

我眼中的 Visual Studio 2010 架构工具

张逸

www.agiledon.com

影响架构质量的是构建体系架构的思想、原则、实践与架构师的经验，绝不是工具。即使是最优秀的架构工具，也不可能像倚天宝剑一般——倚天一出，谁与争锋——似乎谁握住了这把利刃，就能够成为武林盟主。架构工具可以改善架构师的工作，却不能替换架构的过程。软件开发过程中，最重要的依旧是人。

我在尝鲜 Visual Studio 2010 架构工具¹时，偶然看到一篇文章，用夸张的语言吹捧 VS 2010 架构工具，认为它是架构师最怕程序员知道的新工具。这让我有感而发，我想起数十年前甚嚣尘上的一个理论，那就是 CASE 工具在未来可以取代编码工作的论断。随着 DSL 的逐渐流行，这个论断似乎有了能够实现希望。我们已经看到了很多优秀的代码生成工具，通过建模，可以花很少的时间完成编码实现。然而，现实是 CASE 工具至今仍然无法完全取代编码工作；而若要完全取代架构与设计的工作，则近乎不可能，因为工具不可能取代人类的思想和经验。我们可以不断地丰富最佳实践，在知识库中总结出架构的模式，利用工具来展现这些规律与原则¹¹。这意味着我们可以从变化中寻找不变，利用共性分析提高复用的可能（包括架构的复用），但变化始终存在，针对的问题域会因项目而异，即使是抽象，它也不可能无所不包，代表一切具体的实现。

“工欲善其事，必先利其器”。诚然，工具对我们的帮助不可低估，否则，就是拒绝革新的保守思想；然而，盲目夸大工具的效力，忽视人的作用，带来的负面影响并不亚于故步自封对前进过程的阻挠。用正确的态度对待工具，工具才能为我所用，这是我一贯坚持的态度。敏捷宣言认为：个体与交付重于过程和工具。客户满意才是硬道理，架构与设计所使用的工具，与客户满意度没有任何关系。所以，我们评价工具，是要看它能否提高我们的工作效率，改善我们的工作质量。那么，VS 2010 架构工具能够做到这一点吗？

我们需要先思考一下，作为架构师，最希望看到的架构工具是什么？我认为，理想的工具应该具备如下特点：

- 1、易用性：可以非常容易和快速地构建与设计模型；
- 2、可验证性：构建的模型是可以验证的；
- 3、标准化：利于设计人员和开发人员的沟通；

- 4、工程化：支持正向工程与逆向工程；
- 5、可文档化：能够较好的支持文档化；
- 6、可度量性：有助于对架构模型进行分析与度量；
- 7、模板化：支持通用的架构标准与原则，便于快速生成模型；
- 8、方法学支持：支持通用的软件方法学（尤其是架构方法学）；
- 9、可集成性：能够结合在软件开发生命周期过程中；

首先来看易用性。构建软件系统的架构，一部分工作是与图形在作战。无论是物理架构还是逻辑架构，用图形表达整个系统错综复杂的关系，最为直观。我希望在绘制与架构相关的模型图时，并不会因为绘图的不便对我的设计思路产生影响与阻挠，不会因为工具的无法表达或难以表达而影响我的工作质量，更不会因为构建出来的架构模型图被设计人员与开发人员所误解。这意味着模型图的绘制要尽可能简单与快捷，图形的表达能力尽可能丰富，同时还要符合通用的标准，不会因为一套全新的标记而产生沟通上的分歧。以对 UML 的支持为例，我们在构建架构的过程中，常用的 UML 模型涉及到包图、组件图、部署图、活动图以及用例图。vs 2010 克服了之前版本对 UML 支持不够的弱点，充实了 UML 建模的能力，提供了包括组件图、类图、活动图、时序图、用例图五种 UML 模型。例如，我们可以绘制如下的组件图：

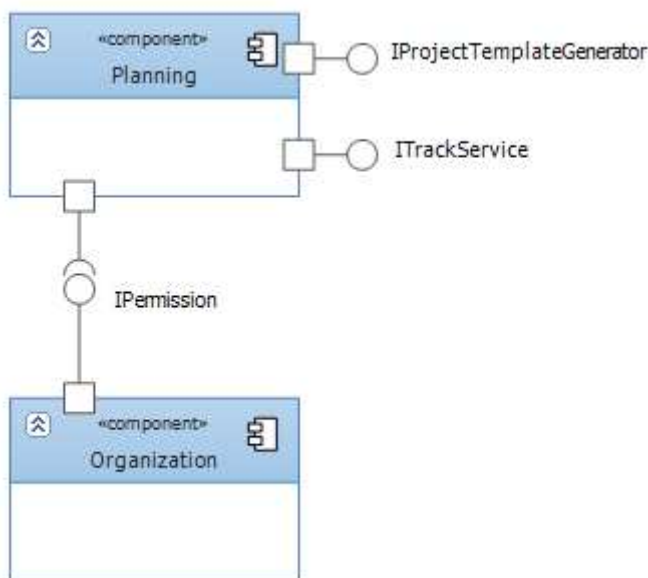


图 1：组件图示例

又或对时序图的支持：

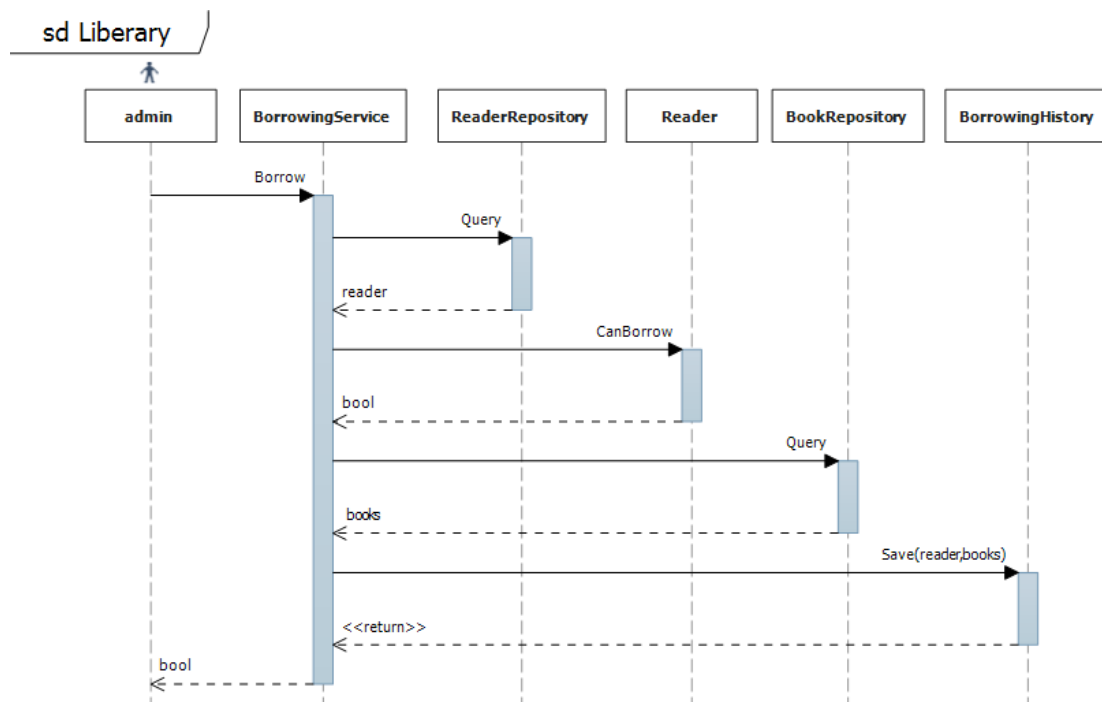


图 2：时序图示例

整体来看，它对 UML 模型的支持差强人意，虽然可以绘制出异常漂亮的 UML 图，但操作并不方便，无法提高建模的效率。例如：在组件图中无法轻易地绘制接口对组件的实现，对提供的接口与要求的接口之间的对应关系也较复杂；在时序图中，Actor 的图形既不符合 UML 通用标识，使用也不方便——它没有将 Actor 作为一级公民，而是作为 Lifeline 的一项属性提供。

作为架构师，如果希望完成 UML 建模，我不会选择 VS 2010，因为它没有提供包图ⁱⁱⁱ和部署图，不过它所提供的层模型却值得尝试。如果我们需要对大型生态系统的建模，或者绘制网络拓扑图，VS 2010 更加难以胜任。从这一点来看，VS 2010 若想成为架构师的首选，还有很长的路要走。

根据我对 VS 2010 的分析，我认为，微软的野心并没有这么大，它并没有期待 VS 2010 提供的架构工具完全涵盖架构与设计的各个领域。它的杰出表现，尤其对于 UML 建模而言，还是在于双向工程（主要是逆向工程）的完美支持。

事实上，微软对 UML 双向工程的支持可以追溯到 1997 年与 Rational 合作开发的 Visual Modeler。在 Rational 被 IBM 收购之后，这一工具就销声匿迹了。从 VS 2005 开始，提供了对逆向工程的支持，但这种支持非常有限，仅限于对类图的支持。VS 2010 完善了对 UML 主要模型的支持，因此双向工程更具有普遍意义。VS 2010 对正向工程的支持并不够，它需要根据 T4 模板来创建，并充分利用了 Visual Studio 的扩展功能。我不明白微

软为何不直接在菜单项中提供对正向工程的支持，因为它本身可以非常完美地做到^{iv}。

VS 2010 对逆向工程的支持极为强大，除了支持之前版本已经提供的类图外，还支持生成依赖图、时序图以及层模型。依赖图可以帮助我们了解程序的结构以及类之间的关系，时序图则有助于理解对象之间协作的方式，至于层模型则在更高的层面上帮助我们了解分层架构。

VS 2010 可以按照程序集、命名空间、类等建立依赖图。以 .NET 提供的示例程序 StockTrader 为例，我希望了解服务层中相关类之间的依赖关系，就可以通过 Architecture 菜单的菜单项“Generate Dependency Graph”。我按照自定义的方式生成依赖图，如图 3 所示：

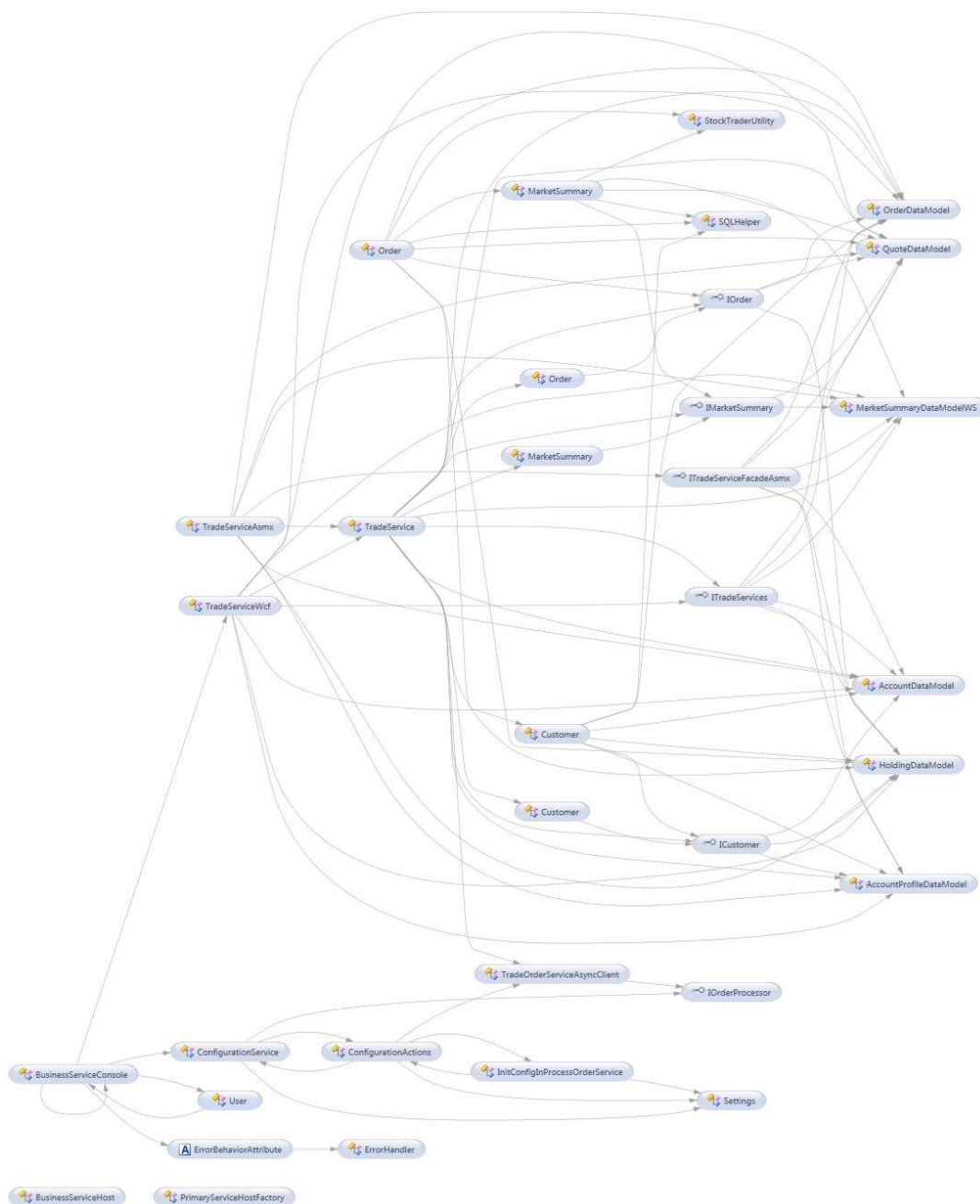


图 3 根据自定义方式（公开的类）生成的依赖图

图 4 为该依赖图的局部：

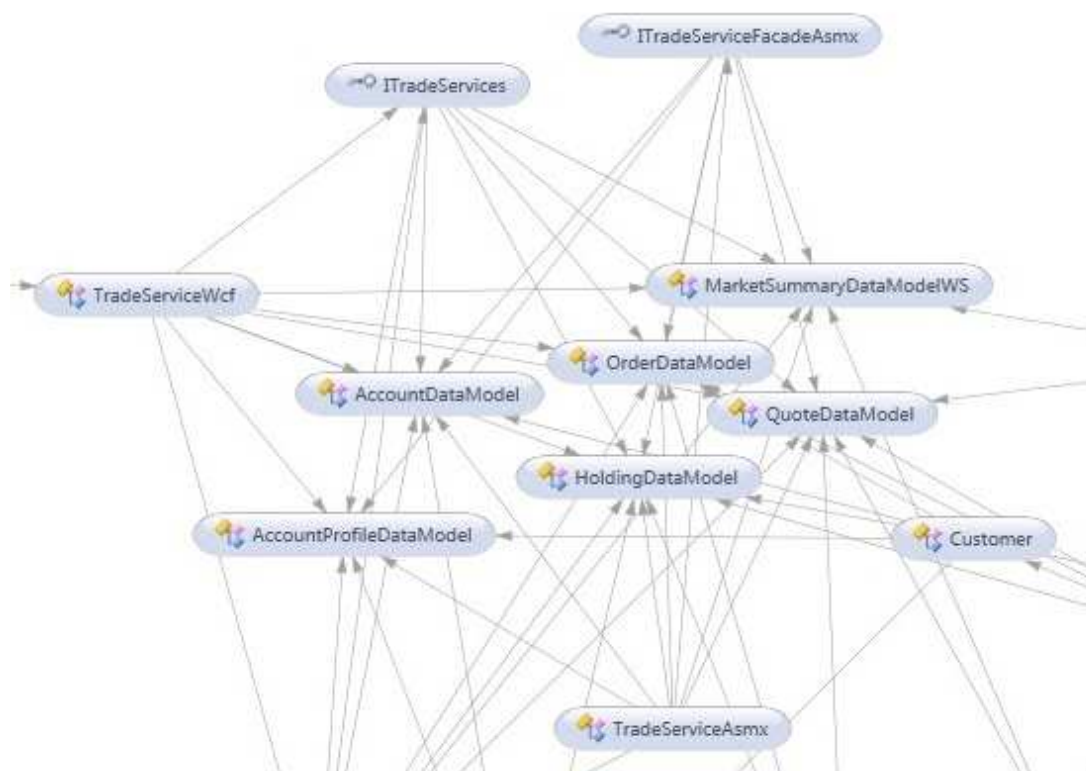


图 4：依赖图的局部

我们可以看到 Model 对象被依赖的强度是最高的，其中 ITradeServices 接口几乎依赖所有的模型对象，这是因为该接口是服务层的主要服务，相当于外观服务，负责协调和操作订单、报价、账户信息等模型对象的消息处理。

就我的感受来看，这样的依赖图显得有些华而不实，因为这样一张蜘蛛网般的图形，实在让人有些茫然，我们可能会在一张超级大型的依赖图中迷路。不过，如果希望对依赖关系来一次鸟瞰，或者需要初步了解各个对象的依赖强度，该依赖图还是有一定的参考价值。此外，倘若系统规模不够大，则可以选择类型级的依赖图；如果系统规模太大，则可以根据程序集或命名空间生成依赖图，这可以在一定程度减小依赖图的复杂程度。

时序图的逆向工程实在太精彩了。静态的类图虽然有助于我们了解类的结构，但类之间的协作却根本无法跟踪。我们在做设计的时候，常常会借助时序图来帮助我们了解某个功能项的执行过程，追踪它的消息传递方式，清晰把握类的协作方式，并以此为基础寻找到类的行为，以及不存在于真实世界的类对象。在阅读并理解代码时，如果能够有时序图的帮助，会更容易理清设计的思路，明白设计的目的。例如，同样在 StockTrader 项目下，我需要了解

服务层中 TradeService 类的 login() 方法实现, 就可以为其生成一个时序图。在 vs 2010 中, 生成时序图只需要在方法上点击右键, 选择“Generate Sequence Diagram...”项即可。图 5 是为 login() 方法生成的时序图, 我们可以清晰地看到 TradeService 与 ICustomer 和 ConfigUtility 之间的交互情况。

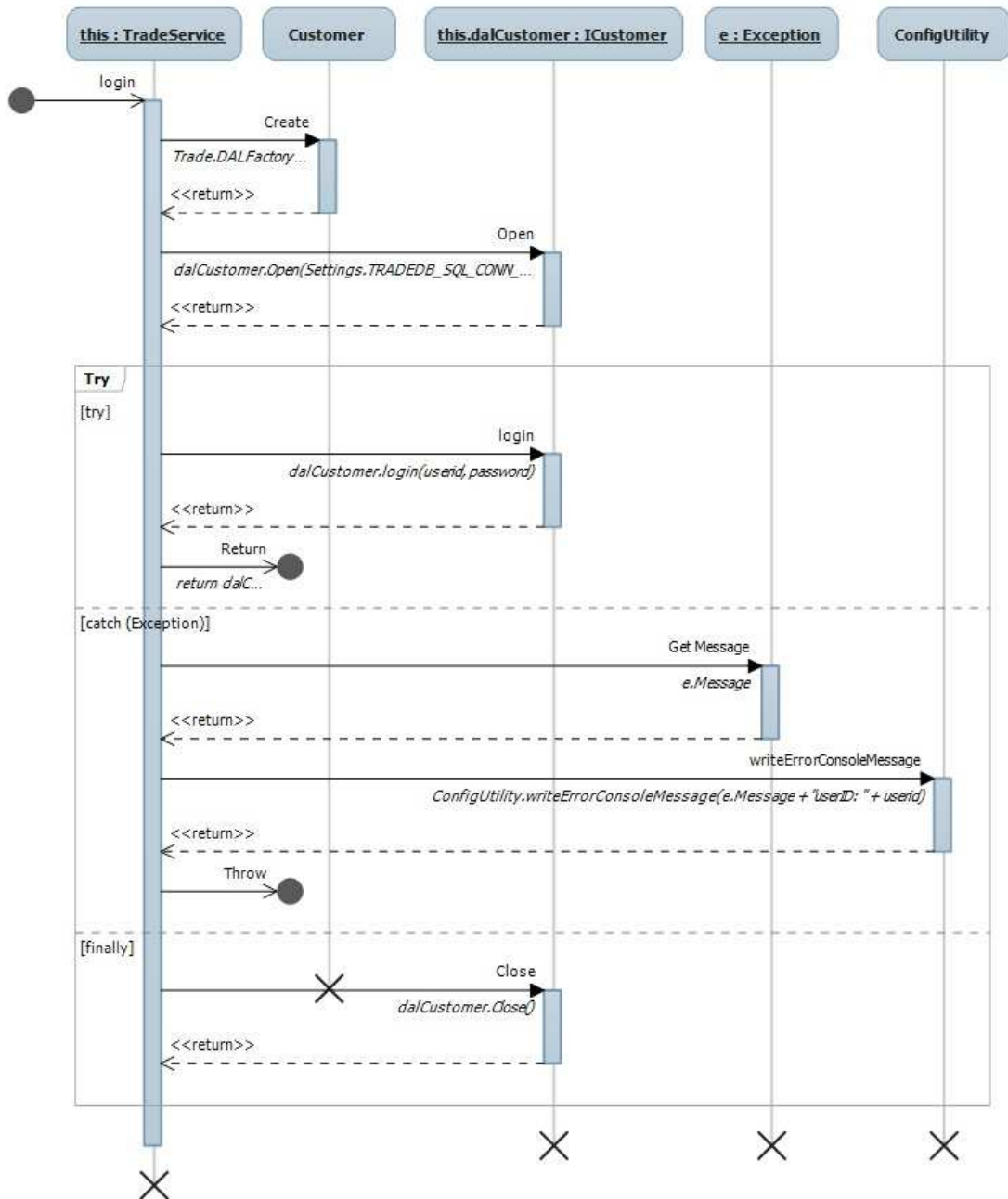


图 5: 为 login() 方法生成的时序图

层模型对架构师的帮助更大。vs 2010 可以根据现有的解决方案生成层模型。在打开现有解决方案后, 添加一个新的 Modeling Project, 并创建一个 Layer Diagram, 然后将解决方案的相关程序集拖拽到 Layer Diagram 的设计器中。通过执行快捷菜单中的“Generate Dependencies”命令, 可以检查并获得各个程序集之间的依赖关系, 并以图

形方式展现出来，如图 6 所示。

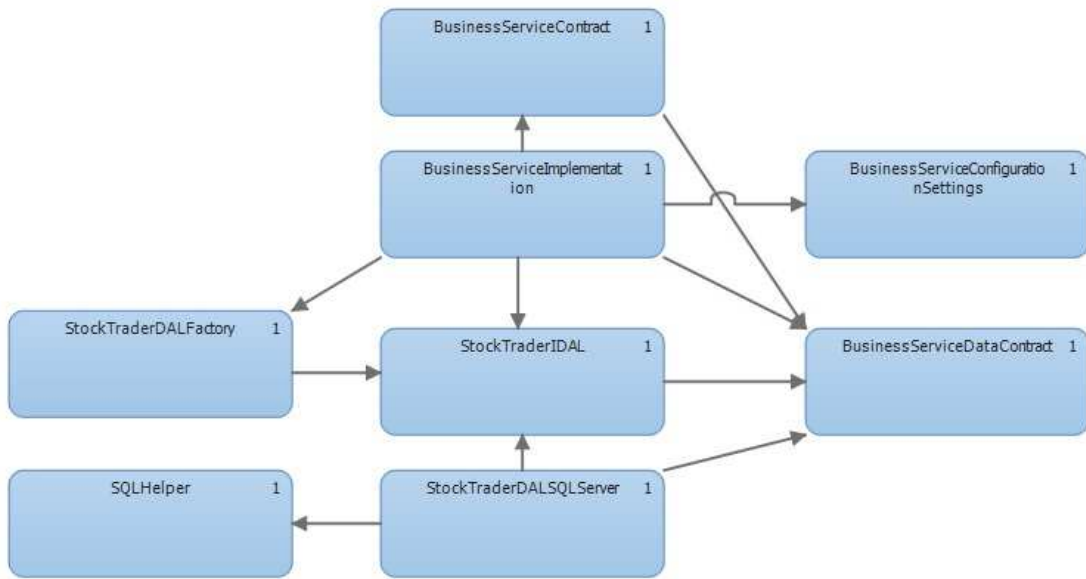


图 6：为服务层生成的层模型

通过图6，我们可以看到BusinessServiceImplementation依赖了 StockTraderDALFactory、StockTraderIDAL、BusinessServiceDataContract 以及BusinessServiceConfigurationSettings，同时它作为 BusinessServiceContract服务契约的实现，还要依赖BusinessServiceContract。很明显，BusinessServiceImplementation与具体的数据访问层实现 StockTraderDALSQLServer之间没有任何依赖关系，这就意味着服务层与数据访问层的具体实现是解耦的，这符合一般的架构原则。利用Layer Diagram，我们就可以很好地了解各个模块之间的依赖关系，帮助我们分析架构的合理性，是否存在双向依赖、循环依赖，或者模块设计是否很好地遵循高内聚、松耦合原则。

VS 2010提供的“Validate Architecture”菜单项还可以对架构进行验证。我们可以直接对上面生成的Layer Diagram执行验证。但这样的验证并没有价值，因为验证的规则就是通过现有实现生成的。微软推荐的做法是架构师根据对层与模块的理解，绘制一份符合架构原则的Layer Diagram，然后将已经实现的程序集拖拽到相应层上，再执行验证。如果实现违背了Layer Diagram要求的原则，就会提示错误^v。这样的验证功能可以帮助架构师快速地验证团队的开发人员在实现过程中是否遵循了自己的设计。

从上述分析可以看出，vs 2010架构工具充分利用了它与IDE集成的优势，为设计人员与开发人员提供了便利的工具，完成模型的转换与输出。这既有利于设计人员对架构的验证，帮

助维护人员理清程序结构之间的关系,通过对依赖关系的度量检验模块和类之间的耦合关系,更有利于团队在项目后期生成设计文档。可以说,VS 2010在可验证性、标准化、工程化、可度量性方面都有闪光之处。遗憾的是,VS 2010似乎并没有提供自动将这些模型图转换到Word的功能^{vi},这不能不说是一种遗憾。

VS 2010似乎并没有足够的功能支持我们快速生成架构,例如经典的三层架构、管道-过滤器或MVC架构。这也正是我想表达的工具不能替代架构师的最大问题。即使可以构造一些模板,就像提供项目计划模板一般,支持这些经典架构的快速生成,但始终无法替代架构师对领域知识以及质量属性的分析与设计。

从方法学支持的角度来看,VS 2010仍然支持不够。我很喜欢Enterprise Architect对ICONIX方法的支持方式,在VS 2010中没有看到对相关方法学的支持^{vii}。即使是对UML的表达,VS 2010都显得过于简单(支持模型少)与复杂(操作不方便),虽然它的图形真的非常炫。当我需要构建一个架构时,VS 2010绝对不会是我的首选;但当我需要为架构的实现进行验证或者提供设计文档时,只要我工作在.NET平台下,我绝对会毫不犹豫地选择它,它真的是一件具有超强战斗力的利器。

现在的Visual Studio 2010已经不是单纯的IDE这么简单,它是一个全方位作战的快速工作平台,通过它可以完成设计^{viii}、开发、测试、重构以及团队的管理与协作。这种涵盖软件开发生命周期各个阶段的综合工具^{ix},是开发人员和管理者梦寐以求,因为我们不用再考虑各种不同工具之间的集成与部署,何况VS 2010还提供了非常强大的扩展功能,使得我们能够因项目或技术而异,实现自己的定制。不过,这种大一统的强权模式,很容易限制开发人员的自由与创新,抹杀人的个性。“知其然而不知其所以然”,开发人员慢慢会产生一种惰性。由工具来完成许多繁杂的工作,固然可以提高工作效率,却失去了探究背后原理的机会,正如当年的ASP.NET,造就了一帮不明HTTP工作机制的程序员一般。这就需要我们进行取舍,回到开篇的话题上,那就是我们必须明确自己对工具的态度,让工具为我所用,却不会被其所制。

ⁱ 只在 Visual Studio 2010 Ultimate 版本中提供。

ⁱⁱ 就好似重构工具对重构原则的支持。

ⁱⁱⁱ 在类图中提供了 Package,但没有专门的包图功能强大。

^{iv} 正向工程的做法请参见 MSDN 文档:

<http://msdn.microsoft.com/en-us/library/ee329480.aspx#What>

^v 具体做法参见微软的官方博客

<http://blogs.technet.com/b/teamarchchina/archive/2009/08/31/vsts-2010.aspx>

^{vi} 只能将模型图粘贴到 Word,而不提供直接导出 Word 文档功能

^{vii} 当然,VS 2010 支持微软解决方案框架,即 MSF,可以实现概念设计、逻辑设计与物理设

计，但就现有的 VS 架构功能来看，对这些设计的支持显然不够。

^{viii} 对 Modeling Project 还支持版本控制功能。

^{ix} VS 2010 加大了对敏捷的支持，并将 Scrum 作为基本的敏捷开发模型。