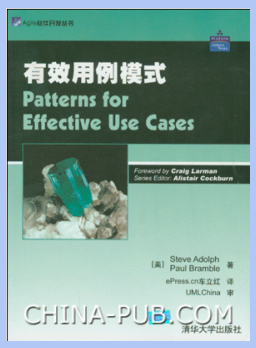
第 29 期（2003 年 9 月）

【新闻】



教材

【方法】

【人件】

X 非程序员
X-Programmer
软 件 以 用 为 本

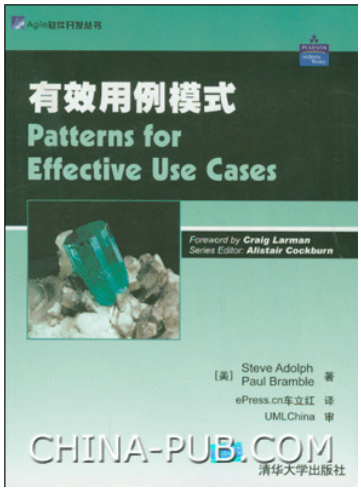
投稿：editor@umlchina.com
反馈：think@umlchina.com

http://www.umlchina.com/

《有效用例模式》中译本发行

[2003/8/20]

日前，《有效用例模式》中译本由清华大学出版社出版。



用例的概念在 1986 年由 Ivar Jacobson 正式提出之后被广泛接受，迅速发展。现在，“用例驱动”已经成为统一过程的基本原则。

Alistair Cockburn 等人在探索用例的本质和如何使用用例上，作出了大量的努力，使用例变得更加朴素、更加实用。本书可以看作是 Alistair Cockburn 的“Writing Effective Use Cases”的续集，它把用例开发过程中的要点组织成了 36 个模式，使读者更加容易理解。

UMLChina（http://www.umlchina.com）已经决定把本书作为 UMLChina 训练的指定教材。UMLChina 从 1999 年开始就已经在中国推广用例技术，迄今为止，实践和指导实践用例技术的项目已近 30 个，我们深深感到用例技术带来的强烈变化，正如面向对象为分析设计所带来的，用例也是软件开发的一种“返璞归真”。

《人月神话》中译本修订版即将出版

[2003/9/2]

《人月神话》中译本自 2002 年 11 月出版至今，书中的许多概念被越来越多的中国读者接受、引用。经历数次重印后，清华大学出版社决定在新一轮重印之前，结合读者的反馈，对《人月神话》中译本进行修订。本次修订版涉及的内容有：

- 修正了翻译上的一些瑕疵
- 增加了著名人士对该书的赞誉
- 增加了国内读者的评论

《人月神话》中译本最新修订版将于 2003 年 9 月上市。本书还将会根据读者的反馈，不断修订，为中国的读者发挥其最大的价值。

基于模型的开发领域，微软从零开始

[2003/7/24]

微软主席和首席架构师 Bill Gates 声称，微软将深入基于模型编程（model-based programming）这一领域。

本周三，Gates 在微软财政分析人员会议上发表公开演讲，将基于模型开发作为公司发展的目标之一，并认为在 2005 年 Longhorn 操作系统以及之前中间产品的开发中，建模技术将会发挥重要的作用。

Gates 认为建模的过程是使用“很高层次的规约来描述软件是如何形成的……最后我们看到的是源代码这种字面的形式”， Gates 认为建模实际上是“创建代码”过程中不可分离的一部分。

在 Visio 绘图和建模工具之前，微软已经为建模技术提供了广泛的支持。

实际上，微软曾经和对象管理集团 (OMG) 在 UML 领域有过合作，现在，微软又在努力地支持模型驱动架构(MDA)。

有消息称，在 Rational 被 IBM 集团收购之前，微软也非常感兴趣。消息人士称，微软的兴趣在于 Rational 的建模技术。现在 IBM 正在其工具集如 WebSphere 应用服务器、DB2 数据库中加入 Rational 的建模技术。

消息人士称，微软希望和 OMG 恢复良好的合作关系（请看 Microsoft 希望与 OMG 重修旧好），因为公司目前在研究建模和架构相关的领域有更多的投入。今年，Microsoft 已经赞助了两个为期四天的 OMG Web services workshops ，一个在德国的 Munich，另一个在上月于 Philadelphia 召开，会上 Microsoft 的代表做了关于 面向服务架构（services-oriented architectures）和 MDA 的讲演。Unisys 是 OMG 成员之一并且和 Microsoft 有着紧密的合作伙伴关系，他们和 Microsoft 举行了联合报告，题目是“Microsoft 的 'Jupiter' 和 Unisys 的 MDA 过程”。

现在，Microsoft 已经在其 Visual Studio .Net 企业架构版本中支持 UML，并且有消息称，Microsoft 计划在 Jupiter 中支持 MDA ，Jupiter 是其即将问世的电子商务软件（e-business suite and collaboration portal software）的代号。.

另外一个和 Microsoft 关系密切的消息来源称，由于企业系统的日益复杂，该公司对模型架构的兴趣日益浓厚。

（自 eweek，袁峰 摘译，不得转载用于商业用途）

Rational 为 IBM 的按需开发添上双翼

[2003/8/25]

Rational 公司正在准备应用开发和管理方面的四个提案（initiative），在未来 18 个月中，它们将给 IBM 公司的按需策略给予强大的支持。

Rational 总管 Mike Devlin 今天将向 ComputerWire 透漏在建模、企业变更管理、设计质量保证（quality-by-design）以及最佳实践上的策略，正是这些将 Rational 与 IBM 的中间件及平台紧密结合在一起。

Devlin 将在奥兰多召开的公司用户会议上概述 Rational 的目标，他演讲的题目是“按需世界中的软件开发平台”。

市场管理部负责人 Roger Oberg 告诉 ComputerWire，在 IBM 的按需业务中，Rational 扮演着非常重要的角色。为了快速响应业务条件的变化，组织们必须依赖于他们的应用架构。

Oberg 说，“我们正在启动一个提案，使得软件架构响应性更强、更加健壮和可靠。”他还补充说，本周 Rational 将不会发布任何产品，但他估计这些提案将会在未来 18 个月中导致大量产品的诞生。

IBM 收购 Rational 的一个目的就是为了实现 WebSphere、DB2、Tivoli、Lotus 以及 Rational 工具套件之间的集成。

Rational 将宣告其重用资产规约 RAS（Reusable Asset Specification ）)已经被国际对象管理集团 OMG 接受进行评审。 Rational 在 2000 年 6 月宣布了 RAS 的诞生，它是一种在 UML 中标识并捕获可重用的软件组件，以在其它项目中进行复用的一种方法。

Rational 曾试图说服 OMG 倒退三年，将该规约纳入到计划中的下一版 UML 中，显然，这个愿望破灭了，但 Oberg 认为 OMG 决定对 RAS 进行评审已经为这个规约大大增加了分量。

按计划，将由 Devlin 来发布这些 OMG 新闻， Devlin 可能会对 Rational 在涉及建模、企业变更管理、设计质量保证以及最佳实践的这些提案中的内容给出一个概述。

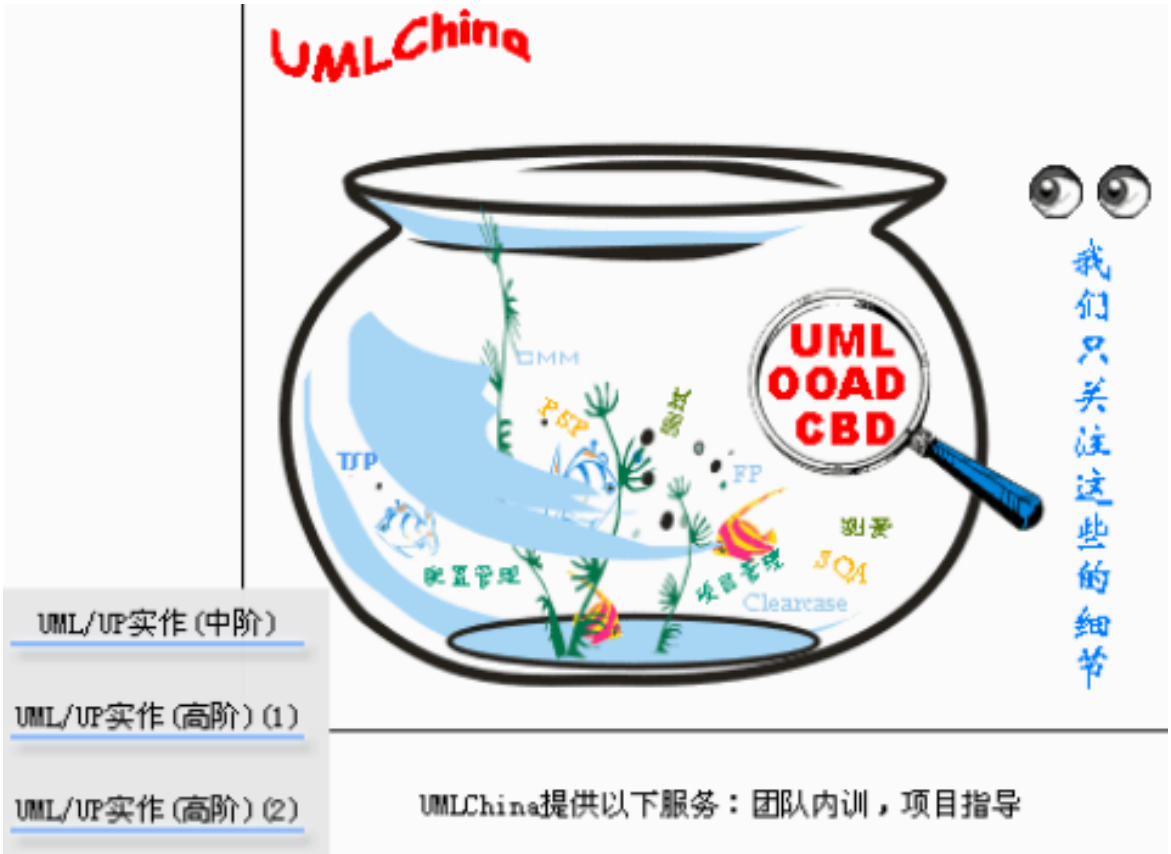
在建模领域， Rational 的努力方向是：统一 IBM'的业务建模软件 WebSphere Business Integrator 和 Rational 的 portfolio。这样，开发人员可以在 WebSphere 中进行建模，在 Rational 的工具中直接对前面建模的结果进行操作。在长期方向上， Rational'的第二个建模策略是增强抽象层次，减少需要手工编码的数量。

和建模领域一样，在企业变更管理上，Rational 同样有两个策略。首先，它的 ClearCase、CearQuest 和需求管理工具将对各种来源，例如 WebSphere Portal，的数据使用统一的方式进行集成表示。目前，这些工具的用户接口和数据仓库都各不相同。

企业变更管理上的第二个策略是通过 ClearCase 和服务器使用的宿主机平台的集成提高 Rational 和 IBM 的集成层次。Oberg 认为，集成后，用户不需要知道建模工具知道一个建模工具到底是部署在 Windows、Linux 还是 Unix 上。

他说，“你可以创建并部署到任意平台上”。“设计质量保证”试图扩展 Rational'在软件设计和创建阶段对结构化和静态测试分析（static testing analysis）的应用，以减少开发流程上后期的测试需求。最后，这些将补充到 Rational 的最佳实践指导原则中去。

（自 cbronline，袁峰 摘译，不得转载用于商业用途）



Rational 为由设计直接生成代码努力

[2003/8/27]

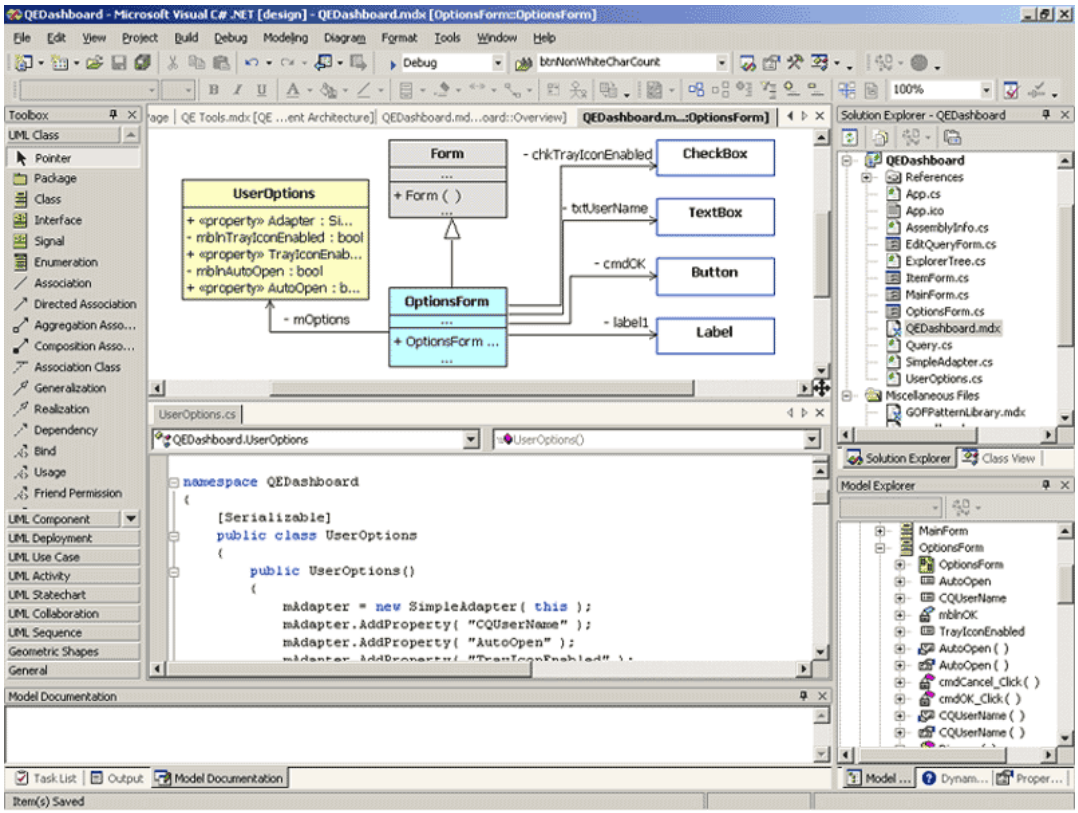
在软件工程领域，从形式化规约与设计自动生成软件，这种想法已经有些年头了。 刚刚被 IBM 公司收购的 Rational 软件公司一直在为这个理想而努力。

Rational 开发了一个完整的产品家族来使模型驱动的开发成为可能，在模型驱动的开发过程中，软件设计主要关注的是业务模型，而不是底层的实现技术。

“大多数业务软件的失败都是因为开发人员开发的应用不能很好地实现业务价值，这就是现实”，在 Putra 世贸中心举行的微软开发者技术大会上，IBM 公司的技术顾问 Martin Lee 如是说。

“开发过程中，现在称为是业务建模的过程中包括分析业务需求，分析业务的瓶颈和待改进的地方，从中得到系统需求。”

“我们的可视化建模工具 Rational XDE，允许用户使用 UML 的各种图可视化地捕捉这些需求。这种方法使得用户可以为整个系统建立一个比较清晰的视图。” 他说。



UML 是一种用来规范和可视化描述复杂的软件系统的通用的标识语言，Rational 公司是其最大的支持者之一。

Lee 解释了 UML 的可视化模型和 Rational 的自动代码生成技术：“我们通过内置的引擎将平台无关的 UML 模型转换成为平台相关的实现。诸如 XDE 的工具允许用户创建模式，后者之中包含了应用的语义”。

“Rational XDE 中的模式技术包括模式积累（pattern harvesting）,用户可以使用过去系统代码中的模式自动生成未来产品中的代码。”

他认为，“这将潜在地缩短开发时间、加速开发过程”。

Randy Tan, IBM 的另一位技术顾问，补充道，“你在模式中表达的信息越多，我们工具为你生成的代码质量就越好，类似地，模式越多，生成的代码也越完整。”

Tan 声称，Rational 公司坚信，在软件开发中寻找各种方法减少重复和多余的工作，是非常重要的。

“IBM 和 Rational 是都是开发资产重用规约的社团成员。很多描述电子商务系统可重用资产的概念、标识和指南都正在开发之中。我们正致力于基于模式来创建描述可重用资产的标准格式。”

无论如何，可重用资产的规约肯定会在很大程度上依赖于 UML，正像 Rational 公司目前提供的各种工具一样。

“UML 是设计软件的标准，我们认为我们提供的工具和过程集成了业界的各种最佳实践，我们总是鼓励用户使用正确的开发过程”Tan 说。

除了自动代码生成，模型驱动开发的另一个重要的优势在于，通过运行模型的仿真实现，开发过程中的测试将变得非常容易。

他认为，理想情况下，这将导致开发和测试成为并行的两个过程。

（自 star-techcentral，袁峰 摘译，不得转载用于商业用途）

最新新闻：Borland 发布 Together Edition for Microsoft Visual Studio .Net

点击进入>>

UMLChina 公开课

“UML 应用实作细节”

（2003 年 9 月 27-28 日，上海）

现在，UML、RUP、Rose 等概念已经传播得越来越广，书籍、资料也越来越多，决定在开发过程中使用 UML 的开发团队也越来越多。

在开发团队应用 UML 的软件开发过程中，自然会碰到很多细节问题：“我这样识别 actor 和用例对不对？”、“用例文档这样写合适吗？”、“RUP 告诉我该出分析类了，可类怎么得出来啊？”、“先有类，还是先有顺序图啊？”、“类怎样才能和数据库连起来啊？”...许许多多的细节问题，而每一个细节都和背后的原理有关。

本课程秉行 UMLChina 一贯的“只关心细节”的原则，内容完全是由 UMLChina 自行设计，围绕一个案例，阐述如何（只）使用 UML 里的三个关键要素：用例、类、顺序图来完成软件开发。使学员自然领会 OOAD/UML 的思想和技术，并对实践中的误区一一指正。整个过程简单实用，简单到甚至可以只有一个文档，非常适合中小团队。

详情请见>>

UML 相关工具一览：S-T（截止至 2003 年 9 月）

整理

查看评论

接 28 期...

工具（最新版本）	厂商	试用允许	UML 支持								支持代码环境	XMI	平台	备注
			用例图	类图	状态图	活动图	顺序图	协作图	构件图	部署图				
Select Component Architect	Select Business Solutions http://www.selectbs.com/products/products/select_component_architect.htm		√	√	√	√	√	√	√	√	IDL, C++, Java, C#, Delphi, Forte, Oracle-DDL, SQL, SQL Server-DDL, VB, Peoplesoft	√	Windows	可以和 ERWin, Caliber-RM协同，Rose输入输出。
SILVERRUN ModelSphere 2.0 Build 223	magna solutions http://www.silverrun.com/modelsphere2_0.html	有 Demo 版		√							Java	√	Java	支持过程建模、数据建模、UML 类建模。
Simply Objects	Adaptive Arts（澳大利亚） http://www.adaptive-arts.com/prod_so.	有试用版	√	√			√	√			C++, C#, Delphi, Smalltalk, Java,	√	Windows	

3.4	htm Adaptive Arts Model Your World™										VB, Eiffel, OO COBOL, Bullant			
SiSy Developer 2.16	SiSy（德国） http://www.sisy.de/ SiSy®		√	√	√	√	√	√	√	√	C/C++, Delphi, Pascal, Java, C#, 汇编			
SmartDraw	SmartDraw http://www.smartdraw.com/ smartdraw.com draw anything easily	30 天试用	√	√	√	√	√	√		√			Windows	
SmartState 3.1	ApeSoft（印度） http://www.apesoft.net/ss/ APESOFT	有试用版			√						C/C++, Java, XML, 汇编		Windows	实现状态设计模式
SoftModeler 3.6	Softera（以色列） http://www.softera.com/products.htm Softera Next Generation Java™ and XML Tools	30 天试用	√	√	√		√				Java		Java	专门针对 Java、EJB 的 UML 工具。
Software through Pictures(Stp/UML) 8.3	Aonix http://www.aonix.com/	有试用版	√	√	√	√	√	√	√	√	IDL, C/C++, Java, Ada95, COM, EJB		Windows	擅长在异种开发环境下工作。

	Aonix													
System Architect v9.1	Popkin Software http://www.popkin.com/products/system_architect.htm POPKIN SOFTWARE	15 天试用	√	√	√	√	√	√	√	√	IDL, C/C++, Java, C#, Delphi, HTML, PowerBuilder, Smalltalk, VB		Windows	能够把数据模型转成类模型，支持 Zachman 和 C41SR 架构框架。
Tau 4.4	Telelogic AB（瑞典） http://www.telelogic.com/ Telelogic Automating the Development Lifecycle!		√	√	√	√	√	√	√	√	CORBA IDL, C/C++, Java, COM+ IDL, EJB, Ada, Delphi, PB, Forte. NewAge, Eiffel	√	HP-UX, Solaris, Windows	和 DOORS 结合，高效构造软件。
Tau Generation 2	Telelogic AB（瑞典） http://www.telelogic.com/ Telelogic Automating the Development Lifecycle!		√	√	√	√	√	√	√	√	CORBA IDL, C/C++, Java, COM+ IDL, EJB, Ada, Delphi, PB, Forte. NewAge, Eiffel	√	HP-UX, Solaris, Windows	支持 UML2.0，支持 MDA。
Tengger	IBM http://www.alphaworks.ibm.com/tech/tengger IBM	90 天试用									Java	√		读取任何一种工具的 XMI，产生 Java 代码。

Together	Borland http://www.borland.com/together/index.html 		√	√	√	√	√	√	√	√	CORBA IDL, C++, Java, C#, COM IDL, EJB, Visual Studio .NET, SAP, WebSphere	√	Java	强有力的工具
Toolkit for Conceptual Modeling (TCM) 2.20	Twente 大学（荷兰） http://wwwhome.cs.utwente.nl/~tcm/ 	开源	√	√	√	√		√	√	√			Linux, Windows	
Trice	Protos（德国） http://www.protos.de/Protos/protos_E.html 				√		√				CORBA IDL, C++, C, COM, Microcontroller(C166, 8051,…)		Windows	遵循 ROOM 的嵌入式和实时系统开发
待续…														

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

重构：当前研究及未来趋势

Tom Mens 著，wnb 译

查看评论

摘要

本文首先从理论和实践两个方面，对软件重建及重构领域的研究情况进行了详细的综述。列举了未来研究方向的问题，并提出了解决某些问题的方法。

1 介绍

现实环境中软件的一个本质属性是需要不断地对其进行改善。在对软件进行改进、修改以适应新的环境时，代码会变得越来越复杂，与其最初的设计相比，发生了很大的变化。因此，软件维护开销逐渐成为整个软件开发中主要的开销[26，49，61]。当前的软件开发方法和工具没有解决这一问题，因为它们主要关注在同一时间框架下实现新的需求[44]，这使得软件变得更加复杂。为了解决这一不断发展的复杂性问题，迫切需要一种技术，通过增量式地改进软件内部结构，减少软件的复杂性。解决该问题的研究领域称为重建，在面向对象软件开发领域中称为重构。

按照 Chikofsky 的工程分类，重建定义如下：重建是在同样的抽象层次上，从一种表示到另一种表示的转换。它保留了主题系统的外部行为（功能和语义）。重建的转换通常是一种表现，例如改变代码以改进其传统意义上所谓的结构设计中的结构。重建建立实现或建议改变主题系统的新版本，它通常不包含为实现新的需求为进行的修改。然而，重建通过改进系统的某些方面的内容，使得主题系统具有更好的可观察性。

重构的定义与之类似：对软件需求进行改进的过程，该过程不会改变代码的外部行为，仅仅改变代码的内部结构[40]。关键的思想从方便未来的修改和扩展的目标着手，对类、变量和方法重新分配职责。

文献[46，102]对重构在复杂需要中的应用进行了描述，在[文献 46]中，重构被引入到测试中处理其不断增加的复杂性。文献[102]描述了重构在大型 JAVA 框架中对代码进行单元测试的影响情况。

2 目前的研究

2.1 形式化

目前已经提出了多种形式化用于处理重建和重构。本文提供了一个可表示的但（不可避免的）不完整的形式化列表。

Program Slicing[91, 12]用于处理特定类别的重建：函数或过程获取[58, 57]。这些基于系统依赖图的技术可用于保证一次重构保存了一些可选择的行为。[56]提出的方法与之类似，但几乎没有采用形式化的方法[56]，使用一个算法用于移动可选择的一种控制流图的节点集合，以便当其保存程序语义时使其可获取。

Graph transformations[30, 36, 35, 73, 28]是另外一种处理重建的方法：软件本身用图表示，重建对应转换规则。图转换的提出提供了对软件重构的支持[70, 71]。文献[54]建议使用图转换方法用设计良好的模式替换历史遗留系统中设计低劣的模式。

Software Metrics[20, 38, 51]是另外一种处理重构的方法。可以在重构前使用，衡量（内部或外部的）软件系统的质量，或在重构后，衡量软件质量改进的程度。文献[31]提出了使用变换的 Metrics 检测两个连续的软件系统版本的重构。文献[85]使用 Metrics 检测是否需要对一个给定的软件系统进行重构。文献[25]定义了一个多项式衡量方法，提供单一可维护索引用以评价重构的效果。

规范概念分析技术[43]可用于处理重建。文献[86]应用概念分析，基于应用集合中类层次的“使用”情况重建面向对象类层次。该方法可以保证与初始层次在行为上保持一致。文献[95]使用同样的技术重建软件模块。文献[98]使用概念分析方法，通过半自动重建原有数据结构识别对象。

在文献[77]中，Philipps and Rumpe 建议重新考虑已经存在的精练方法[103, 5, 76]，使其成为规范化处理行为、行为相同和行为保存的概念的方法。这些概念不仅仅应用于重构领域，而且也使用了更加成熟的代数定义和精练技术：

文献[45]提出的 Hidden sorts 可用于清楚地区分内部和外部的可见的行为，并讨论了保留外部可见行为的含义。

文献[81]讨论了 Behavioural re_nement of state machines。该方法的用处在于，对行为等价物的保留通常受到太多的限制。该方法的思想是增加细节以从抽象定义行为中获取具体实现。

文献[76]采用的 Re_nement of dataow architectures 对可观察行为给出了清楚的定义，以便准确地定义对行为的保留和精练的意义

文献[6]谈到的 The re_nement calculus 是按照一种规定逐步获取需要的程序的框架

2.2 技术

从软件再工程的角度考虑，重建通常用于将原有代码转换成为更加结构化、模块化的形式，或者甚至可能将代码迁移到不同的编程语言中，甚至是不同的语言范畴中[37]。软件可视化是另外一种有助于对软件进行重建的技术。文献[48]提出使用星型图来实现可视化。DupLoc 是一个用于测试代码重复性的图形工具[33]。CodeCrawer 是一种可视化的软件工具，用于确定需要进行重建[59]。

元建模技术由于其能够使重构更少依赖于实现的语言，因此是一种非常有用的技术[90]。Lgic meta programming 技术用于检测面向对象软件中“代码的坏气味”以发现何处需要进行重构[63, 97]。

2.3 语言

大量不同的编程语言和语言范畴可用于支持重建：

强制性编程语言如 Fortran[13]；

功能性编程语言如 Scheme[47], Lisp[60], Haskell[89]；

基于类的面向对象语言如 Smalltalk[80], Java[40], C++[75,94]；

基于原型的面向对象语言如 Self[72]；

最近对重构研究的趋势在比源代码更高层的范畴上，例如 UML 设计模型[3, 17, 87]。[15]开发了一个重构浏览器集成于 UML 建模工具中。它提供重构类图、状态图和活动图的支持。比较新的还包括[8]，在数据库模式进化环境中应用重建操作。

2.4 支持工具

虽然可以采用手工方式进行重构，但一般认为采用支持重构的工具进行重构工作是非常必要的。目前，在重构包含的各个方面都有相当数量的支持工具。从工具和它能够提供的支持种类来看，自动化的程度有相当大的差别。诸如 Refactoring Browser [80], XRefactory [104], jFactor [52]这类的工具提供半自动方法实现重构。一些研究人员提供了实现全自动重构工具的可能性。例如，Guru 能够实现 SELF 程序的重建继承层次和重构方法的自动化

[72]。文献[19, 54, 82, 23]报告了其它一些自动化的重构工具和方法。当前关于重构工具的一种趋势是将它们直接集成到商业软件开发环境中。例如 SmallTalk 的 VisualWorks V7[22]，Eclipse V2[34]，Together ControlCenter V6[92]，Intellij IDEA V3[53]，Borland JBuilder V7[16]等等。这些工具的应用的焦点是满足用户对重构的需要。但是这些工具对于在何时、何处需要应用重构几乎没有提供什么支持。文献[85]提出解决该问题的一些方式，[55]指出如何通过使用 Daikon 工具检测程序不变量的方法自动地应用重构方法。该方法基于对程序运行的行为的动态分析实现，可以将其视为其它方法的补充。

3 未来的趋势

尽管在重构领域相当多的工作已经实现，从实践及规范的角度来看，仍然有许多问题仍然需要关注并解决。本节将对这些问题进行总结，并给出部分的解决办法。问题的组织方式分为基础部分和研究问题，实践部分和工具问题等两个主要的分类。

3.1 基础研究问题

3.1.1 哪种形式最适合我们的需要？

该问题在文章的前部已经作了部分的回答。我们发现下列形式在重构环境中经常被使用：图变换，软件度量，程序分割，概念分析，精炼技术。

显然，也还存在其他形式。例如，统计技术可用于实现经验化地测量重构的实际使用情况和对软件质量的影响。

3.1.2 如何以一种可扩展的方式来组织重构

对于一些大型再工程项目来说，需要提供对复杂重构的支持。目前的开发工具和形式仅仅支持基本的重构。为了能够使工具能够支持工业软件，将基本的重构支持组合，以实现更加复杂的重构是非常有必要的[94]。更加复杂的重构的有利条件之一是它们比使用一系列基本重构来实现更加有效。例如，仅需要对前置条件进行一次检查，而不像采用基本重构方法那样需要检验多次[79, 64, 67]。

文献[88]提出了三种方式解决该问题：顺序表示应用变换，设置迭代，采用迭代的方式对程序元素进行变换，并发表示变换集，并行完成变换。

3.1.3 如何分析重构间的依赖关系？

当进行复杂的重构时，确定哪些重构是相互独立的，哪些重构需要顺序进行，非常重要。重构的并行应用通常导致不可预见的冲突[68]。因此，实现检测并解决这类冲突的形式基础是必要的[64, 65, 66]。从理论的角度来看，可以利用在图变换系统[7]和临界对分析[50]已经得出的关于并行性和并发性的结果。

检测重构间的顺序依赖对处理变化传播[78]问题非常重要：因为众所周知的重构应用的“波纹效应”可能需要其它更多的重构加入应用中[97]。

3.1.4 行为是什么？重构如何使行为得以保存？

重构意味着程序的行为没有发生改变，但是几乎没有对行为进行精确的定义或者定义太不充分难以经受时间的检验。

另外一个问题是对可观察行为等同的直觉的定义，指出“对同样的输入，可以得到完全相同的输出”也并不是非常充分的。在许多应用领域，可观察到的输入-输出所包含的语义并非唯一的：

对实时系统来说，行为的基本表现是操作序列的执行时间；

对嵌入式系统来说，内存约束和能源消耗也是需要考虑的有关行为的重要方面；

对安全至关重要的系统来说，存在一些安全的具体概念需要在重构时保留。

因此，从理想的角度出发，重构也应该保留这些属性。实际上，这些属性不一定需要所有的软件实体都加以保留。由此可知，事实上对行为保留的这一概念有广泛的理解。

从实践角度来看，对行为保留的理解和处理可以采取非常实际的方式，例如，依靠严格的测试规则。如果拥有一个相当完整的测试例，并且在重构后进行全面测试，将是重构行为被保留的有效证据。

从规范的角度来看，可以考虑确定一种具有高度可表达的规范语言，用于表示目标程序的不变因素，并能够将此结果与重构集合关联以保证所有可表达属性被保留。这与目前比较有名的 Courcelle [29]中关于单值二阶谓词逻辑相似。

从研究的角度出发，关于行为保留的不同定义已经应用到重构环境中。文献[11]定义了对象保留类变换。文献[10]使用了保留语言的特定图变换：所有接受的程序输入集合在变换前后必须一致。采用这样的方法，提供一种用于重建的规范语言理论的基础框架。

3.1.5 如何才能确保在重构期间不同层次的软件制品的一致性？

代码级的重构将会影响高层的软件制品（如设计模型、分析文档、架构等），反之亦然。因此，当软件制品进化时，所有不同层次的软件描述应该保持一致。在同一层次上制品间的一致性（如 UML 设计中的状态图、交互图和类图）在重构的情况下也非常重要[17]。

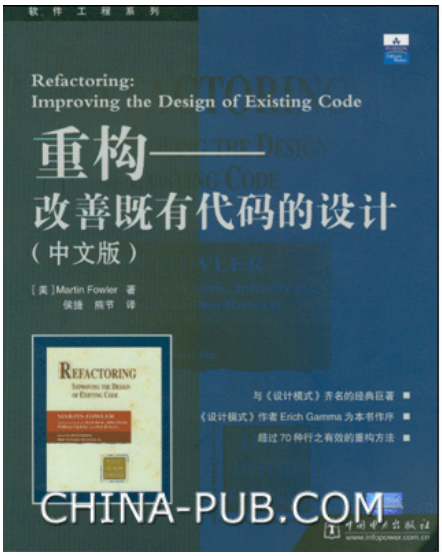
3.1.6 如何为重构比较工具、技术和方式？

为了评价某一重构工具或方式是否比其它重构工具更好，需要以一定的严格的方式进行。要达到该目标，需要获得相关进化的判定标准的分类基础。文献[69]首次提供了比较软件进化的一般化分类工具。文献[84]应用该分类对四个不同的重构工具进行了比较。

3.2 实践问题

3.2.1 如何以与语言无关的方式定义重构

用于重构的工具或者规范的模型应该具有足够的抽象性以应用到不同的编程语言，但也应该提供必要的钩子用于增加语言所特有的行为。OMG 提出的 MDA 通过将平台划分为独立于模型的平台和依赖于模型的平台的方式，允许类似的平台独立性的实现[83]。重构比编程语言能够在实现依赖的层次上更有用，例如在字节代码的重构甚至执行代码的重构等方面。这方面的工作通过编译器的开发者和编译器建立器已经完成[41, 62]，这里的关注点是性能的优化但用于传输（例如字节码）的技术与定义的重构应该是兼容的。从应用领域的角度考虑，这些技术可用于改进系统的其它方面（例如安全性、资源耗费、代码尺寸、内存使用等等）。对运行的编译器的研究[18]可用于重构中，使得运行的应用在需求变化时无需重新开始。



3.2.2 如何将重构应用到更高抽象层次中

重构在更高抽象层次上也可以应用而不仅仅是针对源代码。例如设计模型、设计模式和软件架构：设计模型通常采用 UML 定义和描述。目前已经有相当多的向 UML 模型提供重构支持的工作[3, 15, 17, 87]，但是仍然需要一个更集成化的方法。MDA 对 UML 进行了扩展，实现了对构件模型的改进[39]。MDA 代码生成器基于模型-模型和模型-代码的转换，其配置由所谓的“模式”或者“模板”编辑器[24, 27]完成。从重构角度考虑，需要解决的问题是如何将代码产生器应用于设计优化。

设计模式提供了一种在更高层次上描述程序结构的方法[42]。通常，重构将新的设计模式示例引入到软件中[23, 82, 93, 101]。设计模式也包含对软件结构的约束，这些约束可能会限制一些重构方法的应用。为了检测出这种情况，可以借助于逻辑规则[96]。文献[54]建议使用图形变换技术，通过采用良好的设计模式，重建/替换遗留系统中存在的不良设计模式。对软件结构的重构而言，文献[76]提出了一种方法，重构规则直接基于可能采用的系统结构的图形化表示。这些规则保留了组件间行为中的因果关系。文献[94]还采用了一个更加实际的方法：通过完成一系列基本重构完成两个软件系统间的结构变化。

3.2.3 如何建立更加开放的重构工具？

除了语言的独立性外，重构工具应该使得用户能够方便地扩展新的重构操作。为此，现有工具的可扩展机制（例如，插入、API 或工作环境）不够充分。文献[60]建议使用模式语言来表示重构。

3.2.4 如何确定在何处及为什么进行重构？

多数重构工具仅仅提供应用重构的支持，不能指出在何处及为什么应该应用某些重构。在文献[55, 85]提出的关于解决这类问题的技术可能会有助于解决这类问题。

质量是软件领域的重要因素。如果能够自我约束，例如，在 WEB 应用中，何处及为什么要重构通过文献[1]的高级重构能够获得部分答案。文献[88]以软目标图指导变换过程应用。软目标图描述了质量属性间的关系。重构与软目标中可能存在的影响的结合在于通过基本的和复杂的重构解决了增强可维护能力。

元编程技术可以用于定义与质量有关的关于面向对象软件（如对“坏气味”的检测）的启发式规则，用于发现重构应用于何处的问题[63，97]。

3.2.5 如何处理进化的冲突

另外一个问题与重构中存在的进化冲突有关。该问题在模型对象应用框架下显得更加突出。

框架可能被多个不同的客户实例化，而框架本身经常需要进化。因此框架的重构可能会导致实例间的进化冲突[64]。归并技术[68]可以运用于此。

3.2.6 在软件开发过程中重构的适用性？

如前所述，工具可以帮助确定如何、在何处、何时应用重构。另外一个关于何时应用重构的问题和软件开发过程有关。敏捷开发过程如 XP[9]支持重构。理由是重构活动需要较短的迭代开发周期。出于同样的理由，软件工程中瀑布或者螺旋开发模型不太适合于应用重构[14]。因此，是否以及如何将重构的活动包含于更多的经典软件开发过程中仍然是一个需要解决的问题。文献[32]描述了如何将重构应用于软件再工程过程。

类似于 SmallTalk VisualWorks[22]、Eclipse[34]、IDEA[53]这样的集成开发环境对 XP 都提供了大量的支持，将 XP 中的两个核心行为，重构和单元测试结合起来。文献[99]对测试与重构间的关系进行了更加深入的研究和探索以解决重构使测试无效的实际问题。文献[100]表明测试代码的重构与生产代码的重构在两个方面存在差异：它们有各自不同的“坏气味”集合；对测试代码的改进包含另外一些与测试有关的重构。

最后一个问题是重构如何适应模型驱动的再工程过程。MDA[83]的一个目标是通过从抽象模型建立代码的方法方便平台的迁移实现[4]。粗略看来，该方法将大大减少重构的工作。然而，代码产生隐含前向工程并介入了固定结构，并且不用手写代码表示。重构可以应用于转换存在的代码的设计，成为一种能够被反向工程所理解的格式。需要进一步研究确定哪些重构可以在何时应用到模型驱动的再工程的何处，以及有什么其它的技术可以辅助该工作的实现。

3.2.7 重构对软件质量的影响是什么？

对任何的软件系统，可以定义其外部质量属性（如正确性、健壮性、可扩展性、可重用性、兼容性、效率、易用性、可移植性和功能性）[71]。重构可以按照它们对这些质量属性的影响情况进行分类。采用这种方式可以帮助我们通过正确地应用有关的重构改进软件系统的质量。有许多不同类型的重构，每个都有其特定的目的和效果。一些重构消除冗余代码，一些建立抽象层次，一些增强了可重用性，当然，一些重构可能对性能有负面的影响。通过按照内部质量属性分类重构，可以估计出重构对软件质量的影响。

4 结论

软件重建和重构的研究是非常活跃的研究领域。尽管商业化的重构工具越来越多地出现，但是仍然有许多问题没有得到解决。一般来说，这些问题包括需要能够提供更完善的一致性、通用性、扩展性的重构形式、过程、方法和工具。本文讨论了大量的问题，从基础到实践。我们认为这些问题可以被用到未来软件重构的研究工作中去。

5 参考文献

[1] Alur, D., J. Crupi and D. Malks, \Core J2EE Patterns," Sun Microsystems Press, 2001 .

[2] Arnold, R. S., \Tutorial on Software Restructuring," IEEE Press, 1986 .

[3] Astels, D., Refactoring with UML, in: Proc. 3rd Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering, 2002, pp. 67{70, Alghero, Sardinia, Italy.

[4] Atkinson, C. and T. K uhne, The role of meta-modeling in MDA, in: Proc. UML 2002 Workshop on Software Model Engineering, 2002, pp. 67{70, Dresden, Germany.

[5] Back, R.-J., Correctness preserving program renements, Technical Report Mathematical Centre Tracts #131, Mathematisch Centrum Amsterdam (1980).

[6] Back, R.-J. and J. von Wright, \Renement Calculus," Springer Verlag, 1998.

[7] Baldan, P., A. Corradini, H. Ehrig, M. L owe, U. Montanari and F. Rossi, \Handbook of Graph Grammars and Graph Transformation,"World scientic, 1999 pp. 107{188.}

[8] Banerjee, J. and W. Kim, Semantics and implementation of schema evolution in object-oriented databases, in: Proc. ACM SIGMOD Conference, 1987.

[9] Beck, K., \Extreme Programming Explained: Embrace Change," Addison Wesley, 2000.

[10] Bergstein, P. L., Maintenance of object-oriented systems during structural evolution, Theory and Practice of Object Systems 3(3) (1991), pp. 185{212.

[11] Bergstein, P. L., Object-preserving class transformations, in: Proc. Conf. Object-oriented programming systems, languages, and applications (1991), pp. 299{313.

[12] Binkley, D. and K. Gallagher, Program slicing, Advances of Computing 43 (1996), pp. 1{50.

[13] Bodin, F., Sage++: an object-oriented toolkit and class library for building Fortran and C++ restructuring tools, in: Proc. 2nd Object-Oriented Numerics Conference, 1994, Sunriver, Oregon.

[14] Boehm, B., Software engineering, IEEE Transactions on Computers 12(25) (1976), pp. 1226{1242.

[15] Boger, M., T. Sturm and P. Fragemann, Refactoring browser for UML, in: Proc. 3rd Int'l Conf. on eXtreme Programming and Flexible Processes in Software Engineering, 2002, pp. 77{81, Alghero, Sardinia, Italy.

[16] Borland, JBuilder (2002). URL www.borland.com/jbuilder/

[17] Bottoni, P., F. Parisi-Presicce and G. Taentzer, Coordinated distributed diagram transformation for software evolution, Electronic Notes in Theoretical Computer Science 72(4) (2002).

[18] Buytaert, D. and F. Arickx, A selective runtime compiler for the wonka virtual machine, Presentation at PACT Symposium at University of Ghent (2002).

[19] Casais, E., Automatic reorganization of object-oriented hierarchies: a case study, Object Oriented Systems 1 (1994), pp. 95{115.

[20] Chidamber, S. R. and C. F. Kemerer, A metrics suite for object-oriented design, IEEE Trans. Software Engineering 20(6) (1994), pp. 476{493.

[21] Chikofsky, E. J. and J. H. Cross, Reverse engineering and design recovery: A taxonomy, IEEE Software 7(1) (1990), pp. 13{17.

[22] Cincom, Smalltalk VisualWorks (2002). URL www.cincomsmalltalk.com/

[23] Cinn‼ eide, M., \Automated Application of Design Patterns: A Refactoring Approach," Ph.D. thesis, Department of Computer Science, Trinity College, University of Dublin (2000).

[24] Codagen, Codagen architect (2002). URL www.codagen.com/products/architect/

[25] Coleman, D., P. Arnold, S. Bdo , H. Gilchrist, F. Hayes and P. Jeremaes, \Object-oriented Development: the Fusion method," Prentice Hall, Englewood Clis, NJ, 1994.

[26] Coleman, D., D. Ash, B. Lowther and P. Oman, Using metrics to evaluate software system maintainability, IEEE Computer 27(8) (1994), pp. 44{49.

[27] Compuware, Optimalj pattern-driven generator (2002). URL http://www.compuware.com/products/optimalj/

[28] Corradini, A., H. Ehrig, H.-J. Kreowski and G. Rozenberg, editors, \Graph Transformation," Lecture Notes in Computer Science 2505, Springer-Verlag, 2002.

[29] Courcelle, B., Graph rewriting: an algebraic and logic approach, in: J. V. Leeuwen, editor, Handbook of Theoretical Computer Science, Vol. B (1990), pp. 193{242.

[30] Cuny, J., H. Ehrig, G. Engels and G. Rozenberg, editors, \Graph Grammars and Their Application to Computer Science," Lecture Notes in Computer Science 1073, Springer-Verlag, 1996.

[31] Demeyer, S., S. Ducasse and O. Nierstrasz, Finding refactorings via change metrics, in: Proc. Conf. Object-oriented Programming, Systems, Languages, and Applications, ACM SIGPLAN Notices 35(10) (2000), pp. 166{177.

[32] Demeyer, S., S. Ducasse and O. Nierstrasz, \Object-Oriented Reengineering Patterns," Morgan Kaufmann and DPunkt, 2002.

[33] Ducasse, S., M. Rieger and S. Demeyer, A language independent approach for detecting duplicated code, in: H. Yang and L. White, editors, Proc. Int'l Conf. Software Maintenance (1999), pp. 109{118.

[34] eclipse.org, Eclipse (2002). URL www.eclipse.org/

[35] Ehrig, H., G. Engels, H.-J. Kreowski and G. Rozenberg, editors, \Theory and Application to Graph Transformations," Lecture Notes in Computer Science 1764, Springer-Verlag, 2000.

[36] Engels, G., E. Hartmut and G. Rozenberg, editors, \Special Issue on Graph Transformations," Fundamenta Informaticae 26(3,4), IOS Press, 1996.

[37] Fanta, R. and V. Rajlich, Restructuring legacy C code into C++, in: Proc. Int'l Conf. Software Maintenance (1999), pp. 77{85.

[38] Fenton, N. and S. L. Peeger, \Software Metrics: A Rigorous and Practical Approach," International Thomson Computer Press, London, UK, 1997, second edition.

[39] Flater, D., Impact of model-driven standards, in: 35th Annual Hawaii International Conference on System Sciences (HICSS'02), Lecture Notes in Computer Science 9 (2002), p. 285.

[40] Fowler, M., \Refactoring: Improving the Design of Existing Programs," Addison-Wesley, 1999.

[41] Fraser, C. W., D. R. Hanson and T. A. Proebsting, Engineering a simple, ecient code-generator generator, ACM Letters on Programming Languages and Systems 1 (1992), pp. 213{226.

[42] Gamma, E., R. Helm, R. Johnson and J. Vlissides, \Design Patterns: Elements of Reusable Object-Oriented Languages and Systems," Addison-Wesley, 1994.

[43] Ganter, B. and R. Wille, \Formal Concept Analysis: Mathematical Foundations," Springer-Verlag, 1999.

[44] Glass, R. L., Maintenance: Less is not more, IEEE Software 15(4) (1998), pp. 67{68.

[45] Goguen, J., Hidden algebra for software engineering, in: Proc. Conf. Discrete Mathematics and Theoretical Computer Science, Australian Computer Science Communications 21 (1999), pp. 35{59.

[46] Graham, W., extreme programming in a hostile environment, Sessionpaper at Int. Conf. on eXtreme Programming (2002).

[47] Griswold, W. G., \Program Restructuring as an Aid to Software Maintenance," Ph.D. thesis, University of Washington (1991).

[48] Griswold, W. G., M. I. Chen, R. W. Bowdidge and J. D. Morgenthaler, Tool support for planning the restructuring of data abstractions in large systems, in: Proc. 4th Symp. Foundations of Software Engineering, ACM SIGSOFT Software Engineering Notes 21(6) (1996), pp. 33{45.

[49] Guimaraes, T., Managing application program maintenance expenditure, Comm. ACM 26(10) (1983), pp. 739{746.

[50] Heckel, R., J. M. K uster and G. Taentzer, Conuence of typed attributed graph transformation systems, in: Graph Transformation, Lecture Notes in Computer Science 2505 (2002), pp. 161{176.

[51] Henderson-Sellers, B., \Object-Oriented Metrics: Measures of Complexity," Prentice-Hall, 1996.

[52] Instantiations, jFactor (2002). URL www.instantiations.com/jfactor/

[53] IntelliJ, IDEA (2002). URL www.intellij.com/idea/

[54] Jahnke, J. H. and A. Z undorf, Rewriting poor design patterns by good design patterns, in: S. Demeyer and H. Gall, editors, Proc. of ESEC/FSE '97 Workshop on Object-Oriented Reengineering, Technical University of Vienna, 1997, Technical Report TUV-1841-97-10.

[55] Kataoka, Y., M. D. Ernst, W. G. Griswold and D. Notkin, Automated support for program refactoring using invariants, in: Proceedings of the International Conference on Software Maintenance (2001), pp. 736{743.

[56] Komondoor, R. and S. Horwitz, Semantics-preserving procedure extraction, Technical report, Computer Sciences Department, University of Wisconsin-Madison (2000).

[57] Lakhotia, A. and J.-C. Deprez, Restructuring programs by tucking statements into functions, in: M. Harman and K. Gallagher, editors, Special Issue on Program Slicing, Information and Software Technology 40, Elsevier, 1998 pp. 677{689.

[58] Lanubile, F. and G. Visaggio, Extracting reusable functions by ow graph-based program slicing, Trans. Software Engineering 23(4) (1997), pp. 246{258.

[59] Lanza, M. and S. Ducasse, Understanding software evolution using a combination of software visualization and software metrics, in: LMO 2002 Proceedings, Hermes Publications, 2002 pp. 135{149.

[60] Leit~ao, A. M., A formal pattern language for refactoring of Lisp programs, in: Proc. 6th European Conf. Software Maintenance and Reengineering (2002), pp. 186{192.

[61] Lientz, B. P. and E. B. Swanson, \Software maintenance management: a study of the maintenance of computer application software in 487 data processing organizations," Addison-Wesley, 1980.

[62] Lindholm, T. and F. Yellin, \The Java Virtual Machine Specication," Addison Wesley, 1996.

[63] Mens, K., I. Michiels and R. Wuyts, Supporting software development through declaratively codied programming patterns, Journal on Expert Systems with Applications 23 (2002), pp. 405{431.

[64] Mens, T., \A Formal Foundation for Object-Oriented Software Evolution," Ph.D. thesis, Department of Computer Science, Vrije Universiteit Brussel, Belgium (1999).

[65] Mens, T., Conditional graph rewriting as a domain-independent formalism for software evolution, in: Proc. Int'l Conf. Agtive 1999: Applications of Graph Transformations with Industrial Relevance, Lecture Notes in Computer Science 1779 (2000), pp. 127{143.

[66] Mens, T., A formal foundation for object-oriented software evolution, in: Proc. Int'l Conf. Software Maintenance (2001), pp. 549{552.

[67] Mens, T., Transformational software evolution by assertions, in: CSMR Workshop on Formal Foundations of Software Evolution, Lisbon, 2001.

[68] Mens, T., A state-of-the-art survey on software merging, IEEE Trans. Software Engineering 28(5) (2002), pp. 449{462.

[69] Mens, T., J. Buckley, A. Rashid and M. Zenger, Towards a taxonomy of software evolution, in: Proc. Workshop on Unanticipated Software Evolution, 2003, Warshau, Poland.

[70] Mens, T., S. Demeyer and D. Janssens, Formalising behaviour preserving program transformations, in: Graph Transformation, Lecture Notes in Computer Science 2505 (2002), pp. 286{301.

[71] Meyer, B., \Object-Oriented Software Construction," Prentice Hall, 1997, second edition.

[72] Moore, I., Automatic inheritance hierarchy restructuring and method refactoring, in: Proc. Int'l Conf. OOPSLA '96, ACM SIGPLAN Notices (1996), pp. 235{250.

[73] Nagl, M., A. Sch urr and M. M unch, editors, \Applications of Graph Transformations with Industrial Relevance," Lecture Notes in Computer Science 1779, Springer-Verlag, 2000.

[74] Opdyke, W. F., \Refactoring: A Program Restructuring Aid in Designing Object-Oriented Application Frameworks," Ph.D. thesis, University of Illinois at Urbana-Champaign (1992).

[75] Opdyke, W. F., Refactoring C++ programs, Technical report, Lucent Technologies/ Bell Labs (1999). URL st-www.cs.uiuc.edu/users/opdyke/wfo.990201.c++.refac.html

[76] Philipps, J. and B. Rumpe, Renement of information ow architectures, in: M. Hinchey, editor, Proc. ICFEM'97 (1997).

[77] Philipps, J. and B. Rumpe, Roots of refactoring, in: K. Baclavski and H. Kilov, editors, Proc. 10th OOPSLA Workshop on Behavioral Semantics (2001), Tampa Bay, Florida, USA. URL www4.informatik.tu-muenchen.de/papers/PR01.html

[78] Rajlich, V., A model for change propagation based on graph rewriting, in: Proc. Int'l Conf. Software Maintenance (1997), pp. 84{91.

[79] Roberts, D., \Practical Analysis for Refactoring," Ph.D. thesis, University of Illinois at Urbana-Champaign (1999).

[80] Roberts, D., J. Brant and R. Johnson, A refactoring tool for Smalltalk, Theory and Practice of Object Systems 3(4) (1997), pp. 253{263.

[81] Scholz, P., \Design of reactive systems and their distributed implementation with statecharts," Ph.D. thesis, Technische Universit at M unchen (1998).

[82] Schulz, B., T. Genssler, B. Mohr and W. Zimmer, On the computer aided introduction of design pattern into object-oriented systems, in: Technology of Object-Oriented Languages and Systems (1998), pp. 258{267.

[83] Siegel, J. and OMG Sta Strategy Group, Developing in OMG's model-driven architecture, Technical Report White Paper Revision 2.6, Object Management Group (2001).

[84] Simmonds, J. and T. Mens, A comparison of software refactoring tools, Technical Report vub-prog-tr-02-15, Programming Technology Lab (2002).

[85] Simon, F., F. Steinbr uckner and C. Lewerentz, Metrics based refactoring, in: Proc. European Conf. Software Maintenance and Reengineering (2001), pp. 30{38.

[86] Snelting, G. and F. Tip, Reengineering class hierarchies using concept analysis, in: Proc. Foundations of Software Engineering (FSE-6), SIGSOFT Software Engineering Notes 23(6) (1998), pp. 99{110.

[87] Suny‼ e, G., D. Pollet, Y. LeTraon and J.-M. J‼ ez‼ equel, Refactoring UML models, in: Proc. UML 2001, Lecture Notes in Computer Science 2185 (2001), pp. 134{138.

[88] Tahvildari, L. and K. Kontogiannis, A methodology for developing transformations using the maintainability soft-goal graph, in: In Proceedings of the 9th IEEE Working Conference on Reverse Engineering (WCRE) (2002), pp. 77{86.

[89] Thompson, S. and C. Reinke, Refactoring functional programs, Technical report, Computing Laboratory, University of Kent (2001).

[90] Tichelaar, S., \Modeling Object-Oriented Software for Reverse Engineering and Refactoring," Ph.D. thesis, University of Bern (2001).

[91] Tip, F., A survey of program slicing techniques, Journal of Programming Languages 3(3) (1995), pp. 121{189.

[92] TogetherSoft, ControlCenter (2002). URL www.togethersoft.com/

[93] Tokuda, L. and D. Batory, Automated software evolution via design pattern transformations, in: Proc. 3rd Int. Symp. Applied Corporate Computing, 1995.

[94] Tokuda, L. and D. Batory, Evolving object-oriented designs with refactorings, Automated Software Engineering 8(1) (2001), pp. 89{120.

[95] Tonella, P., Concept analysis for module restructuring, Trans. Software Engineering 27(4) (2001), pp. 351{363.

[96] Tourw‼ e, T., \Automated Support for Framework-Based Software Evolution," Ph.D. thesis, Vrije Universiteit Brussel (2002).

[97] Tourw‼ e, T. and T. Mens, Automatically identifying refactoring opportunities using logic meta programming, in: Proc. Int'l Conf. Software Maintenance and Re-engineering (CSMR), 2003.

[98] van Deursen, A. and T. Kuipers, Identifying objects using cluster and concept analysis (1998).

[99] van Deursen, A. and L. Moonen, The video store revisited { thoughts on refactoring and testing, in: Proc. 3rd Int'l Conf. eXtreme Programming and Flexible Processes in Software Engineering, 2002, pp. 71{76, Alghero, Sardinia, Italy.

[100] van Deursen, A., L. Moonen, A. van den Bergh and G. Kok, Refactoring test code, in: M. Marchesi, editor, Proc. 2nd Int'l Conf. eXtreme Programming and Flexible Processes, 2001.

[101] van Winsen, P., \(Re)engineering with object-oriented design patterns," Master's thesis, Universiteit Utrecht (1996).

[102] Wege, C. and M. Lippert, Diagnosing evolution in test-infected code, in: Succi, G., Marchesi, M. (eds.): Extreme Programming Examined, Proc. 2nd Int. Conf. eXtreme Programming and Flexible Processes in Software Engineering (2001), pp. 127{131.

[103] Wirth, N., Program development by stepwise renement, Comm. ACM 14 (1971), pp. 221{227.

[104] XRef-Tech, XRefactory (2002). URL xref-tech.com/speller/

UMLChina 公开课

“UML 应用实作细节”

（2003 年 9 月 27-28 日，上海）

现在，UML、RUP、Rose 等概念已经传播得越来越广，书籍、资料也越来越多，决定在开发过程中使用 UML 的开发团队也越来越多。

在开发团队应用 UML 的软件开发过程中，自然会碰到很多细节问题：“我这样识别 actor 和用例对不对？”、“用例文档这样写合适吗？”、“RUP 告诉我该出分析类了，可类怎么得出来啊？”、“先有类，还是先有顺序图啊？”、“类怎样才能和数据库连起来啊？”...许许多多的细节问题，而每一个细节都和背后的原理有关。

本课程秉行 UMLChina 一贯的“只关心细节”的原则，内容完全是由 UMLChina 自行设计，围绕一个案例，阐述如何（只）使用 UML 里的三个关键要素：用例、类、顺序图来完成软件开发。使学员自然领会 OOAD/UML 的思想和技术，并对实践中的误区一一指正。整个过程简单实用，简单到甚至可以只有一个文档，非常适合中小团队。

详情请见>>

使 RUP 敏捷

Michael Hirsch 著，Michael Zhou 译

查看评论

摘要

统一开发过程，尤其是它的Rational软件公司的实现，Rational 统一过程（RUP），是一个几乎覆盖了软件开发项目所有方面的综合全面的过程。然而，由于RUP所提供的大量不同级别的细节，许多专家认为RUP并不适合小而快速的项目。本文报告了在两个团队规模为只有3到5个开发员的项目中使用RUP的经历。结果证明RUP可以适应小项目的需要，并且在两个项目中都非常有效。在小项目中成功应用RUP的一个关键是仔细选择合适的工件子集并保持这些工件非常简明，剔除不需要的形式主义的工件。本文就进入如何使RUP敏捷的细节，RUP是如何应用到这两个小项目上，以及它是如何配置的。另外，本文还将介绍RUP的哪些元素对项目的成功是起作用的，以及为什么RUP不能防止其他项目的结果不是最优。

1 引言

本文报告了应用RUP在小型项目上的经历。在本文的上下文中，小型项目指的是开发人员为3到5个，开发周期为6到9个月的项目。本文的焦点是我们如何在Zühlke Engineering AG中为这种类型的项目裁剪RUP，如：我们使用什么工件以及为什么要使用这些工件，迭代的典型次数与长度，用于项目计划与控制的方法。

Zühlke Engineering AG，创立于1968年，是一个独立的系统开发服务公司，总部位于苏黎世，在法兰克福和伦敦拥有办事处。今天，我们雇佣了超过200个工程师。我们的主要业务是为不同行业中的客户定制开发软件、电子、机械和微型机械系统。我们的商业模型的一个基本设想是客户提供领域知识，而我们提供项目管理和技术知识，因此系统和定义清晰的需求工程就显得非常重要。典型的项目团队在集成系统[1]项目上由7到20个工程师组成；对于只有软件的项目则由3到10个工程师组成。我们的商业项目有超过50%是软件开发。

1 集成系统项目是一种需要不同工程学科技能的项目,如软件、电子学和机械工程。

对于软件开发，在1991年以前，我们使用的是结构化的分析、设计和编程方法。在那时，我们就已经意识到，结构化技术对于复杂系统来说不具有良好的伸缩性，我们无法使用它们达到我们所需要的生产率水平，以在客户期望不断增长的市场上保持竞争力。在1992年，我们开始转移到面向对象的分析、设计和编程，到1995年，我们已经完成了这一转移。象许多其它公司一样，我们一开始也定义了我们自己的开发过程，它建立在瀑布模型上。它并不能使我们真正满意，到1998年，我们开始认真地寻找替代物。

经过对公开可用的迭代过程进行一段短期考察后，我们决定尝试RUP。作出这一决定的主要原因是：（1）RUP是我们能够找到的唯一具有良好文档的和全面广泛的迭代过程[2]；（2）RUP非常完整，这为我们节省了创建模板，指南以及其它等的时间；（3）我们对RUP有这么一种印象，那就是RUP是由一些实践者所创立的，他们反对由某些方法学家所假定的我们都生活在一个完美世界的理论；最后（4）RUP的文档具有电子形式，这是在真实世界的项目中被真实世界的软件工程师所接受的重要先决条件。我们已经在使用UML为对象建模的事实也使得我们很容易的采纳RUP。

2. 1998年时，极限编程和其他敏捷方法还没有广泛发布和容易获得。

2 为小项目采纳 RUP

2.1 RUP 的简要介绍

RUP事实上覆盖了典型软件开发项目的所有方面。下图1显示了RUP的两个重要维度，（1）在时间轴（水平轴）上是项目的组织结构，（2）纵轴反映了软件开发项目的工作区域。

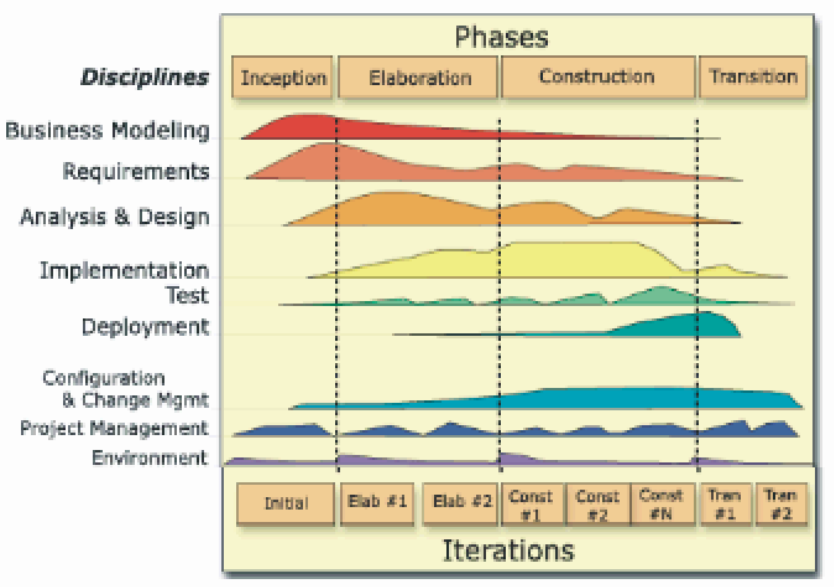


图1 RUP概览

工作区域在RUP中称为科目（Discipline）。RUP中定义了9个科目：

• **业务建模：**描述业务过程和业务的内部结构，以便（1）更好地理解业务；（2）能够为为此业务开发的软件系统抓住适当的需求。

• **需求管理：**引出，组织和对需求进行文档化。

•**分析和设计：**构建体系构架和设计软件系统。

•**实现：**书写并调试源代码，进行单元测试和构建管理。

•**测试：**集成、系统和可接受测试。

•**部署：**为软件打包，创建安装脚本，书写最终用户文档和完成其他使软件对最终用户可用需要完成的任务。

•**项目管理：**项目计划和监测。

•**配置和变更管理：**覆盖下面提及的所有任务（1）版本和发布管理工作（2）变更请求管理。

•**环境：**采纳项目（或组织）所需要的过程，选择、引入和支持开发工具。

图1中与每个科目相联系的条形图的高度说明了在该时间点上花在该科目上的工作量。可以理解，业务建模和需求管理在早期阶段需要更多的工作，而部署工作则在项目的收尾阶段。

对每个科目，RUP定义了一序列工件，活动和角色。工件就是工作产品，如文档，源代码，或者是以UML表示的对象模型。活动是一个小的工作单位，如创建、修改、添加到或查看工件的详细描述。角色是项目中负责某项工作的一个或多人，如项目经理，软件架构师，或测试设计师。

在时间轴上，RUP将项目分为下面的四个阶段：

•**先启**：定义项目的目标，包括业务用例。

•**精化**：创建和验证软件系统的体系架构，捕获最重要的和关键的需求，并对项目的剩余部分作出计划和估计。

•**构建**：基于在精化阶段创建的的可执行体系架构实现系统。

•**产品化**：对系统进行Beta测试和准备发布产品。

每一个阶段又进一步分为一个或多个迭代。每一个迭代都建立在上一个迭代的结果之上，且交付一个系统的可执行（开发）版本。迭代的周期和目标在迭代开始之前被计划好，当迭代完成时，进行一个完整的评估以便可以纠正动作（如果需要的话）。

2.2 采纳 RUP 的考虑

在RUP的最新版本（2001年11月的2002版）中，RUP定义了大约80个主要工件，150个活动和40个角色。当我们在1998年第一次看到RUP时，它还只有很少的工件、活动和角色，但即使是那时，它对我们的项目来说也显得太大了。因此，我们开始创建一个称为轻量RUP（*RUP Light*）的RUP版本，它是RUP的缩减版本，具有尽可能少的过程。今天，我们可能打算称这一活动为“使RUP敏捷[3]”，这也是为了顺从流行用语（现在在业界，敏捷一词比较流行，如敏捷建模——译者注），但是在1998年，术语“敏捷方法”还没有出现，至少我们不知道它。

我们决定在下列区域裁剪RUP：

3 请查看 www. .agilemanifesto.org 上的敏捷宣言

•**工件**：我们只想保持真正需要和有额外价值的工件，所以我们对工件的数目作了大幅裁减。我们保留了剩余工件的建议结构（如文档的目录表），但也对我们认为不确切的模板作出了我们想要的改变。这也正好是RUP所推荐的。我们不认为这是RUP的错，因为我们需要对我们想要维护的文档的范围进行定制。由于RUP是一个针对所有类型和大小的项目的框架，所以适量的定制是正常的和应当的。在下一段落中，有一个我们所保持的所有工件的列表。

•**活动**：我们没有改变任何活动，与RUP建议的正相反，我们决定不将它们用于详细的迭代计划。原因很简单：项目中的所有开发员都在用例建模，对象建模，UML以及其它由RUP强调的技术方面富有经验。进入所谓的工作流细节的活动和它们的集合是十分细节化的，如“描述一个用例”或“为一个类写单元测试”。在这么详细的级别对有经验的开发员做工作计划是没有意义的。我们宁可将活动当作教科书，如果某个开发员在如何完成分配的任务上需要指南，他可以咨询适当的活动描述找到答案，然而在别的方面，活动的描述不被使用。

•角色：我们没有正式将角色指派给开发员。我们仅将角色作为一个检查列表，验证我们团队具有所有需要的技能。

•项目计划：我们决定按照RUP的建议，维护两个级别的计划。第一个级别是粗粒度的项目计划，它基本上列出了所有阶段和迭代的开始日期和结束日期，以及每一迭代的主要目标。第二个级别是每个迭代的详细计划，它在迭代开始前几天准备好，并在迭代过程中根据需要调整。与其根据要被完成的任务构造迭代计划，我们更愿意根据在一次迭代中我们想要完成的成果列表来构造迭代计划。典型的成果有产品特性的规格说明书和实现，某个变更请求或BUG修订的实现，或者是一个主要的非代码相关的工件的准备工作，如软件体系架构文档。 每一个计划项被分配给一个负责完成相关结果的责任人。一个典型的迭代计划将有大约1打特性要实现，至多10个变更请求和BUG修补。除了特性、变更请求和BUG修补外，我们还使用第四种计划项，由于缺乏更好的词汇，我们称之为“附加工作项”。附加工作项是所有不直接与单个特性相关的成果，如软件基础结构组件（例如：O/R映射，错误记录机制等），主要测试的准备和执行，或开发工具和库的安装和配置。不管怎样，计划的主要焦点在产品特性上，因为这才是客户所看到的和想要的（并且是最终为之付钱的）。

•阶段：我们不作任何修改的采纳了RUP的四个阶段及其相关的里程碑。

•迭代：我们决定迭代的的周期大约为4周。每次迭代产生一个可测试的软件发布，制作好发布通知和安装脚本给客户。对某些人来说，4周可能看上去很长。不管怎样，对我们来说，一次迭代首先是一个最初的计划周期，内含我们要在该周期中达成的目标的详细列表。迭代的长度与我们构建和集成系统的频率没有任何关系。每日构建是我们所有的RUP项目的规则，客户可以频繁地查看系统的当前状态。我们将来自客户的微小变更请求合并到正在进行的迭代中，而对大的变更请求，则被延期到以后的迭代中，因为它们通常需要对整个项目重新作出计划。

•项目控制：每周进行的全体项目团队成员参与的状态会议是我们跟踪进度的主要工具。由于每个要被完成的任务是由单个人负责，没有任务由谁负责以及谁负责什么的模糊性。每周，每个开发员估计要完成他的迭代目标所需要的剩余工作量。这样一来，实际与计划的偏差就很快一清二楚了。我们建立了一个规则，那就是，如果我们用完了时间，我们不会延长迭代以完成原始计划中的所有任务。 如果我们不能遵循原始的迭代计划，我们宁可在计划的日期内结束迭代，尽管只完成了较少的结果。我们认为按时交付一个少于计划特性的发布要比交付一个具有所有特性但严重延期的发布更好。这一观点也是大多数客户的共识。

我们将我们如何应用RUP的想法和规则总结在一个只有几页纸的的内部备忘录上。除此以外，没作其他任何客户化工作。我们刻意没有修改RUP的在线文档以反映我们的改变，因为我们不想在RUP的每次新的发布时作重复工作。到2002年八月时，该内部备忘录已经成了公司的专用RUP用户指南，它描述了如何采用RUP，被发布在我们内部网的SEPG(软件工程过程组织) 主页上。

3 项目一：汽轮机设计工具

使用我们在第二段落中描述的指导方针和考虑作为武器，我们在1999年10月1日开始了我们的第一个正式RUP项目的迭代阶段。当前的RUP版本为RUP 5.1.1。

该项目的使命是创建一个用来设计汽轮涡轮机的叶片的软件工具。客户是阿尔斯通电力公司,一个领先的电力发电设备和电厂的生产厂商。该工具被阿尔斯通的遍布全球的几个设计中心的大约20到30个机械工程师所使用。

3.1 挑战

这个项目的主要挑战是：

•非常短的开发时间：从第一概念到在产品环境中部署最多9个月。

•带有2D和3D图形的复杂用户界面。

•我们必须为某些关键算法集成现有软件，一个客户在Mathlab中写的软件。

•一开始的需求非常模糊，因为在此之前没有类似的工具存在。

我们决定使用下面的方法来解决这些挑战：

•一个具有丰富经验开发员的小型团队，由两个来自我们公司的全职开发员、一个来自客户的负责Mathlab和测试的全职开发员组成。团队由一个部分非全职的项目经理（我自己）领导。

•应用上面段落中描述的轻量RUP。我们将迭代周期安排为刚好一个月，在每月的第一天开始，最后一天结束。事后证明这是非常有用的，因为它很容易计划，10.并且在项目中创建了一个稳定的节奏感。 我们都知道如果我们在迭代的第十天还没有把要实现的特性的规格说明书搞清楚就麻烦了，或者我们应该在每个迭代的第25天左右就完成特性。

•客户的尽早和非常强烈的参与。除了来自客户的开发员外（他的大部分时间是在这个项目上的），来自客户

的这个项目的负责人也会参加所有的迭代计划和迭代评估会议。

•基于产品特性的对需求进行系统管理：客户的所有期望都以产品特性的形式来表达，特性的规格说明书遵循最小化需求（即尽可能小），通常每个特性从少量的句子到少量的段落。特性规格说明书由开发团队的两个工程师共同完成，一个来自我们，另一个来自客户。

•每日构建：我们在第一次迭代中就启动了编码活动，并且在整个项目期间都维持每日构建。

•以Java-2实现：我们选择Java版本1.2的标准版作为我们的开发和目标平台是基于下面的原因：支持 2D和3D图形，全面可用的GUI工具包(Swing)，与C++相比所具有的高生产率，最后它的工具（Java SDK)是免费的。

这一方法证明非常成功。项目超前了最初计划两个月完成，实现了所有需求特性，最重要的是客户非常满意。

3.2 一些项目统计数据

下面的表1总结了这个项目完成后的一些统计数据：

团队规模（总人数）	4
团队规模（完全专职）	3.5
用例数目	6
特性数目	40
实现的变更请求数目	53
解决BUG数目	14
迭代数目	6+2
项目周期	7个月
总人天数	260
实现的Java类数目	大约180
总代码行，包括注释行	大约30,000

表1 项目统计

3.3 项目时间规划

表2总结了这个已完成项目的阶段、迭代和发布。项目于1999年10月1日启动，在2000年4月30日交付第一个产品版本。与构建和产品化阶段相比，我们在先启和精化阶段花费了更多的时间和努力。这种情况通常发生在需求在一开始并不清晰，需要在项目期间不断澄清的项目中。客户在使用产品几个月后，反馈一些变更请求和少数新特性，然后我们在附加的两个迭代中实现它们。为简化起见，我们将这两个迭代放到原始项目产品化阶段，而不是定义一个新项目。

迭代	开始日期	结束日期	里程碑	发布
先启阶段				
1	1-Oct-1999	30-Oct-1999	--	R-1.1.1 (GUI布局演示)
2	1-Nov-1999	7-Dec-1999	LCO	R-1.2.1 (概念验证)
精化阶段				
3	8-Dec-1999	31-Jan-2000	--	R-1.3.1
4	1-Feb-2000	29-Feb-2000	LCA	R-1.4.1
构建阶段				
5	1-Mar-2000	31-Mar-2000	IOC	R-1.5.2 (Beta测试发布)
产品化阶段				
6	1-Apr-2000	30-Apr-2000	PR	R-1.6.2 (第一次产品发布)
7	6-Nov-2000	30-Nov-2000	IOC	R-1.7.1 (添加一些新特性)
8	1-Dec-2000	22-Dec-2000	PR	R-1.8.1 (第二次产品发布)

表2：阶段，迭代和发布

表2中的里程碑根据RUP的术语，它们来自于Barry Boehm的工作[4]。

3.4 项目工件

我们在项目的开始确定我们想维护的的工件，并且在“开发情况（*Development Case*）”文档中记录我们的决定。项目期间，只有非常少的工件被添加进来，并且没有工件被去掉。表3总结了此项目使用的工件。

我们包含或排除一个工件的主要标准是：“添加这一工件对客户的价值是什么？”和“如果我们不包含这一工件，会有什么后果？”。如果我们不能找到某个工件的令人信服的添加价值，我们就不会使用它。选择正确的工件总是作出很多权衡。例如：测试计划和测试案例描述是值得采纳的，但我们还是放弃了它们，因为我们赞同非正式的和针对性的的测试。由此节省下来的时间可以用来为客户构建更多的特性。幸运的是，尽管缺乏正式的测试，我们没有遇到任何重大的质量问题。我们将这主要归功于我们使用的系统的软件设计方法以及我们不断从客户中获取反馈的事实。

3.5 经验教训

总而言之，这一项目是客户和我们的一个巨大成功。我们认为下列因素对项目的成功作出了非常积极的贡献：

•迭代和增量开发：使用常规的瀑布过程，将不可能在同样的时间周期和预算内完成这个项目。瀑布过程的所有基本假定，并不适用于这个项目，如需求可以在最前面定义下来，或系统软件架构可以在任何编码工作开始之前定义下来。

•项目计划和监控中的客户积极参与：一个来自客户的代表出席所有的迭代计划和迭代评定会议。哪些特性、变更请求和bug的处理进入迭代计划的决定总是与客户一起作出的。

•来自客户的快速而有用的反馈：通常我们在交付发布后一个工作周之内就能受到来自客户的对新发布的反馈。除了在每次迭代末尾产生的正式发布以外，我们经常性地将我们工作的当前状态以非正式的形式展示给客户，从中获取的反馈通常会被合并到下一个正式发布中。

•一个小的有经验的团队和具有高度激情的开发员。

•用于项目计划、需求和变更管理的低的开销：花在项目管理活动上的工作大约占项目总工作量的7%左右。

•用于变更管理的实用但有效的方法。

4 LCO ：生命周期目标里程碑；LCA ：生命周期构架里程碑； IOC ：最初操作性能里程碑；PR ：产品发布里程碑。

•最后，来自于RUP的用来组织项目的框架和许多模板和例子，为我们在设置项目上节省了许多时间。没有RUP，我们将不得不为项目阶段、工作区域、工件和其它许多事项拿出我们自己的定义。

如果我们可以重新开始,今天我们将会在哪些作法上作出改变呢?首先,我们会在组织和执行测试工作上花费更多的时间。最少，我们将为单元测试使用JUnit框架（查看www.junit.org)，为系统测试使用一个小的记录好的测试案例的集合。 虽然在这个项目中我们没有遇到质量问题,但我们对于系统的质量一点也不感到安全。同样，如果我们的预算能更多一点的话，我们将搜集一些简单易懂的设计理论来对我们系统的设计质量作出定量的观察。

RUP 工件	注解	格式/工具
需求		
前景文档	描述项目目标，重要用户类和所有产品特性。特性是其它工件引用的唯一标识。	文本文档，15页
用例模型	在一个文档中描述系统的所有用例。使用案例是用来更好地理解系统，而不是为了项目计划。 文档的初始版本包括GUI的草图，在后续版本中被实际GUI的屏幕快照所替代。	文本文档，20页
分析和设计		
软件架构文档	描述系统的高级软件设计，包含一些UML图加上重要机制的简短描述。	文本文档，12页
设计模型	系统的详细软件设计，以类和对象的形式，它们以包的形式分组。 我们使用CASE工具：Together / J来为对象建模。这一工具将设计信息保持为源代码中的特殊注释。	Together / J
实现		
实现模型	在RUP术语中，实现模型仅指构建系统所需要的所有工件的集合，如所有的源文件，make程序的描述文件（makefiles），配置文件等。	Java 源代码文件，Makefiles, JBuilder

测试		
缺陷列表	所有打开和关闭的缺陷列表，包括每个缺陷的简短描述。	文本文档，3页
部署		
发布说明	写给客户的关于每个发布的说明	文本文档，2页
安装工件	包括安装系统所需要的所有的可执行文件，配置文件，安装过程，文档等等。对每个版本，都需要创建一个安装工件的集合并交付给客户。	ZIP文件
配置和变更管理		
配置管理计划	描述版本控制、发布管理和变更请求请求的策略。我们使用CVS作为版本和发布管理工具。项目的所有工件都在版本控制管理之下。	文本文档，8页
变更请求列表	所有打开和关闭的变更请求的列表，包括每个变更请求的一个非常简短的描述。	文本文档，4页。
项目管理		
软件开发计划	所有已计划迭代的粗粒度的计划列表。对每次迭代来说，指定了迭代的主要目标，开始和结束日期。这一计划将在随后的每次迭代中更新。	文本文档，8页。
迭代计划	每个迭代的详细计划，描述了将要由谁来完成哪些特性、变更请求、bug修订和其他工作项。	文本文档，4-6页。（每个迭代一份）
迭代评估	迭代结果的汇总，如达到了什么目标，实际实现了什么特性、变更请求和bug修订，以及与计划发生偏差的原因。	文本文档，2-5页。（每个迭代一份）

环境		
开发情形	描述项目是如何采纳RUP的。通常有一个哪些工件被采纳，哪些工件没有被采纳的列表。	文本文档，6页
编码规范	我们重用了公司内部的Java编码规范。	文本文档，10页。

表3：项目工件

4 项目二：付费电视计划系统

当汽轮机设计工具项目在2000年4月完成后，另一个挑战性的项目又要开始了。这一次，客户是一家为付费电视运营商开发和生产设备的公司。该公司目前有一个项目，目标是开发一个电视节目计划系统。该系统应该支持为多个付费电视频道创建节目安排表，并控制设备按照安排表实际地广播内容。技术上，该系统按客户机/服务器方式构建，服务器由客户开发，而客户端则由我们开发。

这个项目的挑战与上面的项目是非常相似的，如不清晰和变化的需求（服务器部分正在开发之中，只有很少的文档），紧张的时间进度，需要集成由客户提供的软件。

由于在汽轮机设计工具项目中，我们非常成功的采用的我们的轻量级RUP方法，我们决定在这个项目上再次使用这一方法。我们为此项目指派相同的团队另加一个开发员。RUP的配置也相同，只有很小的改进，如：我们使用JUnit作单元测试，添加了测试计划和用例情节串连图。实现技术与以前相同，如Java 2标准版。

提供了同一个团队，相同的过程和系统的技术，我们希望重复上一个项目的成功。不幸的是，我们错了，差异发生在客户那里。

4.1 项目时间规划

下面是此项目中按事件发生先后顺序排列的列表：

2000.5.1：项目启动，计划完成日期：2000-10-30。团队对完成项目目标具有高度激情和自信。

2000.5.31：交付1.1.1版本给客户。

2000-6 ：客户没有提供这个版本的反馈。事实上，客户甚至没有安装该软件。许多需求仍然不清晰。客户没有时间与我们一起工作以澄清需求。来自客户的关键人员没有空，因为他们被分配到了其他紧急的项目中，大部分时间被他们遍布全球的客户所占用。我们从客户的组织的不同的人那里收到了互相矛盾的信息。尽管如此，我们仍然冒着遗漏信息的风险继续开发和工作。

2000-6-30 ： 交付1.2.1版本给客户。

2000-7 ： 与六月份一样，客户没有安装交付的这一版本，也没有提供反馈。我们的客户端还没有与服务器集成，因为由客户开发的一个用来访问服务器的中间件层要在两个月后才有，很可能在九月份之前都不可用。我们同意客户，我们将开发一个中间件和服务器的仿真程序，以便能够测试我们的客户端。

2000-7-31：交付1.3.1版本给客户，包括一个中间件和服务器的仿真程序。

2000-8 ：与6月和7月一样....

2000-8-30：交付1.4.1版本给客户，仍然使用仿真的中间件和仿真的服务器。

九月份：一开始，客户简单地看了一下我们创建的软件。与想象中一样，客户有很多变更请求，但是没有时间来足够详细地定义它们。由客户开发的中间件被取消了。我们被要求改变方向，根据先前发布中开发的中间件仿真程序开发项目的中间件，而原来的任务，如开发客户端，则降为第二优先的任务了。

2000-9-30 ： 交付1.5.1版本给客户，带有中间件的一个原型。

2000-10 ：客户雇佣了一个可用性专家来为客户端创建一个新的GUI。该专家在十月中旬交出第一个成果。与此同时，客户将数据库系统由SQL Server改为Oracle，并且多次改变了数据库模式。这影响了客户端和中间件。同时在十月份，我们的一个开发员患了重病，整月无法工作。

2000-10-31：交付给客户1.6.1版本。这是我们的最后版本，因为合同在十月底就截止了。开发团队彻底失败了，拒绝继续在这个项目上工作。

在我们的最后版本中，我们实现了由该专家提出的大约70%的对GUI的变更请求，中间件可以工作，但不能完全适应已经发生改变的数据库系统和数据库模式，许多来自客户的变更请求不能实现，因为没有相应的规格说明书，软件中还留有一些小Bug没有修复。

我们后来了解到，客户收购了一家竞争对手，并将整个系统的维护开发（如服务器、客户端和中间件）交给了收购过来的公司。

4.2 经验教训

这个项目完全没有成功。客户没有得到他想要的，我们不能按计划完成项目，开发团队的士气彻底受挫。这个项目还没有完全失败，至少得到了部分软件，虽然与原始计划的不完全一样。

项目的不幸结局的主要原因是在项目的头四个月期间，完全缺乏客户的反馈，而且客户方的关键人物参与项目不够。整个项目期间，与客户的关系是积极而友好的。来自客户的关键人员也知道用户反馈的必要性，并且很愿意提供更多用户反馈，但由于其他紧急的任务无法作得很好。我们友好地了结了这个项目,虽然项目失败了。

这里要吸取的关键教训是不管软件过程如何复杂，任何软件过程都无法取代客户反馈。我们深信,不管RUP如何设置,都不会对结果有所改进，因为问题的根本原因不是过程问题，而是承诺问题。现在看来，我们应该在第二次迭代后就终止该项目，因为那时情况就已经清楚表明我们没有希望收到来自客户的反馈和足够支持。除此以外，没有别的办法可以得到更好的结果。

5 结束语

自从本文介绍的两个项目以后，我们又使用RUP完成了一打以上的项目，这些项目的规模都为1-4人年，团队规模为3到8个开发员。在第一个项目中确定下来的初始工件集合仍然组成了项目文档的核心。其后，我们添加了xxUnit框架作为单元测试，添加了测试计划，为系统测试添加了文档化的测试案例，以及为每一个项目添加一个强制性的风险列表工件。即使如此，工件的数目依然是很低的，没有给项目带来多余的负担。迄今为止，我们还没有收到来自开发员反映写了过多文档的抱怨，这说明是一个很好的现象。

下面是关于如何是RUP敏捷的一些建议：

•仔细选择一个小的工件子集，并保持这些工件的详细级别到合理的程度。RUP的80多个工件中，只有10到12个是小项目上真正需要的。

•“必须具有”的工件列表包括：软件开发计划；每次迭代的迭代计划和迭代评估；软件体系结构；带有必须的特性列表的前景文档；变更请求列表；缺陷列表；和其他一些工件。详见表3。

•迭代计划应该关注迭代想得到的结果而不是迭代中要执行的活动的列表。在此意义上，RUP联机文档的活动描述和工作流详述可以作为需要时咨询的资料，而不是详细迭代计划的模板。

在我们迄今为止完成的15个RUP项目中，项目管理的工作量大约占总工作量的5%到10%之间。这一数字反驳了RUP是一个为小项目造成大量管理成本的重量级管理过程的说法。除了本文介绍的第二个项目之外，迄今为止我们所作的所有RUP项目都获得了成功。

不管具体采用哪种过程，下面的结论都是适用的：

•对具有大量不确定性的项目来说，迭代和增量的项目计划是项目成功的关键。不确定性如：含糊不清和频繁的变更需求；未经证实的技术；或未知的客户。

•再多的计划和项目管理也代替不了用户和客户反馈。迭代和增量的开发无法补偿反馈的缺乏。

•即使是在小型项目上，一个来自客户方至少有50%的时间用于用户反馈和迭代计划的人是必须的。

我们认为我们的RUP配置体现了敏捷方法的真正精神。最后，敏捷是一种思维习惯，它可以通过不同的过程加以实践，包括RUP的轻量版本。

6 参考文献

1 . Rational软件公司，《RUP联机文档》，版本 2002.05.00 (Rational公司的一个商业产品)

2 . Kruchten, Philippe.《Rational统一过程/导论》 Addison Wesley, 2000

3 . Beck, Kent. 《极限编程解释》 Addison Wesley, 2000

4 . 敏捷宣言, www.agilemanifesto.org(2002年8月)

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

或许我们不应该“写”需求

David Gelperin 著，安丰亮 译

查看评论

编注：这是一份专栏讨论的摘录。

摘要

在本专栏里，David Gelperin 提出了一个我们很多人都熟悉的问题——记录需求最好的办法是什么？本文阐述了静态模板的局限性，那么我们怎么能更好地管理一些更大量的、多维的需求信息呢？

David Gelperin

我们计划开发一个项目，我正在考虑过去的项目中开发需求的方法。我们通常的做法是创建一个需求文档，但这一次这可能不是最好的办法。

对于我们过去的较小的项目，使用文档管理需求效果很好，通过二、三十页纸需求能够被详细说明。我们在本单位或者网上可以获得很多模板，这些模板为需求文档的不同信息提供了编辑样式，然而，文档的效力却无法准确衡量。

大型文档检查和使用都是有困难的。 因为文档的组织形式是固定的，作者必须选择简单的最好的方法组织信息。不幸的是，对于大量的信息，没有简单的、最好的方法甚至是较好的方法。同样的需求信息用不同的方式来组织以满足不同的需要，包括满足正确性和完整性校核的需要。随着需求页数的增加和涉众数量的增加，以一种可以改变的而不是固定的、组织好的方式－－即：使用数据库的需要逐步增长。

例如，我的妻子 Sharon 和我最近为儿子的婚礼建立了一份客人名单。我们曾用过的字处理工具，当名单上只有二、三十人的时候工作的很好。由于邀请了大约 130 人，我就使用了一个电子制表软件来管理，除姓名、地址和关系信息之外，我们还输入了回复和饮食偏好信息。电子制表软件的分类功能使我能够很容易测算出(1)选择蔬菜类用餐、鱼类用餐和肉类用餐的人数；(2)外地宾客以及他们从何处来。电子制表软件的计算功能为我提供了一个重要的导出数据：接待的总数。

当我用关系排序的时候，我注意到计划邀请的一些堂兄不在单子上。虽然这些遗漏的堂兄从按字母排序的名单中能被检查出来，但用关系来做分类排序使得检查这种遗漏容易得多。

需求信息也需要以不同的方式进行聚合。例如包括(1)所有的关键需求，(2) 来自某个涉众的所有需求，(3)所有未完成的需求，(4)所有的接口需求。更大量的、多维需求信息最好应该用能计算、能查询、能重组的工具来管理。

计算能力支持导出信息。例如，允许从测试到源需求的直接链接，允许从源需求信息到测试链接。这显著提高了对于频繁变更的链接信息的多种组织方式的可管理能力。

既然规模较大的需求应该使用电子制表软件或数据库来开发，我现在必须决定使用文档还是数据库能更好地服务于我们的项目。

我通常把电子表格软件看作是功能受限的数据库（有超强计算能力）。他们之间的如何选择应该主要考虑需求的规模。如果要处理带有数十个属性的需求信息，电子制表软件也变得象文档一样难于使用。因此我们在工具的使用上大致可以这样分工：文档和数据库分别负责需求规模小的和规模大的、电子制表软件用来处理中等规模的需求信息。你是如何考虑的？我非常想听听你用什么工具、是怎么考虑的？

读者评论

Erik Petersen

David，你曾提到过敏捷软件开发的问题，很显然我们在这讨论的很多系统需求是有关数量很大的信息需求的，不包含把需求写在索引卡片上和 Alistair Cockburn 的“两人在白板前讨论”这类敏捷开发方法。如果你用敏捷开发方法开展这些大项目，那种一小步一小步的做法，对于处理大系统本身具有的复杂性，很可能是完全不够的。似乎没人提到由于市场压力和其他原因，要求软件的信息能够被自由地传递，需求的状态需要随时被了解....从我了解的情况看，还没有工具能解决这个问题。[Paul Gerrard 在这方面已经做了很多工作] (10/13/02)

Malcolm Stenning

David,如您所知(我来自 RESG, University College London, Oct-2002)，过去几年我一直在研究用于获取需求的分级事件驱动用例建模技术。尽可能的，每个需求都使用基于特定业务情形有明确目的的、详细描述目标如何满足的、不包含解决方案的用例。我发现使用分级方法，使对大量的需求信息的可管理性明显加强。对我来说，做出使用数据库还是文档的决定是非常简单的，每次都选择了数据库，如果你需要所有的或部分需求的拷贝，你简单地运行一下查询，就会产生相应的报告。当你设计数据库结构的时候，你可以把支持你要使用的查询的必不可少的属性加进去。我接触过很多需求数据库，包括上边很多人的回复里提到的Rational公司的产品，然而我主要使用Telelogic的DOORS，我极力推荐这个工具。然而，不管你决定使用的管理需求信息的工具是哪一种，文档还是数据库，对于开发的成功最重要的因素是写高质量的需求。关于这个主题有一本很有用的书，是由Suzanne和James Robertson编著的《掌握需求过程》（Mastering the Requirements Process）（ISBN 0-201-36046-2）（编注：此书中译本已经由人民邮电出版社出版，译者为UMLChina讨论组组长Sealw），一本非常好的书，祝你好运。(10/11/02)

Robert E. Lee

关于需求的生命周期和载体的讨论，这段时期以来似乎是硕果累累。(10/11/02)

Judy Murphy

我花费数年时间为航天项目开发了数千条需求。我们有大量的资料和许多需求文档。所有这些资料都是被手工创建和管理的——包括跟踪。据我所知在我离开航天项目之前都是这么做的。在商业的领域里我发现的一些事情令我感到恐怖。竟然能够忽略需求文档，忽略需求！我现在只用Rational的RequistePro工作，我喜欢这个工具，它支持创建需求文档，也可以把需求信息存储在数据库中。其情形是，在数据库中需求能够进行分类、过滤，能够生成报告。然而，需求组织的真正的不足之处在于创建“好”的需求，不是注重形式，而是内容本身。如果需求疏于管理，那你的需求在哪存放就没有什么差别了。以电子制表软件管理的质量很差的需求是有害的，使用数据库、文档还是其他工具也是一样。(10/10/02)

专家回复－David Gelperin

Judy，你说内容的质量比数量和包装重要的多，显然这是对的。如果在数量少的文档形式的好需求和数量大的数据库形式的劣质的需求之间选择，根本无须选择。(10/10/02)

Joseph Dubowski

这儿的观点都很好。我根本就不熟悉市场上的需求工具（自动化的），尽管如此，我还是想说，我正在寻找的工具的特点是易于使用、具有可搜索能力（按照David提出的说法）、跟踪能力，能够进行冗余检查和变更控制。任何自动化的解决方案都会给需求强加一个规范的模式。我认为《竞争工程》（Competitive Engineering）对于使用工具支持写文档，可能是一个有益的技术框架。(10/09/02)

Mahendra Gupta

David，的确，庞大的需求文档一般来说帮助不大。但是如果我们回顾文档的确切目的，感觉到它规范了开发流程。因此，当一个组织在工程方面逐步成熟的时候，这些需求文档也需要基于所从事项目的类型进行复查和校订。因此文档可以基于相关项目类型分类。影响文档效果的另一个因素是将要使用文档的最终用户对这些文档的理解程度。因此基于最终用户反馈信息起草文档时应该更及时，当用户需求需要改变的时候应该总是反映到需求文档中。只有当人们坚持这些原则，并共享同一个文档视图的时候，文档质量才能得到保证。(10/08/02)

专家回复－David Gelperin

Mahendra：目前我正在阅读Alistair Cockburn编著的《敏捷软件开发》（“Agile Software Development”）一书。和往常一样他写得漂亮，讲述了很多重要的内容。如果你不熟悉敏捷开发方面的理论，我向你推荐这本书。可能你会发现对其中的一些理论很难接受，但是我认为这个理论应该从更广的角度来理解。我的上述建议主要是针对你“文档的目的是规范开发流程”的观点而提出的，敏捷开发理论与你这些意见是不一致的。敏捷开发理论者会说文档减慢了开发进度。最重要的问题应该被研究和讨论，给文档以适当的角色定位就是这样一个问题。感谢你提出的意见。(10/10/02)

April Anderson

最近我们用Access数据库（自定义的、曾用于其他项目）来管理需求，并使用Word的合并功能把需求的文本内容输出到Word文档模板中。一旦需求库建立起来，由开发人员进行一些配置，进行需求更新和复查还是很不错的。使用中存在多用户访问同一个数据库受限制的问题，我们采用Microsoft的共享功能来帮助消除这个问题，但是当进行大量数据更新或创建新表的时候，数据库仍然需要被加锁。另一个限制是版本控制。我们也在bug跟踪工具中跟踪需求变更，这个工具允许我们一直跟踪到更新输出的文档，这对于需求文档进行版本标识是有用的。(10/08/02)

专家回复－David Gelperin

谢谢你，让我们分享你的经验的细节之处。 (10/10/02)

John Daughety

在我工作的几个公司里，工作中我一直忍受者冗长的、令人痛苦的文档的折磨，就没变过样，我一直在考虑如何解决需求文档化的问题。当我最终决定选择数据库，当时间紧张时选用带超链接的EXCEL的时候，我感到对我最重要的是文档化什么内容。许多需求文档化的成果成了巨大的、蚕食时间的怪物，因为需求从没有被反复推敲过。在我最后的公司里我们正在定义支持DWDM光纤网络设备的产品的特征需求。最终我们形成了数千个需求，通过使用access数据库它们被很好地组织起来，但也产生了一些问题，其中许多需求对我们的应用系统来说太详细了，从没被使用过。其余的公司认为搞清楚所有的需求信息将花掉开发计划阶段所有的时间，后来，这些产品因为没有满足真正的需求不得不进行了一些重大改变。当我回顾这些的时候，我感到现在DWDM设备的5页长度的Powerpoint需求图表是非常有效的。对于作为建立项目需求文档的有效媒介，我会对图表投上一票。应该先看看什么内容需要建立文档，然后选择最适宜的方法建立文档。象我的8级英语教师在我们正开始考试时说的那样，“写所有需要写的内容，只写需要写的内容。”在我目睹的太多的案例中，是把需求本身作为目标的，而不是为了需求开发过程运作得更好而使用哪个工具。把需求移到数据库格式的一个危险是建立更多的资料变得比较容易，创建更多的而非对用户有用的信息的风险变得更大。我敢打赌，当你写参加婚礼客人名单的时候，在你的头脑里你已经有了“需求”如何支持你的“项目”的想法，婚礼一定是成功的、毫无麻烦的婚礼。在我最近的项目里，没人对建立需求文档感兴趣，我是把最少限度的时间花费在如此“奢侈”的工作上的唯一的测试者。我使用数据库，开始写我认为需要写的内容，这项工作是我们的产品必然会取得成功的五个因素之一。然后我把这些信息与选定的项目组的成员共同分享，从中再分析出哪些部分还有更多的信息被需要，采用这种迭代的方法建立需求，最终我建立了规模较小但很全面的需求数据集。采用这样的方法，数据库是理想的媒体。(10/08/02)

专家回复－David Gelperin

想法非常好。你提供了这么多观点，下面我打算把它们总结一下，请直接指出我的误解或者多余的部分：1）为了开发真正正确的需求，要依据合理的判断，集中精力研究项目的目标与任务。2)需求太多可能比需求太少效果还要差。3）一些需求信息最好使用图表来表述；4）采用数据库方法引进过多却不是必不可少的需求信息可能是危险的；5）从另一个角度看，数据库的组织方式能够使大量的信息变得条理清晰、便于复查，利于验证正在开发的需求。(10/10/02)

ofer prat

Hi.在市场上有一些需求管理工具(e.g. Calibar RM).哪位有这些工具的经验吗? (10/08/02)

专家回复－David Gelperin

Ofer prat问过，“市场上有一些需求管理工具（比如，Caliber RM），哪位有这些工具的使用经验吗？”我无法设想不使用需求工具怎么来做需求分析。我们公司正在评估CaliberRM和Rational RequisitePro。我非常喜欢CaliberRM，它允许我定义需求，显示它与其他需求的关系，为需求创建属性，以Word形式打印精美的报告。我给你举个例子：在先前的公司，我们评估差别很大的两个程序，对我们考察的每个需求，我给它设定一个数值作为属性，如果这个程序能完全满足该需求，设定值为“2”，如果这个程序能部分满足该需求，设定值为“1”，如果这个程序不能满足该需求，设定值为“0”。然后我把所有这些需求的值加起来，打印一个报告，给出每个程序的最后得分，也列出分值为“0”的需求项。(10/08/02)

Peter Cornish

David把我们引到了这个主题上。当需求变得有点复杂的时候，就有许多其他相关的变化趋势问题应当考虑：1) 需求变更的增长率； 2)已审定的变更增长的数目。这两个问题都影响我们采取什么解决方案。只要我能支付得起购买、维护、培训的费用！(10/08/02)

Laura Murphy

从文档升级到电子表格软件，再升级到数据库，你的选择过程是对的。当我们的公司太小而不能发挥工具的威力的时候，我们曾试图使用象Rational的Requisite Pro那样功能强大的需求管理工具。如果你的项目和你的项目组太小，创建并维护一个需求数据库所付出的劳动与项目的预算和时限是不成比例的。除非所有项目都要求使用数据库，否则是不值得的。我们降低要求直接使用word处理文档，感觉到在我们的公司规范和项目的约束下工作的很好。现在我们公司规模变大了，如你所说，在进行充分的需求跟踪，横向参考不同类型的需求文档方面，文档已经不够用了。我们现在已升级到具有链接、文档交叉参考功能的电子制表软件。这是需要使用需求数据库并采用其文档模板支持的先兆。我认为我们最终会由于项目规模过大而超出电子制表软件的能力，而不得不使用需求管理软件。然而，基于经验，我确信必须使用与你的项目和项目组规模成比例的存储方法。过于强调工具将损害你的产品和利润。不使用工具也可以产生同样的结果。窍门是所用工具的层次与你的开发项目的层次相匹配。(10/08/02)

Clay Givens

这是一个关于需求管理和不断变更的大型文档的有趣的专栏。我一直认为管理随意建立的厚厚的需求文档犹如噩梦。以我的经验看，100页以上的word格式的测试需求文档使用起来将是非常耗时的，在这方面已经超过了它所能带来的好处。还有个问题就是小的需求调整修订经常发生。文档维护简直就如同家务杂事。我天性懒惰，因此我喜欢依靠直觉来执行我的很多测试内容，但不全是这样的，我也采用电子制表软件作为要求详细测试内容的管理工具。每一个测试用例都有三个文本栏目：一是“一般步骤”栏，包括基本的测试步骤；二是“特别步骤”栏，为区别于其他用例的部分；三是“预期结果”栏，便于使用者理解需求的信息。这样做，我就不至于只关注了小树而忽略了整个森林。目前我正在为一个很复杂的应用系统写测试用例和需求，这个系统包括三个数据库的集成和三个不同平台下用户认证会话转换。测试场景包含许多内容，测试是复杂的，相关资料很容易丢失，为此我使用excel来管理需求。相比对于数据库，我发现在没有建立图形界面的情况下，使用数据库太慢而无法进行手工更新，那将花费很多时间。我唯一的使用数据库的时机是把它作为自动化工具的一部分。我成功地使用MySQL作为后台数据存储器，自动化的前端工具为results capturingmedium(SilkTest and PHP)。我仍然认为对于象我工作的敏捷应用开发类型的开发环境，带有基本功能以及拆分功能（象数据库规范化）的电子制表软件，是管理需求的最好解决方案。(10/08/02)

Andrew Raybould

David,当任何信息的数量太大而不能看一看就基本了解的时候，就需要被组织起来。在某些情形中电子制表软件是比较合适的工具。但我发现超链接很有效，特别是对于需求的跟踪能力（包括设计和实现）、对于捕捉需求的依赖性和其他关系。一般来说，图表能够解决文本方式存在的单一化的缺点，图表对于阐述关系是一个好办法，但单纯的图表形式缺乏表现力、充分的说明和自然语言的推理能力（除非你用逻辑判断图）。单纯的图表形式的需求说明（或设计说明）几乎可以肯定是不全面的。最后一点，当然如果你的问题需要大量的严密逻辑，应该使用严格的数学方法。(10/08/02)

Sam Shober

我认为，这篇文章将为以可视化的形式显示需求信息提出一个可行的方法，或者是为需求文档化提出一种通用方法。我认为使用电子制表软件或者是关系型数据库的观点是明智的。任何时候，如果你需要对包含多个子系统和它们的边界的系统建立功能文档的时候，都应该小心谨慎。通过对实体间的关系进行研究，你能较早地理清相关问题。你也能更全面地评估这个项目，你和开发人员能更准确地估算带有所有这些新功能的系统上市将花费的时间。 这种形式的文档还有两个缺点：(1) 设计并维护一个传统的文档可能需要花费大量的时间； (2)你仍然

需要挑选一种格式，把需求内容上会展示讨论，因为你可能还没有完全掌握客户方的真正需求。(10/08/02)

Daniel SUCIU

对David的问题的答案对我来说似乎是显而易见的：采用需求管理工具，关键是“管理”。我认为比较好的做法是在项目生命周期的需求开发阶段就描述出所有的业务活动。工具能够解决David提到的所有问题（从不同的角度进行查询、为不同的目的采用不同的格式/模板，计算能力），特别是与其他工具并行工作，进行版本控制、变更管理、设计、自动测试。无论如何，甚至是单独使用，一个需求管理工具结合了数据库的优点和字处理器的性能，有友好的用户界面，带有内部逻辑，包含需求管理必不可少的规则，工具能够进行自定义，以满足特定的环境、业务规则甚至模板。据我所知，这样的工具有许多，但我唯一熟悉的是Rational公司的产品，它包含用于需求管理的RequisitePro。事实上，使用特定模板、电子制表软件和数据库，都带有特定的查询，带有特定的业务规则，在每个组织中，这也是任何需求工具的核心功能，但是他们没有（或比较少）自动化功能和友好的用户界面。唯一的问题可能是购买工具的花费。如果对于规模大、预算大的项目，可以支付得起自动化工具的相关费用。对于较小的项目支出可能要受到控制。我希望在不远的将来，有更多的这类工具可供选择，花费很低甚至是免费的，象许多缺陷跟踪工具，自动化似乎是比需求管理更远的一步。我不是说需求管理工具将永远代替技术专家（象某些人认为的那样）。它只能帮助她/他组织数据，使其集中精力于重要的问题。(10/08/02)

Hans Thelosen

管理需求与发布需求是不同的，我喜欢以文档形式发布需求（容易使用）。然而如果你管理需求（创建、改变、讨论、推迟），这是操作需求相关属性的问题。这种情形下选择使用文档是错误的。那么为什么不用一个知识库呢？一些工具能够用作知识库来管理需求，并能在完成需求时生成文档。(10/08/02)

sameer nigam

你认为在Designer中使用业务过程模型（Business Process Model） 和业务功能模型（Business Function Model）对获取有用的需求有帮助吗？(10/08/02)

Alberto López Navarro

我感觉这个专栏很有意思。我在一个规模相当大的电信公司工作，公司项目涉及的范围很广，必须按职能区域分成多个部分设置不同的组织分别定义和实现。当所有的相关人员不得不象鱼儿觅食般在整个文档中（尽管只有一小部分与他们的领域有关）寻找需求时，是非常麻烦的事情。这个议题的讨论结果可能是微不足道的，似乎这只是技术/工具及其提供的数据处理能力的问题，但是我认为这样简单的改变很可能对软件生产力产生很大的、

值得关注的影响。(10/08/02)

Fiona Williams

我不得不说对于写需求（或其他任何文档）我曾经得到的最好的帮助来自于一门信息规划课程。它真正集中说明了写需求的办法，它使文档易于快速阅读，文档被设计得能快速浏览，因此查找需要的信息时快得难以置信。我非常想推荐这种方法，关于这种方法的和步骤的更多信息请参阅：:http://www.infomap.com/(10/08/02)

Sandy Flann

我曾看到的最好的需求模板来自于RUP。每一项内容都被分别编号并分类管理。但这仍然是一个“文本”文档，从中我也能看到使用电子制表软件可能存在的问题。(10/07/02)

Kevin Priest

不管是使用字处理器、电子制表软件，还是数据库，最终都是在“写”需求。基于这个专栏题目，我有一个提议——“画”需求（“drawing” requirements），即采用图形化的而不是文本的需求表述方式。在费力地读完数页描述应用的模式间转换的需求说明之后，绘制一个状态转换图将大大地简化需求，并且通常会暴露出一些矛盾和错误）。作为一个测试者，你多少次通过绘图阐明了叙述性的文字说明？有多少次你发现了开发者在做同样的事情？更重要地是，你的图形多少次满足了开发者的需要？我喜欢的需求描述工具不是字处理器、电子制表软件，也不是数据库，而是绘图工具。任何类型的绘图工具，必要的时候也包括纸和铅笔。当然一些表示语言（比如：数据流一致性检验）的知识有助于使用工具，但即使没有它，图形也比单纯叙述性的语言强得多!(10/07/02)

专家回复－David Gelperin

Kevin:我本应该提到图表 (状态转换图、行为图和基本类图) ，对于展示需求信息的外部形式来说，它们是最好的方法。但是就象你所知道的，既然没有最好的需求规格说明方法，它们也不是最好的方法。比如，在概要层，绝大多数的非功能性需求需要叙述性的文本描述。 在细节层，通过接收外部信息/系统测试过程，能被最好地阐释。另一个例子是复合状态的细节定义，例如：客户必须符合某一特别类型的保险单的条件，“条件”是由包含客户的多个基本状态的逻辑表达式定义的复合状态。这个保险单可能被诸如年龄、性别、婚姻状况、住宅所有权等因素的某种组合所限定。对于说明逻辑表达式，图表不是有效的方法，叙述性文本也不是，因此我们使用数学方法。有效的需求说明方法要求实施一个不同的策略，为每一种不同类型的需求信息找到一种最清楚明了的表达方法。也就是说，我们都有我们喜欢的工具。 感谢你参与讨论。(10/10/02)

Gene Fellner

以平面文件记录一个上百万美元的需求，就象制鞋商本人却打着赤脚走路一样。(10/07/02)

Chris DeNardis

首先我对这个标题感到非常奇怪，为什么叫“或许我们不应该写需求”呢？？或许这个标题应该被改成“获取需求的多种方法”或其他什么。象在一篇文章里你说的那样，需求是本已存在的，然而用于建立文档的工具会有所不同， EXCEL不同于WORD，或许甚至power point更多的作为概念图来概略说明需求。我们的做法是用Word作为需求文档的主体，但是在WORD里边有图表——有时一些内嵌的EXCEL工作表，有时是图片，来自于Visio 或者Powerpoint。当我们进行测试的时候，这些文档被用来检验需求，同时通过运行已提供的相关的、已校核并得到审批的测试用例来测试。(10/07/02)

专家回复－David Gelperin

谢谢你又提出了记录需求的另一种选择。我们可以用(1)字处理器, (2) 电子制表软件, (3) 你提到的多种组合,(4)数据库； (5)需求管理平台(可以看作是浓缩的数据库)。(10/10/02)

专家简介

David Gelperin (sqegelp@aol.com)是位于佛罗里达橙色公园大街的软件质量工程协会(sqe.com)的创始人之一 。David在软件工程方面有30年以上的经验，重点关注了软件质量控制。他曾担任的角色有程序员、项目主管，测试主管、质量监控经理、测试咨询和讲师。他主持制定了ANSI和IEEE两个组织的软件测试标准，推动了《软件测试与质量控制》（Software Test and Quality Engineering magazine）期刊的创立。他在俄亥俄州大学Carleton 学院读完了数学专业之后，又继续深造获得了计算机科学专业的硕士和博士学位。

UMLChina 公开课

“UML 应用实作细节”

（2003 年 9 月 27-28 日，上海）

现在，UML、RUP、Rose 等概念已经传播得越来越广，书籍、资料也越来越多，决定在开发过程中使用 UML 的开发团队也越来越多。

在开发团队应用 UML 的软件开发过程中，自然会碰到很多细节问题：“我这样识别 actor 和用例对不对？”、“用例文档这样写合适吗？”、“RUP 告诉我该出分析类了，可类怎么得出来啊？”、“先有类，还是先有顺序图啊？”、“类怎样才能和数据库连起来啊？”...许许多多的细节问题，而每一个细节都和背后的原理有关。

本课程秉行 UMLChina 一贯的“只关心细节”的原则，内容完全是由 UMLChina 自行设计，围绕一个案例，阐述如何（只）使用 UML 里的三个关键要素：用例、类、顺序图来完成软件开发。使学员自然领会 OOAD/UML 的思想和技术，并对实践中的误区一一指正。整个过程简单实用，简单到甚至可以只有一个文档，非常适合中小团队。

详情请见>>

UML 驱动的基于 TTA 协议的处理器设计

Johan Lilius、Dragos Truscan 著，梁涛 译

查看评论

摘要

协议处理器是一种可编程的被特殊设计以有效处理协议的处理器。设计协议处理器的挑战在于发现一种既适合通用的处理器，又适合自定义的跟协议密切相关的处理器的架构。在这里我们描述一种方法，它既设计处理器，也设计一种网络处理的应用程序。我们的设计流程是基于面向对象的技术，并且以 TTA（传输触发架构）处理器的架构为目标。TTA 是一种特殊的处理器架构，它的指令集可以被很容易地裁剪。它由一系列应用于硬件的功能单元组成。我们这里工作的一个目标就是来识别出这样的功能单元。本文中设计流程的步骤是以 Ipv6(Internet 协议，版本 6)路由器的应用为例的。

关键词：

protocol processor (协议处理器)，Internet protocol version 6(Internet 协议，版本 6) ，Ipv6 routing(Ipv6 路由)，UML，design flow（设计流程），transport triggered architecture（传输触发架构）

1 绪论

近几年以来，网络处理面临着复杂程度和性能要求的增加，这已经成为硬件设计人员的一个重要的挑战。一方面，有应用的高性能的要求（这已经通过使用专用的如 ASIC 等硬件来实现），另一方面市场快速的需求使得人们更愿意利用通用处理器的可编程性。网络（协议）处理器对于上述的两种情况通过良好的可编程性和高速的硬件提供了一种解决方案。

使用传统的诸如通用处理器或者专用集成电路（ASIC）来同时达到这些目标是非常困难的。通用处理器提供较高水准的可编程性以及针对较大范围实际应用的一系列产品，但是降低了运算速度，同时芯片需要较大的表面积和较大的能量消耗。而在另一种情况下，ASIC 可被裁减使之适合于特殊的用途，通过专用的硬件来提供最优的性能，但是它们的可编程性差，设计的灵活性低。

为了满足网络处理器性能和可编程方面的需求，可编程处理器已被广泛使用。可编程处理器是集成电路（IC）的一种，它是基于系统芯片（System on chip,SoC）技术的。可编程处理器通过编写程序,相对于通用处理器可以更有效地完成通讯方面的特殊任务。

在这篇文章里面，我们着重于通过使用可编程处理器来设计网络方面的实际应用。从可编程处理器需要完成的实际应用出发，我们提出了处理器的架构。传统的方式是调整软件应用程序来适应硬件平台，或者通过设计一种架构来服务于软件应用方面的需求。一种可编程处理器是采用传输触发架构（TTA）的处理器，它可以提供易于扩展的，模块化的架构以及灵活的配置。设计者的主要任务是决定处理器的配置。这样的配置在一定的诸如速度，芯片表面积，能量消耗等物理条件的限制下，为特殊协议处理的实际应用提供支持。我们现在将此问题看待成软硬件协同设计的问题。我们的解决问题的途径是基于这样的观察结果：对于某一类的协议，一系列的基本的运算可以被提取出来。这些运算在协议处理器构架中被设定成原子运算。

这篇报告的主要贡献在于：

-- 对于协议处理器，提供 UML 高层次的设计流程

-- 引入领域知识，重用硬件组件。

-- 通过检验硬件平台需要实现的功能，提出它的一种配置。

在接下来的几节内容里面，对于 TTA 构架，我们会给出简短的描述，接着是设计流程的步骤，以及各个步骤在为配置处理器而收集必须信息的过程中各自的重要意义。我们将 IPv6 路由器的应用作为案例来研究。我们也在如何寻找所需 TTA 构架功能点方面作了些启发。最后，我们给出使用了基于 TTA 的协议处理构架的一个 IPv6 的路由器配置。

2 TTA 构架

传输触发构架（TTA）是一种新的处理器构架，它给嵌入式系统领域带来了一系列有用的特性。TTA 处理器是由一些功能单元（Functional Units, FU ）构成的，由互联的网络相连接。互联的网络由一条或许多的数据总线形成，FU 在其上通过输入或输出插槽（Sockets）实现互联。FU 可以是属于不同的类型，他们各自独立实现它们的功能。在一个 TTA 处理器里面，相同类型的功能单元可以有多个。

一个 TTA 可以配置成有可变数目的数据总线和功能单元（每个功能单元可以连接一个或多个数据总线）。功能单元通过插槽和数据总线相联，并且通过逻辑地址它们是可寻址的。处理器是仅仅用一种运算（move 运算）来编程的 。这种运算用来描述在互联的网络上数据的传输。处理器运算的发生是功能单元之间数据传输的副作用。每个传输都有起点，终点，还有从一个单元到另一个单元间被传送的数据。

TTA 构架具有诸如模块性，扩展性，灵活性等特点，它可以控制处理器的周期，而这在嵌入式系统的设计中是非常重要的概念。构架的模块性为设计的自动化提供了良好的支持。每个 FU 实现一个功能，而最终的配置可以通过联结不同功能单元的组合而组装起来。在互联的网络中的同一个时刻，每个功能单元完全独立于其他的功能单元。这种结构提供了设计上的灵活性，在设计时，每个 FU 可以独立于其他 FU 或者独立于互联的网络。而且，通过增加更多的 FU，数据总线，或者增加数据的传输和储存，性能可以被扩展开来。当加入 FU 的时候，只要现存的 FU 通过起点和终点的地址长度是可以寻址的，设计人员不需要改变指令的格式。因为处理器运算都是数据在数据总线上传输的副产物，处理器的周期依赖于是否可以获取到功能单元的结果。

为了优化处理器以提高数据的吞吐率，我们可以在功能单元和数据总线间增加存储器，并且同时减少每个功能单元的处理时间。因为 TTA 构架的主要着重点是移动数据，

TTA 构架为高数据密度的实际应用的协议处理提供了一个重要的平台。

2.1 TACO 模拟框架

TACO(Tools for Application-specific hardware/software CO-design,专用硬件/软件 协同设计)是为快速建立原型，模拟，估算和综合运用基于 TTA 架构的协议处理器的一种设计框架[21]。TACO 处理器的整个架构使用一个 SystemC[17]模型，他们的物理参数（譬如表面积，能源的使用）用 Matlab 模型估计，处理器的配置用 VHDL 模型综合而成。SystemC 模型和 VHDL 模型协同开发，这样两个模型的特性相同。决定处理器需要的 FU 类型后，我们在系统的层面来研究目标应用的设计。

通过不同架构配置的 SystemC 模拟和 Matlab 估算，我们发现一个或多个符合目标应用所需性能、能量消耗和表面积需求的候选设计框架。最切实可行的设计框架就可被综合运用到硬件设计中。更多 TACO 处理器的应用细节可以在参考文献[22]中找到。

2.2 TACO 处理器配置

TACO 的开发流程由两部分组成：一是识别出一个专门的应用所需功能单元（FU 的类型和数目，二是根据应用所需功能，针对已知的协议，完成控制结构的设计。识别功能单元同样有两个步骤：

----定性配制 – 通过分析系统的功能，发现所需硬件资源的类型

----定量识别 – 根据物理条件的限制, 通过分析性能方面的需求确定每种类型的硬件资源的数目。

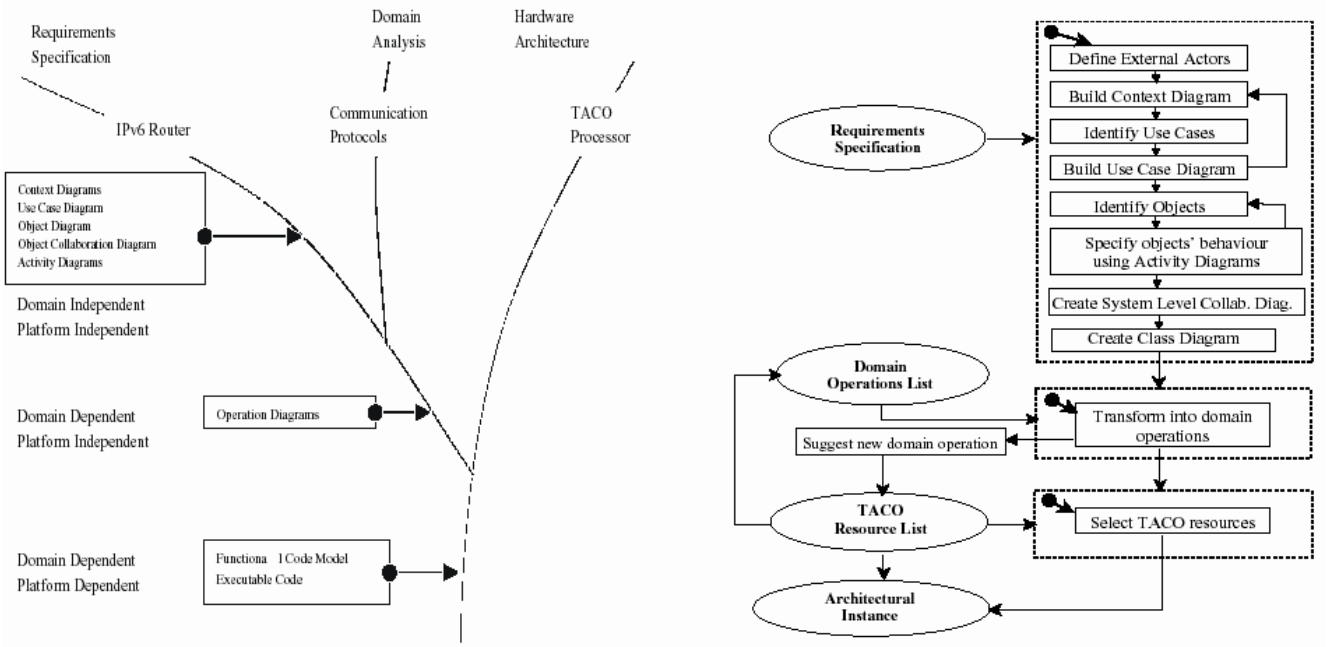


图 1 在 IPv6 路由器应用中的设计流程

接下来我们着重强调开发流程的第一部分，也就是识别出一个专门的应用所需功能单元（FU 的类型。识别出新的功能单元可以通过从已经建立的功能单元中选择，也可以建议和建立新的功能单元。每个新建立的功能单元通过 TACO 的 Matlab 估算模型，提供速度、表面积和能量消耗的估算。在设计阶段的物理参数的估算的可获取性，使得设计者根据应用的需求和限制，完成早期设计空间的探索，以识别出最合适的配置。

3 设计流程

在本文中，对于 TTA 处理器的协议处理应用的开发，尤其是 TACO 处理器，我们实验了一种协议处理设计的方法。因为我们是在讨论可配置的处理器架构，我们对于反应了平台的硬件配置的应用的各个重要的方面的学习必须给出一些启发。通过建议所需资源的类型和数量，这样的信息被用来开发处理器的架构。

在这样的设计过程中，UML 扮演了一个重要的角色。我们使用 UML 作为一种描述性的语言，研究 UML 的行为图（Behavioral diagram）的语义来为我们的模型获得一个良好定义的可执行的语义。而且，近来在代码生成（Code-generation）方面的研究工作使得直接从 UML 的最后规约（Specification）生成 TACO 的微代码（Micro-code）成为可能。

为了使规约更可读，我们需要一种统一的语言来同时描述应用的规约和处理器的硬件资源。由于 UML 是用图形来描述事物的，它被证明是一种有用的分析和规约的工具。通常，描述软件和描述一个系统的硬件部分存在着巨大的不同。类、方法和关联用在软件中，而信号，模块和数据总线用在硬件中。UML 起先是用来做软件规约和构造，但是通过使用原型，它提供了一种灵活的扩展机制。原型是一种扩展机制，它通过允许定义对象的概念性的类型来扩展 UML 的表现力。通过原型，可以增加或改变标志来代表硬件概念，并且使用 UML 来描述硬件平台。通过使用通用的表示方法来代表软件和硬件，硬件和软件工程师都可以很好地理解设计流程，有助于为整个系统形成一个统一的设计过程。

尽管 UML 并没有在嵌入式系统中大规模使用，在文献[2，3，12，19]里面提出了几种基于 UML 的设计方法。在[2]中，有应用分析，并且在用例（Use Case）描述中，设计中的约束（Constraints ）被捕获了。接着使用活动图和对象约束语言（OCL，Object Constraint Language）设计被转化成正式的模型。根据一个固定的硬件资源和面向对象的方法，这个正式的模型可以被分成硬件和软件两个组成部分。文献[12]与传统的 UML 使用方式不同，它提出了一种不同的方式，在此，设计的分析更多是被对象识别（Object Identification）所驱动，而非类（Class）的识别。在该文献中，行为（Behavioral）方面的设计，是用状态转换图（State Charts）来描述的。

针对于所要解决问题的规约，我们的设计流程（图 1）由一些精炼后的步骤组成。在详细设计的阶段，设计流程中会包括必须的架构需求。为了提供组件重用和资源的快速识别，我们把应用的功能性规约和基于领域的知识相结合起来。因为我们在讨论一个基于 TTA 的处理器的应用平台，我们的目标是去发现处理器需要履行的基本运算。基本运算是由一些专门的功能单元或者他们的组合来完成的，它们可以被表达成一些带参数的函数。

设计流程依赖于三种信息资源：

-- 应用需求（独立于领域和平台）

-- 领域知识（独立于平台，依赖于领域）

-- 硬件平台信息（依赖于领域，依赖于平台）

从需求规约开始，我们通过 UML 图把功能性的规约提取出来，把它分解成功能性的小块，以简化分析（需求分析）。这些小块，功能性的对象，在一起协作以获得整个系统的功能。我们集中力量来创建系统的动态视图，动态视图是通过对象间协作和它们的内部功能性（对象分析，Object Analysis）。对象有其内部行为，我们独立于应用的领域和物理的平台架构，用通用的术语将它描述出来。整个过程是在一个独立于领域和平台的方式中完成的。可以在文献[1]中找到更多的细节和在设计流程中产生的 UML 图。

接下来我们通过将功能性的分析限定到一个特殊的应用领域（Domain Mapping,领域映射），例如，网络处理，来缩小功能性的分析。在这点上，这样的描述还是独立于物理平台的，允许在不同的平台上，应用得以实现。

从在领域空间的功能性的分析，使用一个依赖于领域和平台的视图，我们可以提取出关于 TTA 处理器需要履行的功能的有用的信息，并且从功能性方面，建议它的架构。

在提供的架构的基础上，我们可以在不同的架构配置下，模拟相应的应用，并且对每个模型进行估算。

在一个最终的架构产生后，硬件设计流程可以开始了（物理平台的实践，Platform Implementation）。为减小设计空间研究的努力（例如，表面积，能量消耗，速度），可以提供一些启发。

3.1 需求分析阶段

在确定一个应用项目的时候，伴随一堆描述系统各个方面的文件，通常会给出一个非正式的对于问题的描述。大多数情况下，问题是用模棱两可的语言来陈述的，留有需要解释的空间。 为了从不同的资源中收集所有必须的信息以形成一个清晰的规约，我们需要履行一些分析的步骤。我们由问题定义出发，产生一系列用 UML 表达的工件。首先，我们将系统考虑成一个整体，通过识别外部的角色（Actor）和它们跟系统的交流（环境图，Context Diagram）来提取出系统和它的环境的交互。其次，我们识别出系统必须提供的各个功能点和关于功能点的描述（用例列表）。最后，我们将它们表示成提供给外部环境的服务，并且决定是哪一个角色在使用其中的相应服务（Use Case Diagram, 用例图）。

3.1.1 需求规约

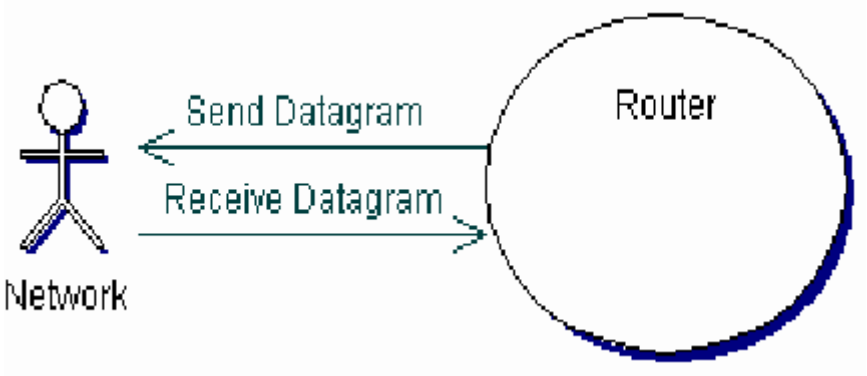
规约是用一种平实的语言以书面形式给出的。需求是以列表的方式表示，每项需求有详细的文字说明。

“本应用的范围是设计一个简化了的 IPv6 路由器，它通过使用路由信息协议（Routing Information Protocol, RIPng）在以太网上发送数据包（Datagram）。路由器是一个网络装置，它将数据由一个网络切换到另一个网络，以使它到达最终的目标。两个主要的功能必须支持：转发（Forwarding）和路由(Routing)。转发是确定将一个 Datagram 往路由器的哪一个接口传送,以使 Datagram 到达它的目的地.,而路由是建立并且维护一个路由表(Routing Table),该表中含有网络的拓扑图（Topology）的信息。路由器通过监听邻近路由器广播的特定 Datagram 建立路由表，来发现网络拓扑结构的信息。按特定的时间间隔，路由表的信息被广播到邻近的路由器，以通知它们网络拓扑结构改变的信息。IPv6 路由器应该可以从它接通的网络上接收 IPv6 的 Datagram，检查地址和数据域的有效性，查询路由表来确定它们被转发的接口，并且将 Datagram 转发到相应的接口。另外，一个路由器需要建立和维护一个含有网络拓扑信息的路由表。“

更多的信息可以从现有的协议定义中得到。IPv6 的协议在文献[9]和[16]中有描述。为建立和维护路由表，我们使用下一代路由信息协议(Routing Information Protocol Next Generation,RIPng,参见文献[15])，这是一个基于UDP的协议。UDP（Use Datagram Protocol,使用 Datagram 协议）是一种无连接（Connectionless）的协议，在文献[18]中有定义。而且，一个 IPv6 的路由器必须实施 ICMPv6 协议[7]，这是 IPv6 协议中的一个内部协议。ICMPv6 是一种通过一些路由信息和其他信息来提供故障检查支持的协议。

IPv6 的 Datagram 是由一个简化了的固定长度的 IPv6 的数据头（header）和一些可选的扩展数据头，这样就提升了灵活性和多选性。而且，IPv6 现在提供一些扩展来支持认证（Authentication）,进行数据完整性的校验和数据的加密。为简化起见，我们假定 IPv6 的 Datagram 只有一个路由扩展头，否则的话 IPv6 的 Datagram 是由 IPv6 的头和上层净荷（payload）组成。

在这篇报告里面，datagram(数据包)是指一个在无连接网络上传送的数据包。无连接（Connectionless）是指在起点和终点无数据连接。



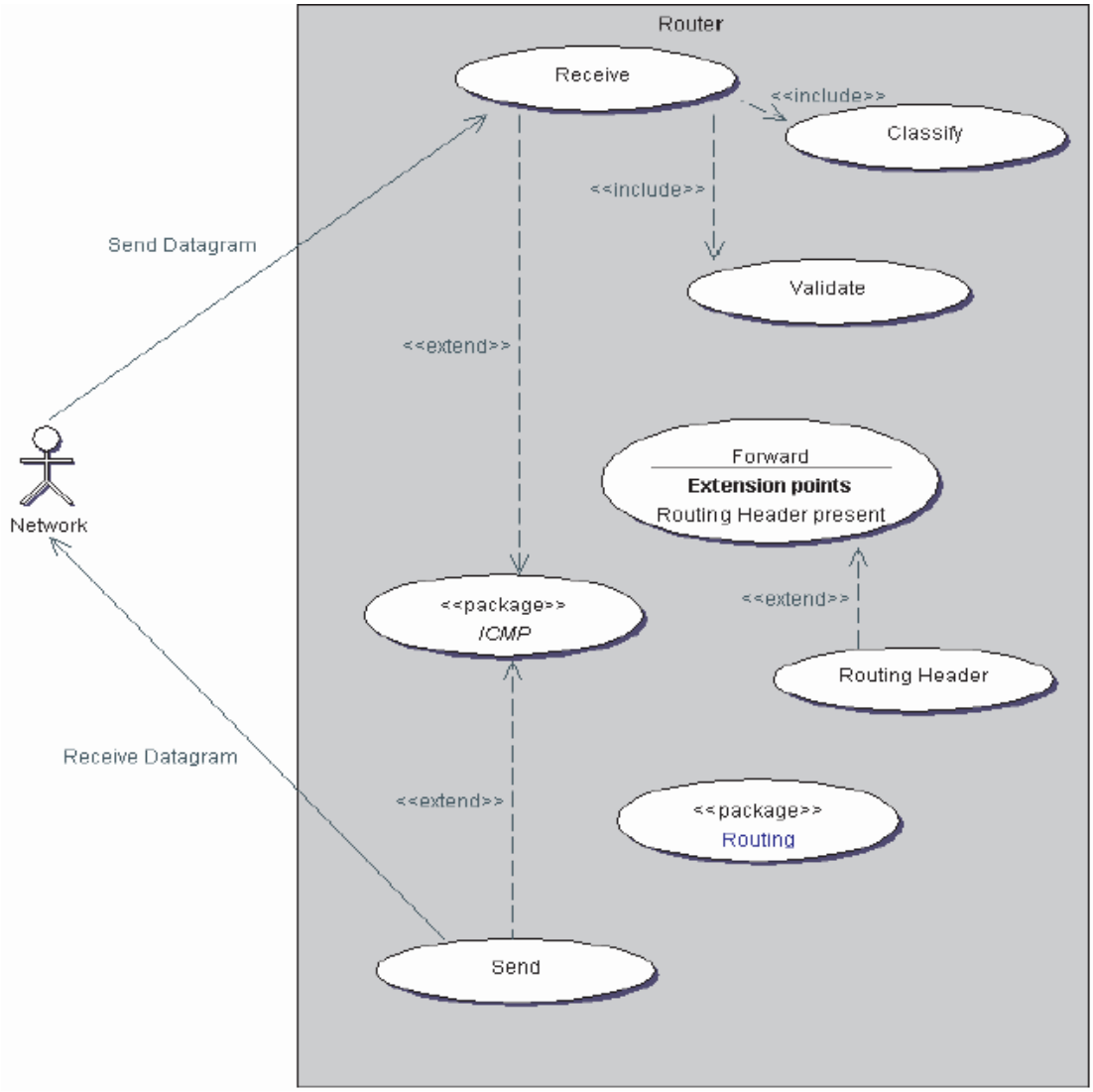


图 2 系统的环境图和用例图

3.1.2 外部角色以及它们和系统之间的交互

系统必须和外部环境相互作用，它必须对外部刺激产生反应，并将信息发送回外部环境中。基于初始的需求，我们绘制环境图（Context Diagram,图 2，左），并且根据它和外部环境的接口定义系统的边界。我们也提取系统和环境之间的通讯（Communication）.

路由器接收从网络来的数据包，处理后，将它们发送至合适的网络中。网络是一个和系统相交互的角色。尽管一个路由器是和不止一个网络相连，我们只绘制一个角色，因为所有相连的网络在和路由的交互中具有一致的结构和行为。与之相应的结果就是，和网络的接口就必须提供接收和发送数据包的方式。

上述的说明是用例图的输入，但它只是迭代过程的一个部分。在定义完毕用例列表和建立了用例图后，一些额外的在环境图中未标明的通讯，会被增加。在环境图中的一些新的细节会被相应地更新。在每次迭代后,对于问题描述的确认，我们就可以应用场景(Scenarios)。在这个层次上，场景包括了作为一个整体的外部角色和系统。这样的方式，有利于更好地理解环境和系统间的信息传送。当图上没有其他改变时，迭代就终止。

3.1.3 将需求捕获入用例分析中

为捕获应用需求，我们提取系统的功能点，并用用例图加以描述。识别用例的几种方法可见于文献[3，11]。在[3]中，作者认为内部自发的活动也应该被考虑成用例，尽管它们对外部世界并不提供服务；相反，在[11]中，内部自发活动在用例分析中是不予考虑的。

我们将同时分析系统内部和外部活动来形成用例，这是一种好的选择，因为它们两者都代表系统需要实施的功能。

从需求规约中，我们识别 7 个主要的用例来表示系统将为外部环境所提供的服务。用例伴随着一个简短描述其功能的文字说明。用例图在图 2，右。

--接收（Receive） ------- 路由器从它所连接的网络中接收到一个数据包。

--确认（Validation） --------接收到数据包后，路由器验证数据包的正确性和地址。

如果发现有错误，IMCPv6 子协议将会向发送者返回一个错误信息。有错误的数据包被丢弃。

--分类（Classification）---- 路由器必须根据接收到的数据包的范围和类型来进行分类。一个进入的数据包要么针对于路由的过程，要么针对于转发的对象，或者针对于路由器的 ICMPv6 协议；否则就要被丢弃。

---转发（Forward） ----- 决定要被转发的数据包是往哪一个接口发送。要决定下一个接口，路由器从路由表中查询路由信息。如果没有相应的转发接口存在，ICMPv6 子协议就会向数据包的发送者传送 ICMPv6 的错误信息。转发过程的一个特例是数据包含有一个路由头。这种情况需要一种特殊的处理。我们将这个用例表示为转发的一个子例（Subcase）,用扩展（Extend）关系和转发相连。

--路由（Routing） ----- 路由器通过收集相邻路由的信息建立和维护路由表（RT, Routing Table）。三个主要的功能必须完成。它们被描述为路由这个用例的子用例：

- --更新（Update）--路由器从相邻的路由器中接收响应（Response）数据包以用来更新其路由表的信息。数据包被接收后，它被检验其有效性和数据域的正确性。
- 如果数据包不正确，一个 ICMPv6 的错误信息被返回给发送者。
- 响应（Respond）--在路由表中的信息被传送给相邻的路由器以通知网络拓扑结构的变化信息。这样的信息周期性地以组播（multicase）的方式传送给所有相邻的路由器，或者在另一个路由的特殊要求下，以单打点广播（unicast）的方式传送。
- 维护（Maintain）-- 当路由表的数据为空的情况下，一个请求信息就会在所有的接口上以组播的方式向所有相邻的路由器播送以要求获取网络拓扑信息。而且，路由器必须处理跟路由表中每条路径相关联的定时器以删除过期的路径。

--ICMP----处理由路由器产生或接收的 ICMPv6 的错误信息和其他信息。我们同样定义一些子用例：

- 处理信息（Treat Info） --对路由器接收的 ICMPv6 信息产生响应信息。

处理错误（Treat Error）--当处理一个数据包时遇见错误或着转发数据包的路径消失，一个 ICMP 错误信息被发送至数据包的发送者。

---发送（Send） ----发送数据包至网络中。依据数据包的范围和类型，数据包被发送至一个或多个接口。这个用例也提供发送 ICMPv6 信息的支持。

UML 的标准允许定义用例包(Use Case Package),用例包包含用例和用例之间的关系。这点可以帮助我们分解大的功能，使之成为多个较小的功能，由此可以建立主系统中的多个子系统。我们建立路由这个用例包，它包含更新（Update）,响应（Response）和维护（Maintain）等子用例；还有 ICMP 用例包，它包含处理信息（Treat Info）和处理错误（Treat Error）这两个子用例。

3.1.4 建立用例图

为建立用例图，我们首先定义系统的边界，和系统交互的主要角色以及外部事件（External events）的列表。

每个用例都代表系统的一个主要功能，每个主要功能都是系统提供给外部环境的一项服务。一个用例可以提供一些子服务（subservices）,或者称为子例（subcases），同时它也和其他用例协作以完成其目标。一系列建立用例和用例图的规则在文献[10]中有定义。下一步是根据用例规约确定环境图中的角色所使用的相应用例。接着我们将用例组合成一个用例图（图 2，右），并且如有必要我们更新环境图。

在环境图中，我们早已识别了角色和系统的边界。通过分析用例列表，我们将发送数据包（Send Diagram）和接收数据包（Receive Diagram）这两条消息（message）同相应的发送（Send）和接收（Receive）这两个用例关联起来。其它的用例（IMCP，路由，转发）尽管同样为外部环境提供服务，但是和上述的两条消息并无直接联系。

3.2 对象分析阶段

在我们提取了系统必须完成的主要的功能后，我们开始识别系统中的对象，和这些对象为完成系统总体功能相互协作的方式。首先我们识别对象，接着通过应用用例场景，我们决定它们是如何同系统中的其它对象相协作的。

3.2.1 识别对象

一些识别对象的技术已经在有关文献中给出。在文献[10]中，一个类图（class diagram）是从需求规约中获得的,系统中的对象则从类的实例获得。其他的途径建议对用例图进行功能性和结构性的分解，然后提取对象[12，19]。我们采用文献[12]中的方法，但略作改动，就是我们从用例中直接提取对象。我们认为这是一种有用的方法，尤其是在嵌入式系统领域。在嵌入式系统中，我们对对象具体执行什么样的功能感兴趣， 而非系统的结构和对象的分类。

我们将每个用例分解成三种对象：接口、数据、和控制对象。我们不将用例包进行如此的分解,我们只将每个用例包转换成一个子系统对象。子系统对象在主系统里面可以被看成是其它的系统，它的行为是由子用例来描述的，而其它由在系统内的外部对象变成它们相对应的外部角色。在子系统里面，我们递归地应用同样的算法，并将它们的用例分解成三个对象。为了对象图有更好的可读性，我们用原型来代表每种对象。

迄今为止我们获得了一系列的包含三种主要功能类型的对象。接口对象（Interface objects）在系统的边界处使用，在和外部角色的通讯中起到中介的作用。控制对象（Control Objects）实现算法并且实现系统的控制行为，而数据对象（Data objects）存储数据并且提供数据的使用。尽管在这里，设计流程里面我们并未涉及一种特别的实现的平台，我们仍然需要十分注意数据对象是如何确定的。我们要将数据对象的接口（Interface）,或者说数据对象的服务（Service）和数据对象里面数据的结构互相独立地识别开来。数据接口是数据对象提供给其它环境的。我们将此用图形表示出来，把每个数据对象分解成数据接口和数据结构对象，两者用组装（composition）的关系联结起来。

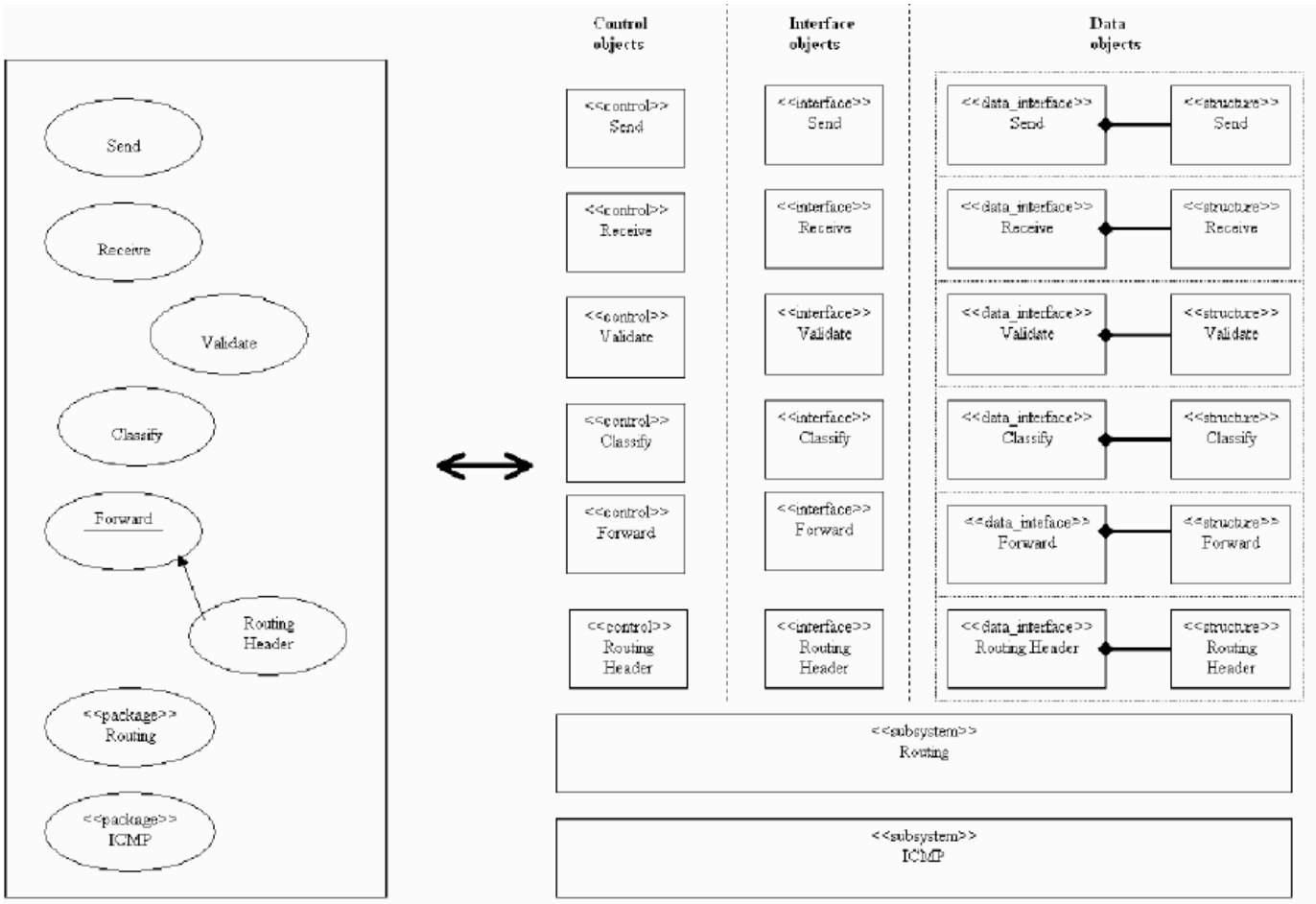


图 3 在系统的层面上将用例分解成对象

首先我们开始分析我们建立的所有对象。为帮助分析，我们检查每个用例和它的场景。从场景所描述的功能来看，一个场景可以涉及超过一个用例。在这个层次上，场景是以书面表格的形式从功能性需求中提取出来。通过应用场景，我们便可以识别对象间的交互，决定获取系统某项功能时具体是哪些对象参与其中，决定对单独的一个用例来说需要保留和去除的对象。一些对象代表部分对于系统来说不必要的功能，这样的对象可以去除。

其次我们分析在每个垂直的类别（图 3，右）中的对象，每个垂直的类别中有同样的类型。一些对象有相似的功能，它们可以被组合在一处。同时，某些对象混合有很复杂的行为（behavior），它们可以被进一步分解。将对象组合在一起并不改变在细节上的层次，或者功能的类型，但是为设计流程的下一个阶段提供了较好的重用性。

在当前描述所处的细节水平上，对象间的通讯只是被确定为对象间的关联关系，并无更多的细节提供。这样的处理过程应用到图 3 后结果可见于图 4。

例如，接口对象（interface objects）在系统边界上使用，在系统和外部角色的通讯中起到中介的作用。只有那些和外部环境（外部系统）直接通讯的对象会需要这样的接口，其余的都被去除。在接口对象的组里面，我们仅仅保留了发送（Send）和接收（Receive）两个对象，它们实现和网络角色间的通讯。

控制对象也被重构（refactor）。当前我们有两个不同的数据对象（路由表数据和转发数据），它们的数据有相同的类型和结构。尽管转发数据这个对象是在路由子系统的外部，我们可以把该对象移入子系统，并且将它合并到路由表数据中。转发控制对象仍然放置在路由子系统的外部，但是它的数据却放置在路由子系统中。这就意味着转发控制对象的部分或全部功能可以被移入路由子系统中。我们把转发控制对象分解成两个控制对象，分别放置在路由子系统的内部和外部。在内部的对象根据特定的地址来查询路由表数据，而外部控制对象用来进行系统控制流程的同步。

相类似地，我们根据数据结构对象（data structure object）所包含的数据类型来分析它，我们发现除了路由子系统所有的对象存储 Internet 数据包，因此它们可以被组合成一个单一的称为数据存储（Data Storage）的对象。每个路由器的网络接口有一个本地化的配置，它包含了网络地址和不同的参数（MTU，costs 等）。网络接口的配置是由发送(Send)和接收（Receive）这两个数据结构对象描述的。由于这两个对象是相同的，我们将两者组合起来并建立一个新的数据结构对象，称之为 LocalInfo。一个特别的情况是转发这个用例，它通过查询路由表的数据得到相应的信息来决定路由器的下一个接口。于是，转发数据对象存储了含有路由表数据的数据。这就意味着我们可以合并路由表数据对象（routing table data object）和转发数据对象（Forward data object）形成一个称为路由表数据对象（routing table data object）的对象。

最后，我们在每个子系统对象（subsystem object）中应用相同的转换。子系统同样有用用例表示的功能，和外部的角色（系统内的对象）进行交互（图 4）。例如，在路由这个子系统内，我们有三个用例：更新（update），响应（response）,和维护（maintain）,它们各自建立一系列的对象。这样的结果是，分类（classify），确认（validate）控制对象和 ICMP 子系统由于使用路由子系统的服务，变成了路由子系统的外部角色。

在这一个步骤中，我们能够了解对象间的通讯，并且了解到具体是哪一个对象参与实现了系统的某一项功能。通讯仅仅描述为对象间的关联，没有其它的细节提供。

3.2.2 用例场景和协作图

每个用例是由一个或多个场景组合在一起来获得其功能的。场景在细节上可以具有不同层次。每个场景有它自身的一系列的对象，这些对象相互协作以获得特定的功能，但是其它用例中的对象可以成为当前场景的角色。场景可以被表示为交互图[6]。协作图是交互图的一种，它着重于对象间如何交互。每个场景产生其自己的协作图（由最初始的规约来绘制），并且在对象图中包含有限数目的对象。

通过显示对象间的信息如何传送，一个对象协作图描述出对象间如何通讯。这帮助我们决定每个对象需要给它的邻居提供什么样的操作。UML 可以描述对象间传送的消息，同时也可以描述消息的时序。在此层次上，我们将对象间的通讯表述成对象对另外对象的服务的请求（request of service）或者操作的请求。我们在每条消息上用杜威（Dewey）十进制数字方法来描述其时间上的次序。每条消息有一个名字，并且可以带一些参数。定义通讯是一个递增的过程，最后的结果就是一个用例的所有的场景已经被实例化了。场景在子系统中被递归地应用。

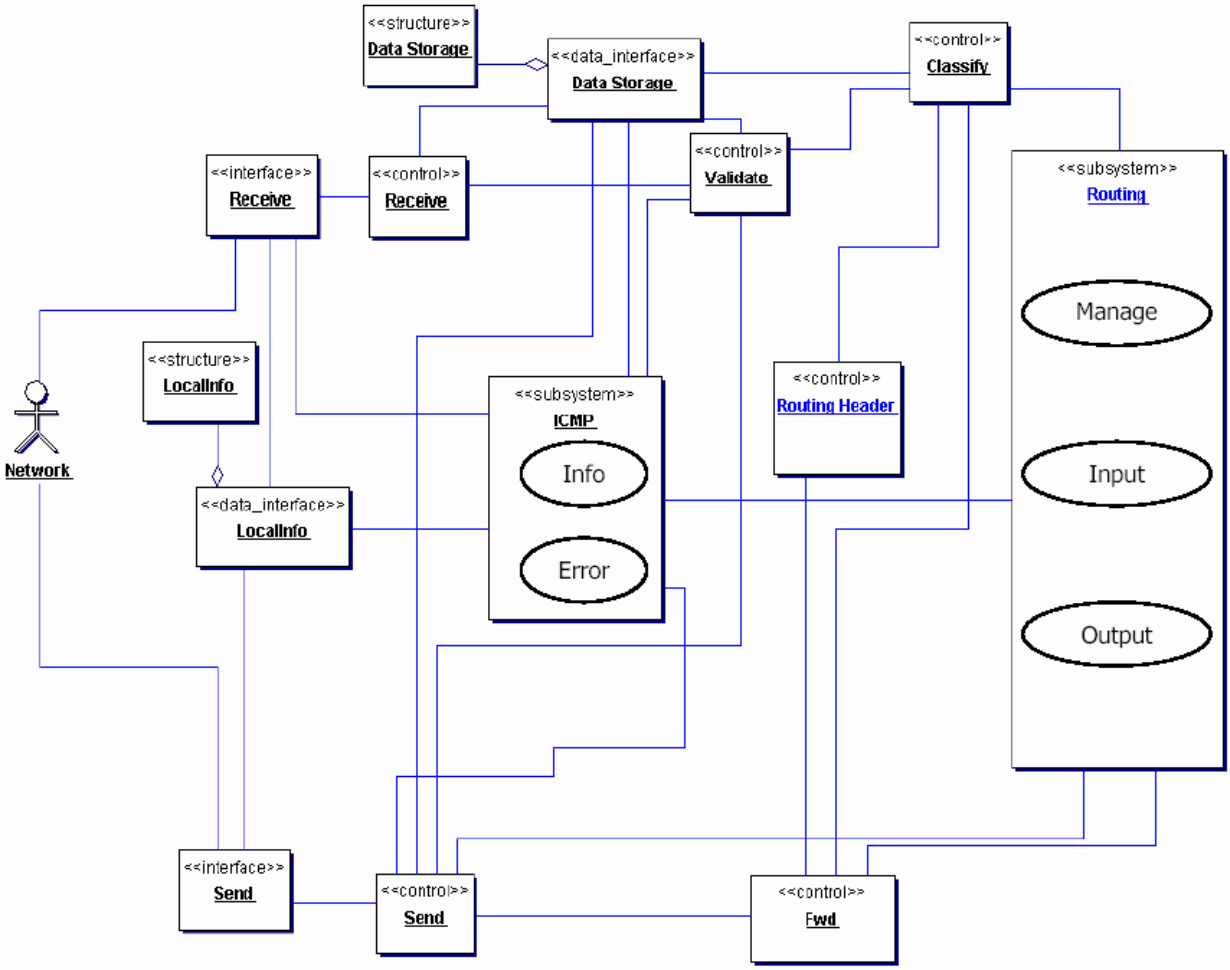


图 4 最后的对象图

3.2.3 建立系统层次上的对象协作图

直到现在，通讯只是从对象间的控制流程的角度上来定义的。由协作图表示的场景应用在不同系列的对象上，对象间的通讯被表示为服务请求（service request）。

UML 规约根据由一个场景实例化的用例来定义协作图，但是允许将由不同的场景产生的协作图组合成一个协作图。我们将以前建立的所有协作图组合成一个协作图，这就为整个系统展示出一个对象协作图。从每个协作图中复制至少一个对象的实例和它的请求（request）到最终的图中，这样就形成整个系统的协作图。在图 5 中，通过组合路由和转发这两个用例，我们展示出部分系统层次上的协作图。

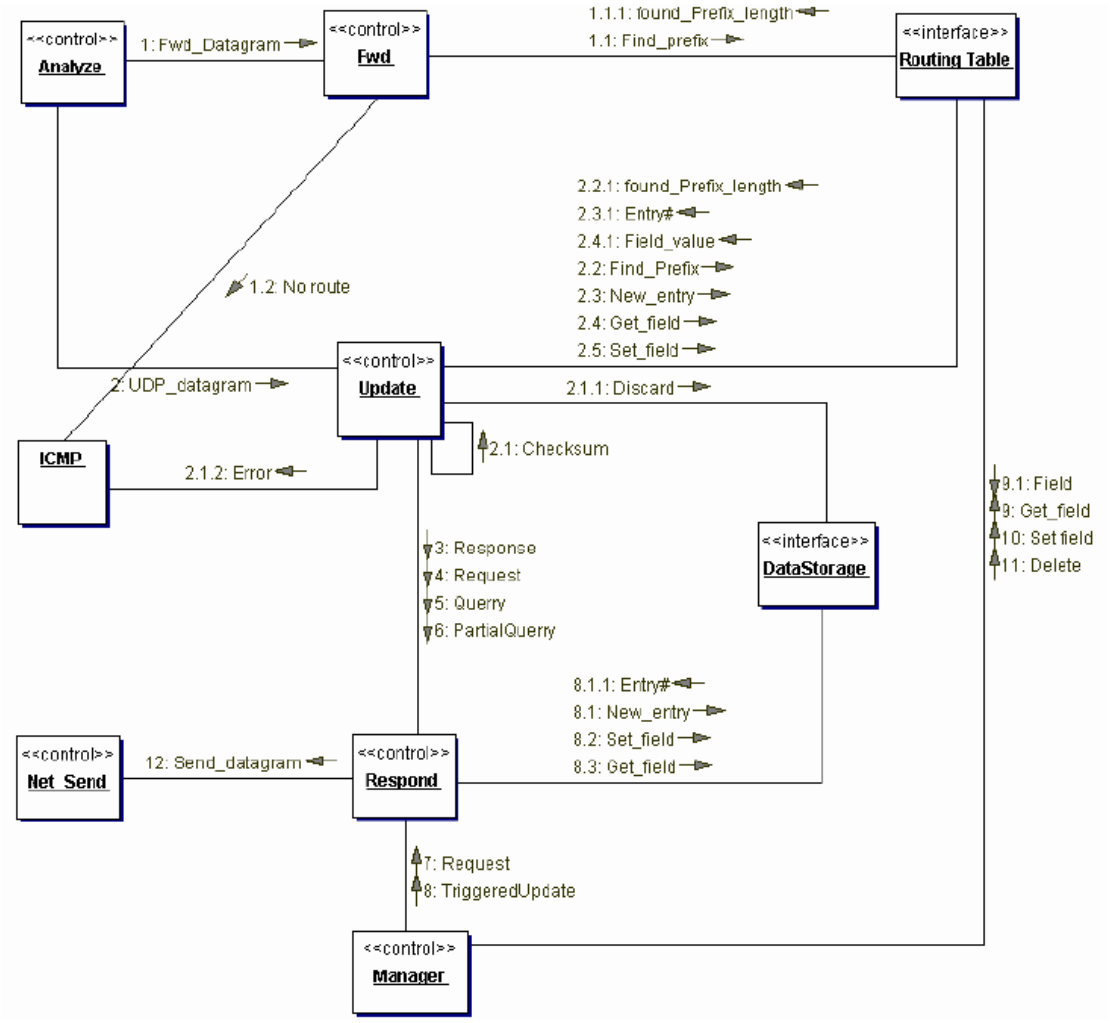


图 5 路由和转发用例组合后的协作图

为了在系统层次上的对象协作图内部精化（refine）其中的通讯，我们将请求信息转化成基于事件的通讯。UML 标准定义的两种事件类型是：信号（signal）和调用（call）。

信号是从一个对象异步（asynchronous）地发送到另一个对象的事件。调用常常是同步（synchronous）的事件，它代表一个对象操作的使用。两种类型的事件都可以有一个名字并且可以带一些参数。我们选择在一个对象发送一个消息的时候就产生一个调用事件（call event），控制权被转移到接收方，接收方改变它的状态，并且将控制返回给发送方。如果一个消息将控制从发送方转移到接收方但是并没有将控制返回，那么我们认为该消息是个信号（signal）。将事件分成信号和调用并没有减弱设计的通用性，因为一个调用可以被看成是两个信号，一个信号触发另一个对象的操作，另一个信号将控制从被触发的对象返回。因为 UML 为设计中的每个对象定义了输入事件队列（input event queues），这就使得上述情况变为可能；事件是以 FIFO（先入先出）的方式来处理的。

将事件分解成信号和调用帮助我们识别出具体是哪些对象提供了触发操作，并且识别出操作的签名（signature）。签名指操作的名称，操作所用参数的数目和类型，和消息的返回值（如果有的话）。

3.2.4 行为分析

在前面的小节里面,我们将系统分解成控制,数据和接口对象。控制对象用来协调系统内控制流程，执行一些算法或者控制系统各部分间的信息流。数据结构对象代表一个存储的地方，通过一些特定的方法，数据可以从中读取或者写入。对象被视为系统功能性的模块，它为环境提供接口并且有其内部的行为。

系统内部行为大多存在于协调系统控制流程的控制对象中。因为我们强调协议处理的领域，该领域的数据密度高，控制流程有一定的算法，我们因而选择活动图（activity diagram）来实现控制对象的行为。一个活动图（图 6）是一系列的履行原子计算（atomic computation）的动作(action)，这些动作的结果是引起系统状态的改变。每个活动是由一个转移（transition）触发并且执行直到完成。UML 没有预先规定必须用哪一种语言来描述活动，这就为设计者提供了极大的灵活性。活动图的另一个重要特性是可以分级地（hierarchically）被分解成子活动图（sub-activity diagrams）。这就允许设计者精化（refine）最初的活动图来获得更为详细的描述。

在一个图中有两种状态，动作状态（action states）和活动状态(activity states)。动作状态是可执行的原子操作，它不可以进一步分解。活动状态是可被分解（扩展）成另外的包含动作状态和活动状态或者只包含动作状态的活动图。有关活动图的更多的细节可以在文献[6]和[10]中找到。而且，活动图提供发送和接收事件的机制，还可以将控制流程同步（join 和 fork）。

在我们的设计中，我们为一个类（class）的每个方法（method）建立一个活动图，在该活动图中，所有的活

动在开始的时候都是活动状态。每个活动都用自然语言来描述。活动图有一个或多个输入/输出点，输入/输出点由发送/接收事件状态或者由图中的开始/结束状态来表示。这些点对应于早已在对象图中存在的通讯路径（communication path），代表另一个对象对一个操作的调用或者向外部环境发送消息。活动状态可以被分解成其他的活动图，该图中包含系统较小部分的功能但是却更为详细。这个过程可以被使用多次直到将所有的活动状态转换成动作状态。

在活动图的精化中，因为我们描述越来越详细的系统的视图，我们需要能够用到系统中的数据和对它们的操作。识别数据实体（data entities）和它们的结构已经是需求规约和系统中数据结构对象的主要输入条件。数据需要根据它所提供的功能来进行描述，而不需要考虑其在物理上是如何实现的。我们感兴趣于一个特殊的数据域的意义和它的大小，而非其在物理上是如何表示和存储的。因为动作状态的描述不需要一种严格的语言，应用于数据上的操作的定义不需要遵循严格的句法，但是它们的定义不可以互相造成歧义。实际上，我们强烈建议系统有一个数据字典（data dictionary）。将活动状态进行多次精化后，每个活动图最终的版本将仅仅包含有活动状态。在当前，因为所有的动作状态相似于编程语言的指令，我们已经获得了一个路由的可执行规约，每个动作状态代表一个作用于一个或多个操作数（operand）的操作，操作数是指数据结构的域。因为我们讨论可以在不同大小的数据上应用的泛型（generic）操作,在操作的附近，我们标识出操作数的数据大小。

图 6 描述了图 4 中的对象 RH（Routing Header,路由头）的活动图。为了提供一个更好的可读性，我们对有条件的活动使用了一个新的概念。一个有条件的执行被表示成一个检测（test），在该检测之前，有一个估算（evaluate）一个条件的值活动状态.。在从一个初始活动图到动作状态的精化过程中，功能性的数据类型已经被确定。它们当前并无具体的类型，它们只是有了功能性的对于它们大小的要求。对于暂时的功能性数据值的存储，我们使用变量，但是并无类型或大小的限定。在对象间的通讯被描述成消息传递协议（message passing protocol），通过事件状态，参数可以被传送到相邻的对象中。

活动图中细节描述的层次是由它的动作状态的粒度（granularity）给出的。活动图的定义方式有助于将复杂的运算分解成复杂度低的，容易处理的部分。因为活动的分级表示，人们可以在不同的细节的层次上扩展或者减小活动，并且重复使用动作状态来实现不同的活动。

3.2.5 建立类图

在系统层面的协作图中，我们定义了一系列的对象，它们在服务的层面（service-level）通过由起点到终点发送的事件进行通讯。我们的目标是识别出一系列的类，这些类被实例化，并且通过它们的操作确定它们必须向外界环境提供的功能。

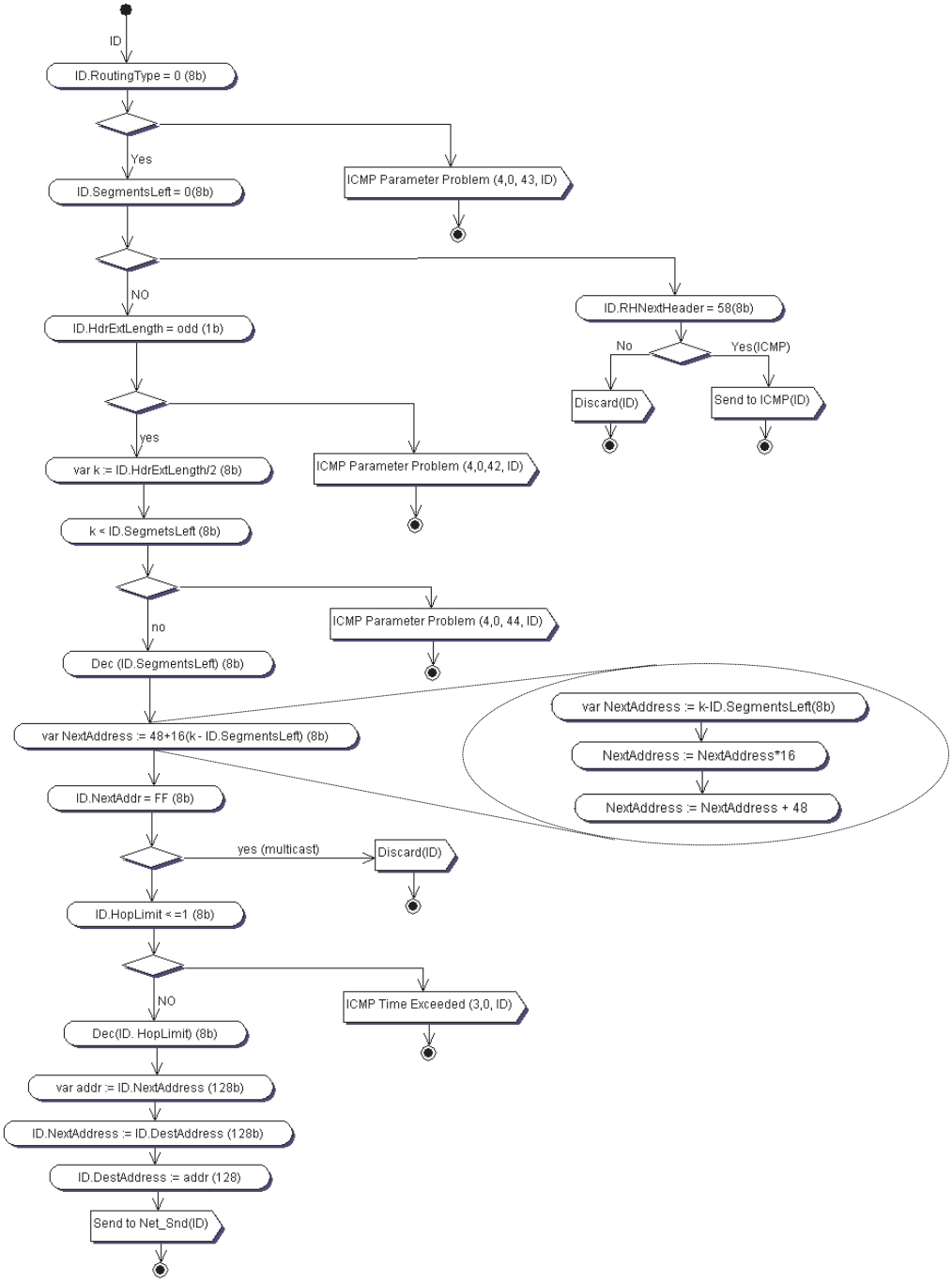

ID
ID.RoutingType = 0 (8b)
Yes
ICMP Parameter Problem (4,0, 43, ID)
ID.SegmentsLeft = 0(8b)
NO
ID.RHNextHeader = 58(8b)
No
Yes(ICMP)
Discard(ID)
Send to ICMP(ID)
ID.HdrExtLength = odd (1b)
yes
var k := ID.HdrExtLength/2 (8b)
ICMP Parameter Problem (4,0,42, ID)
k < ID.SegmetsLeft (8b)
no
ICMP Parameter Problem (4,0, 44, ID)
Dec (ID.SegmentsLeft) (8b)
var NextAddress := 48+16(k - ID.SegmentsLeft) (8b)
var NextAddress := k-ID.SegmentsLeft(8b)
NextAddress := NextAddress*16
NextAddress := NextAddress + 48
ID.NextAddr = FF (8b)
yes (multicast)
Discard(ID)
ID.HopLimit < =1 (8b)
NO
ICMP Time Exceeded (3,0, ID)
Dec(ID. HopLimit) (8b)
var addr := ID.NextAddress (128b)
ID.NextAddress := ID.DestAddress (128b)
ID.DestAddress := addr (128)
Send to Net_Snd(ID)

图 6 RH对象的活动图

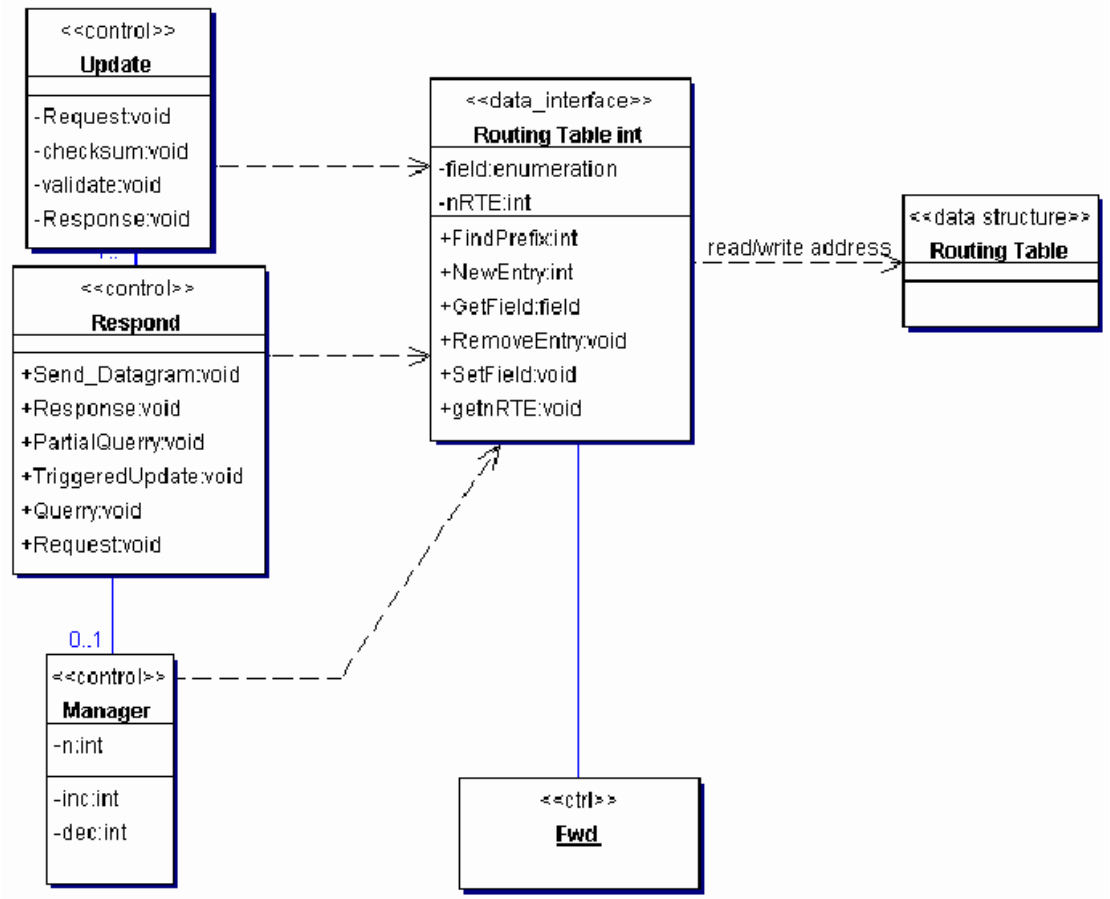


图 7 路由表子系统的类图

因为对象代表类的实例，调用（call）事件代表使用对象的操作，我们可以识别系统中的类和它们的部分方法（method）。每个在协作图中的对象被转化成一个类，每个调用事件被转化成类的一个方法。方法需要放置在事件接收者的类中。

属性代表了类的状态，并且仅可以通过方法被外部的对象可见。在一个类里面，一个属性可以被很多方法共用。人们需要注意到类里面会有隐含的方法和属性，它们不能直接从协作图中被识别出来，但是通过类的个别功能的分析可以获取。

在图 7 中，我们展示了已经被识别出来的路由表接口对象的操作，以及它们被相邻类使用的方式。路由表接口类提供了一系列的操作，通过这些操作其它的类可以调用路由表数据类中的数据。当前，我们只是对路由表数据接口感兴趣，而非它是如何实现的。接口类不包含任何控制（也不包含状态机，state machines）,只包含实现接口的方法。

3.3 领域分析阶段

通常人们都承认在相同领域中，不同应用之间有一个共同点，而且已经有了几种途径着重于面向重用的领域分析(reuse-oriented domain analysis)(例如,文献[5,13]).通过对领域知识进行分析和建模,通过识别一系列应用总体上的抽象性和相似性,一些泛型(generic)的可重用的组件可以被提取出来。对于一定范围内的应用，例如，协议处理，我们可以说存在着一系列的基本操作，它们需要被该系统提供和实现。

基于先前的设计经验，通过逐渐增加的领域分析，设计者可以获取一系列特定的基本操作。在新系统的设计中，人们可以非常容易地建立一个必须要有的基本操作的列表。这个列表包括了功能性的数据类型和对于这些功能性数据的一系列基本操作。稍后，数据类型会被精化成针对于实际应用的特定数据（平台有关的）,而领域操作（domain operation）将会被映射成硬件平台所提供的操作。领域操作是通过它们要完成的功能，它们需要的参数的数目，它们使用的泛型数据类型的宽度来定义的。对列表中的操作，我们可以将它们分成算术的（加法，乘法），逻辑的（与，或，非），测试（match, compare）的或者内存处理（get, put 等）等几类操作。

我们使用领域操作列表（domain operation list）作为一个在应用的 UML 规约和 TACO 处理过程平台之间的接口，以帮助将规约转化成处理器提供的资源。一方面，我们努力使用可以获取的领域操作来表述 UML 的规约（活动图），另一方面，FU(功能单元)或者它们的组合又为领域操作提供支持。为这一点，根据每个操作提供的参数的数目和每个操作提供的功能，我们开始精化活动图，以使它仅仅包含由领域列表（domain list）中的操作表示的动作状态（action state）。

实际中会有直接的映射不可能的情况，或者设计者在目标问题领域会识别出有用的新的操作。作为结果，新的领域操作必须被建立，它们可以被包含和记录在操作列表中。

同时，为新的领域操作增加硬件支持意味着要么改变一个现有的 TACO FU 或者建立一个新的 FU，并且在领域操作列表中的操作要被更新。例如，从映射活动状态到领域操作，一个新的 shift 操作或者一个新的 set 操作（根据一些 mask,在一个位字符串中对位的选择）产生了。最终，两个新的功能性单元被设计出来：Shifter Unit（移动单元）和 Set Unit(设置单元)。

3.3.1 识别领域操作

TACO 构架中有许多在以前的应用实践中建立的类型的资源，类似于一个组件的馆藏（library）。根据它的规约，每个特定的功能单元可以履行一个或多个操作。TACO 实现的操作可以被看成是一个领域操作的列表，它们被定义成在先前为一个应用领域的而进行的配置（或者具体实施）中识别出的所有操作的并集。在我们的情况中，指的是网络领域（协议）处理应用。对不同的应用领域我们建立了不同的列表。

而且，根据数据总线的带宽，会有超过一种的 TACO 处理器架构的存在。因为 FU 是通过输入/输出存储器（input/output register）和数据总线相接的，这就意味着，对于给定的数据总线带宽,输入/输出存储器应该有同样的大小。TACO 处理器在同一时间只支持一种大小的数据总线，那么配置必须根据一个特定的架构（数据总线带宽）来完成。每个架构有其自身的资源列表（FU,数据总线，插槽，互联控制器）并且建立一个独立的领域操作的列表。

3.3.2 将领域操作映射到硬件平台

因为 TACO 处理器资源在 SystemC 中是使用面向对象的机制来被模拟的，我们通过建立一个 TACO 的类图（图 8 的上部）来展示可用的资源。这里我们有三种资源：

数据总线，插槽（sockets）和模块。一个 Socket Manager(插槽管理器)类对插槽和功能单元的连接进行管理，NetControl（网络控制）类实现数据总线上的通讯。所有这些类是从 SC_Module 这个类继承而来的，SC_Module 定义了在 SystemC 中基本的模块描述类。对于一个特定的架构来配置 TACO 意味着首先要选定数据总线的带宽。针对于数据总线的带宽插槽和存储器的大小会自动被设置。

TACO 支持的功能存在于它的功能单元的行为中。数据总线仅仅用来在功能单元之间传输数据并且间接地触发功能单元的操作。每个 FU 有一系列的输入和输出存储器，这是 FU 和外部网络相联结的接口。我们可以将每个 FU 看作是一个黑盒，它接收输入信息，履行一些计算，并且输出结果。并且，一个功能单元实现一个或多个处理器的操作。例如，COUNTER FU （记数功能单元）被设计来做加法，减法，递增，递减和计时（计算数据总线周期）等运算。

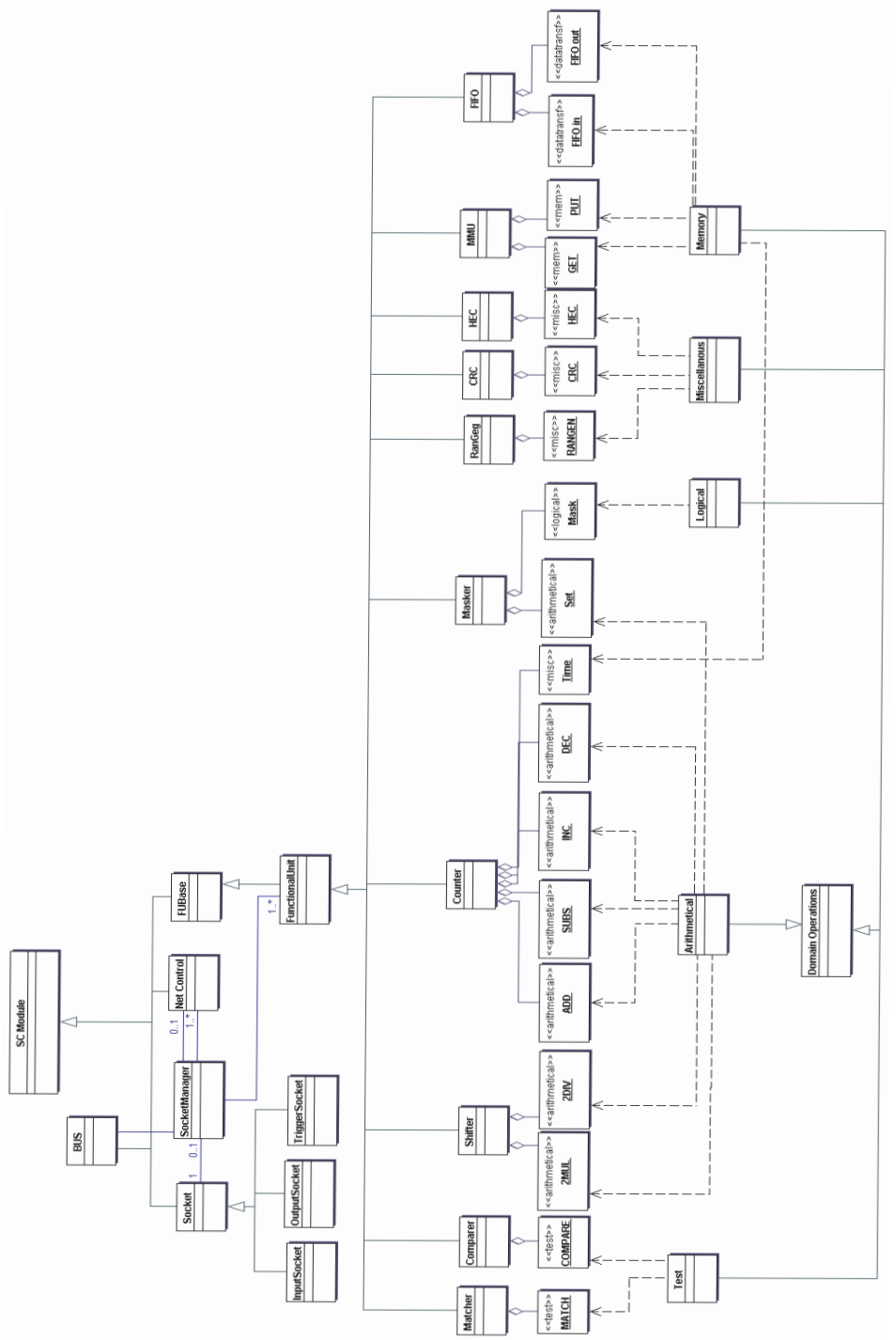


图8 路由表子系统的类图

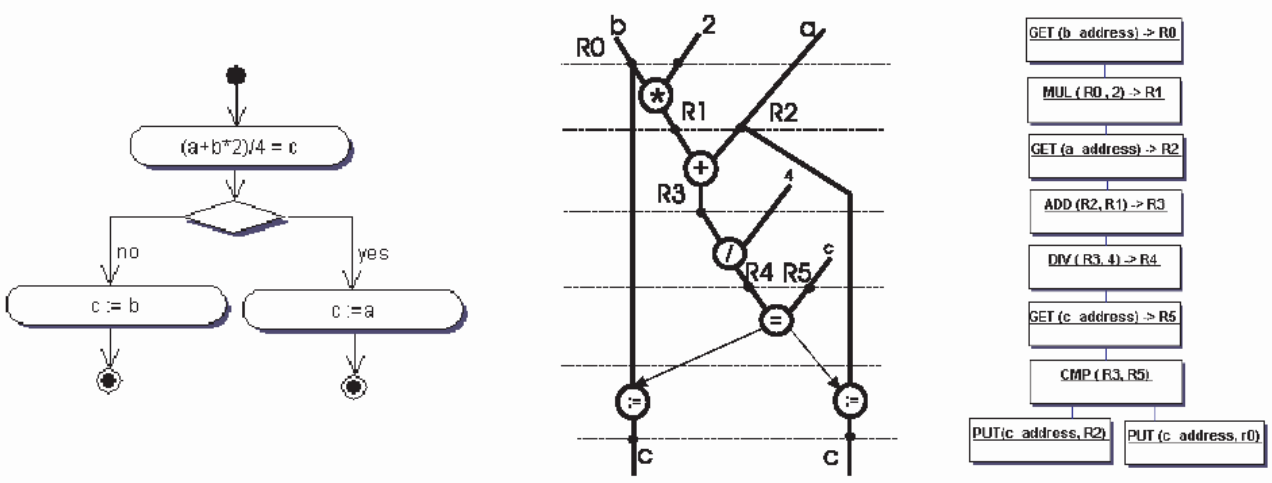


图 9 将活动映射到领域操作

为执行一个 TACO 操作，我们需要一个或多个的数据传输，因为一个功能单元常常必须先要设置好输入值，被触发，然后接着等候结果。这就意味着一个领域操作是等同于一些在处理器的 FU 之间运用的 TACO 的 move 指令。人们应该注意到，对于一个特定的 FU 履行的操作，每次功能单元被触发后，相同序列的 move 指令被执行。这样一来，我们可以将领域操作抽象成一个宏指令，它包含了 TACO 数据总线上的传输（将涉及到的起始和终点的 FU 作为参数）。

作为一个结果，每次执行一个领域操作，该操作就自动被转化成一个 TACO 数据总线的传输。在领域操作列表中，每个领域操作被指定有一个 TACO 操作（包括它的泛型参数的列表）。我们注意到列表中的领域操作可以通过一个 FU 或者多个 FU 的组合来实现。

3.3.3 将 UML 规约映射到领域操作

直到现在 IPv6 路由器的设计完全是独立于领域，没有任何有关领域或者平台的的设计决定。在下一步的设计流程中，我们要将在前面设计阶段获取的活动图中的状态，和可以从操作列表中获取的操作相匹配。这种的映射的过程在图 9 中有解释。这里，一个简单的算术表达式被转化成一系列的领域操作。

为了简化使用领域操作来精化 UML 规约的过程，我们基于领域操作的功能（算术，逻辑，内存，测试，等）来对它们进行分类。我们对这些操作建立了一个层次结构，建立了一个类图，在类图中，原型被应用于每个操作的分类。为了获取领域操作列表的一个更加正规的模型，领域操作的类图可以很容易地通过关联关系（association）和 TACO SystemC 的类图（图 8，下）相联。

因为在活动图中的原子操作是一个问题领域（例如，协议处理）中的操作，领域映射转换的结果是跟领域相关的。因为我们尚未决定在一个具体的 TACO 处理器上，领域操作是如何实现的，我们所建立的模型还是和架构无关的。正如我们在这篇报告中先前所提到的，一个领域操作可以被不止一个 FU 来实现。通过在应用的规约中选择一个要使用的领域操作，并不就是意味着某一个的 FU 将会被使用，只是说明了，我们手边可以获取的领域操作是“可以实现的”（implementable）。

3.4 平台的实现

3.4.1 定性配置

一旦整个规约被映射到领域操作，我们就可以浏览应用的规约，并且选择用于实现要求的领域操作的功能单元的类型。我们再次强调我们仅仅识别那些必须用于实现一个给定应用的 FU 的类型，而不是处理器中每个 FU 类型的实例的数目。当整个应用可以被映射到物理资源上，我们就可以用每个类型的资源配置处理器并且进行整个系统的功能性验证。

3.4.2 建议 FU 的最优化

尽管我们已经将活动图精化成仅仅包含领域操作，它们还是用活动状态来表述。

现在，我们考察一个需要大计算量的递增的操作的序列，目的是为了优化它们。

通过定义，活动图代表系统的控制流程。状态按照一定的顺序成功，在状态之间有转移（transition）。我们可以将成功的状态组合成一个控制块（blocks of control）。当遇到一个开始状态（start state），或者前面的控制块正好结束，控制块就开始了，当遇到一个停止状态（stop state）或者控制流程有分叉的时候，控制块就结束。控制块和它们之间的转移形成一个控制图，在该控制图中，控制块是用弧线来表示，而节点是用对控制流程进行分叉的状态来表示。

而且，每个活动图在动态的情况下能被看作是一棵树（tree），树的节点是分支状态（branch state），弧线包含控制块。控制流从树的根开始，结束于一个或多个叶子上。控制流一路上经过了一些分支（或决定，decision）状态和控制块。为了每个活动图获得最佳的控制流程的性能，我们努力来优化控制块以使得树的每个层次上都有相似的（平衡的）复杂度。

由于领域运算是通过早已存在的功能单元来实现的，我们可以知道对于每个领域操作所消耗的处理器周期，因而得到树中每条弧线的耗费（cost）。为计算领域操作基于实现它们的 FU 的特性，两种计时（timing）必须被考虑。一个是 FU 输入存储器的 setup time(设立时间)，另外一个是直到运算结果在结果存储器可以被获取之前功能单元的运算时间。在 TACO 处理器里面，每条数据总线的传输是在一个总线周期中完成的。如果有多于一条的数据总线，传输就可以并行完成，如此就减少了 FU 的 setup time。运算时间是独立于数据总线的数目的，通过增加数据总线并不能减少它。在领域列表中，我们同样标识出每个操作在一个 FU 上需要完成的周期数目（图 10）。这就允许我们估算不同的控制线程在执行期间所需周期。

从不同的活动出发来组合动作状态可以给我们带来一种新的实现相同功能的方法。例如，对一个数连着进行两次的除法，每次都除以 2，我们就可以组合两次除法形成一个单一的除法运算（除以 4）或者对这个数进行一次逻辑 shift 运算。这种情形在最终的实现中会有一个好的效果因为它减少了应用所需周期数目。我们努力来识别出如下所述的操作的序列：

- 是控制流程中回路（loop）的一部分,并且需要实时的长时间运算。
- 在同一个或不同的活动图中，它们在功能上是相同的。

在第一种情况下，我们分析一个较大的连续状态的序列，该序列在控制流程中会使用许多处理器周期，特别是当它们是有很多迭代回路的一部分的时候。在实际运行时，回路会变成较长的重复的操作序列。我们识别出这些序列并且将它们标识为优化的对象。识别出来的序列实际上是新的活动图（只包含一个序列化的控制流）并且成为进一步研究的对象。识别重复的操作序列通常情况下需要较大的算法复杂度，在这里，我们并没有说我们可以把上述方法应用于任何问题。

在第二种情况下，在对系统中所有的活动图进行分析的时候，人们可以注意到有这样的状态序列，它们在许多活动图中，或者在同一个活动图中，重复出现（该序列没有形成回路）。这些序列指明了一个平台必须有效处理的特殊的功能（模式）。通常，我们在不同的活动图中识别出这种序列。我们同样也将找到的序列标识出来以用于进一步的研究。

W1 – 16 bit	W2 – 16 bit

Domain Operation	FU	SetUp Time	Computation Time
SHIFTR (W, 16, R1) *//shift W right to obtain W1*	SHIFTER	1c	1c
MASK (W, 0xFFFF 0000, 0, R2) *//set to 0 left part of W*	MASKER	2c	1c
ADD (R1, R2, R3) *//compute 2's complement sum*	COUNTER	1c	1c
SHIFTR (R3, 16, R4) *//get carry in R4*	SHIFTER	1c	1c
ADD (R3, R4, R5) *//compute 1's complement sum* *// by adding carry*	COUNTER	1c	1c
SUB (0xFFFF FFFF, R5, R6) *//complement the sum*	COUNTER	1c	1c
	Total time 1 bus: Total time 3 buses:	14 cycles 5 cycles	

图 10 校验码的最优化

所有候选的序列有同样的特点：他们是由一个状态的序列（在内部没有流程分支）组成的，并且代表了部分常应用的功能点。人们可以注意到，相邻的两个动作状态可以是从不同的起始状态产生。活动图的定义允许将它们重组成新的活动状态，该新的活动状态包含有从不止一个相邻活动中来的领域操作。重组是根据状态需要履行的功能来完成的，它极大地依赖于设计者的经验和关于实际操作平台的知识。新形成的活动引入了较大块的需要特殊注意的功能点。重组或者分解动作状态和软件中建立一个函数，或者硬件中建立一个新的特定模块相类似。

例如通过对整个设计的分析我们在 Input Controller(输入控制器)这个类中识别出了 Checksum(校验码)这个方法，在 Routing Table Interface(路由表接口)这个类中识别出了 Find 这个方法，而这些方法可以被优化（属于第二种类型的优化）。IPv6 的校验码的运算是这样的：“16 位的补的和”，它需要重复的加法和 1 的补运算。例如，如果我们使用 TACO32 的架构，我们会使用 32 位的数据总线和存储器。为了计算 32 位的字的校验码，我们需要将它分解成两个 16 位的字，以计算两个数字的 1 的补的和，并对结果进行补运算。这将需要 14 个处理器周期（图 10）（如果使用一条数据总线）或者 5 个处理器周期（如果使用三条总线）。我们已经引入并实现了一个新的功能

单元，它可以在一个处理器周期内计算 32 位字的校验码。设计校验码的功能单元和它的架构完全是硬件设计师的工作。而且，新引入的功能单元配置有 3 个输入和一个输出存储器。这将允许在一个处理器周期内（当使用 3 个 32 位的总线）计算三个 32 位的字。我们在设计和实现该资源上的花费和增加性能这两方面之间权衡，以得到合理的结果。

另一个重要的需要优化的是在路由表中寻找一项内容。尽管 Find 是 Routing Table 这个类的一个方法，它也被描述成一个包含领域操作的活动。它的功能是搜寻 Routing Table 以获得正确的前缀。这个方法决定了该路由器的性能。Find 操作所引发的一系列操作也是优化的对象。

3.4.3 定量的配置---领域空间研究

我们已经识别了，接着改变了或者说建立了所有的 TACO 处理器为实现应用功能上的需求所必须的资源。为获取一个在一定的限制条件下性能最理想化的架构，我们必须通过引入不同的架构的配置来研究设计空间。这包括了物理性能的估算，相应的方法在文献[20]中可见。作为这个优化阶段的结果，新的功能性单元会被引入。这些功能性单元会被加入到 TACO 处理器馆藏中，FU 要履行的操作必须被包含在领域操作列表中。

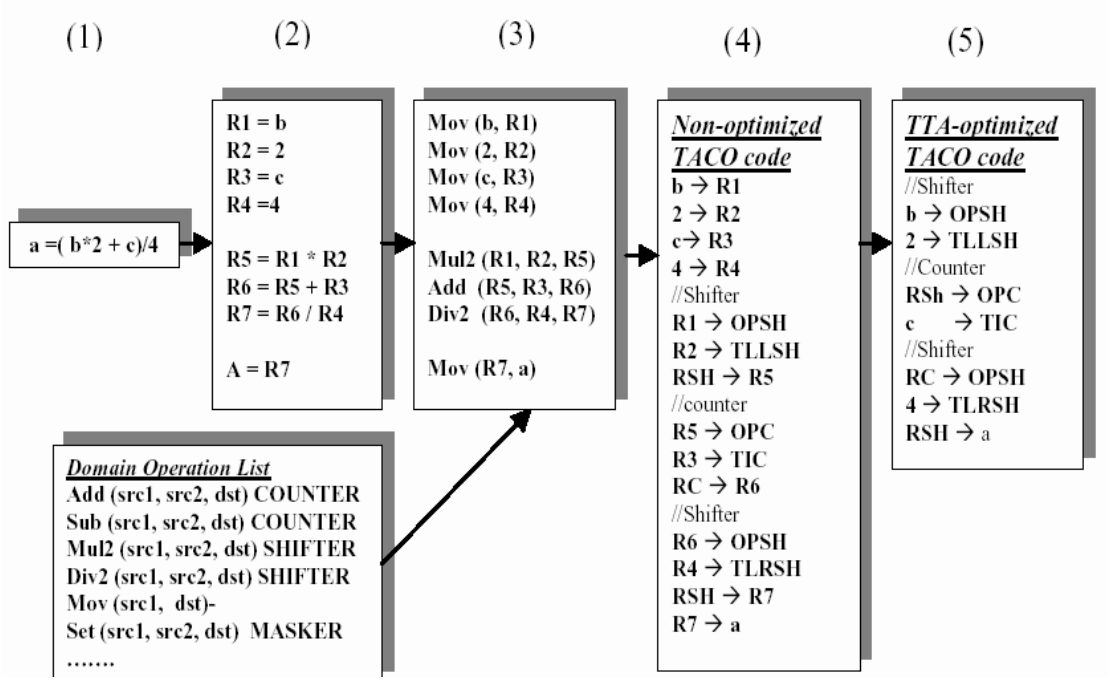


图 11 映射过程

为了获得最优的解决方案来研究所有可能的配置需要大量的计算工作。通过对配置定性的分析，一些初步的信息可以被提取出来，例如，在处理中的瓶颈以及表面积和能量消耗的信息。为 TACO 处理器进行设计空间研究意味着通过在不同配置下进行模拟实验来估算架构的物理性能。根据一些启发，通过增加不同类型资源的数目，我们获得了当前的配置。对每种配置，我们将活动图中的动作转换成平实的 TACO 代码。因为活动图的特点，这样的转换过程是非常直接的。活动状态事实上是 TACO 指令（数据总线传输）的集合，并且它们之间的转移可以通过相对或绝对的（如果需要）JUMP 操作来实现。

3.4.4 产生应用代码

迄今为止我们是通过一系列的领域操作来实现应用的，这些领域操作是可以映射到 TACO 指令中带参数的宏指令。为了得到一个配置的 TACO 可执行代码，我们在每个活动图中使用了一系列的领域操作的转换。一个简短的例子可见于图 11。

1. 我们在活动状态中给操作数（operand）分配泛型的存储器。泛型的存储器仅仅是临时用于操作数的值的存储，它们并不代表物理资源。

2. 我们将活动状态分解成基本的操作，在领域操作过程中，可以被实现。

3. 每个活动状态映射到一领域操作上，并且泛型的存储器被指派给相应的领域操作参数。

4. 领域操作被扩展成在 TACO 功能单元（或插槽）之间 move 操作的宏指令。

5. 最后结果产生的 TACO 汇编代码是根据 TTA 架构来进行优化的。优化包括：将操作数从一个输出存储器移动到一个输入存储器可以不需要经过附加的临时存储空间，使用同一个输出存储器或通用目的存储器来进行数据的多重传输（操作数共享），将不使用的存储器移开等等。所有上述的技巧加速了执行时间，减小了应用的代码量，同样减少了应用所需通用存储器的数量。而且，通用的汇编优化同样可以应用于优化了的 TTA 汇编代码，例如：sinking(下陷)，loop unrolling(回路的解开)等。

为实现规约中不同功能对象的活动图间的通讯，我们将每个事件转换成 jump(跳转)操作。对于参数的传送，我们将参数值存储在泛型存储器中。TACO 架构描述了两种类型的 jump 指令：相对和绝对的跳转。更详细内容，在文献[22]中。

整个设计现在削减为数据总线时序的调度（bus scheduling）和存储器分配问题，在此，我们将泛型存储器分配给物理资源。为此，我们必须排列在数据总线上的 move 指令，并且将存储器分配给指令的操作数（operand）。在文献中，调度（scheduling）和分配（allocation）的规则有被广泛的讨论，我们在此不引入任何方法。编译器（compiler）做些必须的分配和调度，最后再做些优化。

3.5 一个 TACO IPv6 路由的定性配置

接下来我们给出一个在设计流程中开发出来的 IPv6 路由的定性配置。路由器是由一些完成诸如输入/输出处理，算术和逻辑运算任务的功能单元和其他提供数据处理的单元（见图 12）组成的。TACO IPv6 路由器的实现细节可见于文献[22]。

TACO 处理器作为一个独立的处理器为 IPv6 层提供支持。处理器必须通过缓存器（buffer）和外部网络的接口相接。每个接口有一个输入缓存器和一个输出缓存器，并且它们可以独立地接收和发送数据包。当一个数据包在其中的一个输入缓存器被接收，它被储存在主内存中，处理器开始处理它。设计两个新的功能性单元，以管理输入和输出数据的流量。

处理单元（Processing Unit, iPPU）对输入缓存器进行扫描以获取新数据包的信息。如果一个数据包的状态是有待（pending），它被存储在主内存中。有指针指向相应的数据包存放的内存地址，这些指针和输入存储器的接口的标识符（identifier）一起被存放在一个队列（queue）中，iPPU 和在处理器的互联网络中的数据总线相连，并且提供 1 位和互联网络控制器相联的信号，来提示在队列中有数据包是有待（pending）的。后处理单元（PostProcessing Unit,oPPU ）管理路由器的输出数据流量。该单元包含有一个内在的队列，队列中存有指针，而指针指向相应的将要发送的数据包的内存地址，这些指针和输出存储器的接口标识符（identifier）一起被存放在该队列（queue）中。后处理单元查询它的内部队列，对应于队列中的每一项，它将相应内存中的数据包移动到特定的输出缓存器中。

一些算术和逻辑单元支持数据包的处理。Counter Unit(计数器单元)履行算术操作（增加，递减，加法，减法）和计数（从一个起始值向上或向下计数直到一个停止的值）。

当到达停止的值的时候，一个直接和网络控制器相联的结果信号被激活了。为了将一个操作数和给定的值相比较，我们设计了一个 Comparer Unit(比较单元)。比较的结果以结果信号的方式发送给网络控制器。Matcher(配比器)和 Masker(遮蔽器)是进行位子符串操作的功能单元，它们根据一个给定的 mask 只对它们的部分输入操作数进行处理。Matcher 通过一个直接和互联网络控制器相联的结果位信号（result bit signal），将结果汇报给互联网络控

制器。Masker 根据一个给定的 mask 和给定的值来对存储器的位进行设置。不仅仅可以进行逻辑的位的移动，一个 Shifter 能够进行乘以 2 的算术乘法。路由器也有一系列的 FU，它们提供对数据存储的快速处理（路由表和当地信息，Routing Table 和 Local Info），有一个用来读写内存数据的内存管理单元。

其中的一个 IPv6 必须实现的复杂操作是校验码的计算（checksum calculation），因为它包含了大量的数据传输。从分析过程中我们归纳出：为增加应用的性能，一个特殊的功能性单元是必须的。它通过连续不断地输入小块小块的数据包来计算数据包的校验码，这样就减少了在数据总线上数据传输的数目，同时减少了内存处理的次数。

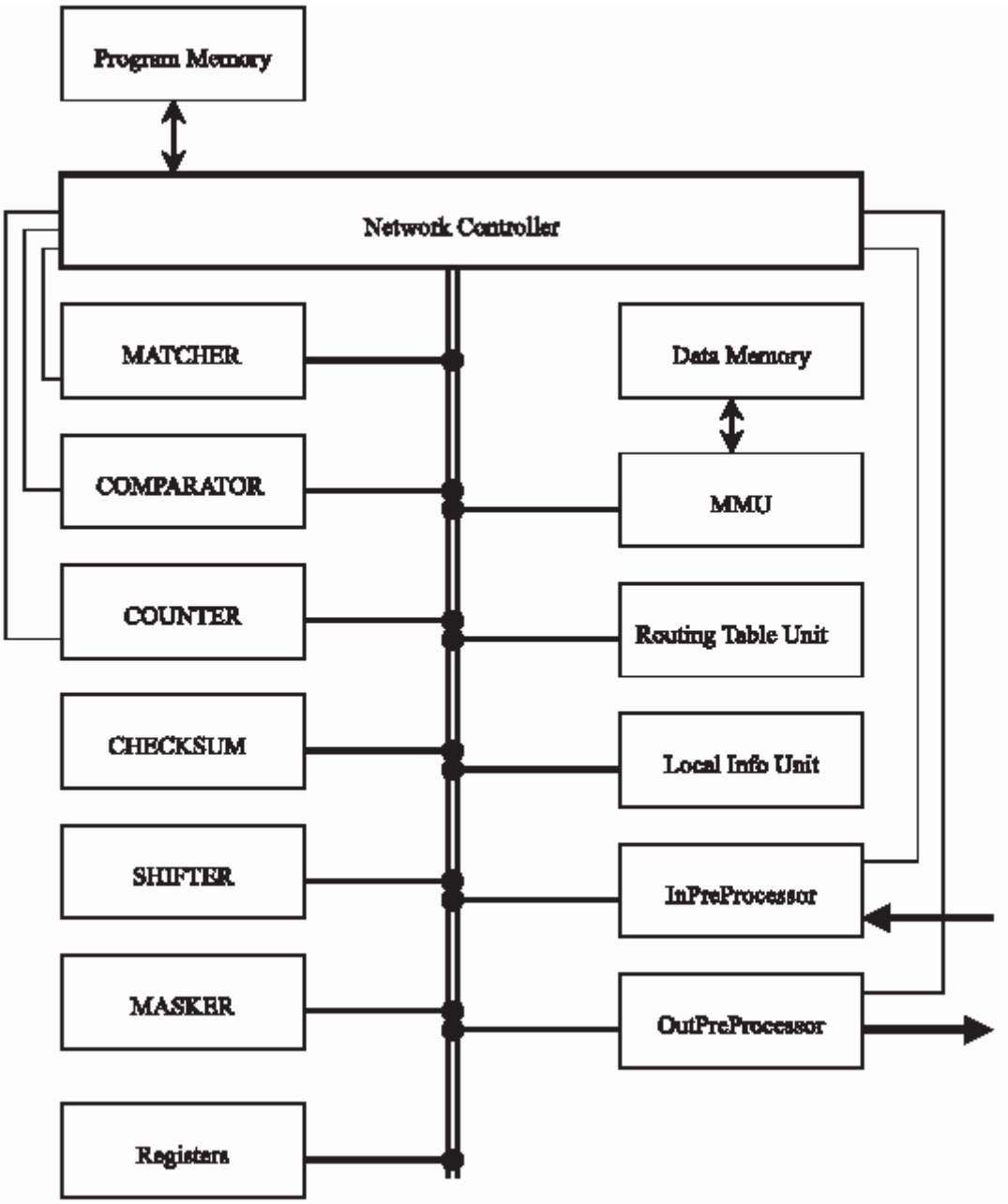


图 12　TACO 处理器的 IPv6 路由器实例

一个重要的 TTA 处理器的设计特点是架构的模块化。每个 FU 独立于互联的网络和其它的功能单元进行计算。于是处理器的性能可以从数据总线上的传输数量来反映,或者间接地由功能单元的输出存储器中的操作数变成可用的时间反映出来。为了减少等待时间，FU 必须在尽可能少的处理器周期内完成它们的工作。在这个层面上，我们必须在复杂度和功能单元的反应时间之间达到一个平衡点。TACO 的功能单元已被设计成可以在一个周期内完成它们的计算。我们注意到，一个周期恰好是在数据总线上一个传输完成的时间。

4 可跟踪性事件

可跟踪性在设计复杂系统的时候是一个重要的事件。它帮助我们决定系统的一个特定的部分是如何建立的，是为了完成什么样的功能。可跟踪性有助于查找程序中的错误，有助于对复杂系统进行逆向工程（reverse engineering）。

在系统规约中，我们从需求规约开始，决定了用例，识别了对象，接着将它们精化到最后的规约。在识别对象的过程中，我们将它们重复地分解组合成较小或较大的功能。如果在某一个时刻，我们必须跟踪一个对象的功能，或者增加额外的功能，我们需要一个机制以使得我们知道每个架构的元素所属的最初始功能点。UML 标准定义了不可见超链接（invisible hyperlink）的概念作为一个建议使用的 UML 工具中的概念性的元素，但是并没有建议如何用工具来实现。超链接功能可以浏览一个项目的元素（对象，类，图等）以决定它们是如何相互联系的。在架构性元素之间的联结必须在设计者的定义时间（specification time）里面手工完成。

为了获得每个设计步骤和决定的历史，在设计的不同阶段，我们使用超链接。从用例图开始。当每次将一个用例分解成三个对象后，我们将获得的对象超链接到初始的用例。超链接是一个单向的链接，为了能够在设计中不同的方向自如地搜寻，我们也从用例到由它创建的对象间绘制了一个超链接。在对象识别阶段，对象可以根据它们的功能被组合或分解。如果两个对象被组合了，那么新定义的对象会包含初始对象中所有的超链接。如果一个对象被分解，那么新产生的对象会包含初始对象中所有的超链接。

识别了对象，并且定义了它们之间的通讯，一个类图就从协作图中被建立起来。每个对象代表一个类的实例。当我们为对象定义类的时候，我们也在对象和类之间以及类和相应的对象之间建立超链接。在类中，系统的功能是通过用活动图表示的方法（method）来表达的。活动图是由活动（activities）组成的，最后，这些活动转换成领域操作。因为整个设计是分级表示的，超链接可以使用在每一个定义的阶段，来显示不同工件（artifact）的来源。

我们现在所用的设计流程是从上到下的结构，从一个高层次的定义开始，结束于一个详尽的描述。每个阶段使用先前阶段建立的图，并且建立一系列新的工件。通过使用超链接，人们可以非常容易跟踪到在具体哪一个设计的步骤，一个工件被建立，从哪一个图中，该工件衍生出来，还可以跟踪到该工件在后续的设计步骤中在什么样的图中被使用。

5 结论

我们已经给出了一个基于 UML 的协议处理应用的设计方法，对于给定的应用，建议和配置了基于 TTA 的处理器的构架。应用是通过配置硬件并且同时建立软件来开发的。

这样的途径对于一个给定的应用领域，允许早期的 TTA 处理器配置的设计空间的探索。相应地通过使用 SystemC 和 Matlab 模型，在实际的硬件组件应用之前，在系统的层面上，可以获得配置的模拟和估算。

通过识别它们的领域操作列表，这样的设计方法可以应用到不同的处理器架构类型。关于应用的定义完全独立于平台这样的事实有助于探寻架构方面的多样性。同时，通过建立领域操作列表，分析人员不需要对他所将要应用的硬件资源有很好的了解。

重用性也在这个方法中被强调。IP-reuse(IP-重用)对于设计嵌入式系统变得越来越重要。它减少了硬件设计和过程实现的费用，缩短了整个系统开发的时间。从前面的应用中，TACO 提供了一系列随时可供综合使用的资源，这些资源根据应用的速度，硬件平台能量的消耗和芯片表面积的使用，可以被模拟和估算。

将来的工作要将重点放在如何对设计流程给出一个正式的描述，允许在早期阶段对规约进行功能性验证。由于操作图和 TACO 可执行代码间的相似性，对于一个给定的应用，我们需要设计一种能够建立一个最优架构配置的方法。采用自动化的工具，在设计流程的不同步骤间实现自动的转换，也被证明是非常有用的。

参考文献：

[1] http://www.abo.fi/~dragos.truscan/ipv6/rd/diags.html.

[2] C. Arpnikanondt and V. K. Madisetti. Constraint-Based Codesign (CBC) of Embedded Systems: The UML Approach. Technical Report YES-TR-99-01, Georgia Tech, 1999.

[3] M. Awad, J. Kuusela, and J. Ziegler. Object-Oriented Technology for Real-Time Systems: A Practical Approach using OMT and Fusion. Prentice Hall, 1996.

[4] D. Bj¨orklund, J. Lilius, and I. Porres. Towards efficient code synthesis from statecharts. In A. Evans, R. France, A. Moreira, and B. Rumpe, editors, Workshop of the pUML-Group held together with the UML2001 conference, Lecture Notes in Informatics, pages 29–41. GI, oct 2001.

[5] D. Bjørner. Software Systems Engineering, From Domain Analysis via Requirements Capture to Software Architecture. In Proceddings of Asia Pacific Software Engineering Conference(APSEC'95), 1995.

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

[7] A. Conta and S. Deering. Internet Control Message Protocol (ICMPv6) for Internet Protocol Version 6(IPv6) Specification. RFC 2463, December 1998.

[8] H. Corporaal. Microprocessor Architectures - from VLIW to TTA. JohnWiley and Sons Ltd., Chichester, West Sussex, England, 1998.

[9] S. Deering and R. Hinden. Internet Protocol, Version 6(IPv6) Specification. RFC 2460, December 1998.

[10] B. P. Douglass. Doing Hard Time. Addison-Wesley, 1999.

[11] B. P. Douglass. Ropes: Rapid object-oriented process for embedded systems.White-Paper, 1999.

[12] J. M. Fernandes, R. J. Machado, and H. D. Santos. Modelling Industrial Embedded Systems with UML. In Proceedings of CODES 2000, San Diego, CA USA, 2000.

[13] R. B. France and T. B. Horton. Applying domain analysis and modeling: an industrial experience. In Proceedings of the the 17th international conference on software engineering on Symposium on software reusability, pages 206–214. ACM Press, 1995.

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

一种基于 Web 的多源数据采集应用开发模式语言

Lei Zhen、Guangzhen Shao 著，刘辉 译

查看评论

摘要

本模式语言描述了基于 Web 上开发多源数据采集应用的模式。它主要处理从多重数据源上采集数据及描述数据的应用。这些多重数据源包括实时数据库，关系数据库以及其他数据源等。这种模式语言包括四种模式.显示组件(Display Component)是嵌在 Web 浏览器上的瘦客户端，主要用来动态显示。配置客户端(Configuration Client)主要用来定义和维护动态显示及数据源。远程数据采集器(Remote Data Collector)获取和传递数据。配置数据库(Configuration Database)存储要动态显示的数据的配置。

1.简介

1.1 意图

正在讨论的模式语言主要用来处理基于 Web 的多源数据源采集应用。这种应用的数据源可能是多样的，比如实时数据库，关系数据库或其他如动态网页数据源。

1.2 场景

价值链之间的整合和信息共享对于企业优化整个价值链以及在不同层次上建立快速和有价值的决策非常重要。考虑一下下面的设想：当一个提炼厂的主管打开浏览器，决策支持系统的主要页面显示在他面前。在页面中间，显示生产过程的模拟，生产过程的主要部分都被显示，相关的部分用颜色标注，比如用红色表示警告；在页面左边，它可以查看企业的重要统计数据；在页面右边，它可以查看石油和产品的价格。如果他想进一步查看详细的数据，他可以点击相应的超链接进入相关的页面。

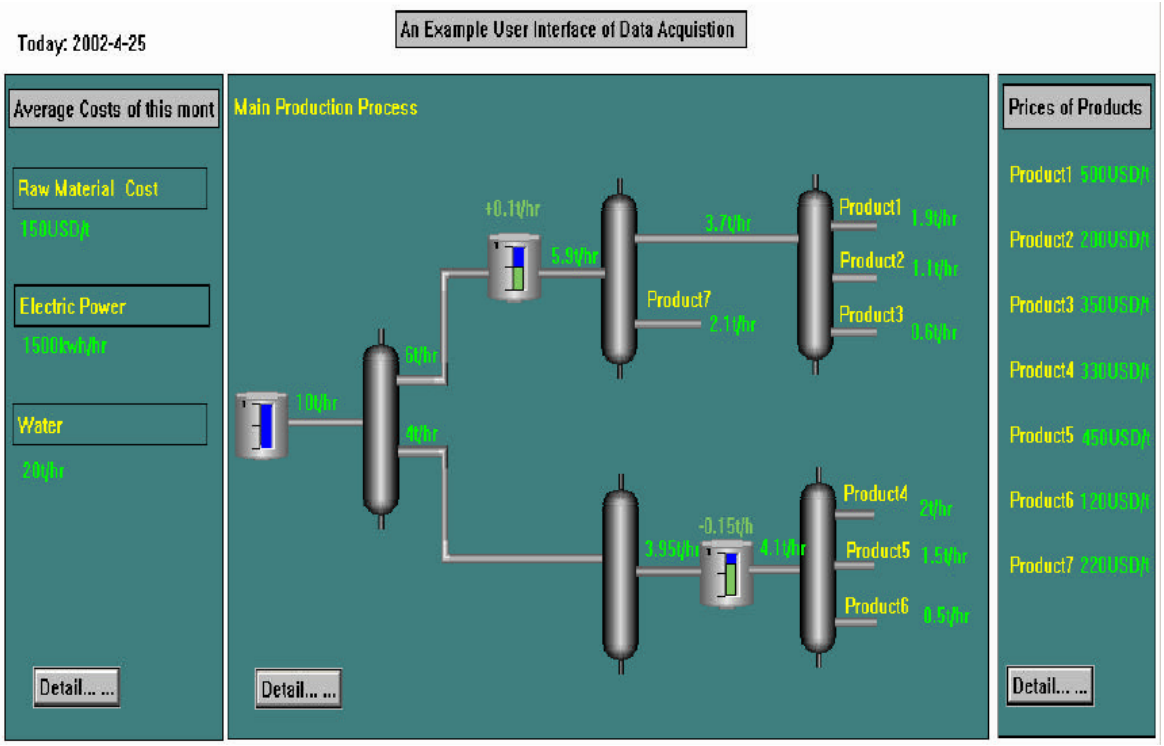


图 1 一个运用在决策支持系统中的数据采集的用户界面实例

在上面的实例中，所有的数据都在一个简单的页面中显示，但是数据却来自不同地方。生产过程的实时数据从实时数据库中获取。统计数据从管理信息系统（MIS）中的关系数据库中获取。而市场数据却是从第三方服务商的网站上获取的。传统的数据收集方法是从生产过程中收集实时数据，然后把他存放到一个过程数据库中并显示给工程师。但现在，不但希望从企业内部（MIS 比如库存管理系统）获取数据，而且希望从企业外部（比如股票价格的网站）获取数据；同时，他的用户不仅仅是工程师，而且还有企业领导、主管、合作方以及企业的其他人员。现在越来越多的数据采集和实时数据库厂商正在开发新的系统或者升级旧的系统，以使他们的数据采集系统能同客户的 ERP 系统或其他信息系统集成。正在讨论的模式语言是建立在我们对不同厂商的许多系统经验和学习上的。我们从 1995 年就开始进行数据采集与决策支持系统的集成。而且从 1997 年开始，我们为中国很多的炼油厂及电力厂建立了基于 Web 的系统。

1.3 模式语言的简介

本模式语言有四种模式，如下表。

编号	名称	说明	相关模式
1	显示组件（Display Component）	被嵌入浏览器中的瘦客户端。可以是 Java Applet 或者 ActiveX 文档	远程用户接口 (Remote User Interface) 远程数据库(Remote Database) 分布式应用内核 (Distributed Application Kernel)
2	客户端配置 (Configuration Client)	定义和修改数据源，并显示给终端用户	远程数据库(Remote Database)
3	远程数据采集器 (Remote Data Collector)	从不同数据源采集和传输数据	远程用户接口 (Remote User Interface) 远程数据库(Remote Database) 分布式应用内核 (Distributed Application Kernel)
4	配置数据库 (Configuration Database)	存储系统的配置数据	远程数据库(Remote Database) 分布式数据库 (Distributed Database)

1.3.1 部署视图

图 2 是本模式语言的典型部署视图

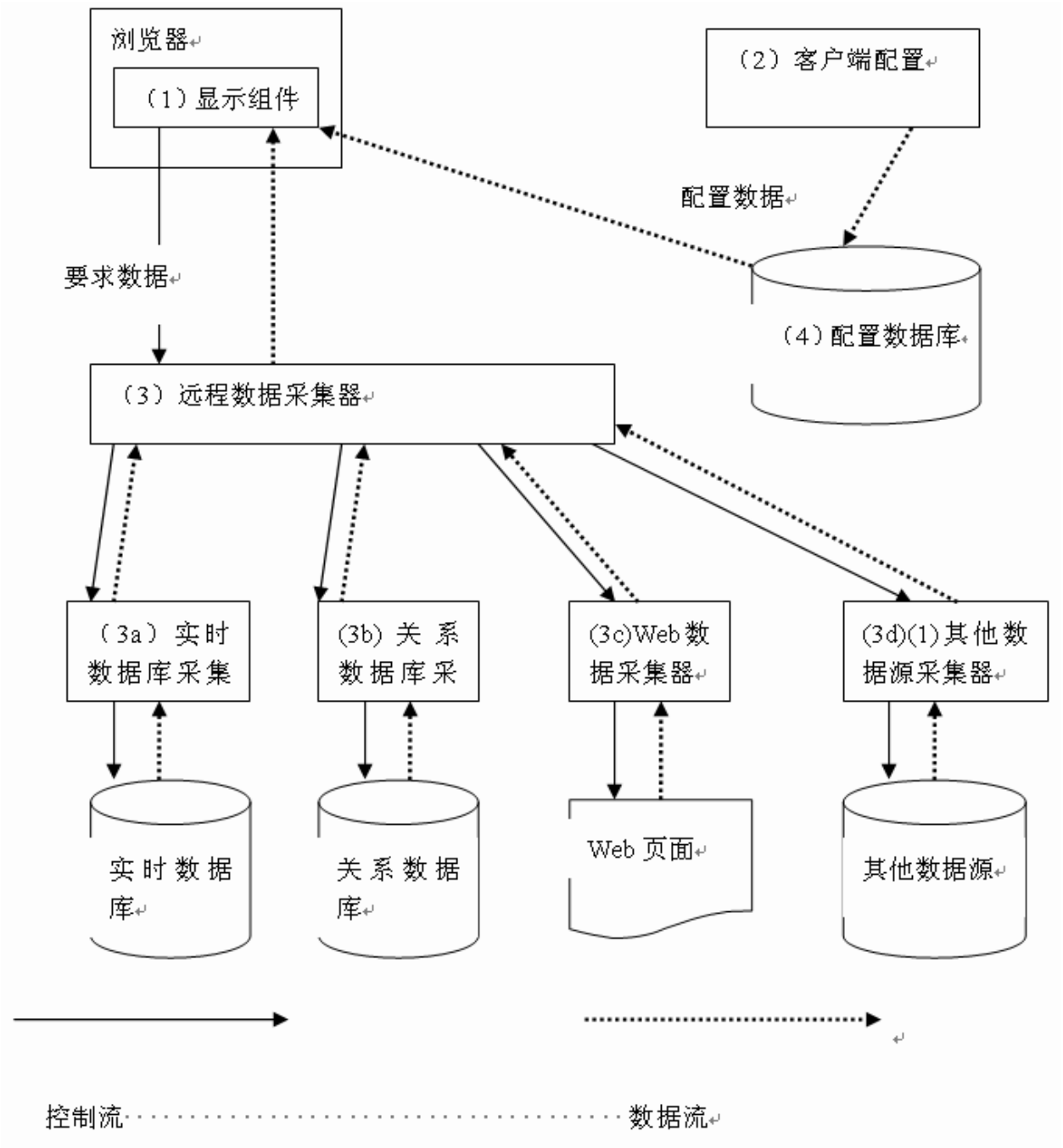


图 2 部署视图(Deployment View)

（1）显示组件(Display Component)是嵌在浏览器中的瘦客户端组件。当终端用户打开数据采集页面时，浏览器调用显示组件（Display Component）。

（2）配置客户端(Configuration Client)可以是一个基于 Web 的客户端也可以是一个桌面客户端，当调用它时，可以为终端用户定义和修改显示界面，也可为数据采集应用定义和修改数据源。

（3）远程数据采集器(Remote Data Collector)以及它的扩展采集器被部署在系统的 Server 端。

（3a）实时数据采集器处理从实时数据库(RTDB)中采集数据的请求。由于实时数据库多种多样，相应的也用多种与之对应的实时数据采集器。

（3b）关系数据库处理从关系数据库采集数据的请求。

（3c）Web 页面数据采集器吃力从动态页面上采集数据的请求。

（3d）还有许多其它类型的数据源，比如控制系统中的数据。要为这些数据源建立特殊的数据采集器。

（4）配置数据库（Configuration Database）是一个关系数据库。

1.3.2 模式的交互

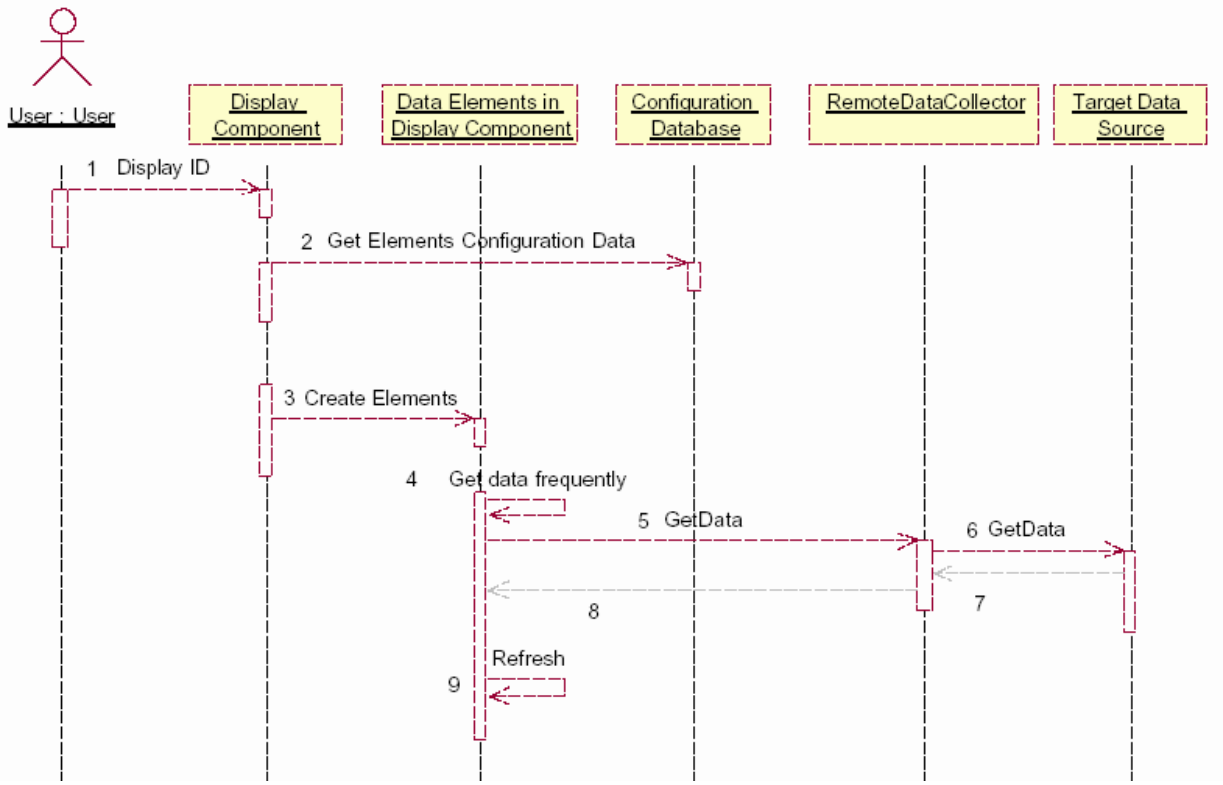


图 3 顺序图

（1）Display ID 作为显示组件（Display Component）的一个参数由 Web 页面提供。

（2）显示组件（Display Component）从配置数据库中获取数据。数据包括显示组件（Display Component）的属性（比如标题或者背景图）和要显示元素的属性。

（3）显示组件（Display Component）创建和显示包括数据元素(Data Element)、普通元素(Normal Element)、控制元素(Control Element)在内的元素。

（4）有一个定时器来控制数据刷新的频率。

（5）每一个数据元素(Data Element)频繁地从远程数据采集器(Remote Data Collector)获取数据。

（6）远程数据采集器(Remote Data Collector)从目标数据源获取数据。

（7）返回数据到远程数据采集器(Remote Data Collector)。

（8）返回数据到数据元素(Data Element)。

（9）数据元素(Data Element)刷新数据。

2. 模式 1－显示组件(Display Component)

2.1 意图

为了将从多个数据源获取的数据整合成一个显示，显示组件以一个嵌入浏览器中的瘦客户端利用配置数据库来表示动态的现实。

2.2 上下文

如果你要开发基于 Web 数据采集系统的客户端部分，你就得把从不同数据源获取的数据整合成一个显示。数据可能包括实时数据，关系数据和其他数据。数据的刷新频率可能从几秒到几分钟。

2.3 约束

交互性和可配置性：动态显示基于终端客户的业务需要。他们必须根据业务需要可改变。另外，用户可能想通过简单的操作，从动态数据的显示中获取更多的信息。比如，当他们在动态数据上移动鼠标时，他们能从提示字符中看到上限和下限信息；当他们点击动态数据时，他们能看到该数据的历史数据趋势。

数据完整性和一致性：必须保证数据的完整性和一致性，即使数据源有错误，或者数据在特定时间内不能获取（比如超时）。

交互性和可配置性 vs.复杂性。动态显示既要保持交互性，也要保持可配置性。为了保证一致的接口，你可能想在一个页面中显示所有的数据。然而，为每一个信息创建一个 asp 或者 jsp 页面在技术上和结构上要简单得多。

多线程和异步：采用异步传输，实现起来容易得多。然而采用多线程，你的系统将健壮得多。

2.4 问题

怎样将从多个数据源中获取的数据整合在一个页面中。

2.5 解决办案

提供一个显示组件（Display Component）。它是一个嵌在浏览器中的“瘦客户端”，从配置数据库中获取数据，并用这些数据来显示。每一个数据元素与一个远程数据采集器（Remote Data Collector）连接并从它获取数据。

采用多线程来防止数据传输阻塞。为每一个数据元素创建一个线程来获取数据。另外为每一个数据元素设置一个超时期限。如果数据元素超过期限仍未获取到数据，其值将设为“N/A”。

2.6 显示组件（Display Component）的结构

一个页面显示一幅背景图和几个元素。背景图可能是 gif 格式也可能是 jpg 格式。元素包括：数据元素，它从远程数据采集器（Remote Data Collector）中获取数据并频繁刷新数据；普通元素，独立于动态数据源，比如文本框，时间标签；控制元素，提供更多的交互功能，比如不同页面之间的链接。这些元素的属性都存储在配置数据库中。

每一个数据元素为显示提供了许多属性比如数据源类型、数据源名称和数据刷新周期等。数据元素频繁从远程数据采集器（Remote Data Collector）中获取数据。所有的数据元素有相同的属性。但有一些属性值与数据元素所绑定的数据源类型有关。如果数据源类型是实时数据库，数据源名字可能是服务器名称，目标名可能是数据点编号；而当数据源类型是关系数据库时，数据源名字可能就变成 ODBC 了，目标名可能是一条 SQL 语句。数据元素本身并不处理这些元素。他们只是把这些属性传递给远程数据采集器（Remote Data Collector）。

图 4 显示了动态数据显示页面的概念模型。

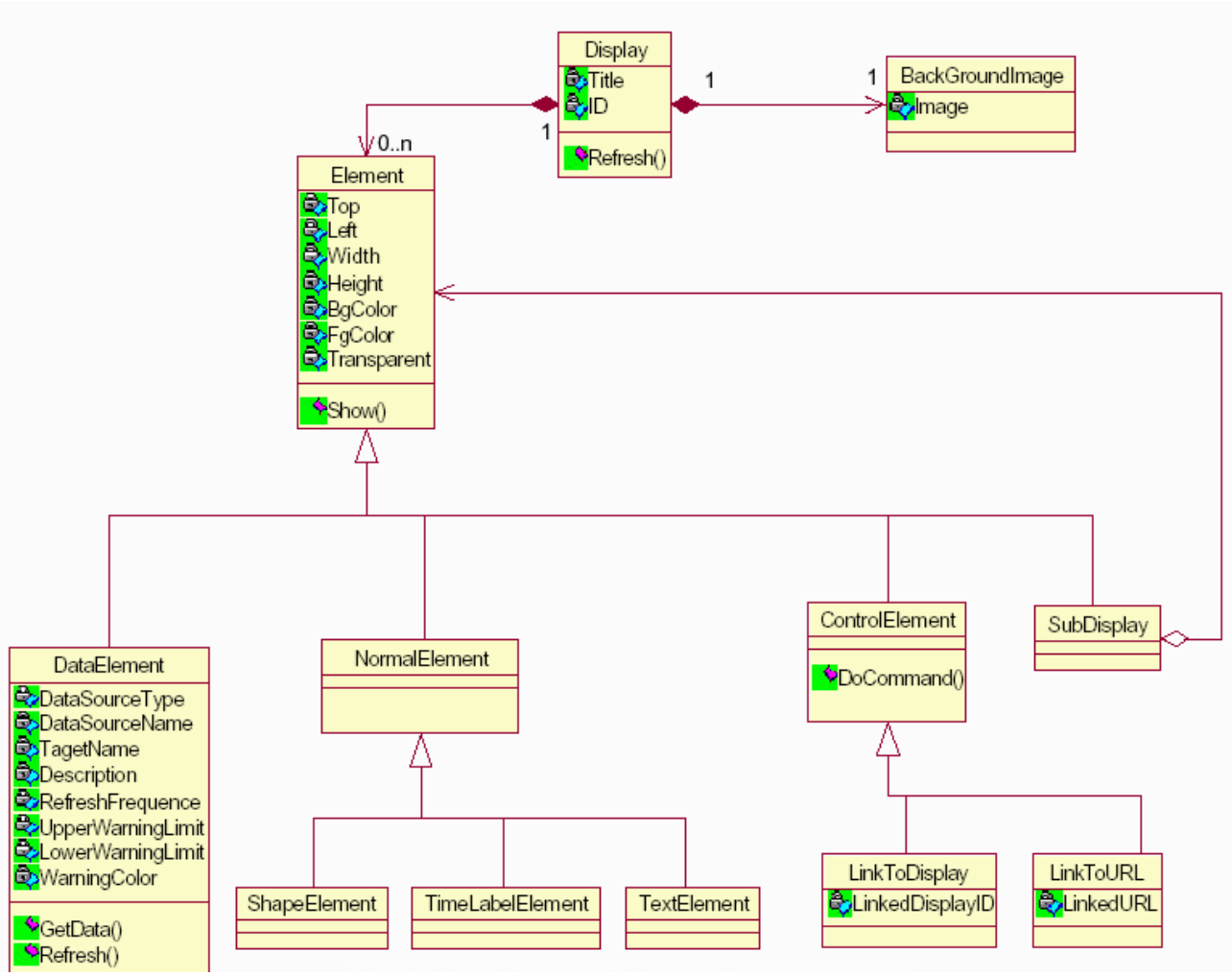


图 4 显示组件的类图

2.7 结论

这个模式有几种好处：

- 你可以用一个组件表示你想表示的足够多的动态显示
- 因为显示组件（Display Component）独立于数据源，所以它可用于多种环境中。
- 用户接口可以变得更加强大和可交互。

它也有一些缺点：

- 如果在一个页面中，有许多从不同关系数据库中获取的数据，将耗费不少获取数据的时间。
- 由于显示组件（Display Component）和远程数据采集器（Remote Data Collector）之间的交互不能通过现成的系统组件或者标准化的协议来实现，开发人员必须处理两者之间的通信问题。

2.8 相关模式

显示组件（Display Component）运用了一些在[RK1997]中的模式：

远程用户接口(Remote User Interface)：显示组件（Display Component）是一个嵌在浏览器中的“瘦客户端”。

远程数据库(Remote Database)：显示组件（Display Component）从配置数据库中获取配置数据。

分布式应用内核(Distributed Application Kernel)：每一个数据元素与一个远程数据采集器（Remote Data Collector）连接并从它获取数据。

还有许多其它的模式[GHJV95]可以应用于这个模式中：

Composite 模式：一个页面就是许多元素对象的组合。

Façade 模式：数据元素从远程数据采集器（Remote Data Collector）中获取数据从而与数据源分离。

Iterator 模式：页面中的元素以一个集合存储，并以 Iterator 实现。

2.9 已知应用

（1）iFix WebServer of Intellution[Intellution]。该产品的客户端是一个嵌在浏览器中的 Applet。

（2）PI Interactive Configurable Environment(PIICE) of OSI Soft[OSISOFT]。该软件的客户端是一个嵌在浏览器中的 ActiveX 控件。

3. 模式 2—配置客户端

3.1 意图

配置客户端（Configuration Client）用来定义和维护显示组件（Display Component）要显示的页面。

3.2 上下文

你正在开发一个数据采集系统，而且采用 Display Component 作为终端用户的接口。用户希望他们自己来定义和维护动态显示。

3.3 约束

- 复杂性 vs.可用性：一个功能齐全的 WYSIWYG 客户端可以帮助用户简单、迅速地定义和修改动态显示。但是它比一个简单的客户端要复杂和耗时得多。虽然一个简单的客户端可以简单和迅速的扩展，但它的可用性可能还是极其贫乏。
- 开发和实施费用之间的取舍：数据采集的实施费用通常很高，因为有许多的动态显示要配置。一个功能强大的工具比如象功能齐全的 WYSIWYG 客户端工具能使这些工作变得简单，并能减少实施费用。但理所当然地，开发此类工具的费用不菲。

3.4 问题

- 怎样正确、简单地存储数据采集系统的配置数据。
- 什么方案能减少整个系统的费用。

3.5 解决办案

采用 Configuration Client,它是一种综合解决方案，因为它同时定义和存储配置数据。它非常象 WYSIWYG 图像编辑器，但是它没有包含一些需要很多时间和金钱的实现的复杂的功能。它能导入一幅背景图，也能容易地定义元素。在本实例中，你可以在 Configuration Client 中利用 Display Component 的一些信息。

3.6 结论

这种模式有几种好处：

- 你可以开发一个 Configuration Client 来满足你系统的需求和条件。
- 你可以在开发费用和实施费用之间均衡，以使整个费用最小。

3.7 相关模式

远程数据库(Remote Database)[RK1997]：在配置数据库中存储配置数据。

其他许多模式也可用于本模式中。比如 ***Composite Pattern, Iterator Pattern***。

3.8 已知应用

（1）iFixWebServer of Intellution[Intellution]。IfixWebServer 通过 iFix 系统来表示图形的动态显示。IfixWebserver 的 Configuration Client 是 iFix Client。

（2）PI Interactive Configurable Environment(PICE) of OSI Soft[OSISOFT]。PI ProcessBook 是一个简单易用的图形工具包，它允许用户创建动态可交互的页面。

4. 模式 3—远程数据采集器

4.1 意图

远程数据采集器(Remote Data Collector)从各种数据源采集数据并把这些数据传向显示组件(Display Component)。

4.2 上下文

你在开发一个数据采集系统。它应该考虑多种类型的数据源。你不能实现假定数据源的数量，而且新的数据源能加进系统中。

4.3 约束

数据源的独立性 vs.复杂性：对数据源数量和类型的独立可以使你的系统变得更加可维护和可扩展。但是这种独立性需要更大的复杂性。

4.4 问题

怎样将数据源与系统的其它部分分离。

4.5 解决办案

采用远程数据采集器(Remote Data Collector)从不同的数据源中采集数据和传输数据。图 5 显示了远程数据采集器(Remote Data Collector)的结构。

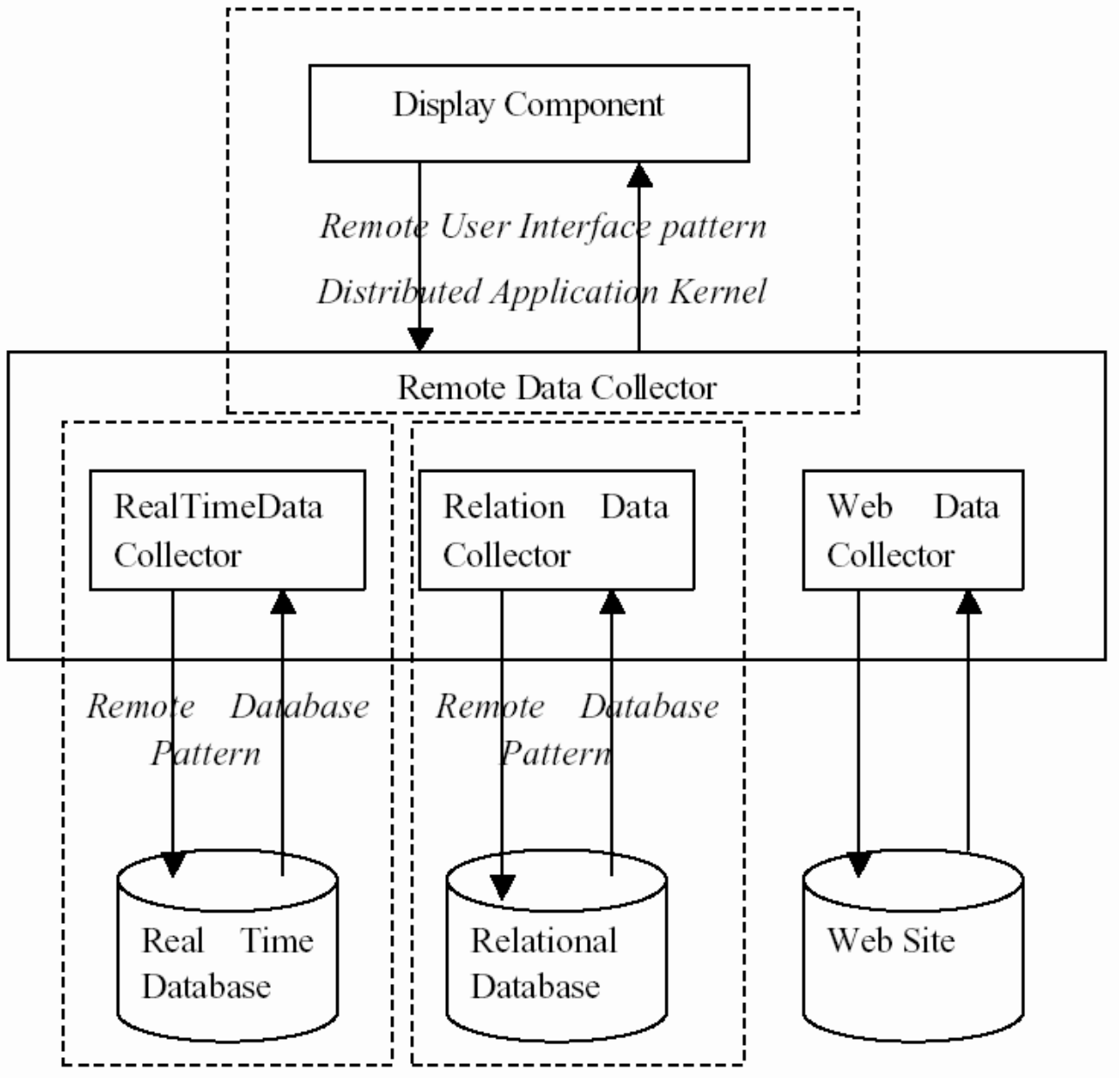


图 5 远程数据采集器(Remote Data Collector)的结构

你可以为每一个数据源设计一个统一接口的组件。有三种典型的远程数据采集器(Remote Data Collector)：实时数据采集器、关系数据采集器及网页数据采集器。

实时数据采集器从实时数据库（RTDB）中获取数据。因为有很多种的采集器而没有统一的接口，我们必须为系统的每一种实时数据库(RTDB)建立一个特殊的采集器。

关系数据采集器从关系数据库中获取数据。它可采用标准的 SQL 语句为协议。你可以为不同类型的关系数据库建立同一个数据采集器。但是你必须考虑不同数据库支持的一些特殊的 SQL 语句。比如在不同的关系数据库中，数据类型的定义就有差别。你也可以为每一种关系数据库建立一个数据采集器。在这种情况下，你可以调用各种关系数据库的专门 API，以使系统的性能达到最佳。

动态网页数据采集器从动态网页中获取数据。你应该使得你的采集器能够利用动态网页的结构。

可能还有其他类型的数据源，比如控制系统。要为这些数据源建立专门的数据采集器。

4.6 结构

你可以为不同的数据源建立一个统一的远程数据采集器(Remote Data Collector)接口，如图 6 所示。对每一种数据源，你可以建立一个从该接口继承的组件。这个组件实施在系统的 Server 上。显示组件(Display Component)通过创建一个远程的实例来获取数据。

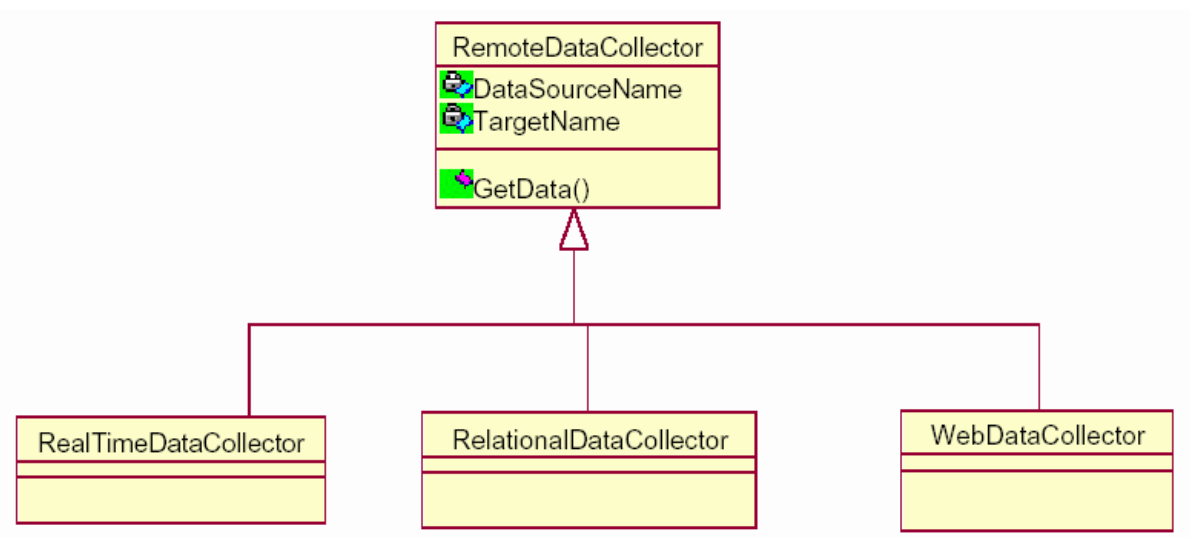


图 6 远程数据采集器(Remote Data Collector)接口

你也可以将远程数据采集器(Remote Data Collector)创建成一个 Server。如图 7 所示，远程数据采集器服务器（Remote Data Collector Server）为每一种数据源包含了一种采集器。在这种情况下，远程数据采集器服务器(Remote Data Collector Server)创建采集器实例，从这些采集器中获取数据并把数据传给显示组件(Display Component)。

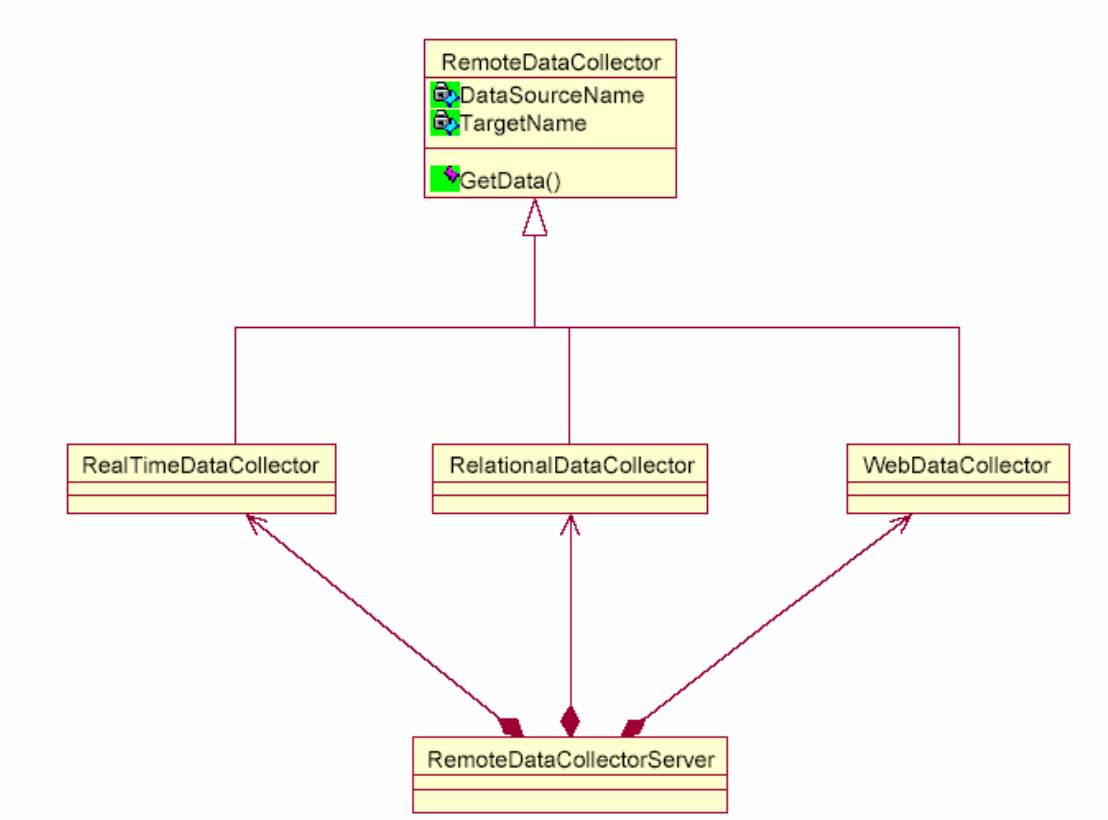


图 7 远程数据采集器服务器(Remote Data Collector Server)

4.7 结果

这种模式有几种好处：

- 更好的维护性：用户自己可以定义和修改接口。
- 系统具有更好的可扩展性：数据源很容易被加进来。

缺点：

- 远程数据采集器(Remote Data Collector)结构比较复杂。为了保证数据的完整性和一直性，你必须考虑数据传输阻塞和异常问题。
- 如果有一种新的数据源，你必须提供一种新的数据采集器。

4.8 相关模式

正如图 5 所显示，关系数据库和实时数据库都是***远程数据库（Remote Database）[RK197]***。它还采用了***远程用户接口(Remote User Interface)***和***分布式应用内核(Distributed Application Kernel)[RK1997]***。

其他许多模式[GHJV95]也被用于这个模式中：

外观模式(Façade Pattern)：数据元素从远程数据采集器(Remote Data Collector)中获取数据,数据元素和数据源独立。

适配器模式（Adapter Pattern）：每一种类型的远程数据采集器(Remote Data Collector)都是一个远程数据采集器接口(Remote Data Collector Interface)与具体数据源 API 之间的适配器(Adapter)。

4.9 已知应用

（1）iFixWebserver of Intellution[Intellution]。IFix Server 就是一个远程数据采集器(Remote Data Collector)。它通过 iFix 接口从支持的数据源中获取数据。

（2）PI USD of OSI Soft[OSISOFT]。UDS 提供了与所有支持的数据源的开放式连接。UDS 无缝与 PI 数据存储（Data Storage）、非 PI 数据存储(Data Storage)比如 OLEDB 提供商（SQL Server 或者 Oracle）、专有系统（比如 PHD,CIM/21）和其他支持数据库连接。

5. 模式 4—配置数据库

5.1 意图

配置数据库存储显示的配置数据。包括要显示的元素和数据源。

5.2 上下文

你在开发一个基于 Web 的数据采集系统，你必须寻求一种存储配置数据的方法。

5.3 约束

- 与现有系统的整合：你系统的数据源大部分是客户的应用系统。你的系统必须保证配置数据与这些已有系统一致和完整，而不是让客户去重新定义。

- 数据一致性和安全性 vs.复杂性：采用关系数据库来存储配置数据可以提高这些数据的一致性和安全性，但采用文件来存储配置数据要简单和经济。
- 配置数据模型的结构：一个定形的数据模型能自然的保持数据的一致性和完整性。但是要保持数据的一致性是一项艰巨的工作，特别是数据必须从不同的系统中进行组合情况下。

5.4 问题

- 怎样存储配置数据？
- 满足整合现有系统的数据模型是什么样子？

5.5 解决方案

为了保持配置数据的一致性，采用关系数据库来存储配置数据。

配置数据库有两个逻辑部分：显示数据库(Display Database)和综合数据库(Integration Database)。显示数据库(Display Database)利用公式化数据模型来实现以存储动态显示的配置数据。显示组件(Display Component)从其获取动态显示的配置数据。综合数据库(Integration Database)存储现有系统的数据元素，作为动态显示的候选。

采用单独的存储过程和其他处理程序来维护现有系统与综合数据库(Integration Database)之间的一致性。当用户要配置动态显示时，他们从综合数据库(Integration Database)选取数据元素，定义显示属性，并通过配置客户端(Configuration Client)把这些信息存储到显示数据库(Display Database)中。配置数据库的结构如图 9。

如果现有系统的配置数据像配置数据库(Configuration Database)一样存储在关系数据库中，我们可以用存储过程来维护数据的一致性。否则，我们要开发单独的过程来处理这项工作。

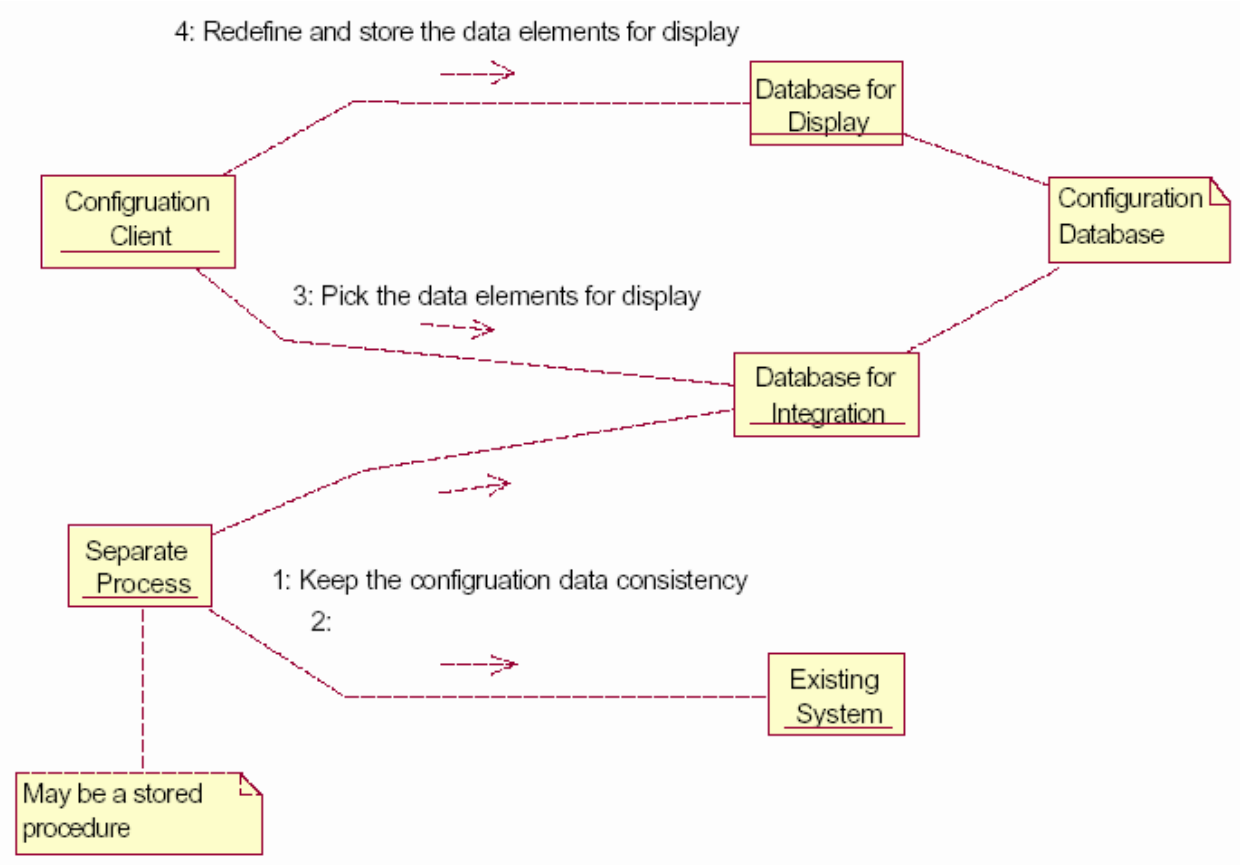


图 9 配置数据库(Configuration Database)的结构

5.6 结构

我们采用 ERA 来表示显示的概念数据模型，如图 10。

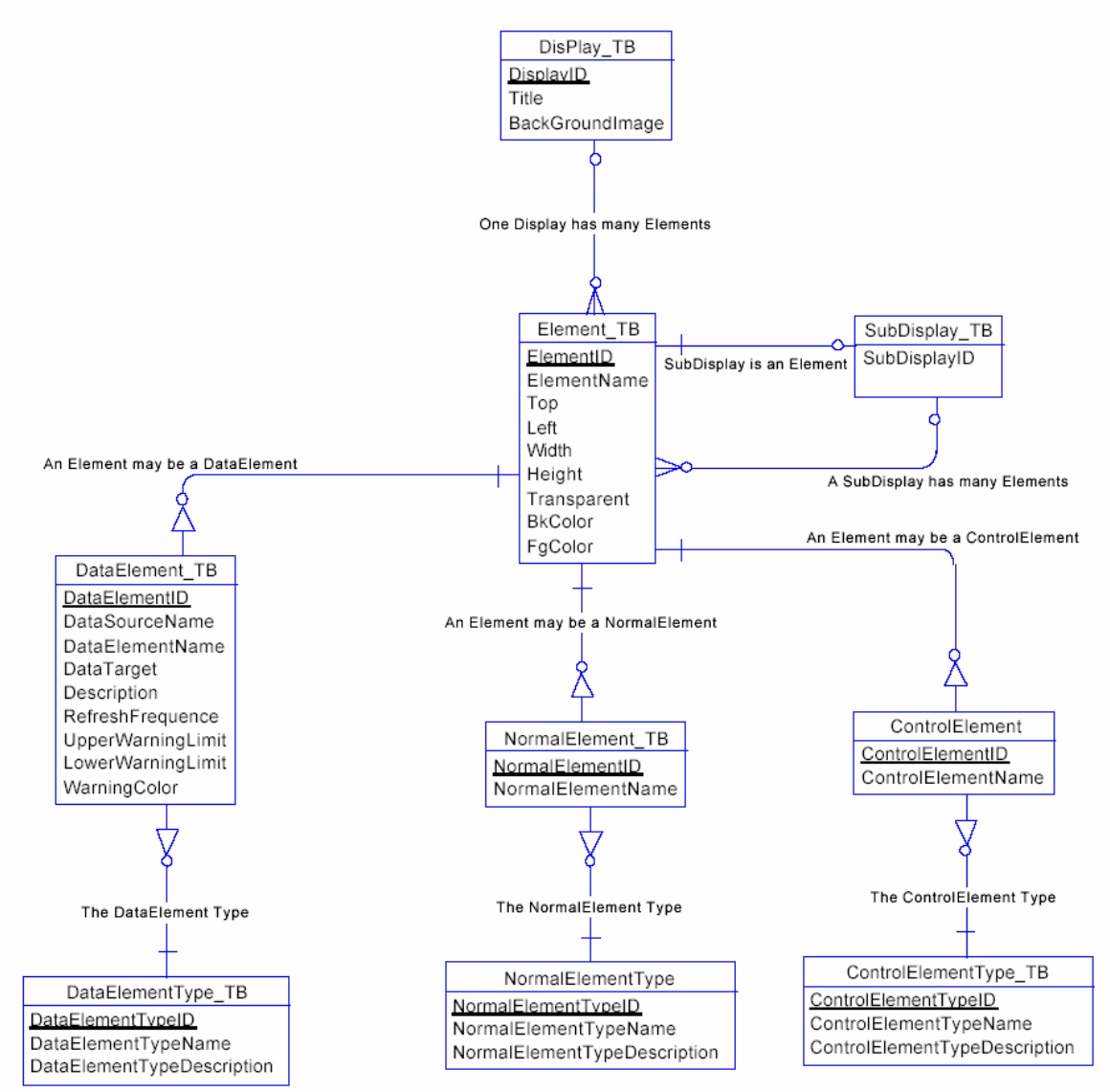


图 10 概念数据模型

由于大部分的关系数据库都支持大二进制字段，因此显示的背景图像可以存储在一个字段中。

元素(Element)与它的子类(数据元素(Data Element),普通元素(Normal Element),控制元素（Control Element）)之间的关系是一对一，这些子类依靠父类(Element)。我们可以在数据库中创建触发器(Trigger)来维护他们之间的关系，触发器(Trigger)采用了 Observer 模式[GHJV95]。

为了保持数据模型的稳定，我们可以为每一种元素创建一个类型实体(Type Entity)。这样我们可以通过添加纪

录来添加元素实体，而没有必要去修改数据模型。

5.7 结果

采用关系数据库来存储配置数据使得系统更加稳定和健壮。但是当你的系统很小时，采用关系数据库将显得不经济和复杂，你可以采用文件来存储配置数据。在这种情况下，你可以为每一个显示配置数据创建一个文件，采用显示的 ID 或者名称来作为文件的名称。显示组件(Display Component)通过 ftp 或者 http 来获取配置数据。

你可以采用这种模式来整合现有系统，但当系统比较复杂时，你必须采用存储过程和单独的处理过程来保证你的系统与现有系统之间的一致性。

5.8 相关模式

业务对象与关系数据库连接(Connecting Business Objects to Relational Databases)[YJW1998]模式有助于你建立面向对象设计模型与关系数据模型之间的连接。

配置数据库(Configuration Database)是一个***远程数据库(Remote Database)[RK197]***。它与显示组件(Display Component)和配置客户端(Configuration Client)连接。如果配置数据库(Configuration Database)与现有系统中存储配置数据的数据库时同种类型时，他们组成了***分布式数据库(Distributed Database)[RK1997]***。

用来保持数据一致性的触发器(Trigger)采用了***观察者模式(Observer Pattern)[GHJV95]***。

5.9 已知应用

（1）iFixWebServer of Intellution[Intellution]。IFixWebServer 的配置数据存储在 iFix 过程数据库中。

（2）PI Interactive Configurable Environment(PIICE) of OSI[OSISOFT]。PIICE 利用 SQL Server2000 来存储个性化和构件信息。

6 总结

越来越多的模式用在基于 Web 的系统上。比如[RIC2001]和[PK1999]。因为这些模式的可维护性和可用性，基于 Web 的系统被广泛的应用在 Internet 和 Intranet 中。越来越多的过程工业公司(Process Industry Company)希望将数据采集系统与 ERP 系统进行集成。数据的集成可以将数据转变为信息和知识。我们讨论的模式语言正是建立在

我们对不同公司的多种系统学习和积累的基础上。除了本文中列出来的实例，还有更多的系统采用了这种模式。例如，Web2.1 of AspenTech[AspenTech][ZHA2001],VisualPHD of HoneyWell[HoneyWell][ZHA2001].

开发人员很容易地应用这些模式。这种语言中的每一种模式都可以根据你系统的范围和大小采用不同的技术实现。

7 致谢

向为本文修订版提出指导性宝贵意见的 Jutta Eckstein 致以特别的谢意。

8 参考文献

[ZHA2001] Zhilin Zhang. The Principle and Application of Real Time Database Management System: published by Sinopec Press, China, 2001

[RK1997] Klaus Renzel, Wolfgang Keller. Client/Server Architectures for Business Information Systems, PloP'1997.

[GHJV95] E. Gamma, R. Helm, R. Johnson and J. Vlissides. Design Patterns: Elements of Reusable Object-oriented Software. Reading, MA: Addison-Wesley, 1995.

[RIC2001] Chris Richardson. A Pattern Language for J2EE Web Component development: PloP'2001.

[YJW1998] Joseph W. Yoder, Ralph E. Johnson, Quince D. Wilson. Connecting Business Objects to Relational Databases: PloP'98

[PK1999] Kimberly Perzel, David Kane. Usability Patterns for Applications on the World Wide Web: PloP'99

[Intellution] http://www.intellution.com/products/fixwebserver/default.asp

[Honeywell] http://www.honeywell.com

[AspenTech] http://www.aspentech.com

[OSISoft] http://www.osisoft.com/

UMLChina 公开课

“UML 应用实作细节”

（2003 年 9 月 27-28 日，上海）

现在，UML、RUP、Rose 等概念已经传播得越来越广，书籍、资料也越来越多，决定在开发过程中使用 UML 的开发团队也越来越多。

在开发团队应用 UML 的软件开发过程中，自然会碰到很多细节问题：“我这样识别 actor 和用例对不对？”、“用例文档这样写合适吗？”、“RUP 告诉我该出分析类了，可类怎么得出来啊？”、“先有类，还是先有顺序图啊？”、“类怎样才能和数据库连起来啊？”...许许多多的细节问题，而每一个细节都和背后的原理有关。

本课程秉行 UMLChina 一贯的“只关心细节”的原则，内容完全是由 UMLChina 自行设计，围绕一个案例，阐述如何（只）使用 UML 里的三个关键要素：用例、类、顺序图来完成软件开发。使学员自然领会 OOAD/UML 的思想和技术，并对实践中的误区一一指正。整个过程简单实用，简单到甚至可以只有一个文档，非常适合中小团队。

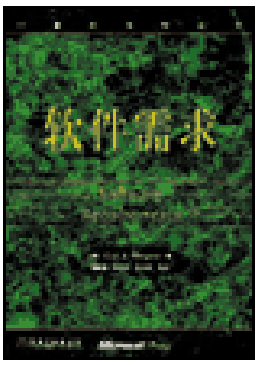
详情请见>>

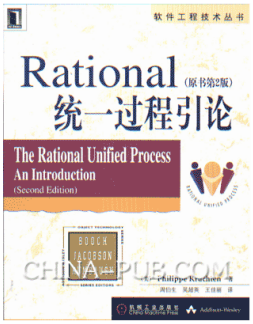
国内中文书对《人月神话》的引用精选（一）

UMLChina 整理

查看评论

按出版时间排序

书	作者	页码	书中评价原文
	Karl Wiegers	4	Frederick Brooks 在他 1987 年的经典文章《没有银弹》中已充分说明了需求过程在软件项目中扮演的重要角色。
	Brian W.Kernighan	48	给我看你的流程图而藏起你的表，我将仍然莫名其妙。如果给我看你的表，那么我将不再要你的流程图，因为它们太明显了。 以上从 Brooks 的经典书中摘录的内容想说的是，数据结构设计是程序构造过程的中心环节，一旦数据结构安排好了，算法就像是瓜熟蒂落，编码也比较容易。
	Alan Cooper	59	Fred Brooks 说，“要计划丢弃某些东西”。1988 年，我把 Ruby 卖给 Bill Gates，当时只是一个原型...接下来要把它和 Bill 的 QuickBasic 结合，成为 Visual Basic。我做的第一件事就是抛弃 Ruby 的原型，除智慧和经验外，一切从零开始。

	Steve McConnel	271	Frederick P. Brooks 在 1987 年发表了一篇《没有银弹——软件工程的本质与故障》，后来这篇文章成为了软件工程领域中最具影响力和最负盛名的文章（Brooks 1987）……Fred Brooks 指出，我们的软件行业正卷入一项长期的探索之中，即，要找到消除低生产率这个恶魔的神奇银弹。亚力山大大帝流泪了，原因是他已找不到没有被他征服过的世界。
		295	增加新手一定要审慎。要记住 Brooks 法则。
	Philippe Kruchten	226	这是一本经典著作，与软件开发有关的每一个人都应该不只一遍地读这本书。我推荐 1995 年的版本。 ——作者在书中把《人月神话》列在推荐书籍的首位
	Kent Beck	182	让你思考四个变量（成本、时间、质量、范围）的一些故事。周年版中还有一段关于著名的文章“No Silver Bullet”的有趣对话。
待续….			

欢迎您提供您的信息。