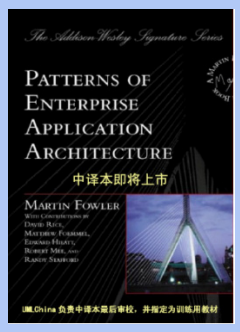
第 33 期（2004 年 1 月）



PEAA

X 非程序员
X-Programmer
软 件 以 用 为 本

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

http://www.umlchina.com/

感谢您

的支持与信任

祝福您

快乐、 健康

WWW.UMLCHINA.COM

《财富》把《人月神话》列为 CEO 必读技术书籍之首

[2004/1/16]

2003 年第 12 期《财富》中译本刊登文章《伟大的书》，作者 Mark Athitakis，文中列出了从古到今的一些书籍，作者认为是每位首席执行官都应该阅读的，其中“技术产业”类图书有 5 本，《人月神话》排在首位。


作者：MARK ATHITAKIS

伟大的书

代又一代成功的 CEO 和企业家都把自传定格为“当今的霍雷肖·阿尔杰式神话”（Horatio Alger，美国作家。阿尔杰式神话指只要勤奋，就能白手起家而致富——译注）。自传中讲述的故事内容如出一辙：野心勃勃、抱负不凡的年轻人最终获得了财富与尊敬，他的成功取决于不懈的努力、敏锐的洞察力、持之以恒、高尚的道德、明察秋毫、悉心保持个人卫生和从不缺乏的个人品行。

但问题是，霍雷肖·阿尔杰的故事情节并非如此。不错，阿尔杰的通俗小说——写于 19 世纪后期，正值美国工业高度繁荣——颂扬自力更生、依靠自己的努力出人头地，但故事所描述的却往往是冥冥之中难以把握的运气带来大笔钱财。有一个阿尔杰小说主人公救了一名溺水儿童，孩子父亲对他印象很好，于是给了他一份工作；另一个主人公持住一辆失控滑行的婴儿车，里面的婴儿正是毕尔街经纪人的女儿；还有一个主人公发现一大块金子……阿尔杰写道：“正是这么幸运的机会，让人受惠而发迹。”阿尔杰笔下的所谓依靠自己的努力出人头地也不过如此。

如果要从中吸取教训的话，除了你应该做银行家的亲戚外，那就是：你不能真的听信某些有关“经典商业书籍”的说法，最佳的办法是学会自己挑选和阅读。

我们推荐书目的意义就在于此。我们像辛勤工作的图书管理员一样，帮你仔细阅读书库里的书，找出你所需要的书。它们向你提供了对今天商业世界最为有用的观点和见解，有写于 2000 年前的，也有几个月前刚刚完成的。需要说明的是，我们选择的书目与 MBA 课程有所偏离。部分原因是，我们认为它们更具趣味性，而且这些小


越来越重要，也越来越难。库尔兹简地介绍了迅猛变化所产生的影响以及技极速度增长的事实。”

（Extraordinary Popular Delusions and the Madness of Crowds）
作者：查尔斯·麦凯（Charles Mackay）

麦凯写道：“金钱总使人产生错觉”。此书初版时并未得到重视，但后来被人们发掘出来。作者通过郁金香狂热和追逐巫师的故事，隐喻充斥当今整个市场的病态发狂和潮流。

《疯狂、惊恐和崩溃》（1978）（Manias, Panics, and Crashes）
作者：查尔斯·P·金德伯格（Charles P. Kindleberger）

从 18 世纪的“南海泡沫”到 20 世纪 30

技术产业

《人月神话》（1975）（The Mythical Man-Month）
作者：弗雷德里克·P·布鲁克斯（Frederick P. Brooks Jr.）

该书的写作接近打卡时代结束，它对软件开发复杂性的分析仍未过时，极具可读性。如果你不明白为什么你所得意的技术方案滞后、古怪或超出预算，就读这本书，它会给你颇有说服力的解释和可能的解决办法。

（think 摘录，不得转载用于商业用途）

IDS Scheer North America 发布 ARIS UML Designer

[2004/1/12]

IDS Scheer North America，业务过程领域杰出的服务和工具提供商，今天揭开了其工具 ARIS(C) UML Designer 的面纱，该工具支持业务驱动的软件开发，是 IDS Scheer 的过程平台的一部分，并加强了 ARIS 从过程到应用（Process-to-Application ，P2A）的能力。

IDS Scheer North America 的 CEO ，Mathias Kirchmer 博士认为，软件组织现在面临的一个问题是，需要把必要的业务过程转换到软件上来，不丢失信息，并且要对它们进行调整。“我们为 IT 专业人才提供工具，（利用这些工具，）他们可以将业务模型中包含的业务逻辑无缝地实施到可执行的应用中，诸如 ARIS UML Designer 等 ARIS P2A 的解决方案提高了 IT 系统的一致性，降低了维护和调整的费用。”

ARIS UML Designer 可以和 ARIS 工具集联合使用，后者是业务领域建模与分析领域领先的解决方案。ARIS UML Designer 具有将 UML 模型连接到业务过程模型的能力，并提供了从业务过程需求到软件开发的追溯能力。“在一个交互的环境中提供了这种连接，并支持多用户的产品，ARIS UML Designer 是市场上唯一的一个”，Kirchmer 补充，“这个解决方案可以和 ARIS 脱离开来独立使用，也可以和 ARIS 工具集捆绑使用。通过一个辅助将业务需求翻译为系统需求的通用语言，这个平台可以把 IT 和业务整合在一起”。

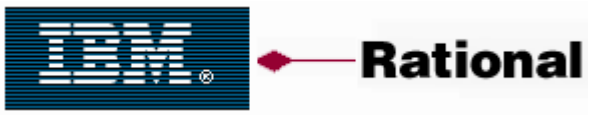
由于 ARIS UML Designer 是一个基于 WEB 的工具，项目组可以基于一个公共的仓库进行工作。ARIS UML Designer 支持 OMG 的 UML 2.0 标准中的八种 UML 技术。在 UML Designer 中创建的模型可以通过 XMI 形式导出，从而可以导入到其它的建模工具中。ARIS UML Designer 提供了业务过程设计和面向对象的软件设计之间的桥梁。Kirchmer 总结，“当前的 UML 建模工具都被设计来为软件工程师服务，而业务用户很难用 UML 来表示他们的需求。我们为业务用户和软件开发人员开发了基于一致的信息仓库进行工作的工具，保证更高质量的软件需求”

（自 businesswire，袁峰 摘译，不得转载用于商业用途）

IBM 收购 Rational 一年总结

[2003/12/26]

12 月，又到年关总结的时候了，这次，是该 IBM 给这个耗资 2.1 亿美元的交易作评价了。



这次收购帮助 IBM 增加了在使用面向服务架构（SOA）发布 Web Service 方面增加了筹码。SOA 指的是计算实体如何相互交互以使得一个实体可以代表其他的实体执行一系列的工作。

这种交互性，和可靠性、安全性以及可管理性一样，都是建筑和发布成功的 Web service 系统的基础。

Rational 以及微软、Boland 等公司都正在努力开发支持 SOA 开发的工具。XML 和 Web service 的研究组织 ZapThink 最近预测，2010 年，基于 SOA 的产品的市场总额将达到 43 亿美元。

在 2003 年 2 月的这次交易之后，Rational 分部面临的一个首要问题就是：在越来越靠向 IBM 基于 Java 的 WebSphere 平台以及开放的 Eclipse 框架之后，他们是否还需要继续支持微软的.NET？

在最近和 internetnews.com 的会见中，Mike Devlin，Lexington 的合伙创办人和管理者，回答了上面这个问题，以及一些其他的问题。

Q：Rational 的方法、信息和概念如何在 IBM 得到发扬光大？Rational 如何保持自己的平台独立性，同时又能够保证 IBM 在构建面向服务架构方面的竞争优势？

A：Rational 要负责 IBM 软件开发平台方面的规划和实现。这意味着我们要为核心的工具负责，不仅包括 Rational 的工具集，也包括 Eclipse 的核心部分和 WebSphere Studio，当然，我们不需要开发所有这些工具。

IBM 的很多部门和业务伙伴、顾客都在开发新的工具集，因此我们将对整体的架构以及市场产生强大的影响。并且成为这个领域的主要驱动力量。但是，我们必须要是整个软件组织架构的一个部分，因此组织其他的部门可以在这个同样的开发平台上开发其他的工具或者技术。

对面向服务的架构来说，我们所做的部分工作，首先是和 WebSphere 合作，然后和 Tivoli 等合作，保证 Rational 的统一开发过程（Rational Unified Process）可以定义为 WebSphere 应用服务器等不同 IBM 技术的开发都可以使用的开发过程，并包括对 SOA 等的支持。

我们已经在很多规约中定义这些能力。举例来说，在我们在用户会议上做的基于资产的开发演示是基于一个使用 WebSphere 应用服务器和 Websphere Studio 的 SOA。我们将 Tivoli 集成进来，作为发布、监控和管理 SOA 的工具，并且为操作者以及我们这些开发人员提供了可视化的途径。我们从开发的视角出发关注对 SOA 的驱动，使用诸如 WebSphere 等平台来实现这些。

Q：在 Rational 日渐紧密地和 IBM 的产品绑定的情况下，Rational 面向微软的那些产品例如 XDE 以及和和.NET 平台的交互性将何去何从呢？

A： 一般说来，我们的顾客使用的是混合的环境。包括微软的技术，同时也包括一个服务器的环境，其中可能有微软的服务器以及其他的操作系统。我们的承诺是：支持对我们的顾客来说重要的所有平台。

最明显的例子就是我们的 XDE 产品，他们完全集成于[微软开发环境] Visual Studio，就象和 Eclipse 的集成一样。我们还在继续改善这些产品。今年我们将介绍 XDE 中对.NET 的新的支持，他将象 VS .NET 的一部分一样工作。我们还继续保持在 Redmond 的工程组，这样以便保持和微软的紧密联系。

我们将继续保持 Visual Studio 集成拍档的身份，而且微软将继续给为我们尽早接触他们的新技术提供便利，这样我们可以为其提供更好的支持。并且微软还将继续支持联合的市场行动，包括 TechEd 和 PDC 等。只要微软的平台对我们的顾客依然重要，我们就会继续在上面大量地投入，至少和现在一样。

Q：公司将会如何继续革新，而且将在哪方面保持竞争力？现在 Borland 已经是市场上唯一的一家平台无关的 SOA 工具提供商了。

A： 革新的方面很多了。不管你是否愿意，这个部分上是要取决于投资额的。我不知道总的投资有多少，但 Rational 在这个领域的研发上投入了超过 150 个亿。

现在我们是 IBM 的一个部门，总共的研发投入相当多了。因此，我们在这些产品的技术革新上投入了很多资金。这还不包括 IBM 研究院，他们同样干了很多年了。我们有一个 2 到 3 年的计划，他们的计划是 3 到 5 年的。这样，我们就既有短期、中期的投入，也有长期的投入。

革新的另外部分，也是更实际的部分，是面向我们的顾客的，在成为 IBM 的一个部分，成为市场的领袖之后，我们可以看到顾客是如何使用这些技术的。我们直接从他们身上学习。原来 Rational 只是一个独立公司的时候，我们接触到的只是一小部分主要的用户。而现在我们要接触到 是更大的用户群。你们将会看到产品的巨大进步因为我们投入了很多资金，而且我们有了一个革新的更好的平台。

Q：你们怎么维护在基于 Eclipse 的工具中的竞争优势？

A： 不论什么时候，只要你采用了开放的标准，两件事情是必要的：你需要使你的竞争对手可以使用你的技术来开放好的产品；另外，你要加快产业化的步伐。

低端产品总是面临着这些情况－主要的厂商今天都面临这些问题－3 到 4 年后，这些都将变成低端的产品。因此，你需要掌握控制权。

我们想把这个工作落到实处，因此我们发明了 UML，我们开发了生命周期的管理套件，开发了 Rational 统一过程，现在，我们正在做基于资产的开发（asset-based development）和 SOA。

很多分析人员和我们的竞争对手都很奇怪，我们把 UML 做成了一个行业标准，这意味着我们所有的竞争对手们都可以使用它。但我们的观点是，如果你是市场的领袖，这就是一个好的策略。我们希望把它做成行业标准一对，它是帮助了我们的竞争对手，但最重要地，它给了我们顾客对我们的信任，并保证了市场占有率的提高。

Q：听起来你们将和 IBM 保持一致。

A：对，这是我们的文化。我们的信念是开放的标准、开放源代码。这确实帮助了我们的竞争对手，但也帮助了用户，开拓了市场并增强了革新的步伐。

Q：我们聊聊面向模型的开发吧，Rational 工具集的下一步将是什么？

A： 我们在这方面已经有所打算，而且有很多事情要做。我们在努力支持用户创建 SOA 的企业架构，并维护其一致性和可测试性，并尽力加强自动化的程度，允许用户使用模式，另外，也同样允许用户基于这些 SOA 快速开发新的应用。

你可以和了解业务细节的开发人员合作开发应用。他们不需要了解分布、安全和 SOA 里面涉及的各种技术细节，因为你的架构里面已经都有了。你们在其基础上开发得到最后的应用。

因此，MDA 是我们的一个关键领域。现在这些术语还处于被接受的早期阶段，但我们也在寻找一些用户在这个方向的例子，我相信，如果有大的 MDA 应用出来的话，将是非常激动人心的。

（自 internetnews，袁峰 摘译，不得转载用于商业用途）

Grady Booch 在 IBM

[2003/12/16]

最近的 IBM Rational 媒体日上，IBM 声称将基于 Rational 的工具集，主要是 UML 建模工具，采取更加集成的开发方法。这次报道也给了 Grady Booch 一个露脸的机会。

“开发是，而且还将继续是一件很艰苦的事情”，Booch 认为，他是 UML 的提出者之一，“看看有什么可以提高开发效率的办法：模型的使用、资产变更管理、开发生命周期内的测试驱动以及使用 Eclipse 的通用开发经验”。

建模可以架起技术和业务计划之间的桥梁，但这并不是纯粹的 pedal-to-the-metal UML。Booch 指出，IBM Rational 不反对提供一些 wizards 以便那些不想了解 UML 的人可以建模。但他认为，业务规则将很可能是建模改进的下一个目标。

在合并后的 IBM Rational 公司，曾经的首席科学家 Booch 成为了 IBM 的一员。他高兴地说，他的工作就是“打破官僚作风，做一个职业的自由的激进分子，我发现了很多乐趣……而且还有报酬”，Booch 曾为 IBM 研究院工作过，他觉得那段经历就象“在糖果店中的孩子”。去年，他认为这些是“有可能的，但得碰机会”，而明年，这种快乐将肯定会继续，他的意思是 Rational 将会开展更多的研发项目，为其他的开发和设计工具的项目提供服务。

（自 adtmag，袁峰 摘译，不得转载用于商业用途）

Grady Booch 加入 Northface 大学顾问委员会

[2003/12/15]

盐湖城，Northface 大学，今天宣布 Grady Booch 已经加入其顾问委员会。Booch 是 IBM Rational 的首席科学家，是提出 UML 的三友之一，并出版了很多著作。Northface 大学能够颁发计算科学和商业管理硕士学位。

“Grady Booch 给我们带来软件架构、建模和软件工程方面 20 余年的丰富经验”，Northface 大学校长 Graham Doxey 说，“非常荣幸能邀请到 Booch 先生作为新的顾问委员会成员。这样一位软件建模和开发领域的资深专家的加入能够保证学校在创新课程上的努力和课程质量，这些课程主要是模型驱动开发方面的，它们将和项目实践结合起来展开”

Booch 是六本畅销图书的作者，其中包括“UML 用户指南”和“面向对象分析和设计”。另外，他还发表了几百篇软件工程方面的技术文章，其中，他在 80 年代初发表的那些论文是面向对象设计实践的先锋之作。

Booch 同时是 IBM 的一员，ACM、世界技术网络的一员，以及软件开发论坛的空想家（Visionary，译注：这里的意思应该是指 Booch 对软件开发的一些设想比较超前吧）。他是 ACM（Association for Computing Machinery）和 IEEE（the Institute of Electrical and Electronics Engineers）、AAAS(the American Association for the Advancement of Science)、CPSR（Computer Professionals for Social Responsibility）的成员，另外，他还是敏捷联盟、the Hillside Group 和世界软件架构研究院（the Worldwide Institute of Software Architects）的创始人之一。

（自 businesswire，袁峰 摘译，不得转载用于商业用途）

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

Scott Ambler 访谈

Clay Shannon 著，李巍 译

查看评论

Scott Ambler 在新西兰

Scott，你在首页（http://www.ambysoft.com/）上说你加入了 Ronin International（敏捷建模之家）。你在 Ronin 有多长时间了？

1999 年夏天，我最初创建了 Ronin International 公司（www.ronin-intl.com），并一直发展到现在。

受雇 Ronin 之前你在何处谋职？

以前我作了多年的独立咨询顾问工作，此前我在一家多伦多业界领先的 Smalltalk 顾问公司就职。过去很久以前我为一家大的加拿大银行的 IT 部门工作。

简单地讲，什么是敏捷建模？它可以代替或扩展哪些类型的建模方法，以及它有什么特殊之处？

敏捷建模（AM），www.agilemodeling.com，是一种乱中有序（Chaordic）的、基于实践的进行有效建模和文档管理的方法。你可以将这些实践裁减到其他像 Rational 统一过程（RUP）或极限编程（XP）这样的方法中去，以帮助改善你的建模工作。像 RUP 这样的过程有助于使建模具有更少的文档驱动（document-driven）的成分，而 XP 则有助于使建模更加明确。

AM 不同于其他的建模方法。像 ICONIX 这样的方法建议「你需要创建明确的模型（用例，健壮图，序列图和类模型）」，而 AM 则推荐「你需要知道如何来创建多种类型的模型，然后在正确的时间使用正确的模型」。AM 和 ICONIX 可以非常好地互动，事实上，有一本关于这个主题的书正在进行之中。

敏捷建模的名称（以下称为 AM）是从何而来的呢？它是作为“脚下的明灯指引你”，还是起到一种协调的作用？两者都是？两者都不是？

由于其最初是以 XP 为基础的，因此开始被称为极限建模（eXtreme Modeling, XM）。我建了一个网站和邮件列表，并让其他人参与到它的定义中来。在这个过程很早的时候就发现其并非局限于仅使用 XP 进行建模的范围。有多个名称被提议，Bob Martin 提出了敏捷建模，从而最终确定这个名称。

AM 反映了敏捷联盟（Agile Alliance，www.agilealliance.org）的价值观和原则，而“敏捷”一词便来源于此。

如何来正确地称呼 AM，“轻量级建模（Modeling Lite）”还是“最简化建模（Minimal[ist] Modeling）”？

敏捷建模指的是恰如其分。我将 AM 形容为“有效建模（effective modeling）”。

UML 是如何来适应 AM 的呢？

可以将 UML 图作为你在进行敏捷建模时所使用的一些技术，但并不意味着只有这些图。UML 并非是全部，请参见 www.agilemodeling.com/essays/realisticUML.htm，因此你需要的并不仅仅是这些 UML 图。例如，每个我所构建的单独的业务应用都有一个位于前端的用户界面和一个位于后台的关系型数据库，然而 UML 当前并没有关于 UI 建模和数据建模的任何内容。当我碰到数据建模的问题时，我已经渐渐厌倦了等待 OMG，并在 www.agiledata.org/essays/umlDataModelingProfile.html 上整理出一个有关数据建模的档案（profile）。这是一个开放的档案——我将会采纳任何感兴趣者的想法，并使之不断演变。我很愿意看到像 Borland 这样的厂商能够对其开始提供支持（暗示而已）。

测试在 AM 中会起着什么样的作用呢？

AM 专注于建模和文档，仅此而已。但是，在 AM 中有关于“易测性考虑（Consider Testability）”的实践。我坚决支持应该在整个生命周期进行测试，请参见: www.ronin-intl.com/publications/floot.html ，并且很可能会使用验证模型（validate models，www.agilemodeling.com/essays/modelReviews.htm）。谈完这些，由于你团队内部以及团队与项目涉众（stakeholder）之间交流的不断增强，通过 AM 能够显著降低保持模型复审（model reviews）的需要。

重构与 AM 具有什么样的关系？

重构是一种编码的技巧，因此它位于 AM 边界之外。我确信有两种重构：代码重构和数据库重构（www.agiledata.org/essays/databaseRefactoring.html）。

你会扩展 AM 与极限编程之间的关系吗？极限编程是不是敏捷建模的一个子集呢？若不是，在敏捷建模和极限编程之间有什么不同的出发点呢？

AM 的实践可以被裁减到 XP 中，以使建模和文档更加明确。它们可以良好地一起工作，请参见 www.agilemodeling.com/essays/agileModelingXP.htm。

你使用什么工具来生成和更新你的敏捷模型呢？

在一开始你可以使用任意的工具来生成模型。AM 建议使用最简单的工具来做事情。往往会是一个白板或一张纸，特别是对于需求工件而言，但是当你在讨论详细设计的时候，则常常会使用像 TogetherCC 这样非常复杂的建模工具。你需要使用正确的工具进行工作。

AM 还建议你丢弃那些临时模型，从而能够照亮你前进的道路。最终你所需要更新的模型会更少。

目前谁在使用 AM 呢？其被接受或采用的增长率是多少呢？

我只能告诉你 AM 正在为世界范围内的多种机构所使用。然而，我不清楚有谁追踪过它的接受率。

除了建模之外，AM 还专注文档工作。AM 是不是推荐在软件结束之前编写最终用户文档（该文档将被用作开发团队规约的一部分）？

AM 并没有明确地推荐这样来做，尽管这绝对是一个好的想法。

在 www.agilemodeling.com/essays/agileDocumentation.html 中列出了我关于有效文档的建议。基本而言，要了解你的听众，与他们一起工作，并且保持你的文档简洁明了。

你是把时间花在“潜心钻研”程序设计上，还是把所有时间都花在咨询和写作上呢？换句话说，你是一个软件实践者，还是一个理论家，或两者都是？

两者都是。我将比较多的时间花在为客户工作上面。前两个月我把大多数时间都用于完成我的新书 Agile Database Techniques（www.ambysoft.com/agileDatabaseTechniques.html）以及 The Object Primer 第三版（www.ambysoft.com/theObjectPrimer.html）。然而，通常我的做法是白天为客户工作，然后晚上写作一两个小时。除 Star Treck 之 Enterprise 号外，我几乎不怎么看电视，因此我有大量的业余时间。

在“人月神话（The Mythical Man Month）”中，Fred Brooks 说过不存在像魔法一样来解决软件挑战的“银弹”？我们欣喜地看到你并未向 AM 的采用者承诺以奇迹，而是说：

“敏捷建模是一种有效的用以改善许多专业人员的软件开发工作的方法。仅止于此。它并非是用来解决你所有开发问题的魔法蛇油。如果你努力工作；如果你聚精会神；如果你心中紧记 AM 的价值、原则和实践；那么将有可能会提高你作为一个开发者的效率。”

(来自 http://www.agilemodeling.com/essays/whatIsAM.htm)

你是不是经常会发现人们对 AM 存在一些不切实际的期望——例如，所有需要他们做的就是参加一个培训，然后那些软件挑战将随之解决？

这很奇怪。我发现“反敏捷（anti-agile）”者认为敏捷学家（agilist）是在宣扬「敏捷方法就是银弹」的言论，因此这些方法常常会被认定是一时兴起而被忽略。事实上就我所知，在敏捷社区内没有人声称他们已有魔法般的解决方案，并且时不时地叫嚷说「就像我这样地使用 AM」。

通过对 AM 价值（http://www.agilemodeling.com/values.htm）的观察，看起来这些品质的发展将适用于一般的生活，而不仅仅是软件开发。你有没有见过这样的例子－学习 AM 能有助于一个人改善其个人的生活？

还没有，尽管这丝毫不会令我感到惊讶。一旦你开始这样去工作，它便很自然地能渗入到你的个人生活中去。看起来敏捷方法正是要让 IT 专业人员对软件开发感到更加满意，从而改善你们整个的精神健康。

关于AM原则（http://www.agilemodeling.com/principles.htm），我便受了“拥抱变化（Embrace Change）”原则的影响。对于每个与我一起工作的开发者而言，变更规约（Changing specs）或边界蔓延（scope creep）一直以来就是一种诅咒。在1998年丹佛的Borcon大会上，Martin Fowler说他永远不会为开发者惊讶他们项目规约发生变化而惊讶。他的意思基本是这样：“规约一直都在变化；去习惯它。”或者，就像你所说的那样：

“项目涉众（project stakeholders）是人，他们是实实在在的人，不管你如何去进行猜测，人们都会改变他们的想法。你曾把客厅里的家具布置成你认为合适的样子，但当你回头看它时你会意识到那并不是你想要的？如果你对于像家具布置这样简单的事情这样做了的话，你的涉众将会改变你为其所构建的那样复杂系统的想法的几率是多少？百分之百。”（http://www.agilemodeling.com/essays/agileRequirements.htm）。

因此 AM 甚至能够超越这种情形，并且说不但要接受变化，而要去拥抱它。为什么？如何去做？

事实上，这是 AM 采纳自 XP 的观点之一。拥抱变化的秘诀是简单地去认识所发生的事情，你不能停下来，而且这的确是一件好事情。Mary Poppendieck 在著名的 Learn Programming 中说到“在生命周期最后变化的需求是一个竞争的优势，只要你能对此有所行动。”这便是拥抱变化。我认为害怕变化的开发者所遵循的方法将无法使其容易地做出反应，因此变化对他们来说非常痛苦。特别是传统的瀑布式项目和 IT 外包（outsourcing）工作。我们知道会发生变化，然而许多机构仍然选择忽视这个基本事实，而去遵循那些无法很好地对变化做出反应的过程。这可能会使官僚主义者高兴，但它不会对你成功的机会有多大帮助。

如果软件开发团队准备实践 AM，那最好是整个团队一起受训（在这种情形下可能所有人都会熟悉 AM，但从中没有专家），还是每个团队具有一位专家，他能够持续地对团队进行指导并监控项目的“AM 性”？

原来这明显是一段广告，<< 露齿而笑 >>。看上去 Ronin 是在兜售 AM 培训（www.ronin-intl.com/training/index.html），我窃以为，但是我更愿意看到对团队整个进行培训。就像其他技能一样。软件开发是复杂的，并且所需解决的问题范围很广。因为「你在编码前进行建模」是一个好的想法，即便是你只生成了一个快速的草图（quick sketch），因此一个好想法是了解一个大范围的建模方法，从而你能够有效地涉及各种复杂性。AM 促进了与其他人一起建模，从而每个人都应该熟悉这些基本的建模元素。说到这里，你仍然需要在团队中有一些富于经验的人，他们能够指导那些不是那么有经验的人，因此你可能想要雇佣一些顾问。非常凑巧，Ronin 也有一些非常优秀的建模顾问（www.ronin-intl.com/services/index.html）。

你说（http://www.agilemodeling.com/essays/agileRequirements.htm）“我坚决相信需求应该独立于技术”。基于此是否意味着在这个阶段（需求采集），建模者甚至不应该去考虑他们的设计方法或编程工具的优缺点？

如果你是在做需求，就请只做需求。然而，经常会看到人们片刻间便从需求转变到分析到设计，一直到编码。例如，如果你的用户说系统必须要维护客户合同信息，许多有经验的开发者便会开始考虑 Party 分析模式，设计一个 Address 类，一个 Address 表，甚至往往有可能立刻编写出验证邮政编码的代码。在这里通常很难把需求、分析和设计活动分离开来，无论理论学家如何宣称。

同时，有些需求在实质上是技术上的，因此技术将发挥作用。

关于软件项目的一个常见问题是过于耗时和效率低下的会议。AM 有什么用来缓解这个问题的实践呢？

不惜一切代价要避免会议。我能给你的关于会议的最佳建议就是去看 Dilbert 卡通片——Scott Adams 对会议具有特别好的理解。

像对其他工件进行迭代这样的实践，能有助于避免分析的停滞（analysis paralysis），由于你不会受困其中，因此可以使你的建模会议更有效率。「带有意图的建模（Model With A Purpose）」原则建议一旦你完成意图便可停止该方面的工作，这有助于缩减你所需满足的时间。「创建简单内容（Create Simple Content）」和「小增量建模（Model In Small Increments）」实践也有助于促进简短的建模会议。

另外一个好的窍门是保持所有的会议为站立式，很快人们便会感到不适，从而使会议保持简短。

另外一个相关但对立的问题是一些团队成员往往愿意呆在自己的“洞穴”中，而不愿与其他团队成员进行任何交互。AM 是如何涉及这一点的呢？

你必须要遵循「与其他人一起建模」这一实践，才能说你是在进行 AM。我想说的是软件开发非常类似于游泳，一个人去做是非常危险的。

你说，“你应该警惕象牙塔架构”

（http://www.agilemodeling.com/essays/agileArchitecture.htm）。假如（我知道，假如永远都是不安全的，但有时我愿意生活在边缘）在杰斐逊和汉密尔顿风格的项目管理之间你更倾向于前者，那么是否便意味着安全呢？

（译注：杰斐逊（1743-1826），美国第三任总统，独立宣言的起草人；汉密尔顿(1757-1804)，美国联邦党领导人。杰斐逊主张分权，汉密尔顿主张中央集权。）

我是一个加拿大人，因此我倾向于迪芬贝克风格的管理。;-)

（译注：迪芬贝克（John Diefenbaker）（1895-1979），曾担任加拿大首相（1957-1963））

然而说正经的，架构设计师必须要准备着卷起袖子参与到项目团队中来。论文（www.agiledata.org/essays/enterpriseArchitecture.html）更加详细地涉及到了这个内容。

你居住在哪里，确切而言？如果你不是这个地方的本地人，你最初是从哪来的呢？

我目前住在安大略省的纽马基特，就位于多伦多的北面。我在安大略省的伯灵顿出生并长大，伯灵顿位于多伦多的西边。

当我在一个长期的项目上工作的时候，我经常租公司的公寓，因此我经常同时住在两个地方。我出差比较多，并且我喜欢旅行。事实上，在这点上我有些让我的朋友们发狂。我出差是为了工作，经常是星期日飞走，星期四的晚上飞回家，并且这样每次会达几个月之久。然后只要我回到家中便会结束旅行进行休假。最近我在假期中完成了生态探险旅行。几年前我去过南极洲并且在 911 之后去了埃及（有点让人害怕）。我下次旅行（即将到来的十二月）是加拉帕哥斯群岛（译注：位于厄瓜多尔西部）和秘鲁的印加古道，2004 年我想去中国的三峡，甚至有可能去爬非洲的乞力马扎罗山（译注：非洲最高山，位于坦桑尼亚东北部靠近肯尼亚边境处。两座被雪覆盖的山峰高达 5,898.7 米。）

我也非常幸运能够进行与工作相关的旅行，目前我已经两次去过澳大利亚和新西兰，并多次去过巴西。这三个国家令人惊异。我出差去过整个美国和欧洲。我希望有时间能出差去冰岛、中国、日本和印度？只要有这样的“异国商务”旅行，我都会尽力做一些观光方面的旅行。因此，任何看到这篇访谈的这些国家的读者，如果你正在寻找一位敏捷顾问，请不要迟疑！;-)

你是如何开始进入软件开发世界的（你是如何被引入的，你在什么时候认识到你要将其作为一个职业）？

我在中学时开始用 Basic 语言在 Commodore PETs 型电脑上编程，也可能是用 FORTRAN 语言。不幸的是，我在学校的辅导导师事实上非常无能，就是说并没有做过什么计算机方面的事情（这是在 80 年代初期），因此在我上大学的时候，我根本就没有预料到将会成为一个程序员。第一年我幸运地选了一门计算机科学的课程，并喜欢上了它，于是第二年我参加了计算机科学的课程学习。为保险起见，我依然同时主修贸易。

你有多少年 IT 行业的经验呢？你现在要走什么样的事业道路呢？

自从 80 年代后期我便进行 IT 的专业工作。当我在学校的时候就在多家公司工作过，为他们编写代码，然后我毕业时就去一家大的银行工作，在那里我见识到官僚和政治的顽固的处事方式。在那期间，我在多伦多大学开始信息科学的研究生课程，并且是计算机科学系的一名兼职助教。我离开银行在业余时间完成了我的研究生学位，并且在做这些的同时又另外在做咨询工作。当我第二次毕业之后我便进入了咨询行业，为客户进行编程工作以及同一个专业培训机构一起工作。后来我加入了一家位于多伦多的 Smalltalk 咨询公司，并在那里工作数年，然后我便出去创立了自己的公司。

你所知的软件语言有哪些？你目前使用哪一个呢？

我现在主要使用 Java 工作，但在过去我用过 Basic，COBOL，FORTRAN，C，C++，Turbo Pascal 和 Smalltalk。我还做过一点诸如 PERL 和 Focus 这样一些语言的工作。

你想不想向当今的年青人推荐程序设计方面的职业呢？

我想把软件开发者这一职业推荐给任何想与其他人密切合作，具有软件才能以及愿意持续学习的人们。

你认为他们需要攻读哪些课程？他们应该准备哪些语言或技术呢？

你需要具有广泛的技能。我坚信人们应该是一个博学的专家，http://www.agilemodeling.com/essays/generalizingSpecialists.htm，其他人将其称为软件工艺者（software craftsmen）。至于程序语言的走向，我依次推荐 java，C#，Visual Basic 和 COBOL。即便 COBOL 看上去不怎么合适，但现实是存在着许多 COBOL 代码，它并没有消失，但维护它的人们正在渐渐离退。

在你所参与的软件项目或产品中，你最为之骄傲的是哪个？

我为许多所参与的项目而骄傲，因此真的很难只挑选一个出来。

你当前工作在哪些项目上？

我想现在恐怕是在 NDA 上面。

你的业务和（或）老板的名称是什么？

Ronin International 公司。

软件开发咨询，指导和培训。

你 web 站点的 URL 是什么？

www.ronin-intl.com。

如果在一个“典型的”工作日里将一个 web 摄像头架在你的肩上，你能向我们描述一下我们将会看到什么？

我坐着和其他人结对编程，或者我在后面与其他一些人坐在一个白板周围。

在你没有相关工作时都作些什么，直接地或不相关地？（爱好？运动？）

我学习武术。我有刚柔流空手道（徒手）的黑带和 Kobudo（器械）的绿带。我还学习太极拳和瑜伽。

我非常着迷于风景和野生摄影以及国际旅行。

你所拥有的与软件开发相关的最有趣的经历是什么？

我真的记不起哪件事情比较突出，尽管我经历过许多“尖发老板（pointy haired boss）”们的决定。

你所拥有的与软件开发相关的最受挫折的经历是什么？

政治，政治，政治。

你对.NET 框架和 C＃程序设计语言怎么看？

我认为它们都很好，尽管我依然更喜欢 Java。

.NET 比 Java 有什么优势？

市场份额，因为其背后是 MS。这便是我所能告诉你的。

Java 比.NET 有什么优势？

多厂商，其出来的时间比较长，其背后有许多伟大的工具。

你收藏了哪些与软件开发相关的 web 站点？

www.sdmagazine.com

www.google.com

你如何来紧跟软件开发的新潮流呢？

阅读，阅读，阅读。参加研讨会。我是多个邮件列表上面的积极分子。

你参加过哪届 Borland 开发者大会呢？

我将在今年秋天的 BorCon 上发表演讲。

在软件开发领域之外，你还有哪些自己的“得意之作”呢？

我的摄影。如果你在 www.sdmagazine.com 上对单词“Antarctica”进行搜索，你会发现一些我拍的照片。

如果你不参与软件开发，你觉得你可以作什么工作？

我将会是一名野生摄影师。

如果你能够在任何时候生活在地球上的任何地方，那将是什么时候和什么地点，以及为什么？

我就住在现在我所处位置的北边，位于 Muskokas 内。加拿大的生活标准是世界最好的之一，Muskokas 令人惊异（乡村农舍），我距离多伦多还有几个小时的路程，多伦多是世界上非常大的都市，并且是北美最安全和干净的城市。

如果在电视节目开始之前免费给你 30 秒钟在全球播放，并且你可以说任何内容，那会是什么呢？

治理环境，在还不算太迟之前。

你最喜欢吃的食品是什么？

寿司。

你最喜欢喝的饮料是什么？

格伦奥德（Glen Ord）纯麦威士忌（Single Malt Scotch）。

你最喜欢的电影是哪部？

银翼杀手（Bladerunner）。

最后，我一直想问的问题是：你更愿意吃什么：驯鹿眼球后面的肥肉，还是霜冻的 poptart（带果酱或果馅的小圆饼）？

驯鹿的肥肉。

对于我没有问到的地方你还有什么补充吗？

请访问我的信息站点——www.agilemodeling.com，www.agiledata.org，www.modelingstyle.info，和www.enterpriseunifiedprocess.info——以及参与相关的邮件列表。在站点上有非常好的素材，在列表上有非常好的谈话内容。

本访谈是通过 email 于 2003 年 5 月进行的。Clay Shannon 是一位 Borland 开发者，著有“Tomes of Delphi: Developer's Guide to Troubleshooting”（Wordware, 2001）一书。

征稿

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

The Addison-Wesley Signature Series

Patterns of Enterprise Application Architecture

中译本即将上市

Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford

UMLChina 负责中译本最后审校，并指定为训练用教材

CASE 工具对用例的支持

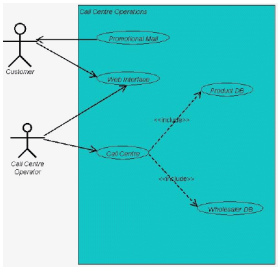
Hüseyin Angay 著， think 摘译

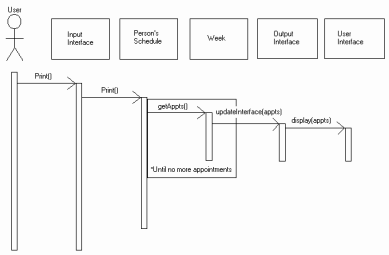
查看评论

引言

用例建模长期以来被看作次于对象建模的二等公民，部分原因是需求分析工作比较单调，没有设计和实现那么有意思；另外的原因就是 CASE 工具支持不够，导致它没有对象建模那么流行。对象建模在现在之所以如此流行，过去几十年支持对象建模的工具持续改进是一个主要的原因。

不论是说“流行”还是说“支持”，用例建模远远落在对象建模后面。为了让用例建模象对象建模那么高效和严格，本文列出了一些所需的工具支持，





用例建模和对象建模的比较

用例建模经历了这样的演变：

- 它在小环境的小项目里成功应用了一些年，而外界并没有察觉该技术的存在。
- 它开始被少数群体接受，认为它能适用于更大范围的应用。
- 它被多数群体接受，当然有些人不是那么情愿，只是他们的经理/客户/同事听说了这样一门好技术，他被迫使用用例。
- 逐渐地，它获得了足够的动力，这时，每个人不得不“做用例”
- 一夜之间，每个人都变成了用例专家

这，就是事实。和对象建模在 80 年代到 90 年代早期的故事十分相似。考虑到用例也是主要用于应用开发，只是比对象建模晚了 5-10 年，我们可以基于已经在对象建模上发生的事情，对它在未来一些年的应用作出一些预言。

用例的当前工具支持

用例建模的工具支持依然不够。我们能够画用例图，把属性和文本说明附加到用例上，把它们组织成包，以及把用例和“系统应…”格式的需求条目链接起来(如果实在走运的话)。

CASE 工具没有进一步去管用例模型，Rose 做的一个让步是：让我们为协作建模，协作是用例的实现；虽然这能使用例建模和编码任务变得容易，但这不是组织用例自身。Rose 和 RequisitePro 的集成提供了更多的方便，但仍然需要手工建立和维护大多数的关系。用例建模的自动化程度之所以没有对象建模强大，原因是用例缺少精确的形式。用例的内容，甚至是整个用例模型，直到现在都还是散漫的。这使工具厂商很泄气，因为这是一笔不划算的投资：制造了工具，却不被大多数建模者使用。内部制作的工具只能内部使用，因为不同企业接受不同的风格，甚至同一企业的不同团队之间也有不同，这意味着这些工具不能用在任意的地方。

UML 最终达成一个用例模型的统一视图，在我为客户提供服务时，扩展和包含之间的区别的争论最终变成不重要，代之以用例的规则。两年前，我们这些研究了许多年的人就用例的内容提出了一种格式，最终出版在 Alistair Cockburn 的书中[Cockburn, 2001]，现在已经成为主流的方法。另外一方面，Larry Constantine 的本质用例（essential use cases）方法[Constantine, 1999]也帮助建立了用例与其他类似于屏幕设计和业务规则工件之间的关系的一个可接受框架。CASE 工具供应商加速建立它们的过程框架，也帮助传播了很多用例建模的实践，这些实践现在已经被接受成为原则。考虑到格式方面的分歧不再那么严重，也许引进一些工具的时机成熟了。

引进工具的另一个理由是可以鼓励建模者吸纳最适合工具的格式，这会给用例建模方法带来推动力。

用例的基本工具支持

那么，我们需要什么样的功能来组织我们的用例模型呢？对象模型的工具带来了很多的方便：

导航。帮助建模者通过有意义的路径跟踪模型元素。 现在，在每种开发环境中，类模型的继承端关联树已经得到了支持。

过滤和搜索。通过只显示相关模型元素来帮助减少模型的复杂性。例如，我们可以只显示类的某些类型的操作，而隐藏其余操作。

分类。基于某些内在属性、根据不同标准来组织模型元素。这和过滤类似，但思路的焦点是把模型元素组织起来，而不是隐藏或显示它们。

验证。分析模型并判断它是否满足了某些规则。例如，我们是否可以映射一个包中的所有类到一个关系模型，或包之间的依赖是否反映了类之间的关联。

同步。保证相关模型元素不会彼此脱离同步。这比验证又进了一步，通过自动保持模型各部分相互一致来避免无效模型。例如，当模型中一个属性的类型改变了，关联的代码也会自动修改。重构可以利用这种方便帮助用户安全地修改他们的代码和模型。

重组。比分类更进一步，它对模型作了改变。例如，为了自动添加数据访问对象，我们可能会改变模型的包结构甚至创建新的类或属性。支持代码重构的工具的出现也是这个原因。

用例建模要想提升以满足我们日益复杂的建模需要，必须提供相似的方便。

导航

CASE 工具不能画出本身有意义的用例。用例文本对于工具来说只是一个简单的文本域，而附带的文档则简单地处理为 BLOB。

需求管理工具和 CASE 工具集成，带来了一些辨别能力，因为它们允许在文档、文本段和模型元素之间建立链接，但链接仍然需要手工引入和手工管理。例如，用户可以建立到另一个模型元素的链接，然后得到一个超链接，帮助在两个元素之间来回移动。这样有两个问题：

(1）需求管理工具通常增强某些可配置的规则，这样一个执行者(actor ）就不会被链接到一条业务规则。但一旦我们允许一个用例可以链接到另一个用例，所有用例都可以链接到其他用例，不需要有效性检查。

(2）在模型框架中管理链接之间的通信，而用例模型必须手工管理：如果我们在模型中移动用例的关联，我们必须在需求管理工具中再来一遍。

有效的导航依赖于是否有以下功能：

- 产生组织好的和可导航的用例列表（例如：一棵树），执行者和其他模型可以通过以下问题导出：“谁和这个用例交互？”，“它会被其他用例包含吗？”，“哪个用例扩展这个用例？”
- 跟随用例文本中的链接到另外的用例、业务规则以及其他相关的模型元素，至少——被包含和扩展的用例和相关的执行者。

过滤和搜索

CASE 工具提供了模型元素的搜索便利，大多数模型元素还是需要通过关键词、位置等来过滤。 搜索不受模型语义的影响。过滤是搜索的延伸，它允许我们搜索模型元素，但也允许我们隐藏我们不愿看到的模型元素。我们过去曾经尝试通过版型化或打包用例来模仿这种功能（实际上，现在我们还在这么做）。在这个过程中，我们确实产生了一些很有趣的模型，虽然这种有趣还停留在纯学院派阶段。我们过去常给用例编号（甚至命名它为"028 - 申请贷款"），这样我们可以产生按照我们的规则排序的报表，而不是按字母顺序（如果我们走运，一些报表工具会使用用例创建时的顺序来排序）。用版型和包来组织用例有助于搜索和给我们一种手工维护信息隐藏的方法，但它也意味着我们要引进不必要的信息到模型中，只是为了说服 CASE 工具让我们用它做有用的事情——更不用说不得不维护这些结构和名称。如果 CASE 工具基于我们已经放进模型的信息来做这个工作，它会更加有用。这样就能避免花费过多的精力来过度组织模型以及给模型引入不必要的信息。 CASE 工具应至少提供以下这些过滤和搜索能力：

（1）列出符合某些条件的用例；例如：

- 和某个执行者有通信 （ communicate ）关系的所有用例
- 有某种优先级的用例（当然，这可以通过链接到象 RequisitePro 这样的需求管理工具来实现）。

（2）为符合条件的用例画图。

分类

过滤和搜索是为了在模型中迅速发现我们要找的东西。分类则更进一步，组织模型元素来产生服务于某个一致目标的子模型，目的是为了能够把子模型当作一个单独的成分来处理，这是出于类似工作分配或工作量估算这样的长远目标。

一些分类的例子：

- 所有形成单个工作流的用例 [Angay, 2003]。
- 所有基于优先级、依赖、复杂性和工作量的考虑，符合下一个迭代要求的用例。

验证

用于验证对象模型的 CASE 工具功能和第三方插件日益增多，验证对象模型变得简单，无论是象映射到关系数据库那么简单还是象验证设计那么复杂（例如：是否所有单向消息都有相应的回调？我们是否依赖于多继承？）

创建不一致用例的可能性大得多。很多类模型用来产生代码，这时编译器会检查和测试它，而用例只是可视化地复审。更大的模型，可视化地验证变得非常困难。

我认为以下的验证是必需的：

- 验证模型的语义；用例文本和相关联模型元素（包括关联和依赖）应该一致；应该避免糟糕的结构（例如循环依赖）。
- 为用例模型应用启发式方法来增强好的格式和可用性——在复杂性度量等的帮助下。
- 允许引进新的启发式方法来满足建模者的特别需要。
- 提供度量来显示复杂性，等等。
- 为建模者提供上下文敏感的帮助和指南以保持模型一致、正确和可用。

同步

独立维护几种相关模型元素的问题之一就是不一致很快会在模型中蔓延，验证有助于指出这些不一致，但最好的办法是一开始就不要产生这些不一致。例如，当我们改变一幅图中的关联时，所有的 CASE 工具都足够聪明来更新该关联在所有图中的显示（主要因为该图仅包含一种关联的表示方法，作为单一的实体存储在库中。）一些 CASE 工具也保持类模型与代码同步。用例方面又如何呢？

我希望 CASE 工具足够聪明，能够自动管理以下依赖：

- 用例文本与：
  - 关联
  - 活动图
    - 活动的名字
    - 活动的顺序
  - 顺序图和协作图
    - 链接用例文本到消息的运行时注释
    - 主要对象的名字
- 业务用例和系统用例
- 用例和相关业务规则与需求

重组

这是终极目标。重组比分类和同步更进一步，管理模型的基本变化。实际上有两方面：

（1）通过模型的现有部分，加上一些假设，产生有意义的内容。能一致地从模型的一部产生价值不高的内容的映射不是我们的目的。如果是这样，衍生的部分可以丢掉，因为它们随时都可以再生成。目标是减少用于简单体力劳动的工作量。一个例子是从现有的业务用例来创建系统用例；另一个例子是同步两个被这样独立产生的模型。

（2）重构，即管理一个或多个相关变化引起的级联变化。

例子：一个用例关联到其他一些用例，当这个用例改变时 CASE 工具能够把建模者带到每个包含它的用例以便他们可以验证影响有多大并作必要的变更。同样的情况适用于所有受变更所影响的模型元素。工具将考虑一些会引起级联变化的主要变更：我们修改了一个被包含的用例的行为，就会影响其他一些用例的行为，这些用例又会引起其他用例的变化。如果变更引起麻烦，可能就需要部分回滚或重头再来。目前提交方面可以通过配置管理系统（CMS）手工管理，但 CMS 和用户的水平有关，保存点和基线依赖于建模者的判断。建模者可能会发现需要做一个重要变更，这个变更在一开始可能觉得不重要，因为在一开始变更引起的影响强度不那么明显，所以在这之前建模者已经做了很多小的变更。在这种情况下，回滚到以前的版本会导致所有其他不相关的变更丢失，导致

所有其他不相关的变更丢失，导致工作量增加。因此 CASE 工具应该允许上下文敏感的回滚（至少是多层次的 undo 功能），独立于 CMS。

模拟

我在有某些保留的情况下加入这一节。模拟用例的行为可以发生在多个层面：步进执行文本，步进执行相关顺序图或更进一步——类图来显示哪些属性被改变了等等。我怀疑不是每个人都会同意这样做有用。

模拟顺序图方面已经有工具实现了。Select Enterprise 就是一个例子，虽然我不敢肯定最后的折衷（例如倒转«extend»的方向）是否值得。我也觉得活动图也会很快在一些 CASE 工具中得到模拟。但这些功能目前还只是一些小伎俩，即：实现起来风险和工作量都不大，却能在产品的特性列表上多加一个勾。我目前还不相信它们有用。

一个更有建设性的尝试是把文本和与用例链单个流相关联的模型元素结合起来，形成单个实体。这在展示一个用例集合的行为时更有用。从头到尾模拟同一条链，在分支点附上相应问题，是一个很有用的扩展。但如果能够象一颗树一样展示链的文本，我已经觉得很舒服了。关于这方面，欢迎各位提供好的建议。

基本功能

上面列出的功能给 CASE 工具增加了更多的需求，主要在用例模型的语义和内容方面。CASE 工具理解用例模型的某些含义，主要在图形级别上。我们需要知道简单关联和包结构，因为用例在这方面复杂性较小。如果用例文本和其他模型元素的语义没有确定，就需要更复杂的建模功能。前面部分内容中我列出的复杂功能可以通过插件引入，但以下这些功能应该是 CASE 工具的基本功能，就象 CASE 工具应该了解类模型的语义一样。

（1）用户应该可以把用例文本联系到其他模型元素——例如，高亮一个文本引用到一个被包含用例，并从列表中选择一个用例。CASE 工具可以在选择和排序用例、执行者等元素时起到帮助作用。更好的功能是，文本元素可以自动被识别。一些用例文本会引用到其他模型元素，例如，执行者参与某个用例。这些都是识别的路线，不需要手工去链接这些元素。这个功能可以通过建模者约定惯例来实现，例如用例名用花括号括起来。

以下是用例体内一些应该可以链接到其他模型元素的文本引用：

- 其他用例
- 执行者

- 类、属性和操作
- 活动

（2）其他应该被追踪的东西，包括：

- 顺序图中的用例行/段以及注解（这样用例流就可以总是链接到顺序图上的流）
- 用例的后置条件应该自动链接到其他用例的前置条件。这样链接起来的用例之间应该有自动依赖存在。

（3）自动给用例文本行编号。行号不止是为了使用例文本更加整齐，而且在构造备选流和引用用例其他部分的时候很有用。不幸的是，我们不能使用文字处理器提供的自动编号功能：当我们移动行时，行号就会改变，和该行相关的元素就会失去同步。CASE 工具应该能够自动管理用例的行号，并在行号改变时更新所有的引用，包括用例内和其他用例的引用——例如，我们在写特化用例时只需简单提及基用例中的行号即可。

总结

如果用例工具能理解用例模型的语义和管理这些语义，就可以使用例建模更加严格、和需求分析及开发流程联系更加紧密。上面所列出的功能不算详尽，而且也未必是每个人都需要的，但是一旦它们被实现，我相信大多数建模者会把它们当成工具本身很自然的一部分。我曾经在很多情况下定制 Rational Rose 和 Select Enterprise，满足了以上的部分需求

参考文献

Angay (2003) Angay, Hüseyin (2003). Use Case Chains. http://www.aptprocess.com

Cockburn (2001) Cockburn, Alistair (2001). Writing Effective Use Cases. Addison Wesley

Constantine (2003) Constantine, Larry L.; Lockwood, Lucy A.D. (1999). Software For Use, A Practical Guide to the Models and Methods of Usage-Centered Design. Addison Wesley

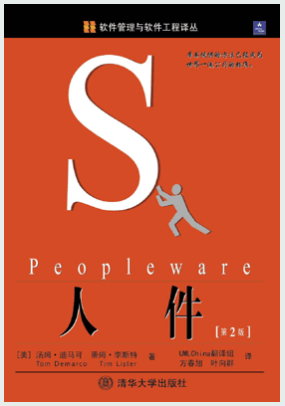
PH PTR

人件集

——人性化的软件开发

The Peopleware Papers

NOTES ON THE HUMAN SIDE OF SOFTWARE



续篇

Larry L. Constantine 著

谢 超 刘 颖 谢卓凡 李 虎 译

人民邮电出版社

极限建模与可执行模型

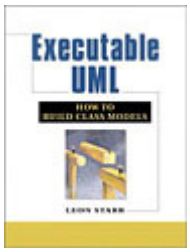
Stephen J. Mellor 著，Ryan Chen 译

查看评论

在本文中，我们把极限编程（eXtreme Programming，XP）和敏捷联盟（Agile Alliance）里的一些合适的原则拿来，应用在建模领域。

极限编程和敏捷过程在关键任务里的一般焦点，是我们去建造最终产品：代码。这一点非常有争议性：一些人喜欢这种主张，而另一些人把它看成是hacking的托词。但是在可执行模型（executable model）的概念中，最终产品是描述系统的模型，是模型定义了系统的设计，而不是代码。

本文覆盖了敏捷联盟和 XP 的 12 条原则，并展示了它们是怎样适用于系统建模的，尤其是高可靠性（high-assurance）的系统。



1. 上下文

新千年开始的时候，Kent Beck写了《*eXtreme Programming eXplained*》（译注：中译本《极限编程解析》已由人民邮电出版社出版）[1]，用了全部适当的大写来吸引注意。这本书和书里包含的实践，已经并将继续给这个产业重大的影响，但是这个书也引起了一些非议，甚至愤怒。原因似乎包含了下面这些：焦点放在写程序，而不是“分析”和“设计”，或者说“建模”；对文档的蔑视；共产主义者的想法，认为一周只工作40个小时。

XP并不仅仅是个“轻量级”的方法：也包含了Cockburn的水晶家族（Cockburn’s Crystal Family）（现在合并到Highsmith的适应性系统开发（Highsmith’s Adaptive Systems Development）），SCRUM，和DSDM。2000年晚些时候，Object Mentor的Robert Martin建议各个轻型方法的支持者们聚会并成立一个社团，来促进我们共同的思想。这个建议在2001年2月份的时候得到了实现，当时16个轻型方法的倡导者和我一起聚会，并成立了敏捷联盟（Agile Alliance）。（为什么用Agile，“extreme”考虑过的，噢，极端的，“lightweight”也考虑了，

遗憾的是，听起来象“不能胜任的”。）。在会议中发布了一个宣言，并公布了一组原则。在本文中，我们将更多的讨论敏捷联盟，而非XP，因为敏捷联盟继承并包含了XP早先的工作，但是我们也将完整地检查XP所特有的思想。

会议一开始，我介绍自己为“间谍”，因为我的兴趣是在可执行模型：模型可以非常精确和详细，足以去执行。这样，那种认为代码是唯一感兴趣的产品、模型是多余的观点就会很奇怪了。当然，我们仍旧将把我们的注意力放在最终产品上，但是这个最终产品完全可以是一个可执行模型。

可执行模型有两种类型。第一种就如iLogix的Rhapsody所做的，是在模型和代码之间使用近似一对一的对应。第二种方法是使用一套规则，来定义同一个精确的可执行模型到多种可能性的目标结果的不同的映射规则：比如一组规则是映射到单任务C++（single-tasking C++,）的，另一组是映射到多任务C++（multi-tasking C++）的，还有嵌入式C，甚至可以映射到VHDL（Very-High-Speed Integrated Circuit Hardware Description Language）。最近的一个例子是Project Technology的BridgePoint®工具集，它提供了建模，然后根据适当的规则把模型进行映射。（更多关于可执行模型的不同方面，以及支持它们的工具的讨论，参见Bell〔2〕，更多关于可执行UML（*Executable UML*），请参见Mellor〔3〕，还有在Starr〔4〕里提供了一个完整的例子）。

无论怎样，可执行模型都是一种产品，是我们把工作转换为比特之前这个阶段努力的成果。在这种意义上，模型就是代码。在一对一的转换中，模型用精确的图形表现了软件结构。这就要求，在模型编译过程中不能有其他输入到模型编译器中，模型编译器只能根据模型内部输出结果。这时，模型只是一个绘图程序。而另外一种，映射方法，模型本身描述了问题（问题域）模型中的潜在的含义，而规则描述了模型编译器适用的软件结构。

极限编程和敏捷联盟的大部分概念不是仅仅针对编程的，经过抽象之后，它们也可以适用于建模。本文的目的在于描述所有的这些概念，看看在一般情况下它们怎样适用于建模，并加以解释。希望能够帮助人们去决定是使用这些概念还是拒绝这些概念，在你们的结论的描述上增加一些洞察力。

2. 敏捷宣言

敏捷联盟有一个Web站点。参见〔5〕。

敏捷宣言是：“我们通过亲身实践以及帮助他人实践，找到了更好的软件开发方法”。在宣言中有四个“价值声明”：“我们认为：<左边的部分>比<右边的部分>更有价值。”，这里关键点是我们并不认为<右边的部分>是错误的，只是说<左边的部分>应该得到更多的强调。

（译注：“左边”、“右边”即

个体和交互 胜过 过程和工具

可以工作的软件 胜过 面面俱到的文档

客户合作 胜过 合同谈判

响应变化 胜过 遵循计划）

让我们依次看看这四个价值声明，当我们看到下面派生出来的12条原则，我们将领会它们是怎样在建模期间层叠起来的。

“个人和交互”胜过“过程和工具”：过程和工具不是银弹。你可以不靠过程和工具来创建一个系统，但是你不可能不依靠人。而且人优秀一点，那么结果就会好一些。到目前为止，这里还没有说过过程和工具是不重要的。毕竟，Project Technology在卖工具，而且任何一个敏捷过程也是一个过程。我们只是想要说人和交互需要更多地被重视。

“可以工作的软件”胜过“面面俱到的文档”更有价值：如果你打算从家里开车到附近的医院，你肯定不会先写一个文档来严密地描述如何去做。当然，你需要一张地图来找到一条最近最快的路，或许你还会听听收音机来收听实时新闻，看看哪条路堵了。为以后而“文档化”这条路是没有用的，不可能止住你正在流的血。

逐字逐句，这个例子是无懈可击的。那么为什么所有人都会喜欢文档而不是关心软件本身呢？这里有个潜在的意思：文档是不重要的吗？是不必要的吗？是一个障碍吗？然而，通过一个可执行模型，模型就是“文档”就是代码。在这种意义上，这个价值声明可以被读作“我们认为X比X更有价值”。

为了解释表面上的矛盾，我们必须区分下面两种类型的文档：

一种是那些帮助我们理解的，例如用例、伪装成模型的草图，是为了满足一些约定和一些内定的要求而交付的。另外一种就是正式的可执行模型，其实它就是软件。

帮助我们理解的东西是设计用来抛弃的。正式的可执行模型就是软件。我们构造了伪装成模型的草图，并小心存放起来，然后看着它们被真实的代码替换，而真实的代码又会出现问题。

“客户协作”胜过“合同谈判”：我们都听说过，那些规格说明已经被冻结的项目，它们的结果往往是不幸的。很明显，我们必须讨论“客户”，但是它并不象看起来的这么简单。手机的“客户”就是手机的使用者，我们当然可以和他们交谈，但是他们的需求，受到“他们知道什么，他们已经在用什么”这样强烈的制约。他们想要的也很大程度上受到当前我们掌握的技术的制约，所以我们也要同业内的技术专家交流。通常，我们客户很多，而且来自各行各业，所以我们没有一份合同，只有一个最后期限。虽然如此，我们必须认识到一份合同就是一个框架，即使是不正式的，我们应该期望同我们的客户一起合作来确保我们做的是一个正确的产品，无论他们是谁。

另一个管理变化中的需求技巧，是把不会变化的部分模型化，然后在数据里声明变化的部分。参见“Coping with Changing Requirements”〔6〕

“响应变化”胜过“遵循计划”：一个计划告诉我们如何得到我们在哪里我们正在去哪里 ，但并没有告诉我们即将去的地方是否是正确的地方。即使今天的计划是正确的，也并不意味着明天它还是正确的。技术和市场变化的比计划快的多。达到计划的里程碑和客户成功之间，并没有必然的联系。

这个声明，依赖于预言性方法和适应性方法的差别。在建筑行业，执行计划这个过程，也就是建筑阶段，比计划编制阶段，以及设计阶段花费的时间多很多，我们通过计划来预知项目什么时候可以完成。在软件行业，设计阶段和计划编制阶段花费的时间比实现阶段花的时间多很多。一些人把实现定义为程序的编辑，但是即使我们把实现定义为编码动作，它仍旧是整个项目的总时间中的一个小部分。因此，计划没有了预报的价值。相反，我们必须认识到：我们必须根据环境的变化，来改变我们作的事情。关于这部分更详细的讨论，请参见Fowler[7]。

这并不是诋毁计划。没有一个计划，我们对目标将没有一个认识，甚至走弯路。这里的关键就是适应变化来保证我们的产品尽可能接近于用户所需要的样子。事实提供真实的反馈。

3. 敏捷原则

在这个章节，我们来具体检查12条原则。每一条都被编号和命名，以方便参考。（这些名字是我自己从XP那里偷来的。）

#1 交付价值：“我们最优先要做的是通过尽早的、持续的交付有价值的软件来使客户满意。”。就象以前芝加哥的很早投票和经常投票的投票者，这条原则要求及早并持续集成和交付。关于草图式建模（modeling-as-sketches），最明显的批评就是只能进行同行评审。就象Bertrand Meyer在〔8〕里所说的，模型不会告诉你它是否是正确的，但是代码会。

可执行模型避免了这个问题，因为它们可以根据真实的输入得到真实的输出。这意味着我们得到了来自真实的正在运行的系统的反馈。

#2 利用变化：“即使到了开发的后期，也欢迎改变需求。敏捷过程利用变化来为客户创造竞争优势。”反馈暗示着变化。我们不可能抵挡住变化，只能利用它们。

合作胜过“合同谈判”的要点是，敏捷过程需要由用户来决定需要哪些功能，以及按照怎样的顺序构造它们。比如，SCRUM，提供了两组卡片来记录功能。一组卡片提供了整个系统的有顺序的功能集，另一组卡片则记录了在当前30天冲刺要被执行的那些功能。作为必须的，客户和开发者争相去重新排列这些卡片，也就是这些功能的相对优先级。

#3 经常发布：“经常性地交付可以工作的软件，交付的间隔可以从几个星期到几个月，交付的时间间隔越短越好。” 结构化分析的先驱者之一Steve Mcmenamin曾经说过，一个项目如果在18个月之内无法发布，那个这个项目已经死了。现在，甚至18个星期看起来也象一个很长的时间。敏捷过程建议尽可能把你的迭代过程做得短，尽管每次迭代的时间不是完全的相同。

“交付（delivery）” 并不总是意味着“发布（release）”。“交付”是内部的，面向团队的，而“发布”是外部的，面向客户的。一个客户没有必要每三个星期就得到一个新的产品，但是我们自己的团队应该可以尽可能经常使用集成的系统，因为这样可以增强我们自己对产品的信心。

可执行模型就是可运转的软件。

#4 现场客户：“在整个项目开发期间，业务人员和开发人员必须天天都工作在一起。”。这个原则的目的是利用变化。（参见#2原则）。在一个实时的和封闭的环境中一起工作，意味着开发者和硬件工程师，操作人员，甚至是市场部门，一起工作，从而保证我们的确是在构造一个正确的系统。这个原则也有助于建立相互之间的信任。

一起工作是重要的。相比其他方式，它让非软件专业的知识成为工作的一个部分。

#5 信任有活力的团队：“围绕有活力的个体来构建项目。给他们提供所需的环境和支持，并且信任他们能够完成工作。”构造一个产品时，有两样东西必须属于同一组人，一是责任；二是权力，决定“怎样做才最好”的权力。因为我们知道人的影响因素是第一位的，我们必须找到最好和最有活力的人，并让他们做他们的工作。信任他们。

这暗示着团队管理者的工作就是让项目的工作轻松一点，去除障碍，并不让自己成为障碍。当然，管理者们需要知道这个项目符合日程表，并符合需求，团队中的人们也需要知道。

#6 面对面：“在团队内部传递信息，最有效和最有效率的方法是面对面交谈”。通讯要用全部的带宽进行。虽然e-mail是一种伟大的技术，但是它压制了其他的交流方式。在e-mail中没有“身体语言”，也没有声音的音调。在这样的情况下，反馈无法很直接，也很难注意到细微差别。

把一些交流保存下来也是有价值的，所以任何的项目都必须问：文档化记录的交流和单纯的交谈的应该保持怎样的比例，才是有利于理解的，包括提供给不在项目中的人们？什么才是正确的文档？正确的文档中应该包含什么？

在AA中有一种对文档的蔑视，认为文档是为他们自己的好处才做的，所以我们也必须问问我们自己，什么是一个文档的真正目的，当它具有永久的价值（相对于临时的价值）时，才制作它。

#7 可工作的软件：“可工作的软件是进度的首要衡量标准”可执行模型的一个重大的优势在于可以尽早的让系统运行，这能帮助你早点知道你是否有麻烦。因为我们现在可以通 过运行来验证模型。我们可以得到真实的结果，来清楚地预示软件所能做到的。

翻译（translative）方法比行为（behavioral）方法更有价值，这点特别的真实。因为翻译方法把软件架构从应用中分割了出来，使得可以更早更快地来构造一个可验证的的模型。

这条原则经常被程序员和管理者曲解。这条原则不是允许去hack，敏捷过程强调良好的设计和持续的实施。这条原则强调了真正的产品的重要性，并强调了只生产必要的东西。

#8 可持续开发。“敏捷过程提倡可持续的开发速度。赞助人（sponsors）、开发者和用户应该能够保持一个长期的、恒定的开发速度。”这条原则反对依赖英雄，但是并不排除冲刺。我们都有这样的经历，有时候要排除前一天晚上偶然发昏埋下的错误。

这条原则是管理方面和技术方面的原则，同时也是生活的原则（我们总是希望拥有工作之外的生活），告诫大家不要耗尽精力。

你应该问问自己什么样的组织是你希望的。如果成功的唯一道路是成为一个英雄，而且不管这是否是一个有效的策略，那么现在是时候去寻找新的工作了。

你应该问问自己你希望从工作中得到什么。我们中的一些人想把自己全身心投入到工作中去，另外一些人不是这样想。两种想法都不是“正确的”，但是我们所有人都应该警惕，不要让其他人的期望妨碍了自己的判断。

#9 技术优秀： “不断地关注优秀的技能和好的设计会增强敏捷能力。”这条原则真的是一句断言。良好的设计真的可以提高我们的能力来应付变化吗？我们是否应该时刻警惕着来保证我们做的是良好的设计？如此重新陈述，明白无疑的答案必须是响亮的“是”。那么什么是良好设计的特征？这条原则请求开发组织和个人不要进入“快速却杂乱”的状态？

模型可以被设计得很好，也可以被弄得很差。不幸的是，我们已经倾向于在语法上理解模型：如果它看起来可以工作，图形和连线不太松散，那么就是好的。Leon Starr在他第一本书中[9]，有力并且风趣地谴责了“模型编码（model hacking）”。良好的分析提高了敏捷。

相关的一个概念是重构（refactoring）。它建议，一旦一部分代码已经很清楚地变得不再内聚，你就应该重新组织这部分代码。这样的概念同样适用于建模。就我个人而言，我曾经忍受很多分析会议，在讨论中，建模者和专家们宁愿延迟去定义什么是铣床，解释说，这是因为如果要找出每种完全不同的铣床，需要作太多的工作。“噢，我们会再细分它的”。继承和划分子类是（很多中方法中）的一种方法，来完成一个重构过的模型，但是首先你要已经把模型做好。当你要编码的时候重构你的模型。（参见 Kent 和 Martin 的《Refactoring》〔10〕）。

#10 简单是根本的：“简单——使未完成的工作最大化的艺术——是根本的”前面的原则都是关于“优雅”的，这会有一个副作用，把“简单”仅仅当成一种最简单主义。毕竟，如果代码是优雅的而且易于修改，我们就不需要去作这些(不必要的)事情。这个概念是一个在XP里用一个缩写YAGNI描述，“你将不需要它(You Ain’t Gonna Need It.）”。

在建模中，这个思想当然也是适用的，只做必须做的事情，并优雅地做。

#11 自组织的团队：“最好的架构，需求和设计出自自组织的团队。”让一个值得信任的专业人士组成的团队，和客户或者领域专家，在同一地点一起工作，来构造良好的设计，那么他们当然可以得到良好的需求，架构和设计。(如果这样都得不到，还能有什么方式可以得到？)。有一种期望，如果有很多的交互和一些处理规则，那么“正确的”架构，需求和设计就会魔术般的“出来”了。

在构造团队时，我建议要基于你可用的技能，而不是试图去强迫你可用的“资源”(这是管理上对你的同事的称呼)去作你需要的事情。如果他们喜欢他们想作的事（还有，他们希望去学习的事情），人们通常非常有动力，而且他们知道如何去作。一旦一个团队领会到他们具有什么技能，他们就可以如何去弥补他们的缺陷。

#12：反省和调整：“团队要定期反省如何才能做得更有效，然后相应调整他们的行为。”Cockburn建议给每个项目使用不同的方法，在你还没有开始项目的任何工作之前，当然不可能非常敏捷地来定义你的方法，这意味着在某种意义上你必须“你要在你前进的过程中调整你的方法。”如果你盲目地一直用同样的方式做事情，而且它并不能做得好，你将 注定失败。

Fowler[7]引用了Kerth下面的问题，周期性的问自己，来作为“过程检查（process check）”：

哪样东西我们做好了？

我们学到了什么？

哪些东西我们可以做得更好？

什么东西困扰着我们？

因此，这条原则是一个警告，来监视你在作什么，并定期改造你的方法。“过程检查”是 一个方式，来问问我们自己，我们是否在正确的时间做正确的事情。

4. XP

下面是一个表格，包含了XP的12个要素，以及它们是怎样和敏捷原则关联的。

XP概念（XP Concept）	敏捷原则（Agile Principle）
1. 计划游戏	#4 现场客户(一半)
2. 短发行周期	#3 经常发布
3. 隐喻	
4. 简单设计	#10：简单是根本的
5. 测试	
6. 重构	#9 技术优秀
7. 结对编程	#5 信任有活力的团队 #6 面对面 #11 自组织团队
8. 集体所有制	#5 信任有活力的团队 #6 面对面 #11 自组织团队
9. 持续集成	#7 可工作的软件
10. 每周工作40小时	#8 可持续开发
11. 现场客户	#4 一起工作（一半）
12. 编码标准	

先看看右边的列，我们发现敏捷原则中的“交付价值”（#1）和“利用变化”（#2）没有关联到任何XP的原则，因为它们非常明确的作为动机被描述在XP中。“反省和调整（#12）也因为同样的原因没有被明确包含。

有一些XP原则（XP#3：隐喻，XP#5：测试，XP#12：编码标准）没有在敏捷原则中找到对应的原则，还有一些，比如#7：结对编程，#8：集体所有制，则对应了多个原则。我们现在讨论那些XP概念中与上述12条敏捷原则相关的部分。

XP#3：隐喻：一个系统的架构能够用隐喻来描述。系统的形状由在客户和开发者之间共享的隐喻来决定。这能够适用应用级别或者说软件架构级别。

我们在实时系统和嵌入式系统中，可以看到三个主要的软件架构：监视和控制（Monitor and Control），传输机（Transporters），事务（Transactions）。更多的细节可以参考〔11〕。监视控制系统就是那些看起来象控制器的，PID控制回路等等。例子是电传飞行操作飞行器，铝、铁的旋转炉，居家温度控制，汽车控制系统，等等自动制动系统。

传输系统就象一个电话系统，流数据毫无变化的通过它。 另外的例子是遥感测控，以及一有点令人惊讶的一离线信用卡处理，流数据通过系统，分别通向客户和商业银行，并定期总计金额。

事务系统就象银行：它们维护一些抽象世界的图像，并且接受查询或者更新操作。很多工程系统，比如仿真系统就是遵循这个模式，制造业的系统也是这样。

通过隐喻可以提供一个快速的相似物，通过说明这个系统象什么，来描述一个系统，这比直接去读详细的文档更有效。另外需要指出，一个隐喻也可以被写下来，本身成为“文档”。

XP#5：测试：先设计和构造测试用例。这里的一个目标是保证高速度，易于构造和继承。如果测试用例都是马上可用的，而且不是在写代码之前，才能达到这个目标。另外一个目标是强迫去复查我们所做的东西是否符合需求。当我们写测试用例时，我们会更多考虑去符合需求，而不是代码。这样我们就可以得到更好的代码，因为我们已经考虑过了所有会走向错误的途径。

这条原则也适用于构造可执行模型。可执行模型能够被校验者解释，或者被一个简单的 模型编译器编译。当我们完成分析，并开始建模，我们也必须定义一个可以测试的预先存在的实例，并定义场景（可能与用例的实例相反）。编码的时候，工作就趋向于专心确保我们得到的是正确的。

XP#7:结对编程。成对的编程。这个概念就是指一个人编码时，另外一个人在反思以保证得到正确的代码。比起两个人各自单独编码，这条原则的确可以得到更高的生产力，因为失误以及错误的思路马上就被发现了，减少了浪费的时间。

3－4个人的分析和建模团队是最有效率的。团队里应该有下列的一些角色：

收集者：从领域专家、文档、背景源码（图书馆或者网络）收集信息。

分析者：吸收这些信息并建立抽象。

建模者：真正建造模型，登记模型，审核模型 等等。

组织者：持续跟踪所有信息。

无疑单独一个人也有能力担任所有的这些角色，但是关键是每个角色的要求是非常不同的。我们暂时回到结对开发，编码的人注意细节，这时另外一个人，他的手并不在键盘上，但是在发明，创造，和检查整个抽象产物，这是完全不同的行为。类似的，分析者在创建抽象的时候，用的是右半脑，而这时建模者在注意细节。参见《建模用的是右半脑》（*Modeling on the Right Side of the Brain*），Starr [12]

XP#8：集体所有。代码属于我们所有人。当我们在读一些不是我们自己写的代码，而且我们找到了一个问题，把“别人的东西”改正确是可怕的。集体所有为了拒绝这个问题，通过鼓励知识传播，以及不鼓励私下里的实践。

同样的思想可以不用修改地应用于建模。想这么做，要保证把所有的信息和模型放到一个仓库（知识库）中。这样节省了庞大的寻找信息的时间，减少了信息传递路径的数量。

XP#12：编码标准。哦，是的，建模也是如此。注意这和#11，集体所有，是相对应的。

5. 值得推荐的几点

现在，你也许已经有了你自己的一些想法， 这里有几点我觉得值得强调一下。

刚好足够。XP和敏捷过程是一个受欢迎的让人振作的东西， 但是象很多对习惯位置的反应，有可能会做的太过分。一些特定类型的高可靠系统可能支持这些新的过程，但是我们被强迫 去作其他的一些事情。在这种情况下，我们必须扪心自问，什么是“刚好足够的”过程，才能尽快得到可被描述的需求。

团队。结对编程是“团队”的一个变种。组织经常说它们支持协同工作，而且有文章支持这个思想。（参见〔13〕可以找到一个受欢迎的矫正方法），但是令人惊讶的是，一个小小的 成就就可以让一个成功的团队保持在一起。3个人一起进行一个分析/建模任务，并没有减少 2/3的生产力，而是增强了生产力和质量。而质量在高可靠系统中是特别重要的。

适应性。敏捷过程强调适应性。这里考虑到三个主要的方面。市场，今天它可能会要求一些功能，它们不同于那些你已经做到的功能。技术，它改变（缩短）了作新的事情可能需要花费的时间，但是也同样设置了一些限制，可能直到我们通过项目才会发现这些限制。还有过程本身。把“过程检查”当作你的颂歌，来保证你正在尽可能用最好的方式做需要做的事情。

文档和文档化。我认为敏捷的拥护者们可能对下面几个概念感到困惑，草图（sketches），伪装的模型（masquerading as models），真正的可执行模型（real executable models）。 我也认为很多文档是简单重述的，言语乏力，很不严密，说“真东西”才是更好的。

对于高可靠系统，我主要关心它们的生命周期，也就是说必须有一些方法来把信息传输 到未来，一个口口相传的传统是，我们大家围绕着火堆坐在一起，叙述着我们这些建模的日子的光荣， 这个传统不要丢掉。模型也不是足够的。我们要着重记述并详细描述隐喻。我们 不应该把模型文档化，而是要把我们为什么选择一个细节的抽象写成文档。参见Starr〔9〕

衍生的需求。敏捷过程依赖客户交互来告诉我们下一步要被实现的功能（故事，用例，无论我们叫它们什么）。在很多实时和嵌入式系统中，用户可见的需求是很少的。当有一个需要去构造基础部分时，基础结构中的需求不会来自客户。它们是衍生的。在客户不在（无法起作用）的时候，我们需要一些方法来区分这些需求。

6. 参考文献

引用了许多的论文，书籍和类，下面列出了这些参考文献，以及一些其他可以帮助你整理思路的相关文章。

[1] Kent Beck, *eXtreme Programming explained,* Addison-Wesley Publications Co; ISBN: 0201616416,

[2] Rodney Bell, *Code Generation from Object Models*, Embedded Systems Programming, March 1998. Also available on the web at www.embedded.com

[3] Stephen, J. Mellor, *Executable UML*, Embedded Systems Conference, July 2001

[4] Leon Starr, *Executable UML*, Model Integration, LLC.; ISBN: 0970804407 2 CDs included

[5] www.agilealliance.org

[6] Stephen J. Mellor, *Coping with Changing Requirements*, Embedded Systems Conference, July 2001

[7] Martin Fowler, *The New Methodology*, www.martinfowler.com/articles/newMethodology.html

[8] Bertrand Meyer, *Object-Oriented Software Construction* 2nd edition, Prentice Hall; ISBN: 0136291554 ;

[9] Leon Starr, *How to Build Shlaer-Mellor Models*, Prentice Hall

[10] Fowler and Beck, *Refactoring*, Addison-Wesley Pub Co; ISBN: 0201485672

[11] Stephen, J. Mellor, *System Design, Architectures and Archetypes*, Embedded Systems Conference, July 2001

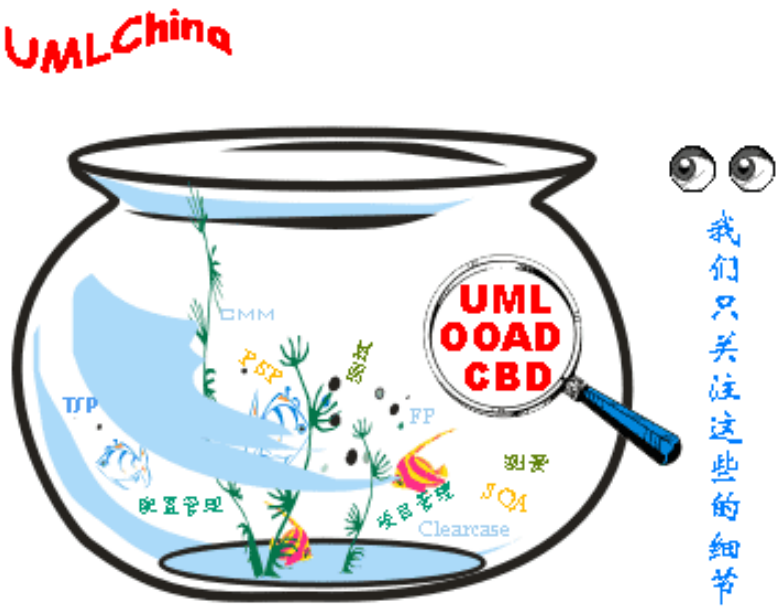
[12] Leon Starr, *Modeling on the Right Side of The Brain*, Shlaer-Mellor User Group, May 2001, www.projtech.com

[13] www.demotivation.com For a brief description of XP, see:

[14] Kent Beck, *Embracing Change with Extreme Programming*, IEEE Computer October 1999 For a discussion of the extreme ideas by Mellor, Bollinger and Humphrey, among others, see:

[15] www.computer.org/seweb/dynabook

Stephen J Mellor是Project Technology公司的创始人。Project Technology公司提供实时系统系统和嵌入式系统的构造工具。Steve是Shlaer-Mellor方法的作者，这个方法是这些工具的理论基础。他正在为OMG工作，扩展UML使之成为完全可执行并支持模型驱动架构。他是IEEE软件产业咨询委员会的成员。



感谢您

的支持与信任

祝福您

快乐、 健康

WWW.UMLCHINA.COM

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

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

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

适应性的实时分布嵌入式中间件的新兴模式

Joseph P. Loyall 著， Li Haiming 译

查看评论

摘要

这篇文章提出描述适合于 QoS 适应性应用问题的解决办法的两种模式：一是服务质量(QoS)合同模式适合于应付适应性的决策和折衷，一是快照模式用于获得对系统当前状态的有用估计。这些模式出现在实时分布嵌入式(DRE)应用的实现中。实时分布嵌入式(DRE)应用通常需要是 QoS 适应性的，因为它们有着严格的 QoS 需求，而且它们部署在有着严厉的资源限制，情况恶劣，以及动态和不可预测的资源争用等特点的环境里。

服务质量合同(QoS Contract)

服务质量合同模式从一个功能应用的角度分离服务质量(QoS)的测量、适应和管理。它提供了一个结构性构造用于表达所需要的服务质量(QoS)和系统中可获得的QoS。

例子

假设有一架负责执行预先计划的任务的战斗机，和一架协同命令和控制(C2)飞机，主要用于当战斗机接近目标时不断更新战斗机的任务参数。特别地，C2可能想要把战斗机指引向一个新的更危险的目标或者警告战斗机在到达目标的途中有威胁，如图1所示。无论如何，这项从C2到战斗机的合作式的任务重计划必须处理资源的限制和争用，因为这两架飞机之间的无线网络的带宽是有限的，而飞机上的处理器同时还服务于其它可能的关键任务。由于任务重计划和有关数据的动态自然属性，通过静态地预分配足够的资源来支持这种合作是困难甚至是不可能的。如果系统依赖于资源总是可获得的和可预测的，它可能不能及时交换重要的任务计划信息并以此采取行动。

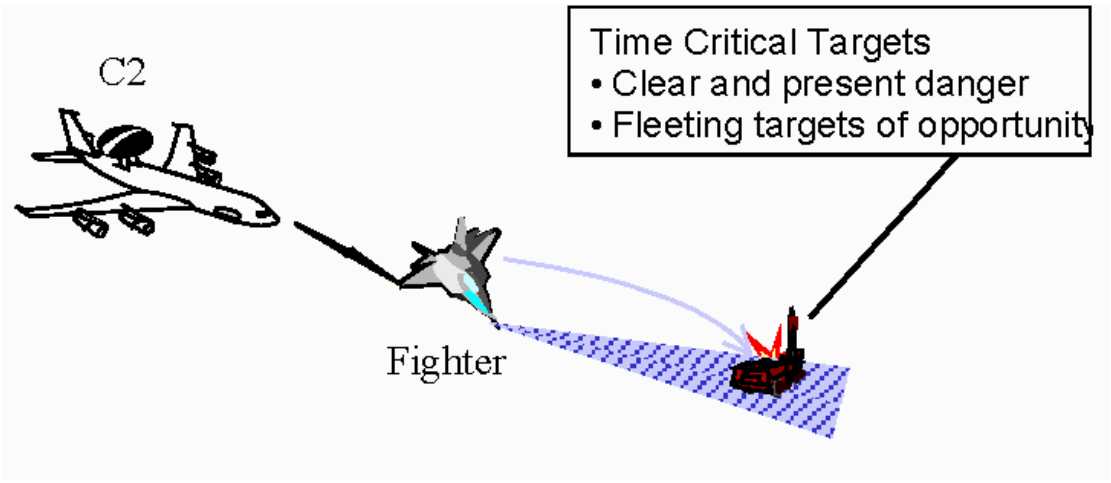


图 1 飞行环境下的动态任务重计划

处理被约束的共享资源所引起的问题的一种方法是压缩正在发送的信息，在这个例子里是图像。当带宽变低时，可以通过压缩图像来节省网络资源。无论怎样，这将增加要求信息处理器的工作量，从而会影响其它工作并且迫使系统决定哪个工作应获得处理时间。压缩图像也可能降低飞行员所依赖于完成任务的信息的质量。

图像瓦片化，即将一幅单个的图像分割成一系列的小图像，并在接收方再将它们拼接起来，是优美地降低图像质量的一种可能的解决方法。图像的重要部分进行高质量的传送，而非重要的区域则被质量降低。该方法以软件的复杂度为代价更好地利用可获得的资源。

考虑折衷的知识和在应用过程中解决它们的方法增加了应用的复杂度。之前程序员只需考虑图像的发送和接收，现在它们不仅需要考虑系统的功能方面，而且还有QoS方面。应用变得更复杂，由于QoS因素和功能性纠缠在一起，涉及带宽监控的因素，以及图像压缩的数据质量的折衷，CPU和带宽，战斗机中处理器的分配，和图像的瓦片化。

上下文

一个可以(采用不同的服务质量)运作于资源可用性的多种水平的应用。

问题

DRE应用具有竞争的QoS目标并且必须能够运作于严格的资源限制下。在一个应用中采用硬编码的服务质量进行控制会使该应用变得复杂，不可移植和难以维护，并且当系统演进发展时会导致问题。网络化的嵌入式应用会为共享资源竞争，这种竞争需要适当的仲裁从而使系统作为一个整体能够满足对它的需求。当资源不足以满足所有的应用时，以及在一个动态的环境里，有些应用不得不在少于最佳数量的资源下运作。此外，DRE上下文中还会

对于为了某一特定目标环境或配置而对通常的应用的运转状态进行定制有连续的需求。这种定制包括在一个更大系统的知识指导下的本地级的运行时适应。这种指导下的运行时适应能优美地降低任务和应用的颗粒，释放不需要的资源，并且在需要时再请求额外的资源。那些能够在不同操作条件下连续有效运行的DRE应用在面对停机和失败时更鲁棒，比(降低对资源的过度需求)非适应性的应用更加动态，并且不会有单个的故障点(集中资源管理)。

解决这个问题需要以下约束的决定：

- 在全部可能的操作条件下，并且考虑到所有可能的参数 – 计算的，物理的，和逻辑的；内部的和外部的，对一个复杂系统的所有状态进行建模是困难的。因此，在抽象，近似或者忽略其它条件时，决定必须能够考虑所感兴趣的条件。

- 所感兴趣的条件和这些条件定义的系统区域可以是相关的或者正交的。因此决定必须能考虑状态的组合，例如分层或者聚合。

- 在每个应用中直接加入关于整个系统和反应的信息会增加耦合度并且使系统作为一个整体更难以扩展。

- 功能性行为和应用都需要被配置和定制为在不同的环境中使用。

- 决定应该能够在本地或全局信息的基础上做出，并且应该能够在做分布式决定的过程中与其它应用合作。

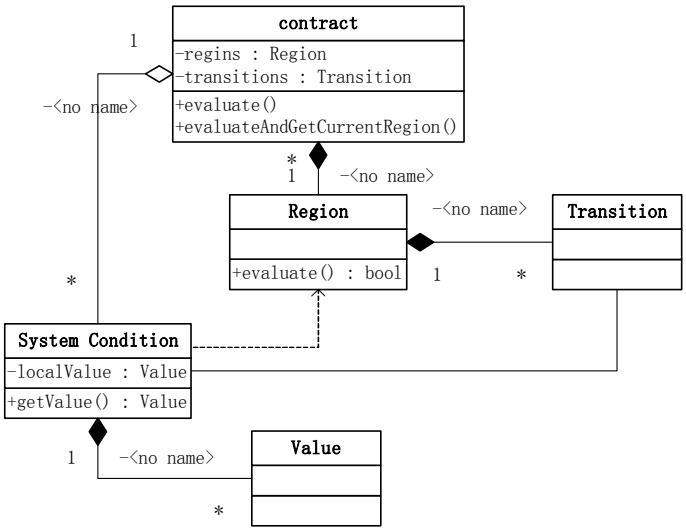


图2 QoS合同模式的UML图

解决方案

使用QoS合同模式将QoS的知晓和使用从应用的功能性中进行去耦分离。

QoS合同指定了一个应用所关心的QoS区域，包括该应用所能操作的QoS 水平，是用完全的还是降低的功能性。QoS区域使用对当前系统条件的过度测量的推断来指定。转换指定了由于QoS合同进入以及离开QoS区域所触发的行为。合同在区域之间的移动反映了运行时系统条件的变化，和它如何影响应用所期望的资源，以及应用所能够工作于其上的资源。

结构

在QoS合同模式中有六个主要的参与者，如图2所示。

- 一个合同(contract)指定客户想要的服务的水平,一个对象所期望提供的服务的水平,表明可能被测量的QoS的操作区域,和当QoS水平改变时所需要采取的行动。
- 系统条件(System conditions)提供了对于资源的接口,和需要被合同所测量和控制的系统的机制。
- 一个谓词(predicate)是关于系统条件的收集的一个布尔（Boolean）函数。如果系统的系统条件满足谓词，则该谓词将被评价为真。
- 一个区域(region)由一个谓词指定并包括转换。当系统条件符合那些由区域的谓词所指定时，则该合同被称为在一个给定的区域内。
- 一个转换(transition)描述了当合同从一个区域移向另一个时所需做的事情。转换可以为进入或者离开一个区域所指定，并且是命令模式的一个实例[3]。
- 一个值(value)是由系统条件所报告的系统的一个测量值。

动态性

对一个QoS合同的评估的测量和/或对一个系统的状态的影响包括以下由图3所标明的动态性：

1. 系统获得相应系统状态的一个快照(见快照模式)。这是重要的，因此对所有谓词的评估是基于相同的系统信息的值。

2. 通过评估哪些谓词(predicates)为真，来决定合同(contract)所处的当前区域。

3. 如果当前区域不同于先前的区域，则触发任何与转换(transition)相关联的行为。

合同可以用于带外(out-of-band)或带内(in-band)适应，即分布式对象相互作用的同步或异步。带外(out-of-band)合同与能够异步地执行应用的方法调用的一个控制线程相关联。控制线程可以周期性地触发合同评估，合同评估也可以和与系统条件的变化相关联的事件链接在一起。带内(in-band)合同在应用的远程方法调用的路径中的相关点作评估，例如远程方法的调用和返回。合同的评估可以被插入到方法调用的路径中，通过使用代理(Proxy[3])或拦截机(Interceptor[7])模式的实例。

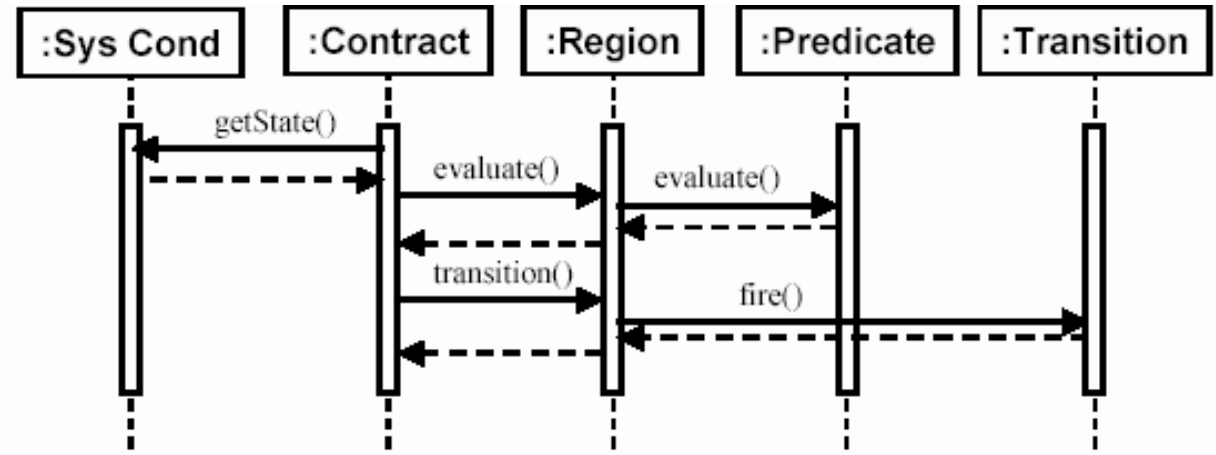


图3 QoS合同的交互图

实现

服务质量(QoS)合同模式的实现包括以下步骤：

1. 决定所关心的服务质量(QoS)区域。决定应用所能操作的服务质量(QoS)范围，从导致一种最佳应用运作环境的QoS的最高水平，应用能运作于某些降低的功能性或者折衷的较低的范围，到应用不能运作的最低的水平。

2. 确定包括或者控制QoS的系统条件。系统资源，管理者，基础结构的接口等可以提供在一个给定的时刻测量和控制系统中的服务质量(QoS)的方法。

3. 创建谓词(predicates)。每个区域需要通过对系统条件的一个谓词来描述。确定与每个区域相关联的系统条件的谓词。

4. 决定转换(transition)和动作(actions)。决定在运行时刻随着系统QoS变化所需要做的补偿，恢复，或者占有优势。把这些动作译成转换行为。

5. 将合同集成到应用中去。决定合同是被用作带内(in-band)还是带外(out-of-band)。如果合同被用作带内，创建一个代理(Proxy)或拦截机(Interceptor)将合同评估插入到应用的方法调用路径上的某个合适的位置。如果合同被用作带外，则为它创建一个单独的线程或者将它挂到已经存在的一个异步(对应用来说)线程的路径内，例如资源管理程序或者正被合同所观察的监视器。

例子的解决

战斗机和 C2 可以使用服务质量(QoS)合同来将应用代码和 QoS 管理分离开来，如图 4 所示。合同可以定义那些测量处理器使用，网络带宽，以及战斗机和 C2 结点之间数据交换速度的系统条件。合同可以定义处理由于网络带宽的约束或者在源或目标上数据处理的限制所引起的数据交换减速的区域。作为在区域之间的合同转换，应用可以通过请求更多的资源，通过网络和 CPU 的折衷(例如，使用压缩)，或者通过改编数据(例如，按比例缩放或者用瓦片图像)来适应。这种适应性的行为可以通过在转换，代理或者截流器中的对象调用来实现。

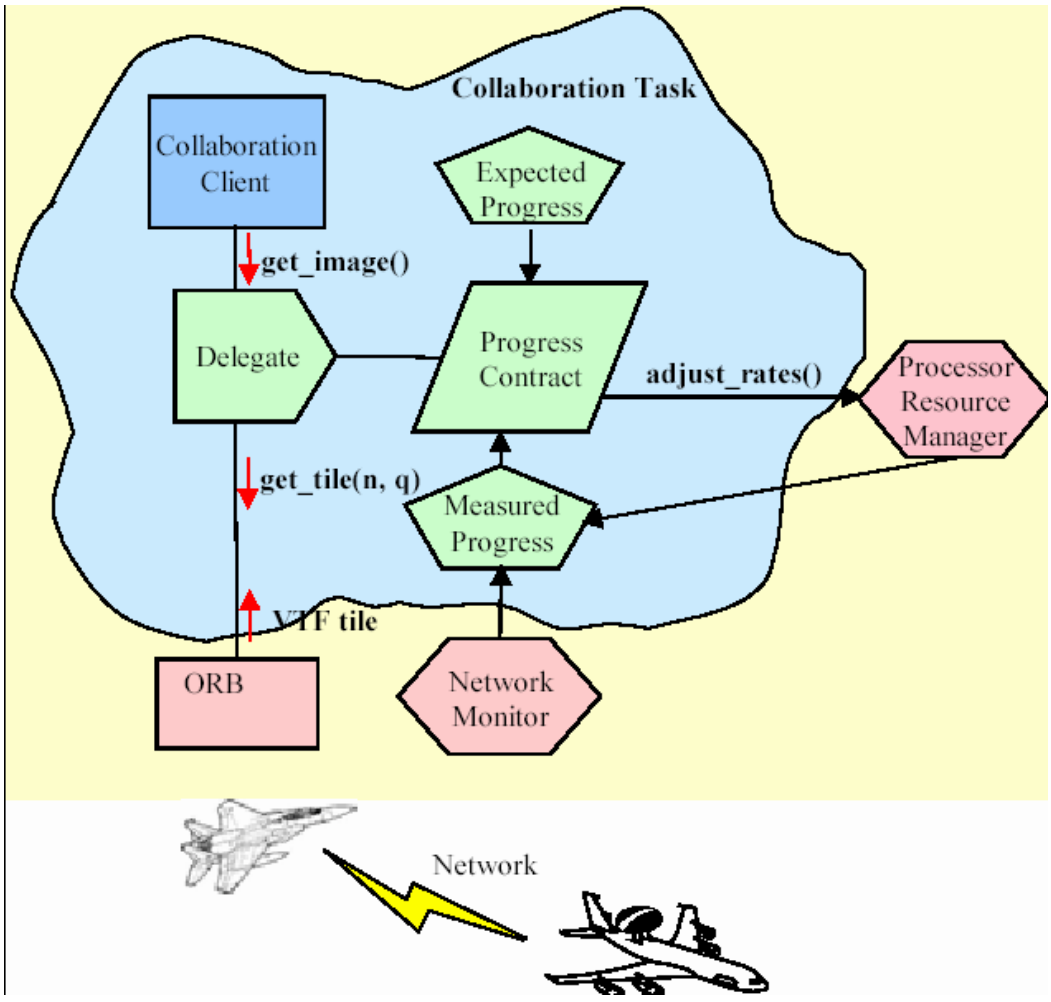


图4 在动态计划例子中，可以测量数据交换进程和可用资源的合同

变体

状态/区域。服务质量(QoS)合同模式的一个变体就是引入状态来取代区域。如果一个合同使用区域，转换可以从任何地区到任何其它地区。状态使得合同更像一个状态机[11]。从一个给定的状态，转换只允许转向有来自当前状态的转换的其它区域。

当使用状态，谓词是和转换而不是区域相关联。为了计算下一个状态，从当前状态开始，评估与每一个转换相关联的谓词来确定应该遵循哪个转换。

观察的/非观察的系统条件(Observed/Non-Observed system conditions)。系统条件有两大类,即观察的和非观察的。由观察系统条件所测量的值的改变会触发合同评估，可能导致区域转换并引起带外适应行为。非观察系统条件描述正在测量的任何条件的当前值，但是无论值如何变化都不会触发事件。相反，每当合同下一步被评价的时候，它们根据需要提供值。

已知应用

航空电子设备平台上的动态任务计划。武器系统开放式体系结构(WSOA) [2,6]是动态任务计划的航空电子设备应用，由一架命令及控制(C2)飞机和一架战斗机协作，在飞行期间不断地更新战斗机的任务参数，如图5所示。C2飞机将由图像数据组成虚拟目标文件夹(VTF)送至战斗机，并被处理以不断更新战斗机的任务参数。因为在C2和战斗机节点之间无线连接的自然约束，以及与其它战斗机的资源竞争，更多的飞行和关键任务，战斗机的任务，合作任务需要测量可获得的资源并且适应有效地使用可得到的资源继续在变化条件下完成它的目标。

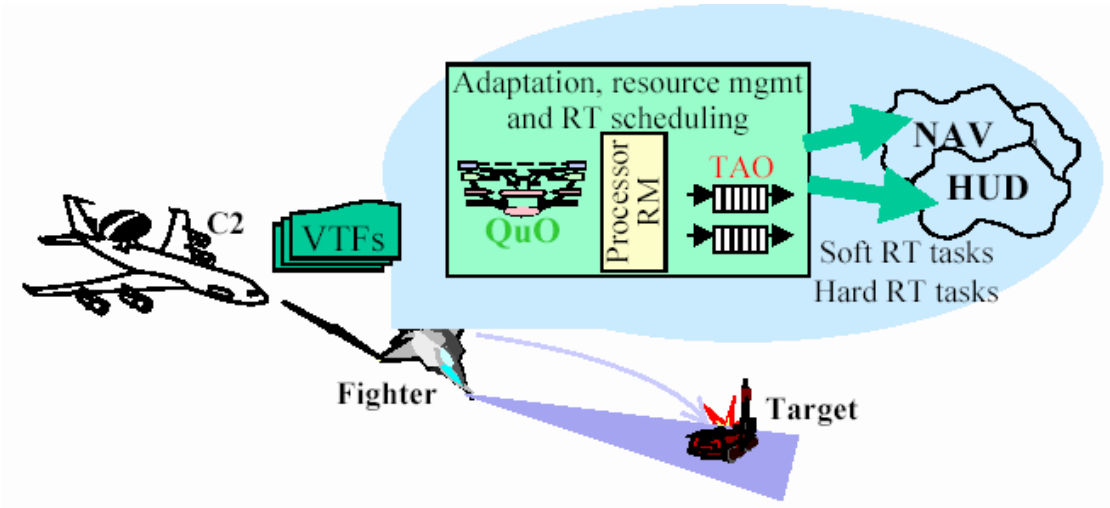


图5 C2飞机发送VTF数据到战斗机，以执行动态任务计划

该应用使用一个合同模式的带内适应的实例，以及在战斗机侧的带外测量和适应。在VTF图像下载期间合同通过测量下载进程并且与用图像压缩,图像瓦片化，处理器资源管理和网络资源管理一起调整适应，管理及时性与图像质量的折衷。

模拟UAV上下文中的视频分发。为美国海军和用于DARPA法定人数和嵌入式系统(PCES)程序设计所开发的一种模拟无人天线车(UAV)应用，详见[4,5]，专注于将由UAV捕获的视频传送给分布式的控制站并将控制信号返回给UAV。图6说明了模拟系统的体系结构。它是一个三极管道，有多个UAV(以多种格式，例如压缩MPEG 和非压缩PPM)用于将视频发送到图像分发进程。这些UAV通过连续读图像文件和实时相机输入来模拟，其中有一些UAV附于在步行机器人上。应用包括无线和有线的Ethernet连接，用多种网络资源条件实验。目标控制站和视频显示有不同的任务要求：一些需要低延时的录像，另外一些要求高分辨率，而其它的-用于自动目标识别(ATR)过程-则要求重要的视频内容。

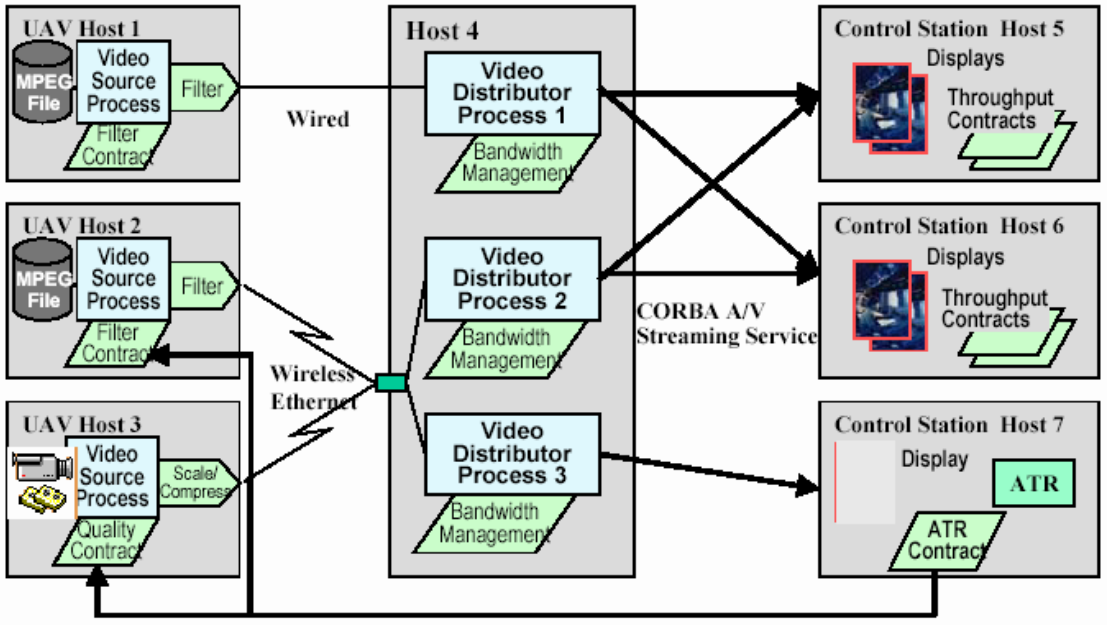


图6 UAV架构展示

在整个分布式应用中的一套合同可以用来提供不变负荷的性能。合同管理带宽(使用RSVP，DiffServ和应用适应)，流量(管理视频帧率)，视频源(管理过滤，缩放和压缩)和控制信号(对ATR报警的响应)。为了支持这个，有监测帧率，主机负载，ATR状态，网络负载等的系统条件对象，以及不同的合同用于获取相应系统状态的快照。

结论

服务质量(QoS)合同模式具有以下好处：

- 与变化隔离。把行动与决定使用行动的过程分开以允许它们独立改变。
- 简化重新瞄准应用。通过把合同与它们调停的应用及它们测量并且控制的系统条件分开，修改应用的范围受到了限制。
- 增加重用。将数据收集从用于促进系统条件创建的行动分开，使得其在设计之初就考虑了重用。
- 减少开发。合同可以通过公用中间件得到支持。

服务质量(QoS)合同也会导致以下缺陷：

- 额外开销。由合同引入的间接附加(这和把QoS控制直接嵌入应用中相反)可以导致开销增加。
- QoS区域的静态决定。关心的QoS区域必须被预先决定，这将限制控制的颗粒性和合同的灵活性。

参见

- 质量连接器(Quality Connector)[10]。质量连接器涵盖给服务质量(QoS)合同的相似区域。两个都是处理如何把QoS所关注的与应用所关注的分开。质量连接器关心怎样实现应用和QoS部件之间的交互作用。另一方面，QoS合同则专注于QoS规格和决策而较少考虑部件之间的连接机制。

快照

快照(snapshot)模式提供了在一个给定时刻系统状态的连续性视图的一个有用的近似值。这是一种使运行时间适应性决策变得容易的模式。

例子

考察一个天气报告系统。系统由一组分布式的传感器组成，并且负责当被请求时创建一个系统的综合图像。传感器被安置在各个位置不显眼的地方，并且具有可能被恶劣情况间歇影响的连通性。即使是在最佳情形下通向传感器的带宽也是有限的。

当一个用户想知道当前情形时，系统需要分别联系每个传感器并且查询它的状态。不幸的是，由于低的带宽和不可达的传感器，经请求产生一份报告是十分费时的，并且由于不可达的传感器，信息可能丢失。

上下文

一个需要系统视图但是其收集此类信息的能力又被约束的分布式或嵌入式系统。

问题

一个适应性分布式系统需要在一定的时间范畴内有相应的分布式系统的一个合理准确的视图。这需要以下的约束：

- 一旦被请求，系统状态的视图应当在一个可预测的有限时间内是可获得的，以使其对于决策是有用的。这意味着评估系统状态的请求必须立刻或者仅仅在一个非常小的有限计算时间内返回。
- 一个快照必须是系统状态的连续性视图的一个有用近似，意味着它应该由所收集到,经过计算的，或者不晚于一次小的时间阀值和在从一个到另外一个小的时间阀值的预测的测量值组成。
- 多个分布式的消费者可能为不同的目的对相同的数据感兴趣，并且不必知道彼此。
- 收集和总计数据应该和使它可获得断开。

- 一张系统状态的快照经常包括具有不同属性的系统参数的多个值。有些并不频繁改变，而另一些比如系统时钟则频繁变化。有些是要收集的简单度量，而其它可能需要一些处理或者计算才能得到。

解决方案

使用快照模式提供信息的供应者和消费者之间的间接方式。通过从对信息的要求中分割信息的收集，信息可以在当资源允许时被收集并且被查询而不干扰供应者。

细节：建立一套系统条件和收集者。系统条件负责查询状态和存储结果。状态可以是可能有用的系统状态的任何一个片断。收集者负责收集所有这些数据，以及将它们综合整理成系统的一个视图。

结构

在快照模式中有两个主要的参与者，如图7所示。

- 系统条件提供一个系统部件的Wrapper/Facade[7]，称为值(Values)，并且通过一个方法getValue提供访问这些组成部分的状态。

- 收集者收集并且将系统条件(可能是分布式的)所具有的值转换成系统状态的一个快照。收集者可能是系统条件本身，如果它们是提供给更高层次的收集者的话。

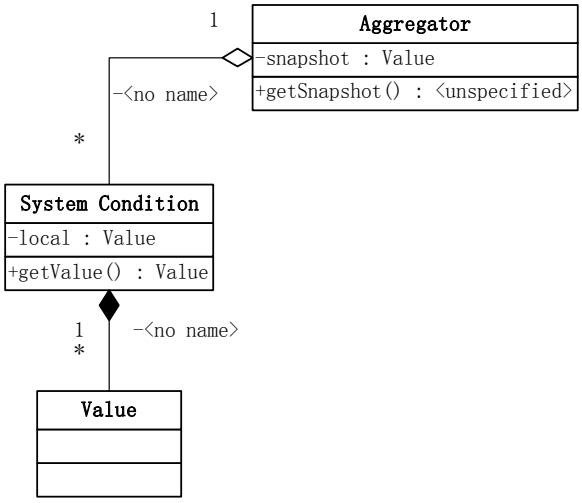


图7 快照模式的UML图

动态性

获得系统状态包含以下几步,如图8所示：

- 一个系统条件从与它有关联的值中收集状态。
- 一个收集者使用getValue查询每个系统条件的当前值。
- 收集者接着通过调用getSnapshot方法，收集值并且将它们转换成可以被返回的一个综合的快照集（如一个链表或数组）。

实现

有3 项主要活动与实现这种快照模式实现相关。

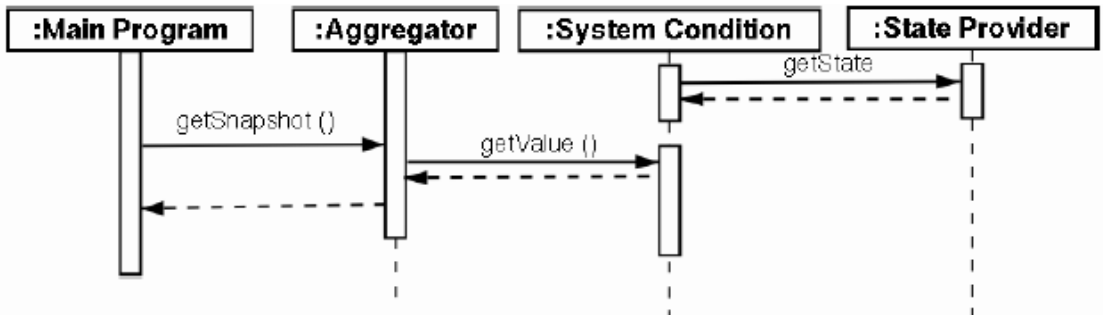


图8 快照的交互图

- 实现封装状态的来源并且提供一个getValue方法的系统条件。当一个系统条件的getValue被调用时，系统状态立即或者在一个非常小的有限计算时间内返回值。系统条件对象的实现可以是任意的，复杂的，只要值总能够被及时地返回并且系统条件值的替换是一个原子操作。系统条件可能缓存值以便于它总能够及时地准备好一个响应用于返回。
- 决定哪一种方式更合适：当这些系统参数是远程分布式的时候，将系统条件对象放在和收集器相同的主机上还是放在它所要测量的系统参数的主机上。该决定必须考虑哪个位置提供了更精确和更及时的系统状态的快照，以及不同位置对系统的影响。
- 实现收集器。当通过一个 getSnapshot 调用被询问状态时，收集器串行或并行地查询它所监视的系统条件，并且返回相应的系统状态。

图9表明了不同类型的系统条件的例子。从左到右分别是：

- 一个简单的值，仅仅维护一个由客户或其它实体设置的值。
- 一个被测量的值，例如网络流量或者系统的平均负荷。
- 一个组合值，它可能集合了来自许多源的值，而那些值可能也是系统条件对象。
- 一个经过计算或处理的值。
- 一个状态值，可能被维护或者收集于其它地方，例如在ORB、OS或者其它源。
- 一个基于历史行为的预测值。

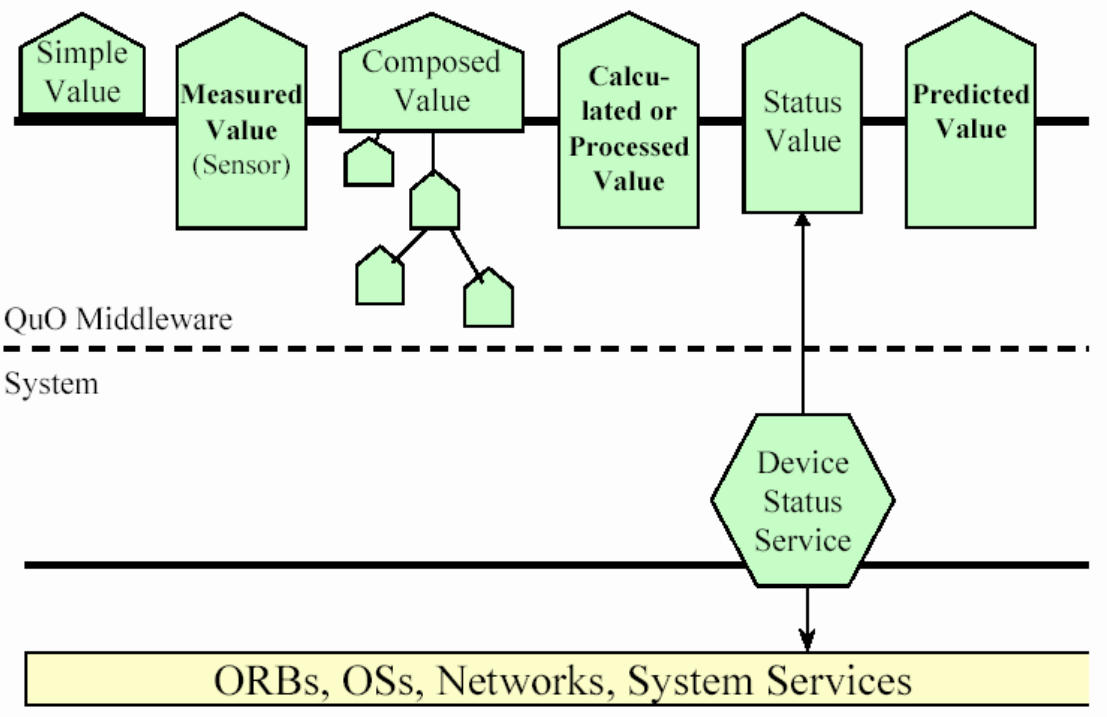


图 9 系统条件对象的不同类型

例子的解决

天气系统可以使用快照模式。当资源允许时，系统条件可以建立在将会查询传感器的值的显示器的附近。当被请求当前系统状态的一个视图时，显示器可以查询收集器，而收集器会从系统条件处得到传感器的值并且将它们及时地返回用于显示。遗漏的值可以用上一次已知的值或者一个预定的值代替。

变体

一致快照(Consistent snapshots)[1]。迄今为止所讨论的模式是一种松散的快照(loose snapshots)。当使用松散快照时，一些额外处理可能产生更接近最近回复的事实是容许被交换作为一个有利的回复的。如果使用了多于一个从公共系统参数继承来的测量值，而那些参数又在快照生成时发生了变化，系统状态的松散快照可能包含不一致。当此类行为不可接受时可以使用一致快照。在一个一致快照中所有要收集的状态应该是一致的并且来自于相同的基础值。获得一致快照的一个方法是使用事务处理，即当状态被收集时系统从处理中停止。为了得到系统的一致视图，代价是系统停止而产生延迟。

系统状态推/拉。系统条件可以查询它们所监视的状态，即拉模式，或者使用推模式，即具有的状态是由那些其状态被系统条件所监视的部件所提供。

已知应用

航空电子设备平台上的动态任务计划。在QoS合同模式中所描述的WSOA项目中,QoS合同通过获得测量图像下载进度和CPU资源分配的系统条件对象的快照来监控图像下载的进程。

模拟UAV上下文中的视频分发。在QoS合同模式中所描述的一个模拟UAV系统中,有系统条件对象用于监控帧率,主机负荷，ATR状态，网络负载等等，和各种在确定什么是系统QoS的状态之前用来获取相应系统状态快照的合同

人员会议。当有关工作人员为一个组会议聚集时,每个成员分享他们当前状态的一个高层次的视图。而当会议结束时整个小组都作为一个整体了解该小组的状态。

结论

快照模式具有以下好处：

- 通过将计算系统状态的任务从查询系统状态的值中分离开来，可以容易并且有效地增加新的收集器。
- 不管它们的数据格式、可获得性和其它特性的差别，系统条件的接口允许新的系统条件以一致的方式被测量。
- 松散的快照可以有效地获得。
- 一致快照可以精确地表述系统的状态。
- 快照提供一个那些影响系统行为的条件的一个可见焦点。

快照也会导致以下缺陷：

- 如果没有人对它感兴趣，状态可能被不必要地计算。
- 系统状态快照的精确度仅在一个特定的时间阀值内有意义。
- 一致快照的开销通常大于松散快照。它们可能需要额外的通讯，而松散快照则不需要。
- 系统视图的松散快照不一定总是一致的。

参见

- 快照模式使用了Memento [3]模式来封装系统的状态。
- 收集器可以被看作Observer[3]模式的实例。

致谢

作者希望感谢我们的牧师Joe Cross以及来自PloP 2002和OOPSLA 2002的DRS focus小组，为他们所提供的帮助和极具价值的意见。

参考文献

[1] Chandy K., Lamport L. “Distributed Snapshots: Determining Global States of Distributed Systems,” *ACM Transactions on Computer Systems*, Vol. 3, No. 1, February 1985.

[2] Corman D, Gossett J. “Weapon Systems Open Architecture – Using Emerging Open System Architecture Standards to Enable Innovative Techniques for Time Critical Target Prosecution,” 20th Digital Avionics Systems Conference (DASC), IEEE/AIAA, October 2001, Daytona Beach, Florida.

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

[4] Karr DA, Rodrigues C, Loyall JP, Schantz RE, Krishnamurthy Y, Pyarali I, Schmidt DC. “Application of the QuO Quality-of-Service Framework to a Distributed Video Application,” Proceedings of the International Symposium on Distributed Objects and Applications, September 18-20, 2001, Rome, Italy.

[5] Karr DA, Rodrigues C, Loyall JP, Schantz RE. “Controlling Quality-of-Service in a Distributed Video Application by an Adaptive Middleware Framework,” Proceedings of ACM Multimedia 2001, September 30 - October 5, 2001, Ottawa, Ontario, Canada.

[6] Loyall JL, Gossett JM, Gill CD, Schantz RE, Zinky JA, Pal P, Shapiro R, Rodrigues C, Atighetchi M, Karr D. “Comparing and Contrasting Adaptive Middle-ware Support in Wide-Area and Embedded Distributed Object Applications”. *Proceedings of the 21st IEEE International Conference on Distributed Computing Systems (ICDCS-21)*, April 16-19, 2001, Phoenix, Arizona.

[7] Schmidt D., Stal M., Rohnert H., Buschmann F., *Pattern-Oriented Software Architecture: Patterns for Concurrent and Networked Objects*, Wiley and Sons, 2000.

[8] Zinky J., Bakken D., Schantz R. “Architectural Support for Quality of Service for CORBA Objects,” *Theory and Practice of Object Systems* 3(1), 1997.

[9] Cross, J., Schmidt, D. “Quality Connector – A Pattern Language for Provisioning and Managing Quality-Constrained Services in Distributed Real-time and Embedded Systems”, *Proceedings of the 9th Annual Conference on Pattern Languages of Programs, PloP 2002,* Allerton Park, Illinois, September 2002.

[10] Bo I. Sandén. “The State-Machine Pattern”, *Proceedings of the conference on TRI-Ada '96: disciplined software development with Ada*, Philadelphia, Pennsylvania, 1996

征稿

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

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

一半是欣喜，一半是迷惑--XDE for .Net 体验

Windy. J 著

查看评论

XDE 面世已经一年多了，前段时间终于在 2003 版的.net(C#)下安装体验了一下。安装的时候软件名是“Rational XDE Plus .net (Evaluation)”，看起来还是无可厚非的，但是，当我想到 Rational 网站上找一些相关介绍的时候，居然怎么也找不到这个软件，在琢磨了好大一会之后，才发现它的大名原来是“IBM Rational Rose XDE Developer”（.net 版），与之遥相呼应的竟然是 IBM Rational Rose XDE Modeler，Rose 乎？XDE 乎？IBM 乎？Rational 乎？Developer 乎？Modeler 乎？真不知道是厂家想把我们弄得晕头转向呢，还是他们自己已经被他们的产品弄得晕头转向了，晕乎哉，晕乎矣！

Rational Rose XDE Developer

闲话不表，下面我们就步入正题吧，为了简单起见，我将在下面用 XDE 代表上面的“IBM Rational Rose XDE Developer”，用 Rose 代表他们的“IBM Rational Rose XDE Modeler”，在这里敬请厂家见谅了。

安装

安装就省略了吧，其中只有一个需要注意的小地方，就是如果你同时安装了.net 开发环境的 2002 和 2003 两个版本，到时候需要选择其中一个版本来安装 XDE，另外一个版本下不能使用 XDE。也就是说，XDE 只支持其中一个.net 开发环境。

开始同步

首先，建立了一个 C#的 Solution，Solution1，然后增加了一个 ClassLibrary 的工程，因为是试用，就存在 temp 目录下，ClassLibrary1 好了（其他类型的工程也可以，不过 ClassLibrary 比较简单，不像 Windows 应用那样会引入一大堆系统模型），然后就可以选中工程，开始同步（Synchronize）了。

非常快，同步完毕，在输出窗口中可以看到同步完成的提示，因为还没有实际的类或模型，这次同步只会在工程所在目录生成一个同名的 mdx 文件，双击这个文件，将打开模型浏览器，可以看到，在模型浏览器里，基本上没有什么，就是一个跟工程同名的名字空间。

这个时候，可以在 C#工程里头增加一个类（代码），或者在模型里头增加一个类（图形），例如叫做 Carfield，增加几个属性，Birthday，Weight，Color，等等，同步，一切 OK，将会自动生成相应的类（模型）或代码（C#文件）。

下面在给模型增加一个 ClassDiagram（选择模型工程，右键，有相应的菜单），然后把 Carfield 拖动到这个 ClassDiagram 上（可惜它将是一个跟 Rose 中类似的方框，不是那只橙色贪吃的胖猫），再增加几个新的类，例如咖啡猫喜欢吃的 MeatLoaf，给它一个 Toy（它实在是不缺这个），建立关联，命名，给它们增加属性。

再同步，没问题，新的代码又自动生成了。这个时候，同步非常好用哦！而且，也就这么简单，一个工程对应一个 mdx 文件，对于已经同步好的模型和代码，还可以方便地切换，一般是双击，或选择“Browse model”和“Browse Code”菜单。

同步设置

只有工程在“Code Project”状态下才可以进行同步，在 Model Explorer 中选择具体的 Model，右键，选择属性窗口，最末一栏“*Stereotype*”，可以看到默认的选项是“*C#*”，这样就是“*Code Project*”，即代码工程，如果再选中“*Assembly*”，工程就会变成“*Project Reference*”，从而不能同步了。

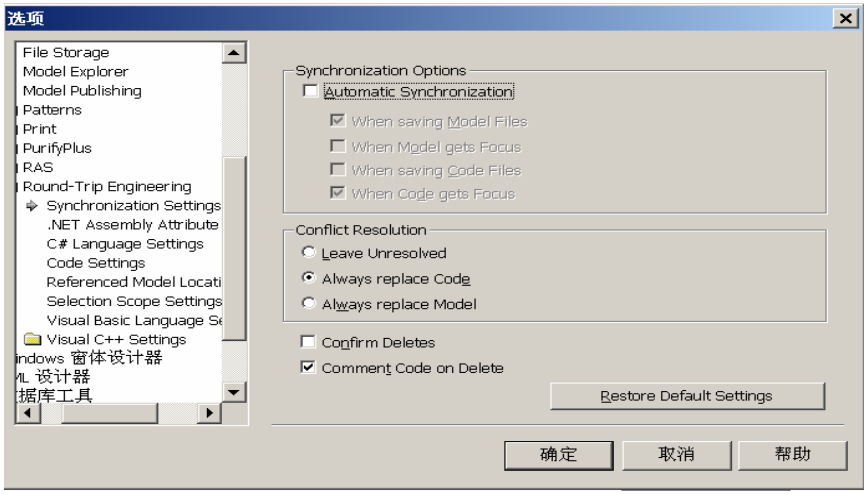
例如在上面的 Solution 中增加两个 ClassLibrary 工程 ClassLibrary2 和 ClassLibrary3，如果它们与 ClassLibrary1 之间是程序集引用，那么它们的 *Model* 属性将自动设置为“*C#*”，您的同步不受任何影响，如果是动态库引用，就要每次手工调整模型的版型设置，需要同步的时候设置成 *C#*，同步完以后设置成 *C#*和 *Assembly*。

从代码同步到模型，可以选中工程，右键，选择 *Reverse Engineer*；

从模型同步到代码，可以选中模型，右键，选择 *Generate Code*；

如果选择 *Synchronize*，这是代码和模型同时同步。

同步策略的设置在*工具—选项—Rational XDE —Round-Trip Engineering*，*Synchronization Settings*，如图所示：



如果选中自动同步（*Automatic Synchronization*），就可以在子选项中选中自动同步的时机设置，如果不选中，需要象前面讲述的那样手工进行同步。

中间是冲突解决方案，根据各自情况的不同，分别选择保持不变，总是替换代码，或总是替换模型。

后面的设置是“*确认删除*”和“*删除时注释代码*”，分别设置删除时的动作。如果选中“*确认删除*”，在删除代码或模型的时候，相应的同步删除不会自动完成，而是在同步信息输出窗口给出用“*!!!*”开头的警告提示，提醒你手工进行相应的模型或代码的删除。

如果不选中，代码和模型的删除会在同步时删除对应的模型和代码。

删除时注释代码，表示对代码的删除是以注释的形式进行的。

进行代码与模型同步时，如果需要同步的元素多，则需要一些等待时间，同步完成之后，注意查看输出消息窗口，如有错误这里会给出提示，同时，还会在 Task List（任务列表）中列出可能的错误，在某个任务上单击右键，会出现一些选择，其中有“*Browse Code*”，“*Browse Model*”等。

还可能会出现这样的提示，表示同步未进行：

--- 11:05:39 Synchronizing project "ClassLibrary3"...

--- 11:05:39 Getting implementation model for ClassLibrary3

--- 11:05:40 Nothing synchronizable selected.

--- 11:05:40 Synchronization took .526 seconds

此时需要检查引用模型，工程编译情况，环境设置，看是什么原因导致不能同步。

或者会出现这样的提示：

“不能取消同步，因为已有一个任务正在进行中……”

遇到这种情况，目前还没有什么好办法，可以试试“*Stop Round-Trip Engineering*”，或者重新打开工程。

注意，同步时信息将输出在 output 窗口中，其中的错误还会出现在 Task List 中，错误可以手工排除（特意这么提是因为代码模型同步是自动进行的，容易导致一种不能手工排除错误的错觉），例如，一个 C#工程原来引用了另一个工程，后来删除了这个引用，但模型中还存在模型引用，就可以手工删除模型中的引用。

显示设置

你可能想改变一下 Diagram 上的模型显示，让它们更整洁，尤其是在一个类有了许多属性和方法，可能每个属性还有一对 get/set 方法的情况下。

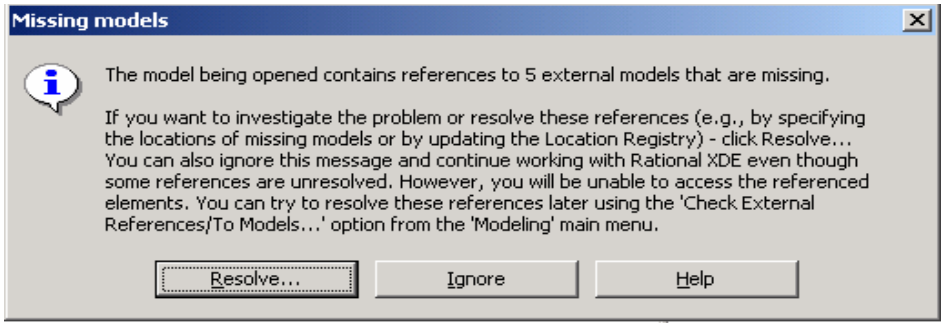
字体，填充样式，线型样式，在工具栏的 *Rational XDE Appearance* 中可以改变，显示样式在“*格式*”下拉菜单后面的英文子菜单中指定。其中有个 *Compartment* 菜单，指定模型显示的间隔类别，可以指定属性，操作等，这样可以自定义图形显示的详细程度，也可以避免因为显示了太多的内容让屏幕拥挤不堪。

还可以选中某一个具体的图形，改变它的显示样式，显示内容，例如，可以指定所有的私有变量和 get/set 方法都不显示，这样模型看起来比较简洁，按鼠标右键，有相应的菜单。

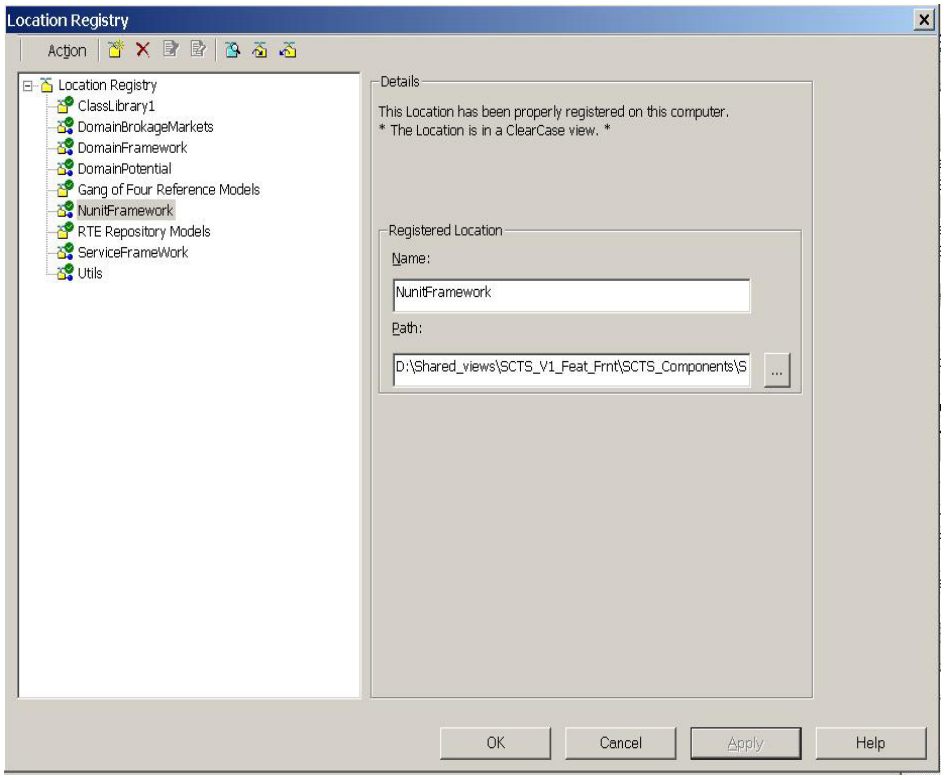
缺点是：显示设置不够灵活，同步以后原来设置将被改变，影响模型的显示。

引用模型路径

在打开 mdx 文件（模型）的时候，可能会遇到模型引用问题，这是因为工程引用到别的动态库，相应地，模型也要引用到动态库对应的模型，当出现“缺少引用模型”的对话框时，



选择“Resolve”，将出现类似下图的窗口，

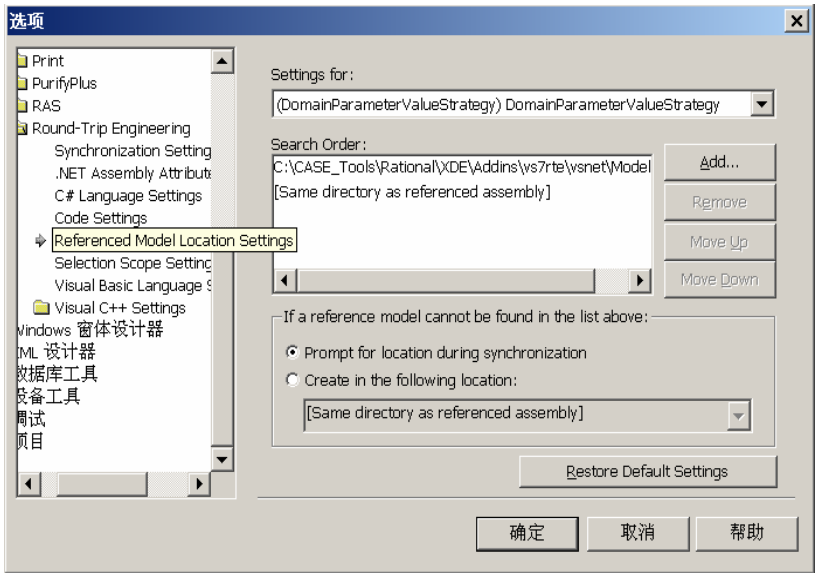


并在“*Path*”选择相应的 mdx 文件，就可以了。

可以由 *Modeling*－*Check External References* 来检查模型和代码的外部引用情况。

模型引用路径还可以通过手工打开 *Modeling---Location Registry* 菜单设置。

另一个设置的地方是：*工具－－选项－－Rational XDE －－Round Trip Engineering*，如下图所示；



这两种设置引用路径的方法一样让笔者很迷惑，有什么差别呢？如果说是外部系统的引用模型和本系统中互相引用的模型之别，又不尽如此，一个比较有说服力的说法是，前者设置的引用模型路径支持 Clearcase（配置管理工具），而后者不支持。但经笔者多方试验，前者同样支持本地路径（同后者），那么，后一种设置方法到底有什么用，或者这两种设置方法有什么深意？目前还不得而知。

发布和报告模型

在工具菜单里有，*Publish Modal* 和 *Generate Model Report*，可以发布模型，和生成报告，注意，发布和报告时不支持中文路径。

文字说明和注释

对模型的注释可以写在 *Model Documentation* 窗口，在代码中将生成如下的注释，可以同步：

/// <summary>

/// 注释文字

/// </summary>

美中不足

目前 XDE 不支持协作图，也不支持代码和序列图的同步。可在运行时产生整个系统的一个大序列图，但是，毕竟跨度太大，而且无法控制消息的详细程度（想想看，可能一个方法里面有深度调用的各个方法，每个方法都画到序列图上，将会多恐怖）。所以，最好是支持代码到序列图的同步，而且展开的级数应该能指定，这样会比较友好。

另一个是模型的显示策略目前还比较单一，也比较繁琐，如果有更友好的方法就好了。对模型工程的属性设置也显得不够灵活。

其次，自动产生模型（mdx）文件时，笔者还没有找到自定义设置文件路径的地方，似乎只能生成在代码工程的同一目录。

Q&A

Q1：前面提到先创建 C#工程，再同步产生模型工程，那么可否反过来，先创建模型工程，再同步产生代码工程呢？

Sorry，好像不可以。

Q2：前面说到一个代码工程对应一个模型工程，如果我们的系统拥有多个代码工程，那只能对应到多个模型工程了？

对，但是可以创建一个总的模型工程，在这个总工程里引用其他子工程里的模型，可以把图画在这个总工程里，放心，模型会跟子工程同步更新的。

文末感言

从笔者试用的程度来看，XDE for .net 的正逆向工程功能确实不错，又快又准，实现代码和模型的同步比较理想，当然，如果能改善前面那几个美中不足，以及明确那个引用模型的疑问，再有就是，干脆起个简单的（例如就是 XDE）名字，不要那么长那么多形容词，就更好了。

其次，Rational 曾多次强调，XDE 不是 Rose 的替代版本，而且，Rose 也还有新的版本出来，但是，他们也同样强调，在 XDE 里可以和 RequsitePro，ClearCase 协作，可以完成从需求建模到分析，到设计，开发的一系列建模工作，也就是说，完全可以替代 Rose，这样一来，使用者就非常迷惑了，到底该用什么呢，怎么用呢，用到什么程度呢？在纯粹的分析人员看来，在一个代码开发环境中进行建模，将会是一种全新的体验呢，还是难以承受的郁闷？

The Addison-Wesley Signature Series

Patterns of Enterprise Application Architecture

中译本即将上市

Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford

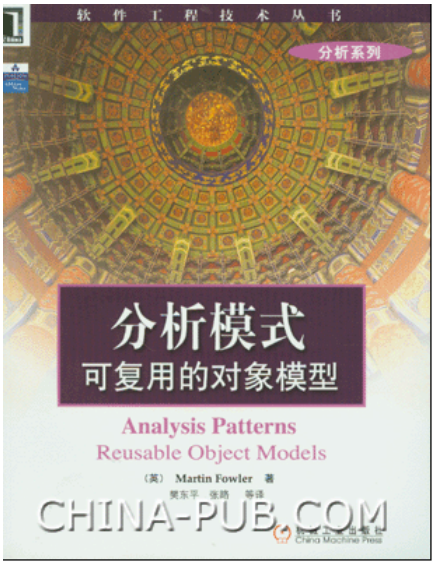
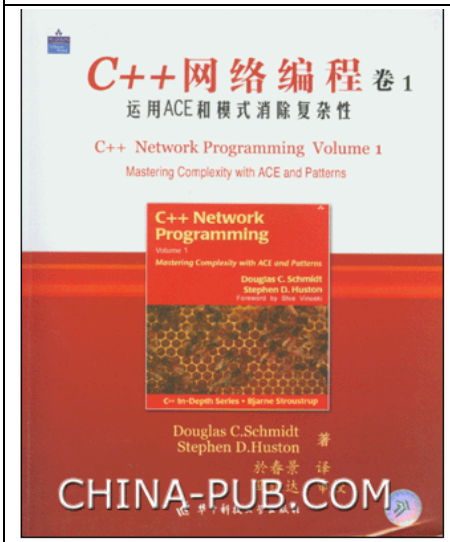
UMLChina 负责中译本最后审校，并指定为训练用教材

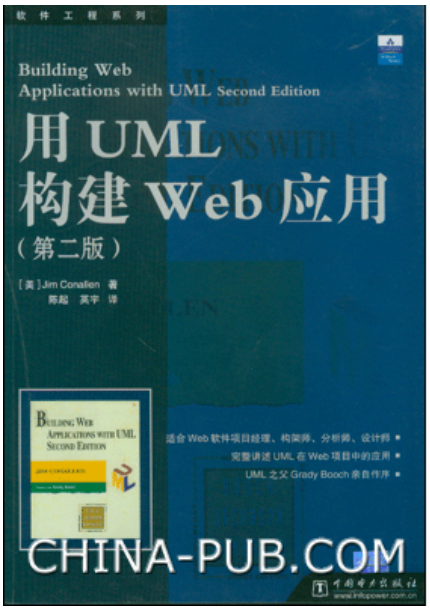
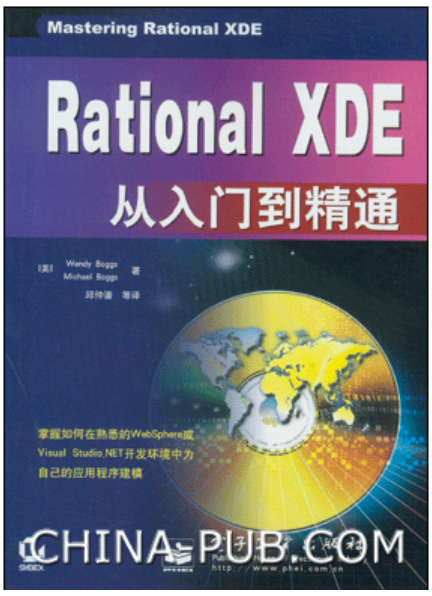
值得看的中译本 UML 相关书籍（截止至 2004 年 1 月）

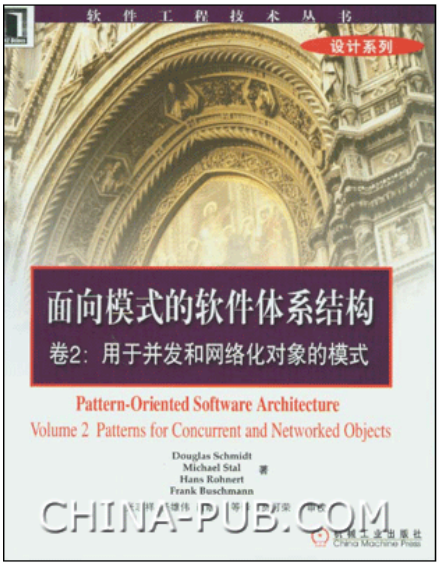
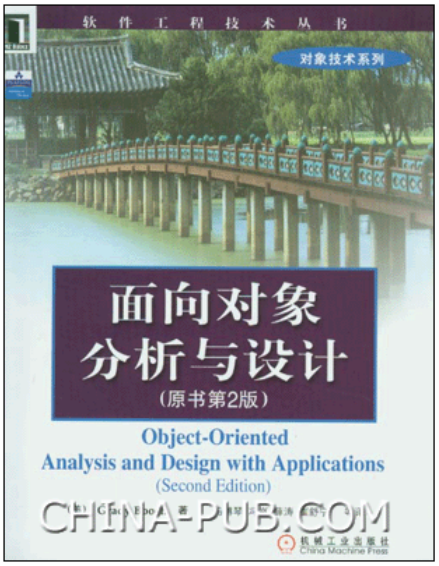
UMLChina 整理

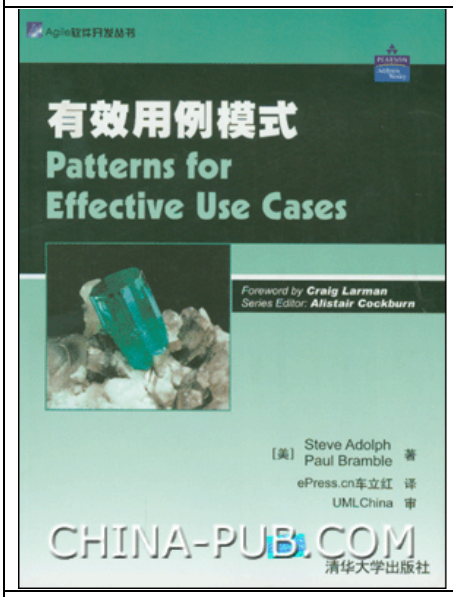
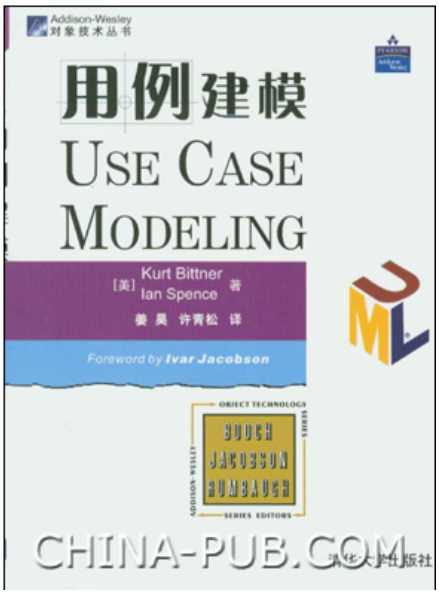
查看评论

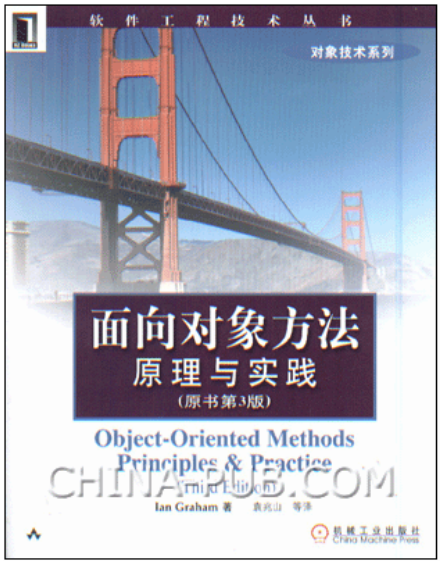
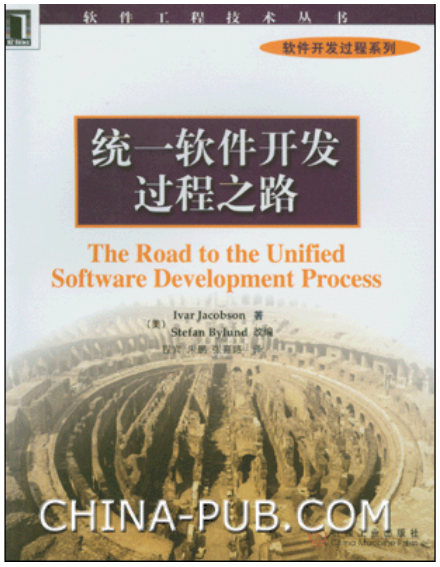
现在，引进翻译的 UML 相关的书籍越来越多，在不可能一一阅读的情况下，开发人员面临着一个问题：哪些书能真正带来帮助？UMLChina 的专家总结了 2004 年 1 月之前出版的所有 UML 相关书籍，根据自己的认识，“过滤”出以下优秀书单，按出版日期排序。不作任何评点，仅为一家之言，供大家参考。最后要说的是，就算读 10000 本书，如果没有将其中的知识用于你自己的实践，对你来说也是无用的。

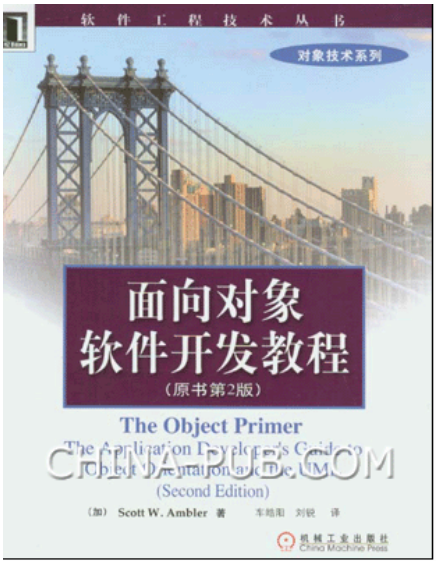
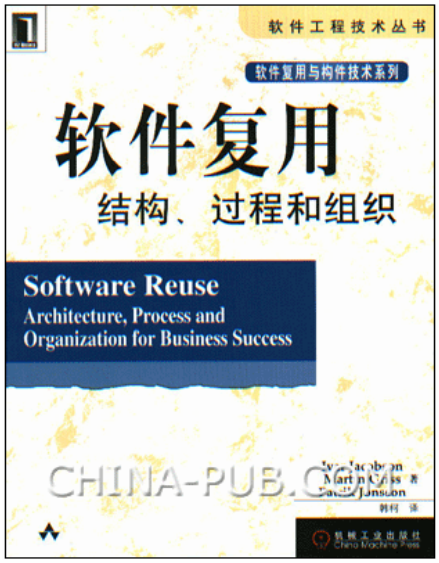
书	作者	出版社	出版年份	理解难度
	Martin Fowler	机械工业	2004	★★★★★
	Douglas C.Schmidt	华中科技	2003	★★★★★

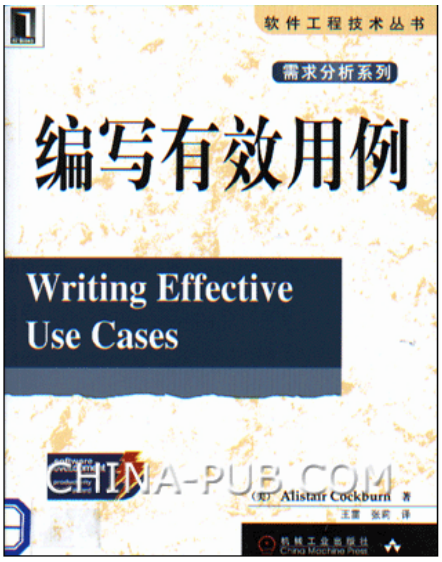
	Bruce Powel Douglass	中国电力	2003	★★★★
	Jim Conallen	中国电力	2003	★★★★
	Wendy Boggs, Michael Boggs	电子工业	2003	★★★

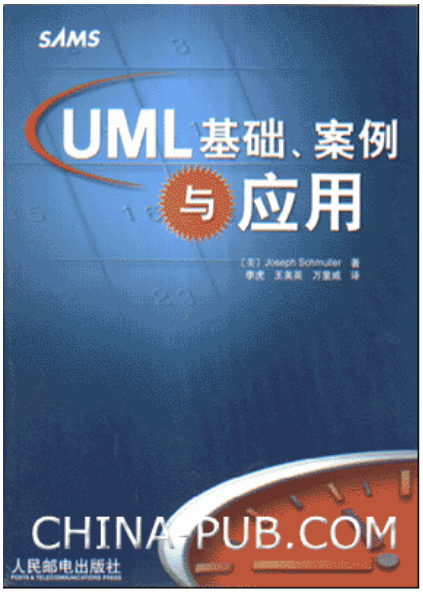
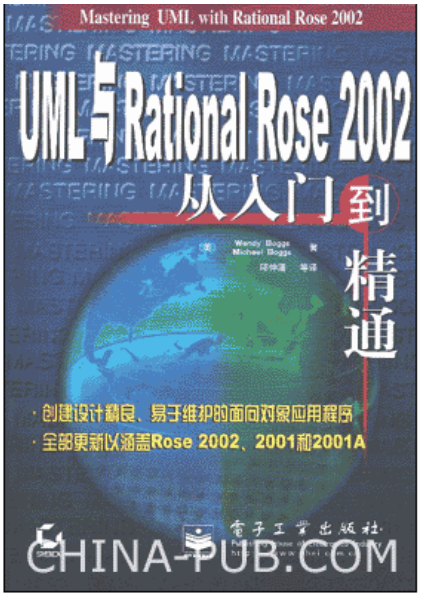
	Douglas Schmidt	机械工业	2003	★★★★★
	Grady Booch	机械工业	2003	★★★★
	Robert C. Martin	清华大学	2003	★★★★

	Rebecca Wirfs-Brock	电子工业	2003	★★★★
	Steve Adolph, Paul Bramble	清华大学	2003	★★★
	Kurt Bittner	清华大学	2003	★★★

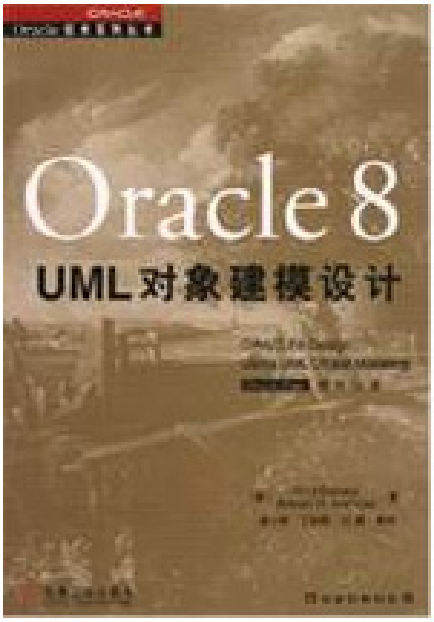
	Ian Graham	机械工业	2003	★★★
	Ivar Jacobson	机械工业	2003	★★
	Scott W.Ambler	机械工业	2003	★★★

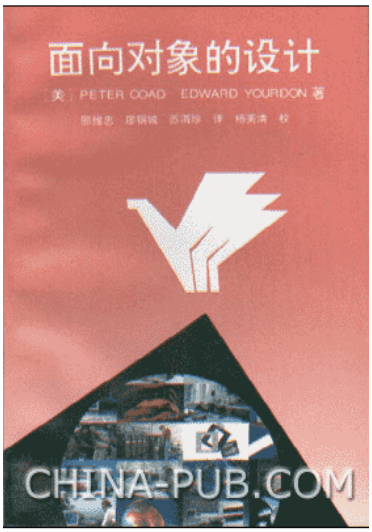
	Scott W.Ambler	机械工业	2003	★★★
	Jim Arlow,Ila Neustadt	机械工业	2003	★★★
	Ivar Jacobson	机械工业	2003	★★★★★

	Frank Buschmann	机械工业	2003	★★★★★
	Alistair Cockburn	机械工业	2002	★★★
	Martin Fowler，Kendall Scott	清华大学	2002	★★

	Dean Leffingwell ，Don Widrig	机械工业	2002	★★★★
	Joseph Schmuller	人民邮电	2002	★★
	Wendy Boggs，Michael Boggs	电子工业	2002	★★

	Craig Larman	机械工业	2002	★★★
	Ivar Jacobson，James Rumbaugh ，Grady Booch	机械工业	2002	★★★★
	Grady Booch，Ivar Jacobson，James Rumbaugh	机械工业	2001	★★★★

	Meilir Page-Jones	人民邮电	2001	★★★★★
	Paul Dorsey	机械工业	2000	★★★★
	Ivar Jacobson，James Rumbaugh ，Grady Booch	机械工业	2001	★★★★

				★★★★★
	Edward Yourdon, Carl Argila	电子工业	2000	★★★★
面向对象的分析（缺图）	Peter Coad，Edward Yourdon	北京大学	1999	★★★★
	Peter Coad，Edward Yourdon			★★★★

感谢您

的支持与信任

祝福您

快乐、 健康

WWW.UMLCHINA.COM

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

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

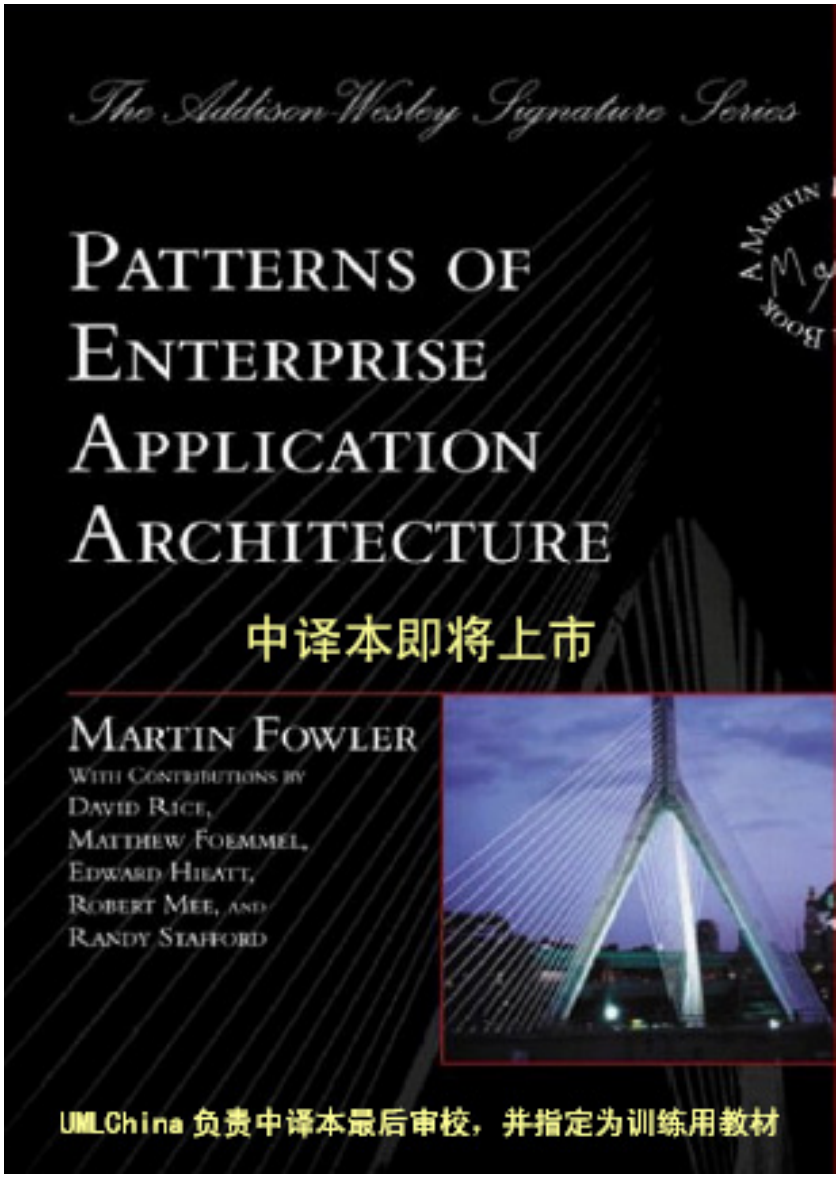
UMLChina 聚焦最后一公里，所提供服务全部与您自己的项目密切结合，帮您走完最艰难的一段路。

PEAA 中译本精选（草稿）

Martin Fowler 著，王怀民 译

查看评论

无论什么技术平台，构建企业应用都不是一件容易的工作。架构设计师和开发者需要收集行业中其他人所积累下来的经验。这些设计和实现的最佳实践通常会被表示成软件的设计模式。Martin Fowler 抓住了这许多模式的本质，并将它们写到他的 PEAA（Patterns of Enterprise Application Architecture）书中。本书的中译本即将由机械工业出版社出版。



引言

……

我的工作主要是关于企业应用的，因此，这里所谈及的模式也都是关于企业应用的。（企业应用还有一些其它的说法，如“信息系统”或更早期的“数据处理”）那么，这里的“企业应用”具体指的是什么呢？我无法给出一个精确的定义，但是我可以罗列一些个人的理解。

先举几个例子。企业应用包括工资单、患者记录、发货跟踪、成本分析、信誉评估、保险、供应链、记帐、客户服务，以及外币交易。企业应用不包括车辆加油、文字处理、电梯控制、化工厂控制器、交换机、操作系统、编译器以及电子游戏等。

企业应用一般都涉及到持久化数据。数据必须持久化是因为程序的多次运行都需要用到它们——实际上，有些数据需要持久化若干年。在此期间，操作这些数据的程序往往都被修改过。这些数据的生命周期往往比最初生成它们的那些硬件、操作系统和编译器还要长。在此期间，数据本身的结构一般也都会被扩展，使得它在不影响已有信息的基础上，还能表示更多新信息。即使是有根本性的变化发生，公司安装了一套全新的软件，这些数据也经常需要“迁移”到这些全新的应用上。

企业应用一般都涉及到大量数据。一个中等规模的系统往往都包含 1GB 以上的数据，这些数据是以数百万条记录的方式存在的。巨大的数据量导致数据的管理成为系统的主要工作。早期的系统使用的是索引文件系统，如 IBM 的 VSAM 和 ISAM。现代的系统往往采用数据库，绝大多数是关系型数据库。数据库的设计和演化已使其本身成为新的技术领域。

企业应用一般还涉及到很多人同时访问数据。对于很多系统来说，人数可能在 100 人以下，但是对于一些基于 Web 的系统，人数则会呈指数级增长。要确保这些人都能够正确地访问数据，就一定会存在这样和那样的问题。即使人数没有那么多，要确保两个人在同时操作同一数据项时不出现错误，也是存在问题的。事务管理工具可以处理这个问题，但是，它通常无法做到对应用开发者透明。

企业应用还涉及到大量操作数据的用户界面。几百个用户界面不足为奇。用户使用频率的差异很大，他们的技术背景差异也很大。因此，为了不同的使用目的，数据需要很多种表现形式。系统一般都有很多批处理过程，当专注于强调用户交互的用例时，这些批处理过程很容易被忽视。

企业应用很少独立存在，通常需要与其它企业应用集成。这些各式各样的系统是在不同时期，采用不同技术构建的，甚至连协作机制都不同：COBOL 数据文件、CORBA 系统或是消息系统。企业经常希望能用一种统一的通信技术来集成所有系统。当然，每次这样的集成工作几乎都很难真正实现，所以留下来的就是一个个风格各异的集成环境。当商业用户需要同其业务伙伴进行应用集成时，情况就更糟糕。

即使是某个企业统一了集成技术，它们还是会遇到业务过程的差异，以及数据中概念的不一致性。一个部门可能认为客户是当前签有协议的人；另外一个部门可能还要将那些以前有合同，但现在已经没有了的人计算在内。再有一个部门可能只关心产品销售而不关心服务销售。粗看起来，这些问题似乎容易解决，但是，一旦几百个记录中的每个字段都有可能存在着细微差别，问题的规模就会形成不小的挑战——就算唯一知道这些字段之间差别的员工还在公司任职（当然，也许他在你觉察到之前就早已辞职不干了）。这样，数据就必须被不停地读取、合并然后写成各种不同语法和语义的格式。

再接下来的问题是由“业务逻辑”带来的。我认为“业务逻辑”这个词儿很滑稽，因为很难再找出什么东西比“业务逻辑”更加不合逻辑。当我们构建一个操作系统时，总是尽可能地使得系统中的各种事物符合逻辑。而业务逻辑生来就是那样的，没有相当的政治力量不要想改变它，你所能做的只有照着它说的做。你必须面对很多奇怪的条件，而且这些条件相互作用的方式也非常怪异。这种情况很容易出现，比如，某个销售人员为了签下其客户几十万的一张单，可能会在商务谈判中与对方达成协议，将该项目的年终到帐时间推迟两天，因为这样才能够与该客户的帐务周期相吻合。成千上万的这类“一次性事件”最终导致了复杂的业务“逻辑”（其实不符合逻辑），也给商业软件开发带来了很大困难。在这种情况下，必须尽量将这些业务逻辑组织成有效的方式，因为我们可以确定的是，这些“逻辑”一定会随着时间不断变化。

……

企业应用的种类

在我们讨论如何设计企业应用、使用哪些设计模式之前，明确这样一个观点是非常重要的，即，企业应用是多种多样的，不同的问题将导致不同的处理方法。如果有人在这个问题上没有引起足够重视，就应当敲响警钟了。我认为，设计中最具挑战性（也是我最感兴趣）的地方就是了解有哪些候选的设计方法，以及各种不同设计方法之间的优劣比较。如何进行选择的空间很大，但我在这里只选三个方面。

考虑一个 B2C（Business to Customer）的网上零售商：人们通过浏览器浏览，通过购物车购买商品。这样一个系统必须能够应付大量的客户，因此，在做方案选择时，不但要考虑到资源利用的有效性，还要考虑到系统的可伸缩性，以便在用户规模增大时能够通过增加硬件的办法加以解决。该系统的业务逻辑也许非常简单：获取订单，进行简单的价格计算和发货计算，给出发货信息。我们希望任何人都能够访问该系统，因此用户界面可以选用通用的 Web 表现方式，以支持各种不同的浏览器。数据源包括用来存放订单的数据库，以及某个支持发货的库存系统。

再考虑一个租约合同自动处理系统。在某些方面，这样的系统比起前面介绍的 B2C 系统要简单，因为它的用户数很少（在特定时间内不会超过 100 个），但是它的业务逻辑就比较复杂。计算每个租约的月供，处理提早解约和延迟付款，合同签订时验证各种数据等，因为租约领域的竞争主要以过去的交易为基础，每一项工作都非常复杂。正是因为规则的随意性很大，才使得该领域非常复杂。

这样的系统在用户界面（UI）上也很复杂。这就要求 HTML 界面要能提供非常丰富的功能，而这些要求往往是 HTML 界面目前无法达到的，需要更复杂的胖客户界面。用户交互的复杂性还会带来事务行为的复杂性：签订租约可能要耗时数小时，这期间都应该维持事务性。数据库设计中可能也会涉及到两百多个表以及一些其它的软件包。

第三个例子是一家小型公司使用的简单“开支跟踪系统”。这个系统的用户很少，功能简单，可以通过 HTML 表现方式很容易实现，涉及的数据源表项也不多。尽管如此，开发这样的系统也不是没有挑战。一方面你必须快速地开发出它，另一方面你又必须为它以后可能的发展考虑：也许以后会为它增加赔偿检验的功能，也许它会被集成到工资系统中，也许还要增加关于税务的功能，也许要为公司的 CFO 生成汇总报表，也许会被集成到一个航空订票 Web Service 中等等。如果在这个系统的开发中，也试图使用前面两个例子中的一些架构，可能会影响开发进度。如果一个系统会带来效益，系统进度延误同样也是开销。如果现在不做决策又有可能会影响系统未来的发展。但是，如果现在就考虑了这些灵活性但是考虑不得当，额外的复杂性又可能会影响到系统的发展，进一步延误系统部署，减少系统的效益。虽然这类系统很小，但是一个企业中往往有很多这样的系统，这些系统的架构不良性累积起来，后果将会非常可观。

这三个企业应用的例子都有难点，而且难点各不相同。当然，也不可能有一个适合于三者的通用架构。选择架构时，必须很清楚地了解面临的问题，在理解的基础上再来选择设计。本书中也没有一个通用的求解办法。实际上，很多模式仅仅是一些可选方案罢了。即使你选择了某种模式，也需要进一步根据面临的问题来修改模式。在构建企业应用时，你不思考是不行的。所有书本知识只是给你提供信息，作为你做决定的基础。

……

我们面临的主要问题是，关于系统设计的决策并不是对所有性能指标的作用都相同。比如，在某个服务器上运行着两个软件系统：Swordfish 的容量是 20tps，而 Camel 的容量是 40tps。哪一个的性能更高？哪一个的可伸缩性好？仅凭这些数据，我们无法回答关于可伸缩性的问题，我们只能说 Camel 系统在单机上的效率更高。假设又增加了一台服务器后，我们发现 Swordfish 的容量是 35tps，Camel 的容量是 50tps。尽管 Camel 的容量仍然大于 Swordfish，但是后者在可伸缩性上却显得比前者更好。假设我们继续增加服务器数目后发现，Swordfish 每增加一台服务器提高 15tps，Camel 每增加一台服务器提高 10tps。在获得了这些数据后，我们才可以说，Swordfish 的水平可扩展性比 Camel 好，尽管后者在 5 个服务器以下会有更好的效率。

当构建企业应用系统时，关注硬件的可扩展性往往比关注容量或效率更重要。可扩展性可以及时满足用户需要的性能愿望，而且容易操作。有时，设计人员费了九牛二虎之力才提高了少许容量，其开销还不如多买几台服务器。换句话说，假设 Camel 的费用比 Swordfish 高，高出的部分正好可以买两台服务器，那么选择 Swordfish 可能更合算，尽管你目前只需要 40tps。现在人们经常抱怨软件对硬件的依赖性越来越大，有时为了运行某些软件就不得不对硬件进行升级，就像我一样，为了用最新版本的 Word，必须不断地更换笔记本电脑。但是总的来说，购买新硬件还是比修改旧软件来得便宜。同样，增加新的服务器也比增加程序员来得便宜——只要你的系统有足够的可伸缩性。

第 3 章 映射到关系数据库

……

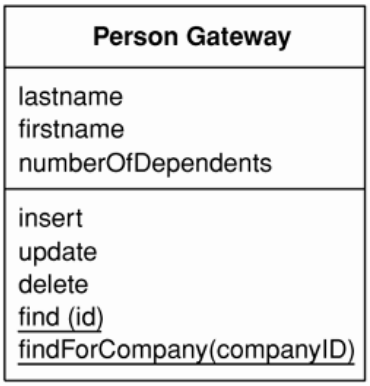


图 3.1 行数据入口（152）对于一次查询逐行返回一个实例

基于这些原因，把 SQL 访问从领域逻辑中分离出来，并把它放到独立的类中实在是明智之举。有一种方法能很好地组织这些类：让它们以数据库中的表结构为基础，这样，每一个数据表对应一个类。这些类为该数据表建立了一个入口（466）。应用程序的其他部分根本不需要了解任何与 SQL 有关的事情，而且很容易就能找到所有访问数据库的 SQL。这样对于专门进行数据库访问的开发者就十分清晰了。

使用入口（466）的方法主要有两种。最显而易见的是为查询语句返回的每一行产生一个它的实例（图 3.1）。这样，行数据入口（152）就象是用面向对象的方法来看待数据。

许多环境提供记录集（508），这是表和数据行的一种通用数据结构，用来模拟数据库的表格式属性。因为记录集（508）是一种通用数据结构，可以用在应用程序的很多地方。GUI 工具常用记录集（508）来对工作进行控制。如果使用记录集（508），对数据库中的每个表仅仅需要一个类来管理。这样，表数据入口（144）（见图 3.2）提供了查询数据库的方法，返回一个记录集（508）。

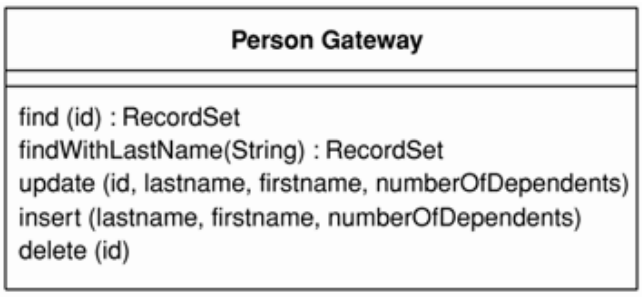


图 3.2 表数据入口（144）中，每个表都有一个实例。

即使对于简单的应用程序，我也倾向于使用入口模式，浏览一下我的 Ruby 和 Python 脚本就可以看到这一点。清晰的 SQL 和领域逻辑分离是相当有益的。

表数据入口（144）与记录集（508）非常匹配，这使得它们成为使用表模块（125）的当然选择，它也是一个组织存储过程的模式。许多设计者都喜欢通过存储过程来完成所有的数据库访问，而不是直接使用 SQL 语句。在这种情况下，可以考虑把存储过程的集合看成是为一个表定义的表数据入口（144）。可能还有一个内存中的表数据入口（144）来包装对存储过程的调用，这样就可以把对存储过程的调用封装起来。

如果使用领域模型（116），还会有更多的选择。当然可以将一个行数据入口（152）或者一个表数据入口（144）与领域模型（116）一起使用。不过，就我看来，这样会过于间接，或不够间接。

在简单应用中，领域模型（116）是一种和数据库结构相当一致的简单结构，对应每个数据库表都有一个领域类。这种领域对象的业务逻辑复杂度通常适中。在这种情况下，让每个领域对象负责数据库的存取过程是有意义的，这也就是活动记录（160）（见图 3.3）。从另一个角度来考虑活动记录（160），就是从行数据入口（152）开始，并把领域逻辑加入到类中，特别是在从多个事务脚本（110）中发现了重复代码的时候。

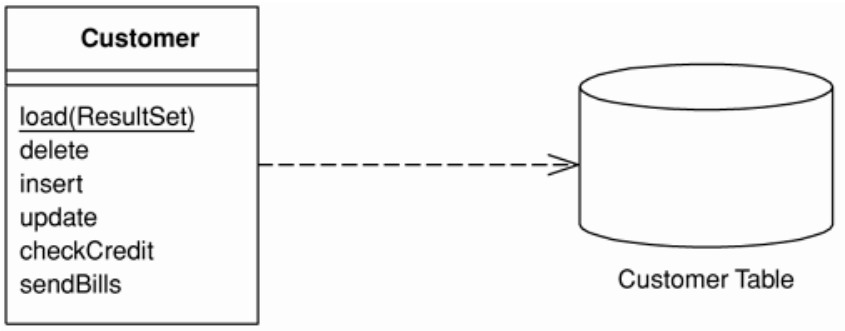


图 3.3 在活动记录（160）中，一个客户领域对象知道如何与数据库表交互。

在这种情况下入口（466）增加的间接性价值不大。随着领域逻辑变得更加复杂，它就慢慢转变成了一个大的领域模型（116），简单的活动记录（160）开始不能满足要求了。领域类和表的一对一匹配也开始随着把领域逻辑放入更小的类而失效。关系数据库无法处理继承，因此使用策略[Gang of Four]和其他轻巧的面向对象模式非常困难。随着领域逻辑的活跃，需要不访问数据库就能随时测试它。

所有这些都迫使你随着领域模型（116）的增大而采用间接的方式。在这种情况下，入口（466）可以解决一些问题，但是仍然将数据库方案和领域模型（116）耦合在一起。结果就是会有一些从入口域（466）到领域对象域的转换，这种转换会使领域对象变得复杂。

一种更好的办法是把领域模型（116）和数据库完全独立，可以让间接层完成领域对象和数据库表之间的映射。这个数据映射器（165）（见图 3.4）处理数据库和领域模型（116）之间所有的存取操作，并且允许双方都能独立变化。这是数据库映射框架中最复杂的，但它的好处把两个层完全独立了。

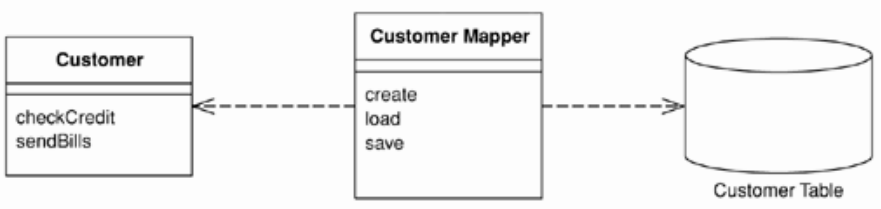


图 3.4 数据映射器（165）使领域对象和数据库彼此完全隔绝

我不推荐把入口（466）用作领域模型（116）的首选持久化机制(Primary Persistence Mechanism)。如果领域逻辑非常简单并且类和表十分一致，使用活动记录（160）就足够了。如果比较复杂，数据映射器（165）才是需要的。

这些模式并不是完全不能兼容的。在很多讨论中，我们关心的是首选持久化机制，通过它可以确定如何将某种内存数据模型保存到数据库中。因此，有必要从这些模式中选择一个，而不要把他们混合在一起，凌乱不堪。即使用数据映射器作为首选持久化机制，还可以使用数据入口（466）来封装被视为是外部接口的表或者服务。

……

第 4 章：Web 表现层

……

在最近几年中，对企业级应用来说，最大的变化莫过于基于 Web 浏览器用户界面的兴起。它带来如下许多好处：不需要安装客户端软件，通用的 UI 和简易统一的接入方法。同样，在许多环境下很容易建立 Web 应用。

……

首先，也是最重要的，使用模型-视图-控制器（330）的原因是要保证模型和 Web 表现层的完全分离。这使得修改表现层，以及加入其它的表现层都会很容易。同样，把处理过程加入到分离的事务脚本（Transaction Script）（110）或分离的领域模型（Domain Model）（116）对象中和对他们进行测试同样也会变得很容易。如果你正在用服务器页面作为视图，这一点尤其重要。

在这里我们来看看“控制器”的第二种用法。许多用户界面设计通过应用控制器（Application Controller）（379）对象的中间过渡层来分离表现层对象和领域对象。应用控制器（379）的目的是处理应用程序流，决定把哪个视图呈现给哪个命令。它可能表现为表现层的一部分，或者，也可以把它当成一个在表现层和领域层之间作为中介的隔离层。应用控制器（379）独立于任何特定的表示层，在这种情况下它们能在表示层之间重复使用。虽然给每个不同的表现层一个不同的流是一种很好的方法，但是如果不同的表现层伴随着相同的基本流和导航，则上述这种方式可以工作得很好。

并不是所有的系统都需要应用控制器（379）。如果你的系统有许多在不同屏幕次序和导航之间的逻辑关系，那么应用控制器（379）是非常有用的。如果你的网页与领域上的行为没有映射的话，同样这个应用控制器（379）也是很有用的。但是，如果屏幕之间的顺序随意性很大的话，你也许就不需要它了。可以进行这样的测试：如果机器处在屏幕流的控制下，那么你就需要应用控制器（379）；如果这台机器是在用户的控制下，就不需要应用控制器（379）。

视图模式

在视图方面有三种模式：转换视图（*Transform View*）（361），模板视图（*Template View*）（350）和两步视图（*Two Step View*）（365）。这就带来了两种选择：第一，使用*转换视图（361）*还是*模板视图（350）*，第二，无论使用哪一种，在使用方式上是在一个步骤内完成，还是使用*两步视图（365）*。*转换视图（361）*和*模板视图（350）*的基本模式是在一个步骤内完成，但也可以把两步视图（365）应用于它们其中任何一个。

首先，在*转换视图（361）*和*模板视图（350）*之间如何进行选择。*模板视图（350）*允许你在网页的结构中编写表现层，并允许在网页中嵌入标签，用以指明网页中的动态内容需要导向到哪里。很多流行的平台都基于这种模式，这些平台中的许多都是应用服务器页面技术（如：ASP，JSP，PHP），它们允许你在网页中输入程序语言代码。这种方式提供了强大的功能和灵活性；但不幸的是，这也将导致代码的混乱以至于很难被维护。一种解决办法是，在使用服务器页面技术时，必须非常小心而不致于使程序的逻辑脱离网页的结构（通常使用一个辅助对象）。

*转换视图（361）*使用程序的一种变换风格。常见的例子如 XSLT。如果领域数据是以 XML 格式存在的，或者很容易转换成这种格式，那么使用转换视图（361）将是非常有效的。可以通过一个输入控制器来控制对 XSLT 样式表单的选择，然后把它作用在由模型返回的领域数据上。

如果使用程序脚本作为视图，首先，必须在转换视图（361）、模板视图（350）或者两者的结合方式中编写代码。大部分脚本选择这两种模式中的一种作为主要框架。

其次，要决定是选择在一个步骤内完成还是选择*两步视图（365）*。第一种方式通常在应用程序中为每一个屏幕都准备一个视图组件。视图提取领域数据并把它返回到 HTML 网页中。这里，“通常”是指类似的逻辑屏幕还是可以共享同一个视图的。虽然如此，大多数情况下，还是可以把它想象成一个“一个屏幕对应一个视图”。

两步视图（365）（图 4.3）把这一过程分解成了两个步骤：由领域数据产生一个逻辑屏幕，然后把它发送到 HTML 网页中。其中，每一个屏幕都有一个第一步骤的视图，而整个程序中只有一个第二步骤的视图。

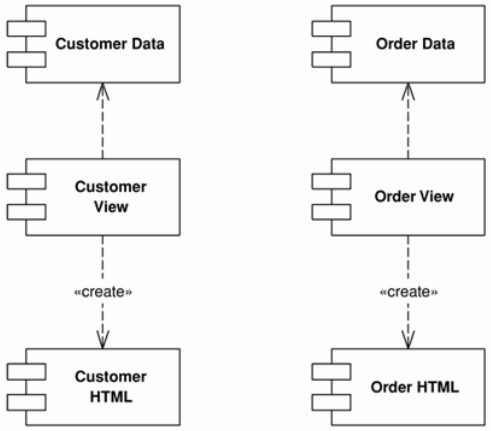


图 4.2 单阶视图

*两步视图（365）*的优点是可以把 HTML 网页放在单个地方。这使得全局改变 HTML 网页变得很容易，因为，如果要改变每一个屏幕的内容的话，只需对一个目标对象作修改。当然，如果逻辑表现层是一样的，它仅仅能带给你这些好处，因此，当不同位置的屏幕使用相同的基本设计时，它可以很好地工作。所以，当站点设计得过分精细时，通常不容易提取出很好的逻辑屏幕结构。

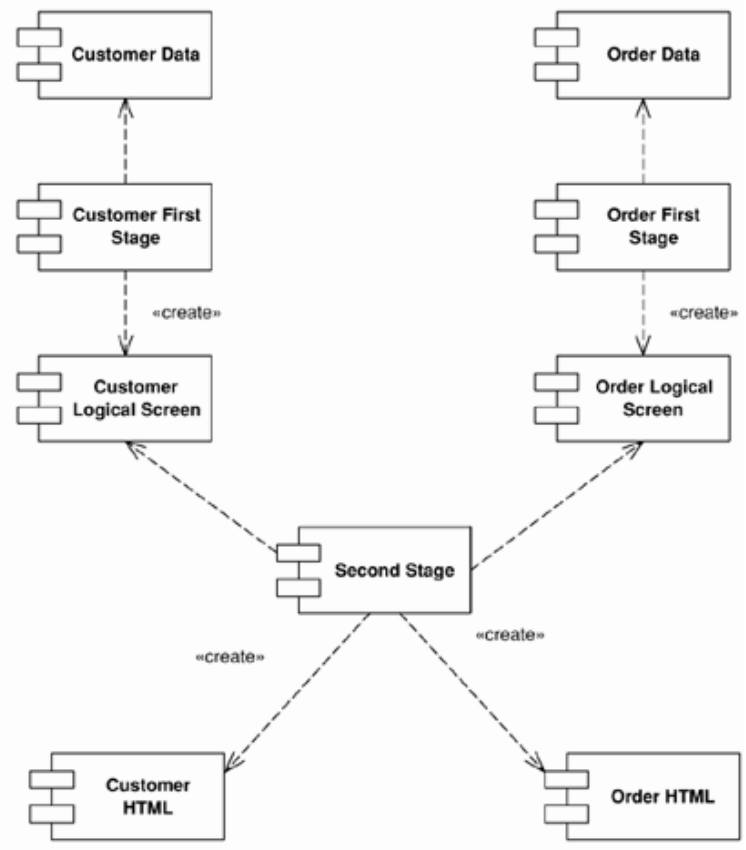


图 4.3 两阶视图

如果 Web 应用程序提供的服务有多种前端用户，那么运用两步视图（365）将更加得心应手。例如，多家航空公司后台使用的是相同的基本订票系统。在逻辑屏幕的限制下，不同的前端用户由于第二个步骤不同，可以得到不同的视图。类似的，还可以使用两步视图（365）来处理不同的输出设备，例如，可以在服务的第二步骤区分 Web 浏览器或掌上电脑。同样，共享公共的逻辑屏幕也是受限的，当两个用户界面差异非常大时，比如在浏览器界面和手机屏幕之间，这种共享是不可能的。

第 12 章 对象关系结构模式

……

创建一个对象需要一个键。这个问题听上去十分简单，实际上却会带来很多麻烦。有三种选择，数据库自动生成、使用 GUID 或者自己创造一个。

自动生成策略是最简单的。每次你要往数据库里插入数据，数据库会产生一个唯一的主键而不需要你做任何事情。然而事实上却没有容易。并不是所有的数据库都使用同样的方法。这样会在对象关系映射时出现问题。

最通用的自动生成方法是声明一个自动生成域，无论什么时候插入一行数据这个域都会增加一个新值。这种策略的问题在于难以确定为键生成的是什么值。如果要插入一份包含多条子项的订单，那么就需要新订单的键值，这样才能把它作为订单项的外键。当然在事务提交之前也需要知道这个键值，这样才能保存事务中的所有记录。可惜数据库通常并不提供这一信息，因此，对于需要插入关联对象的任何表不能使用这种自动生成方法。

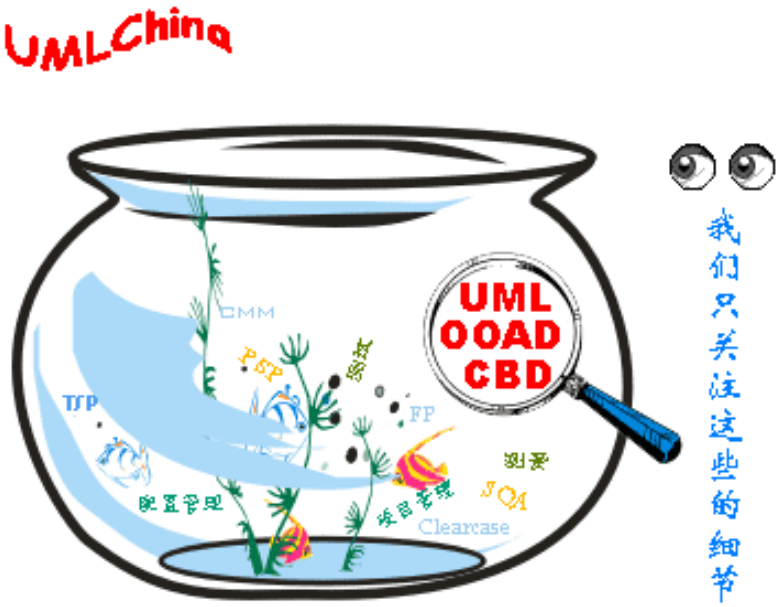
还有一种办法可以做到自动生成，这就是数据库计数器。Oracle 的序列（sequence）就使用数据库计数器。发送一个引用序列的 Select 语句，数据库就会返回包含下一个序列值的 SQL 记录集。你可以让一个序列以任何整数来增加，它允许一次设置多个键。序列查询是在一个独立的事务中自动进行的，这样访问序列时不会锁住那些想要对同一个表执行插入操作的事务。对于我们的需求来说，数据库计数器非常合适，但它没有统一的标准，并且也不是所有的数据库都支持。

GUID（全局唯一标识符）是指在一台机器上生成的数字，它保证对在同一时空中的所有机器都是唯一的。通常平台会提供生成 GUID 的 API。生成算法很有意思，用到了以太网卡地址、纳秒级时间、芯片 ID 码和许多可能的数字。这样得到的结果数字完全唯一，因此是安全的键。GUID 的唯一缺陷在于生成的结果串会比较大，这可能会成为一个大问题。经常，当某人需要在窗口中或 SQL 表达式中输入一个键的时候，长键总是既难输入也难读取的。这也会带来性能问题，尤其是对于索引。

最后一个选择是自己来产生键值。对小系统的一个简单例子就是使用 SQL 的 max 函数来进行表扫描，这个函数会找到表中的最大键并用它增加一项。可惜，这样做会用读锁锁住整个表，这意味着如果插入比较少的话效率比较高，但如果在更新表的同时进行插入操作效率就十分成问题。必须保证在事务之间完全独立，否则会造成多个事务得到同一个 ID 值。

一个更好的办法是使用独立的键表。典型的键表有两列：名字和下一个有效值。如果使用数据库唯一键，在这个表里面就只有一行数据与之对应。如果使用表唯一键，那么对于数据库中每一个表都有一行数据与之对应。使用键表的方法就是读出数据行，获得数字，进行增加操作，得到新数字并把这个数字写回到数据行中。在更新键表时，可以通过增加一个适当的数值来一次获取多个键。这样做减少了昂贵的数据库调用，也减少了对键表的争夺。

（本草稿经该书出版社机械工业出版社同意刊登）



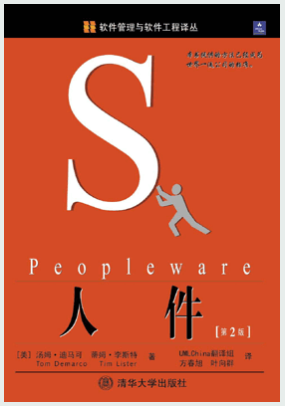
PH PTR

人件集

——人性化的软件开发

The Peopleware Papers

NOTES ON THE HUMAN SIDE OF SOFTWARE



续篇

Larry L. Constantine 著

谢 超 刘 颖 谢卓凡 李 虎 译

人民邮电出版社

需求工程师的素质要求

Think 整理

查看评论

Pixar公司制作的每一部电影票房都是轻松过亿，今年的全美票房冠军《海底总动员》更是达到了创记录的3.9亿，而且Pixar还凭借这几部电影捧回了13座奥斯卡奖。很多人认为Pixar靠的是技术，看他们自己怎么说：

“在Pixar，我们的共识是，故事永远是最重要的。”

“如果没有好的故事，即使有再高超的技术，也不可能做出好的电影。”

“Pixar从不用任何一部旧故事做剧本。”

软件开发就象拍电影，如果需求阶段出了问题，后面再好的技术也没有用。

公司需要需求工程师

这个图在很多书上都出现过：

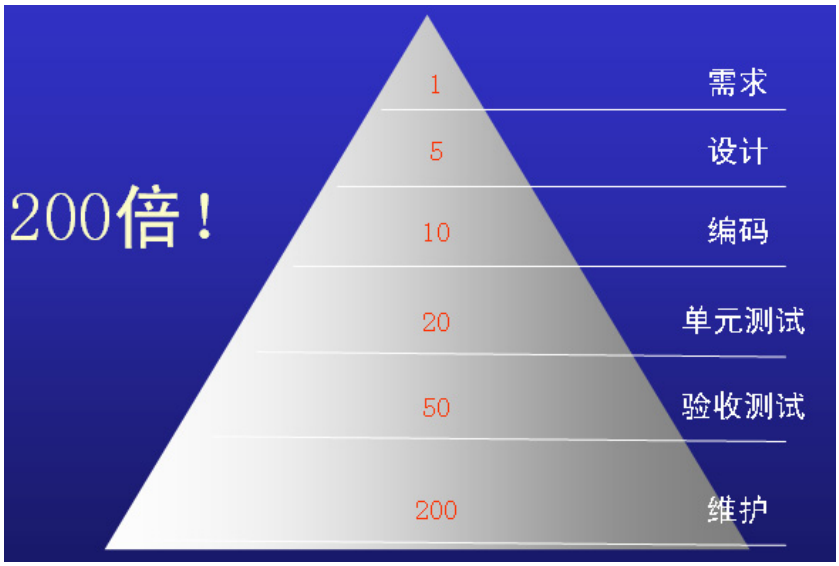


图1 需求环节的重要性

图中意思是软件中的问题，如果在需求阶段解决的代价是1，拖到维护阶段再解决的代价就是200。由此看来，需求是软件公司团队最值得改进的技术环节。随着软件业的逐渐发展，“能写代码”、“能使系统运行起来”这些以前的标准已经远远不能满足现在客户的需要，客户关心的是，“你的系统能为我带来什么价值？”这样一来，需求技术在软件公司日益受到重视。

目前，很多公司里，担任需求工程师角色的开发人员，同时也会承担着设计和编码的任务。实际上，这不是一种好的做法。需求工程师的任务是出题，需要把各种碎片捏合成题目（或者说，用例），需要强的综合能力；设计员程序员的任务是解题，需要强的分解能力。这两种能力方向是相反的，这也是程序员经常会“误用”用例的原因。一个好的程序员并不能自动成为好的需求工程师。

需求工程师的素质要求

需求工程师需要什么样的素质呢?《软件需求》一书的作者Karl E. Wiegers在文章“So You Want To Be a Requirements Analyst?”[1]中列出了需求工程师的任务和素质要求，简要摘录如下。

Karl E. Wiegers认为，需求工程师需要和系统的各种涉众沟通，成为他们之间的纽带。所以，需求工程师需要具备以下素质:

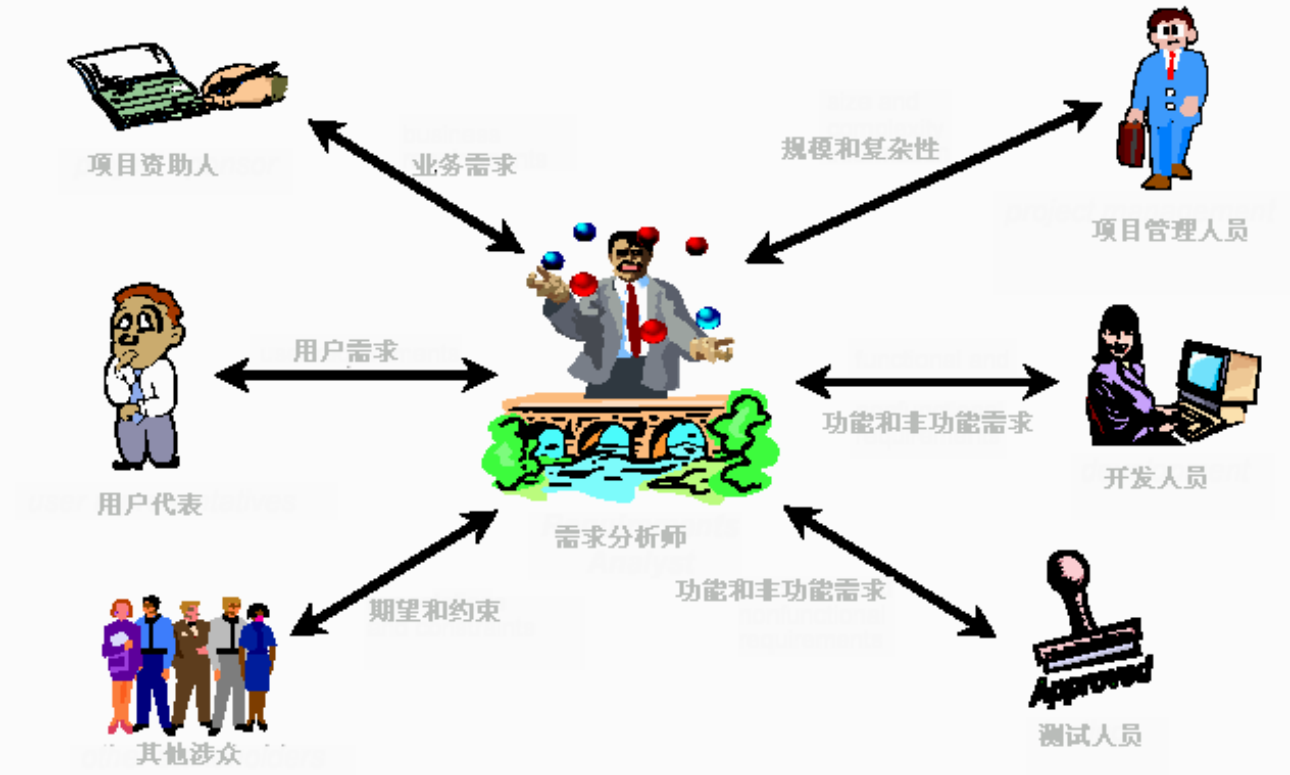


图2 需求分析师负责沟通各种人

关于需求工程师的素质要求，Karl E. Wiegers列出了以下几点:

1. 倾听的能力。积极倾听包括排除干扰，保持专心的姿态和眼神接触，并重复要点来确认你已经理解。你需要抓住要点并从字里行间推断出对方犹犹豫豫没有说出来的东西。了解对方喜欢什么样的沟通方式，尽量避免把你的个人理解强加到客户身上。

2. 访问能力。大多数需求输入来自讨论，因此一名分析员必须能够询问正确的问题。例如，用户一般会把精力集中在正常的、符合期望的行为上，而每个程序员都知道需要大量代码来处理错误，可以问类似这样的问题“如果这样，会发生什么事情”或“会出现这样的问题吗？”Donald Gause和Gerald Weinberg在书中[2]描述了“context-free questions”的技术。

3. 分析能力。有在不同层次进行抽象的能力。有时你需要从高层往下钻到细节，有时需要从某个用户的特定需要泛化到适用于整个用户类别的需求集合。评价各种从不同来源收集到的素材，调和冲突，从用户的需要中分离出真正想要的，并把解决方案从需求分离。

4. 协调能力。涉众需要需求工程师作为中立的协调员，协调员需要强的提问和观察技巧，以帮助团体建立信任、改善业务人员和技术人员之间有时会出现的紧张关系。Ellen Gottesdiener的书[3]中提供了协调员的建议宝库。

5. 观察能力。在持续观察用户做工作或使用一个应用系统的时候，你可以推断到一些微妙的东西，这些东西可能在讨论的时候没有涉及。这样就揭示了新的需求领域。

6. 书写能力。你为顾客、营销人员、经理的主要交付物是书写好的规格说明。为了完成这个任务，你需要有坚实的(自然)语言能力。力争做到清楚，避免含糊词语和语法错误。

7. 组织能力。你会面对一大堆在启发和分析中得到的混乱信息，快速地把它们组织起来，把碎片拼成相关的整体。这需要高超的组织技巧，还有耐心和韧性。

8. 建模能力。从流程图到结构化分析模型(数据流图、实体关系图等)到UML标记，需求分析师的工具箱里应该有这些画图工具。在和用户交流的时候，一些这样的技巧是非常有用的，和开发人员的交流也一样。你会经常需要教育其他涉众这些技能的价值以及如何阅读它们。

9. 交际能力。你必须能够使有不同兴趣的人一起工作。你应能自如地和组织中不同级别的个体交谈不同的工作，你可能还需要和分布的团队交流，它的成员分布在不同的地点，时区、文化、语言都有区别。

10. 创新能力。分析师不只是一名抄写员，顾客说什么他就记什么。James Robertson[4]建议最好的分析师应提议需求，分析师可以构思创新的产品功能，设想新的市场和业务机会，想出能够使顾客震惊和高兴的好方法。一名真正有价值的分析师能发现创新方法来满足用户一直不知道自己已有的需要。

11. 领域知识。事先拥有一定的领域知识，才能使以上的能力发挥出来。

对于需求工程师，并没有标准的教育程序和工作叙述，他们大多数来自不同的背景。如果你从类似于系统用户这样的角色进入到需求分析角色，你需要在你丰富的业务知识和软件工程工作环境之间作出平衡，并学会和你的技术同事们沟通。如果你是从开发人员转换而来，你需要增强对业务的理解，随时提防滑进自己的技术陷阱中。

需求获取技术

Karl E. Wiegers还列出了一些搜集需求数据的方法：

- 访谈
- 需求工作组
- 文档分析
- 问卷调查
- 现场观察
- 业务流程分析
- 工作流和任务分析
- 列出外部事件以及相应的系统响应
- 竞争对手产品分析
- 逆向工程现有系统
- 回溯以前的项目

参考文献

[1] So You Want To Be a Requirements Analyst?, Software Development, July 2003

[2] Exploring Requirements: Quality Before Design, Dorset House, 1989

[3] Requirements by Collaboration: Workshops for Defining Needs, Addison-Wesley, 2002

[4] Eureka! Why Analysts Should Invent Requirements, IEEE Software, July/Aug. 2002