



白皮书

使用 **UML2.0** 来解决系统工程问题

Ian Barnard

产品市场经理, Telelogic

摘要

为了交付日益复杂的系统，系统工程师正面临着越来越大的压力。系统工程师需要更好的技术来用于分析复杂的问题并用于描述解决这些问题的复杂系统。

本文讨论了UML2.0怎样被用于处理系统工程师所面临的一些问题。我们将利用UML2.0的例子来探讨描述系统构架、接口与行为的技术。

目录

1	引论	3
1.1	什么是系统分析与结构化设计?.....	4
1.2	系统工程 —— 问题是什么?.....	5
2	为什么选择 UML 2.0 来可视化信息?.....	6
3	UML 2.0 是怎样帮助创建有效的解决方案的?.....	7
3.1	捕获与可视化需求	8
3.2	描述构架与接口	8
4	UML 2.0 是怎样帮助管理技术复杂性的?.....	15
4.1	通过分解来开发构架	15
4.2	在 UML 2.0 中描述行为.....	17
4.3	但是设计可行吗?.....	19
5	UML 2.0 还能提供怎样的帮助?.....	21
5.1	部件与重用	21
5.2	UML 2.0 的测试大纲.....	21
5.3	软件开发者的通用语言	21
6	结论	22
	参考文献	23

关于 UML2.0 UML (Unified modeling language) 是一种用于描述、构造及书写软件系统的可视化语言。于 2002 年底推出的 UML2.0 将是被改进的下一代 UML 语言。UML1.0 基本上是为分析和模型化小规模软件系统而设计的，而 UML2.0 则被改进为更加适合软件开发行业的新挑战，对基于组件的开发提供了更好的扩展性，为构架建模和动态行为的描述提供了更强的支持。UML 是 Object Management Group (OMG) 组织的标准。在此资料中，凡提及“UML2.0”的地方即指 U2 Partner 组织最近提交给 OMG 的提议，并被 Telelogic Tau 所实现。

“系统工程是关于为问题创建有效的解决方案并管理开发结果的技术复杂性的工程。”¹

1 引论

今天的系统工程师正面临着越来越大的压力来及时且在预算内交付更大、更复杂的系统；为了获得成功的机会，工作必须第一次就正确。

这种压力不会减轻。任何能够通过更好的技术与自动化工具来提供工作效率的机会都应该被抓住与最大化地利用。

当问题与解决方案变得越来越复杂时，对创作者来说越来越难以描述它们，并且对评审人员与其他用户来说也越来越难以理解这种描述。真正需要的是更加强大的能够描述问题与解决方案的方式，它能够提供抽象的层次，允许设计人员把注意力放在设计上而不是过多地担心实施的细节。

当系统描述变得越来越复杂时，发现描述的错误也变得越来越难。这里真正需要的是能够增加形式化与精确性的方法，以使描述变得无歧义并能够使用工具来自动检查错误，从而消除需要昂贵的人工来完成的这种单调乏味的工作。这解放了那些他们的能力在系统的创新开发上的人。

每个项目都在不同方面使用图形来帮助开发。你曾经见过多少次类似图 1 所示的白板上的图形，每次都不可避免地写着：不要擦除？如果图形对于帮助理解与分析问题是如此重要，那么以更正式的方式并作为开发方法的一部分来使用它们会成为改进图形描述问题与解决方案的一种方法，这也许能够最终替代白板。²

由 OMG 组织所标准化的 UML 1.0 是一种图形化的语言,它作为可视化描述软件的方法已经取得了成功。但是，因为它不能深入到开发的更深阶段并且不能满足系统工程师所有的需求而很少被他们所采用。

UML 也在不断地被改进。UML 2.0 的开发肩负着建立一种不仅

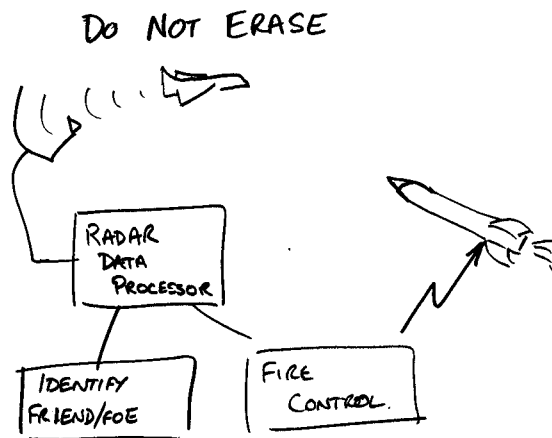


图 1: 导弹火控系统的第一张构架图

适合于软件开发，也包括所有系统工程师在内的更为广泛适用的语言。

通过对系统工程师会问到的 UML 2.0 的四个问题的回答，本文详细描述了 UML 2.0 能为系统工程师所带来的好处。

1.1 什么是系统分析与结构化设计？

许多年来，系统工程师使用系统分析与结构化设计技术获得了相对的成功。如果你熟悉像 Yourdon, Harel, Jackson 等等这样的符号，那么本文所讨论的一些概念就会变得不那么陌生了。

他们中的那些成功的方法已经被 UML 2.0 的符号所采纳。需要记住的是，结构化分析与结构化设计是从两个不同的角度来处理问题的。

结构化分析开始于需求并采用数据流程图的方式来可视化地描述需求，然后把它们分解到我们能够完全理解的程度。结构化的设计着眼点在于我们需要采用一种好的方式来描述最终的软件方案，以使我们能够理解并最终实施（例如一程序结构图）。

两种技术都是可行的。但是却使用了两种不同的符号。同时也缺乏把数据流程图与状态图转化成程序结构图的一种确定的方式或自动转化功能。我们只能凭经验来一一对应，但还是有一条的鸿沟无法被自动化所连接。

我们可以看到 UML 2.0 可以适用于系统分析与软件设计方面，但是使用的是一种通用的符号。不同符号的问题与两者之间的鸿沟被消除了。

1.2 系统工程 – 问题是什么？

在几乎所有的系统工程方面，正式的文档都是采用文本描述的。用户需求、接口描述与系统设计文档就是这方面很好的例子。书面语言如果使用得当就可以灵活与相当准确地表达。在很多情况下，在正式文本的辅助下，图形能够使信息更容易传递与理解。在系统工程中，快速且准确地理解系统需求是非常重要的。当然，画非正规的图形很容易。***为什么选择 UML 2.0 来可视化信息？***

项目越来越多地由分布式的团队来开发，这种物理、逻辑与合同上的分割意味着参与者需要更好的方式来沟通所有相关的细节信息，并管理由于信息变更所带来的影响。***UML 2.0 是怎样帮助创建有效的解决方案的？***

当系统变得越来越复杂时，参与者需要更好的工具来描述复杂的问题与解决方案，不仅适用于小项目也能够扩展到大项目。***UML 2.0 是怎样帮助管理技术复杂性的？***

UML 2.0 在这些方面对系统工程师有很大的价值，但是从其它方面我们还能从 UML 2.0 获得那些好处呢？***UML 2.0 还能为系统工程师提供怎样的帮助？***

2 为什么选择 UML 2.0 来可视化信息？

在发明书写之前，图形被用来传递信息已经有成千上万年的历史了。创建图形不一定是一种简单的过程——这是模型工具可以帮助的一个方面——但是图形理解起来很容易，这是因为它是一种基本的、直观的信息沟通方式。

可以用很多种类的图形来代表信息。但是，如果每个团队、项目或公司选择他们自己的图形符号，总体的效果就会使事情变得更糟。用不同的符号来沟通信息的困难是显而易见的。

真正所需的是单一的、标准化的表达图形信息的方式，这就是：

- 在语义与语法上精确，使每个人都无歧义地理解图形所表达的意思
- 强大的表达能力，使它能够应用于更广泛的问题与甚至项目
- 抽象，这可以使开发人员与构架师能够在高层工作，使他们能够集中精力于所要解决的问题
- 在工具间可以交换，使图形（与整个模型）能够被不同的团队、项目与公司所使用和重用。

UML 1.0 不能很好地满足这些需求，这就是为什么系统工程师还没有广泛采用它的一个原因。但是在设计 UML 2.0 时就考虑了这些问题，并将满足系统工程师的许多建模要求。OMG 组织正在与 INCOSE 组织（国际系统工程协会）联合来面对 UML 2.0 怎样用于系统工程的问题。³ 同时，UML 变更建议委员会也正在努力工作来保证他们的建议能够满足系统工程师的需求。⁴

为什么选择 UML 2.0 来可视化信息？

它提供了单一的、标准化的、强大的语言来精确地描述系统设计与软件设计，它可以互换也能够被其它 UML 2.0 用户所重用。

3 UML 2.0 是怎样帮助创建有效的解决方案的？

在任何项目中所面临的两个基本问题都是沟通与变更管理。UML 2.0 可以对这两方面有所帮助。

促进正式沟通

交换正式（形式化）的描述的能力变得越来越重要。今天的项目往往是在不同的团队、公司与国家间分布开发的，这不可避免地对沟通的质量造成压力。同时系统越来越复杂也需要在更多的部门间进行沟通。由于需要更多地沟通，就更加需要精确地描述——由于误解而产生的错误所造成的影响，对有合同关系的双方来说特别重要。UML 2.0 标准以两种方式来解决这个问题：图形本身有精确的含义，消除了歧义性；同时，UML 2.0 将包含一个形式化的交换描述（使用 XML），以使接口描述或整个模型能够实现电子化的交换，消除了正式沟通可能出现的错误。

促进非正式沟通

由于 UML 2.0 语言的可视化特性，这可以为沟通想法带来直接的利益——它很容易理解。由于也存在其它的可视化符号，因此有人会争论这种利益不是 UML 所独有的。UML 2.0 所特有的是表达信息的广度与描述的精确性。

管理变更

各种原因都可能引起变更：技术、竞争、客户需求、错误、降低成本、缺乏时间，等等。变更是不可避免的，唯一可行的方式是包容变更并在它发生时接受它。重点应该是管理变更并使变更的影响变得最小。在 UML 中的可视化模型是精确而容易理解的，同时也容易进行变更而不会引入新的错误。

团队与公司间的沟通不可避免地引入系统工程的这两方面问题，我们通过如何处理这两方面的问题来展示 UML 2.0 能够以精确与容易被理解的方式来描述信息的一些方法：

- 捕获与分析需求，
- 描述构架与接口。

3.1 捕获与可视化需求

通过发现系统所必须响应的事件与所做的响应应该是什么来理解系统必须做什么，是试图理解系统需求的最基本的部分。这是用例或场景分析的基础，它所面对的是功能需求。

当开始建立功能需求文档时，需要一种方式来捕获场景的细节，但是，随着系统变得越来越复杂，场景的数量也随之而增长。有一种能够通过场景进行浏览的方法就变得很重要。信息容易维护与更新也变得越来越大。如果一些细节可以在一个地方来定义然后在需要的地方被引用，则维护描述的工作就会被大大地减少。

UML 用例分析

UML 用例分析辨识与系统交互的外部实体（UML 把它们叫做执行者 actors，用枝状图表示），并用线把它们与所参与的场景或用例联系起来。如图 2 所示；在一个完整模型中可以有許多这样的图形。

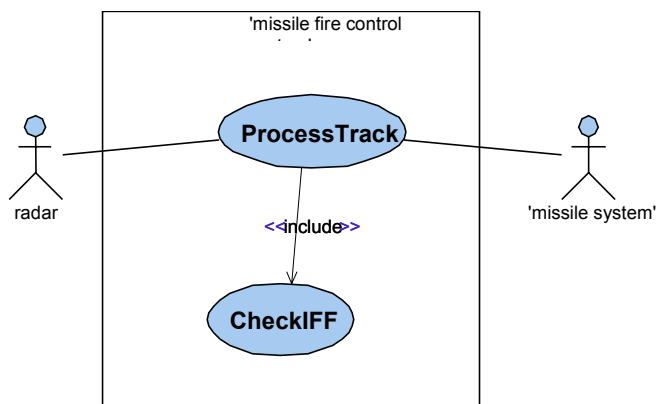


图 2: 用例图

值得注意的是执行者（actors）不一定是人；它们通常是其它系统，例如一个网络、一个数据库、一个雷达传感器等。

用例图非常简单，携带少量的形式化权重。它的存在是因为它为可视化系统功能需求提供了非常有效的方法。它把所有的使用场景与执行者放到了背景中并允许你浏览对场景的细节描述。

在图 2 中，两个执行者雷达（radar）和导弹系统（missile system）与过程跟踪（ProcessTrack）的用例交互，同时重用（包含）了另一个交互过程，判断敌友（CheckIFF）。这也可以被其它用例所包含，因此可重用的定义可以在一个地方被建立（利于维护）并在许多地方被引用（使模型紧凑）。用例的实际细节与怎样建立重用并没有在用例图中被描述。这种描述通常采用另一种 UML 图，顺序图，在其它地方来创建。UML 绘图工具能够允许把对

用例的文本描述存储于图形中也是非常重要的。⁵ 我们不是放弃文本。通过文本，我们使用可视化符号作为收集、总结、表达与浏览的一种方式。

UML 对细节行为的描述

对细节行为的表达与描述是 UML 被主要增强的第一个部分。UML 2.0 的顺序图符号现在可以用更简捷和强大的方式来描述复杂的行为，这特别适用于描述功能需求。

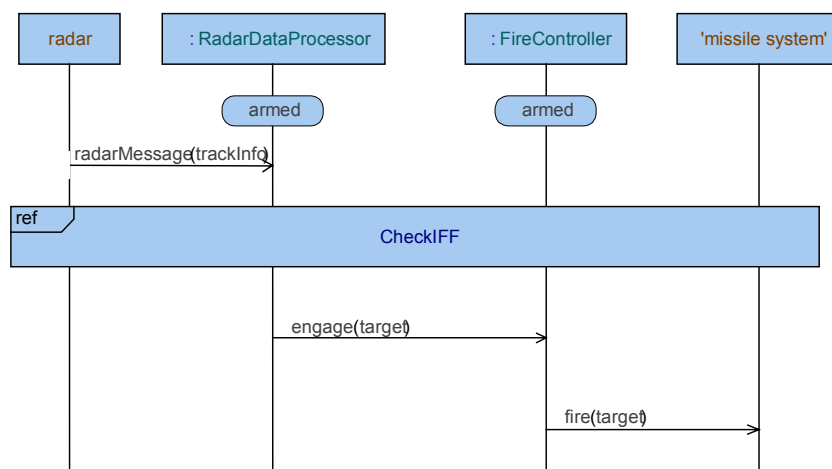


图 3: ProcessTrack 顺序图

在图 33 的例子中显示了 ProcessTrack 用例的详细交互细节。实体间交互(叫做实例)通过包括实例名称的矩形及其所附带的竖的时间线来表示。在时间线上，事件按照时间的顺序来排列，时间的箭头指向下方。事件在时间线之间用水平线来表示。事件所附的文本用于描述事件的名称与它所携带的数据，所以在一个场景中可以描述数据。

所有系统都有状态；对于本文的场景，导弹发射控制（Missile Fire Control），系统具有像 armed 或 passive 这样的状态。UML 2.0 的顺序图可以包含状态信息。在图 3 的例子中说明了在描述顺序开始时，RadarDataProcessor 与 FireController 必须处在 armed 状态。这为理解需求提供了非常重要的信息。

在用例 CheckIFF 中，通过包含一个带有另一个交互名字的 **ref** 符号，图 3 也显示了怎样创建行为的重用。当然，可以在一个图中建立多个引用，并且一个引用的顺序图可以包含更深的引用。

可选的顺序

所有系统都会有这样的情况，有时它们需要以一种方式来响应一个事件，但在另一个时候却以不同的方式来响应同一个事件，也许这是因为事件所带有的数据不同或因为系统处在不同的状态。

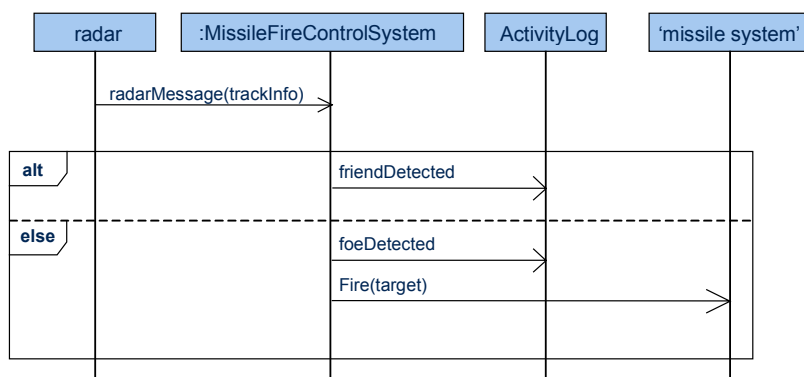


图 4: 可选的行为顺序

UML 2.0 的顺序图可以在一个图中描述这些可选的行为类型，如图 4 所示。这个 `alt` 符号可以包容任何数目的可选顺序，每个选项用点划线分开并标有 `else` 符号。可选顺序可以很复杂，包括引用其它用例或嵌套选择项。

更多 ...

顺序图也可以显示重复的、可选的与特例的顺序。时钟可以被明确地表示。使用可选项，从时钟到时所引起的各种行为结果也可以被显示——同样取消时钟所引起的结果也可以被显示。

小结 – 用例与顺序图的好处

作为系统描述的一部分，用例与顺序图有许多好处；它们使需求更容易理解与更新。

但是，好处并不仅限于这些。其中之一是 OMG 组织正在开发基于 UML 2.0 的测试大纲，它将允许把顺序图用作正式的测试描述。

使用 UML 2.0 来捕获与分析功能需求的好处是：

- 用例图提供了系统使用的高层视图。用户可以很容易地理解场景的范围与背景，他们可以容易地浏览文本细节与图形描述。
- 顺序图允许系统工程师以直观的方式可视化且准确而详细地捕获功能需求，这有助于正式的沟通与理解。
- 通过顺序图的引用来重用，使对系统的使用细节的描述变得紧凑而且容易维护。

- 顺序图可以被用作正式流程的一部分（例如定义可执行的测试）。使原来非正式的信息与开发活动变得正式，这有助于消除错误与误解。

3.2 描述构架与接口

在开发中，完整的系统不可避免地要被分成更小的部件或子系统。很多理由来这样做：

- 使系统更容易开发。解决几个小问题要比解决一个大问题来得容易。
- 有利于重用。子系统可以被其它或未来的项目所重用，减少开发完整系统所需的成本与时间并提高质量。
- 允许对开发项目进行分支。在不同区域或公司的几个小的团队可以独立地工作来开发系统。
- 可以反映系统的物理构造或系统的制造。
- 为后勤提供支持。有些可维护性的需求只能在把系统分解成子系统（有些时候叫做配置部件）后才能被满足，因为替换或维护子系统要比一个大系统容易得多。

使用子系统有许多优势，但是系统工程师需要工具来使这种优势最大化。例如，把一个大问题分解成小问题的策略只有在独立的团队能够快速完成小项目的情况下才能获得成功。为了使这一点成为可能，在描述中就不能有很多的歧义。一旦实施活动开始，就不能浪费时间来讨论子系统的边界问题。

在开发子系统时，有两种基本类型的信息需要被处理：**子系统必须做什么与它们的接口是什么？**

如果它们听起来很熟悉，这可能是因为这些问题与在项目开始时所问的系统级的问题是一样的。UML 用例图与顺序图可以被用于帮助捕获与分析子系统的需求，并回答这样的问题：**子系统必须做什么？**

虽然这些图可以建立使用例子的文档，但是它们并不定义完整的接口。例如，它们并不精确地定义流入与流出系统的消息的内容。为了提供这种正式的描述，UML 2.0 提供了部件图与接口类。

部件图在 UML 2.0 中是全新的。它们允许你把系统分解为层次化的子系统，每个都被作为黑箱来对待，这叫做部件。

定义部件 (子系统)

在 UML 2.0 模型中的所有部件都独立地执行（异步），通过部件的接口传递消息来通信。一个部件只能通过它的端口来与外界通信。

一旦定义了部件，也就定义了它的端口，图 5 显示了 RadarDataProcessor 的定义。

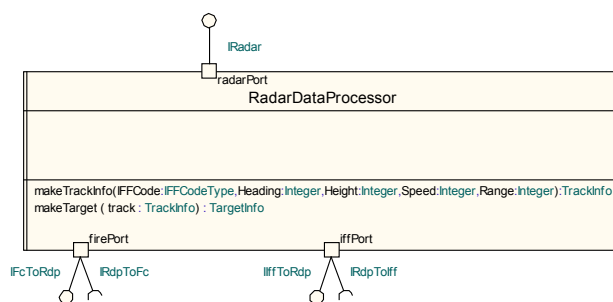


图 5: RadarDataProcessor 的部件定义

每个端口的定义可以有请求

接口（required）与提供接口（provided）。一个提供接口（表示为棒棒糖或球状符号）从外部的部件接收消息并进入它，而消息则从一个请求端口（表示为窝状）流出。如图 5 所示的例子，RadarDataProcessor 有三个端口，表示为类符号边界上的小方框。RadarPort 端口有一个 IRadar 提供接口，其它的两个端口有提供接口与请求接口。

对子系统的定义简单而清晰；端口的定义描述消息从何处流入与流出部件，而接口被定义在端口上，说明什么样的消息可以通过。

定义接口

接口的定义（用于端口的定义）可以在定义部件时创建，但是采用接口类在其它地方定义会更有用，如图 6 所示。这使得对模型的维护变得容易，因为接口的定义不仅仅会用在—个地方。通过只进行一次定义，可以在需要的地方进行引用，而且改变定义也不会引入错误。

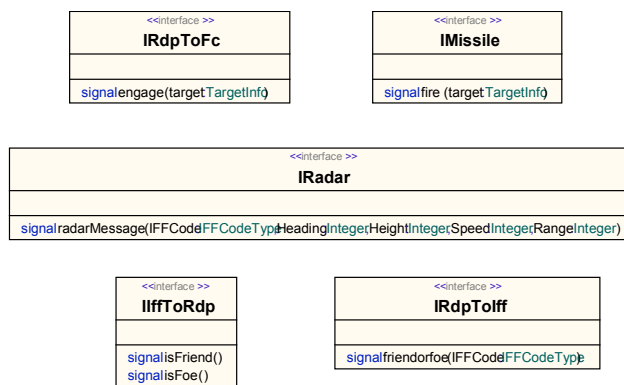


图 6: RadarDataProcessor 接口的定义

接口的定义包含消息列表（UML 2.0 把它们叫做信号），并用数据类型与名称列表来定义它们的数据内容。例如，在图 6 中，IRadar 接口定义一个单一的信号，radarMessage，并带有五个参数。

接口的定义是紧凑而准确的；它们非常清楚地描述了接口的消息与那些消息的数据内容。

定义数据类型

在 UML 模型中，用于消息参数的数据类型也被完整地定义了。例如，图 6 中所定义的信号 radarMessage，图 7 显示了所定义的参数类型。在这个图中，有两个枚举类型和两个数据结构。

对消息内容的准确定义意味着开发人员可以无歧义地描述子系统与实现所需的信息。

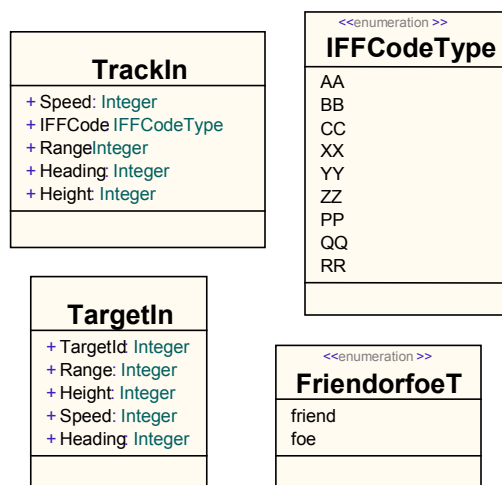


图 7: 数据类型定义

把子系统装配成系统

构架是怎样被建构的呢？一个部件图，如图 8，装配其它部件的实例，定义部件的内部结构。

在部件图中，每个实例的端口与其它实例的端口相连，并与部件边界上所生成的端口相连。UML 2.0 工具可以通过检查来确保所有提供接口/请求接口被正确地连接。使创建不匹配的接口成为不可能。

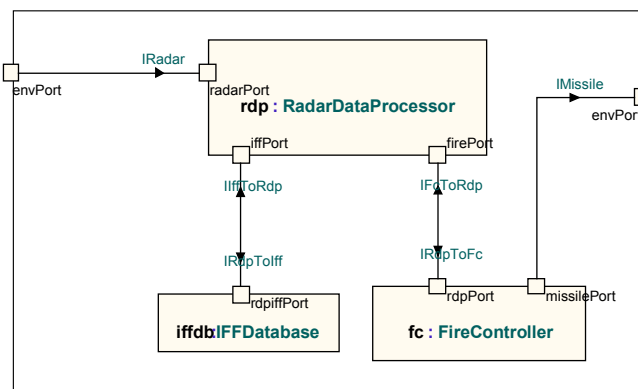


图 8: 部件图

在图 8 的例子中，三个实例通过它们的端口相互连接并与部件的边界相连。最终的图形很容易被理解与修改。

在部件图所组装的部件可以来自于其它部件实例的构造。这就是你能够以层次化的方式来描述并且把一个大而复杂的系统分解成小的，更易于管理的子系统的方式。

但是为什么要用 UML 2.0 来描述构架？

上述图形中的许多基本信息可以用常规的文本来描述，或者使用一些非正式的图形，为什么要用 UML 2.0 的部件图来描述构架呢？

标准化的语言与图形无疑可以帮助沟通与理解。但是好处还不仅仅是这些。要真正有效地使用 UML 2.0 还必须使用 UML 2.0 的设计工具来创建与管理

这些图形。现在图形中的信息被收集到了我们所要描述的系统的描述模型中。这会带来额外的好处：

- 容易创建、重用与维护设计,
- 容易浏览设计 (例如从图形到定义), 并且
- 对模型自动检查错误与一致性.

这是基于文本或非正式方法所没有的好处, 并且当系统越来越复杂时, 这种好处的价值就会逐渐增加。

UML 2.0 是怎样帮助创建有效的解决方案的?

UML 2.0 给系统工程师一种方法来可视化地定义他们想描述的信息, 如构架、接口与功能需求。而且, 这种可视化描述不仅仅是一组图形:

- UML 2.0 图形具有精确的且无歧义的含义, 使它能够很容易地被用于可视化需求与描述构架和接口。
- UML 2.0 图形很容易被理解, 有助于开发与维护。
- 构建模型的图形信息可以被自动检查。
- UML 2.0 模型与图可以用电子化的方式来交换。

4 UML 2.0 是怎样帮助管理技术复杂性的？

系统变得越来越复杂表现在两个方面：内部结构的日益复杂与行为的日益复杂。

如前所述，通过定义的接口来通信所形成的层次化的部件（子系统）是 UML 2.0 的部件图提供的描述复杂构架的有效方式。

但是，在开发解决方案时，还有两个日益复杂的方面：

- 怎样开发构架的层次？
- 怎样描述复杂的行为？

这些问题之后就是受到日益关注的：**它可行吗？**的问题。当复杂度增加时，在设计中引起的实施与最终测试错误的风险或不可预知的缺陷也会增加。如果 UML 2.0 能够帮助提高系统设计的质量并减少失败的风险，这会给系统工程师带来好处。

4.1 通过分解来开发构架

随着开发过程的进行，会针对初始构架部件的第一层或第二层进行细化，通过把这些部件分解成更小的部件，直到包含的细节可以用于实施。

开发一个完整的构架的逻辑流程是先根据部件包含的功能需求用顺序图来表达，然后从这些图中提炼出用于新的子部件的子需求。顺序图可以用来显示子部件间是怎样相互作用来满足部件需求的。

UML 2.0 直接支持这种方式。顺序图符号包含了分解的概念。

这意味着 UML 2.0 模型使用分解可以通过可视化的开发阶段来支持与加强开发流程并且允许用户在模型中浏览这些阶段。通过以这种方式来组织模型并反映出模型的开发过程，UML 2.0 解决了系统工程师所面临的一些技术复杂性问题。

构架与顺序图的分解

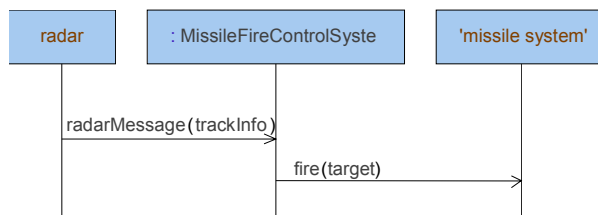


图 9: 初始的系统级顺序图

如图 9 所示,首先在分析与文档阶段开发顺序图,即系统对事件的响应行为及与执行者的交互。

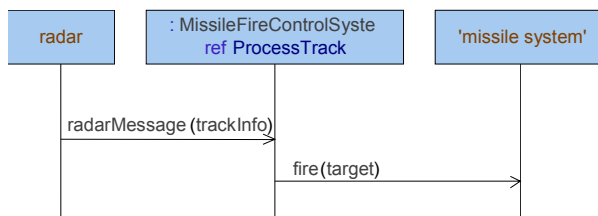


图 10: 更新引用的分解顺序图

当下一阶段的分析开始时,图形被更新,但仅仅是对将被分解的实例进行修补。例如,在图 10 中,表示系统的实例 MissileFireControlSystem 被添加了一个新的顺序图引用 ProcessTrack,它在分解时将描述 MissileFireControlSystem 的内部行为。

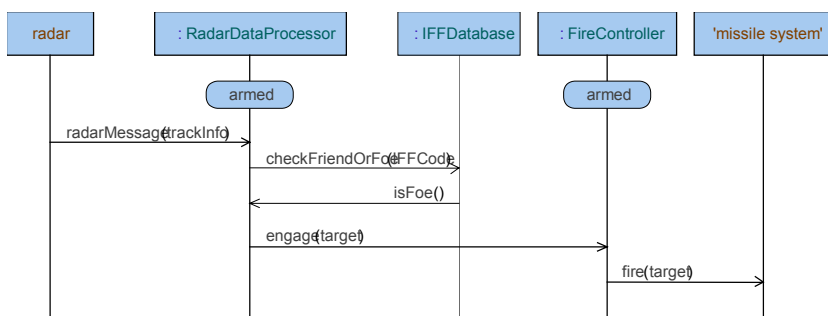


图 11: 被分解的顺序图描述的系统内部行为的细节

然后创建新的顺序图来描述被分解的实例的内部行为,描述它怎样响应接收的事件及与其它子系统和执行者的交互。图 11 是 ProcessTrack 的顺序图,有三个子系统相互交互来响应 radarMessage 事件。

4.2 在 UML 2.0 中描述行为

当开发用于系统的描述时，目的是满足需求，因此开发人员需要描述系统怎样实现在用例与顺序图中的功能需求。

在 UML 2.0 的术语中，行为是在构架的最底层（即部件不能够被再次分解）来描述的。这些部件被 UML 2.0 叫做活动类；它们彼此异步执行，并包含对输入事件的响应逻辑，使用叫做状态机的设计方式来描述。

UML 2.0 的行为描述

用状态机来开发行为描述的第一阶段是找出状态是什么以及状态间是怎样转移的。这是描述系统的非常有价值的一部分，因为这些信息包含了所有顺序图满足需求的精华信息，并且在开发实际的解决方案时可以被实施。

UML 2.0 提供了以状态为中心的状态图的概念，如图 12 与图 13 所示，它非常适合于分析的早期阶段，在这个阶段的主要问题是发现状态。在分析的后期，已经知道了大多数的状态，问题就变成了清晰描述在状态转移时会发生什么的细节。UML 2.0 的新的以状态转移为中心的状态图的概念允许对此进行全面地可视化描述，如图 14 所示。

UML 2.0 状态图概念的两个方面互相补充，并且，在完整的可视化模型中，当用于开发完整的行为描述时，它们是非常有效的，而这在 UML 1.0 中是不可能的。

UML 2.0 行为描述的开发

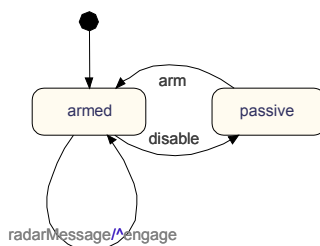


图 12: 初始状态图

图 12 显示了用于 RadarDataProcessor 子系统的一个初始 (高层) 状态图。它开始于状态 armed (用实心的圆点表示) 并还有一个叫做 passive 的状态。它们对发现目标的事件 radarMessage 做出响应并发送叫做 engage 的消息。它与在部件图 FireControl 子系统相连。

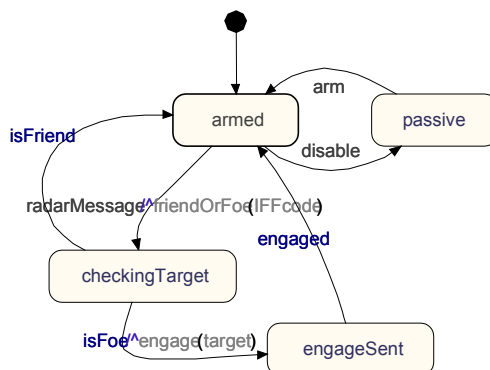


图 13: 状态图的部分细节描述

当设计继续进行时, 会在状态图中加入细节。如图 13 所示, 对事件 radarMessage 的响应被细化。它初始化了一个请求 (到 IFFDatabase) 来辨识敌人或朋友。如果响应是敌人, 就发送 engage 消息 (到 FireControl)。当响应 engaged 被收到时, 状态机回到 armed 状态。

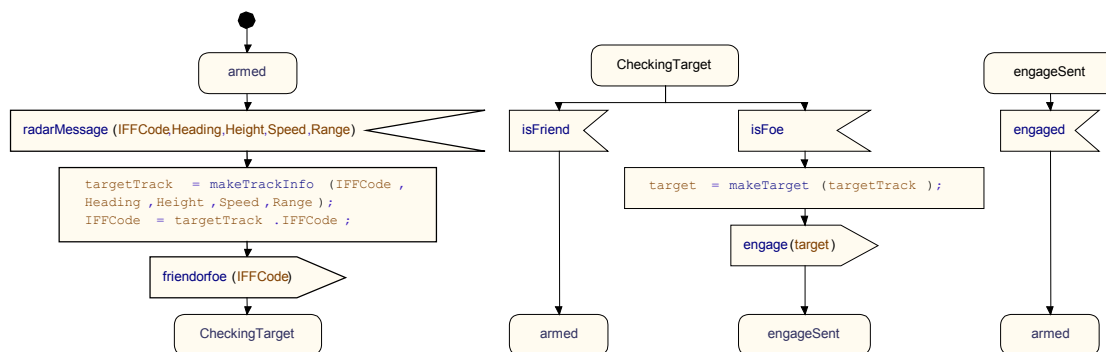


图 14: 一部分的状态图的完整细节

在最后阶段, 作用在转移的细节动作被加入到状态机中, 如图 14 所示。图 14 所示的转移与图 13 是一模一样的, 但是加入了动作的细节。现在, 完整的细节行为在 UML 2.0 模型中被完全可视化地描述出来了, 并且可以对一致性与错误进行自动检查。

4.3 但是设计可行吗？

当系统行为变得复杂时，很难确信设想的解决方案一旦被实施就会真正工作起来。你能够保证它工作的唯一方式就是在设计的早期进行试用——最好在创建它时就开始。

使用传统的技术需要进行附加的软件仿真开发，这即耗时又增加成本。通常你会根据风险的大小来仿真设计的特定方面。对于其它的方面，你会应用最佳的手工设计并对之进行评审，然后你就只能寄希望于运气了。当系统设计变化时，前期的仿真工作的意义就会大打折扣。更新仿真来反映变化并重新运行它并不容易，因为它需要额外的开发努力。除非在项目中已经计划了重新仿真，否则就是不可行的。

因此，通常在装配系统之前的最终测试阶段才能发现系统的设计问题。或者更糟的是在用户使用时被发现。为什么会这样呢？因为最终测试与实际使用往往是第一次对整个系统进行的完整试用。

通常，在最后阶段使系统能够工作的唯一方式是快速修补；已经没有时间来重新回到设计过程并恰当地解决问题了。这种匆忙实施的修补使得第一次交付的系统缺乏过渡期，这会对系统产生深远的影响，也会增加维护的成本。

对复杂与软件为主的系统的研究显示，整个生命周期成本的 50%到 80%是在交付后发生的。⁶ 降低维护成本就有机会重新分配成本在各阶段的比例。

如果有一种方式能够在实施之前就可以对设计进行实际的测试并能够消除错误，那么质量就可以被提高而维护成本就可以被降低。

UML 2.0 的仿真

在这里描述 UML 2.0 的仿真是不合适的，因为本文的重点是 UML 2.0 语言而不是工具。

(更多的信息与例子可以参见 Telelogic 的其它技术白皮书。⁷)

但是，正如你所期待的，能够对你的可视化模型的行为进行验证是 UML 2.0 的一大优势。它无需 C++ 或 Java 编程或开发测试工具。

使用 UML 2.0，设计人员可以详细描述他们的系统，并在创建设计时就可以进行测试。

在描述中的错误与不一致可以通过自动检查来消除，并且自动测试可以用来消除动态的错误。

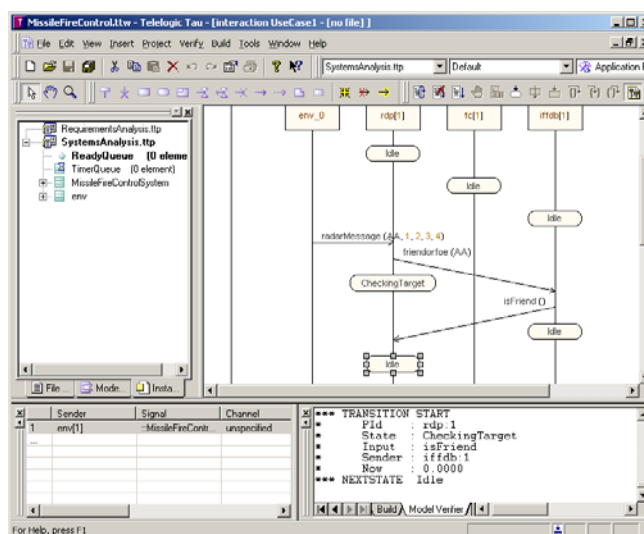


图 15: UML 2.0 仿真与跟踪

UML 2.0 是怎样帮助管理技术复杂性的？

UML 2.0 为系统工程师提供了可视化描述复杂技术信息的方法：

- 通过分解构架，顺序图分解反映了系统开发的流程
- UML 2.0 状态机允许完整地可视化描述系统行为
- UML 2.0 状态图的精确描述使它们无需软件代码就能够被仿真，所以设计能够很容易地被测试，在实施开始之前就能消除错误。

5 UML 2.0 还能提供怎样的帮助？

以上所讨论的 UML 2.0 对系统工程师的优势是使用这种语言的根本原因，但是还有其它优势。

5.1 部件与重用

如 3.2 节 *描述构架与接口* 所述，系统的构架是从部件的层次建立起来的。这些部件在一个地方被定义，在构架的层次中被引用。但是如果部件可以被独立地定义，那么它就可以被系统的其它层次不限次数地引用。

这是有可能的，因为部件图不仅仅是一个图形。连接端口的线精确地描述了信号的路径。例如，它们可以驱动自动代码生成，因此在装配部件时无需手工修改部件。自动生成的代码确实可以工作。最终的结果就是可以真正重用设计而无需进行修改。

5.2 UML 2.0 的测试大纲

OMG 组织正在开发的测试大纲（test profile）^{8,9} 将允许对 UML2.0 中的模型进行全面的测试。

这意味着，例如，测试人员将能够利用 UML 2.0 来定义测试系统的构架与定义测试。这会为测试人员带来很大的好处。但是，如果项目也用 UML 2.0 来进行系统工程与软件开发，那么就会带来更多的利益，这是因为在项目中可以重用通用的定义，从而消除许多错误的源头。

5.3 软件开发者的通用语言

如果你的系统需要软件开发，那么 UML 2.0 就提供了通用的语言。系统工程师可以使用 UML 2.0 来描述系统，而软件工程师可以用 UML 2.0 来开发实现的细节，因此 UML 2.0 模型可以被共享或重用。

这意味着两个小组间的鸿沟被弥合了，公司可以从提高了的准确性中获得利益并消除由于共享信息所造成的错误。

UML 2.0 还在哪些方面有帮助？

UML 2.0 在项目中与项目间帮助增加重用：

- 通过把 UML 2.0 的部件无需改动地重新装配成新系统，真正实现了原有设计重用。
- 软件开发人员与测试人员可以重用设计模型。

6 结论

我们已经讨论了对 UML 2.0 的四个基本问题的答案:

- ***为什么选择 UML 2.0 来可视化信息?***

它提供了单一的、标准的、强大的语言来描述系统设计与软件设计，并可以与其它 UML 2.0 用户交换这些信息。

- ***UML 2.0 是怎样帮助创建有效的解决方案的?***

它为系统工程师提供了他们所要描述信息的可视化定义方式，如构架、接口与功能需求。然而，这种可视化描述不是单纯的图形；它是精确的描述并且能够被自动检查。

- ***UML 2.0 是怎样帮助管理技术复杂性的?***

它为系统工程师提供了可视化描述复杂技术信息的方法，并能够反映开发流程，完全描述复杂的系统行为而且能够在创建设计时就进行测试。

- ***UML 2.0 还在那些方面对系统工程师有帮助?***

通过重用设计部件，它增强了项目间的可重用性，并在系统工程师与软件开发人员和测试人员之间增强了对通用语言的可重用性。

UML 2.0 语言允许系统工程师用标准化的设计符号来精确地可视化需求、构架与接口。

通过使模型容易理解与修改可以帮助项目提高沟通能力并减少变更的影响。

当系统变得复杂时，UML 2.0 能够帮助系统工程师精确描述复杂的系统，并使他们能够在项目的早期对设计进行经常的测试。

UML 2.0 代表了系统工程师一种新工作方式的发展方向，并帮助他们第一次就正确、准时、在预算内地完成工作。

参考文献

- ¹ Stevens et al. Systems engineering, coping with complexity. Prentice Hall Europe, 1998. ISBN 0-13-095085-8.
- ² Hull, Jackson & Dick. Requirements Engineering. Springer-Verlag, 2002. ISBN 1-85233-577-7.
- ³ OMG Systems Engineering Domain Special Interest Group. <http://syseng.omg.org/>.
- ⁴ U2-Partners SysEng Workshop. <http://www.u2-partners.org/syseng/>.
- ⁵ Cockburn, Alistair. Writing Effective Use-Cases. Addison-Wesley, 2000. ISBN 0-20170-225-8.
- ⁶ “Guidelines for Successful Acquisition and Management of Software-Intensive Systems: Volume 1 -- Version 3.0 Chapter 12.” United States Air Force, May 2000. Available at http://www.stsc.hill.af.mil/resources/tech_docs/.
- ⁷ “Tau/Architect for Systems Engineering, Application Note.” Telelogic 2002. Available at <http://www.telelogic.com/>.
- ⁸ “UML 2.0 Test Profile RFP.” OMG. Available at <http://www.omg.org/>.
- ⁹ UML 2.0 Testing Profile Consortium. <http://www.fokus.gmd.de/u2tp/>.