个网络端口提供一个访问特定服务的入口。这种多访问点的特性使得 UNIX 系统难以保证整体的安全性。但是，在 UNIX 系统上运行的单个程序依然可以保证其安全性。例如，如果一个应用程序有 WEB，EMAIL 和编程界面，就必须采取措施以保证其中任何一个界面都必须通过一个共享的单访问点（Single Access Point）才能访问程序的其余部分。

在“一起工作”一节中，详细描述了初始化进程。在这里，单访问点（Single Access Point）的重要思想就是提供一个合适的位置来封装初始化进程，使它可以验证初始的安全步骤是否正确执行。

例子

有许多单访问点（Single Access Point）的例子。当使用一个 NT 工作站，所有用户都必须通过一个登录界面才能访问系统。这个单访问点（Single Access Point）验证用户并保证只有合法用户才能访问系统，它同时为合法用户分配角色以保证他们可以查看和操作他们有权限查看和操作的部分。大多数 UNIX 系统同样提供单访问点（Single Access Point）来得到一个终端外壳（Console Shell）。Oralce 的许多应用程序，如 SQLPlus 或者类似的程序都提供单访问点（Single Access Point）作为唯一可以运行程序的方法。

结果

✓单访问点（Single Access Point）提供一个位置，在那里，应用程序的所有部分都可以被正确地创建。这个唯一的位置可以保证所有的值都被正确地初始化，应用程序被正确地创建，而且应用程序没有到达一个错误的状态。

✓控制流程可以更简单，因为所有访问都必须通过单访问点才能确定是否有权限访问系统。注意，单访问点（Single Access Point）上的安全性与它的前置操作相同。

× 应用程序不能通过提供多个入口使进入应用程序更容易，更灵活。

相关模式

● 单访问点（Single Access Point）通过一个检查点（Check Point）验证用户的登录信息，并使用此信息初始化用户的角色（Roles）和会话（Session）。

● 登录类可以使用单件（Singleton[GHJV 95]），特别是你只允许用户有一个登录会话或者只允许登录系统一次。单件（Singleton）可以用于跟踪所有的会话并用一个关键字（Key）来确定使用哪个会话。

已知应用

● UNIX telnet 和 Windows NT 的登录程序使用单访问点（Single Access Point）来登录系统。这些系统为当前的会话（Session）创建了必要的角色（Roles）。

● 许多应用程序的登录界面是程序中的一个单访问点（Single Access Point），因为它是启动并运行程序的唯一方法。

● The Caterpillar/NCSA 金融模型框架[Yoder]有一个 FMLogin 类，它提供了单访问点（Single Access Point）和检查点（Check Point）。

● PLoP'98 的注册程序[Yoder & Manolescu 98]提供了一个单访问点（Single Access Point），当用户注册 PLoP'98 时需要登录系统并输入信用卡信息。

● 安全的 WEB 服务器，如 Java Developer's Connection 看起来为每个 URL 提供了多个访问点。但是，WEB 服务器强制每个用户在下载先期试用（early access）的软件时都必须通过一个登录窗口。

与安全无关的已知应用

● 任何程序只通过一种方式执行，以保证其正确的初始状态。

● Windows95 的登录窗口是一个单访问点（Single Access Point），但是它并不安全，因为它允许用户覆盖此登录界面。

● 单个创建方法提供了创建类的唯一方法。例如，VisualWorks Smalltalk[OS 95]中的 Points 类提供了一组创建方法来创建正确的点以保证对象被正确地初始化。Kent Beck's 描述的构造方法（Constructor Methods）是一个简单的创建正确对象实体的方法。它们遵循唯一的"实体创建"协议。当创建新对象时这成为一个单访问点（Single Access Point）。

● 参数化构造函数（Constructor Parameter Method [Beck 97]）通过唯一的方法初始化所有实例变量。它是类里面初始化实例变量的单访问点（Single Access Point）。

● 并发程序可以把非并发的对象封装在一个并发对象中。同步可以通过单访问点（Single Access Point）来实现。***Pass-Through Host*** 设计通过把所有合适的方法传递给使用非同步方法的***Helper*** 来实现同步。这样能够成功，因为对于***Host***类来说，这些方法都是无状态的。

检查点

别名

访问验证（Access Verification）

验证和授权（Authentication and Authorization）

制止黑客（Holding off hackers）

验证并处罚（Validation and Penalization）

处以与罪相适的惩罚（Make the Punishment Fit the Crime）

动机

军事基地的工作人员都有一个徽章，它会在通过检查岗时被检查。他们同样有一些钥匙，允许他们进入他们已被授权的区域。任何人如果没有徽章都不允许进入受限区域。任何人如果没有经过许可而进入受限区域都将受到严厉的处罚。

应用程序安全的一个目标就是阻止非法用户获得应用程序的权限，否则他们将会发现应用程序的保密信息或者破坏应用程序的数据。单访问点（Single Access Point）通过限制进入应用程序的途径只有一个来解决这个问题。在单访问点（Single Access Point）上，必须检查用户是否被允许进入。不幸的是，这种检查可能会使合法用户进入系统变得困难。例如，一个常见的错误就是输入错误的密码。如果用户短时间内连续输入错误的密码可能表示用户正在猜测密码以进入系统。应用程序必须能够容忍偶尔的错误，同时还要尽最大的可能将黑客拒之门外。开发人员可以设计多个检查以确定用户是入侵系统还是犯了一个常见错误。这种检查可能会变得越来越复杂而且可能会分布到程序的各个模块中去，这样管理和维护应用程序就变得比较困难。检查点（Check Point）通过把这些检查组织起来以解决这个问题。

问题

应用程序必须能够阻止入侵企图，当这种企图发生时，应当采取正确的行动。不同的组织有不同的安全策略，这些策略需要集成到应用程序的设计中，并且对应用程序保持独立。

条件

- 提供验证用户和验证用户能够做什么的方法非常重要
- 如果用户犯错，不应立即惩罚这种错误
- 如果错误频繁发生，就应该采取行动
- 对严重程度不同的错误要采取不同的应对措施
- 如果你的程序中有太多的错误检查代码，那么你的程序将变得难以调试和维护

解决方案

创建一个封装公司安全策略算法的对象。这个对象通常记录有多少异常发生并根据异常的破坏程度决定采取何种行动。任何安全检查都可以是检查点（Check Point）算法的一部分，如密码检查，欺骗（spoofing）超时等。

任何组织都会有某种类型的安全策略需要开发人员去实现。在应用程序设计时，安全策略可能不可用，而且这些策略可能在应用程序的生命周期中会改变。这里的首要目标就是在设计应用程序时，预留一个位置以封装安全策略。在应用程序原型中，开发人员可能创建一个假的（dummy）对象，它允许任意用户进入系统。这个假的（dummy）检查点可以在后续的开发中加入安全检查。

检查点（Check Point）的实现综合了如下几种模式。单访问点（Single Access Point）保证安全检查被正确执行，并且没有初始的安全检查被跳过。检查点（Check Point）的一个关键特性就是某个点上的安全策略是可变的。因此，检查点的算法是一个策略（Strategy[GHJV 95]）。策略包含了应该执行那种安全检查，检查顺序是怎样的，如何处理错误等。可以根据不同的安全等级嵌入不同的策略（Strategy）对象。作者发现，检查点（Check Point）的策略质量（Strategy quality）非常有用，因为安全检查可以在开发时被关闭，而在测试和部署时被打开。

在一些系统中，这个模式被分解为两个概念部分：验证和授权。验证检查用户是谁，他在哪儿。授权检查一个已验证用户的权限。这种分离技术在分布式系统中会比较有用，因为在此情况下，应用程序运行在不同的地方。用户可以通过一个登录程序集中验证，使得他们不需要为每一个应用程序都重新输入一次密码。授权可以根据每个应用程序的不同操作来进行。检查点（Check Point）使用用户角色（Roles）来为系统提供正确的授权。

检查点的设计可能会因为安全策略的不同而不同。如果一个公司的安全策略还未定义或者可能会改变，检查点（Check Point）的算法就需要仔细地设计。在设计未确定安全策略的检查点时，应该考虑以下三个因素：错误的行为，重复的频率，和延迟的检查（deferred checks）等。

根据不同的安全破坏或错误，不同的失败动作被执行。失败动作可以根据错误的严重程度来分类。这些失败动作（failure actions）与你实现的安全策略有关。例如，最简单的动作就是返回一个警告或错误信息给用户。如果错误不是很严重，安全算法可以把它当作一个警告而继续执行程序。更高一级的失败可能会强制用户重新登录。再高一级的错误可能导致登录进程失败或者程序退出。最高一级的失败会导致用户或机器被锁住。在这种情况下，管理员需要重置用户帐号和机器的访问权，不幸的是，如果稍后合法用户想要登录系统就不太可能了。因此，如果破坏不是很严重的话，可以把用户和机器的失效标记打上时间戳，在过了一个小时或更长的时间后，重新激活用户帐号和机器的访问权。还有，所有的安全失败信息都应该被记录下来。

有时，一个安全破坏的严重等级与破坏重复了多长时间有关系。用户输入密码错误一次两次是不应该立即受到惩罚的。连续三次或四次输入错误密码可能表明黑客在猜测密码。为处理这种情况，策略（Strategy）应包含一个计数器以记录安全破坏（security violations）的频率，并把算法参数化以处理不同的情况。

如果用户进入系统时，有些安全检查不能在检查点上执行，那它们必须在后面的某个地方执行。在这种情况下，检查点可以有一个二级接口供程序的其他组件使用，或者是提供一个单独的授权组件。这个二级接口是设计检查点时要考虑的第三个因素。

由于安全模块是应用程序中比较特殊的一个模块，所以如果能够开发一个可在多个应用程序中重用的安全模块就可以节省很多工作。但是，如果应用程序的安全需求差别很大，要达到这个目标就比较困难。所有的项目组都需要参与以保证框架可以满足应用程序的安全需求。

重用安全代码的一个方法就是创建可插入的安全组件库和一个能够将这些组件集成到应用程序中的框架。但是，把这些组件组合起来的算法常常是相互覆盖的，使得框架难以通用化。另一个方法是创建可配置的安全选项，它允许关闭或开启安全算法的某一部分。检查点（Check Point）是应用程序安全的核心部分，因此每一个逻辑分支都必须仔细检查。

例子

图 1 总结了在 Caterpillar/NCSA 金融模型中使用的检查点算法。进程从登录窗口的单访问点开始。密码检查和错误循环有三个计数器。其中一个用于记录程序从开始执行后的失败次数。另两个记录用户连续的失败次数和此终端上其他用户的失败次数，后者可以扩展为记录此终端上应用程序的启动次数。如果有太多的连续失败，用户或终端就会被锁定一个固定的时间段直到管理员重置用户帐号或终端。这种决策是 Caterpillar 安全策略的一部分，而且是可变的。密码通过安全访问层（Secure Access Layer）在数据库中验证。一旦输入正确的密码，其他几个检查将同时被执行。这是因为登录可能被允许访问数据库，但是登录对应用程序来说可能是非法的。密码的过期时间在这几个检查之后被检查。注意，如果因为发生连续错误而锁定帐号或机器，输入新的密码时可以直接关闭应用程序。当检查点上的检查完成之后，将根据角色（Roles）创建会话（Session）。检查点上的检查可以被记录到日志文件中。

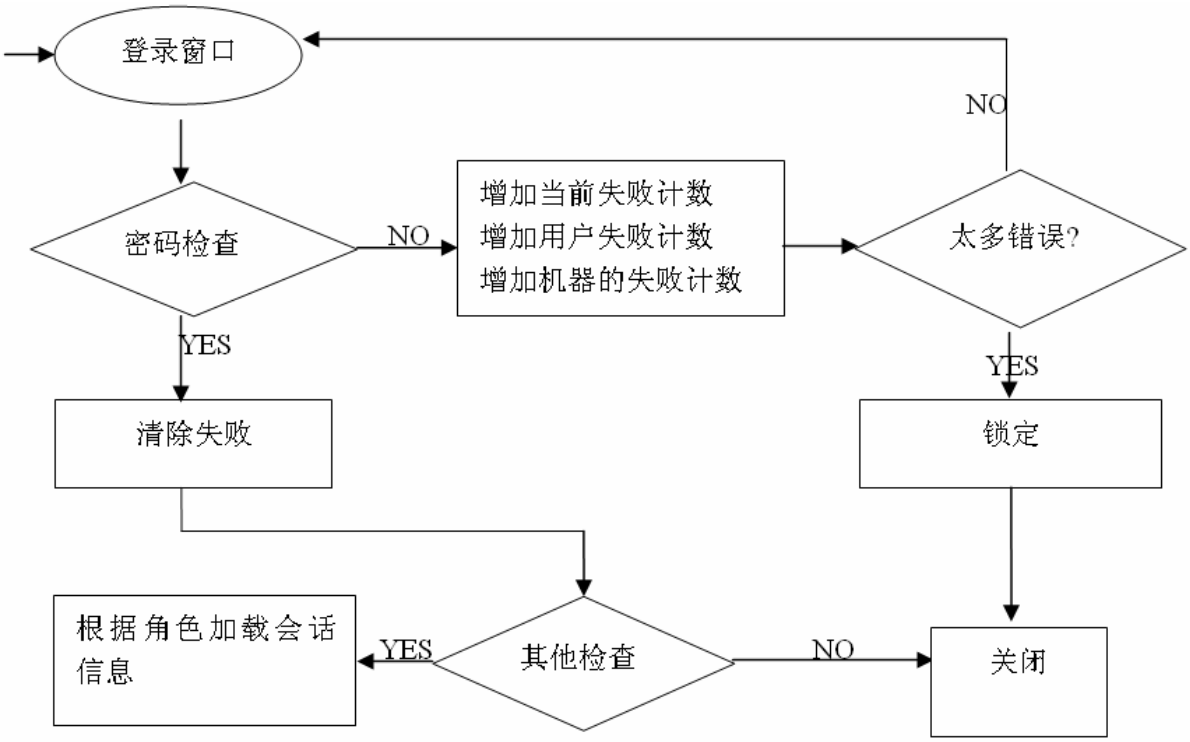


图 1 检查点算法的例子

❋其他检查包括：机器是否合法？机器是否锁定？用户帐号是否锁定？用户是否有正确的角色？用户密码是否过期？这些检查与公司的安全策略相关。

结果

✓检查点是一个关键安全位置，在检查点上必须进行安全检查。检查点把安全模型中需要确认的信息本地化。

✓检查点可以是一个复杂的算法。如果这种复杂性是无法避免的，那至少应该把它孤立在一个地方，使得改变安全算法比较容易。

✗一些安全检查不能在程序启动时执行，所以检查点必须有二级接口供程序中需要做这些检查的部分调用。这些检查所需的信息必须保存以供将来使用。这些信息，包括用户名，密码，角色等可以保存在会话（Session）中。

相关模式

● 检查点算法使用策略（Strategy）

● 单访问点（Single Access Point）保证检查点被正确地初始化，并且没有安全检查被跳过

● 角色（Roles）被用在检查点上做安全检查，角色的信息在检查点上被装载

● 检查点通常配置一个会话（Session）并把必要的安全信息保存在其中。在授权过程中，它与会话（Session）交互来得到用户的角色（Roles）。

已知应用

● FTP 服务器的登录过程使用检查点。根据服务器的配置文件，匿名用户登录可能被禁止也可能被允许。匿名用户登录时可能会要求输入一个正确的 Email 地址，这一点与 TELNET 类似。

● The Caterpillar/NCSA 金融模型框架[Yoder]使用检查点来检查密码，角色（Roles）和机器。上述的例子概括了它的实现。

● Xauth 使用 cookie 来提供一个检查点，使得 X-window 的应用程序可以在客户机和服务器间安全地通信

● 需要写用户硬盘的 Java 小程序用不同的方式来使用验证和授权过程。小程序使用 CA 来验证是谁创建了此程序。接着，用户手动授权小程序可以在用户机器上的沙盒（SandBox）之外运行。

● PLoP'98 的注册程序[Yoder & Manolescu 98]使用检查点，它只允许授权用户登录系统并更改他们的用户信息。

角色

别名

演员（Actors）

组（Groups）

项目（Projects）

资料（Profiles）

职位（Jobs）

用户类型（User Types）

动机

每个军事基地都有一个指挥官。这个基地指挥官可以下达基地关闭或进入紧急状态的命令。下达这些命令是基地指挥官的权力，而不是坐在这个位置上那个人的权力。当基地指挥官调离或退休后，新的人会坐上基地指挥官的位置。“基地指挥官”是人的角色。

在多用户的系统中，安全变得更加复杂。用户可用查看、改变、或“拥有”应用程序的不同部分。如果用户数量巨大，用户的安全许可以被分为若干类。这些类别可以与用户的职位，经验或部门相关。在这种情况下，管理员管理大量用户的安全资料的任务就变得非常繁重。管理员需要一种相对简单的方式来管理这些安全许可。

问题

用户有不同的安全资料，有些资料是类似的。如果用户数量足够大或者安全资料足够复杂，那么管理这些用户权限就变得非常困难。

条件

● 如果用户数量太多，为每个用户分配独有的权限就变得非常困难

- 一组用户通常有共同的安全资料（profile）
- 用户可能需要单独的安全资料（profile）
- 安全资料（profile）可能会重叠
- 用户的安全资料（profile）可能会随着时间的改变而改变

解决方案

创建一个或多个角色对象，这些对象定义了用户组的许可权限和访问权限。

当用户登录到系统中时，他会被分配一组权限，这些权限描述哪些数据是可访问的和程序的哪个部分可以被激活。从管理员的角度来看，这种用户－权限之间关系是M-N的[如图2]，难以管理。

图2 用户权限关系图

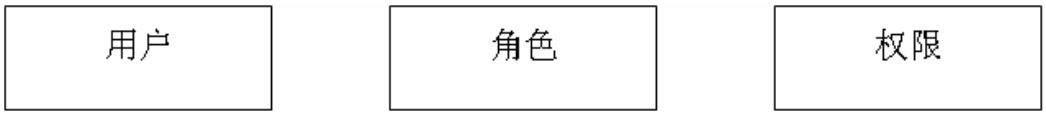


图3 用户－角色－权限关系图

引入角色（Roles）产生了两种新的需要管理的关系：用户－角色和角色－权限。这种新的关系使得管理安全变得容易。当工作职位的权限改变时，角色－权限的关系可以直接改变。当用户升迁时，用户－角色的关系可以直接更改，而不需要检查用户－权限关系是否正确。

有些时候，原来的用户－权限关系还需要维护以允许用户可以拥有私人的权限。简单的办法是为每个用户创建一个独立的角色，这个角色与用户名类似。只有在增加的间接层能够提供比原来的用户－权限关系更好的概念模型和更强的可管理性时，角色才应该被采用。

对角色－权限关系来说，角色对象知道它被授权了何种权限。一个反向的实现将会要求每个权限在返回值或执行操作时都检查它的角色。这种方法把安全代码分布到了整个系统中，所以应该避免使用。常见的实现是在角色里定义权限，例如受限视图（Limited View）模式。

给用户只分配一个单独的角色可能会简化组织结构，但是在现实中，用户可能既是会计又是经理。可以创建一个“会计经理”角色，但这不是一个通用的解决方案，因为它将会为每一种可能性都创建一种角色。一个更好的解决方案是用户对象的角色变量可以保存多个角色。从管理员的角度来看，这种方法使得给用户分配一组角色变得容易，但它却使得在应用程序中的查找权限变得复杂。角色－权限关系必须在每个用户角色中被检查，这就意味着需要用循环比较替代简单的比较。

这个模式展示了使用角色进行设计的一个简单的方法。这个模式的一个变体允许使用多种角色。例如，用户对象可以有一组角色描述它的编辑权限而另一组角色描述它的查看权限。如果角色重叠得很厉害，那么创建一个角色和子角色的继承树可能会更直观一点。你可以考虑在角色中包含其他角色。那么，你可以创建一个组合（Composite）角色来管理各种角色。主管的角色可以包含雇员角色，并且拥有访问他所负责的这些雇员详细信息的权力。Martin Fowler 的论文中有关于角色及其实现的详细论述。在这篇文章中，我们只讨论角色在安全领域的应用，但是，需要指出的是，角色在各种应用程序的设计中都很有用处。

例子

角色在安全设计中的用法有很多种。The Caterpillar/NCSA 金融模型框架使用了一种相对比较直接的方法。当用户成功地通过验证和其他安全检查后（检查点），应用程序在一个受保护的表中查找用户角色。接着，用户从他所能担任的角色中选择一种角色（如会计，经理，或开发人员等）。应用程序使用被选择的角色作为数据库会话的角色（数据库访问使用安全访问层）。由于数据库表的存取权限可以根据角色来定，所以设置数据库会话的角色就自动创建了一部分受限视图（Limited View）。用户的登录名可以在后续的权限检查和视图限制中被使用。

结果

✓管理员不管理用户－权限关系，而是管理用户－角色和角色－权限关系

✓角色是管理员管理用户的有效手段

✓角色是组合一般权限的好方法

✓角色可以简化管理任务。比如，所有的新员工都允许查看和编辑训练数据库，但是只能查看真实数据库。可以为这种权限创建一个“训练”角色。接着，所有的新员工都会被分配一个训练角色而不是一个可能会变得非常庞大的权限集。

✗角色增加了一个层次，使开发变得复杂。

× 即使使用角色，每个用户依然可能需要私有的角色去维护私有权限和属性

相关模式

- 使用角色（Dealing with Roles）[Fowler 97-2]提供了一个完整模式语言讨论角色及其实现细节
- 如果用户没有正确的角色而执行某个操作，检查点（Check Point）就应该起作用
- 用户角色在整个应用程序中都会用到。角色信息可以保存在会话（Session）对象中，在使用时调用。
- 角色可以用于确定受限视图的范围
- 当一个程序的行为与用户的职责有关时，用户的角色可以被用做选择策略（Strategy）的条件。

已知应用

- UNIX 把对文件和目录的访问分为三类。中间的一类是“组”，这是角色的一个例子。用户－角色的关系保存在文件/etc/group 中，并且以角色排序。文件系统保存着角色－权限的关系。
- 一些 web 服务器使用.htaccess 和.htgroups 文件定义用户组（角色）可以访问 web 站点的哪些区域。
- Oracle 数据库使用角色定义安全权限。用户－角色和角色－权限的关系保存在表中。
- GemStone 数据库的数据保存在段（segment）中。GemStone 以类似 UNIX 处理文件的方式处理段。GemStone 中的用户可以有一个或多个组（角色），并且每个段都为所有用户，组集合和所有者定义了读写权限。由于段可以有多个组，这一点比 UNIX 的组稍微强大一点。
- PLoP’98 的注册程序[Yoder & Manolescu 98]有两个角色：参加者和管理员。

与安全无关的已知应用

- Office 97 的帮助系统是在非安全应用中使用角色的一个例子。卡通曲笔针帮助形象可以配置为不提供帮助(专家用户)和提供一般性帮助(初学者)。这个配置与用户的当前角色有关。Windows ’95 profile 也是个角色。

会话

别名

用户环境（User's Environment）

名空间（Namespace）

基于线程的单件（Threaded-based Singleton）

全局变量本地化（Localized Globals）

动机

当军事人员在一个高度安全的军事设施内时他们的活动都被记录下来。他们的进出都有日志。他们的徽章必须时刻佩戴以便确认他们是否待在合适的地方。军事基地的卫兵假设这些佩戴徽章的人已经在基地入口处被检查过了。因此他们只作最小的检查就让这些人进入受限区域。基地同一时间有很多人在工作。每一个安全徽章唯一地标示出这个人是谁，能够做什么。它同样可以记录佩戴徽章的人正在做什么。

安全程序需要记录在整个程序中用到的全局信息如用户名，角色，和他们的权限。当应用程序只保存一份这样的信息时，它常使用单件[GHJV95]模式。单件通常保存在一个全局的位置，如类变量。不幸的是，在多线程，多用户或分布式的环境中使用单件非常困难。在这种情况下，每一个线程或每一个分布式进程都可以看作一个独立的程序，它们都需要自己私有的单件。但是，如果应用程序不能共享一个公共的全局地址空间，这些全局单件就不能共享。 所以需要一个允许多“单件”的机制，每个应用程序一个。

问题

许多对象需要访问共享值，但这些值在整个系统中不是唯一的。

条件

- 引用全局变量可以使代码更干净，更直接
- 每个对象都可能需要访问一些共享值
- 这些共享值可能随时间改变
- 多个同时运行的程序可能不共享同样的值

- 在应用程序中传递多个共享对象会使 API 更复杂
- 一个对象此时不需要某些值，但它可能以后会需要这些值

解决方案

创建一个会话对象，它保存所有变量，这些变量会被许多对象共享。

会话对象定义了一个名空间，会话对象中的每一个成员变量都共享这个名空间。会话对象被传递给那些需要访问它的成员变量的对象。

某些用户信息在整个系统中都会用到。一些信息与安全相关，如用户的角色和权限。会话对象提供了共享这些全局信息的一个很好的方式。这个对象可以传递给需要使用它的对象。

如果类继承的层次结构允许，一个会话变量可以加到所有需要会话对象的类的超类中。大多数情况下，在一个已存在的框架上扩展并构造程序，使用超类的方法使是不合适的，除非你想扩展一个没有良好设计的对象。因此，通常是在每一个需要会话对象的类中增加一个会话变量。

所有共享同一会话的对象都有一个通用的边界（scope）。这个边界就像是编译器在执行变量查找时使用的环境一样。主要的不同在于会话的范围由应用程序创建而查找由应用程序执行。

由于每个对象都有一个会话的引用，它就是一个存放应用程序当前状态（State [GHJV95]）的好地方。当然，为了保证安全，状态（State）的实现不一定要放在会话中。受限视图的数据和角色也可以在会话中做缓存。需要重点指出的是，用户不应该被允许访问会话对象中保存的安全数据如用户密码和权限等。使用会话对象组织应用程序是个好主意。这个对象保存所有的共享信息，这些信息会在用户与系统交互时使用到。

例子

除了保存安全数据，会话还可以用于保存许多不同类型的数据。The Caterpillar/NCSA 金融模型框架有一个 FMState 类[Yoder]。一个 FMState 对象可以当作一个会话。它为应用程序的组件提供了一个单独的地方来访问受限视图的数据，如当前选择的产品，用户的角色，和系统当前的状态。许多金融模型中的类都保存一个 FMState 对象的引用。这里不能使用一个真正的单件，因为用户可以根据不同的选择条件打开多个会话，每个会话都与一个不同的受限视图相联系。

图 4 展示了金融模型的 FMState 对象。安全信息包括对象名和角色。安全信息和选择条件定义了受限视图。每个 ReportView 和 ReportModel 类都有一个 FMState 类的引用，所有它们可以访问 FMState 的其他数据。

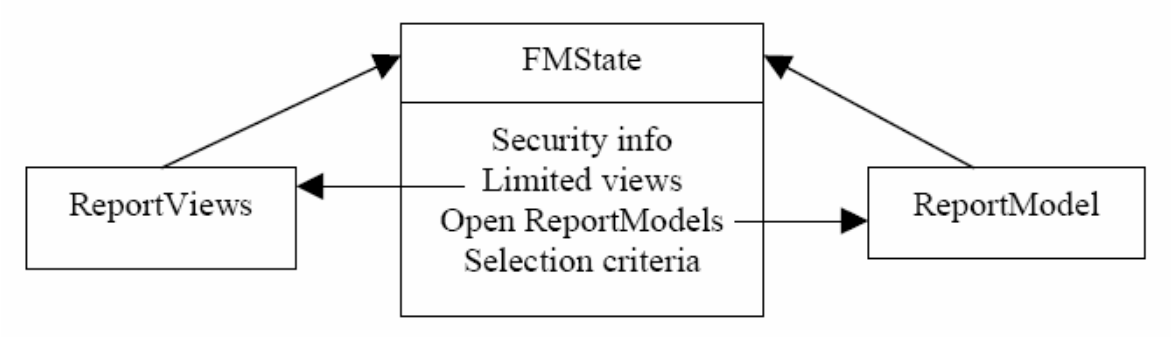


图 4 金融模型中的 FMState

结果

✓会话对象提供了应用程序中许多组件访问重要变量的公共接口

✓与在应用程序中分别传递许多值不同，现在可以只传递一个会话对象

✓当有一个新的变量或对象需要共享时，可以把它放入会话对象，所有可以访问会话对象的组件就可以访问此共享变量或对象了

✓变化的传播变得简单了，因为线程或进程中的每一个对象都只依赖于一个共享的会话对象

× 假设一个对象不需要一个会话，但它可能创建一个需要会话的对象。在这种情况下，第一个对象必须保存一个会话的引用，以便将它传递给新对象。有些时候，这会使得应用程序中的每个对象好像都有一个会话。会话变量在整个设计中的繁殖性增长不是件好事，但是，它是会话模式的一个必要的结果。

× 在开发阶段的后期加入会话对象比较困难。每一个对单件的引用都需要改变。笔者有在单件中更新会话的经验，并且可以肯定地说，如果单件分布在许多类中，这项工作会非常繁琐乏味。相同的，把多个需要传递的全局变量统一放入一个会话对象也有这样的效果。

× 如果有多个值存储在会话对象中，它就需要一个组织化的结构。可以把会话对象中的变量分类组织起来，从而有可能分解会话对象以降低其耦合度，但是分解会话对象需要详细的分析以确定哪些组件需要访问哪个子集的值。

相关模式

- 会话是单件[GHJV 95]在多线程，多用户，分布式环境中的替代模式

- 单访问点（Single Access Point）用检查点（Check Point）验证用户，如果用户通过验证，它返回一个会话对象
- 会话是实现状态（State）模式的好地方，因为状态在整个程序中都会用到
- 会话可以跟踪用户的角色并且可以缓存受限视图的数据
- Lea's Sessions[Lea 95]讨论了在资源管理中会话开始，中间，结束需要做的动作。这篇文章中的会话模式，与本文相反，主要保存的是在共享环境中，分散的，不能放在单件中的数据。
- 线程特殊存储模式允许多个线程使用一个逻辑的全局访问点去访问线程特殊数据而不需要在每次访问前都先加锁

已知应用

- 对 VisualWorks 来说，Oracle 的 Lens 框架和 GemStone 的 GemBuilder 分别有 OracleSession 和 GbsSession 类。每一个都保存诸如交易状态和数据库连接这样的信息。这些会话对象可以被相同的数据库上下文中的任何一个对象引用。
- The Caterpiller/NCSA 金融模型框架中有一个 FMState 类[Yoder]。一个 FMState 对象就是一个会话，它保存受限视图（Limited View）的数据，当前产品类别的选择，还有当前系统的状态。金融模型中的大多数类都保存一个对 FMState 对象的引用。
- PLoP'98 的注册程序[Yoder&Manolescu98]有一个会话对象，它记录用户访问系统时的用户全局信息
- 许多数据库使用一个会话来保存用户信息
- VisualWave[OS 95]的 httpd 服务使用了会话，它记录了所有对 httpd 的 web 访问
- UNIX 的 ftp 和 telnet 服务使用会话来记录用户请求并限制用户的动作

与安全无关的已知引用

- VisualWorks中的项目可以被用于划分两个或多个变化集。关于窗口位置的信息保存在每一个项目中，图像编码被所有的项目共享。因此，项目可以考虑使用与安全无关的会话。

带错的完整视图

别名

带异常的完整视图（Full View With Exceptions）

显示全部并处理异常（Reveal All and Handle Exceptions）

带通知的视图（Notified View）

动机

一旦一个指挥官被允许进入军事基地，他或她就可以去基地的任何一个建筑物。实际上，这个指挥官具有这个军事基地的全视图。如果指挥官没有正确的凭证就想进入一个受限区域，或者他会停下来而什么都不做，因为他知道自己不被允许进入受限区域，或者警报会响起并且卫兵出现来逮捕这名指挥官。

图形化的应用程序经常提供多种查看数据的方式。用户可以动态选择哪些数据用哪个视图查看。当应用程序具有多个视图时，开发人员就必须注意在当前的应用程序状态和用户权限下，哪些操作是合法的。那些决定操作是否合法的条件代码可以非常复杂并且难以测试。给用户一个所有用户都可使用的全视图可以了解用户如何方便地使用系统，并且可以编制出更通用的 GUI 界面。

问题

用户不应该被允许执行非法操作

条件

- 当一些操作不可见或者被禁止时用户可能会感到困惑
- 如果一些选项是根据用户的角色来决定出现或者隐藏的话，用户可能会对哪些选项是可用的而感到困惑
- 用户不应该看到那些他们不被允许的操作
- 用户不应该查看那些他们没有权限查看的数据
- 用户不喜欢他们被告知哪些事情他们不能做

- 用户会被安全错误，权限禁止或者非法操作信息等搞得很烦

解决方案

在设计应用程序时允许用户可以看到他有权看到的所有东西。当用户执行一个操作时，检查这个操作是否合法并且在他执行非法操作时给出一个错误提示。

这个模式在用户可以执行所有操作时非常有用。它很容易给用户显示所有的内容，并且在有非法操作时给用户一个错误提示。

这个模式的解决方案在有很少的错误信息需要显示时可以非常简单。可以只显示错误信息到标准错误上或者一个对话框中。如果整个程序中有很多错误提示，一个独立的错误报告机制就非常有用。这个机制也可以用于错误日志。

典型的，一个错误报告框架有两个主要的组件。日志事件对象有一个描述错误情况的消息，还有一个严重程度来说明事件是个警告，错误或者仅仅是用户消息。当一个日志事件被创建时，它自动将自己登记到*日志服务器*上。日志服务器是一个单件，它自动接收并处理日志事件。日志服务器可以被配置成显示一个对话框或者是将日志写到文件中，这与日志的严重程度有关。

例子

一个例子就是 oracle 数据库。当你使用 SQLPlus 去访问数据时，你可以执行任何命令。但是，当你试图去访问哪些你没有权限查看的数据时，相应的错误信息就会显示出来。

结果

- ✓ 培训材料对每类用户都是一致的
- ✓ 改进此模式并把它应用到一个已有系统中可以很直接。仅仅写一个 GUI 界面去处理所有的选项，当一个操作发生问题时，可以简单的推出并打开一个错误提示框
- ✓ 可以很容易地修改一个操作的权限，因为授权是在操作发生时才进行的
- ✓ 这个模式比较容易实现，因为你不需要为数据创建多个视图
- ✓ 如果错误提示太多，用户会感到迷惑
- ✓ 操作验证会很困难，因为用户可以执行任意操作

✓ 如果用户能够看到哪些他们不能执行的操作，他会感到很生气

相关模式

- **受限视图**是与此模式竞争的一个模式。如果完全的受限视图不可能实现，这个模式可以帮助解决这个问题。在本论文最后的“一起工作”一节中比较了这两个模式并描述了如何去使用它们
- **Checks [Cunningham 95]**描述了大量关于实现 GUI 的细节，其中包括如何放置错误提示
- **角色**可以用于错误提示或者验证用户可以做什么而不能做什么

已知应用

- 登录程序在用户输入错误密码时提示用户
- SQLPlus，用于访问 Oracle 数据库，允许你执行任意操作并在执行非法操作时显示相应的错误信息
- 大多数的文字处理器和编辑器，包括 Microsoft Word 和 vi 都允许你保存一个只读文件。当保存被执行时，程序会显示一个错误信息，而且相应的保存错误失败。
- 路透社的 SSL 开发包有一个报告错误，警告和通知事件的框架。你可以配置把错误报告输出到标准错误，文件，或者与系统相关的一个地方，比如一个对话框。

与安全无关的已知应用

- Assertions [Meyer 92]是此模式在一个编程接口中的应用。因为一个类的接口不能修改，并预先保证每个方法都在一个合法的上下文中被使用
- java 的编程模型抛出异常符合此模式。与设计一个只在正确上下文中执行的方法不同的是，异常可以被抛出并迫使程序的其他部分处理此错误。

受限视图

别名

眼罩（Blinders）

小孩勿动（Child Proofing）

不可见路段 （Invisible Road Blocks）

隐藏甜点瓶（Hiding the cookie jars）

早期授权（Early Authorization）

动机

如果一个国会议员想要参观她所在街区的军事基地，她的行程应该受到保护。在参观开始前，基地里的一些地方，例如有些地区会标记为危险地区，有些可能会标记为机密区域或者无须知道的区域。整个参观过程被预先根据参观者的安全权限来设定为军事基地的一个受限的视图。

图形界面的应用程序通常提供多种方式来查看数据。用户可以动态选择采取何种方式查看何种数据。如果应用程序有多种视图，开发人员必须关注在当前的应用程序状态和当前用户权限下，哪些操作是合法的。判断操作是否合法的规则可能会非常负责并且难以测试。通过限制用户可以访问的视图，判断规则可以被省略。

问题

用户不应该被允许执行非法操作

条件

- 如果有些选项有时出现，有时被禁止，用户就会感到迷惑
- 如果选项是根据用户的角色来确定是否出现，用户就会对哪些是可用的而感到迷惑
- 用户不应该看到那些他们不能使用的操作
- 用户不应该看到他们没有权限查看的数据
- 用户可能不喜欢被告诉哪些事情他们不能做
- 用户会对过多安全或者权限错误信息而感到厌烦
- 如果限制用户只能看到他们允许访问的数据，用户验证就可以非常简单

解决方案

只允许用户看到他们有权看到的。只提供根据用户当前访问权限允许的选择项或者菜单项。应用程序启动时，用户可以有一定的角色或者应用程序有缺省的视图。根据用户的角色，系统只允许用户选择哪些他们的角色允许的选项。如果用户不能编辑数据，应用程序就不显示相应的编辑数据的菜单或者按钮。

受限视图可以通过多种方式来控制。首先，受限视图根据用户的当前权限集来配置哪些选项是可选的。这就使得用户只能选择他们被允许查看的数据。其次，受限视图获得当前的会话和当前用户的角色，并把它应用到当前的程序状态上，同时，它根据这些信息动态创建用户界面。空对象(***NULL Objects*** [Woolf 97])可以用于创建用户没有权限去查看的用户界面。

第一种方法允许用户根据自己当前的角色来看到不同的列表和数据。这种方法一般用于有一组可选列表供用户选择去查看数据。用户界面是静态的，但是界面上显示的列表是根据当前用户的角色来变化的。一个用户可以有多个角色，而且可能选择一种角色来运行程序。当用户改变他的角色时，受限视图也随之变化。

例如，在一个金融应用程序里，经理可以访问一组有限的产品。如果在一个受限视图中显示可以选择的产品，应用程序只显示那些经理允许查看的产品。所以，当经理在选择可选的产品时，他不会得到一个“禁止访问”的错误。

把受限视图与一个由角色创建的会话一起使用时，由当前应用程序状态得到的视图同样是受限的。用户在屏幕上看到的图形界面是动态创建的。例如，如果用户的角色允许编辑，那么受限视图会增加用于编辑的按钮或者菜单。另一种可选的方式是，编辑选项一直被禁止，但是它们可以被动态的被允许，这依赖于用户的角色和应用程序当前的状态。***Strategy*** 模式可以用在这里以根据不同的角色来显示不同的界面。

通过限制用户只能看到他们有权看到的数据和有权执行的操作，用户知道哪些操作是合法的。例如，在 Microsoft Word 种，如果用户没有打开文档，文件菜单中不会显示保存文件的菜单项，因为此时没有可保存的文件。同样，在预览网络文件时，用户不能看到编辑文档的选项。

受限视图可以通过多种方式来实现。图形界面可以通过 Composites 和 Builders 模式来实现[GHJV 95]。Type-Object[Johnson & Woolf 97],Properties 和 Metadata[Foote & Yoder 98]等模式对创建动态图形界面也非常有用。另外，State 模式可以通过不同的类来代表不同的视图。Strategy 可以用于选择正确的 ***State***。

受限视图使安全和可用行最大化。不幸的是，这个模式比较难实现。如果你所面临的用户有很多不同的权限而且你不想使太多的错误提示激怒用户，你就应该考虑使用受限视图模式。

例子

一个可看的受限视图的例子如图 5 所示的选择框。在那里，用户只有一列产品供他们查看或编辑。但系统中同时存在其他的产品，但是那些产品没有显示，因为用户没有访问它们的权限。例如，如果图形界面要显示一组产品的交易详情，那么受限视图只会给用户显示大豆，玉米和干草的信息，因为用户只有权访问这些产品的信息。

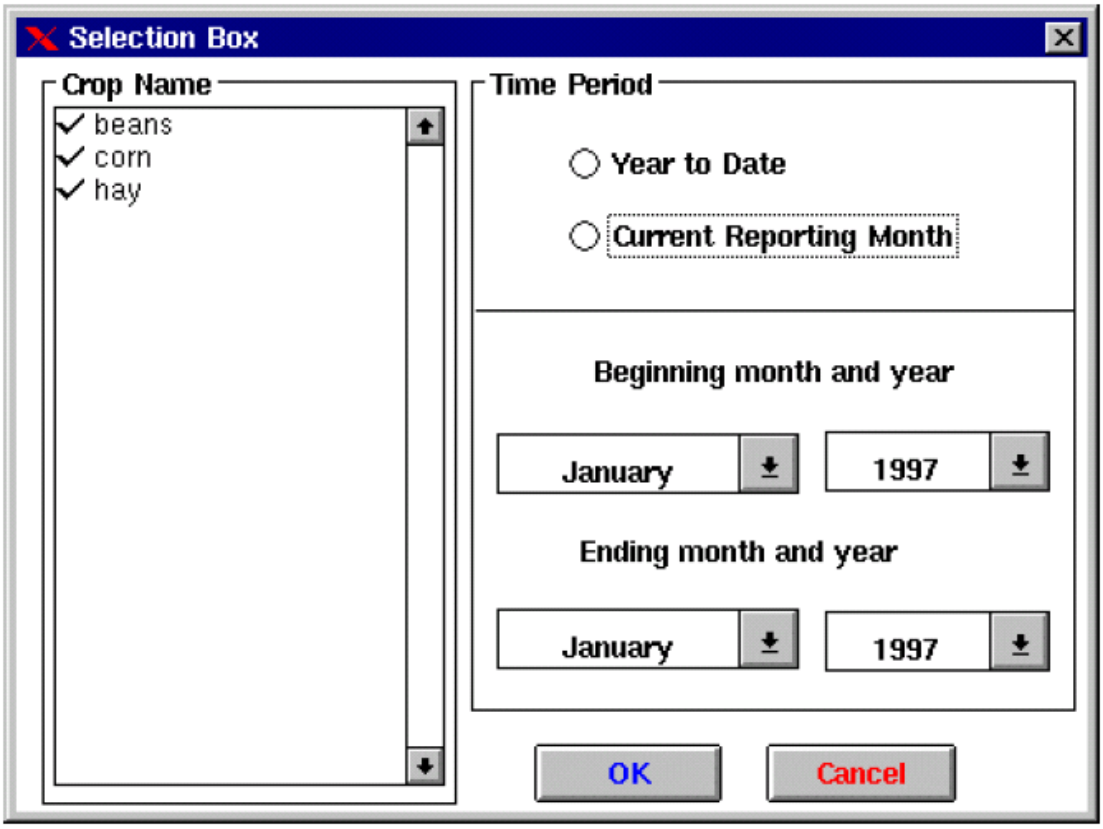


图 5 选择的受限视图

另一个例子如图 6，图 7 所示。我们注意到，图 6 没有显示修改数据库值的按钮，而图 7 有。图 6 的视图限制了用户修改数据的能力，因为用户没有权力修改详细的收入。在图 7 中，视图禁止了一些按钮。这些按钮禁止的原因是用户没有修改任何交易，所以这里没有任何值需要接收或者保存到数据库中。

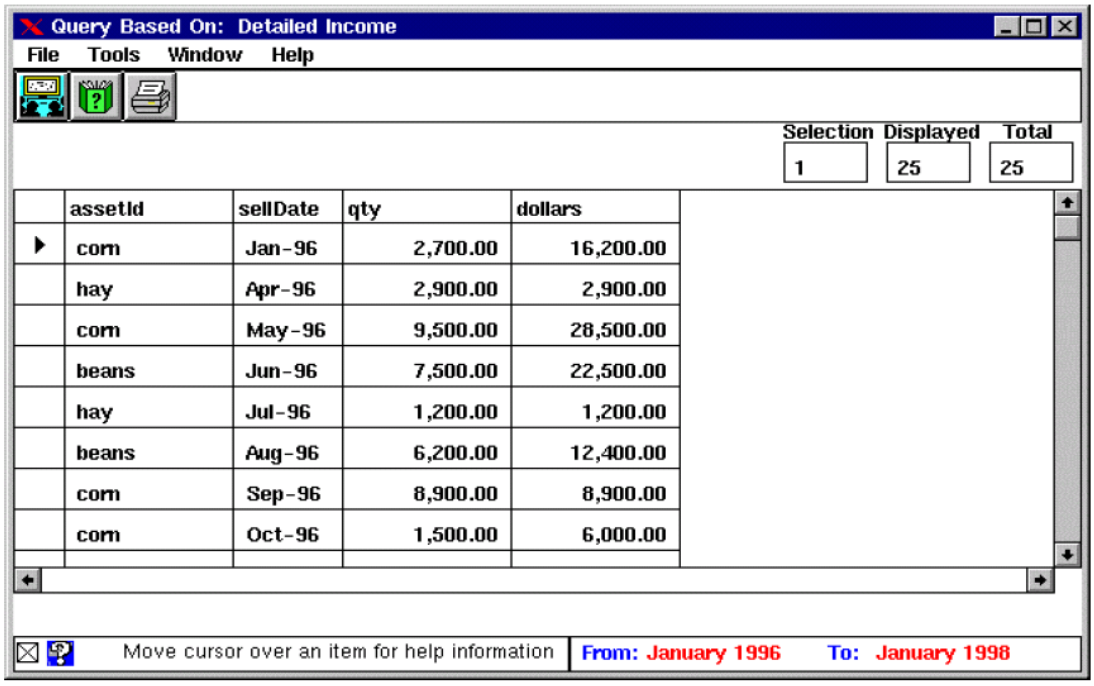


图 6 交易不可编辑的受限视图

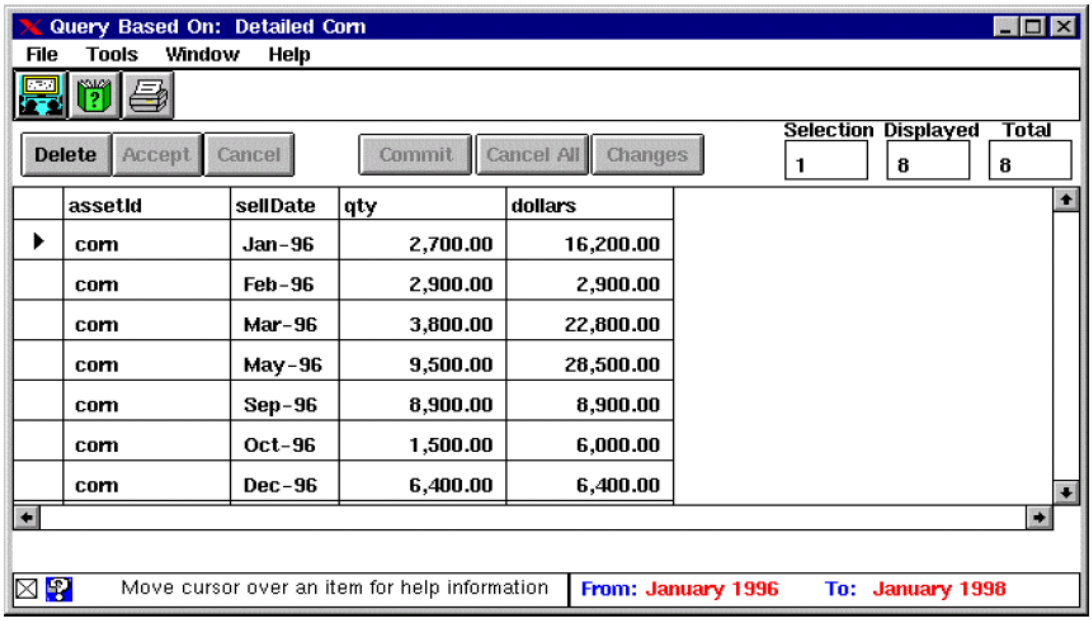


图 7 交易可编辑的受限视图

通过禁止或隐藏的方式来实现受限视图都有一定的缺陷，设计者需要根据整个应用程序的需要来做出权衡。如果主要用户不能编辑值，受限视图可能会隐藏编辑的按钮。相反，如果主要用户可以执行编辑操作，受限视图可能会简单地根据要求禁止按钮或菜单项。

已知应用

- Caterpillar/NCSA 金融模型框架[Yoder]给用户显示了一个数据的受限视图。这个框架同样在用户界面上提供了受限视图，它根据用户的角色来改变界面上编辑视图的显示。
- 防火墙提供了数据的受限视图，它通过过滤网络数据来保证数据只对某些系统可用
- WEB 服务器提供了一个受限视图，它只允许用户查看 WEB 根目录和用户 public_html 目录
- 许多操作系统提供了隐藏文件和目录的功能，这是受限视图的一种
- 微软的 Windows NT 根据用户角色来提供受限视图。用户只能看到他们有权查看的文件，而且所看到的用户界面是根据他们的角色动态生成的
- PLoP'98 的注册程序[Yoder& Manolescu 98]提供了受限视图，它有一个给管理员的视图和一个给那些注册 PLoP 用户的视图。注册 PLoP 的用户得到一个受限视图，此视图只允许他们查看和编辑他们的信息。

与安全无关的已知应用

- Netscape Communicator，Microsoft Office 和许多其他应用程序根据用户可以查看或编辑的数据来改变它们的用户界面。这通常通过上下文相关的菜单和按钮的允许或禁止来实现。例如，Excel 中编辑一个图和编辑一个表的可用菜单区别就非常大。同样，如果没有打开的文档，"File"菜单列表中的"Save"和"Save As"菜单项就不可用。
- Windows 95's 的鼠标右键菜单是上下文相关的，它只显示对鼠标所指对象合法的菜单项
- 在许多操作系统中提供的隐藏文件和目录的功能，是受限视图的一种方式
- 不像 Microsoft Word 和 Vi，Netscape Composer 不允许用户试图保存一个只读文档。如果打开的是只读文档，"Save"菜单项是置灰的。"Save as…"是受限视图的一部分。
- 所有的现代图形应用界面都提供某种受限视图，它动态的使能或禁止按钮或菜单项。

安全访问层

别名

使用底层的安全（Using Low-level security）

使用非应用程序的安全（Using Non-application security）

与链条上最弱的一环一样强壮（Only as strong as the weakest link）

动机

当安全文档从军事基地的一个安全区域传递到另一个安全区域时，保证安全文档在传递过程中不被破坏非常重要。如果文档通过磁盘传递，数据应该被加密并且锁到公文包中，而且公文包应该和传递员的手臂锁在一起。这些提供了一个独立的层次来保护传输过程中的信息安全。

许多应用程序都需要和其他系统来集成。程序集成的地方就会成为一个最弱的安全点和最容易被攻破的点。如果开发人员在应用程序和其他系统的交互部分加入强制的安全检查，那么代码就会变得复杂而且难以抽象。在不安全的基础上创建的程序是不安全的。换句话说就是，加固你的窗户而打开了你的后门并不能保证房间的安全。

问题

应用程序的安全如果不能和外部系统的安全很好地集成，那么应用程序就是不安全的。

条件

- 应用程序的开发不能考虑特定的操作系统，网络和数据库，这些在应用程序的生命期中是可以变化的。
- 把底层的安全代码分布的整个应用程序中会使得应用程序难以调试，修改和移植到其他系统上
- 即便应用程序是安全的，好的黑客可以通过拦截数据或者拦截钩子函数来访问敏感数据
- 与外部安全系统的交互是困难的
- 外部系统可能没有足够的安全设施，那么实现需要的安全是不可能的或者是不可行的

解决方案

在现有的操作系统，网络和数据库的安全机制上创建你的安全应用程序。如果安全机制不存在，你可以创建自己的底层的安全机制。然后在底层安全机制上，创建一个安全访问层来与应用程序本身或其他系统交互。

通常，应用程序需要和许多已存在的系统交互。例如，一个 Windows NT 上的金融应用程序客户端可能使用远程服务器上的 Oracle 数据库。因为大多数的系统都提供了安全接口，你的应用程序需要封装这些接口来安全地访问外部系统。应用程序与外部世界的所有交互都会通过这个安全访问层来发生。

这个模式的重点是创建一个层次来把开发人员和底层变化的安全机制隔离开。这个层可能有许多种协议，它依赖于需要使用那种通讯方式。例如，这个层可能使用一种协议来访问 Oracle 数据中的安全数据，而使用另一种协议通过 SSL[Netscape]与 Netscape 服务器通信。这个模式的关键在于模块化这些外部协议，使得它们可以被方便地使用。不同的安全访问层的架构差别可能非常大，至于如何模块化并集成这些外部协议已经超出了此模式的讨论范围。

通过创建一个安全访问层来封装与外部系统通信的标准协议，可以把这些外部接口本地化。应用程序开发人员可以把注意力集中到应用程序本身上，与外部系统的通信可以通过安全访问层的接口来完成。

这个模式假设一个合理的抽象是可能的。例如，VisualWorks's 的 LensSession 不能支持 Microsoft Access，因此，QueryDataManager 不能与 Microsoft Access 数据库一起使用。安全访问层提供了对数据库访问的一个更一般的抽象。第三方的 ODBC 驱动已经开发出来了，它可以与 Microsoft Access 通信。通过使用安全访问层，你的应用程序可以使用 ODBC 协议，这样，你的应用程序就可以与任何支持 ODBC 的数据库通信了。

例子

PLoP 的注册程序使用了一个安全访问层。创建这个层的原因是所有的注册通信都是通过 WEB 来完成的。这个层位于 Apache 的 SSL 之上。它防止信息在传输过程中被窃听，比如信用卡密码。同样，在数据库端，这个层通过把存入数据库的信用卡信息加密来提供额外的安全。这个层使用一个密钥来加密，解密数据。因此，即使有人可以通过后门来获得数据库中的数据，信用卡数据还是受保护的。

结果

✓安全访问层可以隔离应用程序与外部安全系统的通信。隔离的安全访问点可以使得集成新的安全组件和升级现有程序变得简单

✓安全访问层使得应用程序的可移植行更强。如果应用程序后来需要和Sybase数据库而不是Oracle数据库通信，由于数据库的访问是局部化的，所以只需要修改一处就可以了。QueryObjects[Brant & Yoder 96]使用了这种方法。它把所有的数据库访问都通过QueryDataManager来进行，而QueryDataManager是建立在LensSession[OS 95]基础上。LensSession可以映射到Oracle可Sybase数据库上。因此，应用程序开发人员不需要关心使用那种数据库或者将来数据库有什么变化。

×与你的应用程序集成的不同系统可能使用不同的安全协议和访问方式来与你的应用程序交互。这使得开发一个与所有系统集成的安全访问层变得很困难，而且开发人员可能需要保存一些并不是所以系统都需要的信息。

×在一个已经把安全访问代码分散到程序各部分的应用程序中使用安全访问层非常困难

相关模式

- 安全访问层是分层架构的一部分。Layers[BMRSS 96]讨论了构建分层架构的细节
- Layered Architecture for Information System [Fowlers 97-1]中讨论了在部署分层系统时所用到的技巧

已知应用

- 安全外壳[SSH]包含了与 X11 会话安全交互的协议，它在 TCP/IP 连接上使用 RSA 加密来保证安全
- SSL（Netscape Server）提供了安全访问层，客户端可以通过它来保证通信的安全
- Oracle 数据库提供了自己的安全访问层，应用程序通过这个层与 Oracle 数据库通信
- CORBA 安全服务[OMG]描述了在 CORBA 分布对象系统中如何使用验证，管理，审计和安全。任何 CORBA 应用的安全访问层都会与 CORBA 安全服务通信。
- Caterpillar/NCSA 金融模型框架[Yoder]使用了 ParcPlace's 的 VinsulWorks Smalltalk[OS 95]中的 LensSession 提供的安全访问层
- PLoP 的注册程序[Yoder & Manllescu 98]使用安全访问层访问系统。首先，应用程序运行在 Apache 服务器的 SSL 上。而且，所有用户注册的信用卡信息都是加密保存的。

一起工作

重构的问题

值得指出的是，上述的大部分模式都很难重构到一个已存在的系统中，特别是安全访问层，会话，受限视图。如果应用程序在开发时没有考虑安全因素，那么在后来要重构到安全访问层，会话和受限模式时就非常困难。如果预先创建了单点访问，那么在后来加入检查点就非常直接。如果角色被用于定义会话，而且角色是在检查点上被初始化的，那么以后增加新的角色就很容易。一个假的检查点可以被看作一个占位符，复杂的安全策略可以在后来添加进来。同样，如果所有的外部请求都通过某种形式的安全访问层来传递，那么这个安全访问层可以在后续的开发中被加强。由于存在这些问题，应用程序开发人员必须在开始设计基础架构时就考虑到安全的因素。在设计之初就考虑到安全需求也很有用，这样可以尽可能的避免以上所说的问题。

受限视图和带错的全视图

受限视图和带错的全视图时两个竞争的模式。受限视图用在登录时可以简化安全检查。应用程序的其他部分可以比较简单，因为不需要通过代码来处理安全异常。同样，这样也比较友好，因为用户不需要面对大量的错误对话框。带错的全视图容易实现。它可以简单到只弹出错误对话框或者把错误输出到标准错误上。带错的全视图可以用到当安全检查不能预先进行的情况中，出现这种情况的原因是因为检查耗费时间太长或者所需的必要信息还没有得到。

如果应用程序对用户操作的检查很少，那么带错的全视图提供了一种最简单的解决方案。带错的全视图同样在大部分选项对用户角色都是独立的情况下很合适。如果安全异常检查分布在整个应用程序中，那么，就应该使用带错的全视图。如果有些检查在受限视图中不能进行，这些检查依然可以单独处理并报告错误。因此，即使受限视图和带错的全视图对单个的安全检查来说是竞争模式，他们依然可以同时用在一个应用程序中来提供“报告最少错误的受限视图”

一起工作

现在来看所有的模式，你也许会问，“如何把他们放在一起工作?”。所有的模式都可以作为应用程序的一个安全模块并且提供一种与外部通信的机制。图 8 显示了这些模式如何一起工作。

当用户登录系统时，单访问点得到用户信息。单访问点使用检查点，检查点与安全访问层交互来验证用户信息。验证用户信息后，检查点查找用户角色并创建一个会话。会话有一个角色的引用供将来使用，并且可以根据用户的权限来定义受限视图。

检查点可以被需要安全操作的其他应用程序组件使用，这些操作不能在系统启动时执行。如果系统的任何一个部分需要全局信息比如角色，状态或受限视图数据时，它们可以引用会话。用户界面，包括带错的全视图或受限视图，都通过会话来创建。

在普通的操作中也可以调用检查点来完成安全检查，如果有必要，它们也可以使用安全访问层。

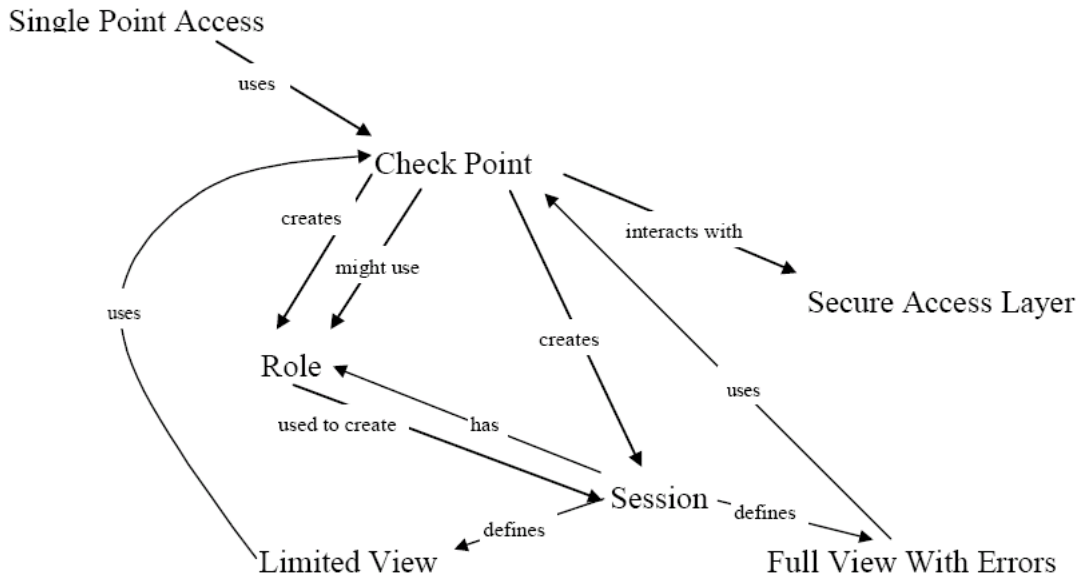


图 8 模式交互图

图 9 显示了一个用户正常登录的典型场景中所需的步骤。登录界面可以是单访问点。它通过检查点来初始化系统。检查点验证用户并创建用户的一组角色。用户验证通过安全访问层完成。角色可以用于创建应用程序使用的当前会话。这个会话根据用户角色和检查点中的用户密码来把自己初始化成一个受限视图。任何请求都会被传递给受限视图和检查点，然后检查点会把它传递安全访问层，接着安全访问层把结果返回给受限视图，并确认受限视图的数据被维护着。带错的全视图可以用于补充受限视图的缺陷。

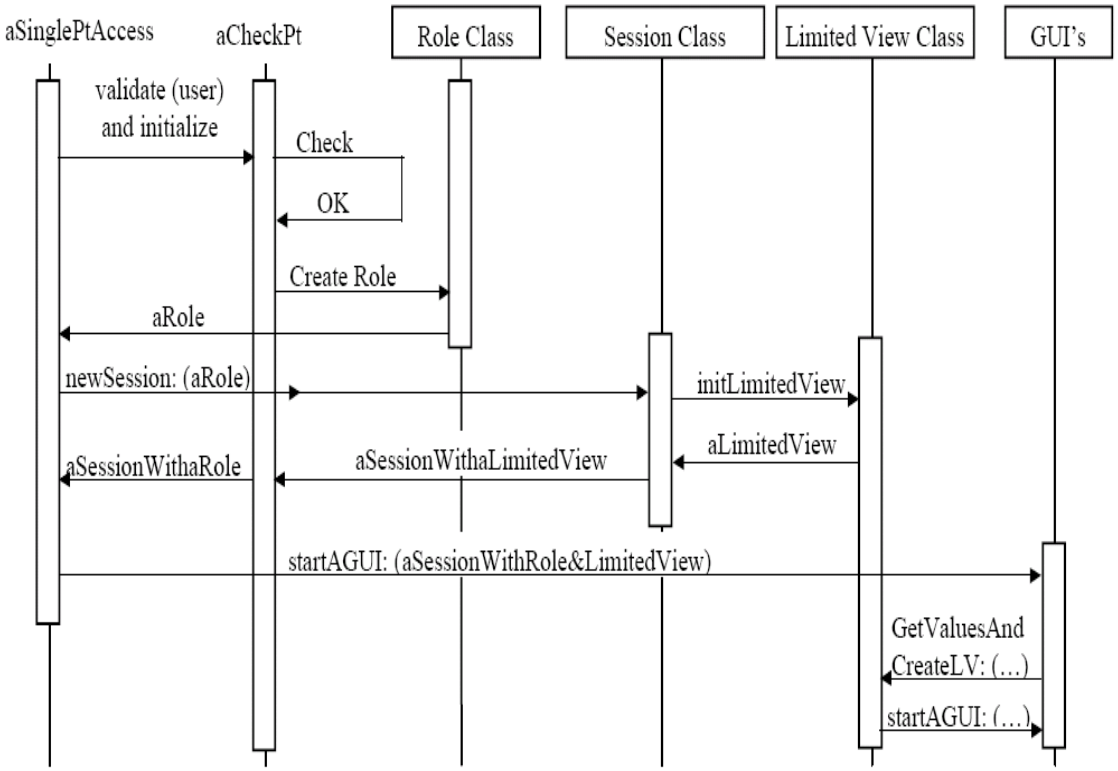


图 9 模式协作图

致谢

我们非常感谢 Johnson's 教授的模式研讨会的成员：Federico Balaguer, John Brant,

Ian Chai, Brian Foote, Alejandra Garrido, John (Zhijiang) Han, Peter Hatch, Ralph Johnson, Ral Lu, LewisMuir, Dragos Manolescu, Eiji Nabika, James Overturf, Ed Peters, Chieko Shirai, Rick Kirchgesner, andImad Zorob。Dirk Riehle 和Eugene Wallingford,审阅了早期的草稿并提出了有价值的反馈。Federico Balaguer, Eric Evans, Martin Fowler, Robert Haugen, Joshua Kerievsky, Eiji

Nabika, Tim Ottinger, Dirk Riehle, and Wolf Siberski, 在我们的PLoP 97作者会议上提供了非常好的建议。我们同样感谢Caterpillar, Illinois Department of Public Health, National Computational Science Alliance (NCSA)的员工和同事。Reuters Information Technology, and the University of Illinois给了我们支持和经验来撰写和开发这样一个系统。

参考资料

[Barcalow 97]Jeffrey Barcalow. *Strategic Planning Support for a Financial Model Framework*, M.S. Thesis, University of Illinois at Urbana-Champaign, Department of Computer Science, 1997. URL: http://www.uiuc.edu/ph/www/j-yoder/papers/thesis/barcalow.html

[Beck 97]Kent Beck. *SMALLTALK Best Practice Patterns,* Prentice Hall PTR, Upper Saddle River, NJ, 1997.

[Brant &Yoder 96]John Brant and Joseph Yoder. "Reports," *Collected papers from the PLoP '96 and EuroPLoP '96 Conference,* Technical Report #wucs-97-07, Dept. of Computer Science, Washington University Department of Computer Science, February 1997. URL: http://www.cs.wustl.edu/~schmidt/PLoP-96/yoder.ps.gz.

[BMRSS 96]Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal. *Pattern-Oriented Software Architecture: A System of Patterns*, John Wiley and Sons Ltd., Chichester, UK, 1996.

[Cunningham95]Ward Cunningham. "Checks*," Pattern Languages of Program Design*, James O. Coplien and Douglas C. Schmidt, eds., Addison-Wesley, 1995

[Foote &Yoder 96]Brian Foote and Joseph Yoder. "Evolution, Architecture, and Metamorphosis," *Pattern Languages of Program Design 2,* John M. Vlissides, James O. Coplien, and Norman L. Kerth, eds., Addison-Wesley, Reading, MA., 1996.

[Foote &Yoder 98]Brian Foote and Joseph Yoder. "Metadata and Active Object-Models *Collected papers from the PLoP '98 and EuroPLoP '98 Conference,* Technical Report #wucs-98-25, Dept. of Computer Science, Washington University Department of Computer Science, September 1998. URL: http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/

[Fowler 97-1] Martin Fowler. *Analysis Patterns: Reusable Object Models*, Addison Wesley, 1997.

[Fowler 97-2] Martin Fowler. "Dealing with Roles" *Collected papers from the PLoP '97 and EuroPLoP '97 Conference,* Technical Report #wucs-97-34, Dept. of Computer Science, Washington University Department of Computer Science, September 1997. URL: http://www2.awl.com/cseng/titles/0-201-89542-0/apsupp/roles2-1.html.

[GHJV 95] Eric Gamma, Richard Helm, Ralph Johnson, John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software,* Addison-Wesley, Reading, MA, 1995.

[GemStone 96] Gemstone Systems, Inc. *GemBuilder for VisualWorks, Version 5*. July 1996. URL: http://www.gemstone.com/.

[ICSP] *International Cryptographic Software Pages*. URL: http://www.cs.hut.fi/ssh/crypto/.

[Johnson &Woolf 97]Ralph Johnson and Bobby Woolf. “Type Object,” *Pattern Languages of Program Design 3,* Robert Martin, Dirk Riehle, and Frank Buschmann, eds., Addison-Wesley, Reading, MA., 1997.

[Lea 95] Doug Lea. ``Sessions: A design pattern", ECOOP Workshop on Patterns. August 1995.

[Lea 97] Doug Lea. *Concurrent Programming in Java,* Addison-Wesley, Reading, MA, 1997.

[Meyer 92] Bertrand Meyer. ``Applying `Design by Contract'," *IEEE Computer.* October 1992. pp 40-51.

[NCSA] NCSA HTTPd Development Team. *Mosaic User Authentication Tutorial*. URL: http://hoohoo.ncsa.uiuc.edu/docs-1.5/tutorials/user.html.

[OMG] Object Management Group. *Security, Transactions, ... and More.* URL: http://www.omg.org/corba/sectrans.htm#sec.

[OS 95] ObjectShare, Inc. *VisualWorks User’s Guide*. 1995. URL: http://www.objectshare.com/doc/vw/Default.htm.

[Schmidt, Price,& Harrison 97]Douglas C. Schmidt, Nat Pryce, and Timothy H. Harrison. “ThreadSpecific Storagefor C/C++ - An Object Behavioral Pattern for Accessing perThread State Efficiently”*Collected papers from the PLoP '97 and EuroPLoP '97 Conference,* Technical Report#wucs-97-34, Dept. of Computer Science, Washington University Department of Computer Science, September 1997. URL: http://www.cs.wustl.edu/~schmidt/TSSpattern.ps.gz.

[Schneier 95] Bruce Schneier. *Applied Cryptography, 2nd edition.* John Wiley & Sons, 1995.

[SSH] *SSH (Secure Shell) Home Page.* URL: http://www.cs.hut.fi/ssh/.

[SSL] *The SSL Protocol*. Netscape Communications, Inc., URL: http://home.netscape.com/newsref/std/SSL.html.

[Woolf 97] Bobby Woolf. “Null Object,” *Pattern Languages of Program Design 3,* Robert Martin, Dirk Riehle, and Frank Buschmann, eds., Addison-Wesley, Reading, MA., 1997.

[Yoder] Joseph Yoder. *A Framework to Build Financial Models*. URL: http://www.uiuc.edu/ph/www/j-yoder/financial_framework.

[Yoder &Manolescu 98]Joseph Yoder and Dragos Manolescu. *The PLoP Registration Framework*, 1998. URL: http://www.uiuc.edu/ph/www/j-yoder/Research/PLoP.