第33卷 第14期 计 算 机 工 程 2007年7月
Vol.33 No.14 Computer Engineering July 2007
·软件技术与数据库· 文章编号：1000—3428(2007)14—0062—04 文献标识码：A 中图分类号：TP311

面向方面的软件工程指南

莫 倩，刘 晓
(北京工商大学计算机学院，北京 100036)

摘 要：面向方面的软件开发(AOSD)技术的目标，是在整个软件生命周期中提供系统化标识、模块化以及组合横切关注点。随着AOSD技术的成熟，需要一个指南来支持良好工程化的面向方面系统的开发。该文综述了现有面向方面软件工程的各种方法，分析了在需求分析、设计和编程实现阶段对方面进行考虑的方法，并提出了比较这些方法的准则。文章为面向方面的实际应用选择专门的方法(方法组)提供了指南。

关键词：面向方面；软件工程；面向方面软件开发

Guidelines for Aspect Oriented Software Engineering

MO Qian, LIU Xiao
(School of Computer Science, Beijing Technology and Business University, Beijing 100036)

【Abstract】The object of aspect oriented software development(AOSD) is to provide methods of systematizing marking, modularizing and assembling transverse cut attention in the software life cycle. With the development of the AOSD technology, a manual supporting good projecting aspect oriented system development is required. This paper presents a study of existing approaches to aspect-oriented software engineering. It examines approaches such as consideration for aspects at the requirements, design and implementation phases and provides criteria for comparing approach. It provides guidelines for the selection of a particular approach(or set of approach) for actual aspect-oriented applications.

【Key words】aspect-oriented; software engineering; object of aspect oriented software development(AOSD)

面向方面软件开发(AOSD)技术的目标，是在整个软件生命周期中提供系统化标识、模块化以及组合横切关注点。当前，在编程实现阶段，已经有了许多可用的面向方面编程方法，如AspectJ[1]、组合过滤器[2]、自适应编程[3]和Hyper/J[4]。面向方面的概念还被应用到了软件工程的早期阶段。在需求分析阶段，文献[5～8]提供了处理方面需求的手段。同样，许多面向方面的设计[5,9～12]方法也已经被提了出来。对于软件工程师来说，在每个阶段基于如此众多的可用技术，实现一个面向方面系统的软件工程任务，会遇到许多严重的挑战。在软件工程的不同阶段，软件工程师需要为待开发的应用采用最合适的面向方面技术。对技术的选择取决于很多因素，包括系统需求、工具的约束条件或者开发环境，还包括横切关注点的自然属性。

随着AOSD技术的成熟，需要一个指南来支持良好工程化的面向方面系统的开发。本文的目标就是为软件工程生命周期的关键阶段提供这样的指南，描述在不同阶段AOSD方法的独特特征，以及它们对于被应用或者被模块化的关注点的适应性。

本文能够帮助软件工程师在每个阶段选择优化的面向方面的技术或者技术集合，对典型的软件工程生命周期进行了镜像，在软件工程每个阶段考量面向方面技术的应用。

1 需求分析

在需求分析阶段进行面向方面的工程，需要完成两部分工作：(1)为模块化和组合关注点提供新的需求分析抽象和组合机制，当前只有部分需求工程技术专注于这样的目标；(2)通过分离功能性和非功能性的需求实现关注点分离，这是大部分基于需求分析阶段的面向方面工程技术所研究的内容。非功能性的需求是指影响系统的多个组件并与服务质量相关(例如可用性，安全性)的约束条件。因此，它们能够被看作是方面的一个良好的备选。

在需求分析阶段，该如何选择最能满足一个组织及其项目需求的方法呢?决策过程应该包含如下2个主要步骤：

(1)从整个已有集合中选择备选方法的子集。这项工作的指导方针是要定义好一组结构化的约束条件，范围从预算限制到有关某个方法的专家技巧。

(2)从缩减的集合中选择一个特殊的方法或者几个方法的集成组合。这项工作需要考虑笔者希望使用的开发过程。可能会影响决策的关键点有：

1)对方面概念的支持。该方法是否直接支持基本的AOSD概念？

2)生命周期的活动。开发过程的哪项活动受到支持？

3)组合规则。该方法是否提供显式的规则来组合方面与需求？

4)处理冲突。该方法是否提供标识和消解冲突方面的方法？

5)成熟性。该方法是否已经在真正的项目中应用？

6)映射到后阶段的构件。该方法是否预见到怎样将方面需求映射到设计和实现构件上？

7)工具的支持。是否有一个工具支持该方法？

表1给出了对当前几种主流的面向方面需求分析方法应用这些关键条件的比较结果。

作者简介：莫 倩(1972－)，男，博士、副教授，主研方向：面向方面的软件工程，Web挖掘，Web Intelligence；刘 晓，硕士研究生
收稿日期：2006-07-25 **E-mail：**doctormo@263.net

表 1　主流面向方面需求分析方法比较

关键方法	对方面概念的支持	生命周期的活动	组合规则	处理冲突	成熟性	映射到后阶段的构件	工具的支持
KAOS[25]	无	需求与描述	无	有	有	有	Grail
I*[25]	无	组织的请求与需求	无	有	有	无	OME
问题框架[27]	无	需求与体系结构	有	无	有	有	无
PREView[8]	无	需求	无	无	有	无	JPREView
AOCRE[6]	有	(CBS)需求	无	无	部分	有	Jcompose
AORE[7]	有	需求	有	有	无	有	ARCaDe
Theme[5]	有	需求	有	无	无	有	Theme/Doc

考虑这个比较表，可能会帮助笔者决策的主要问题包括：

(1)KAOS。KAOS 是一个针对需求工程的面向目标方法的实际建立好的应用。除了允许形式化建模之外，它的一个主要优点是使用了一种形式化语言(一阶时序逻辑，带有实时构件)来描述系统的关键部分。KAOS 使用目标来探测和管理需求间的冲突。它已经应用在了几个工业化项目之中。

(2)i*。 i*框架在标识和建模一个组织的商业领域非常高效。它没有直接支持映射到软件生命周期的后期活动，但是近期的工作正在开发从 i*模型派生到 UML 模型的方法。

(3)问题框架。问题框架分类了软件开发的问题。目前没有系统化的方法来应用此概念，该方法可能难以学习和应用。近期的工作采用软件体系结构组合了问题框架。

(4)PREView。PREView 已经在一些小型和中型的安全性关键系统中使用。因此，它提供了稳定的概念集合和一种可靠的方法。为了不让需求工程师冒视点需求与关注点可能产生不一致性的风险，该方法没有标识横切属性对后期开发阶段构件的映射关系和影响。

(5)AOCRE。AOCRE 支持在基于组件的软件开发中模块化方面需求。该方法的优势在于重点集中在处理组件系统中的横切关注点。其主要缺点在于不清楚如何为每个组件标识方面，而且缺乏处理冲突的机制。

(6)Theme/Doc。Theme/Doc 通过提供一种方法标识基本和横切行为，以支持需求分析活动。它提供一种工具支持需求中的词法分析，标识出一个领域中的主要动作，并将它们与原始的需求相关联，这是一种非常有意义的特征。让这些主要动作彼此通过需求相关联，这种方式提供了一个良好的基础，能够帮助标识那些可能被横切的行为。

(7)AORE。在 AORE 方法中，从方面需求到非方面需求的组合信息的分离，使得它们彼此高度独立，因此也提供了关注点间的改进的分离。这也在某种程度上改进了方面的可重用性。由于组合的规则是操作在独立需求的粒度之上，因此在一个精确的粒度上标识和管理冲突是可能的。它优化了需求工程师为业主标识出谈判点的任务。映射关系的识别和需求层次方面的影响，为在系统开发、维护和演化过程中的大范围的需求和约束条件提供了更高的可追踪性。不过，这个方法仍然处于初级阶段，需要在真实项目中验证。

2　设计

当前人们已经提出了许多方法为设计层次的横切关注点提供支持。这些方法的大部分是通过横切关注点的帮助来实现面向对象技术和UML的扩展。设计层次的一些关键方法有：组合模式[9]和它们的后继方法主题/UML[5]，面向方面组件工程(AOCE)[6,10]，超空间[12]，方面体系结构视图[13]，以及文献[11]中的方法。

表 2　面向方面设计方法比较

设计技术	可追踪性	变化的传播	可重用性	可理解性	灵活性	易于学习/易于使用	并发开发
组合模式	好	一般	好	好	好	差	好
AOCE		差	好	一般	好	好	好
超空间	好	一般	好	好	好	差	好
体系结构化视图	好	一般	好	一般	好	差	好
文献[11]中的模型	一般	一般	一般	一般	差	差	一般

这 4 种面向方面设计在几项性能指标上的比较如表 2 所示。下面是 4 种设计方法的具体介绍。

(1)组合模式方法。基于面向主题的设计模型[14]的组合，用以组合分离的交织设计和UML模板。它基于这样的观察结果——横切行为的模式存在于它们所横切的基本设计之中。这种行为没有考虑关注点的主题切入点。因此，可重用的主题和关注点能够被彼此独立地设计。目前，组合模式的研究工作处于完善之中且尚未有重大的改变，并正在被合并到主题/UML[15]，它是主题方法的一部分[5]。

(2)AOCE。在设计层次，将前面所标识的面向方面组件需求(来自 AORE)优化到设计层次的方面。AOCE 基于这样一个思想——在一个基于组件的系统中，每个组件使用/提供服务从/至其它组件之中。一组 UML 元模型扩展和可视符号，被用来使得所提供和所要求的方面细节显现。因此横切关注点分离被按照它们对整个系统属性的贡献，被用来分类组件和推导组件间的服务。

(3)超空间方法。提出使用超切片来封装模块集合，它们通过不同维度来处理关注点，而不是使用一个主导的形式化机制。超切片可以重叠；即一个超切片给定的单元可能以另一种形式，出现于其它超切片之中。该模型能够以任何形式实例化，此时要作出一个选择——在符号化构件和单元/模块之间进行映射，以及如何表示该超切片。目前的结果是，该模型还没有任何标准化的设计符号。不过，它已经按照面向对象的形式化机制被实例化了[16]。

(4)按照方面的体系结构化视图[13]。方面被表示为元类型设计包，它包含了属于某个关注点的不同UML设计。每个类图的元素具有某种关系(例如使用、定义)，元素一定要与其他方面的元素以颜色区分标记。一个关注点类图的符号被引入，这允许查看所设计系统的重叠的关注点。该组合采用了文献[17]中所讨论的叠印原理。本质上，这个设计方法非常类似于另一种超空间模型UML解释方法，它增加了某些可视的记号(例如关注点类图)和组合机制(例如叠印)。

Suzuki 和 Yamamoto 提出了 UML 的扩展，用来建模方面和 UXF/a，它是一种基于 XML 的方面描述语言，用来在不同开发工具之间提供面向方面模型的可交换性(如 CASE 工具、方面编织器)。一个方面被描绘成一个类的矩形框图，带有属性、操作和关系。属性被操作的集合所使用，它们是为方面所定义的编织。这些定义的表示反映了被给定方面所影响的元素。

“好的”面向方面设计技术应该提供一组符号封装整体的横切需求，帮助理解复杂系统，并提供可重用和演化。结果设计应该展示程度相当高的可追踪性、变化的传播性、可

重用性、可理解性、灵活性、并发开发、易于学习和易于使用。表 3 展示了前面所讨论的每个设计方法，以及它们所产生的设计所能满足这些特性的程度。

在 AOCE 中，设计不需要被合并而是要保持完整，这样在整个生命周期中容易被追踪。在组合模式和超空间两种方法中，横切需求对于组合模式和超切片是明显各自可追踪的，但却像设计被合并一样被混杂在了一起。体系结构化视图通过关注点类图帮助实现可追踪性，它在第一眼显现的是粗层次的重叠区域。它们还支持设计每个横切需求为一个独立的方面，并在源码标注组合的设计。不过，组合设计在某种意义上过度了。Suzuki 和 Yamamoto 的模型只提供对可追踪性的部分支持。横切需求对方面是可追踪的，而反过来引用会影响对象元素。

组合模式和超空间支持非侵入式的改变能力，但组合规则在改变之后需要进行回顾。在 AOCE 中，变化被监控但没有被有效传播。体系结构化视图引用叠印组合规则，它需要检查每个组合。Suzuki 和 Yamamoto 模型只提供对可变化能力的部分支持。不过 UXF/a 支持高效的变化传播。

除了 Suzuki 和 Yamamoto 的模型，所有的方法都支持高可重用设计的生成，使用了不同的实现技术来实现(高度的灵活性)。另一方面，Suzuki 和 Yamamoto 的模型支持在 UXF/a 兼容工具间方面描述的互换。

独立的设计，包括组合模式和超空间，具有高度的可理解性，但集成的设计却不行。在 AOCE 中，方面帮助理解组件间的关系，但设计却由于高度的交叉引用而变得混乱。相似的，在体系结构化视图中，组合设计由于交叉引用注解而变得混乱，而在 Suzuki 和 Yamamoto 的模型中方面是分离的，它们仍然需要引用基本的对象元素，因此降低了可理解性。

附加运行时刻的可配置性和动态性支持 AOCE 的易用性，但附加的设计工作却妨碍它的易用性。学习组合模式和超空间是比较复杂的，因为需要学习面向主题的设计和相关的组合语义。体系结构化视图要求理解使用单一的和组合的视图和叠印。这与 Suzuki 和 Yamamoto 的模型也很相似，由于它严重依赖于 AspectJ 实现模型，因此需要理解 AspectJ。

AOCE、组合模式、体系结构化视图和超空间支持并发开发。横切关注点和基本设计能够并发开发并彼此独立。Suzuki 和 Yamamoto 的模型对此目的只提供部分支持，因为当设计横切关注点时，设计者需要引用他们无法横切的对象。

使用哪种设计方法需要依据每一个项目前面所提及的设计质量要求来作出。

3 实现

在实现层次对AOSD的初始关注产生了许多编程方法。AspectJ[1]是一种基于Java的方面语言。它支持独立于Java类代码的方面代码的形式化表示。方面代码的编写是采用在动态对象调用图中对联接点的引用来实现的。它提供了一种方面编织器，它组合了方面和类，还以IDE(集成开发环境)扩展的形式提供了附加开发的支持。对其他语言的AOP扩展当前也被开发出来。例如AspectS[18]是对Smalltalk的扩展，AspectC[19]是像AspectJ一样对C语言的AOP扩展。

(1)组合过滤器方法[2]使用输入和输出过滤器扩展了对象。这些过滤器可以是不同类型(例如，分发过滤器用来分发消息，而等待过滤器用来缓存消息)，可以用来局部化非功能性代码。JAC(Java方面组件框架)[20]采用了反射和封装器来支持基于Java的动态方面组合。还有几种其它框架支持动态方面组合，例如AspectWerkz[21]和JBoss[22]。

(2)自适应编程[3]是AOP的一个特例，其中的一种构建模块是以图的形式表示的。其它构建模块以遍历策略引用这些图。一个遍历策略可以看作是一个类图的部分描述。该遍历策略横切了类图。取代对这些类中硬性的结构化知识路径，这种知识是相对独立的。

(3)Hyper/J[4]是支持超空间方法的编程工具[12]，它通过将每个关注点表示成超切片的部分对象模型，在实现层次引入了多维关注点分离概念。超切片通过声明性组合规则被组合成超模块。这允许对多重、潜在重叠的和交互的关注点同时进行分离，这是通过在任何时间封装新关注点和对按需重新模块化的支持实现的。

图 1 描述了在实现一个应用时，标识一个合适的面向方面编程技术的过程。大部分面向方面的编程技术支持一种或者多种基本的编程语言。例如，方面语言可以采用Java(AspectJ)，C(AspectC)和Smalltalk(AspectS)来实现横切关注点的分离。Hyper/J工具支持基于Java使用超空间方法实现多维关注点的分离。在另一方面，组合过滤器方法曾经采用Java，C++和Smalltalk实现过。因此，第 1 步选择合适的技术就是要减少在应用开发中采用的基本编程语言的集合数目。如果对基本编程语言没有合适的技术，然后才去实现一个定制语言和编织器，就像SADES对象数据库演化系统 一样[24]。

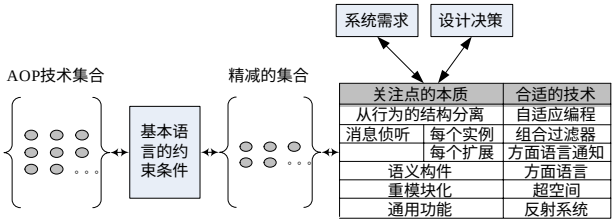

图 1 合适的方面编程工具

在标识一个合适的编程语言技术的第 2 步不像第 1 步那么简单。它是由 3 个不同因素动态组成的(图 1)：被模块化的关注点的本质，系统需求和设计阶段的决策。所有这些因素需要被相互参照，只有这样才能将可用的技术集合减少到最适合应用的一个。

被模块化的关注点的本质，常常会强烈地影响面向方面编程技术的选择。这是因为，尽管不同的技术是互补的，但会有一个最适合实现某个专门的关注点。例如，自适应编程，非常适合表示这样的关注点——它们包含了从对象结构中分离出的行为。它在这样一种环境中最为高效，行为是专门为某个类图的部分描述或者对象结构的一般性描述而编写的(不是与单独示例的专门信息相关)。这种一般性描述例子包括针对面向对象系统中的变化传播和参考完整性语义遍历策略[23]。这些策略需要知道关系的类型或者操作的类型，但无需清楚单独关系示例的专门信息。

组合过滤器对实现消息侦听，以及在一个方法之前和之后执行的动作的关注点十分有效。它们的关键特征就是，对实例的粒度进行操作和基于每个实例基础的附加动作的能力。例如，当在一个面向对象系统中模块化关系时，一个处理关系的分发过滤器能够在这个关系建立起来的时候附加到一个实例上[23]。如果不需要在每个实例基础上进行消息侦听，那么类似于方面语言通知的机制(例如AspectJ)，它基于类的程度进行操作，可能比组合过滤器更为合适。

方面语言都非常适合实现需要使用语法构造符显式表示的关注点。如果某个方面语言是自然声明的，那么表达这样的关注点就非常简单。

其他技术都有着各自的长处。例如，Hyper/J 在对一个遗留应用进行重新模块化的环境中非常有效。另一方面，JAC 的反射式体系结构，使它非常适合开发通用性的方面，能够跨越不同的应用进行重用。不过，如前所述，技术的选择还受到系统的需求和所选择的设计所约束。动态方面，例如 JAC(或者像 AspectWerkz 和 JBoss 框架)只适合方面需要在运行时刻被适配和重新组合的环境。如果一个方面不需要在运行时刻做任何定制，那么出于性能优化考虑，最好采用这样的技术，方面的生存没有超越编译时刻，或者只在运行时刻内省时进行了某种程度的具体化，例如 AspectJ。设计决策也会有类似的影响作用于合适技术的选择。例如，在 SADES 的实现中，将实体间链接的信息分离成关系对象的设计决策，它规定了这些在每个实例基础上被附加到专门实体上的关系对象，因此导致了对组合过滤器的选择。

4 结束语

面向方面软件开发在其短暂的历史中，一直被面向方面的语言如 AspectJ 所驱动发展。换句话说，AOSD 在大体上是与 AOP 同步发展的。尽管面向方面语言帮助面向方面技术普及化了，但它所处于的并不是一个长期健康的位置，只有更多完善的 AOSD 软件工程技术的不断应用才是至关重要的。本文的目的就是为这个方向给出一个指南。

本文尽可能提供了评估不同技术的定性分析矩阵，从而为软件开发者作出选择提供了指南。笔者的观点是没有完美的解决方案。相反，所有的技术都有自己的长处和弱点，这些是依赖于通过领域专有的方式所限定的，领域专有的方式又是与客户组织的需求和可用工具支持的开发约束条件相关的。而且，混合解决方案常常是最合适的，它利用了基于不同系统开发部分的不同方法的最好特点。考虑这样的混合解决方案和相关组合问题是Lancaster当前正在研究的关键内容[23]。这些考虑应该巩固面向方面软件开发过程，并使得从早期的方面到后期的设计、实现的可追踪性的高效支持，都能够被集成到这样一个混合环境之中。只有这样，面向方面技术才能被充分利用来开发良好工程化的面向方面系统。

参考文献

1 Aspectj Project. 2005. http://www.eclipse.org/aspectj/[Z].

2 Bergmans L, Aksit M. Composing Crosscutting Concerns Using Composition Filter[J]. Communications of the ACM, 2001, 44(10): 51-57.

3 Lieberherr K, Orleans D, Ovlinger. Aspect-oriented Programming with Adaptive Methods[J]. Communications of the ACM, 2001, 44(10): 39-41.

4 Ossher H, Tarr. Multi-dimensional Separation of Concerns for Java[C]//Proceedings of International Conference on Software Engineering, Limerick, Ireland. 2000-06: 734-737.

5 Baniassad, Clarke, Theme: An Approach for Aspect-oriented Analysis and Design[C]//Proceedings of the 26th Int'l Conf. on Software Engineering, Edinburgh, Scotland. 2004: 158-167.

6 Grundy J. Aspect-oriented Requirements Engineering for Component-based Software Systems[C]//Proceedings of the 4th IEEE International Symposium on Requirements Engineering. 1999: 84-91.

7 Rashid A, Sawyer P, Moreira A, et al. Early Aspects: A Model for Aspect- oriented Requirements Engineering[C]//Proc. of Joint Int'l Conf. on Requirements Engineering, Essen, Germany. 2002: 192-202.

8 Sommerville I, Sawyer, P. Requirements Engineering——A Good Practice Guide[M]. John Wiley & Sons, 1997.

9 Clarke S, Walker R J. Composition Patterns: An Approach to Designing Reusable Aspects[C]//Proc. of the 23rd Int'l Conf. on Software Engineering, Toronto. 2001: 5-14.

10 Grundy J. Multi-perspective Specification, Design and Implement-ation of Software Components Using Aspects[J]. Int'l Journal of Software Engineering and Knowledge Engineering, 2000, 10(6): 713-734.

11 Suzuki J, Yamamoto Y. Extending UML with Aspects: Aspect Support in the Design Phase[C]//Proceedings of ECOOP'99 Workshop on Aspect-oriented Programming . 1999: 299-300.

12 Tarr P, Ossher H, Harrison W, et al. N Degrees of Separation: Multi-dimensional Sparation of Concerns[C]//Proc. of the 21th Int'l Conf. on Software Engineering. 1999: 107-194.

13 Katara M, Katz S. Archetectural Views of Aspects[C]//Proc. of the 2nd Int'l Conf. on Aspect-orientd Software Development. 2003: 1-10.

14 Clark S, Harrison W, Ossher H, et al. Subject-oriented Design: Towards Improved Alignment of Requirements, Design and Code[C]//Proc of the 14th Conf. on Object-oriented Programming, Systems, Languages, and Applications. 1999: 325-339.

15 Clarke S, Walker R J. Towards a Standard Design Language for AOSD[C]//Proc. of the 1st Int'l Conf. on Aspect-oriented Software Development. 2002: 113-19.

16 IBM Research. Hyperspaces[Z]. 2000. http://www. research.ibm.com /hyperspace/.

17 Sihman M, Katz S. Superimpositions and Aspect-oriented Programming[J]. Computer Journal, 2003, 46(5): 529-541.

18 Hirschfeld R. AspectS Home Page[Z]. http://www.prakinf.tu-ilmenau.de/~hirsch/Projects/Squeak/AspectS/.

19 Aspectc Web Page[Z]. 2001. http://www.cs.ubc.ca/labs/spl/ projects/ aspectc.html.

20 Pawlak R, Seinturier L, Duchien L, et al. JAC: A Flexible Solution for Aspect-oriented Programming in Java[M]. Springer Verlag, 2001: 1-25.

21 Boner J. What Are the Key Issues for Commercial AOP Use——How Does Aspectwerkz Address Them[C]//Proc. of the 3rd Int'l Conf. on Aspect-oriented Software Development. 2004: 5-6.

22 Jboss. Aspect-oriented Programming[Z]. 2005. http://www.jboss.org/ developers/projects/jboss/aop.

23 Rashid A. A Hybrid Approach to Separation of Concerns: The Story of SADES[C]//Proc. of the 3rd Int'l Conf. on Metalevel Architectures and Separation of Crosscutting. Berlin: Springer-Verlag, 2001: 231-249.

24 Rashid A. Weaving Aspects in a Persistent Environment[J]. ACM SIGPLAN Notices, 2002, 37(2): 36-44.

25 Dardenne A, Van Lamsweerde A, Fickas S. Goal-directed Requirements Acquisition[C]//Proc. of the 6th Int'l Workshop on Software Specification and Design. 1993: 3-50.

26 Yu E. Modeling Strategic Relation Ships for Process Reengineering [D]. University of Toronto, 1995.

27 Jackson M. Problem Frames: Analyzing and Structuring Software Development Problems[M]. Addison-Wesley Publishing Company, 2001.