第 1 章

引　言

何谓 UML?

统一建模语言 UML 是由单一元模型支持的一组图示法。这些图示法有助于表述与设计软件系统，特别是采用面向对象(OO)方法构作的软件系统。这是一个稍微简化的定义。事实上，UML 却多少因人而异。这一方面是源于 UML 本身的历史；另一方面，是源于人们关于构作有效软件工程过程所持的不同观点。因此，本章的大量工作是通过解说人们看待与使用 UML 的不同方式来为本书的后续部分进行准备。

图示建模语言在软件工业中已经长期存在。其产生与发展的基本动力在于对促进有关设计的讨论来说，编程语言的抽象级别还不够高。

虽然图示建模语言存在已久，但在软件工业中对其作用却颇有争议。这些争议直接影响到人们如何看待 UML 的作用。

UML 是由对象管理组(OMG)管理控制的一种较为开放的标准。OMG 是由各家公司组成的一个开放联合组织。组建 OMG 的目的是构作支持互操作性(特别是面向对象系统的互操作性)的标准。也许,OMG 以 CORBA(公共对象请求代理体系结构)标准而享有盛名。

UML 是在经过统一 20 世纪 80 年代后期与 90 年代初期成长起来的很多面向对象图示建模语言而诞生的。自从它在 1997 年出现,便将特定的传统巴别(Babel)塔抛入历史。这是一项令我以及很多别的开发人员都深感欣慰的贡献。

UML 的使用方式

UML 在软件开发中作用的实质是人们不同的使用方式。这些不同的使用方式传承于其他图示建模语言。它们导致了关于应如何使用 UML 这一问题的长期而艰巨的争论。

为了解决这一纷争,Steve Mellor 和我独立提出了 UML 三种使用方式的分类法:用作草图绘制语言,用作蓝图绘制语言以及用作程序编制语言(即,编程语言,程序设计语言)。迄今,至少依我的偏见看,这三种方式中最为常用的是将 UML 用作**草图绘制语言**(UML as sketch)。在这一用法中,开发人员使用 UML 帮助系统中某些部分进行交往。和对蓝图绘制语言一样,你可以按正向工程方向或逆向工程方向来使用草图绘制语言。**正向工程**(forward engineering)在编写代码前绘制 UML 图,而**逆向工程**(reverse engineering)则根据现有的代码来绘制 UML 图,以助于

理解代码。

绘制草图的实质是选择。对正向工程中的草图绘制，你草拟出要编写的代码中的问题，通常要和你的项目组中的同事进行讨论。你们的目的是，使用草图去帮助交流你们要进行工作的想法和取舍，并不涉及要编制的所有代码，而只是那些打算先交给同事去做的或者想在开始编程以前能看到的一些设计段的重要问题。像这样的讨论会可能很短。十分钟的会议讨论几小时的编程，或者一天的会议讨论两周的迭代开发。

对于逆向工程，你利用草图去阐明系统的某一部分如何工作，并不去示明每一类，只是阐明那些在深入到代码以前有意义以及值得讨论的问题。

由于草图绘制是相当非形式的和动态的，你就必须按合作方式快速进行。因此，常用的介质是白板。草图在文档中也有用，在这种情形下，重点是进行交流，而不是苛求完备。用于绘制草图的工具是一些轻便的制图工具。往往人们并不过于遵循 UML 的严格规则，在一些书（像我的别的书）中所示的 UML 图都是草图，其重点在于有选择地交流而不是完备的规约。

与此截然不同的是，**用作蓝图绘制语言的 UML**（UML as blueprint）却强调完备。在正向工程中，蓝图是由设计人员开发的，设计人员的任务是为编程人员编写代码构作一个详细的设计规约，这种设计规约应该足够完备，所有的设计选定都要安排好，编程人员应能按此编程。这是一项相当简单的工作，无须太多的思考。设计人员和编程人员可以是同一个人，但通常设计人员则是更高层的开发人员，他为一组编程人员进行设计。这一途径是受到其他形式的工程的启发。在那些工程中专业工程师创作工

程图,再把这些工程图转交给建造公司去建造。

蓝图绘制可用于所有细节,或者,设计人员可对一特定领域绘制蓝图。通常的途径是,设计人员开发蓝图级的模型直到子系统的接口,然后,让开发人员产生出实现那些细节的细节。

在逆向工程中,蓝图的目的在于表达代码的详细信息,或者是用纸文档,或者是作为交互图形浏览器。蓝图可以用开发人员较易理解的图示形式来示明类的每一细节。

与草图相比,蓝图需要更多的先进工具,以处理任务所需的细节。专业化的 CASE(计算机辅助软件工程)工具就属于这一类工具,即使术语 CASE 已经成为一个令人讨厌的词,而且目前卖主已打算避免使用。正向工程工具支持绘图,并用一仓库支持它保存信息。逆向工程工具读源代码,并据此解释以进入仓库且生成图。既可进行正向工程又可进行逆向工程的工具称为**双向**(round-trip)工具。

有些工具使用源代码本身作为仓库并使用图作为代码上的图形视见区。这些工具和编程联系十分紧密,并且往往和编程编辑程序相集成。我喜欢把这些工具想象成**无行程**(tripless)工具。

蓝图和草图之间的界线有些模糊。但是,我认为区别在于,草图是故意不完备的,要突出重要的信息,而蓝图却打算使内容广泛,并往往具有这样一个目的,即将编程简化成一种简单且相当机械的活动。简言之,草图是探究性的,而蓝图是定义性的。

随着你在 UML 中的工作越来越多以及编程日益机械,变得明显的是,编程应该自动化。的确,很多 CASE 工具进行了某种形式的代码生成,后者使系统的相当一部分构建工作自动化。可是,最终你达到的是,整个系统可以用 UML 来指明,并且把 **UML**

用作程序编制语言(UML as programming language)。在这种环境下,开发人员绘制 UML 图,后者直接编译成可执行的代码,而 UML 成为源代码。显然,UML 的这种用法特别要求使用先进工具(而且,正向工程与逆向工程概念对这种方式已不具任何意义,这是因为 UML 和源代码相同的缘故。)。

模型驱动体系结构与可执行 UML

当人们谈到 UML 时,他们也往往谈到**模型驱动体系结构**(MDA)[Kleppe et al.]。本质上说,MDA 是 UML 用作程序编制语言的一种标准途径。与 UML 一样,该标准是由 OMG 管理控制的。通过产生一种符合 MDA 的建模环境,卖主能够创建也可以在其他遵从 MDA 的环境中工作的模型。

往往在谈到 MDA 的同时也谈到 UML,这是因为 MDA 使用 UML 作为其基本建模语言的缘故。但是,在使用 UML 时,并无须使用 MDA。

MDA 将开发工作分为两个主要方面。建模人员通过创建一个**与平台无关的模型**(Platform Independent Model, PIM)来表示一个特定的应用。PIM 是一个与任何特定技术无关的 UML 模型,而后,使用工具将 PIM 转换成一个**平台特定的模型**(Platform Specific Model, PSM)。PSM 是一个标定为特定执行环境的系统模型,这时,别的工具接受 PSM,并为该平台生成代码。PSM 可以是 UML 模型,但也不一定是 UML 模型。

就要利用一些卖主工具,分别创建两个 PSM,一个平台一个。这时,别的工具便为这两个平台生成代码。

如果由 PIM 到 PSM 再到最终代码的过程完全自动化,UML 便是编程语言。如果其中任何一步是由人手工进行的,UML 便是蓝图绘制语言。

Steve Mellor 长期活跃在这一工作领域,并且近来使用了**可执行 UML**(executable UML)这一术语[Mellor and Balcer]。可执行 UML 与 MDA 类似,但使用了稍微不同的术语。类似地,你也是从一个和 MDA 的 PIM 等价的与平台无关的模型开始,不过,下一步是利用一个模型编译程序把 UML 模型一步转换成一个可部署的系统;因此,使用不着 PSM。正如术语编译程序(compiler)所暗示的,这一步是完全自动化的。

模型编译程序是基于可复用的本型。**本型**(archetype)表述如何接受一个可执行的 UML 模型,并将它转换到一个特定的编程平台。因此,对仓库例子来说,你要购买一个模型编译程序和两个本型(J2EE 和 .NET),将每个本型在可执行的 UML 模型上运行,这样就有了仓库系统的两个版本。

可执行 UML 并不使用 UML 全集标准,UML 的很多构造都认为不是必需的,因而都未使用。这样,可执行 UML 就比 UML 全集简单。

所有这一切,看起来都不错,但其现实程度如何?依我看,这里有两个问题。第一个是工具问题。执行这一任务的工具是否足够成熟。这一问题因时而变。的确,在我写这本书时,这样的工具并未广泛使用,并且我也未曾看到它们当中多数在起作用。

一个更为基本的问题是 UML 用作编程语言的整体看法。依我看,将 UML 用作编程语言仅在如下情形才是值得的,即它会导致比使用其他编程语言更加明显富有成效的结果。基于过去我工作过的各种图示

开发环境，我未能确信它是如此。即使它更加富有成效，仍然需要拥有一大批用户，才能使它成为主流，这本身就是一大障碍。像很多 Smalltalk 老用户一样，我认为 Smalltalk 比当前一些主流语言更富有成效，但是，由于目前 Smalltalk 只是一种置入壁龛的语言，我未看到很多项目在使用它。为了避免出现 Smalltalk 的结局，UML 必须是更为幸运的，即使它是优越的。

围绕 UML 用作编程语言的一个有趣的问题是如何为行为基理建模。UML 提供三种行为建模方式：交互图，状态图以及活动图。所有这三种方式都有支持其编程的人。如果 UML 用作编程语言真的得到普及，那么，看出上述技术中哪一种成为成功的技术，将是很有意义的。

人们看待 UML 的另一方式就是用于概念建模和用于软件建模之间的幅度。很多人都熟悉 UML 用于软件建模。在这种**软件视面**(software perspective)中，UML 成分相当直接地映射到软件系统中的成分。正如我们将要看到的，映射绝不是规范的，但当我们使用 UML，便要谈到软件成分。

对于**概念视面**(conceptual perspective)，UML 表示研究领域中概念的表述。这里，我们并不谈到软件成分，甚至在构作谈论特定领域词汇时也是如此。

关于视面并无严格规则；它一出现，就有多种用法。有些工具将源代码自动转换成 UML 图，将 UML 图看作源代码的另一种外貌。如果你利用 UML 图去努力理解一群会计师的资产池(asset pool)术语的各种含义，那你便更多地进入思维的概念架构中。

在本书的前几版中,我将软件视面分成规约(接口)视面与实现视面。实践中我发现很难在这二者之间画出一道严格界线。因此,感到不值得如此小题大做,进行区分。不过,我总是在图中倾向于强调接口,而不强调实现。

UML 的这些不同的使用方式引起了很多关于"UML 图的含义为何以及 UML 和外界的关系为何"的争议。特别是,它影响到 UML 和源代码之间的关系。有些人主张 UML 用于创建和用于实现的编程语言无关的设计,另一些人则相信,与语言无关的设计是牛一般的蠢事。

观点上的另一区别是什么是 UML 的精髓。依我看,UML 的多数用户,特别是草图绘制人员,把图看作是 UML 的精髓。然而,UML 的创建者们却把图看成次要的,他们认为元模型是 UML 的精髓,图只是元模型的表现,这一看法对蓝图绘制人员以及 UML 的编程用户来说,还是有意义的。

因此,当你阅读任何涉及 UML 的读物时,重要的是要了解作者的观点。只有如此,才能认识到 UML 激起的(往往是激烈的)争论的意义。

说到这一步,我需要澄清我的偏见。几乎所有时间,我都将 UML 用于草图绘制。我发现 UML 草图对正向工程和逆向工程都是有用的,并且在概念视面和软件视面中也都有用。

我不是一名正向工程师的详细蓝图的热烈爱好者,我相信,详细蓝图很难做好,而且它会使全部开发工作放慢。对某一级子系统接口绘制蓝图是合适的,但即使那时,当开发人员实现跨接口交互时,你也该指望改变那些接口。逆向工程师蓝图的价值与工具如何工作有关。如果它是用作动态浏览器,则可能很有帮

助；如果它生成大量文档，它所做的一切就是在砍树。

对于将 UML 用作编程语言，我认为这是一种好想法；但怀疑这样做的意义。我并不确信，对大多数编程任务而言，图示形式要比文字形式更富有成效；并且即使如此，对一种语言来说，也很难被广泛接受。

由于我的偏见，本书过多地着重于 UML 用于草图绘制。幸运的是，这对一本简明入门读物来说，却是有意义的。在这样小篇幅的书中，我不能平等对待 UML 的其他使用方式，却是其他可平等对待 UML 三种使用方式的书的一本好的引导读物。建议你将本书看作一本引导读物，并在需要时转去阅读其他书籍。如果只对草图感兴趣，本书很可能就是你的全部需要。

UML 发展简史

我承认自己是一名历史爱好者。我钟爱的轻松读物就是好的历史书。当然，我也知道，这并非所有人的娱乐方式。我在这里谈历史的原因是，我认为，在很多场合，如果不了解 UML 如何发展到这一步，也就难以理解 UML 现今居于何处。

20 世纪 80 年代，对象概念开始离开实验室，朝向“现实”世界迈出最初的步伐。Smalltalk 趋于稳定，成为人们可用的平台，并且诞生了 C++。那时，许多人都开始思考面向对象的图示设计语言。

面向对象图示建模语言的重要书籍出现于 1988 年到 1992 年间。领衔人物包括 Grady Booch[Booch，OOAD]，Peter Coad

[Coad, OOA], [Coad, OOD]; Ivar Jacobson (Objectory) [Jacobson, OOSE]; Jim Odell [Odell]; Jim Rumbaugh (OMT) [Rumbaugh, insights], [Rumbaugh, OMT]; Sally Shlaer and Steve Mellor [Shlaer and Mellor, data], [Shlaer and Mellor, states]; 以及 Rebecca Wirfs-Brock (Responsibility Driven Design) [Wirfs-Brock]。

这些作者中的每一位当时都非正式地领导着一个喜爱各自想法的实践人员小组。所有这些方法都很类似,但也含有不少往往令人讨厌的细微差异。相同的基本概念以不同的表示出现,从而引起客户混淆。

在那一段令人兴奋的时期,有人谈到标准化问题,也有人忽视。OMG 的一个小组试图考察标准化,但得到的只是来自所有方法学学者的一封公开抗议信。(这使我回想起一个古老的玩笑。问题:一名方法学学者与一名恐怖主义者的区别何在?答案是,你可以和恐怖主义者协商。)

最初促成 UML 的重大事件是 Jim Rumbaugh 离开通用电气公司,与 Rational 公司(现在是 IBM 的一部分)的 Grady Booch 联合。Booch/Rumbaugh 联盟一开始就被看成是可以占有大部分市场的联盟。Grady 和 Jim 宣称:"方法战已告结束——我们是赢家。"这基本上表明,他们要去实现 Microsoft 方式的标准化。一些别的方法学学者建议组织一个反 Booch 的联盟。

到 OOPSLA'95,Grady 和 Jim 已经准备好一份关于他们合并方法的公开表述:统一方法(Unified Method)文档 0.8 版本。更为重要的是,他们宣布,Rational 软件公司已经收购了 Objectory 公司,因而 Ivar Jacobson 要加入统一小组。Rational