第 47 期（2005 年 3 月）

【新闻】

Ivar Jacobson

【方法】

【工具】

X 非程序员
X-Programmer
软 件 以 用 为 本

联系：think@umlchina.com

http://www.umlchina.com/

微软将在 VSTS 中集成 CMMI

[2005/3/9]

微软宣布在 VSTS 2005 中提供两套过程指南：MSF for CMMI 和 MSF for Agile Software Development。

VSTS 高级产品经理 Prashant Sridharan 还宣布，除了这两套之外，微软还将进一步集成 ISO 和极限编程（XP）等过程指南。

（自 UK Comment Wire，think 摘译，不得转载用于商业用途）

征稿

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

Genuitec 宣布 MyEclipse 4.0

[2005/3/3]

Genuitec 宣布了 MyEclipse Enterprise Workbench 4.0，更智能、更快捷、更简单和更便宜的 J2EE 工具的新版本的诞生。它的价格对于个人和企业开发人员来说都是非常有吸引力的。这是 J2EE IDE 市场一个重量级的选手。通过增加 UML 双向建模工具、WYSIWYG 的 JSP/Strutsdesigner、可视化的 Hibernate/ORM 工具、Spring 和 Web services 支持，以及新的 Oracle 数据库开发，MyEclipse 4.0 继续为业界提供全面的产品。

Genuitec 总裁 Maher Masri 说，“今天，MyEclipse 已经提供了意料之外的价值。其中的每个功能在市场上单独的价格都比 MyEclipse 要高。但是，我们承诺为顾客提供全面并且可以买得起的解决方案。按照这个传统，我们的顾客将继续享受年度订购活动的好处，该活动提供了所有发布功能的入口以及伴随 MyEclipse4.0 一起的专业技术支持。”

MyEclipse 4.0 计划于 2005 年 4 月中旬正式发布。

Genuitec 赞助了一个新的技术项目，目的是使 Eclipse 可以在通用的方法中使用可选的操作系统相关的服务和技术，并提供了扩展点和 API 供开发人员使用和定制。MyEclipse 的开发 VP 和产品主管 ayne Parrott 指出，“该项目对 Eclipse 和 RCP（译者注：Rich Client Platform ）来说都是一个新的高地，通过利用这些服务和技术，这个项目将不仅可以从 RCP 开始继续 Java 在桌面上的重建，也将通过将 Eclipse 从 Java-only 的情况中释放出来从而诞生今天所无法想象的新的技术和服务。”

Genuitec公司基于Eclipse提供各种技术产品、培训和咨询服务。关于Genuitec公司的更多情况可以访问 www.genuitec.com。

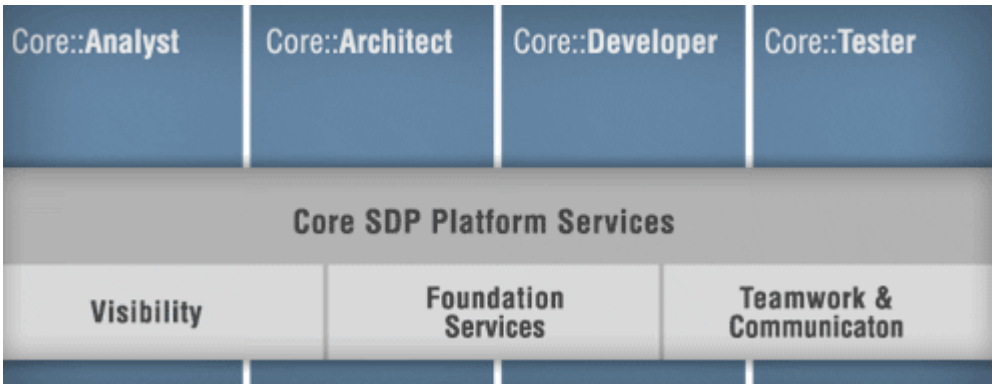
（自 prweb，袁峰 摘译，不得转载用于商业用途）

Borland 升级在 Eclipse 的会员资格

[2005/3/1]

应用生命周期管理（application lifecycle management，ALM）领域的专家 Borland 现在成为了 Eclipse 委员会（board）的战略开发者的一员，为此付出了$250,000 每年一席位，过去 Borland 在这个组织中的身份只是一个插件提供者。Borland 将为 Eclipse 提供 8 名开发人员。

在一个为 Eclipse 的 ALM 项目 roster 增加建模能力的项目中 Borland 担任了领导职位，同时，将 Borland 核心的软件开发平台（core SDP）中的各种工具应用在 Eclipse 丰富的客户架构上的计划也在进行中。Core SDP 包括 Together、CaliberRM 和 Optimizeit。



Borland 是投身于 Eclipse 怀抱的最新成员。上周，J2EE partner－BEA 公司－也宣布了他们的加入，也是作为战略级开发者，在 Web 工具项目（WTP）中联合担任领导职位。WTP 面向于简化使用 J2EE 的 Web 应用开发。

此前三年，Borland 一直徘徊在 Eclipse 组织的边缘，角色是一个插件提供者。Borland 在 Eclipse 中最初的会员资格得来得非常意外，它来自于对 UML 专家工具 Together 的收购，后者是 Eclipse 的奠基者之一。

对于在奠基人以及 Java 工具对手 IBM 掌控下的 Eclipse，Borland 并不乐意加入。但是，一年前 Eclipse 已经成为了一个独立组织，并成功吸引了 SAP、Intel 等公司的加盟，它们的积极参与淡化了 IBM 在该组织中的影响力。

Borland 产品市场部的高级副总裁 Erik Frieberg 谈到了这个变化：“Eclipse 过去为 IBM 所控，但今天已经成为了一个典型的行业组织，我们觉得现在是在最高层次参与进去的时机了”。

在 Eclipse 项目中更深的介入将潜在地提升 Borland 的市场。将 Core SDP 置于 Eclipse 的客户架构中将使得第三方为 Eclipse 所写的插件可以很好的集成进来，扩充 Borland 现有的功能。Eclipse 也意味着 Borland 的工具可以潜在地和兼容 Eclipse 架构的环境如 SAP 的 NetWeaver 等合作，而无需大量昂贵、耗时的自定义工作。

Frieberg 认为 Eclipse 的良好进展使得 Borland 加入进去变得非常重要。“我们需要在 Eclipse 上有一个更强大的姿态”。

Eclipse 的执行总裁 Mike Milinkovich 认为 Borland 的加盟有助于加强并支持 Eclipse 在应用生命周期管理方面的能力和价值。

他还补充，Borland 的出现也有助于加强 Eclipse 作为一个独立组织存在的信誉。

Borland 的这个决定导致 Sun 成了主要的 Java 开发商中唯一一个徘徊在 Eclipse 之外的了。但 Sun 并不乐意加入，而是继续支持它自己已经相当受限的 NetBeans 开源项目。

（自 cbronline，袁峰 摘译，不得转载用于商业用途）

Ivar Jacobson 中国行

[2005/3/23]

UML 创始人之一 Ivar Jacobson 于 3 月中下旬来华。

3 月 21 日下午，在机械工业出版社的组织下，Ivar Jacobson 在北京和翻译他的书籍的译者们作交流。

3 月 23 日上午，在北京燕翔饭店 3 楼多功能厅，Ivar Jacobson 做了主题演讲：Unified Foundation；

3 月 23 日下午，在 CSDN 会议室，Ivar Jacobson 和来自各企业的技术主管举行了圆桌会谈，就软件工程的最新方展趋势和软件工程师的职业发展，展开了深入的讨论。现场还安排了 20 多个观众席位。整个会谈共分为三个议题：软件工程领域技术发展趋势（潘加宇主持）、软件工程师职业发展道路（熊妍妍主持）、软件咨询业的发展机遇（青润主持）

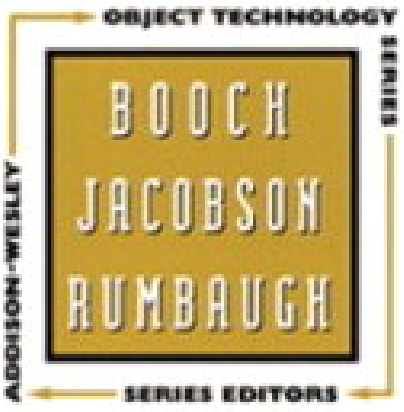
（本图片摘自 CSDN）

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

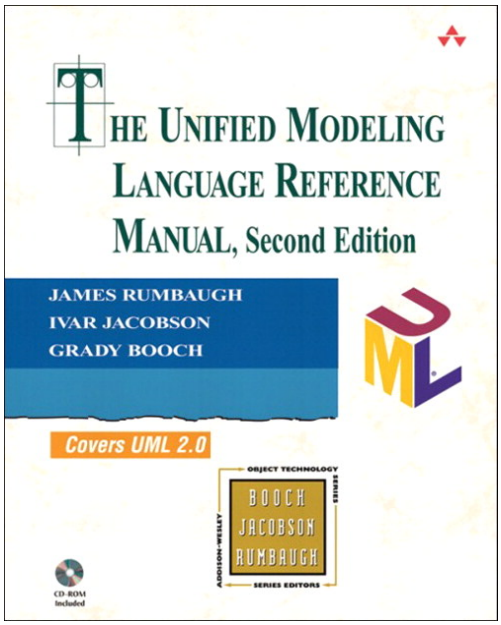
《UML 参考手册》2.0 版中译本
即将由机械工业出版社华章分社出版

字斟句酌看 UML 变迁（上）

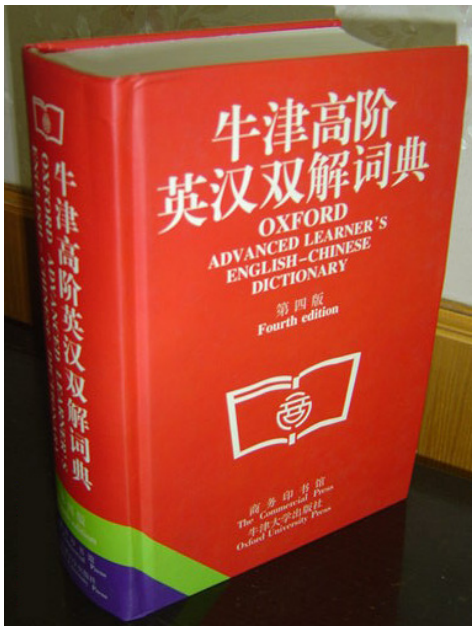
潘加宇、李嘉兴 文

讨论

由UML三位创始人James Rumbaugh、Ivar Jacoboson、Grady Booch合著的《UML参考手册》对于使用UML建模的开发人员来说，就像学英语的《牛津英汉词典》，虽然不会随身携带，架上却是必备的。2000年，《UML参考手册》第一版中译本【1】推出。今年上半年，面向UML2.0的第二版中译本【2】也即将推出，我们在审校译稿的过程中，萌生了这个有趣的想法，把两本书一些有意思的前后变化对比一下，从中看看UML的变迁。



=



本文中笔者只针对此书的某些片段前后版本的不同作出比较和阐述，并不是系统的全面的比较。对最新的UML2.0规范和元模型感兴趣者可以到http://www.omg.org/technology/documents/modeling_spec_catalog.htm#UML下载。另外，这也不是一篇书评。

前言

第一版：不过为了快速学习UML，他们（注：读者）会发现阅读《用户指南》很有帮助。to learn UML quickly, they may nevertheless find it useful to read the User Guide.

第二版：如果需要一本展示如何对一些常见问题建模的入门指南，可参考《UML用户指南》和《UML精粹》。

For a tutorial introduction to UML that shows how to model a number of common problems, see The Unified Modeling Language User Guide [Booch-99] or UML Distilled [Fowler-04].

“快速学习UML”变成了“入门指南”，我猜想大师也许感觉到UML背后的博大精深，快速学习谈何容易？把“对一些常见问题建模”作为入门，够用就好的原则，才是团队起步使用UML的正道。

另外我们可以注意到，推荐书籍中添加了 Martin Fowler 的《UML 精粹》。对于刚听说“UML”这个字眼的开发者，面对复杂丰富的规范，经常会不知从何处下手了解，《参考手册》显然不合适，于是经 Martin Fowler 和 Kendall Scott“Distilled”之后得到的这本小书对推广 UML 起了很大的作用。《UML 精粹》在国外不断重印，也说明开发者用钞票为它投了票。善于总结和普及是 Martin Fowler 的特点，他的其它书如《重构》、《企业应用架构模式》等也很畅销。虽然其中知识可能不是他最先发明的，但在推广方面，他却取得了巨大的成功。Fowler 的这种能量显然征服了三位大师。

第一版：首先，我们必须感谢Rational软件公司，特别是Mike Devlin和Paul Levy……

第二版：自从UML1.0以来，向UML规范贡献自己思想的人员数量不断增加……如果算上那些其工作对UML有影响的人，这个数字还会更多。现在已经不可能制定出一份完整的名单了。

UML 早已不只是 Rational 的 UML。在 UML2.0 的道路上，参与的人和组织越来越多。ADTF（http://adtf.omg.org/）的两位主席，James Odell 是 Agentis 公司的，Cris Kobryn 就是 Telelogic 公司的。UML1 之后的几年内，UML 相关的工具就增加到了 100 多种【4】，很多公司比 Rational 更早推出支持 UML2.0 的工具。Gentleware 在 2003 年 9 月就推出第一个支持 UML2.0 的工具 Poseidon for UML 2.0，该产品基于开源的 ArgoUML。还有 Telelogic、iLogix、Borland 等，推出时间都比 Rational 早。

第一部分 UML 概述

第二版添加了“UML2”和“其他来源”段落。

“UML2”简要描述了UML2针对于UML1的重要改变。而UML2相对于UML1，一些图还受到了其他领域的影响。UML2序列图的符号标识来自ITU制定的消息序列图（MSC）标准[ITU-T Z.120]。这个标准在电信行业被广泛地使用，它替换了UML1中序列图的符号标识，增加了许多结构化元素以解决UML1中出现的问题。UML2中结构化类元（structured classifier）的概念受到了SDL[ITU-T Z.100]、MSC和ROOM[Selic-94]方法中实时工程概念的巨大影响。活动图的符号标识受到了各种业务建模符号标识的影响。

第二版还添加了“UML的复杂性”和“UML评价”段落。

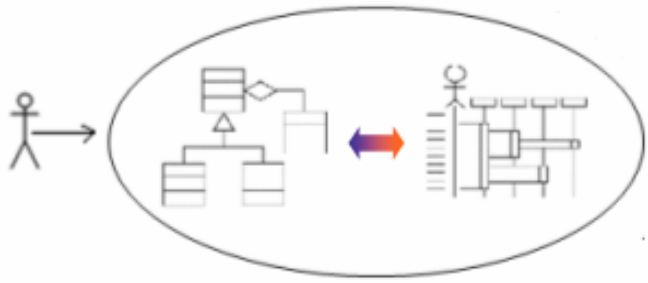
UML 问世以来，常常被人们指责为太复杂，作者特地添加文字对此作出评论。

要点 1：软件的复杂性本来就是固有的，如果做成玩具，UML 也会很小，但 UML 的开发者们希望它不只是一个玩具。

2004 年 6 月，Grady Booch 在一次访谈中就曾经反驳微软的“UML 太复杂”的评价。Grady Booch 认为把 UML 元模型拿出来说 UML 复杂，是不公平的。打开一个操作系统或任何一项技术的内部，都是非常复杂的，元模型相当于 UML 的内部，对于工具开发商来说非常重要，但 UML 的实际使用者根本不需要看到这些。（http://www.devsource.com/article2/0,1759,1608940,00.asp）

要点 2：你不必知道或者使用 UML 的每一项功能，就像你不需要知道或使用一个大型软件或编程语言的每一项功能一样。有一些被广泛使用的核心概念。其他特性可以逐步学习，在需要时再使用。

2004 年 7 月的一条新闻中，微软曾对UML的使用做过调查，UML类图、顺序图、用例图这三种图是使用得最多的。（http://www.cbronline.com/article_news.asp?guid=CAB30E09-41A7-4A47-9811-8D18FAD19403）



UML 一览

两个版本都是一开始先用一张表格列出了各种UML视图和相关的图。我们可以对比一下：

主要区域	*视图*	*图*	*主要概念*
结构	静态视图	类图	类、关联、泛化、依赖、实现、接口
	用例视图	用例图	用例、执行者、关联、扩展、包含、用例泛化
	实现视图	组件图	组件、接口、依赖、实现
	部署视图	部署图	节点、组件、依赖、位置
动态	状态机视图	状态机图	状态、事件、转换、动作、
	活动视图	活动图	状态、活动、完成转换、分叉、结合
	交互视图	序列图	交互、对象、消息、激活
		协作图	协作、交互、协作角色、消息
模型管理	模型管理视图	类图	包、子系统、模型
扩展	所有	所有	约束、构造型、标记值

UML视图和图（第一版）

主要区域	*视图*	*图*	*主要概念*
结构	静态视图	类图	类、关联、泛化、依赖、实现、接口
	设计视图	内部结构	连接器、接口、部件、端口、供给接口、角色、需求接口
		协作图	连接器、协作、协作使用、交互、角色
		组件图	组件、依赖、端口、供给接口、实现、需求接口、子系统
	用例视图	用例图	用例、执行者、关联、扩展、包含、用例泛化
动态	状态机视图	状态机图	状态、事件、转换、完成转换、效果、执行活动、区域、触发
	活动视图	活动图	动作、活动、控制流、控制节点、数据流、分叉、结合、异常、对象节点、扩展区域、引脚
	交互视图	序列图	交互、消息、信号、生命线、出现说明、执行说明、交互片断、交互操作域
		通信图	协作、消息、角色、序列数、监护条件
物理	部署视图	部署图	节点、工件、显现、依赖
模型管理	模型管理视图	包图	导入、包、模型
	特性描述	包图	约束、构造型、标记值、特性描述

UML视图和图（第二版）

笔者用红色标出了发生变化的地方。

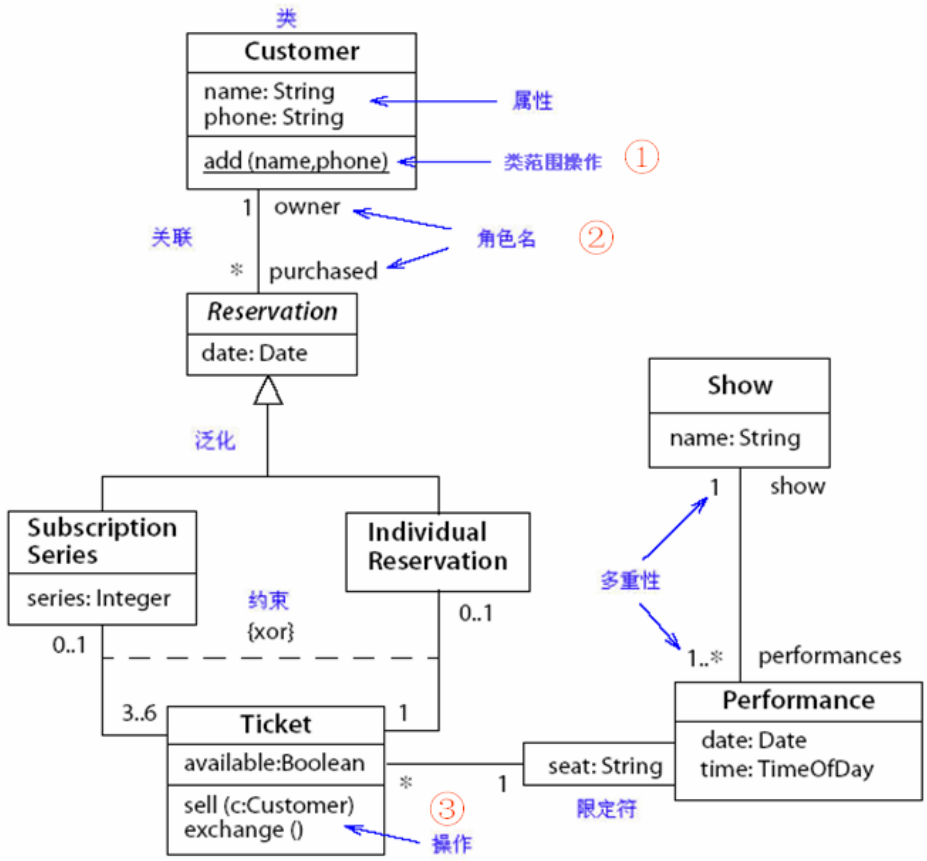
部署视图单独分到物理区域。设计视图取代了实现视图，模糊了设计和实现之间的界限。

两个表对比，增加了两种图：内部结构（Internal Structure）图和通信（Communication）图。通信图就是原来的协作图，而协作图依然存在，但有了新的含义。其实还有一些新增的图，表中未列出来，包括：时间（Timing）图、交互概念（Interaction Overview）图、协议状态机（protocol state machine）图，它们可以看作是序列图、通信图、状态机图的变体。

****

“UML一览”部分简要介绍了UML的各种图，每种图都给出了一个售票系统的图例，第二版和第一版用的例子相同，所以给了我们一个很好的机会来比较印证。作者在书中只是把图重新画出来，并没有对其中的变化作出说明。下面，笔者将以第一版的顺序为序，从使用的角度针对这一章中的例图来比较图中的微妙差别，并分析其中的变化，也算是一种学习的心得吧。

类图



第一版类图

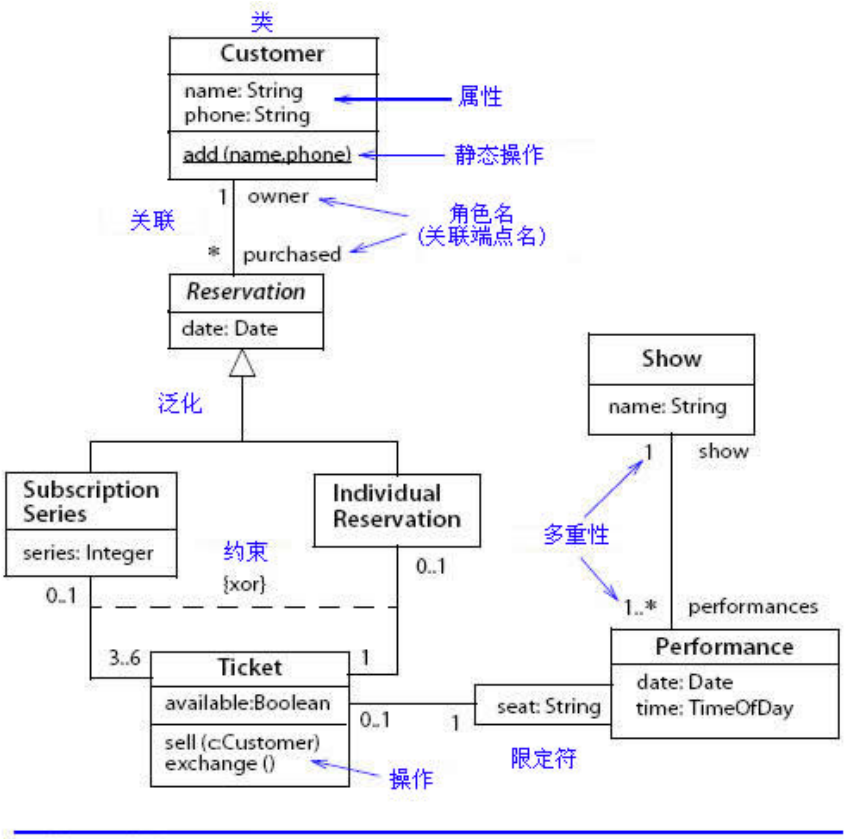


图3-1. *类图*

第二版类图

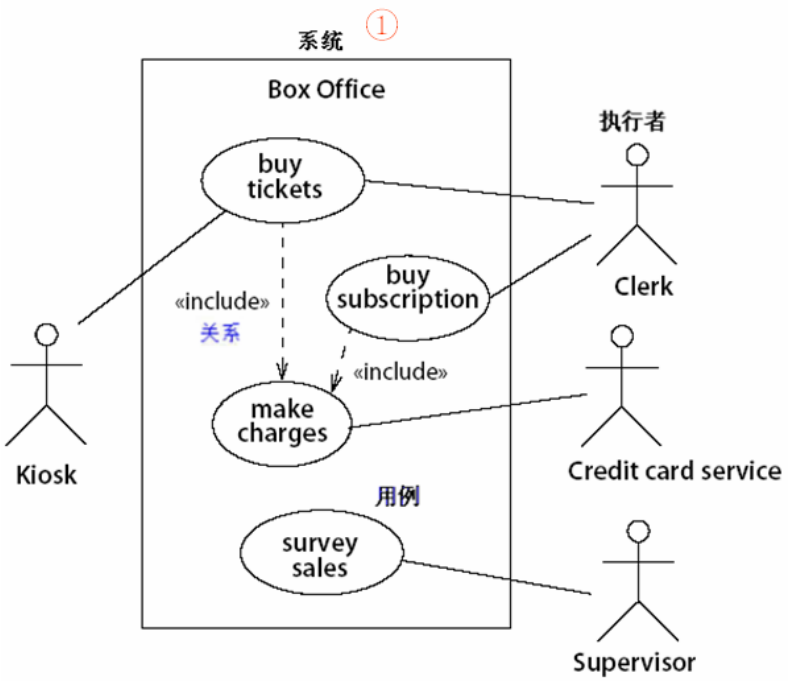
①Customer类的带下划线的操作，原来的注解是“类范围操作”（class-scope operation），现在是“静态操作”（static operation）。UML1中的源范围（source scope）和目标范围(target scope)概念已被大家更熟悉的静态特性（属性、操作）所取代。

②“角色名”下面括号加注了“关联端点名”。角色名（rolenames）是关联端点（association end）的名字，关联端点除了名字还有其他特性：例如可见性（visibility），多重性（multiplicity）。多重性是最常被关注的。

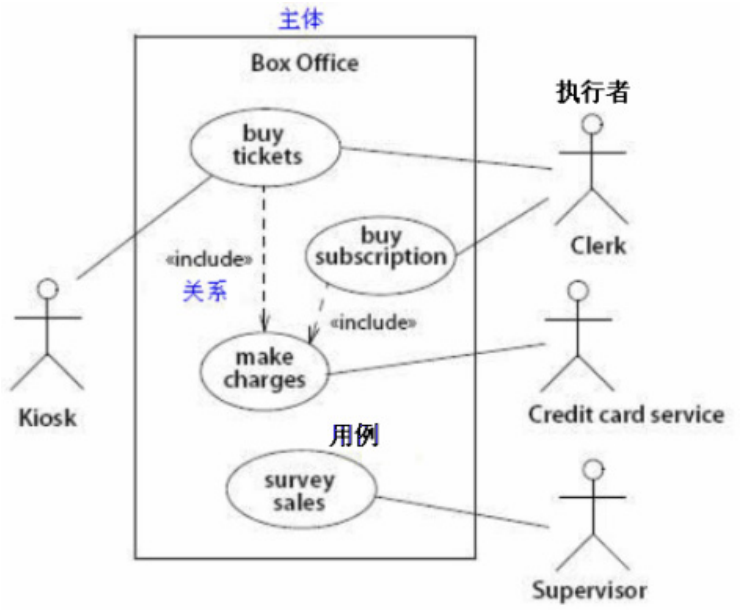
③Ticket（票）和Performance（演出）之间的关联，第一版的多重性是*，第二版改正为0..1。因为这是一个限定关联，seat是一个限定符。一场演出可以有很多张票，但针对一个座位而言，一场演出只能有0..1张票。限定符可以看作一个关联的遍历索引（可以对照数据库中的“索引”来理解），暗示着Performance要找Ticket时，可能是通过seat来找的。

总体来说，类图在UML2中变化不大。

用例图



第一版用例图



第二版用例图

①唯一的变化是“系统”（System）变成了“主体”（subject）。我们来比较一下图下的说明：

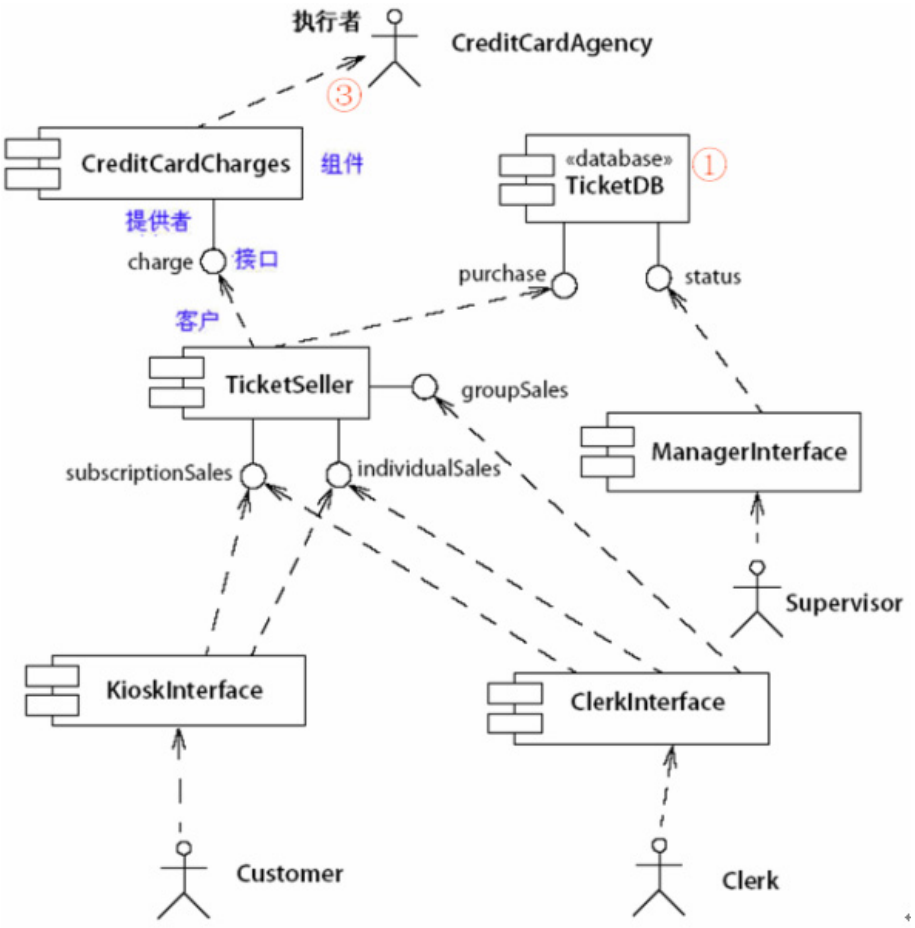
第一版：The use case view models the functionality of the system as perceived by outside users, called actors…. **用例视图对被称为执行者的外部用户所观察到的系统功能建模**

第二版：The use case view models the functionality of a subject (such as a system) as perceived by outside agents, called actors…**用例视图对被称为执行者的外部智能代理所观察到的主体（比如一个系统）功能建模**

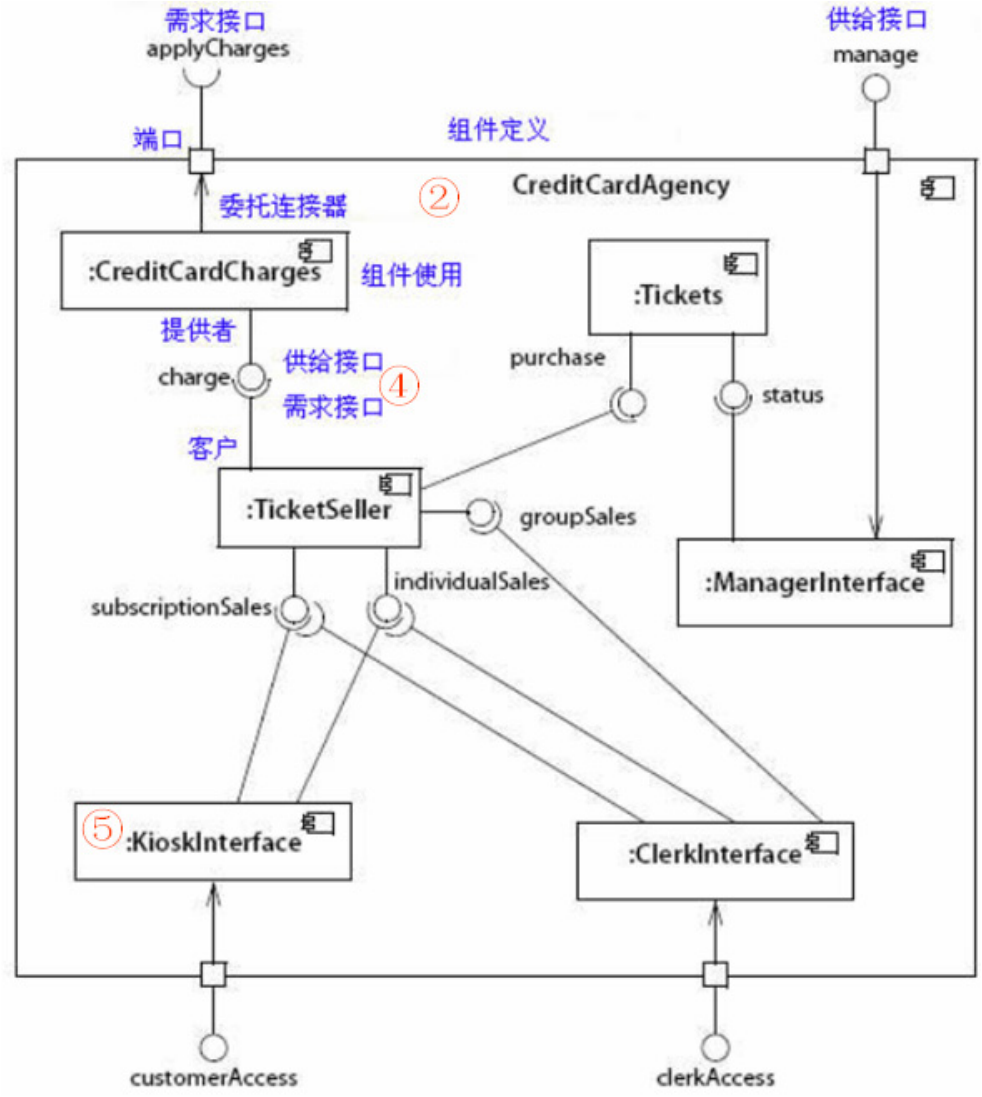
执行者不只是“用户”，用例也不止用于“系统”，所以Agent代替了User，Subject代替了System，使用例和执行者的概念更加广义了。用例不止可以用于“系统”（软件系统），也可以用于一个组织（业务用例，业务执行者），一台设备（**用例，**执行者）……

用例图在UML2中也变化不大。

组件图



第一版组件图



第二版组件图

①那个《database》不见了。在UML1中，组件可以用来表示物理结构，象数据库、DLL、EXE、JSP…等，在UML2中，这样的表达由部署图中的工件承担。组件的重点已经从UML1中的物理视图转向了更加逻辑的概念，这样它们就能够在概念模型中使用。

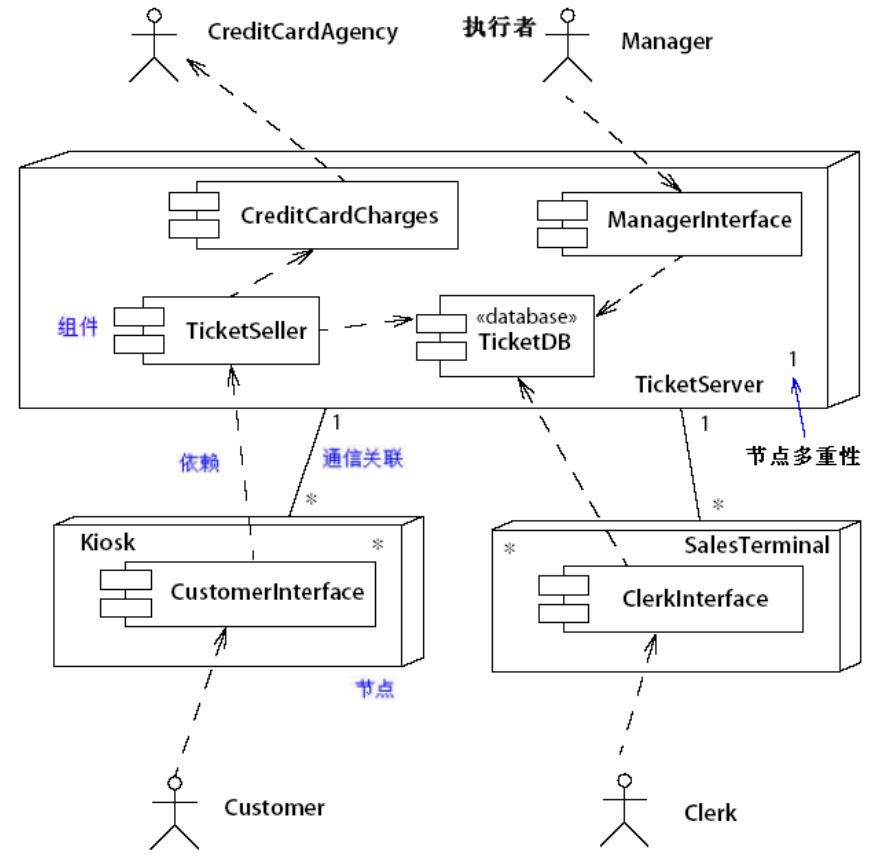
②组件分成了组件定义和组件使用两部分来描述，把内和外分开了。

③四个执行者没有了，变成了四个端口（Port），例如：和Supervisor执行者对应的是manage端口，和Customer执行者对应的是customerAccess端口。组件只定义自己的“协议”，不把执行者拉进来是合适的。端口是UML2中出现的新概念，它可以看作是一些经常一起使用的供给接口和需求接口的组织（可以对照PC机上的硬件端口来理解）。

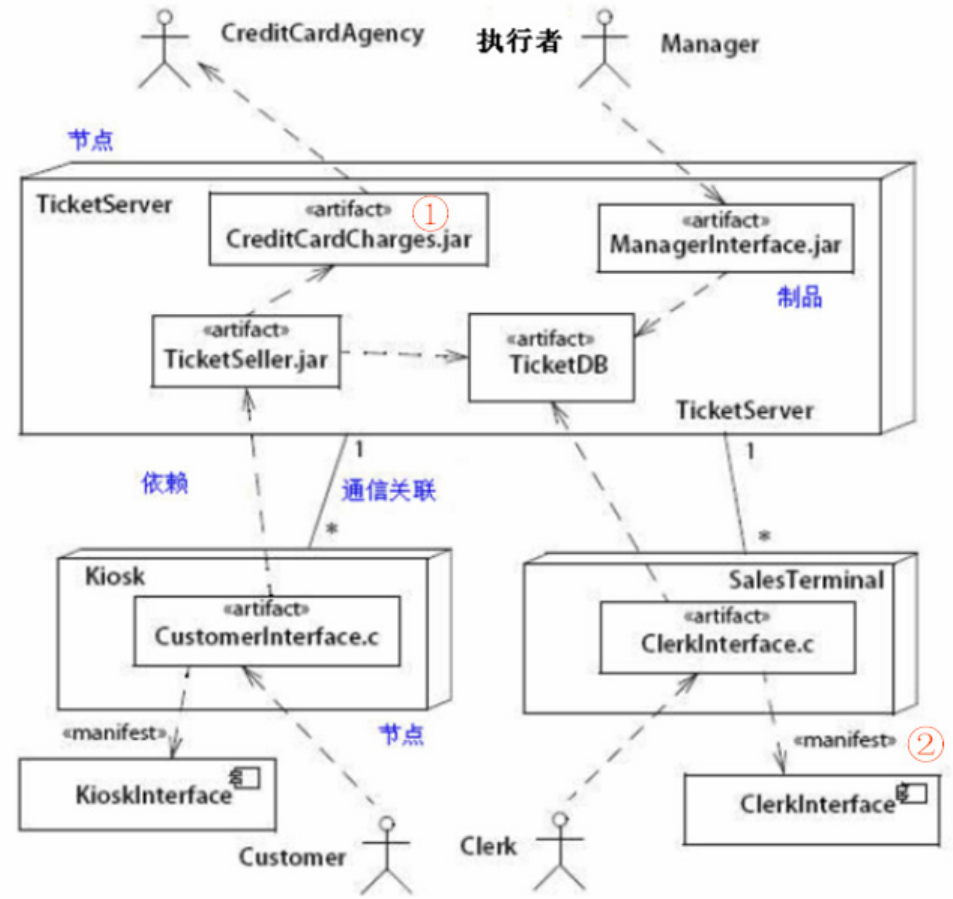
④接口变成了需求接口和供给接口。需求接口表示组件需要的服务，供给接口表示组件提供的服务。如果一个组件有一个供给接口，另一个组件有相同的需求接口，这两个组件就可以“无缝集成”，无需另外的结构。这些改变可能和基于组件的开发（CBD）相关。组件代表着一个独立开发、独立购买的，可以置换和按需要组装的单元，就像人们到五金店买零件组装一样。UML2更强调这个特点，并不在意组件是用面向对象方法或者面向过程方法构造的，甚至，不在意它是软件还是硬件。

⑤KioskInterface前面有个冒号，说明“KioskInterface”也是一个类。在UML2中，组件图和普通类图之间没有严格的界限，也不再有单独的一个图标来表示组件。KioskInterface的框框就是一个类框，组件的标志缩成一个小图标放在右上角，也可以用构造型《component》取代图标。

部署图



第一版部署图



第二版部署图

①多了一个《artifact》的构造型。部署在UML2中被重新定位，被应用于工件而不是模型元素。UML1组件图中的“TicketDB”，作为一个工件在这里出现了。另外，上面UML1组件图中被取消的执行者，也在部署图中出现了。

②多了组件和工件之间的《manifest》关系。UML2把模型元素和工件（artifact）分开了，而追踪模型元素与实现它们的工件之间的对应关系非常重要，把它们联系起来的就是显现（manifestation）。显现的含义是：将一个模型元素物理实现为一个工件。在软件中，模型最终会被实现为一组工件，诸如不同类型的一些文件。工件是被部署到物理节点（node）上的实体，物理节点可以是计算机或存储设备等。

参考文献

[1]James Rumbaugh、Ivar Jacobson、Grady Booch 著，姚淑珍、唐发根等译，《UML参考手册》（第1版），机械工业出版社，2001/1

[2]James Rumbaugh、Ivar Jacobson、Grady Booch 著，UMLChina王海鹏、李嘉兴 译，《UML参考手册》（第2版），机械工业出版社，2005/5

[3] Martin Fowler、Kendall Scott 著，徐家福 译，《UML精粹》（第2版），清华大学出版社，2003/6

[4] http://www.umlchina.com/Tools/Newindex1.htm

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

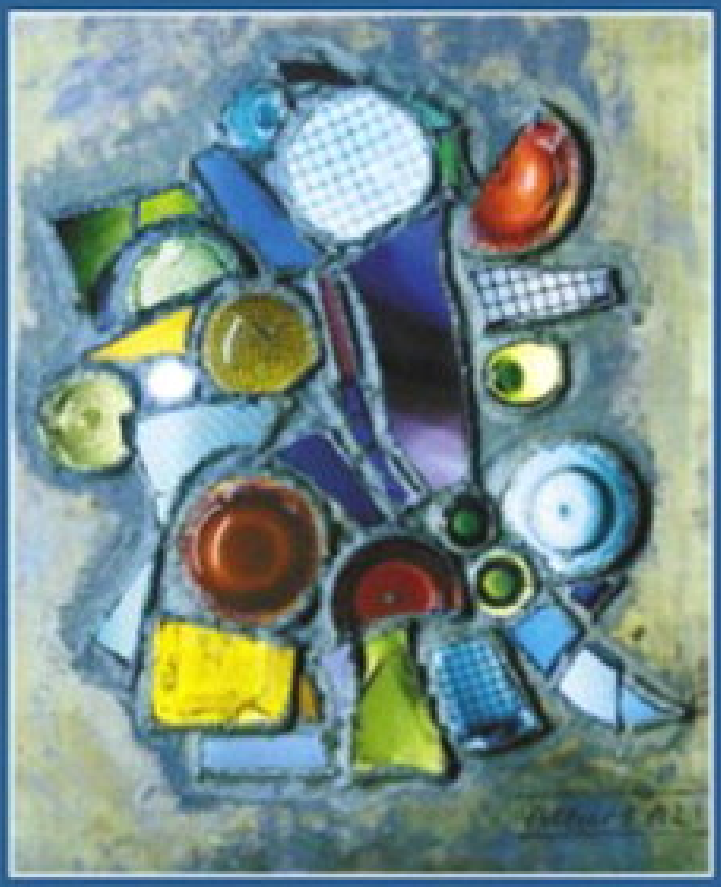
Foreword by Martin Fowler

《领域驱动设计》中译本

即将由清华大学出版社出版，UMLChina 审稿

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

用 UML 为系统的系统建模

Rob Cloutier、Andrew Winkler、John Watson,、Clay Fickle 著，徐异婕 译

讨论

摘要

如今，随着系统的复杂程度日益提高，产生了“系统的系统（Systems-of-Systems）”这一概念。传统系统工程（System Engineering）中，为复杂系统进行建模的方法是进行功能分解，从而确定系统的每个主要功能；接下来，一旦主要功能被确定，就再使用同样的方法将大的功能划分为小的功能。该建模方法倾向于从工程师视角，而不是用户视角来描述系统。这种做法有失偏颇，因为它可能导致产生一个技术复杂、而未必满足用户需求的系统。相反，本文介绍了一种现代面向对象系统工程（object oriented systems engineering ，OOSE）方法，来解决“系统的系统”相关的规格说明及分解问题。该方法能够确定在多种不同情况下，用户期望系统应执行何种任务；而对于这些情况，该方法创建一个用例模型来捕捉各种场景（scenario），以及创建一个类模型来捕捉领域信息和功能。上述两种 UML 模型提供了充分的信息，从而，可以将系统的一部分创建成子系统，而每一个子系统都由它们自己的一组用例概述（survey）和领域信息类概述来描述 [译者注：注意，此处以及本文后面出现的“概述”，均为专有名词，可参考 2.2 节的概念讲解] 。这些概述构成了子系统的规约（specification）。所有子系统用例概述共同实现系统级用例，这种识别子系统的技术可以层层迭代地进行下去。

1.0 介绍

当对系统的系统进行设计细化时，需要创建多个模型。理解这些模型的层次、以及不同模型之间的关系非常重要。顶层模型称之为“企业模型”，或者叫“系统的系统模型”。该模型展示了将被集成在一起的“系统模型”之间的关系。要创建的下一个模型是“系统模型”本身。“系统模型”详细说明了系统的全面信息和功能，它还允许架构师将功能分配到各个子系统中，而各子系统从逻辑上组成了该系统。最后建立“子系统模型”，识别出每一个子系统的服务、组件和类。图 1 描述了上述三种模型的层次关系。对于读者来说，需要理解的很重要的一点是，根据系统的复杂程度不同，子系统可能需要进行更进一步的逻辑分割，分解为更小的子系统，我们可以称它们为片（segment）、以及子片（sub-segment）等等。建立上述几种模型的目的是，获得一个可管理的系统定义，我们利用了复杂性作为分割设计的依据，而不是直接将复杂性展开。

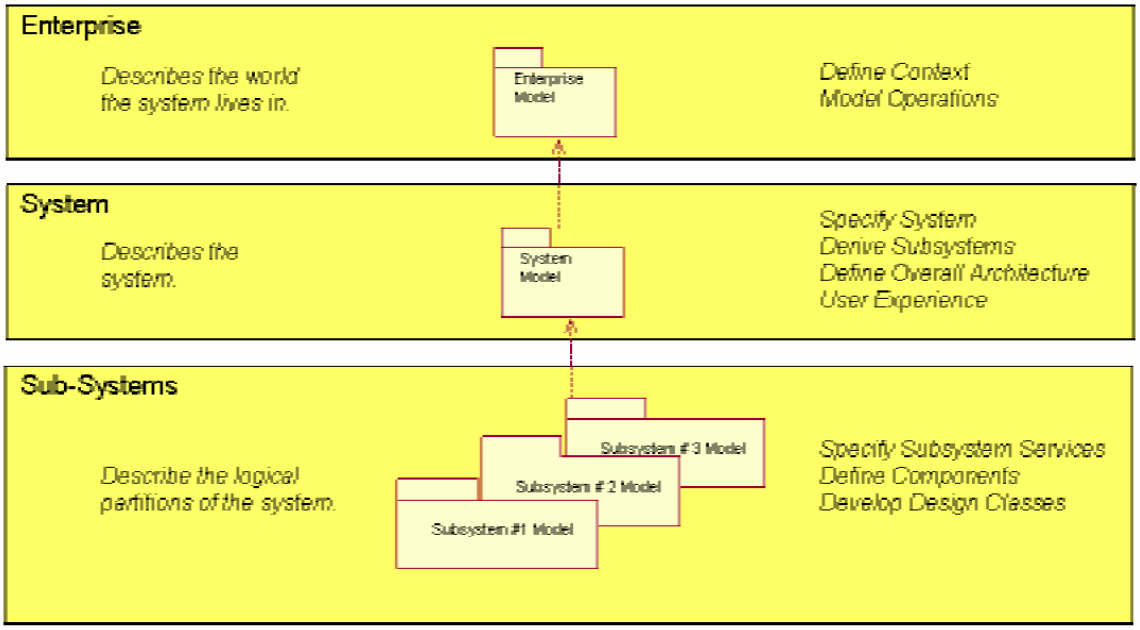


图 1 模型层次

本文要讨论的步骤如下。这些步骤是按照“从系统模型中提取子系统模型（即用例和类这两者的概述）”的方式来组织的。可以遵循同样的步骤，从企业模型中提取系统模型，当然，实际上更常见的情况是，应用的系统边界是预先定义好的。即便如此，创建企业模型依然非常重要，它可以作为提取系统用例和生成最初领域类的基础。

1. 定义系统模型上下文（context）

2. 开发系统用例和领域类模型

a. 建立系统用例概述

b. 用类模型捕捉领域“语言”

c. 用活动图细化系统用例

d. 用领域对象实例增强活动图

3. 用领域对象的方法绘制序列图，来细化上述活动图的内容

4. 考察顺序图中类的内聚性、和子系统的松耦合关系，从而把领域类分组，形成子系统的类概述（class survey）

5. 识别序列图中的子序列（subsequence），从而提取出子系统的用例概述（use case survey）

6. 根据用例概述中的执行者（actor），产生子系统的上下文图（context diagram）

前两步详细地定义了系统模型，接下来的三步将系统模型扩充，从而，识别子系统，并通过用例概述和领域对象概述定义每个子系统。

我们以一个在海军战役组的水面舰艇系统作为企业模型的案例。它是一个系统的系统，包括武器系统、传感系统、全体人员、以及指挥与控制系统（Command & Control System，C2 System）。对我们的案例而言，指挥与控制系统是要开发的系统，在开发过程中，除全体人员之外，现有的系统和系统接口都必须保持不变。图 1a 中说明了该海军舰艇系统。我们将讨论如何使用企业模型，产生指挥与控制系统的用例概述、领域类概述、和上下文图（context diagram）。然后我们将讨论，如何使用同样的步骤，来识别指挥与控制软件系统的一系列子系统。

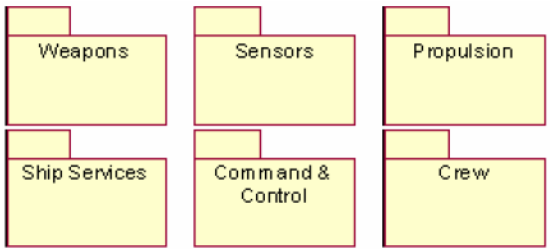


图 1a 系统的系统——海军水面舰艇系统

2.0 定义企业上下文（enterprise context）

本方法要求首先理解系统的系统——即企业，而系统是在其中运行。为此，要创建企业上下文图。在企业上下文图中，被建模的企业位于图的中央，通过链（link）和外部系统相连。图 2 所示的是一个简单的企业上下文图，它明确了系统的边界。

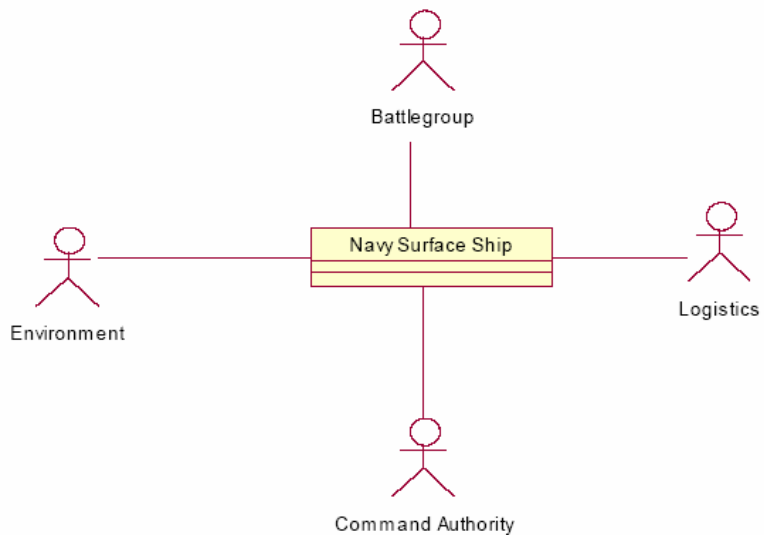


图 2 企业上下文图

2.1 开发企业用例和领域对象模型

企业用例模型和领域对象模型的开发，是紧密联系、互相交叠的活动。但是，这个过程是以开发一个指挥与控制系统的企业用例概述“开始”的。然后，通过活动图来描述用例细节，并建立领域对象模型。最后，通过领域对象流（domain object flow）增强活动图。

2.2 企业用例概述

用例概述（use case survey）是一个用例列表，每个用例用几句话来描述，并确定其主执行者（primary actor）。使用 UML 用例符号开发用例概述，可以采用多种方式，但我们发现会见领域专家是最为有效的一种。一种替代方案是，让领域专家来建立实际模型。目标（goal）是捕获主执行者对企业的需要（need），或者说，企业需要为主执行者做什么，以及辅执行者（secondary actor）如何来为实现这一目标做贡献。这些需要（即主执行者的目标）是相互关联的，每个需要都是一个用例。就本文而言，企业用例概述包括进行空中自我防御和再供给舰艇两部分的示例，如图 3 所示。

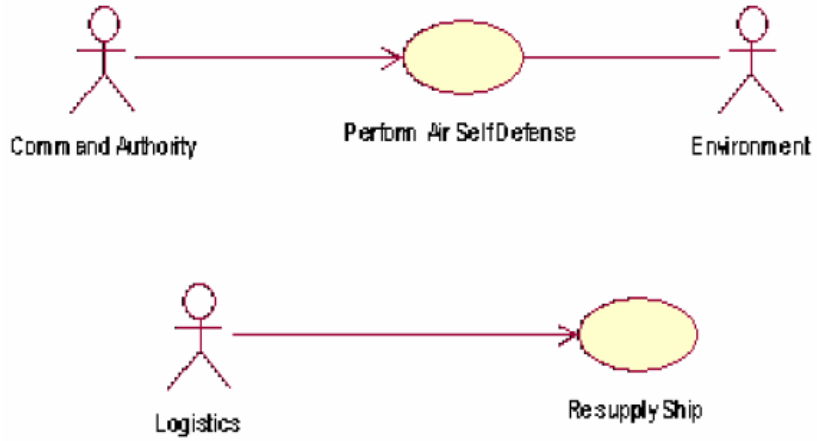


图 3 企业用例

2.3 领域对象模型（Domain Object Model）

领域对象模型（Domain Object Model）是一组逻辑相关的类以及类之间的关系，该模型通过类图（class diagram）来捕获。类图中每一个类表示了一个重要的领域信息块。但此时，这些类不包含方法（method），并且可能不包含属性（attribute），但是每个类都有一个清楚的、书面化的定义。类中包含了现实世界的信息，用户需要使用这些信息来执行在用例中描述的任务。信息能够通过系统外部和系统内部很多形式来描述。在前面提到的海军舰艇系统就是一个外部信息的例子，它包括出版物、文档、电子信息、手写日志。内部信息的例子包括一些战术图片及其他实体，通过传感器、计划、以及战斗方针所记录下来。我们发现会见领域专家是识别类来很好的

来源。

领域对象模型的好处主要有三点：

1. 为将看似无关的用例联系起来提供了机制，因为领域对象模型中的信息对于不同用例并无不同。

2. 为即将开发的分析和设计类提供了框架。

3. 从用户视角准确地定义了系统词汇表。

在我们的例子中，组成企业模型的系统已经知道了，因此，每一个内部类应该分配到“拥有”此信息的系统中。重点在于，区分外部类和属于 C2 系统内部的类。

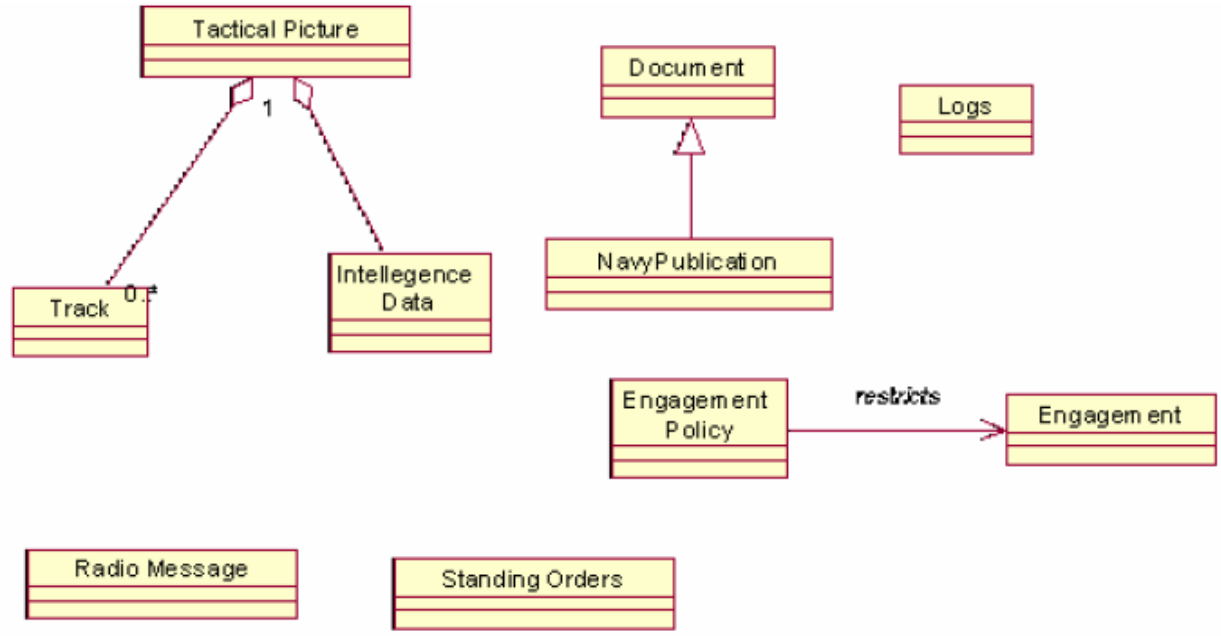


图 4 领域对象模型

我们增加了表示每个系统的代理类，至于其详细原因将在以后说明。图 4 所示是一个领域对象模型的例子。

2.4 用活动图细化用例

活动图（activity diagram）通过捕捉各种场景来细化用例。活动图由对象、活动和事件关系组成，其中事件关系捕捉系统和外部执行者之间的交互。注意，活动图不是数据流图（data flow diagram），因为活动图展示的是事件流而不是数据流。图 5 所示的是一个“进行空中自我防御”活动图的例子。

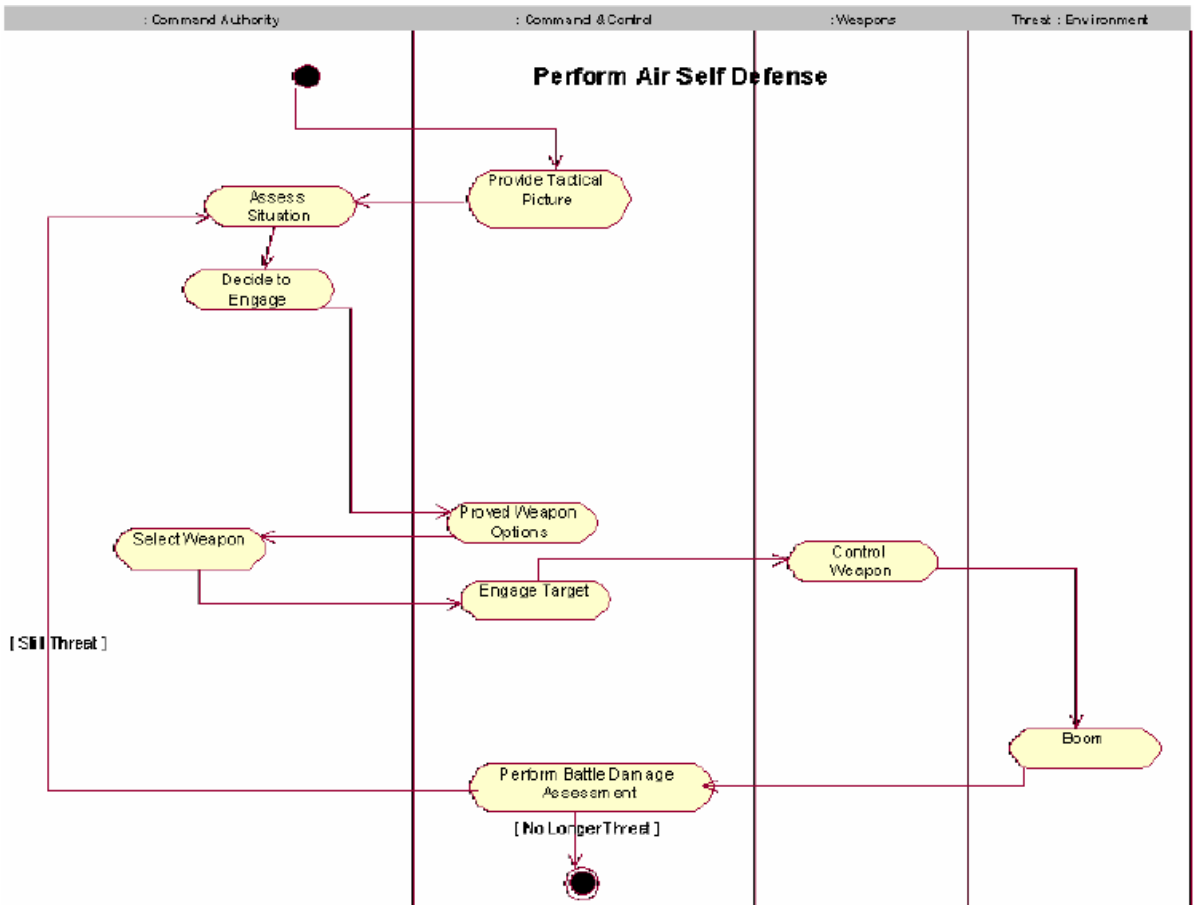


图 5 活动图

在我们的例子中，为每个用例创建了一个活动图，来说明这个“系统的系统”中的交互；而我们要构建的系统是 C2 系统，所以应对它给予特别关注。既然组成企业模型的这组系统我们已经知道了，那么，活动图的绘制就应该以 “白盒”的方式进行，也就是说，我们可以为每一个参与用例的系统都增加一条泳道。既然我们只准备细化 C2 系统，那我们的关注点就只在这个系统上。与用例概述和 DOM 一样，对于活动图的开发来说，领域专家也是至关重要的信息来源。

2.5 为用例的活动图增加对象流

在前面我们提到过，领域对象模型和活动图是一起相互交叠地开发的。既然活动是在活动图中被识别的，那么用户会问：活动期间使用什么信息？如果这个类已经存在，该类的一个对象实例就应该增加到活动图中，放置到拥有该信息的泳道中。

既然我们的示例只是细化 C2 系统，那么重点应放在 C2 那个泳道中的活动上。如果这个类不存在，就要在领域对象模型中增加一个新的类。

图 6 所示是一个含有对象的活动图。对象流的箭头表示被哪个相关联的活动所使用或修改。图 6 展示的活动图和图 5 相同，只是包含了对象。

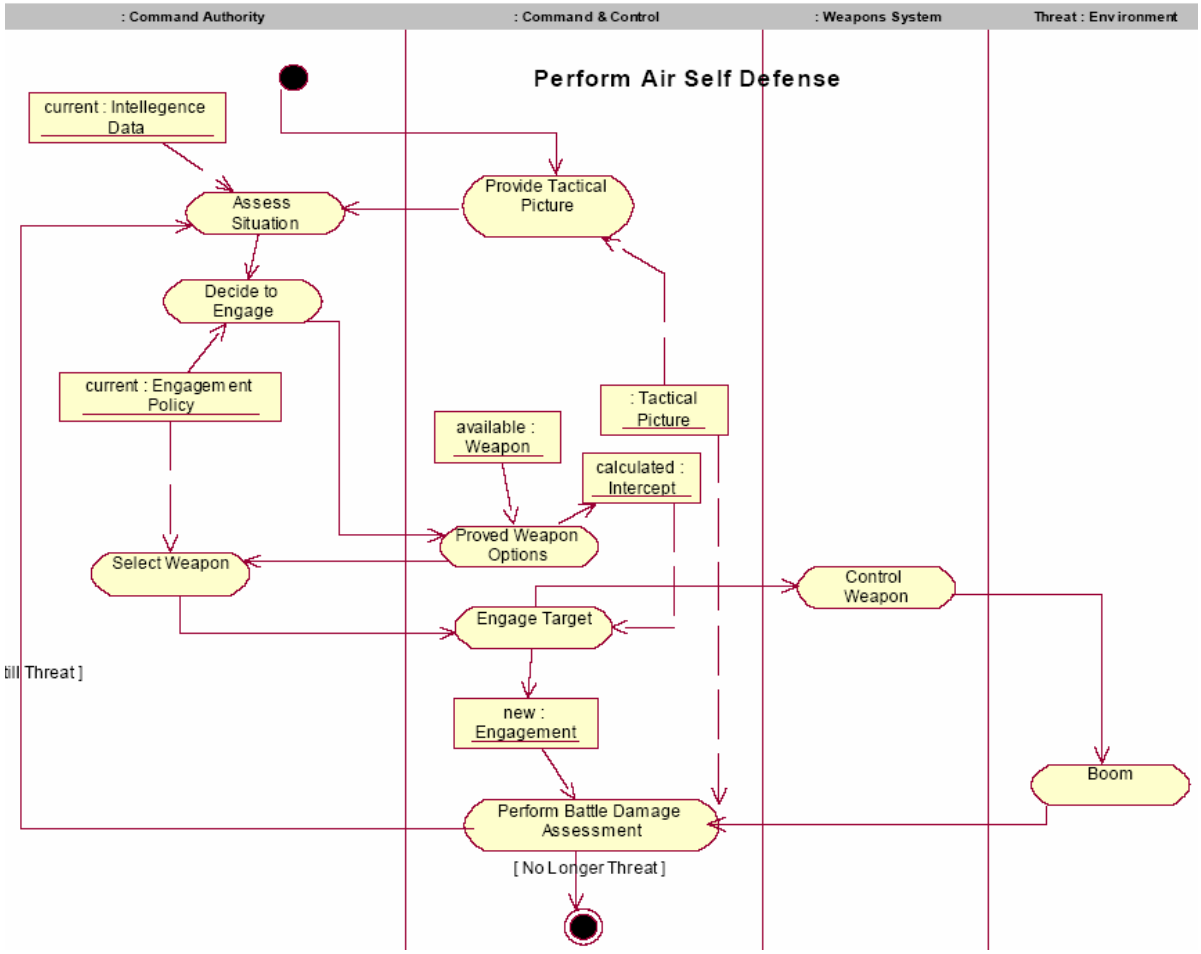


图 6 包含对象的活动图

3.0 开发C2系统的用例和领域对象模型

例子进行到现在，我们已经细化了和 C2 系统有密切关系的企业（即系统的系统）级用例，并准备开始开发 C2 系统分析的上下文图、用例概述和领域对象模型概述。第一步工作是为每个企业用例产生一个或多个序列图，在此过程中，将为一部分领域对象模型中的类增加方法；领域对象模型中的一部分类，将会组成 C2 系统的类概述。接下来，要以序列图的形式，来抽取 C2 系统的用例概述，而这些用例的执行者决定了 C2 系统的上下文图。

3.1 序列图

最终，将为每个用例创建一个序列图。随着用例场景复杂度的增加，描述每个用例所需的序列图个数会增多。为了使活动和特定用例场景紧密关联，建模者通过序列图，定义位于所关心系统的泳道中的 C2 领域对象与其他系统和执行者之间的协作。当操作（operation）被定义以后，要将它们分配到合适的领域类、代理或角色中，成为方法（method）。每个方法所执行的动作要认真描述并书面化，因为它们会被多个序列图频繁重用。同时，通过考察子系统之间的交互，来识别 C2 系统的接口依赖。图 7 是基于图 6 的活动图产生的 C2 系统的一个序列图。本例没有细化其他系统（如武器系统），而且方法也只是简单地增加到了系统的代理类中[译者注：这就是引入代理类的原因所在，而不是设计上的考虑]。

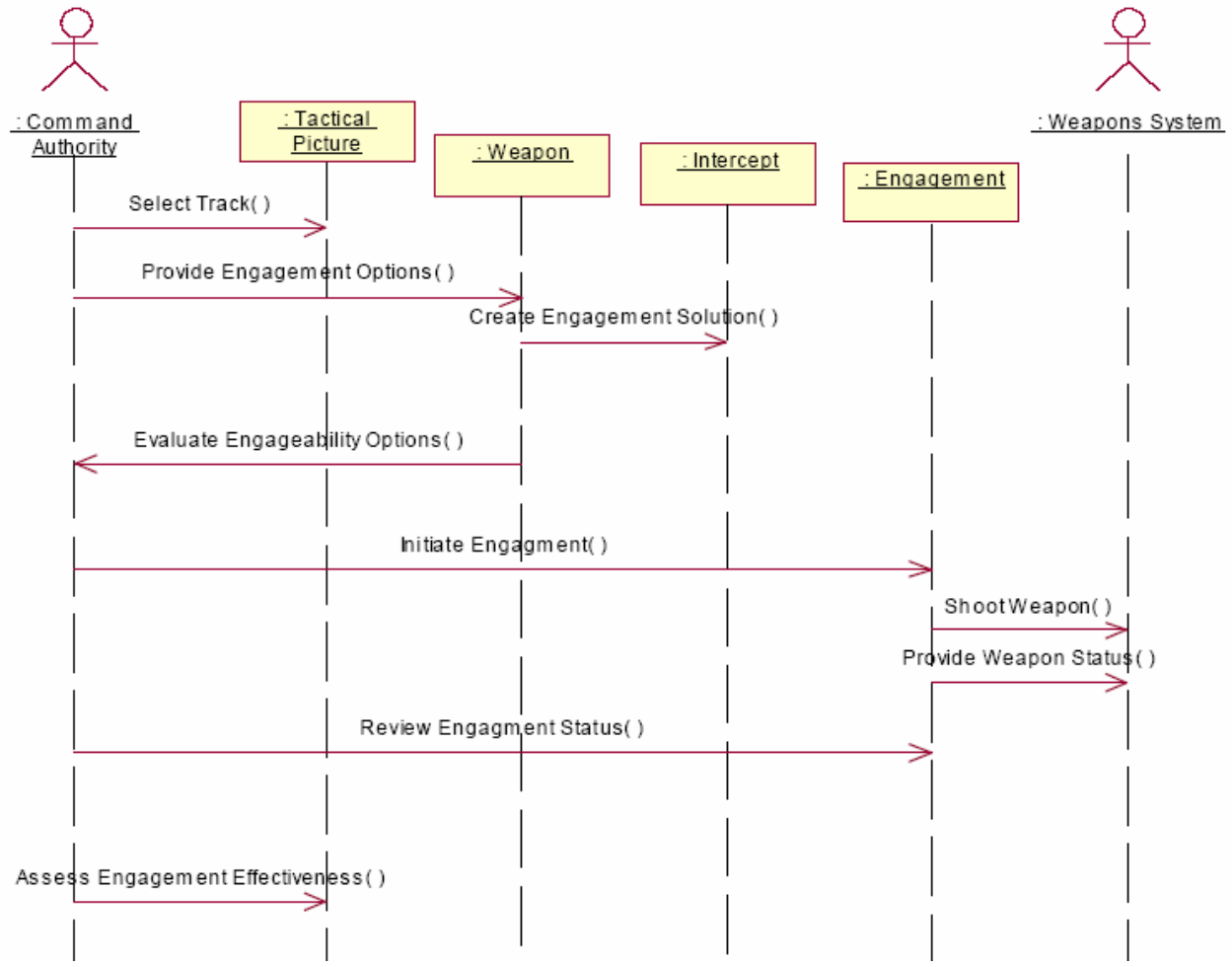


图 7 序列图

3.2 C2系统领域对象模型概述

当开发序列图时，会往隶属于不同系统的领域对象模型的类中增加方法。因为我们的重点是 C2 系统，所以我们的大部分领域对象模型类都和 C2 系统相关，但是每个系统至少有一个称为代理类的类。如上所述，除 C2 系统之外的所有系统，方法都被增加到代理类中了；而 C2 系统的方法，会分别增加到相应的类中。注意，不是所有的类都必须包含方法。那些包含方法的类以及被分配在 C2 系统中的类，组成了 C2 系统的领域对象模型概述。每一个操作都被明确定义，这些操作共同定义了 C2 系统要执行的功能。图 8 是一个领域对象模型概述的例子。

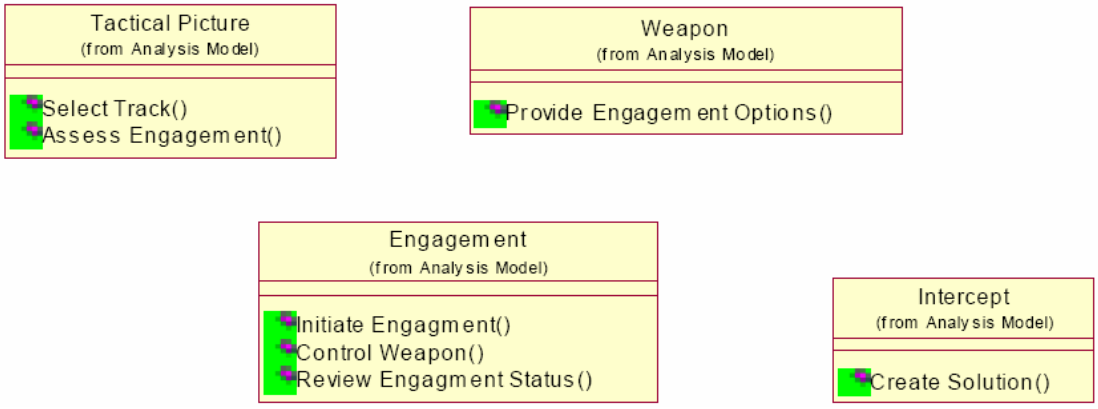


图 8 C2 系统领域对象模型类概述

3.3 C2系统用例概述

序列图不仅可以作为一种使用方法来定义系统功能的工具，还可以作为识别系统用例的工具。序列图中包含了 C2 系统的对象、以及其他系统的代理对象。从 C2 系统的角度来看，序列图中出现的代理对象都是外部执行者（external actor）。因而，序列图展示了 C2 系统及其外部执行者之间的交互。每一组连贯的交互成为一个用例，而包含了这些交互的相应的子序列图，可以作为描述用例概述的主要成分。

在系统用例概述中，企业用例和用例之间经常是多对多的关系。图 9 是一个 C2 系统用例概述的例子。

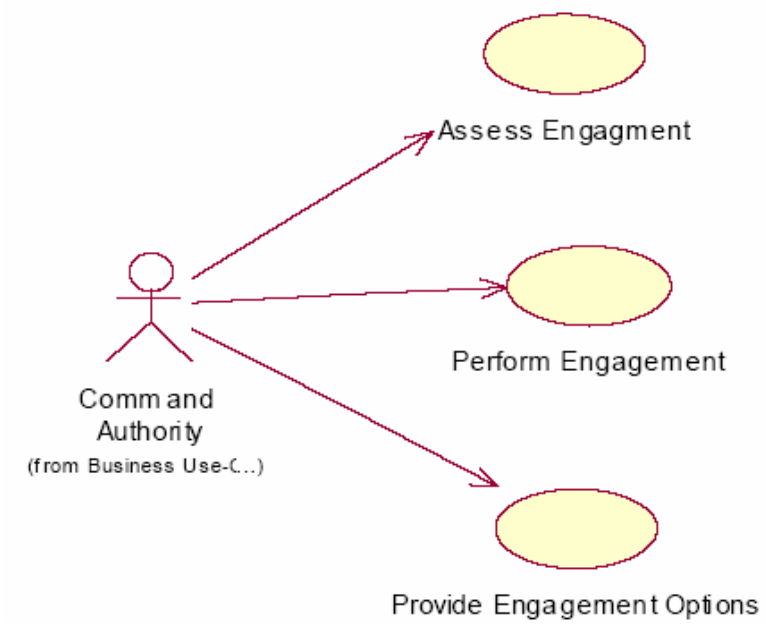


图 9 用例概述

3.4 C2系统上下文

用例和类概述开发完成之后，还应开发 C2 系统的上下文图。为此，应当回顾用例概述中的执行者，最终的上下文图将包括部分或全部企业执行者（Enterprise Actor）、以及部分或全部其他企业系统（Enterprise System）。图 10 是 C2 系统的上下文图。

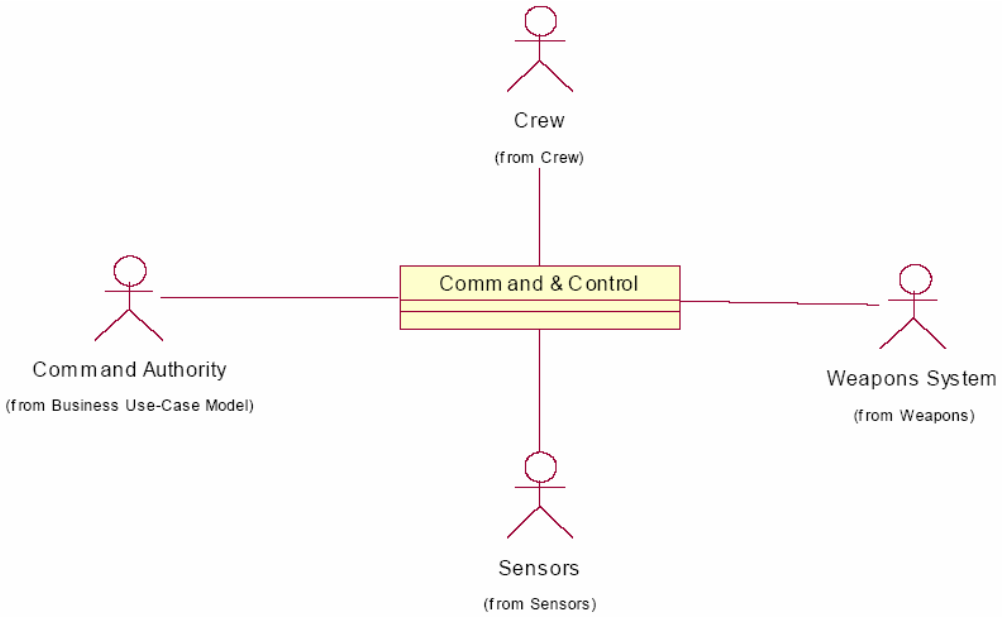


图 10 C2 上下文图

4.0 转换到下一层

现在，我们已经为 C2 系统开发了下列企业模型中的工件：

- C2 上下文图
- C2 用例概述
- C2 类概述

这些工件为下一层模型——C2 模型——提供了框架。实际上，它们是 C2 系统的规约。领域对象模型类概述中包含的方法，明确规定了 C2 系统的功能。每个项目的目标都不尽相同，那些“shall”式的说明，可能已经明确规定了每个方法的功能。C2 用例概述明确规定了 C2 系统如何被使用，它可以作为测试场景的规约。C2 上下文图明确规定了系统的外部接口。而用例概述中，和每个用例相关联的子序列图、以及外部执行者（系统代理）上的操作，为外部接口定义提供了更多细节。

现在，需要重复同样的过程，来发现 C2 系统的子系统。当然，也并非完全相同，其主要区别在于如何识别新的子系统，因为 C2 系统已被假定在我们的企业模型之中了。到目前为止的工作中，我们通过“白盒”活动图来整理我们对系统的认识，这意味着每个系统都有一个相应的泳道。这样一来，细化活动图时，我们只需专注于 C2 对应的那个泳道上；而在系列图中，除了 C2 之外的系统可以统统使用系统代理。当识别出领域对象模型类时，就把它们分配到拥有它们的系统（或外部系统）包中。而在并不预先知道子系统的情况下，所有活动图必将以“黑盒”方式绘制，也就是说，我们将只有一个名为“C2 系统”的泳道。和前面一样，仍然用对象流来细化泳道，仍然产生序列图。通过寻找关系密切的类的集合，并将它们集中在一起形成子系统的类概述的方式，来定义一个子系统。而子系统提供的功能的规约，是相关类的类操作说明组成的。分解的目标是为了得到一套松耦合的、几乎没有多余功能的子系统。

5.0 结论

本文介绍了一种面向对象系统工程的方法，来解决“系统的系统”的规约和分解问题。这是一个以用户为中心的过程，目的是细化功能需求、并产生子系统规约。这种方法以建模为基础，涉及到上下文图[译者注：采用类图语法]、用例图、类图、活动图和序列图等多种 UML 图。

这种基于模型的、图形化的方法有不少优点：图形化模型利于需求的启发、评审和用户确认，是该过程的关键部分。一个组织良好的模型框架，对方便地增加新能力和修改旧功能大有裨益。而模型的分级的、图形化的本质，为新的项目成员可以研究和学习“系统的系统”提供了极好的途径。当前的建模工具支持团队在“系统的系统”模型上协同工作，并提供模型直接的链接关系。

为“系统的系统”建模，还要求传统系统工程活动的参与。例如，性能模型和时序预算仍然需要诸如仿真模型等传统系统工程工具和方法来开发。我们认为：对于至关重要的活动，采用多种方法是值得称赞的。比如，当制定象 AP-233 这样的数据交换标准时，可能本文所讨论的方法和其他系统工程方法及工具之间的界线将会很模糊。

本方法中使用的图都符合 UML1.4 标准。这个标准还包括其他本文没有出现的图，如部署图、状态图等。UML2.0 标准正在制定中，有望在 2004 年发布。UML2.0 将包括专门针对系统工程的扩充，我们了解到的情况是，这些扩充只会增强这里讨论到的方法。

本文描述的方法集中在“系统的系统”的行为（behavioral）方面。当然，对于系统定义，一些非行为（non-behavior）需求（如可扩展性、可靠性、可用性和可管理性）也很关键。目前笔者正在研究用 UML 描述这些系统的非功能方面、以及把它们分配到子系统的具体技术。

参考文献

Bahill, Terry and Daniels, Jesse, Using Objected-Oriented and UML tools for Hardware Design: A Case Study, Systems Engineering (6) (2003) 28-48.

Cantor, Murray, Rational Unified Process for Systems Engineering RUP SE1.0, A Rational Software White Paper TP 165, (8/01).

Friedenthal, Sanford A., Object-Oriented System Engineering, Proceedings of the Lockheed-Martin Systems Engineering and Software Symposium, (July 1997).

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

在核仪器控制软件中应用敏捷方法

P.V.Hathaway、T.Lam、N.Hauser、A.Gotz、F.Franceschini 著，iampole 译

讨论

“让有能动性的人去做项目。给他们合适的环境，满足他们必须的条件，

并相信他们能把工作做好。”［1］

1 摘要

ANSTO(Australian Nuclear Science and Technology Organisation，澳大利亚核科学与技术协会)的中子波仪器项目最多控制八台和特定辅助设备相关联的仪器。该项目预计在2006年中期完成。

这个系统的控制软件的需求由一个国际委员会提出，软件架构将取决于来自于至少三个洲的专家的建议。项目的成功在很大程度上依赖于开发队伍的沟通能力，以及他们对于突发需求和资源变更做出适当反应的能力。

同传统的有严格的文档、确定的计划和过程的软件开发方法不同，敏捷软件开发方法更强调对于项目目标的持续逼近、开发人员的个人能力，以及团队成员之间的沟通。

本文探讨了在科学组织中应用敏捷开发方法的一些主要原则，并说明了到目前为止这些原则在中子波仪器项目中的效果。

2 科学组织中的软件开发过程

“每隔一段时间，开发团队反思一下如何能够提高效率，

然后相应的调整以后的开发过程。”［1］

目前的软件工程通过一些要素来界定软件的规模，这些要素至少包括时间、范围、包括预算在内的各种资源。在项目开展的过程中，这些要素经常会有一些突发的变化，某一个要素的变化往往会对其他因素造成很大的影响。对这些变化的不成功的处理往往会导致整个项目的失败。

科研项目往往会有更多的变更，允许项目根据以后研发的情况做出调整，甚至为了迎合项目成果的变化重新定义项目的范围和资源。在项目规划的时候，往往对项目的情况作出最坏的打算，允许项目花费比收益高很多的成本，或者仅仅取得远远比它可能获得的效果少得多的效果。

目前，科研软件开发的成功往往在很大程度上依赖开发人员的个人能力，以及一个在这个领域积累了多年经验的开发团队。这种情况在已经发布的科研软件中占有越来越大的比例。

下一代的项目需要的是将开发人员的技术经验和他们应用工具的能力尽量结合起来。

3 敏捷软件开发宣言

过去十年中提出的很多软件开发方法都是建立在开发人员的实力的基础上的。他们认识到了个人的贡献，密切的和频繁的交流，以及和软件本身多次的交互的重要作用。其实，软件本身才是项目的唯一目标。

这些方法论的提出者和实践者在2001年组织了专门的会议，提出了他们对于这些方法的宣言：

我们试图通过实践或者帮助别人来揭示开发软件的更好方法。

我们达成一致：

个体和交互 胜过 过程和工具

可正常工作的软件 胜过 面面俱到的文档

和客户合作 胜过 合同谈判

响应变更 胜过 遵循计划

一般，别人关注右侧的项目，而我们更关注左侧的。

此外，开发团队根据宣言制定了一些原则，并把它们形成文档。这些原则和一般的开发方法的主要区别如下：

- 敏捷方法更强调在预测的基础上的变化。设计和构建的计划中包括了对于需求、设计概念等的变更的管理，并通过有效的沟通把风险降低。

- 敏捷方法更强调人而不是过程。开发人员被认为是具有专业技能和主动性的技术专家。团队中最适合做出计划的人来充当管理的角色。客户被认为是评判系统应该如何更有效运行的裁判。

被称为敏捷的方法很多，例如XP，特性驱动开发，Scrum, Crystal,以及自适应软件开发。这些方法的适用领域各自不同，很难有对所有项目都适用的方法。

Fowler认为，自适应的过程适合于以下类型的项目：

- 需求不明确，或者不稳定。

- 有责任心和主动性的开发人员。

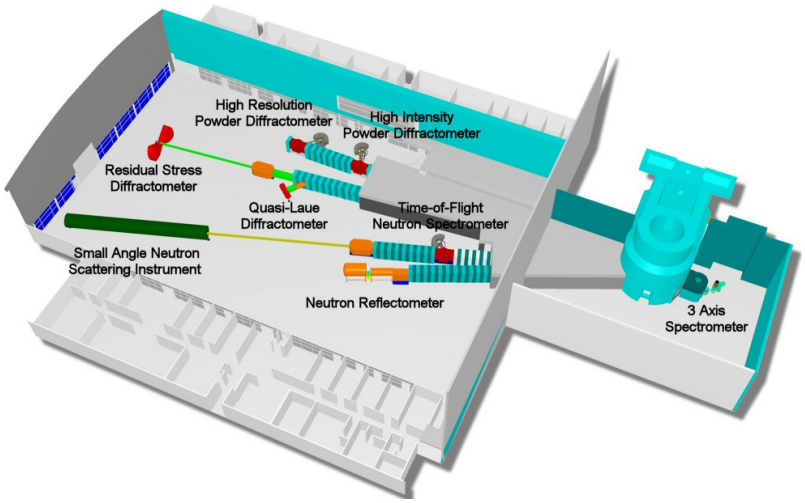
- 客户了解项目的情况，也愿意参与项目。

此外，科研组织内很多项目是有小团队完成，组织内有比较好的合作氛围，开发内容会持续变化，这些都很适合使用敏捷方法。

4 案例学习：中子波仪器项目

"朴素—把工作效率尽量提高的艺术—是最基本的。" [1]

NBIP（The Neutron Beam Instruments Project，中子波仪器项目）的主要任务是为一个正在准备实施的仪器提供一个先进、稳定、可扩展的控制系统，并能够积累操作员的经验。这个项目的特殊之处在于，开发团队是在没有硬件设备，甚至有时连元器件都没用的条件下开始工作的。



目前，准备各个仪器所需要的组件正在快速进行，在当前的条件下开发一个通用系统的时机已经成熟。最关键的问题是，随着仪器设备的到位和安装，必须同步开发相关的验证软件。控制系统必须能够同八台完全不同的仪器和辅助设备集成使用。

为了实现这些目标，必须开发以下适用于所有的仪器的系统：

- 控制系统：一个公共的核心控制系统，可以是应所有的仪器。
- 用户界面：在公共框架的基础上可以用户订制的界面。
- 配置管理系统：一个记录所有仪器配置情况的数据库。
- 开发方法、实践和工具：确保这些系统的开发是可控制的，能够在一定的时间内实现。

最终我们选择了一套已有的软件模块，一些开源工具，传统的软件开发方法和敏捷开发技术。

5 NBI 项目管理和软件开发方法

“只有自律性强的组织才可能产生出最好的软件架构、需求、设计。”[1]

NBI项目作了严格的计划，详细的预算、风险控制，进程跟踪以及文档管理，这些使得该项目成为一个非常成功的案例。在实现这些需求的同时，项目组希望该项目的开发过程是最好的。

大多数软件开发都包括一些共同的部分，例如：

- 目标陈述

- 具有不同背景、完成不同任务的角色
- 具有足够的经验和技术，可以完成这些任务的团队
- 为开发系统建立的一组正式或非正式的规范
- 一套技术，例如开发语言、工具、组件以及用于开发目标系统的其他产品
- 一个新的或者是基于以前的系统的框架结构，并在此基础上作出目标系统的设计和计划

在这个案例中，目标是明确的，角色和成员都已经选定。他们会选择一些可以实现我们的项目目标的方法和工具，以及开发项目所必须的一些原则。由于这个项目组一贯主张敏捷方法，因此选择了极限编程(XP)和按照特性计划的方式。

5.1 极限方法

极限编程(XP)是为了实现敏捷开发而选择的一套方法和原则。XP的发明者，Kent Beck和Ward Cunningham，都是敏捷宣言的发起人。

Astels对极限编程的方法和原则作了一个简洁的概括，正如图5.1所描述的。

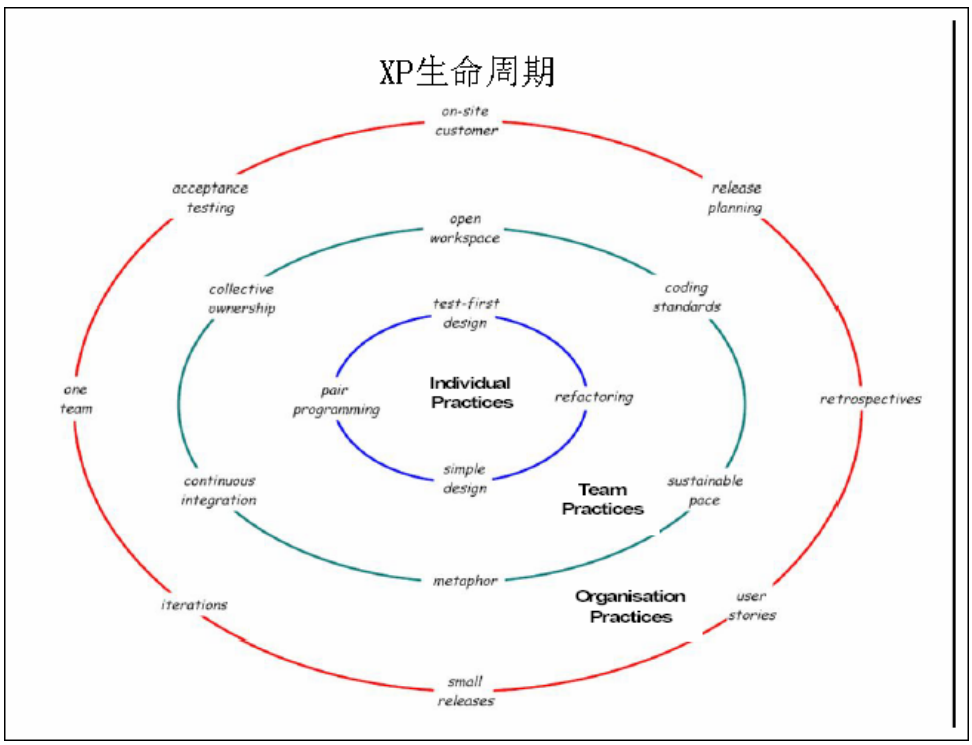


图 5.1 极限编程方法

正如敏捷方法关注的是人，XP方法以开发人员、开发团队以及开发团队和客户之间的共同方式为核心。

5.2 开发人员实践

NBIP项目组的每个人都要收集需求、修正软件结构和设计方案、做初步的测试和集成。由于开发环境引入很多工具，开发团队成员还需要将安装、配置和数据恢复程序文档化。

传统的软件开发过程在这里被认为都是同级过程，它们并行进行，并频繁同步，以便能够不脱轨。

为了能够对软件的设计和架构有一致的理解，每当一个主要的组件成熟的时候，这个部分的完成人将为项目组的其他成员组织一个讨论会或者培训，以便项目组的其他成员了解该模块的信息。这样的合作也提供了一个在模块的开发人员和用户之间的协调机会。

5.3 团队实践

XP方法的最大特点是不断的鼓励开发团队形成需求和理解。为了能及时了解需求的变化情况，敏捷方法要求经常和客户沟通，XP方法可以保证开发团队随时都可以取得客户的支持。

在紧密的合作过程过程中，需求的反馈和变化都可以很快的被吸收。为了及时响应，开发团队可以通过短时间的交互关注变化的情况。

NBIP通过每天的会议来同步他们的行为。为了保证例会在短的时间内结束，并能取得效果，项目组采取了“站立会议”的形式。所有的参与者站着参加会议，在一个简短的会议后马上回到工作中。项目成员汇总前一天工作的成功经验和遇到的问题，并考虑在下一步工作中如何调整。项目组的每一个成员可以指导团队现在做什么以及以后需要做什么，团队成员也可以自由地请求帮助，以便解决遇到的问题。即使是一个开发人员没有参加某一次会议，他也可以很快知道项目的信息。

这些工作是由开发人员完成的，当产生了一些新的问题需要和客户沟通的时候，客户也可以很自由地参加这项工作。

为了能够更好的沟通，XP提供了很多共享资源。例如，项目组的成员都可以使用已有的仪器设备，并可以共享仪器操作界面的图片。

5.4 按照特性计划

Coad在书中用了独立的一章概要的描述了FDD方法。它提供了一种相对大的、在技术和经验上有相当的差异的用来开发复杂的软件应用的方法。

FDD方法包括五个过程：

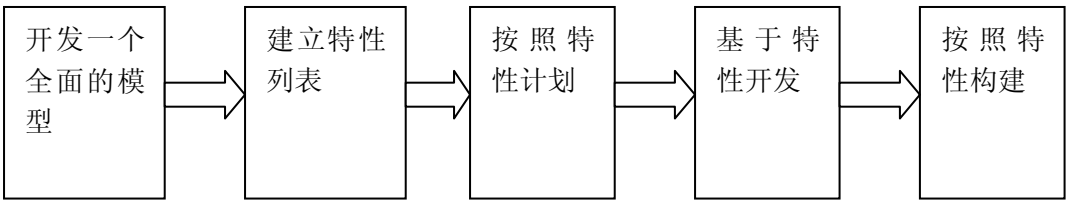


图 5.2 特性驱动方法的步骤

NBIP没有采用FDD方法，但是借鉴了FDD特性的概念，以及通过一系列特性制定计划和构建系统的优点。

特性是在需求获取过程形成的工件。

特性是对于系统的某一个部分为了获得特定的结果而执行的一个操作的简单描述。

用户故事是软件系统的用户执行的操作的自然语言描述。

开发团队记录在同客户进行的例会上收集的用户故事。通过这些用户故事可以获得一系列的特性，这样可以一直保持对于各种特性的理解。客户代表还可以帮助确定这些特性的优先级。

通过这种方法，开发足可以记录独立于实现细节和软件架构的用户需求。

为了和里程碑的概念区分，实现一个特性所需要的范围被称为粒度。FDD每个特性用不超过两周的时间实现，这样，对于实现特性列表的工作所需要的任务分配、工作量平衡、进度度量等都会比较容易实现。进度可以以单元为单位，这样既便于用户理解下一个版本所作的修改，也可以很方便地测试。

在FDD的方法中，基于特性的开发和构建会反复地进行，直到所有的特性都已经实现。特性计划仅仅提交目前的修改，说明哪些特性列表被实现，这些修改针对的是哪个用户群。

按照它们的优先级和依赖关系，特性集排成一个实现序列。在每一个交互过程结束时，计划的变化都会反过来影响到那些已经实现的特性。这样的反馈过程有助于更好地管理需求的变更。

6 协作

“在一个开发团队中传递信息最有效的方法是面对面的交谈。”[1]

6.1 控制系统

NBIP选择了SICS（SINQ Instrument Control System，SINQ指令控制系统）作为基于服务器的控制软件的基础。SICS是一个在PSI应用的操作简便、稳定、经过严格测试的平台。Hauser的著作帮助我们确定了这个系统的标准和决定。

2004年上半年，NBIP项目组邀请了SICS的主要作者之一，Mark Koennecke参加了ANSTO关于开发的一系列研讨会和培训。除了介绍有价值的开发理念和SICS的使用，还介绍了一些有关SICS的进展。

这项协作的最直接的影响是，可以在SICS的架构添加站点库，通过一组驱动程序和脚本把站点维护和核心系统分开。同时，PSI所使用的模块提供了很多可以被ANSTO使用的模块和例子。

只有SICS的服务器和设备驱动库可以同ANSTO的系统完全结合起来。每台仪器在虚拟的局域网上运行一个SICS服务程序，它可以通过TCP/IP和设备控制器以及数据服务器在局域网上交换数据。

这样的连接范型可以使SICS服务器不必同设备分享内部总线，而是通过网卡连接。这样，服务器可以运行在仪器附近一个可靠的、独立的PC上，可以通过任意数量的客户机访问服务器，不管是在仪器附近还是在远程，只要有一定的授权，就可以访问。服务器是基于面向对象结构的，内嵌的Tcl解释器可以实现以下特性：

- 解析客户端发来的命令行指令
- 在服务器启动的时候执行初始化脚本，定义仪器的配置参数
- 翻译和执行服务器发来的提供扩展特性的脚本
- 翻译和执行用Tcl编写的客户脚本

设备驱动程序是为各种特殊设备编写的，它们在启动时候通过一个脚本为特定的仪器定义数据、组合方式和配置数据等。

通过提高系统的可靠性，开发过程中减少了测试工作，这样使对仪器的访问得到了限制，维持在了一个典型的水平。为了保证维护计划的弹性，需要增加新的设备以扩展系统的容量。

SICS的模块通过PSI和代码仓储同步。

6.2 客户端

基于Hauser提出的原因，NBIP很快选择了C/S的结构。

ESRF的软件工程师，Andy Gotz为SICS控制的仪器提供了选择平台无关的框架的图形化用户界面客户端的指导。为了达到这些要求，通过一定的调研，我们很快决定选择基于Java的平台。

RCP（Eclipse Rich Client Platform，Eclipse富客户端平台）天然集成了Eclipse开发工具，它为我们提供一套成熟的、可以客户化的、标准的、独立于操作系统的开发工具。我们把客户端软件称做Gumtree。Gumtree本身足够灵活，足以满足ANSTO的规范(GumNIX)，完全有可能很成功地在以后的控制系统，例如EPICS或TANGO上应用。

6.3 其他合作的可能

敏捷开发提高了稳定开发的可能。发起人、开发团队、用户都可以取得比较稳步的进步。[1]

只要有可能，开发团队总是想办法提高系统的效率。

SICS可能为以后的站点提供同步的扩展。SICS服务器程序的核心部分是PSI（Paul Scherrer Institute）的Mark Koennecke开发的。

核心代码已经扩展到可以支持站点描述的设备和方法。伴随着系统的扩展，核心系统已经由一个新的团队在一定程度上作了修改。

开源思想在仪器控制客户端的开发中起到了重要的作用，开发团队从开源工具获益，并因此提高了效率。我们的客户端软件，Gumtree在SourceForge注册为一个开源项目，对其他同我们的开发团队同等地位的合作者开放，来自ESRF和PSI的开发人员已经表示出了对我们的平台的兴趣。

由于采用了开放的态度，事实上，很多其他用户和程序员扩展的程序和工具被我们所采用并加入到我们的系统，例如Eclipse及其插件，NeXus以及TANGO。

7 敏捷开发工具集

随着一个新的开发团队的建立，必须选择一个开发方法论和核心工具集。这一定要是简单易学的工具，并有比较高的效率可以在一定范围的设备上建立稳定的方案，还要有充分的扩展性，以便能够满足仪器科学发展的需要。

开源集成开发环境

Eclipse是一个集成开发环境和应用程序平台。除了为项目提供Java和C/C++的编辑工具，Eclipse的插件还给我们提供了和源代码管理工具（cvs或者vss）集成的图形化的基于菜单的工具。

- 源代码管理工具（cvs或者vss）
- 调试工具(gdb)
- 源代码文档化工具(javadoc和doxygen)
- bug/发行数据库(bugzilla, MySQL)
- 外部编辑器(针对Tcl, perl, Python等)

插件也提供了本地功能的支持。

- 针对java和C/C++的优化工具
- java单元工具
- 用SQL输出建模的数据库环境
- UML支持(自行开发)
- 可执行程序的分析工具(Hyades)

此外，Eclipse还有很多其他免费的或者商业的插件，这些插件为软件提供了扩展的功能或界面。

整个开发团队都使用相同的集成开发环境，不论是C/C++或者Java的开发，调试，测试，还是形成相关的文档。这样提高了团队成员交换工作的可能性。

其他基于社区的工具

- Apache / Bugzilla / Perl / MySQL
- MySQL同时作为配置管理系统的后台数据库
- NeXus和HDF是SICS所需要的。这些产品也提供了对于开发工作很重要的API和数据浏览器
- Tango包omniORB目标分解器
- DejaGNU测试环境

配置管理和持久化

开发过程中的一个主要目标是从中心知识库自动生成配置指令的文件和对这些配置文档的可视化控制。

在仪器操作过程中，保留模块的调用参数可以使模块从不正常的状态恢复。

把这些参数保存在一个分布式的系统有利于远程指令状态监测和调试。

为了保证系统所有的特性都没有重复的完成，我们需要一个集中、可靠、快速、基于网络的模块，通过它可以知道任何一个配置的关系，以及它们当前的状态。

目前的数据库被定义为在SICS初始化的时候完全覆盖。这样，SICS可以在适当的时候根据物理配置脚本重建。我们倾向于使用JDBC完成查询和构造响应脚本的编码。

数据库本身使用Eclipse的插件。配置数据库的脚本可以翻译成各种形式的SQL。数据库的设计是文档化的，并且有严格的版本控制。

8 HIFAR MRPD：自适应计划和提交代码

“可执行的软件是衡量进度的唯一要素。” [1]

在仪器还没有安装的情况下，开发团队想到了早期的一个项目，于是想把软件系统原型运行在这个已经存在的仪器（HIFAR1）上。虽然增加了一定的工作量，但是也产生了一些值得期待的成果：

- 取得了在一个实际的仪器上调整和扩展SICS性能的信心

- 通过团队的开发实践，取得了开发过程、任务管理等方法论的信心

- 通过和现实的系统的交互，拓展了对现行系统和NBIP仪器科学的视野

- 提供了一个测试小的持续重构的平台

- 测试了可配置的服务器和客户端目前已经实现的细节，客户端包括控制选项，仪器状态显示，试验状态显示，数据显示

- 提供了调试和测试过程的验证

- 提供给所有的用户一个客户端以得到关于可用性的反馈

- 取得了存取实际的数据的经验并对于如何管理它展开了讨论

- 测试了我们的前端数据压缩和分析工具

我们选择了仪器HIFAR MRPD，因为它在控制的设备数量以及设备系列方面和目标系统比较接近。现在的用户界面所具有的一些功能可以被新系统的客户端Gumtree所使用。这些仪器的使用进度刚好可以提供给项目组使用，也可以披露给开源社区。

通过对问题域的分析确定了项目的目标和限制。然后开发分两个方向进行，分别开发客户端和服务器端，通过定期的会议同步进度。

团队和一个这个领域的专家组同时工作，这个专家组包括这方面的科学家和HIFAR的开发人员和维护人员。专家组提供了有关很多专业知识，包括仪器的使用，目前的和计划中的功能的细节，如果有新的操作代码，他们也会马上反馈。作为回报，他们对于仪器的结构和使用取得了更深的认识，这些对于他们以后的工作和决定有帮助。

每个设备的驱动程序完成的时候，都会组织针对通讯和特性的测试。在可控的状态下对每个驱动进行现场测试。

作为先行特性列表的一个部分，MRPD的设备和特性也集成在了开发计划中。

在很短的时间内，一个完整系列的设备驱动已经编码完成。MRPD目前正在等待最后的集成，然后仪器科学家和用户可以提供关于应用程序使用的反馈。

9 方法论的展望

作为一个新成立的团队，我们没有在开始的时候很急切地建立所有的开发环境以及提供所有的支持工具。NBIP的开发环境和项目开发的过程同步进行。这不是最理想的，但是在我们所拥有的资源的情况下却是最现实的。尽管我们拥有一些传统的开发过程的工具，但是我们还是选择了更能够支持我们的敏捷实践的方法和工具。

调整目标特性集可以提供一种在项目组成员之间平衡工作量的机制。

作出特性的计划以后，开发人员集中精力关注这些内容，而团队负责人维护一张进度列表。如果发现一些特性是有问题的，或者开发人员不能继续原来的工作，资源将通过一个定义完整的机制重新分配。

我们一直坚持每隔三个月在开源社区公布自己的项目里程碑。

尽管我们在这个间隔调整计划，但是我们希望在项目组内部，我们的频率要更高些。这样做的一个先决条件是高度自动化的打包和部署。一旦实现了自动部署，一个新开发的特性可以在几分钟之内反映在软件包里面。

在极限编程里面，我们希望细化开发任务的粒度。XP需要把任务分解成一对程序员可以一天之内完成，其中包括单元测试。

如果不能做到这点，就说明任务的范围太大，需要再分解。简单的说，可能存在潜在的没有其他的支持就无法实现的可能。我们的测试还是一种不是很正式的模式——开发人员本人测试他自己的代码。

程序编译是自动进行的，单元测试框架是不完整的，也是非正式的。集成测试仍旧是在开发人员在完成一系列的特性以后进行的，尽管我们尝试把测试的频率提高。

程序员自己测试的方法被实践证明是非常有效的。似乎，心理上的障碍比实践的障碍更加难于克服。根据我们的有限经验，程序员在潜意识中都希望在开发结束后马上进行测试。去除了bug的程序可以保证更加稳定，这样的做法可以提高整个开发团队的效率。这种方法在很大程度上要依赖其他方法的使用。

10 结论

科学软件的成功在很大程度上会受到有经验的参与者和专注的团队的影响。为了更快、更好地完成工作，必须花更多的经历来吸纳当前软件开发的原则和实践。

敏捷软件开发原则关注的是人。敏捷开发包括很多方法，例如XP和FDD，同重量级的文档驱动的开发过程相比较，敏捷方法在灵活性等方面更有吸引力。方法创始人强调了在软件实践过程中的变更而不是孤立的进行一些实践。然而，在科学和软件开发的精神中，这并不排除在实践过程中会有一些变化。

通过尝试一系列敏捷的方法，ANSTO的中子仪器项目产生出了高质量的应用。项目组每天的协调会和与客户的频繁沟通对于鼓舞这个新建立的项目组的积极性起到了重要的作用。协作、公共的开发工具以及面向最终用户的应用、信息的流动和反馈都对结果产生了积极的作用。

很多方法很难独立使用。开始的时候是凭着直觉选择了测试驱动的开发，结对开发，自适应计划周期以及持续重构，不过，后来的结果证实，这些方法都取得了成功。

使用这些方法并不能保证一定成功。现在，我们已经发现开发人员的经验和技术仍旧是影响开发结果的最主要因素。然后，我们相信，对于合适的人，基于敏捷原则的开发方法可以产生更好的结果，同时形成一个愉快地、有激情的工作环境。

参考文献

[Astels] Astels, D., G.Miller and M.Novak *"A Practical Guide to eXtreme Programming"*, Upper Saddle River NJ: Prentice-Hall PTR, 2002. ISBN 0-13-067482-6

[Coad] Coad, P., et al. *"Java Modelling in Color with UML"*, Upper Saddle River NJ: Prentice-Hall PTR, 1999.

[Fowler] Fowler, M. *"UML Distilled"* 3rd Ed., Addison-Wesley Publishing Co., 2003. ISBN 0-321-19368-7

[Hauser] Hauser, N., et al, "Choosing an Instrument Control System", NOBUGS Conference 2004.

[Palmer] Palmer, S.R., and J.M.Felsing *"A Practical Guide to Feature-Driven Development"*, Upper Saddle River NJ: Prentice-Hall PTR, 2002. ISBN 0-13-067615-2

URL

[0] "Manifesto for Agile Software Development" <http://www.agilemanifesto.org/>

[1] "Principles behind the Agile Manifesto" <http://www/agilemanifesto.org/principles>

[2] "Eclipse" <http://www.eclipse.org>

[3] "eXtreme Programming" <http://www.extremeprogramming.org/>

[4] "Feature Driven Development" <http://www.featuredrivendevelopment.com/>

[5] "The New Methodology", Martin Fowler <http://martinfowler.com/articles/newMethodology.html>

[6] "The Agile Alliance" http://www.agilealliance.org/

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

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

游戏设计模式

Jussi Holopainen、Staffan Björk 著，anyion 译

介绍

这篇文章是在2003年GDC（Game Developers Conference 游戏开发者论坛）上的一个演讲 “游戏设计模式”的笔记。该演讲是游戏设计团体和研究团体的研究结果。这些团体致力于证明半形式和形式的游戏设计方法将优于已讨论过的游戏设计方法。 我们相信游戏设计模式是一种既可用于商业领域，又可用于科研领域的可行方法，而且它是这两个领域之间的桥梁，允许二者在比以前更广泛的范围内协作和共享知识成果。

如演讲中所陈述的，我们认为模式只有在设计者能够学习和合理地应用它们的情况下有效，它们可以被剪裁以适用于任何项目。换句话说，模式需要支持实际工作，例如开发最初的游戏设计或解决游戏中交互元素的问题。因此该演讲主要侧重模式的实际应用，以及包括如研讨会等元素。这些笔记不仅讨论了模式的实践应用，还讨论了设计模式的各个方面。例如，如何定义模式和使用模式的优缺点。

本文并不打算定义一种“规范的模式语言”。要想证明一种语言的优点，该语言必须经过使用并被使用者(speaker)扩展。因此，我们不准备给出完整的最终的模式集合，甚至定义，取而代之的是展示我们正在做的工作，以便设计者能改进该技术使之更好地适应其需要。这篇文章一方面描述了模式的一种结构，另一方面侧重于介绍如何将模式作为一种技术去应用，以及创建设计者自己的模式集合。本文从多个角度来阐述这两种方法，并提供了在游戏中使用模式的两种可能性。在2002年GDC关于设计模式的圆桌讨论会(Kreimeier 2002b)上的相关资料和Gamasutra的文章中有更多信息可供参考。这部分笔记是在电脑游戏设计模式研讨会上展示的论文的续篇(Björk&Holopainen 2002)。

如果模式被业界人士和研究者广泛接受，我们希望会有多个不同的方法和模式集合涌现。不要拘泥于具体的规范化的形式，如果游戏设计者和研究者最终开发出一种能表达该团体所拥有的知识和经验的语言，那我们就是成功的。

为什么引出设计模式

在详细阐述我们的用于游戏设计模式的模型之前，使用设计模式的动机是不言而喻的。我们相信下列（各不相关的）事实是需要模式的有力证明，如一位有经验的项目经理领导一个较大的团队，一位新的设计人员因不能完成他的第一个游戏概念任务而沮丧，或者一个沉溺于某种爱好的人。

为游戏交互设计解决问题

模式包含对游戏中可确认的交互元素的描述，以及明确如何展示这些元素的方法。如果一名设计人员对一个游戏概念(game concept)中的部分交互有疑问，他可以从模式中找到答案。模式中所描述的方法包含所需的交互元素并提供了几种可能的解决方案。

灵感

收集模式在本质上是为了拥有一个其他游戏设计者在游戏设计中总结出来并行之有效的概念列表。有了这些概念，游戏设计者就相当于具有了基本知识，可以应用这些知识来发掘新的游戏设计的核心，或者能使一个游戏独树一帜，区别于其他游戏。

能激发创造性的设计工具(Creative Design Tool)

如Alexander et al (Alexander 1977) 所描述的：设计模式的最初应用是提供一个工具来帮助设计者（在原文中是建筑师）以基本结构化的形式完成各层面的设计。游戏设计模式集合的用途是类似的。一个设计者可以基于核心游戏概念(和外部需求)着手选择几个模式。这些模式在具体的环境下被分析过，以供设计者使用，可能用到的子模式也经过验证。子模式的分析和挑选基于它们的可行性并且它们的子模式也经过验证。该过程递归进行直到初始设计完成。

和同事沟通

模式提供了在很多游戏中定义的公共元素。根据模式来描述一个设计提供了一种以标准格式来描述设计的可能性，这使得理解和比较不同的设计更加容易。进一步讲，如果读者不熟悉一个模式所描述的游戏概念，他可以去查模式的定义，就象在字典中查找词汇定义一样。

和其他专业人士沟通

使用设计模式的方法在一系列学科中被引入，包括建筑学(Alexander 1977)，软件工程(Gamma 1994)，人机交互（HCI Human-Computer Interaction）(Borchers)，和交互设计(Erickson)。尽管不是所有的领域都与开发游戏相关，软件工程和HCI是与开发游戏相关的。在设计一个真正的游戏中使用模式可以使在项目组的不同部分之间传递信息变得更加容易，就象传递普通的工作技巧一样；获得对来自于其他学科的模式的基本理解也变得容易。

游戏设计模式的模板

Gamma(1994)引用了Alexander的定义：“每一个模式描述了一个在我们周围不断重复发生的问题，以及该问题的解决方案的核心。这样，你就能一次又一次地使用该方案而不必做重复劳动” (Alexander 1977)。因此，模式可以被描述为在不同的设计中可发现的公共特征。一个典型的描述模式的模板包括它的*名称*，*问题描述*，*解决方案*以及应用模式的*效果*。对于游戏设计，我们使用以下模板：

名称：对模式的简洁的描述性名称，使这些名称自己就能被当作独立的概念来使用。“命名一个新的模式增加了我们的设计词汇。设计模式允许我们在较高的抽象层次上进行设计。(Gamma 1994)。 然而，对隐含的模式命名可能是很困难的，并会带来问题。例如：模式剪刀－石头－布（Paper-Rock-Scissors）描述了*避免绝对的统治权*这样一种策略。但该命名容易让人误以为它描述了*玩家秘密地各自选择一种策略，然后同时揭露选择结果*的行为

描述：不同于大多数设计模式的是，我们决定不将模式定义为纯粹的问题－解决方案这样的二者组合。这基于两个观察结果：首先，根据问题来定义模式可能会带来一个问题，即将模式看作只能用来去除设计中不想要的效果的方法。换句话说，是将模式作为解决问题的专用工具，而不是进行创造性设计的工具。其次，对于很多模式，当我们发现了一个特征以后，在一定程度上该特征保证了游戏必将具有其它某些特征，例如：在模式中描述的问题可以通过应用更具体的子模式来更容易地解决。

效果：每个解决方案都有其正面和负面的影响。换言之，方案解决了一个问题，但会扩大其它问题。采用一个设计方案或反对一个设计方案时应考虑它的成本和收益，并将它与它的替代方案进行比较。本节描述了应用模式所建议的方案可能会带来的效果。

使用模式：由于模式是通用解决方案，对于一个在给定情况下的具体应用，需要一些针对当前环境的设计选择。可是高层次的选择常常以种类来划分。本节是为了总结设计者在应用模式时所面对的共同选择，示例取自正式发布的游戏中的具体游戏元素。

相关模式：表述了不同游戏设计模式之间的关系。模式有三种基本联系：父模式用以描述更抽象的特性（一般在效果部分提及），它们可以通过应用所给的模式来实现；子模式，它们可以被用来实现所给出的模式（一般在使用模式部分提及）；以及冲突模式，它们很难用给出的模式来实现。

对模式的批评

在其它领域使用模式遭到以下几方面批评（模式只是附庸风雅；模式过于形式；模式不够形式；模式过于随意）。我们希望读者注意2002年GDC圆桌讨论会的相关资料中关于设计模式的讨论（Kreimeier 2002a）。无论如何，在游戏设计中使用模式也会遭到类似的评判。下面我们提出了一些可能存在的反对在游戏设计中使用模式的观点。

模式只是附庸风雅

问题：模式在其它领域开始流行，在游戏设计中使用模式是否只是为了追随这种潮流呢？

答案：我们认为设计模式的好处在于它们展现了一种有助于设计过程的通用方法，本质上它们定义了一种可在具体设计领域使用的语言。在其它领域流行的模式能被其它方法替换或挑战，例如：所开发的方法介绍或趋势是针对该领域的。无论其原因何在，游戏团体当前缺乏一种通用的讨论设计的语言。我们认为模式是该问题的解决方案，但其他等价的解决方案也可能存在，真正重要的是找到一个方案。

模式过于形式

问题：形式化设计过程是否会抑止设计者和艺术家的创造性？

答案：设计模式不应被视为工具；设计者应该只在适宜条件下使用它们。根据它们的定义，对它们的解释是开放的，为创造性发挥留有足够余地。而且，它们侧重于游戏交互，并不涉及真正的游戏内容，如创建角色、情节等。

模式过于抽象

问题：看起来游戏设计模式并不能真正对编写代码提供任何帮助。模式是否过于抽象呢？

答案：游戏设计模式是从交互的角度，而不是以程序语言来表达游戏的角度来处理游戏设计的。因此，根据定义它们比把游戏设计形式化为代码的方法更加抽象；它们也不应该被用于形式化代码方面。

模式不够形式

问题：游戏设计模式看起来不够形式，不能被翻译成软件工程中的设计模式。假定某人设计了一个游戏并通过游戏模式来描述它，这样做有什么实践意义？

答案：我们认为游戏设计模式和面向对象编程的设计模式不是简单的一一对应。然而，使用了游戏模式的游戏描述提供给程序员一个蓝图，并勾画出游戏的基本架构，这种描述的基本结构和程序员编程时所使用的结构是类似的。我们认为从这种描述到代码之间的转换将比从场景(scenarios)或故事板(story-boards)到代码的转换更加容易。

模式过于随意

问题：如何确定模式？是否存在一种形式的选择方法？

答案：创建一个模式集合就好像编写一本字典。编写者必须研究人们如何使用词汇（并决定如果他们使用该词汇来交谈，人们是否会对其感兴趣），确定需要被描述的词汇，以及与其他词汇相联系的词汇。如果希望字典能产生任何影响，必须有人真正去使用它。

我们通过研究游戏产业中概念的使用来收集模式。通过观察电脑游戏和传统游戏，研究与游戏有关的文档，访谈游戏设计人员，以及真正地玩这些游戏我们确定了几种模式（主要根据游戏类型和游戏结构）。其实我们发现了更多的模式，只是还没有为它们命名；这些模式被用于游戏的最终设计和存在于设计者的设计活动中，这证明了它们的存在。我们正在试图描述和分析发现的所有模式。然后将这些精华方法展示给游戏设计团体。只有那些对设计者有意义并被真正使用的模式才会被保留下来。

使用设计模式

那些对使用游戏设计模式感兴趣的读者可能会提出三个重要问题。本节我们试图简要地回答这些问题；无疑读者还会有其它问题，欢迎读者与我们联系并讨论这些问题。

模式是否在游戏设计中有效？

给这个问题下定论还为时尚早。显然设计者可以在设计过程中使用游戏模式，但使用设计模式所带来的收益是否大于它带来的负面影响？我们试图确认设计模式可能带来的好处，并查找可能存在的负面影响，以及能够避免或减轻负面影响的方法。但在设计模式被广泛应用以前，这些都只是推论。

无论如何，我们认为游戏模式是否成功是无法单单通过游戏销售人员来衡量的。取而代之的是，如果设计人员认为模式能在设计过程中帮助他们，并在创建游戏的过程中增进他们与其他人的交流效果，游戏设计模式则被认为是成功的。

设计模式是否适用于各种类型的游戏？

我们通过观察电脑、棋盘和纸牌等不同类型的游戏来构建模式集合。因此，我们认为模式可能在这些游戏中有效。而且，我们工作的研究目标之一是提供一个理论框架以帮助设计新形式的游戏，包括基于位置或环境的移动电脑游戏，使用新型输入输出系统的游戏，和陆续移植到嵌入式电脑中的传统游戏。

应如何使用设计模式

请通过电子邮件联系演讲者。我们正在收集、定义、和分析模式，构建一个供从业者应用的有关模式的全面集合。这些模式必须经过实际应用来验证它们的价值，能够完成这项工作的只有游戏设计者。

替代方案是从构建你自己的模式集合开始。这需要进行更多的工作，但允许你选择不同的切入点；我们的模式集合源自用户经验或可能的交互，其他来源也可能存在，例如：游戏状态的操纵控制或游戏引擎的构成。

存在哪些设计模式？

对这个问题的简单回答是：任何已确定的模式。然而，到底存在哪些有用的设计模式仍然是很难回答的。我们正在进行确定那些在问题解决工具，和在分析游戏设计时描述游戏元素方面有效的模式，或是那些我们认为有助于设计过程的模式。为了完成这项工作我们研究和玩过很多游戏（电脑游戏、棋盘游戏、纸牌游戏、角色扮演游戏，等等），我们也访谈了许多专业游戏设计人员。在试图描述其它模式或模式之间的相互联系的过程中，我们找到和提炼出许多模式。

本节我们将展示一系列我们已确定的可供游戏设计参考的模式。除了描述一些在游戏中反复出现的模式，我们还提供了如何使用游戏模式的例子。下文例子中描述的模式 **设计瘫痪（Analysis Paralysis）**是游戏设计者通常应该避免的设计之一（另一个例子包括**Invisible Wall和King Maker模式**）。然而，在游戏设计中由于错误，或由其他模式的副作用，导致这些模式反复出现，因此它们值得注意。使用模式的目的之一是为了提高游戏的可玩性，这也可以被看作是一个设计挑战。

注意在下面的例子中相关游戏模式以黑体标出，大部分模式并没有在本文中提及。

成功就在眼前（Perceived Chance to Succeed）

描述

游戏通常提供实现特定目标的可能性，这是人们玩游戏的主要原因之一。如果一个游戏的既定目标不可能被实现，就说明其可玩性不高；反之如果目标总能被实现也是一样的道理。这在多人游戏中尤其明显。如果在一定时间后，能感觉到只有一位玩家是唯一的赢家，这样的游戏通常更吸引其他玩家。因此许多游戏试图让玩家始终感到赢得胜利是可能的，这包括赢得胜利所需完成的子任务也应是可能被完成的。

效果

如果玩家认为他/她总有机会完成目标，尽管可能性不大，也会激励他/她继续和对手战斗。这种态度进而会增加战斗的压力并使胜利更吸引人。

使用模式

当使用这种模式时，设计者需要警惕，除了游戏最终时刻，玩家的表现始终与游戏密切相关。玩家应该一直被激励着试图发挥他们的最好水平，这样才能增加胜利的机会。但直到游戏结束前，这个机会都不是必然的。在大多数游戏中，游戏最终结束前谁是胜利者会变得非常明显。而Monopoly (Parker Brothers)是打破这个模式的一个例子。使用**坚持到最后（Last Man Standing）**模式作为胜利条件的游戏也打破了这个模式，例如Magic: The Gathering (Wizards of the Coast)，即玩家一一出局，最后只有胜利者留下。

逐渐增加难度，例如提供**平滑学习曲线（Smooth Learning Curve）**。如果在完成任务的同时，允许玩家学习所需的所有技能，会让玩家觉得当前的子任务是可能完成的。例如**等级游戏（Level-based Game）**让玩家觉得他们能通过修练等级获得进步，当然游戏会设计为初级非常容易，而最后一级是新手不可能完成的。该类游戏的一个子类是玩家在游戏最终总会失败(Space Invaders, Asteroids, Tetris)，但却给玩家一种信念，就是他们总是有赢的机会。这带来一种“胜利就在眼前”的强烈感受。参考(Loftus&Loftus 1983)对这种现象的心理分析（间断强化Partial Reinforcement）。

平衡效果（Balancing Effect）可以被集成到游戏系统中，用于试图缩小玩家之间的差距。一些游戏采用**动态难度调整（Dynamical Difficulty Adjustment）**技术。例如，如果玩家经常失败，系统会使怪物很容易被击败，或增加玩家的生命值。同样的例子体现在众多赛车游戏中，领先玩家或被电脑控制的汽车会自动减慢速度，使其他玩家容易追上来。这甚至使游戏对领先玩家更具吸引力，因为它增加了竞争的难度。

相关模式

父类模式：无

子类模式：平衡效果，间断强化，平滑学习曲线

冲突模式：Early Elimination, King Maker, Last Man Standing

分析瘫痪（Analysis Paralysis ）

描述

分析瘫痪发生于当面临太多可能情况时，玩家希望获得对不同结果的预览，但该结果集是巨大的，因而对游戏产生负面影响。仅仅存在很多可能结果还不是这种模式产生的唯一原因；玩家必须认为分析当前的情况是可能的，而且这种分析会对该玩家有利。

效果

分析瘫痪强迫玩家花时间去决定该做什么，而不是单纯地与游戏交互。这可以导致游戏中**允许游戏主宰**（Allow Game Mastery） 和 **成功就在眼前**（Perceived Chance to Win）不再是单凭运气来决定的。对其他玩家而言，分析瘫痪意味着游戏没有**合理的等待时间**（Reasonable Waiting Times ），但可以降低紧张感，因为这些玩家没有需要在游戏中独当一面的压力。

使用模式

限制玩家分析游戏状态的可能性会去除分析瘫痪的影响。**常数级的玩家活动**（Constant Player Activity）和**移动**(Constant Movement)迫使玩家持续认为继续分析游戏中的反馈和时间是有益的，此时应使用时间限制参数来限定花在分析情况上所用的时间。所有这些方法并没有真正地为玩家解决分析问题，而只是简单地迫使玩家采取下一步行动，或使玩家受到负面影响。不以惩罚为威胁手段的温和模式通过执行**有限预测**（Limited Foresight 即只分析一定步数，而非所有可能结果）来实现，这样行动的后果将更难计算。

当某个玩家思考时，让其他玩家去实现一些明确的目标，这样可以减轻分析瘫痪的影响，而且不影响游戏的状态。有不同形式的**社交活动**（Stimulated Social Interaction）可供选择：如允许进行战术谈判，或联盟之间的交流，或游戏外的交谈，还有学习游戏规则或在等待时训练不同的游戏技能等。

相关模式

父模式：无

子模式：无

冲突模式：合理的等待时间，允许游戏主宰

共同目标(Mutual Goal)

描述

所有或部分玩家在游戏中试图完成同样的目标或子目标。当任何时刻有两个以上的玩家有完全一样的目标，就说明这种模式存在。例如：“我们都想让红车领先”而不是“我们都想让各自的车领先”（后者是**对等目标** Symmetrical Goals）。

效果

通过给出一个共同目标为几个玩家定义了一个胜利条件，这创建了**团队精神**Team Play。在游戏中作为子目标的共同目标可以用来加强团队合作的优势，或用来创建一个玩家的**临时联盟**Temporary Alliance。在所有情况下，共同目标促进了社交活动的进行。

使用模式

就象任何一个目标一样，共同目标可以是在游戏开始前预先定义好的已知或未知的目标，也可以是由玩家定义的有奖励的目标集合。后者的一个例子是外交活动（Avalon Hill），玩家能谈判和定义自己的共同目标范围，例如：可以是简单目标 “支持这支部队”，或宏伟目标 “防守意大利”。

未知的共同目标可以是其他玩家甚至具有该目标的玩家都不知道的目标。第一种情况导致其他玩家**秘密协作**（Secret Collaboration）试图揭露谜底；在第二种情况下，玩家具有**不明身份的盟军**（Unknown Allies），游戏的一部分是判断谁是你的战友，同时还不能让你的对手发现。Royal Turf (Reiner Knizia, Alea 2001)是这种模式的一个例子.在该游戏中通过赌马来决定输赢。下注（秘密地）后，玩家轮流沿轨道移动赛马。仅移动自己下注的马匹会暴露自己的战略，但同时玩家可以据此确定谁是同伴以便合作。

共同目标中的奖金分配（如果存在）对玩家协作程度的影响是至关重要的。**个别奖励**（Individual Rewards）可以制造紧张感，因为玩家争先恐后想达到设定的游戏状态；**分享奖励**(Shared Rewards)依赖于明确的共享规则来促进协作。后者促进协作的一个例子是自动分配经验值或在角色扮演游戏中的金子（尽管如何来分配宝物是由玩家自己决定的）。在奖金定义不明确的情况下，例如外交上的共同目标，这种目标可以导致玩家之间产生的紧

张感成为对其他玩家的目标价值，因此到底各玩家投入多大力量来实现目标是很难衡量的。

如果玩家能够与参与完成共同目标的其他玩家协商，这会带来**平衡效果**，因为落后的玩家更有可能组成临时联盟来对抗当前的领导者。然而，共享奖金的分配机制将大大影响协作结果。

相关模式

父模式：团队游戏，临时联盟，社交活动，叙述性结构（Narrative Structure），不断打破平衡（Punctuated Equilibrium）。

子模式：预先设定目标，未知目标，玩家构造的目标集合(Player-constructed Closures)，秘密协作，未知盟军，个别奖励，分享奖金，平衡效果。

冲突模式：对等目标，非对等目标（Asymmetrical Goals）。

分享奖励(Shared Reward)

描述

在游戏中以某种方式参与完成一个目标集合的玩家通常会分享某些形式的奖励。并不是所有分享奖励的玩家都必须参与协作来完成目标。例如，分享可能是根据对一次冒险的打赌结果来进行的。

效果

分享奖励为玩家交互开发出多种可能性。在通过**协作**（Collaboration）来分享奖励的情况下，分享奖励给予玩家一个**共同目标**，并促进**社交活动**。在玩家之间必须分享奖励的情况下，该模式鼓励玩家之间的**竞争**。更进一步，可以在竞争者之间分配的奖励具有一种特定的**平衡效果**，因为奖励不会分配给仅仅一个玩家。当玩家必须选择谁将和他们分享奖励时这一点尤其显著；通常玩家会选择一个比自己差得多的对手。

使用模式

分享奖励模式的使用主要依赖于是否设置了奖金额度和分配机制，或依赖于游戏状态。如果无论有多少人来分享奖励，玩家总会获得相等数目的奖励，那协作并不会使玩家有任何损失。然而(至少在真正进行奖励分配时)这需要一个**无限大的资源池**（Unlimited Resource Pool）。而且还可能导致**通货膨胀**。

如果奖金总额固定（有限资源池Limited Resource Pool），玩家必须互相竞争以争取更多奖励。对这类奖励而言如何定义分配机制将大大影响游戏的进行：先到先得的机制制造了一场**竞赛**；基于机会的奖励分配促使玩家**投资**；当分配奖金时，股份和奖金之间的预定关系鼓励玩家在**限定时间**内竞争。在最后一种情况下，**对投资的几何奖励(Geometrical Reward for Investment)**比**对投资的算术奖励(Arithmetical Reward for Investments)**制造了更多的竞争。注意虽然奖金总额是固定的，股份的数目却不必固定。

玩家可以通过自身价值或妥协来获得股份奖励。简单通过价值来决定玩家的股份意味着他们必须达到一定标准以便有资格控股，例如：拥有公司股票。通过妥协来分享意味着玩家不得不将自己的部分奖品给他人以便得到他们的奖品。例如：棋盘游戏*Kohle, Kies & Knete* (Sid Sackson, Schmidt Spiele 1994)就是一种交易性游戏，每次交易一定数目的钱。然而，交易可能很难仅被单方认可，交易中的主导者必须从其他玩家那里寻求帮助以求得回报。

实际上分享奖励可能是玩家之间谈判过程的结果。例子之一是**买卖**，它可以被看作一种分享奖励，因为参与买卖的玩家得到的利益高于未参与买卖的玩家。买卖规则中允许**诈骗**（Bluffing），分享奖金的实践内容有时是不确定的，会带来**不确定的奖励**(Uncertain Reward)。

相关模式

父模式：社交活动，协作，共同目标，竞争，平衡效果

子模式：买卖，对投资的几何奖励，对投资的算术奖励。

冲突模式：个别奖励

为实验性游戏设计启发创造性的工具（Creative tool for experimental game design）

在前文中我们提到，模式不仅可用来分析已存在的游戏，还有助于设计过程本身。使用模式的益处之一是设计者能够以清晰一致的方式表达设计问题，与以前的游戏相比，这样在做出设计决策时就有的放矢。

这一点在实验性游戏设计中尤为重要（参看http://www.experimental-gameplay.com）。在该情况下，设计目的是创造出打破已有游戏的传统类型、主题和风格。在基本元素和模块已知，并在某种程度上形式的情况下，即使创新不使用大家熟知的游戏元素，而是使用人们不熟悉的元素，创新和实验也较为容易。下面给出了这样做的一个例子，描述了一个设计者如何来设计一个手机上的小型多人游戏的原型。所用到的设计模式以黑体字标出。

主要设计需求是游戏中玩家之间应能进行**社交活动**。**交易**是该模式的子模式，例如玩家不得不相互交易，因此他们必须进行沟通，这样纯粹的形式的交流很容易变为社交活动。为了有物品可以用来交易， Source/Sink模式被用来定义基本游戏机制。由于这个游戏很简单，只需要定义四种不同的资源。在资源生成部分，即Source/Sink的source部分使用**非对等分配**模式(Falstein 2002)，这样每个玩家会随机得到一种资源。根据设计，玩家需要得到所有四种资源才算获胜，Source/Sink模式的sink部分，导致游戏中需要进行交易和讨价还价。

然而，仅使用交易模式并不够，还需要使用共同目标和分享奖励模式，以及社交活动的子模式，来创建一个系统。在该系统中玩家有相同的或互相支持的子目标，并且他们需要分享奖励才能完成目标（在该例子中奖励仅仅是积分）。

诈骗模式用于在玩家之间制造紧张感。现在玩家从一个确保会有重迭的集合中秘密选择他们自己的目标，他们的选择也以某种方式和资源生成器相关，这使得单个玩家是不可能独自完成目标的。因此，游戏设计要求玩家相互协作，甚至欺骗其他玩家，利用他们的资源来达到自己的目的。

其它一些适用于多人游戏的模式也用在最终设计中，如**观众**、**选举**、和**个性化**（Spectators, Voting and Personalization）。在这个实验中，即使是初级的模式在设计过程中也是很有帮助的。关于最终设计的更多信息请与Jussi Holopainen联系 （从上文中可以找到他的联系方式）。

模式示例

为了说明这些方法，让我们来看看一些待选模式。下面所选的模式非常简单，并不是为游戏设计特意选择的。

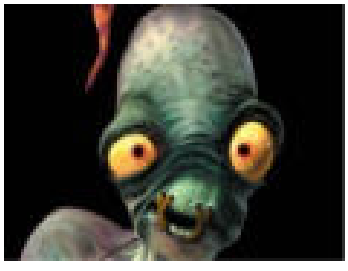
代理 PROXY

问题：将某种游戏行为直接应用于一个目标物体可能不是所期望的，或是不可能的。换言之，我们必须避免接近或冲撞一个所给行为的真正目标。

解决方案：引入另一实体（对象）作为代理位于目标物体之前。相对于目标物体，代理可以被独立地放置和移动。在缺省情况下，对目标物体的限制并不应用于代理。对同一目标可以使用多个代理。

效果：如果期望玩家来利用这个连接和间接关系，在代理和实际目标之间的连接必须告知玩家。用于代理和目标的限制应根据玩家的角度进行，而且可以应用从玩家控制角度的限制，以确保玩家能感到效果体现在目标上。

例子：*Munch's Oddyssee*有几个供玩家角色使用的代理：拥有orb能允许Abe来接管其他游戏角色（也就是将它们变为代理），Snoozer机器人和H-Crane是Munch的代理。玩家角色代理的一个好处是安全死亡：释放一个代理并不会使游戏结束。



用来开门或启动/停止机器的按钮/开关/控制杆等常被放在不同的地方，而不是作为目标物体的一部分。在这些情况下，设计者想从目标中分离触发器，以便增加游戏难度。Half-Life中的final boss（大鬼）的头部有一怕攻击的部位。该部位可以视为整个（不怕攻击的）boss的一个代理。Boss的第二个弱点是，它依赖于三块可以提供治疗和攻击的水晶。攻击水晶并摧毁它们，就是对怪兽进行了有效的攻击。

使用代理是为了说明一个潜在问题：在不同环境下使用的不同模式可能具有相同的名称。在软件工程领域，Gamma et al使用了同名模式，然而这两者之间没有任何联系。另一方面，在某些情况下，如果在设计环境中一个给出的游戏设计实体的意图和在组成游戏引擎的软件实现之间存在联系，一种给定的模式语言可以被翻译为相应的面向对象的实现。

可预测后果 PREDICTABLE CONSEQUENCE

问题：玩家必须能认识到她的错误会导致失败，而不是随机的、或预先定义的、即使行为合理也无法避免的无关事件，。

解决方案：如果一个可能行为的后果不能被预测，玩家可能无法决策是否该行动。一个有效的玩家决策是一个理智的决策：在行动前她必须能够猜测出行为的结果。选择游戏机制，根据玩家已获得的知识（来自实际生活经验，或其它游戏，或游戏的前几部分），通过可视方式来展示预测行为。不要破坏你的可视语言的一致性，例如，确保行为不同的物体其外观也不同。

效果：不要实现无法预测的惊讶结果。玩家可能依赖环境外的信息来预测设计者的埋伏，然后或者瞒过设计者，或发现由于设计者禁止进行某组操作，自己在执行某行为时被困。最后，设计者可能会设计一些危险行为，有经验的玩家从来也不会去执行它们，他们知道这些行为的恶劣后果。设计者也许不得不花很大力气来与玩家沟通管理行为后果的游戏规则。由于使用机制，设计也许会歪曲真实世界中的物体，以尽可能减少解释的必要。对玩家而言，缺乏不确定性使游戏更为透明，潜在的减少了探索和试探的需要。最终，对预测结果的需要由回溯（错误的严重性）的需要和回溯的成本（在选择和结果之间的潜在关系，参见行为科学中关于时间的问题）决定。另一个可能效果是在设计中可能不得不限制只允许有限的玩家行为，以便能够在游戏世界中使它们一致。只能一次有效或在很少几个地点有效的行为，其交流成本和在设计者/玩家部分的教/学成本都是无法折旧的。设计也可以夸大不同物体之间的区别以避免含糊。任何变化或范围都必须遵循易于观察和依从的规则。

例子：Doug Church的文章依赖于Mario64，可预测结果的教科书例子[11]。该模式可应用于任何具有一致的物理近似性的游戏。如那些在FPS游戏中被展示的巨大的手榴弹。其他例子如：柳条箱和木桶被标记为炸药；火球在击中时会伤害玩家，水会淹死玩家，熔岩会烧死玩家。反例也很多：写着“点按”的开关和按钮是用来释放一个必须的机关（该机关也会打开前进的唯一道路），如所预料的那样。在Opposing Force 和Ico中，这样的机关打开后游戏自动跳转到另一面。这里游戏引擎常常使用隔离物/平移机制

参考：Doug Church引入了被察觉/可察觉后果[11]的概念，这些概念也描述了可预测结果。对玩家行为的立即反馈的需求和玩家必须有机会预测自身行为的可能后果的需求是不同的。该模式源于后者，前者需要使用另一模式。

可预测结果是我对FADT“可察觉的后果（Perceivable Consequence）的改写，它由Doug Church（最初描述请参见[11]）提出。除了对描述本身的改变，我重定义了它的含义（根据我的认识），此候选模式说明FADT能够被表述为模式。

剪刀－石头－布 PAPER-ROCK-SCISSORS

问题：避免那种会使玩家的决策变得微不足道的主宰策略。

解决方案：在一组可选事务中引入非传递性关系，就像游戏剪刀－石头－布那样。

效果：使玩家不可能找到一个单独的适用所有情况、所有环境的最佳策略，她必须重审她的决策，依靠系统约束调整策略以适应情况的变化，或因过早决策而自食其果。

例子：该例子由Andrew Rollings提供，它是战士－野人－射手（warrior-barbarian-archer）的集合，源自Dave 和 Barry Murray的游戏*The Ancient Art of War*(Broderbund 1984)。他用类似的术语描述了Quake的武器/怪兽这样一种关系：Nailgun能打败shambler，shambler 可以打败 rocket launcher，rocket launcher 能打败 zombie，zombie 能打败 nailgun[28]。

参考：Chris Crawford（参看 "Triangularity" [15]）提供了第一份对使用非传递性关系的清晰描述。例子中Andrew Rolling的讨论使用的游戏理论包括详细的结局，以及非形式的虚构的设计者之间的对话、

剪刀－石头－布是一个反复使用的游戏设计工具，它大概是最早被半形式描述的模式之一。这里并没有新的内容；只不过是重新组织已有知识以使其与例子对应。该模式可以被扩展以包括对结局的描述和例子（超过三个相互制约的决策， 相关讨论参见[28]）。但即使是最简单的描述也已经体现了它的本质。

特权操作 PRIVILEGED MOVE

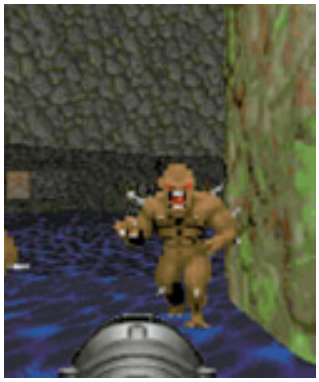
问题：有时不允许指定的游戏实体干扰其它实体，或者其它实体不允许干扰它。

解决方案：引入专属操作和地点来隔离和保护游戏实体。如果不执行特权操作，就不能进入或离开这些专属操作和地点。通过引入不同的、相互隔离的空间或介质，游戏环境被分割为不相交的区域，作为安全区或安全通道使用。

效果：依赖于如何实现，这可能是一种非常专制的保护方法。除非保护从玩家的角度来实现（一扇只有该玩家可以打开或关闭的门），玩家可能会认为约束是讨厌的和令人失望的。如果约束是暂时的，允许取消它的机关必须告知玩家，并由她来发动。

例子：在DOOM中,怪兽可以穿过硫酸池，但玩家却会受到伤害并很快死亡。在Half-Life和Halo中，敌军士兵可以被汽车载来，但玩家不可以使用运输工具，而且不能干扰敌军的这种操作。在Half-Life中，有一个象蜘蛛的boss，它可以冲破自己结的网，但玩家会被网拦住，直到玩家迫使怪兽撤退。在Thief中，玩家可以相对安全地穿过屋顶，并允许查看脚下周围的地形。在Ico中，幽灵对手可以不同维度穿过世界，自门口出现而不受明显的制约。在*Munch's Oddyssee*中，敌人在水中无法生存，但玩家享有特权可以游泳安全通过水域（在没有狙击手和地雷的前提下）。在许多游戏中，水军被限制只能在水中活动（例如*Quake*, *Half-Life*）。飞行物体（*Heretic 2*中的鹰身人

妖harpies）可以进行玩家所不能执行的移动，因此享有一定的安全特权。




特权操作是一种建议模式，它很少被讨论，但经常被使用在很多方面。一个单独的解决方案可以解决或针对多个问题；在Alexander的模式语言强调了面向问题的模式使用，而没有清楚地指出这种情况。特权操作和FILTER也关系密切。这个例子集合并不全面，如Gamma et al所指出的：“找到模式比描述它们要容易得多”。

过滤器 FILTER

问题：如果提供给玩家手段去创建游戏物体，和/或组合已存在的物体，和/或任意移动游戏物体，对设计者而言游戏的复杂度是相当惊人的。

解决方案：通过过滤器限制局部复杂度，以便能创建较小的、能够被清晰定义的可处理子集。

结果：玩家的自由通过显式或隐式的手段被限制。意外地过滤可能导致玩家无法解决的局面。理想状态下，对玩家来说过滤器是显而易见的，是游戏世界的固有部分。过滤器可以是特权操作的蓄意后果或无意带来的副作用。

例子：作为处理可能游戏状态的组合复杂度的一种手段，过滤器提炼自Jon Blossom对*DroidWork*的总结分析，“我们的level设计人员无法百分之百地保证什么样的droid会穿过房间，在许多情况下，他们创造出独特的物理过滤器机制，在一个level只允许特定类型的droid能通过：一个陡峭的山丘会清除两足的droid，一个深坑会清除有轮子、不能跳跃的droid，一个峡谷会清除宽的droid,一扇矮门会挡住高的droid。设计者使用地形作为障碍，来降低迷宫的复杂度。”

法兰克福香肠 WEENIE

问题：游戏世界如何展开可能会影响玩家并使其失去方向感。在前后衔接的玩家行动中，这是一个由玩家驱动的脚本处理的特定问题。

解决方案：游戏必须被安置得有条理，这样可以告知玩家该向哪儿走，走到目的地后应采取什么行动。

效果：随着时间，玩家将越来越依赖于指南的级别，如果忽略指南或丢失指南，玩家将会迷惑，不知道下一步该做什么，这导致偏离WEENIE CHAIN的结果。根据指南机制的硬性程度，玩家可能会渐渐意识到隐藏在游戏设置中的游戏议程。

例子：有时这被称为level设计中的“carrot on a stick”方法 [8]. Stephen Clarke-Wilson [13] 解释说：“Walt Disney构造了一个奇怪的术语，他建议在设计3D环境时（主题乐园），必须引导游览者以同样的道路穿过环境。训练狗的方法之一是在狗鼻子前吊一香肠。显然Disneyland中的法兰克福香肠是睡美人的城堡，这鼓励游客从主入口进入再到达中心；前者的Rocket Jets鼓励游客另辟蹊径；马克吐温轮船和码头鼓励游客探询边界；King Arthur 酒会鼓励游客沿城堡护城河进入幻境。[…]你的3D 虚拟环境需要有出色的地界标，这样即使没有地图也可以导航。最好的游戏设计通常只使用很少的图例，总是保存很少的图片来指示应该被调查的特殊和有趣的东西”。可选事物用以告知玩家所建议的或必须的下一步行动，这些信息由NPC传送，或用简洁的照片来吸引玩家的注意力到附近的门，象在*Munch's Oddyssee*中所使用的那样。Cliff Bleszinksi描述了最细微的部分并总是使用香肠来刺激玩家去发现新区域：玩家总是倾向于从“在这儿，完成这项工作”去探索未知领域[8]。

香肠链 WEENIE CHAIN

问题：在缺乏清晰指示下一步该做什么的时候，玩家可能会失去方向感。特别是当游戏在大的非线性的环境中（它不依赖于轨迹或切断道路以强制玩家移动），或使用再访区域时，这个问题尤其突出。

解决方案：创建不被打断的香肠链，链的每个环节都可清楚地感知到前者，用来指导玩家从游戏开始到结束。

效果：香肠链在使用分支时是有限制的，或反之引入含糊性或玩家选择。强制情况下，香肠链可以被重新组装成新的轨迹。

例子：Noah Falstein将 *Indiana Jones and the Fate of Atlantis作为例子来说明如何使用NPC使者来建立清晰的玩家目标链[18]。另一个例子是Super Mario 64和它所使用的标记NPC，以及*Halo。*Munch's Oddyssee同样也使用了标记，用*Shaman NPC来进行最初的指导。

来自许多渠道的待选模式都可以被接受。例如，FILTER模式原封不动地取自一个游戏的总结分析。WEENIE来自对主题公园的讨论和环境故事讲述技巧，并在level设计讲座中以不同的名字被展示。WEENIE也是香肠链的基础，解释了模式依赖、组合和层次关系。

一旦你开始寻找模式，你就会注意到它们无处不在。Erich Gamma et al.指出：不论他们自己是否意识到，小说家和剧作家总是在使用模式。许多“叙述性设备”（参见例子[32]）也可以使用模式模板来描述。

有时在抽象的不同层次对区分模式的需要导致对它们的不同命名：“一个人的模式可以成为另一个人的简单积木”（来自[19]）Alexander et al.展示了共253种模式，按“城镇，房屋和建筑[3]”划分为三类。根据所给目的的优点可以应用参数：模式只是处方、手段，而不是最终方案，对最终方案有争议不属于模式的范畴。设计者也会从不同的视角来评估一个方案的质量：一个人想使用的模式可能是另一个人所反对的 ——一个反复出现的错误设计决策的例子。理想情况下，模式描述本身只是对原因和结果的中立性描述，描述了达到所需目标的一种方法。

Alexander的模式对游戏开发人员来说并不陌生。很多程序员熟悉Gamma的软件设计模式概念。Will Wright赞赏这本书，称它为The Sims的创造灵感。至于Christopher Alexander的作品（也就是[1]），LucasArts的早期level设计者，Steven Chen 和 Duncan Brown建议level设计者应将Alexander的253种模式仅仅作为指南来创建“有意义的环境”[10]

取而代之的是，游戏开发人员努力去开发他们自己的形式和模板。Doug Church建议的形式抽象设计工具（FADT）可以很容易地被改写以适用于规范的模式模板。最近另一个将游戏设计知识以半形式的方法来文档化的尝试是Hal Barwood 和 Noah Falstein 所进行的“The 400 Project”。该项目的目的是收集经过证实的游戏规则和技巧。400 Project起源于Hal Barwood在2001年GDC上的讲座，在讲座中他举了四个例子。

他们的成果发表在*Game Developer*杂志的一个专栏中，在该专栏中Noah Falstein介绍了规则“提供清晰的短期目标(Provide clear short-term goals)”作为开篇。The 400 Project使用一个五部分的模板：命令语句－应用域－主宰规则－被主宰规则－例子别名(Imperative Statement - Domain of Application - Dominated Rules - Dominating Rules - Examples Aliases)（详细内容参见[18]）。统治概念（一些规则优于其它规则，而且这些规则可能轮流成为王牌）是Alexander模式中所没有的。一个Alexander模式简单列出对所给问题的可能解决方案，告知其它方案可能存在。Alexander将每个模式看作一个处方，该模式努力去平衡竞争命令和冲突力量，以根据特定目标获得最好结果。使用某个模式的决定也依赖于是否有多个目标需要实现，或应用某个解决方案时存在副作用，Falstein规则的影响显得更僵化，意味着所有游戏都有相同的目标、相同的利害关系。在400 Project中的规则名称是命令语句，就像提供清晰的短期目标，它们的描述仅仅包括模式的解决方案部分。模式所解决的问题仅仅被暗示或假定为显而易见的。香肠、香肠链和提供清晰的短期目标有部分共同点，而且也显式地声明了模式的角色。规则、模式名称常常表明了解决方案，而不是问题：一个匹配模式可以被称为SHORT-TERM GOALS。

展望

模式是文档化的形式手段，该手段为维护和编辑游戏设计文档的软件工具打开大门。以电影剧本为例：文字处理器可以被定制以处理脚本的规范格式，一些应用程序被设计为特别支持电影产业格式。许多游戏公司和游戏设计者发明了他们自己的游戏设计文档的内部标准。但是如果没有编辑和查找功能的支持和强化，为游戏设计文档定义一个标准格式并没有太大用处。

一种模式语言是对叙述性标识的自然匹配，例如XML，对基于现成软件提供工具支持有巨大的潜力。任何现存的文档的价值是与它是否易于维护直接相关的。结构化的编辑使思考条理化：如果一份设计文档条理不清楚，关键字和命名的使用也不一致，当需要时不能迅速找到相关信息，那这份文档就没有实用价值。

因此，游戏开发者必须坚持不断地努力，去定义和描述在日常工作中反复出现的元素，无论是使用模式、规则或其他方法。然后我们就能够创建软件工具来适应游戏设计的特定用途。Alexander游戏设计模式[22,29]的例子看起来很有说服力：它们向其他和不同领域的专业人员证明了自己；它们是直观的、充分文档化的、对软件工程师来说很熟悉的一个概念，而且足够灵活，可允许对艺术性选择进行风趣的或非形式的描述。

参考文献

[1] Christopher Alexander. *Notes on the Synthesis of Form*. (Harvard University Press 1964, 1966, 1979.) ISBN 0-674-62750-4 (cloth) ISBN 0-674-62751-2 (paper)

[2] Christopher Alexander, Murray Silverstein, Shlomo Angel, Sara Ishikawa, and Denny Abrams. *The Oregon Experiment*. (Oxford University Press, 1975.) ISBN 0-19-501824-9

[3] Christopher Alexander, Sara Ishikawa, Murray Silverstein, Max Jacobson, Ingrid Fiksdahl-King, and Shlomo Angel. *A Pattern Language: Towns, Buildings, Construction*. (Oxford University Press, 1977.) ISBN 0-19-501919-9

[4] Christopher Alexander. *The Timeless Way of Building.* (Oxford University Press, 1979.) ISBN 0-19-502402-8

[5] Brad Appleton. *Patterns and Software: Essential Concepts and Terminology*. Last update: Feb. 2000.

[6] Hal Barwood. "Four of the Four Hundred 2001". (GDC lecture, 2001.)

[7] Hal Barwood and Noah Falstein. "More of the 400: Discovering Design Rules 2002" (GDC 2002 lecture)

[8] Cliff Bleszinski. "The Art and Science of Level Design." (GDC 2000, pp. 107--118.)

[9] Jon Blossom and Collette Michaud. "Postmortem: LucasLearning's Star Wars DroidWorks" (Gamasutra 1999.) Originally *Game Developer* magazine, Vol 3, Issue 28, pp. 52-58, July 1999.

[10] Steven Chen and Duncan Brown. "The Architecture of Level Design." (GDC 2001 Proceedings, pp. 167--175.)

[11] Doug Church. "Formal Abstract Design Tools." (Gamasutra, 1999. Originally *Game Developer* magazine, Vol 3, Issue 28, July 1999.)

[12] Doug Church. "Abdicating Authorship: Goals and Process of Interactive Design." (GDC 2000, San Jose, Lecture 5403 (not in proceedings).)

[13] Stephen Clarke-Willson. "Applying Game Design to Virtual Environments" (*Digital Illusion*, ACM Press, Vol. 2, Issue 1, January 1, 1998.)

[14] (N/A).

[15] Chris Crawford. *The Art of Computer Game Design*, Chapter 6: "Design Techniques and Ideals." 1984.

[16] Troy Dunniway. "Using the Hero's Journey in Games." (Gamasutra, 1999. Originally published in *Game Developer* magazine, August 1999.)

[17] Troy Dunniway. *Professional Game Design.* (New Riders. To be published June 2002. ISBN 0-7357-1184-4.)

[18] Noah Falstein. "Better By Design: The 400 Project". *(Game Developer magazine*, Vol. 9, Issue 3, March 2002, p. 26.)

[19] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. (Addison Wesley Longman, 1994.) ISBN 0-201-63361-2.

[20] Chris Hecker and Zachary Booth Simpson "Game Programming Patterns & Idioms." *Game Developer magazine*, Sep. 2000.

[21] John Hopson "Behavioral Game Design." Gamasutra, April 2001.

[22] Bernd Kreimeier. *Game Design Patterns*. Wordware Publishing, Inc. To be published March 2003.) ISBN 1-55622-967-4

[23] Brenda Laurel. *Computers as Theatre*. (Addison Wesley Longman, Inc. 1991, 1993 ISBN 0 -201-55060-1.)

[24] Marc LeBlanc. "Formal Design Tools: Emergent Complexity, Emergent Narrative." GDC 2000, San Jose, Lecture 5304 (not in proceedings).

[25] Gerard Meszaros and Jim Doble "A Pattern Language for Pattern Writing"

[26] Pierre -Alain Mueller. *Instant UML* . (Wrox Press, Ltd., 1997) ISBN 1-861000-87-1.

[27] Karen Pryor. *Don't Shoot The Dog!* (Bantam Doubleday, 1999) ISBN: 0-55338-039-7 (revised paperback edition)

[28] Andrew Rollings and Dave Morris. *Game Architecture and Design*. (The Coriolis Group, 2000.) ISBN 1-57610-425-7

[29] Andrew Rollings and Ernest Adams. *Patterns in Game Design*. (The Coriolis Group, to be published May 2002.) ISBN 1-57610-873-2

[30] Richard Rouse. *Game Design: Theory & Practice*. (Wordware, Inc., 2000) ISBN 1-55622-735-3

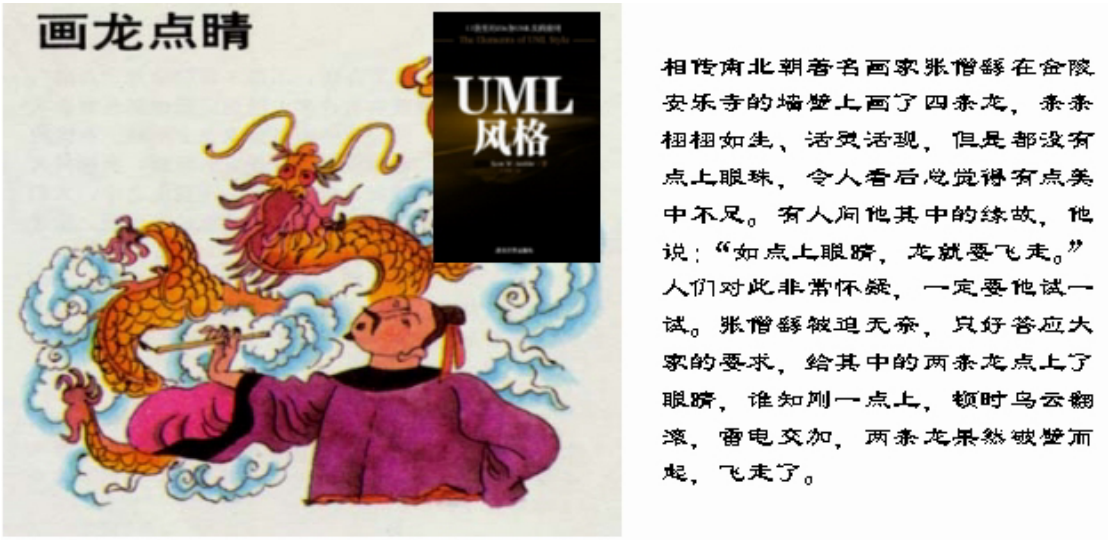
[31] Aamod Sane "The Elements of Pattern Style." December, 1995.

[32] Viktor Shklosvsky. *Theory of Prose Dalkey*. (Archive Press 1990, 1991.) ISBN 0-916583-54-6 (cloth) ISBN 0-916583-54-6 (paper).

[33] Zachary Booth Simpson "Design Patterns for Computer Games." 1998 CGDC Austin, TX, November 1998, also San Jose, CA, May 1999.

[34] John Vlissides. "Pattern Hatching - Seven Habits of Successful Pattern Writers." *C++ Report.* Nov/Dec 1996, and *Pattern Hatching: Design Patterns Applied* (Addison Wesley, 1998).

[35] Christopher Vogler. *The Writer's Journey: Mythic Structure for Writers*. (Michael Wiese Productions, 1998.) ISBN 0-941188-70-1



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。

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

UML 相关工具一览 A-I（截止 2005 年 3 月）

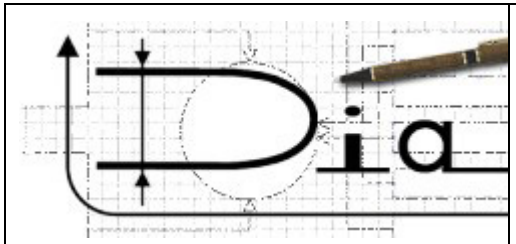
UMLChina 整理

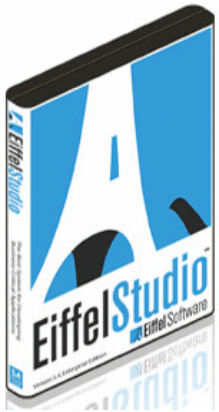
以下总结了全世界的各种 UML 相关工具，按工具名称字母排序。 为 MDA 标记。

工具（最新版本）	厂商&地址	试用允许	UML 版本	支持代码环境	XMI	平台	备注
Ameos	Anoix ttp://www.aonix.com/ameos.html	有演示版下载	2	Java		Linux, Solaris, Windows	实时嵌入式系统建模工具。
ArcStyler 4.0	Interactive Objects Software（德国） http://www.io-software.com/products/arcstyler_overview.jsp		2	Java, C#, Web Accessors, EJB 1.1, EJB 2.0, BEA WebLogic 7.0 (EJB 2.0), JBoss 2.4.4,, ASP .NET			覆盖 J2EE/.NET 系统开发工作流程的套件，遵循 RUP，有针对 Rose 的插件。
ArgoUML v0.13.1	Tigris.org http://argouml.tigris.org/	开源	2	Java	√	Java	最流行的开源 UML 工具，支持 OCL，支持认知式开发，不再只是画图，例如可以自动评价设计、自动更正…等等。

Nucleus BridgePoint Nucleus UML Suite	Accelerated Technology http://www.projtech.com/embedded/nuc_bridgepoint.html	有评估版	**2**	C, C++		Windows, Solaris	专门针对嵌入式系统的 MDA 工具，使用 XT UML（UML2 的一个子集），可直接运行模型。
Konesa 2.2 Konesa	Canyonblue http://www.canyonblue.com/products.htm	免费	**1.4**	Java, C++		Java	基于 Internet 的 UML 建模实时协作工具，支持协作建模和知识共享。
ClassBuilder 2.4	Jimmy Venema（荷兰） http://members.lycos.nl/JimmyVenema/ClassBuilder/ClassBuilder.htm	免费		C++	✕	Windows	专门针对 C++用户。精细的顺序图编辑器。可以以 RTF 和 HTML 格式产生文档。
Codagen Architect 3.2 Codagen Architect	Codagen http://www.codagen.com/products/architect/default.htm	15 天评估		VB, C#, C++, Java		Windows	遵循 MDA 流程，能产生超过 90%的 J2EE 和.Net 平台代码。2003 年获得 Jolt Productivity Award。 支持 Rational Rose 2000e 或更新版本，Rational XDE 2003，Borland Together ControlCenter 6.0，或带 Visio 的 Visual Studio .NET for Enterprise Architects。
Code Logic 2.1 LOGIC EXPLORERS	Logic Explorers http://www.logicexplorers.com/products/codelogic/details.html	有试用版		Java, C#		Java	强有力的、动态的双向工程，特别是从代码到顺序图的自动生成。

CodeModeler 1.6	Aruba Development http://www.arubadev.com/	30 天试用版				Windows	
Cohesion 1.0.2	Team Synergy（澳大利亚） http://cohesion.it.swin.edu.au/teamb/cohesion.shtml	开源				Java	元模型建模工具，可以使用 UML，OCL，或者其他喜欢的建模符号。
Composum 1.3	IST http://www.ist-dresden.de/products/Composum/index.html	可以试用			√	Java	可以导入 Rose 模型
Cradle 5	3SL（英国） http://www.threesl.com/						强有力的需求管理和基于模型的系统工程环境，支持 UML。
Describe 6.1.6	Embarcadero http://www.embarcadero.com/products/describe/dedatasheet.asp	15 天试用	**2**	C++, Java, C#, VB, CORBA IDL,	√	Windows, Solaris	自称为“IMDE”（集成建模和开发环境）。双向工程以垃圾代码少而颇受好评。提供对 Visual Studio .NET 2003 和 Sun One Studio Java 开发平台的集成支持。还可以和 ER/Studio, Caliber, DOORS 集成，支持协作开发。
Developer-EP	EmbeddedPlus PolySpace http://www.embeddedplus.com/UMLVeriBroc.php			C++			基于需求的 UML/C++验证试环境，提供基于模型的验证。
Dia 0.94	Alexander Larsson http://www.gnome.org/projects	开源		C++, Java, CORBA IDL,		Linux(Debian,	类似 Visio 的工具。还支持 E-R 图

	/dia/					Redhat),Windows	
DocExpress 3.3 Telelogic DocExpress	Telelogic http://www.telelogic.com/products/additional/docexpress/index.cfm						Telelogic 套件的一部分，可以与 Tau，Rose，Paradigm Plus 和 Aonix StP 结合产生可裁剪的 Word、Interleaf、RTF、Framemaker、HTML 文档。
Documentator	Henk Rippen（德国） http://www.rippen.de/de/produkte/documentator.htm					Windows	可以从 Rose 或 Select Enterprise 产生 Word 文档。
DOME (the DOmain Modeling Environment) 5.3 DOME Official Web Site	Honeywell http://www.htc.honeywell.com/dome/index.htm	开源			×	Linux, MacOS, Windows, Solaris	元模型工具，如果需要添加你自己的标记法时很有用。有 ftp 站点供大家交换模型。
EclipseUML 2.0 BUILD EclipseUML Studio 2.0.0 - Mon, 28 Feb 05	Omondo http://www.omondo.com/	有免费版	**2**	J2EE	√	Java	和 Eclipse 及 CVS 集成的 UML 工具。可以从字节码逆向工程到类图和序列图。
Eiffel Studio 5.5	Eiffel Software http://www.eiffel.com/products/studio/	有试用版		Eiffel		Linux, MacOS, Windows, FreeBSD	按契约设计的工具，基于简化版本的 UML 和 Eiffel 语言，作为 Visual Studio 2005 的插件。

							
Ensemble Glider 	Ensemble Systems http://www.ensemble-systems.com/products.html	有试用版		Java		Java	Rose、Eclipse/WSAD、XDE 插件，支持 BEA, Weblogic, Tuxedo, SAP
Enterprise Architect 4.51 	Sparx Systems（澳大利亚） http://www.sparxsystems.com.au/	有试用版	**2**	C++, Java, C#, Delphi, SQL-DDL, VB.NET, VB	√	Windows、Linux	便宜又好货。还包含需求管理、项目估算、测试支持。
ESS-MODEL 2.2 	Eldean AB（瑞典） http://essmodel.sourceforge.net/	开源		Delphi/Kylix, Java	√	Windows	支持 Delphi/Kylix, Java 双向工程
FL 0.4.12 	Novosoft（俄罗斯） http://www.novosoft.us/solutions/tools.shtml	免费		DB/2, Oracle, Interbase, Microsoft SQL Server, MSAccess, PostgressSQL		Java	Rose 插件，简化基于 Java 的关系数据库应用开发。简化持久对象的生成，使用 OQL（对象查询语言）访问数据库。

Flywheel 7.2	Velocitis http://www.velocitis.com/	30 天试用		C#、VB.Net	√	Windows	UML 到.NET 代码的映射，完全结合 VS.Net 2003。可视化重构。
FUJABA 4.3.1	University of Paderborn Software Engineering Group（德国） http://www.uni-paderborn.de/cs/fujaba/	开源		Java		Java	学院派作品，支持 Patterns
GRADE Modeler 4.0	GRADE Development Group（拉脱维亚） http://www.gradetools.com/default.htm	有试用版				Windows	业务分析和系统分析工具，擅长复杂模型图的界面显示、界面操作，为你自动整理纷乱巨大的图－还带语音功能。可以和 Rose 交互。
HAT	E2S（比利时） http://www.hoora.org/		**1.3**	C++	×	Windows	HOORA（Hierarchical Object Oriented Analysis）方法原来是为欧洲太空总署（ESA）开发的一种面向对象方法 m，提供了如何使用 UML 来开发软件的清晰指南。HAT 严格遵循 HOORA，可以和 Rose 交互.
INNOVATOR 8	MID GmbH（德国） http://www.mid.de/de/innovator/object/			Java ， C/C++ ，Smalltalk，Forte，Object COBOL，IDL，VB	√	AIX，DEC VMS ，HP-UX ，Linux ，OS/2 ，Solaris ，Windows	可以和 BPR 工具集成，良好集成版本控制工具（PVCS, Clearcase…），自动产生 Word, FrameMaker，PS 文档。

IRIS 2.0 IRIS™	Osellus（） http://www.osellus.com/						提供 RUP 桥接（RUP-Bridge）技术，RUP 剪裁和部署工具。

*有一些上一次总结曾经列出的软件因没有更新或者不再存在而被废弃。

ARIS 6.1	IDS Scheer（德国） http://www.ids-scheer.com/english/index.php			Oracle，SAP	×	基于 Web，平台无关	强有力的业务流程套件，为业务流程设计引入 UML。
BetterState 6.1	WindRiver http://www.windriver.com/products/betterstate/index.html	有 Lite 版		C, C++, Java		Windows	在 UML 状态图或 PetriNet 和代码双向工程，可直接运行在嵌入式操作系统如 VxWorks、OSEKWorks 平台上
Bold 3.2	BoldSoft(Borland) http://www.borland.com/			Delphi, C++, COM, XML, SOAP		Windows	BoldSoft 原为瑞典公司，2002/10 已被 Borland 收购。功能已并入 Borland Delphi 和 Borland C++ Builder 中
Development Accelerators	Blueprint Technologies http://www.blueprinttech.com/Products/Accelerators.asp						模式&框架工具，与 Rose 整合，支持 Gamma、Buschmann、Fowler、Hay 的模式，也可以自定义模式和框架，支持模式的模糊查询。
devine	Tom Jones http://www.tojosoft.de/Produkte/dvine/dvine.html	有试用版		Delphi		Windows	在 Delphi 代码和 UML 之间转换
Delphia Object Modeler（D • OM）	Atos Origin（法国） http://www.si.fr.atosorigin.com/rhone-alpes/Dom/				√	Windows, Unix (Solaris,	UML 原型工具，支持到目标架构的转换规则。支持 HTML 和 RTF 文档化。

						Linux, AIX, HP-UX)	
EctoSet Modeller 2.2	EctoSet（澳大利亚）  http://www.ectoset.com/	有试用版		Delphi/Kylix, C++ Builder, Java, VB	√	Windows, Linux	强有力的内嵌工具
Elixir CASE 1.2.4	Elixir Technology（新加坡） http://www.elixirtech.com/	有试用版		Java	√		
Enterprise FrameWork	Ptech Inc  http://www.ptechinc.com/						基于知识库的业务流程建模、BPR 工具。
FreeCASE	FreeCASE Project http://www.freecase.seul.org/details.html	开源				Windows, Linux	一个开源项目，已经停止了。
Ideogramic UML 2.3.3	Ideogramic ApS（丹麦） http://www.ideogramic.com/products/uml/	有试用版		Java，C/C++	√	Windows, Linux	关注“用手建模”的 UML 建模工具，强调创造性和弹性。支持电子白板，支持在桌面、可移动物体上建模。（似乎已经停止更新）