IBM Rational

可视化建模技术
IBM Rational 技术白皮书

版本 1.0

Rational software

IBM

目录

Rational. software

IBM

可视化建模技术

1. 为什么要建模

自从 90 年代中期对象管理组织 OMG 发布统一建模语言 UML(Unified Modeling Language)以来，伴随着面向对象技术的发展，可视化建模技术受到越来越多开发人员的追捧。并且从原先的软件设计领域逐渐扩展到业务流程重构等新领域，使用人员也从专业软件人员扩展到非专业的业务人员，可视化建模技术已经成为一种成熟标准的软件开发技术规范。

1.1 什么是模型？

1.1.1 模型是对现实世界的简化和抽象

现实世界中的系统是纷繁复杂的，直接去认识现实世界并且解决其中的问题是非常困难的。所以人们往往会构造一个模型来对现实世界中的复杂系统进行简化和抽象，通过这种简化和抽象来帮助设计人员加深对于系统的认知，在进行简化和抽象时我们抓住的是问题的本质，而过滤掉很多其他非本质的因素，从而帮助我们来简化问题的复杂性，有利于问题的解决。

模型在现实世界中大量存在，无论是研制飞机还是制造汽车，设计师们都会利用模型来研究目标课题的某一个侧面，如汽车的风阻系数、飞机机身的空气动力布局等等。在研发过程的大部分阶段中，设计师都不会去构造一个真实的系统来进行研究，因为这样的话成本太高了（或甚至是不可能的），同时问题本身没有得到足够的简化，很难找到问题的正确答案。

1.1.2 模型是沟通的手段

我们平时所见的模型有的是一种概念上的模型，如刚才提到的数学模型；有的是对实际系统外观的一个缩小，如轮船、飞机模型；还有的是对设计思想的一种展示，如建筑物的设计图纸等等。无论是哪一种模型，它的另外一个主要目的是帮助人们进行思想上的沟通，数学模型使别人了解你的逻辑思路，飞机模型向观众展示飞机的外观，设计图纸将设计师的设计思想传递给建筑工人。

语言和文字是人们进行沟通的主要手段，但语言和文字往往有二义性存在，较难保证人们的理解完全一致。所以在工程技术中，我们更多地是使用各种各样的模型来进行思想的沟通，模型可以精确地描述系统，同时保证整个系统开发过程的语义的一致性。

1.2 什么是可视化建模

软件系统也是一种非常复杂的系统，它的最终表现形式为可运行的目标代码。但是最终的软件代码是非常复杂的，包含了太多的细节信息，直接阅读代码很难对系统有一个全面的了解。我们需要有一个中间过程来得到这些结果，同时也需要对系统进行简化和抽象，这就是我们通常所说的系统设计。利用统一建模语言 UML 来对系统结构进行全面的分析设计，即构建系统模型的过程，这就是可视化建模(Visual Modeling)。

在可视化建模的过程中，我们应用的是面向对象的方法。面向对象方法的主要思想是把现实世界中需要解决的问题（业务流程、控制逻辑等）映射到计算机软件系统中去，利用软件世界中的对象来代表现实世界中的实体，并利用对象之间的交互来描述现实世界中实体之间的动态关系,从而帮助我们通过软件来解决现实世界中的问题。与传统的分析设计方法（结构化设计、面向过程的设计等）相比，对象模型是一种对于现实世界最为直接的映射，当需要解决的问题发生变化时（如业务需求发生变化），对象模型作适当的改动就可以迅速地适应这一变化。

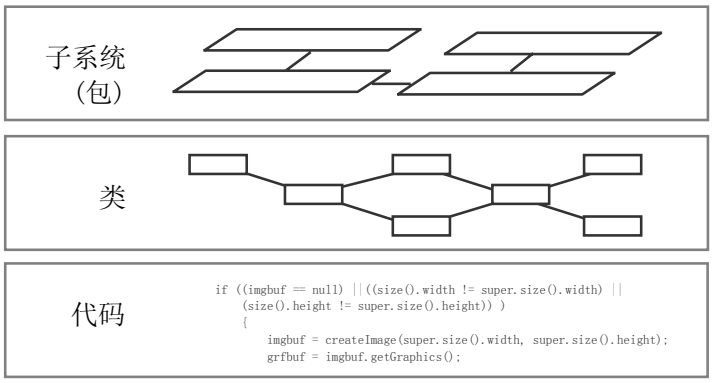
可视化建模技术利用 UML 这一统一的符号语言，可以把对象模型准确地表达出来，便于开发人员之间的沟通。UML 语言统一了以前各种不同面向对象方法（如 Booch、OMT 等），是对前人经验的总结。UML 语言提供了非常丰富的描述手段,包括用例图、类图、对象图、组件图、部署图、序列图、协作图、状态图、活动图等，分别从静态结构和动态行为两个角度来对系统进行全面的描述。

1.3 可视化建模技术的好处

1.3.1 有效管理系统复杂度

面向对象方法最基本的原则就是抽象，把一类具有相同属性和行为的实体抽象成为一个类(Class)，再通过把类实例化成对象(Object)来映射现实世界中的某一个具体实体。对象通过操作(Operation)来对外对供相应的服务，在对象模型中我们只需要描述对象所实现的功能，而封装了操作实现的细节。与软件代码相比，对象模型描述的也是同一个系统，但它展示的是系统结构中最关键的元素以及它们之间的关系，所有的编码细节都已经被忽略掉了，从而有利于开发人员把握理解整个系统。

复杂的对象模型往往由成千上万个模型元素构成，UML 中的包(Package)可以很好地管理这些模型元素，我们在设计过程常用的一些概念如子系统、模块等概念也可以用包来表示。所以整个系统的对象模型就呈现出一种层次结构，当你想要了解系统的整体结构时，你看到的是子系统之间关系；你想了解子系统内部的设计时，你又可深入到子系统内部观察它的内部结构。在这种层次结构中，每一层所包含的信息量都是有限的，以便于每一位开发人员的理解；但随着工作的需要，开发人员随时可以了解更深层次的设计细节。



1.3.2 增强团队的沟通

对象模型同时也作为软件设计的蓝图，记录了开发人员的设计思想。对于设计者而言，对象模型提供了一个工具来帮助他来整理设计思路，整个的设计过程都可以被记录下来；同时，也避免开发者在整个系统架构明确之前就陷入编码的细节之中，对于模型的调整修改相对于代码的改动要简单得多。

另一方面，对象模型也使得设计的结果很容易被其他人所理解，设计者的设计意图可以被完整的传递而不发生信息的失真。可视化建模采用的是标准的统一建模语言 UML，所有的开发人员都应该采用这种统一建模语言来进行系统的设计，从而保证大家工作的结果是所有人都可以理解的。这也是 UML 语言的设计目的之一，即使用 UML 来统一整个开发团队的沟通手段。

1.3.3 提高系统设计的可重用性

面向对象技术最基本的原则就是抽象，即把整个系统的功能尽可能地分配到多个类中去，每个类应该只做并且做好一件事情。因为每个类实现的功能比较单一，所以可以有更多的机会被重用。同时尽量利用构件化的思想把关系比较紧密的类组合成构件，构件具有定义明确的功能并且以接口的形式对外提供服务。基于构件的架构具有最大的可重用性，一方面可以重用现有的商业构件来搭建系统，另一方面当前系统中的构件也可以被其他的系统所重用。

1.3.4 增强系统架构的灵活性

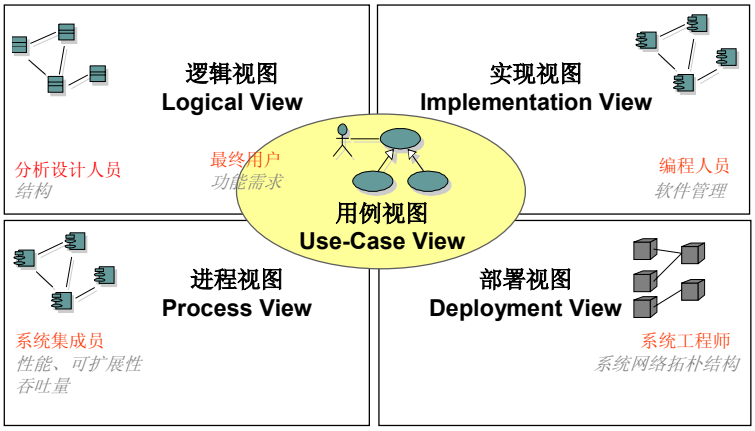
好的系统架构应该是具有最大的灵活性，即它不仅能满足系统目前的需求，更重要的是它还可以满足系统将来的需求。可视化建模技术可以从以下几个方面增强系统架构的灵活性。

- 把系统中易变的功能和不变的功能分开，把跟需求相关的功能和通用的服务功能分开。这样当系统需求发生变化的时候，我们可以使大部分的系统结构保持不变，变动地只是局部。
- 尽可能使用接口来调用另一对象所提供的服务，而不是为了一些所谓的“效率”而直接访问对象内部的数据结构。当必须修改某一对象的功能

时，只要该对象实现的接口不发生变化，该对象的使用者就不需要做任何的变化。

2. 可视化建模方法

设计一座建筑需要从多个不同的角度（结构、外观、水电等）来设计很多张设计图纸，开发一个软件系统同样需要从多个角度来对系统架构进行完整的设计。Rational 统一开发流程 RUP (Rational Unified Process) 采用了“4+1 View”模型来进行可视化建模工作，“4+1 View”指的是：用例视图、逻辑视图、进程视图、实施视图、部署视图（如下图所示），这几种视图从不同的角度来对系统进行完整的描述。它们在 RUP 中被称为“架构视图(Architecture View)”，即通过这样几种视图可以完整地展示系统的架构。



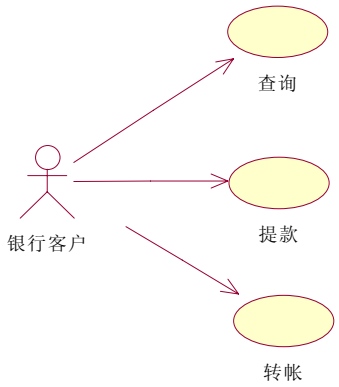
2.1 用例视图(Use-Case View)

用例视图是“4+1 View”模型的核心，它定义了系统的功能需求。用例模型主要由以下模型元素构成:

- 参与者(Actor)
  参与者是指存在于被定义系统外部并与该系统发生交互的人或其他系统，他们代表的是系统的使用者或使用环境。
- 用例(Use Case)
  用例用于表示系统所提供的服务，它定义了系统是如何被参与者所使用

的，它描述的是参与者为了使用系统所提供的某一完整功能而与系统之间发生的一段对话。

下面的用例图所展示的就是银行自动提款机(ATM)的例子。ATM 的主要使用者是银行客户，客户主要使用自动提款机来进行银行帐户的查询、提款和转帐交易。对于用例模型来讲，用例图只是展示了系统功能的一个总体概貌，每一个用例还应该有详细的文字描述（用例规约）。



用例方法完全是站在用户的角度上（从系统的外部）来描述系统的功能。在用例方法中，我们把被定义系统看作是一个黑箱，我们并不关心系统内部是如何完成它所提供的功能的。用例方法首先描述了被定义系统有哪些外部使用者（抽象成为 Actor），这些使用者与被定义系统发生交互；针对每一参与者，用例方法又描述了系统为这些参与者提供了什么样的服务（抽象成为 Use Case），或者说系统是如何被这些参与者使用的。所以从用例图中，我们可以得到对于被定义系统的一个总体印象。

用例方法完全是从外部来定义系统的功能，它把需求与设计完全分离开来。在可视化建模技术中，用例模型主要用于表述系统的功能性需求，系统的设计主要由对象模型来记录表述。另外，用例定义了系统功能的使用环境与上下文，每一个用例描述的是一个完整的系统服务。用例方法更易于被用户所理解，它可以作为开发人员和用户之间针对系统需求进行沟通的一个有效手段。

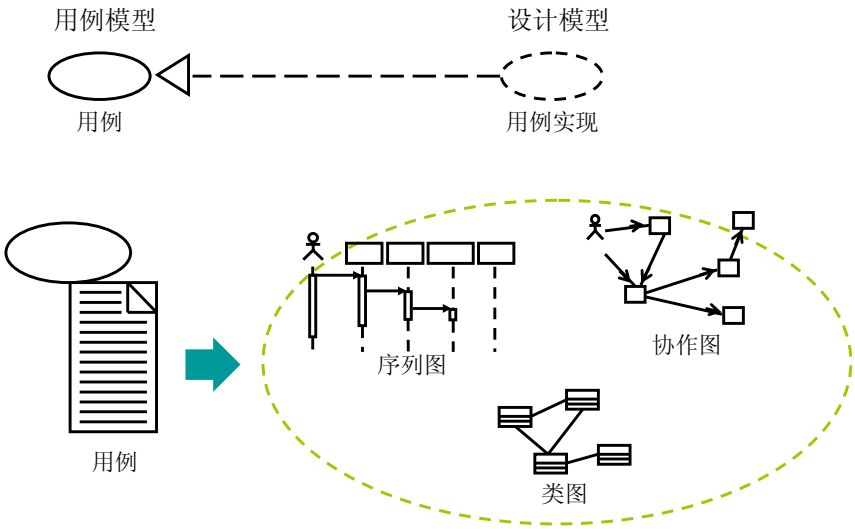
在 RUP 中，用例被作为整个软件开发流程的基础，很多类型的开发活动都把用例作为一个主要的输入工件(Artifact)，如项目管理、分析设计、测试等。如根据用例来对目标系统进行测试，可以根据用例中所描述的环境和上下文来完整地测试一个系统服务，可以根据用例的各个场景(Scenario)来设计测试用例，完全地测试用例的各种场景可以保证测试的完备性。

2.2 逻辑视图(Logic View)

系统的主要设计都在逻辑视图中展示，构建系统模型过程中最重要的工作就是用例实现(Use-Case Realization)。如前所述，在可视化建模技术中，需求是由用例来描述的，开发一个软件系统的过程就是要找出一组对象，利用这些对象之间的相互协作来实现用例所描述的系统需求。所以分析设计的过程，就是实现用例的过程；什么时候所有的用例都实现了，什么时候系统的设计也就完成了。

在用例实现的过程中，我们主要是从静态和动态两个维度来刻画系统的架构设计。一方面我们要确定有哪些类参加了该用例的实现，这些类之间的关系怎样；另一方面，我们必须描述这些类的实例（对象）之间是如何相互交互来实现用例中所定义的系统功能的。在可视化建模技术中，我们用类图来描述类之间的静态关系，用交互图（序列图和协作图）来描述对象之间的动态交互，每一个用例实现都是由一个或多个的类图和交互图组成的。

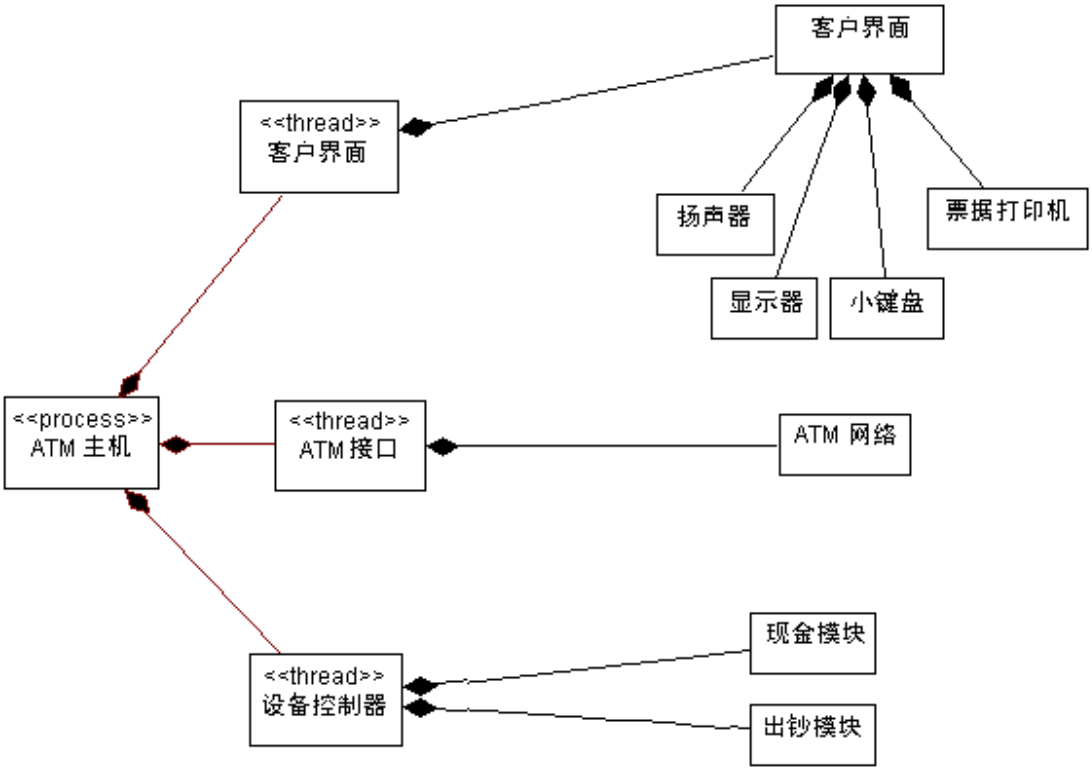
在模型中，我们用一个虚线的椭圆来表示用例实现，下图中所表示的就是一个用例实现和用例之间的实现关系（表示为一个虚线加一个空心箭头）。在分析设计的过程中，每一个用例都要有至少一个用例实现与之相对应；因为同一种需求可能有多种不同的设计实现，所以有的用例可能有多个用例实现，如一种是 Windows 客户端界面实现，另一种为 Web 浏览器界面的设计实现。在每一个用例实现中，都必须包括至少一个的类图，用来展示参与该用例实现的类之间的关系；需要有多个交互图来描述如何实现用例的不同场景(Scenario)，一个交互图展示一个场景或一组场景的实现。



用例实现在需求和设计之间搭起了一座桥梁，通过用例实现我们展示了设计架构是如何来满足系统需求的，每个模型元素分别是为哪些用例提供了服务。另一方面，用例实现也记录了设计的一个中间过程，我们最终需要的是模型中的类，用例实现记录了这些类是如何被设计出来的过程，便于我们理解系统的设计以及对设计结果进行审视。

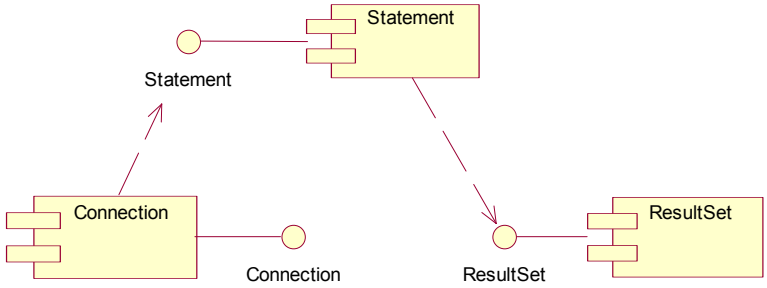
2.3 进程视图(Process View)

进程视图展示的是系统运行时的动态逻辑，它所关注的是系统的运行效率。我们主要是从系统性能的角度出发，考虑系统中应该有那些进程和线程，寻找出时间和空间的最佳平衡点，以使系统的性能能够满足需求。下图是一个关于 ATM 的进程描述，可以看到，ATM 机运行时存在三个线程，每一个线程负责控制不同的功能模块。



2.4 实施视图(Implementation View)

实施视图中主要考虑的问题是如何将逻辑视图和进程视图中所包含的系统设计转换成最终的编码实现，这部分工作是编程人员比较关心的问题。构件是实施视图中的一个主要概念，它代表系统中的一部分物理实施，包括软件代码（源代码、二进制代码或可执行代码）或其他等价物（如脚本或命令文件）。构件图表示了系统构件之间的相互关系，如下图所展示的就是 JDBC 中一些主要构件之间的关系，Connection 会产生一个 Statement，而执行一个 Statement 又会产生一个 ResultSet。

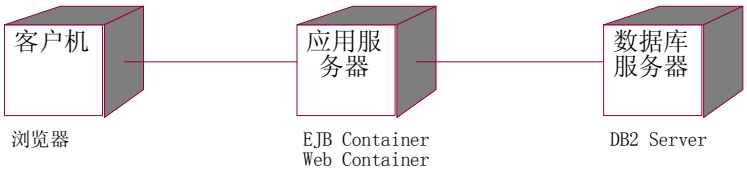


实施视图的主要作用是为实施制定构架决策，实施视图通常包括以下内容:

- 列举实施模型中的所有子系统及其所包含的构件;
- 说明子系统如何组织为层次和分层结构的构件图;
- 说明子系统间的导入依赖关系的图解。

2.5 部署视图(Deployment View)

对于系统工程师而言，他最关心的是系统的网络拓朴结构，有多少台服务器，有多少台客户机，它们之间的关系如何，发布得到的软件制品该如何部署到这些平台上去。部署视图中用到的 UML 工具就是部署图，如下图所示是一个典型的 J2EE 体系统架构：用户通过客户机上的浏览器来访问应用服务器，应用服务会与后台的数据库服务器联系来访问数据库中的数据。



3. 可视化建模最佳实践

3.1 建立以构件为基础的软件架构

在应用可视化建模技术进行系统分析设计的过程中，一个重要的实践经验就是建立基于构件的架构(Component-Based Architecture)。构件技术将为软件系统提供最大的灵活性，构件的特点是：构件内部的元素之间高内聚、构件之间低耦合，整个系统的功能由一组构件相互协作而完成，构件之间通过定义明确的接口来使用相互的服务。构件体现了面向对象技术封装(Encapsulation)和模块化(Modularity)的思想，既能够适当隐藏不需要关心的细节，又可以在需要的时候随时展现设计的详细结构，从而有效管理系统的复杂度。

另一方面，系统功能是由一组相互协作的构件提供的，系统功能的增加修改只需要增加新的构件或改动相关的构件，使得系统的扩展性大大增强，能够迅速地适应业务快速变化的需求。构件技术也大大提高了系统开发的重用水平，不但可以重用自己开发的构件，也可以重用其他的或商业化的构件

3.2 保证模型和代码之间的一致性

在传统的软件开发过程中，一个最常见的问题是很难保证系统设计和代码之间的一致性。开发人员在进入到编码阶段后，为了满足编程工具的约束或基于非功能性需求（性能、可靠性、可维护性等）的考虑，必然会对系统的设计进行一定的修改，但这种修改很少会反映到系统设计中，而直接体现在系统代码中。这就造成了系统设计和代码之间的脱节，设计文档已经不再精确反映系统的实际情况，使得系统在日后难于维护。在迭代化的开发流程中，这种问题会更加显著，因为每一次迭代都是在上一次迭代的基础上进行的，这种设计和代码的不一致性会严重影响到下一次迭代的开发。

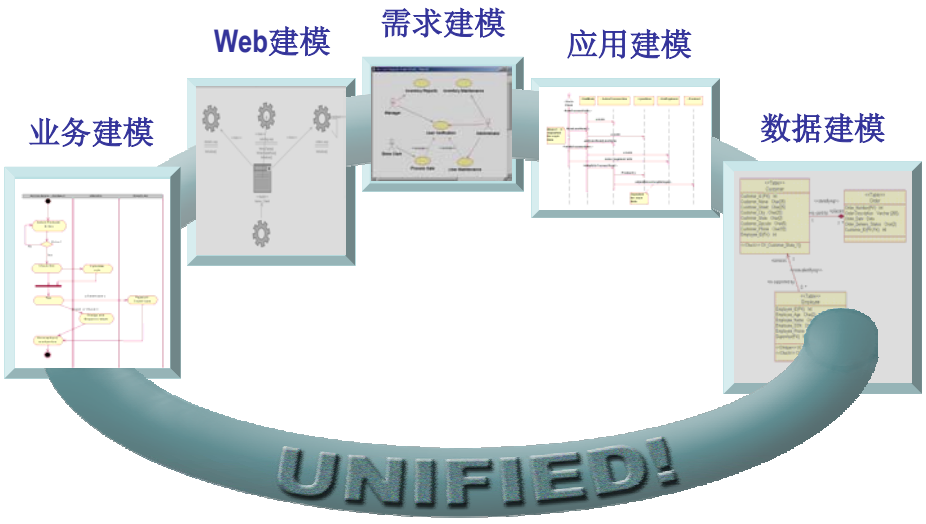
由于传统设计方法在语义上缺乏严格的定义，模型和代码之间的一致性必须由手工来维护。在可视化建模技术中，由于 UML 语言的语义没有二义性，可以由工具来实现这种模型和代码之间的同步，从而很好地保证模型和代码之间的一致性。下文中将要介绍的 IBM Rose 和 XDE 都具有这样的同步功能，这种功能包括了从模型自动生成代码的正向工程(Forward Engineering)技术，以及根据代码变化来自动地更新模型的逆向工程(Reverse Engineering)技术，统称为双向工程 RTE(Round-Trip Engineering)。

3.3 使用 UML 统一软件开发生命周期

在项目开发的开始阶段，开发团队应该对客户未来的业务有一个全面地了解。我们可以通过业务建模(Business Modeling)来达到这一目的，通过 UML 来建立完整的业务流程描述，使得开发团队和业务部门能够对软件系统所处的业务背景有一个全面而一致的认识，在此基础上再进一步抽取具体的系统需求。

对于很多涉及到数据库的应用软件，以往我们都是用实体关系图(ER 图)来描述数据库的内部结构。现在利用 UML 我们可以更好地描述数据库内部的表结构，并利用双向工程技术来从现有的数据库或数据定义语言 DDL 中抽取数据库表结构并建立模型，或是根据定义好的数据模型在数据库中自动产生相应的表结构，这种技术称为数据建模(Data Modeling)技术。

所以软件开发的整个生命周期（包括业务建模、用例建模、应用建模、数据建模）都可以用可视化建模技术统一起来。在传统的开发技术中，这些步骤是由不同的技术完成的，如业务模型是由 IDEF 语言来描述，分析设计由数据流图来表示，数据库结构是用 ER 来定义等等。在可视化建模技术中，所有的这些开发活动都可以由同一种语言 UML 来描述，这样可以大大增强团队的沟通，提高开发效率和软件质量。



用同一种语言统一整个开发流程

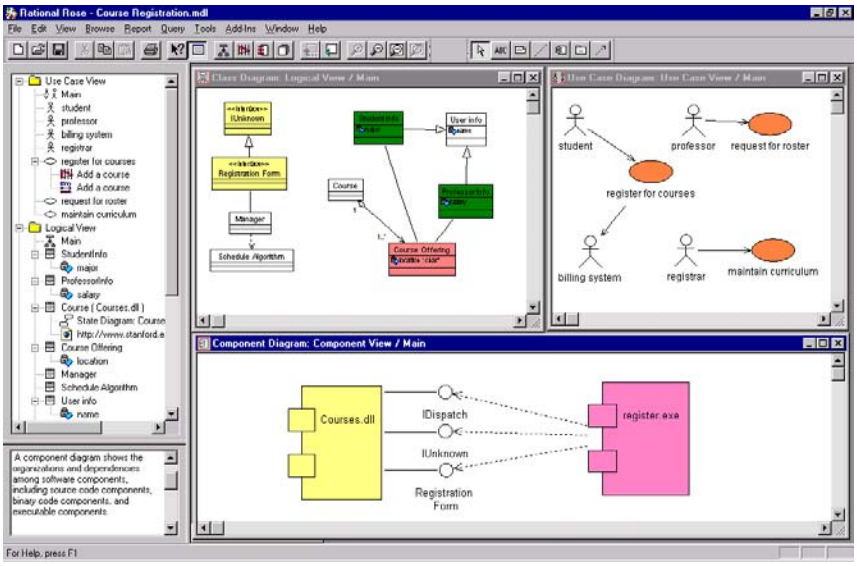
4. 可视化建模工具

市场上有很多支持可视化建模技术的工具，IBM Rational 的 Rose 和 XDE 是其中的佼佼者。从 UML 创始至今，Rose 始终保持着市场占有率第一的位置；而 XDE 则是 IBM Rational 最新研发的的新一代建模工具。无论是 Rose 还是 XDE，它们都有着对于 UML 最全面的支持。

4.1 最受欢迎的建模工具 Rose

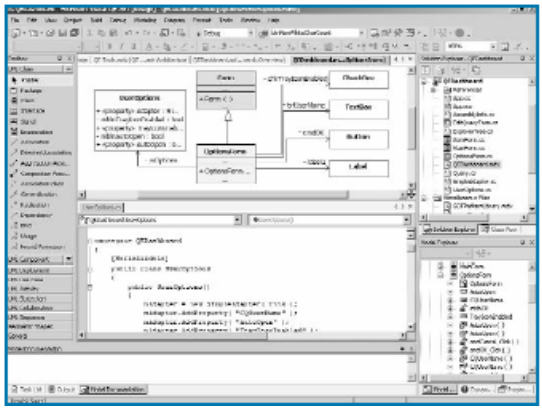
Rational Rose 是一个非常经典的可视化建模工具，它已被 IDC 连续五年评为业界最佳的可视化建模工具，在全球拥有最多的使用者。它全面支持 UML 建模标准，可以在同一个模型中实现业务建模、需求建模、应用建模和数据建模。

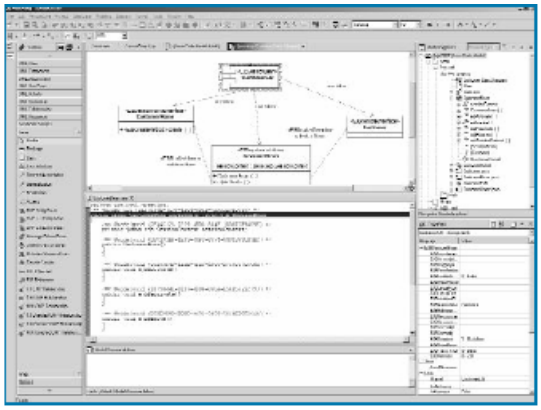
Rose 支持多种语言（C++, Java、VB 等）的代码生成及双向工程，实现代码和模型的互相转换，并且可以将历史代码引入模型中。Rose 也在业界最早提出了基于模型的测试概念，通过自带的的模块测试工具 Quality Architect，可以在设计阶段就及早对设计模型进行测试，尽早发现设计中的问题，真正实现“质量从头抓起”。



4.2 新一代建模工具 XDE

IBM Rational XDE Developer 是新一代的可视化建模工具，它是对传统 IDE 开发工具的扩展，XDE 的含义就是扩展的开发环境（eXtended Development Environment）。使用户能够在同一种开发环境下进行建模和编码工作，因此避免了在建模工具和 IDE 工具之间的切换。对于目前没有 IDE 开发环境的用户，它提供了内置的 IBM Eclipse IDE；对于已经使用 IBM WebSphere Studio Application Developer 的用户，它又可以被无缝地嵌入到 WSAD 开发环境中。对于.NET 的开发人员，XDE 也支持 Microsoft Visual Studio.NET 开发环境。





XDE 为软件开发人员提供了一种模型驱动的开发方法，用于快速地建立应用程序，它的主要特点体现在以下几个方面：

- 在 IDE 内完成更多工作
  传统的建模工具与开发环境 IDE 的耦合非常松散，它们拥有自己的用户界面，开发人员必须按 ALT-Tab 键在两种环境之间切换，这样就形成了一种笨拙和低效的用户体验。XDE 通过将其自身嵌入到 IDE 环境中，使可视化建模成为 IDE 环境不可缺少的组成部分，使开发人员能够在 IDE 内完成更多工作。
- 更迅速地开发可靠的代码
  开发人员往往希望以更快的速度编写优秀、可靠的代码，他们常常担心建模工具会减缓开发速度。XDE 能够帮助开发人员以更快的速度编写更好的代码。XDE 中包含了独有的代码模板(Code Template)和模式(Pattern)功能，允许代码级和模型级的重用，从而大大加快编码的速度和质量。
- 运行时分析工具
  XDE 的增强版 XDE Developer Plus 提供了运行时分析工具，该工具能够检测内存故障、显示应用程序性能瓶颈、产生代码覆盖率报告，它还可以可视化地追踪软件运行逻辑，以序列图的形式展现该运行逻辑。这些功能将模型驱动的开发扩展到开发人员的测试功能，极大地方便了开发人员快速地开发高可靠代码。
- 使团队开发更简便
  XDE 还可以与 IBM Rational 的其他开发工具相集成，更大地提高团队开发的效率。如 XDE 与需求管理工具 RequisitePro 集成，可以查看和管理持续变化的需求；与 ClearCase 集成，从 IDE 的内部来提供版本控制功能；与 ClearQuest 集成，帮助开发人员即时创建并追踪缺陷报告。此外，XDE 还提供了 Web 发布和其它报告功能，方便开发人员与项目内外的其他人进行交流。