



应用白皮书

## **Tau/Architect** 在系统工程中的应用

以火箭控制系统为例进行介绍

October 25<sup>th</sup> 2002

## 目录

<b>1</b>	<b>项目介绍.....</b>	<b>3</b>
1.1	UML 2.0 标准语言与系统工程师.....	3
1.2	系统工程师的专业要求.....	4
1.3	没有任何代码的系统行为描述.....	4
<b>2</b>	<b>我们从哪着手开始? .....</b>	<b>5</b>
2.1	需求驱动.....	5
2.2	白板方式.....	5
2.3	使用 UML 语言.....	6
2.4	使用开发工具.....	6
2.5	我们开始对系统工程建模.....	7
<b>3</b>	<b>模型驱动的开发 (Model driven development) .....</b>	<b>8</b>
3.1	什么是我们努力想达到的目的? .....	8
3.2	每一个子系统行为怎样描述? .....	8
3.3	用例图 Use Cases.....	8
3.4	顺序图 Sequence Diagrams.....	9
3.5	状态图 State Charts.....	9
3.6	动作语言 Action Language.....	10
3.7	模型验证 Model Verification.....	10
3.8	模型执行 Model execution.....	10
3.9	模型调试 Model debugging.....	11
3.10	场景重放 Scenario Replay.....	11
3.11	局部行为 Partial behaviour.....	12
3.12	非严格的行为 Non-rigorous behaviour.....	12
<b>4</b>	<b>总结.....</b>	<b>13</b>
4.1	参考.....	13
	<b>附件 A – 火箭控制系统 (MFCS) 的需求.....</b>	<b>14</b>

## 1 项目介绍

### 1.1 UML 2.0 标准语言与系统工程师

UML 2.0 标准的模型描述语言能够给系统工程师带来巨大的设计帮助 – 我们这里所说的系统工程师是什么意思呢？在很多的开发团队, 系统工程师的角色可能被叫作架构或系统分析师, 实际上, 我们所提及的系统工程师就是将客户文本形式的需求转变为可实现的系统架构体系。

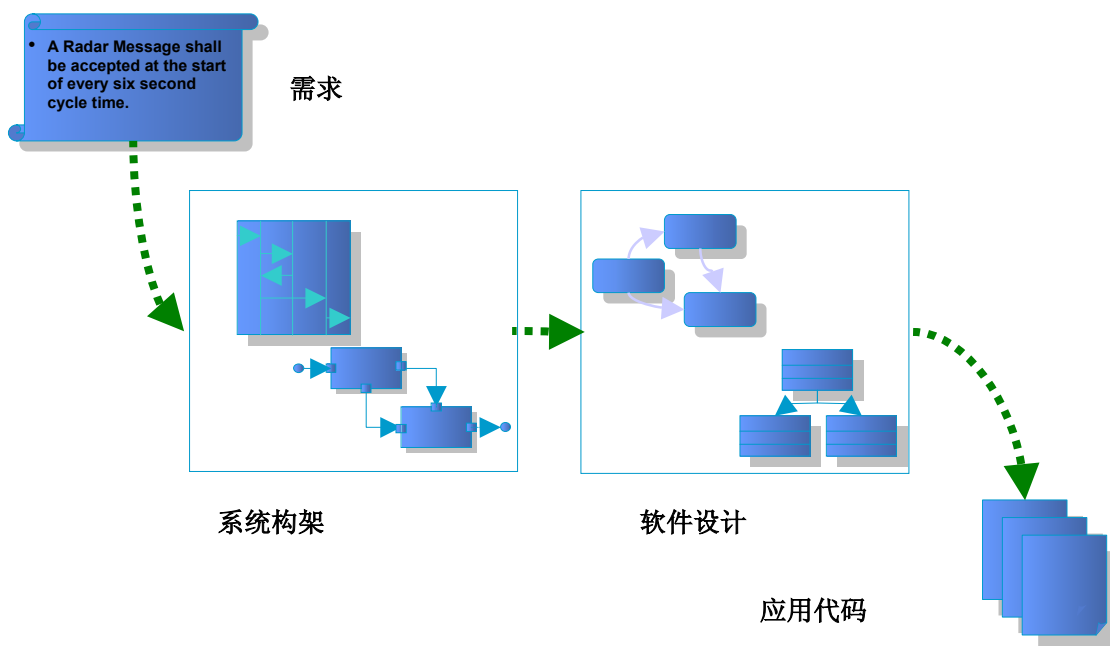


图 1: 系统工程师创建的系统结构体系

一个描述系统的蓝图被创建, 子系统被确定下来, 所有接口也被定义好, 通常这样的蓝图是绘制在一张纸上或一个白色书写板上, 或用绘图工具绘出。我们可以想象, 有多少项目我们可以在一个白色书写板上通过描画块示意图和草稿流程图描述出来, 并且在白色书写板的某个角上还要标注“不要擦掉它”。见图 2。

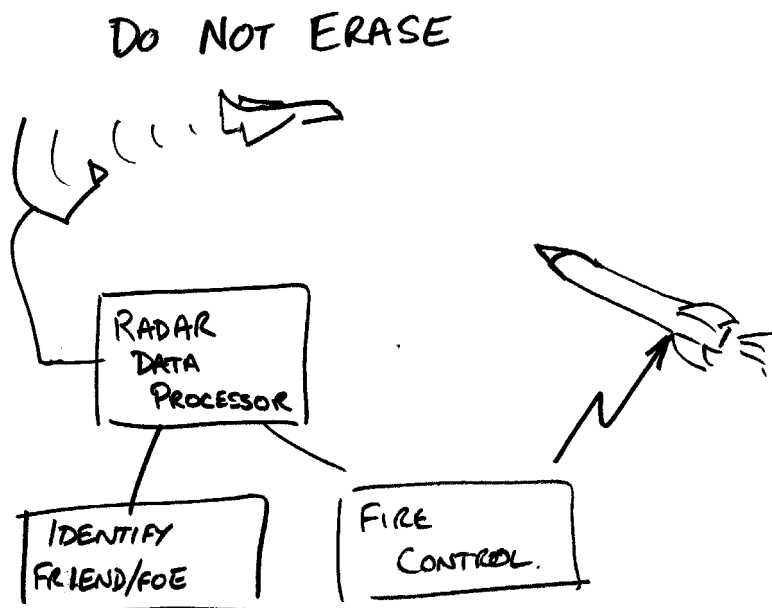


图 2: 一个白色书写板描述的系统

我想说的是，UML 2.0 标准建模语言为系统工程师提供了很多的描述方法，你能在[Ref 1]中读到很多有关的详细介绍。在一个很短的摘要里，UML 2.0 为我们提供了非常强大的支持去规范化地描述系统架构和系统接口，以及系统的分解。通过 Tau/Architect 支持我们去画出系统的 UML 图，更重要的是它能帮助我们做好以下两件事：

- 使用 Model 验证器动态地调试（debug）系统架构体系；
- 能够给你很强的信心，你不但可以从容地与你的客户进行需求上的沟通协调，同样你也可以将你的系统架构体系变为实际应用的软件。

## 1.2 系统工程师应是哪方面的专家？

系统工程师需要的是在火控系统、电话交换系统、发动机管理系统等方面的技术专家，并不希望他们是精通“C”语言的程序员。--为什么会是这样呢？这就是为什么在新一代的航空航天、通信和汽车领域市场上，所有公司需要的是象有火控系统等专业知识的专家，而不在与他们在编写代码上的能力。

## 1.3 描述系统行为不需要写任何代码

UML 2.0 标准建模语言和 Tau/Architect 支持了一个非常重要的设计概念：一种行为语言。通过提供一种用自然行为语言描述系统的方法，就可将系统工程师解放出来专注于其精通的系统方面的技术研究，并且不需要写代码就可产生出一个可运行的系统模型，更早地产生一个无需代码的可运行的系统原型。这样，在没有任何代码的情况下，系统工程师就能够很容易让他们的客户直观地看到需求的未来结果，软件工程师很容易明白系统的真正需求要求。下面我们将使用火箭控制系统的例子进行讲解。

## 2 我们怎么开始工作？

### 2.1 需求驱动

这个简单例子的需求列于附件 A，需求分析与管理 DOORS 与系统分析 Tau Architect 的集成性表现在[Ref 2]，这里我们不再多说。

我们通常在获得一定的需求后，很快就能在脑海里形成一个系统的概念轮廓图形，比如火箭火控系统(MFCS)”，也许我们可以在脑中形成如下的：

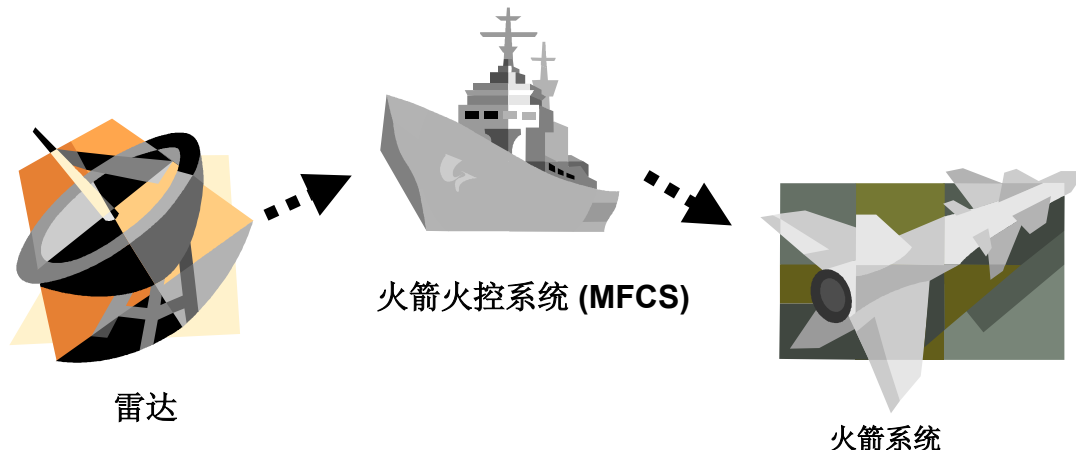


图 3: 我们脑中虚幻出的一个系统描述图形

### 2.2 白色写字板.

然后我们来到白色写字板，可以象这样如下的方法画出系统的方块图：

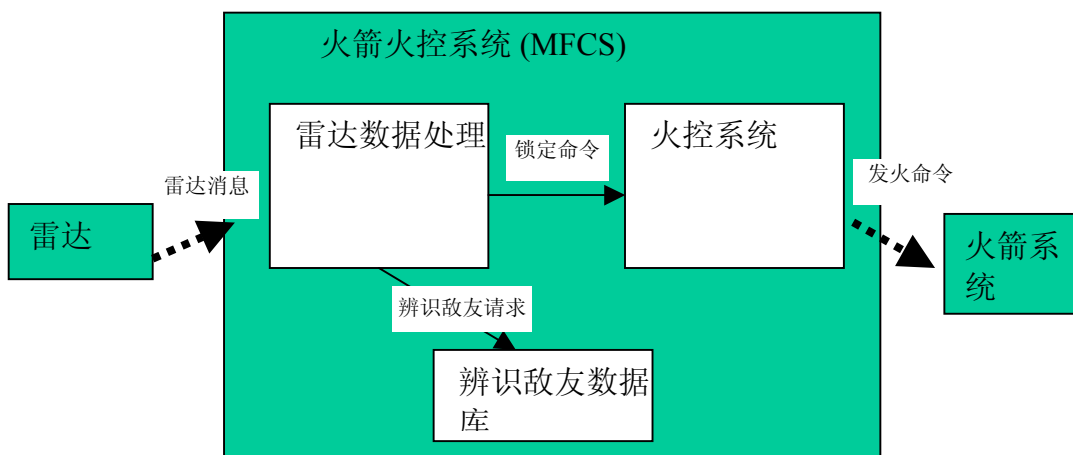


图 4: 一个非正式的系统描述方块图

## 2.3 使用 UML 替代上面方法

如果你有 UML 的知识与经验，那么你就会使用 UML 符号画出系统块图，这样的工作要显得正式得多。如所示：

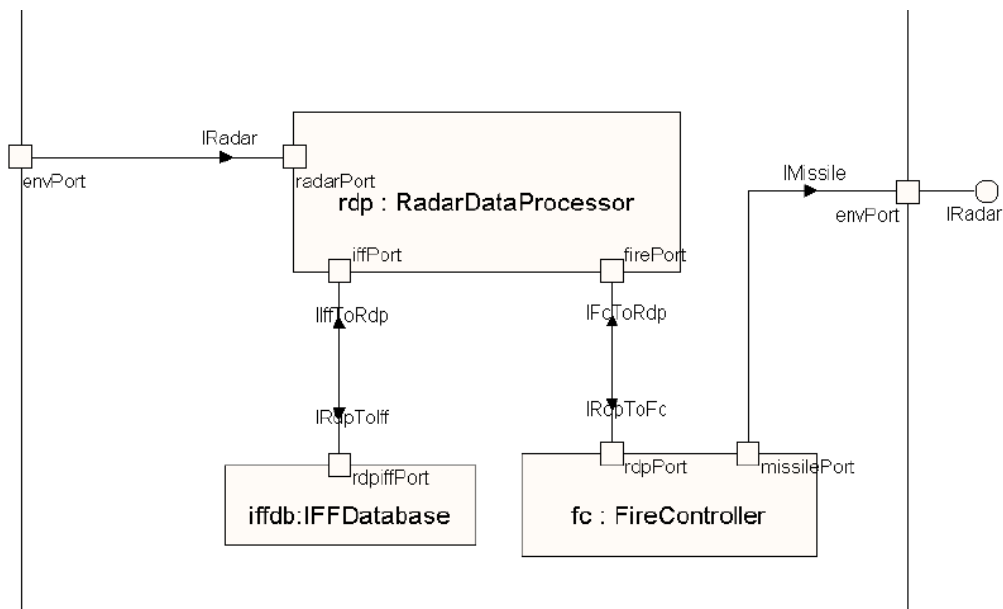


图 5: 用 UML2.0 描述的系统块图

## 2.4 使用工具

当然，你完全可以在 Tau Architect 工具里建立真正的系统架构图，从而你就可以将你所画的系统描述应用到真正的系统工程和软件工程应用中。

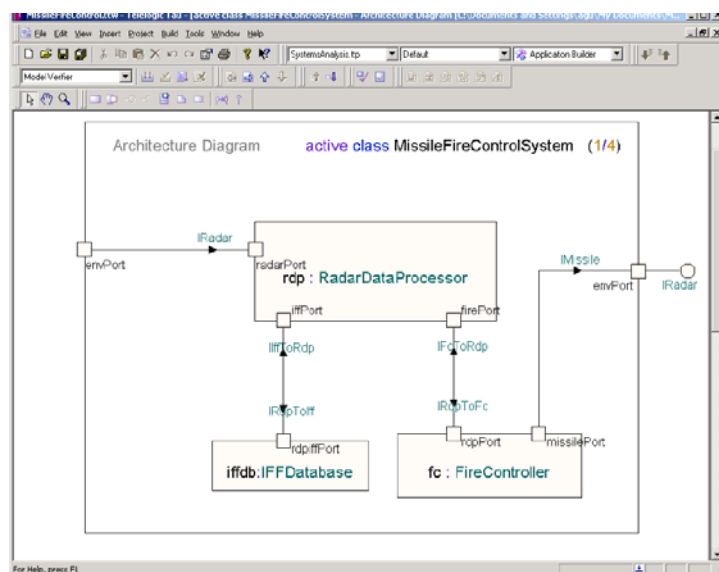


图 6: 在 Tau Architect 中描述的系统架构图

## 2.5 我们开始系统建模工作

一旦我们在工具中建立电子的需求图形，你就可以对它进行打印、共享、发邮件、移动或修改、产生文档等。但是，更重要的是，你将从现在开始就可以去对系统建模。

### 3 模型驱动的开发

#### 3.1 我们将对系统架构做什么？

“一个 UML 模型的建立从需求分析开始，首先变成系统体系结构，然后是软件设计，最后是应用系统。”

以上意味着：我们一定要努力在第一步获得正确的模型，如果我们能做到这点，那么我们才能确信软件设计的正确性和最终应用的正确性。“越在早期发现错误，就越容易修正错误，我们为修正错误付出的代价就越小。”这就是我们需要做的。

#### 3.2 怎样描述每一个子系统的行为？

在我们系统架构图中，我们已经指定了一些分系统和定义了一些接口，对于每一个 UML 所称为的“part”，我们需要去指定或描述他们的行为。你能够通过对于每一个“Part”创建状态机去描述一个易于定义的系统，我们一会儿将看到怎样去做。无论如何，你将发现它很有用地去完成其他的 UML 步骤建。

#### 3.3 用例图（Use Cases）

一些系统工程师喜欢首先选择将他们的文本需求转化成 Use Case, (see [Ref 3]), 因此，Use Case 图(UCDs) 被创建。另一些系统工程师更喜欢与客户或最终用户进行面谈并首先建立 Use Case 图(UCDs)，，然后再继续描述每一个 Use Case 文本。任何方法，最终形成的结果都如下图所示。

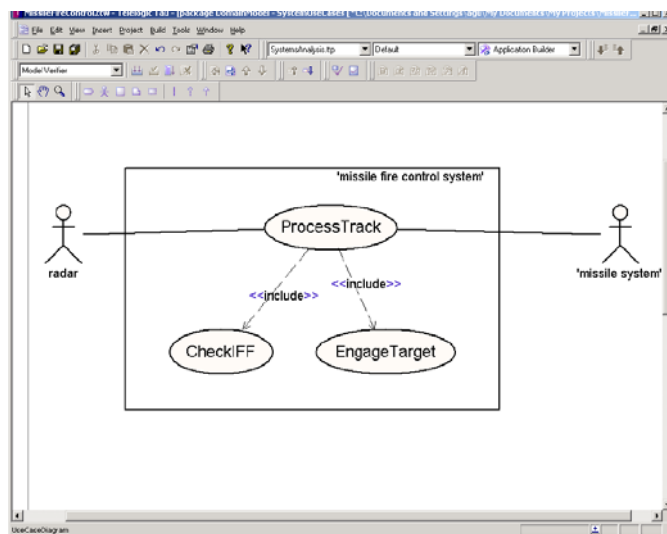


图 7: Use Case 图



### 3.4 顺序图（Sequence Diagrams）

然后，对每一个 Use Case，你将从它向下去描述它的动态行为，即期望事件发生的顺序。顺序图非常的直观和容易理解，下面是一个简单的顺序图例子：

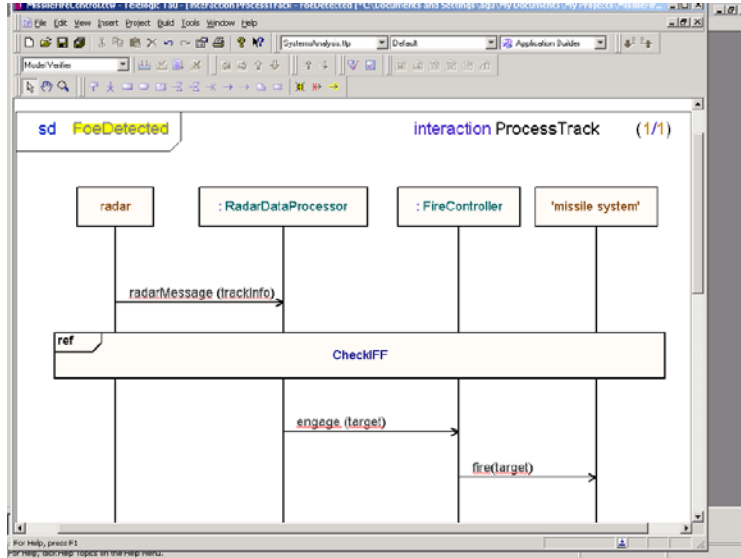


图 8: 一个顺序图例

### 3.5 状态图（State Charts）

好的，那么现在我们描述系统架构图中我们系统的主要“Parts”，我们将看到我们怎样期望整个系统被使用，卫星控制系统如何与外空世界打交道，事件顺序如何发生。现在，我们在某个位置去描述每一个“Part”的行为，我们所做的工作就是为每一个“Part”画一个状态图。

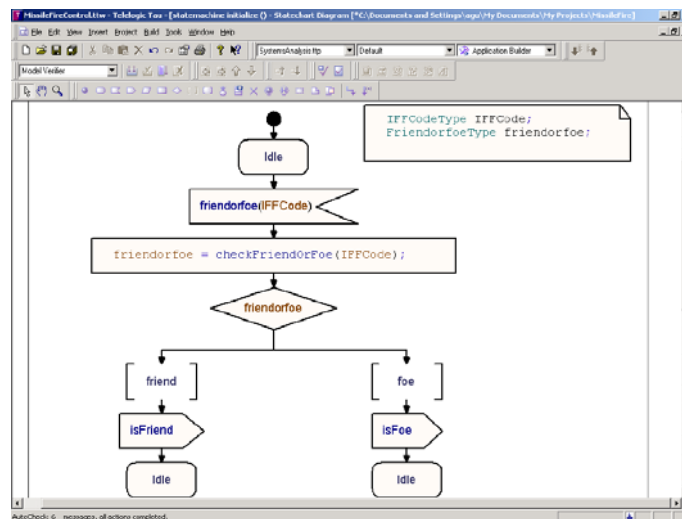


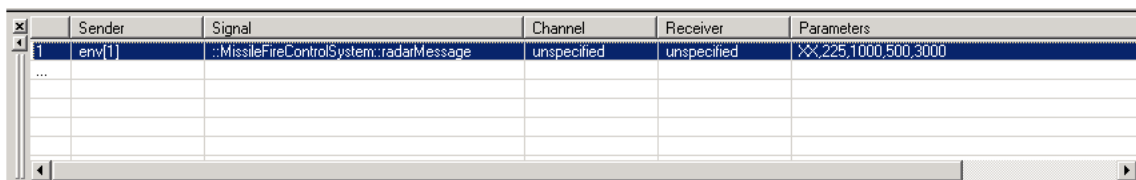
图 9: 一个状态图的例子

### 3.6 Action Language

描述系统工程的各种图形上的文本就是我们术语上定义的“Action Language”。UML 2.0 标准描述语言已经对“Action Language”的语义进行了详细的定义，我们的工具 **Tau Architect** 完全遵从这些语义法则，并且允许我们用直观和概要的方式去描述系统的行为。这种被建立在 UML 语义集中的“action language”，当然非常好定义系统，这就意味着使用象 **Tau Architect** 这样的工具很容易去实现多个有用的事件。在此白皮书中，我们将带给大家一个快速的模型验证演示，对系统工程师来讲这是一个非常重要的性能和功能。另外，“action language”语言提供的能力是 100%的代码自动生成，我们称之为“自动应用生成”，这项功能是由 **Tau Developer** 支持。

### 3.7 模型验证

到了这里我们已经完成了大量的建模工作，我们实际做的工作就是在 **Tau Architect** 中画了很多的 UML 图形，这就是系统的建模。象其他的模型建模，我们可以将以前的模型作为桥梁和参考进行建模，并找出这些模型在某种条件下是怎样的行为。Like other models that we can think of like models of bridges and models of planes, we would like to find out how the model behaves under certain conditions. 一旦我们完成了这个系统的建模，我们就能够通过发送一个消息或信号对系统进行仿真，这个例子的仿真将卫星控制系统在外太空的一些事件表现出。In our MFCS, 我们想发送一个雷达信号，期待系统去处理这个信息并在必要时反馈回一个发射信息。发送一个信息很容易做到。如下图：



	Sender	Signal	Channel	Receiver	Parameters
1	env[1]	::MissileFireControlSystem::radarMessage	unspecified	unspecified	XX,225,1000,500,3000
...					

Figure 10: 消息或信号输入窗口

### 3.8 系统模型的运行

随后我们可以按下设计界面中“Go”按钮，这时系统将接受我们发送的信息，并开始运行模型。我们将从顺序图上获得系统运行时所发生的情况，在我们眼前产生出围绕每个模型的传递信息和生成结果、不同动作的发生以及最后结果的生成。

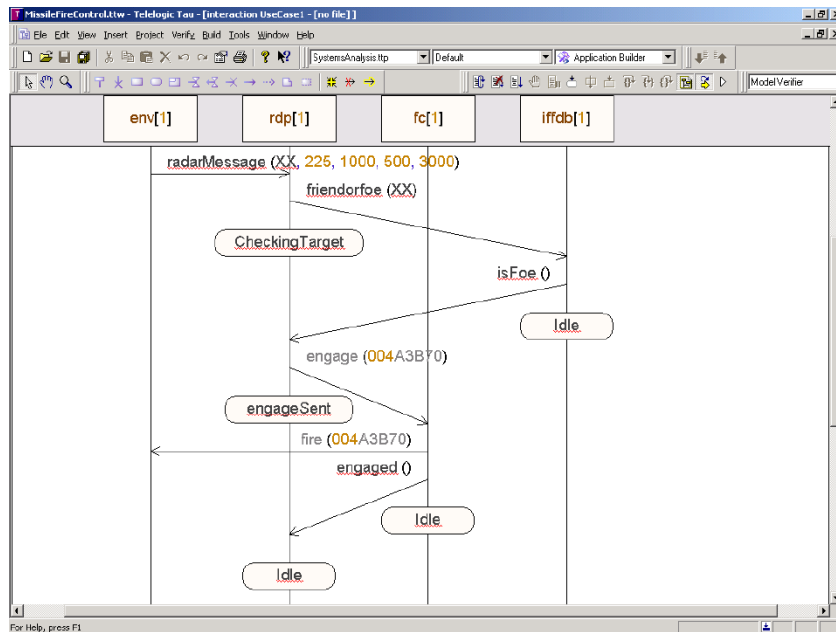


Figure 11: 一个动态生成的顺序图

### 3.9 模型调试 (Model debugging)

在我们的这个适当的小例子中，我们能发送一个信息并跟踪和观察屏幕上所将发生的情况。但是，对于一个非常复杂的系统，我们可能想使用模型验证器的全部性能，那么，我们能在每个变量上设置观察点，通过设置时间，能够单步一个时间一个状态的执行系统的运行过程。这将使我们非常容易的控制模型的执行过程，如果你有过 C 或 C++ 调试经验的话，这个调试过程对你是非常的熟悉。事实上，这儿所做的工作就是使用模型验证器帮助你去调试系统的模型，所有这些过程都不需要写一行代码就可完成调试工作。

### 3.10 情节重放 (Scenario Replay)

一旦你已经完成多个信息的调试，你将注意到这些信息的一个列表已经为你存储在一个窗口中。

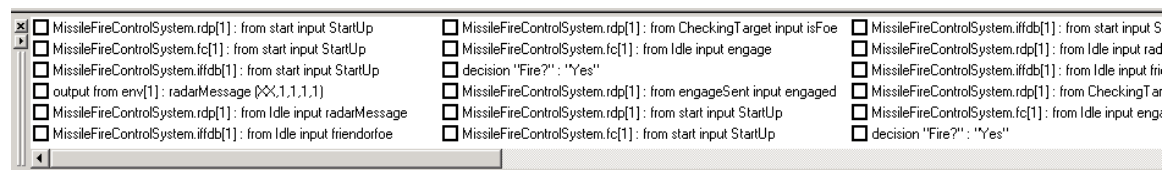


图 12: 捕获的调试情节准备用于调试过程重放

意思是如果你想回到模型调试开始的地方重新开始工作，你能够重放以前每一次的消息执行情况，这对开始调试任务、初始化系统、对感兴趣的地方交互式调试是非常有用的。同样，你也可以存储这些调试情节以便将来使用。

### 3.11 Partial behaviour

一般来讲，一个基于建模的工具是能够对具有真正意义的事件产生出某种设想结果的，因此，如果你想运行已经建立的系统构架模型，但是你还没有给每一个组成系统构架模型的“Part”建立状态图，那么，Tau/Architect 将为你生成一个“虚拟的状态机”，这意味着你就能够得到正确的系统构架，然后你可以插入行为，有效地测试每一个部件就好像你完成全部的工作和画好全部的图形一样。这至关重要。如果你建立一个复杂巨大的系统就象建简单系统一样，它也反映出最终系统自动被产生出代码的时候和在集成测试期间你正集成部件的整个过程。

### 3.12 非严密行为的描述

时常会发生这样的情况，特别是在项目的早期，对一些看起来可能很复杂的事件，可能你还没有准备好用何种正确的方法去描述它们，或者有可能你正想用多个方法去试探着调试模型，给这些事件预留出的空位置是非常有用的。Tau/Architect 允许你用通常的语言去指定这些疑问，这些通常语言在遭遇模型验证期间，显示作为对使用者的回答。

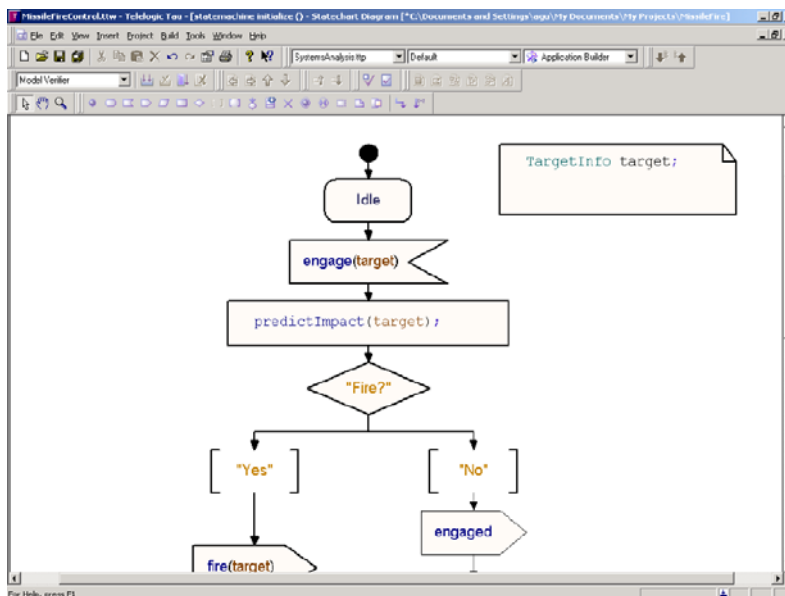


图 13: 带有非正式文本语言的状态图

## 4 总结

这是一个系统工程非常高层次的设计例子，它表达了系统工程师如何一步一步地将一个系统工程的文本需求转移完整的系统架构体系设计，通过对系统工程的主要“**Parts**”或子系统的规范描述，然后定义一些或全部“**Parts**”的行为，我们就能够在不写一行代码的情况下运行系统的模型，获得应用系统运行的结果。这将帮助我们达到如下的目标：

- 确保系统需求的正确性；
- 确保系统能够正常工作；

这样，系统工程师能够放心的将系统架构模型交付给软件工程师，软件工程师也可放心地将系统架构模型转到 **Tau Developer** 中进行软件设计和实现，并使用 **Tau Developer** 自动应用生成的性能，最终完成由系统设计到正确运行的应用系统。

请记住我们从什么地方开始系统设计，到什么地方完全实现应用系统：

**“ UML 模型，从系统需求分析开始，到变成第一个系统架构, 接着是软件工程设计到最后的正确工作的应用系统” .”**

### 4.1 参考

[Ref 1] White Paper “Using UML 2.0 to Solve Systems Engineering Problems”, Ian Barnard

[Ref 2] White Paper, “Using DOORS with Tau/Architect”, Thomas Hjelm

[Ref 3] 15-minute guide, “Visualising requirements in UML”, Ian Alexander

All of the above are available from <http://www.telelogic.com>.

## APPENDIX A - MFCS Requirements

1. A Radar Message shall be accepted at the start of every 6 second cycle.
2. A Radar Message shall contain for all tracks.  
*Track Data [IFF Code (2-letter string), Range (metres), Height (metres), Speed (metres/sec), Heading (0-359 degrees with 0 being Magnetic North)]*
3. MFCS shall process each Track Data before the end of the six second cycle period within which and Radar Message is received.  
 3.1 *Derived requirement:* compare IFF Code with IFF Database
4. Tracks determined as Friendly shall be ignored.
5. Tracks determined as Foe shall be engaged.  
 5.1 *Derived requirement:* FCS shall be instructed to engage via Engage Command [Target Id, Range, Height, Speed, Heading]  
 5.2 *Derived requirement:* FCS shall determine 'point of impact' by processing Engage Command information and predicting point of impact from own position and known missile speed.
6. MFCS shall issue Fire Command [3D-Heading] to Missile System.  
 6.1 *Derived requirement:* FCS shall issue Fire Command to Missile System

### MFCS Requirements organized in DOORS

