1. **学习上个案例剩余部分：**
2. **Mobile Robot**

这个问题集中在嵌入式实时系统上。这些系统必须处理外部传感器和执行器，并且必须根据系统在其环境中的活动及时做出响应。

考虑移动机器人通常必须完成的以下活动：

* 获取其传感器提供的输入。
* 控制车轮和其他可移动部件的运动，
* 规划其未来之路。

许多因素使任务复杂化：

* 障碍物可能会阻塞机器人的路径。
* 传感器输入可能不正确。
* 机器人可能没电了。
* 机械限制可能会限制机器人移动的准确性。
* 机器人可能会操纵有害物质。
* 不可预测的事件可能会留下很少的时间进行响应。

[历史](http://www.cs.cmu.edu/~ModProb/MRhist.html)

多年来，移动机器人领域提出了许多架构建议。在这里介绍的解决方案中，我们将考虑四个建议，从分层范例[Elfes87]到黑板结构[Shafer86]。

该领域的丰富性使我们可以对不同的研究人员为他们的机器人项目选择的重点进行有趣的比较，并在各种选择之间进行权衡。下一节将调查要考虑的因素。

[设计注意事项](http://www.cs.cmu.edu/~ModProb/MRdsgn.html)

我们陈述了机器人结构的以下要求。

R1：体系结构必须适应协商和响应行为。机器人必须协调其为实现其指定目标（例如，收集岩石样本）而故意采取的行动，以及环境（例如，避开障碍物）对其施加的反应。

R2：架构必须考虑不确定性。机器人操作的所有情况永远无法完全预测。该架构必须提供一个框架，即使面对不完整或不可靠的信息（例如，相互矛盾的传感器读数），机器人也可以在其中发挥作用。

R3：架构必须考虑机器人操作及其环境所固有的危险。通过综合考虑容错（R3a），安全性（R3b）和性能（R3c）属性，该体系结构必须有助于维护机器人，其操作员及其环境的完整性。诸如电源减少，危险的蒸气或意外打开门之类的问题都不会带来灾难。

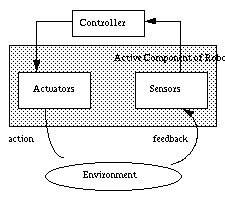
R4：体系结构必须赋予设计者灵活性。移动机器人的应用程序开发经常需要进行实验和重新配置。此外，任务的更改可能需要定期进行修改。

这些要求的适用程度既取决于机器人编程要执行的工作的复杂性，又取决于其环境的可预测性。例如，作为太空任务的一部分，当机器人在另一个星球上运行时，容错能力至关重要。何时可以将机器人带到附近的维护机构，这仍然很重要，但不太重要。

[解决方案](http://www.cs.cmu.edu/~ModProb/MRsoln.html)

我们研究了已经在机器人上实现的四种主要架构。这些包括

* 洛萨诺（Lozano）的[控制循环](http://www.cs.cmu.edu/~ModProb/MRsol1.html)[Lozano90]
* 这是控制循环范例的模型：

  
*控制回路架构*

大多数工业机器人都支持对不可预测事件的最小处理：任务是完全预先定义的（例如，将某些汽车零件焊接在一起），并且机器人对其环境不承担任何责任（环境是负责不干扰环境的责任）。机器人）。开环范式自然适用于这种情况：机器人启动一个动作或一系列动作而不必费心检查其后果[Lozano90]。

将这种范例升级为移动机器人需要添加反馈，从而产生一个闭环架构。控制器启动机器人动作并监视其后果，并根据此返回信息调整未来计划。

（R1）闭环范式的一个优点是它的简单性：它捕获了机器人与外界之间的基本相互作用。

在更不可预测的环境中，它的简单性也是一个缺点。一位专家[Lozano90]评论说，反馈回路假设环境的变化是线性的，并且需要线性反应（例如，通过逐步打开和关闭阀门来控制压力）；然而，机器人大多面临不同的离散事件，这些事件需要在非常不同的行为模式之间进行切换（例如，在控制机械手运动和调整基本位置之间，以防止失去平衡）。该模型没有提供有关如何管理各种事件的任何提示。

对于复杂的任务，控制循环无法将软件分解为协作组件。如果必须完善感测，计划和执行的步骤，则其他范式必须提供控制回路模型所缺乏的细微差别。

（R2）为了解决不确定性，控制回路范式偏向一种方法：通过迭代减少未知数；具有行动和反应的反复试验过程消除了每一步的可能性。如果需要更多微妙的步骤，则该体系结构不提供用于将这些步骤与基本循环固定在一起或将其委托给单独的实体的框架。

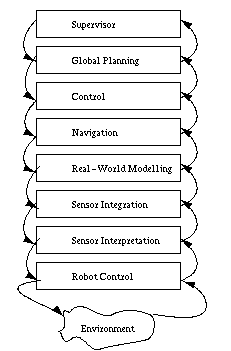
（R3）闭环范例支持容错性和安全性，因为它的简单性使复制变得容易，并减少了错误蔓延到系统中的机会。

（R4）机器人体系结构的主要组件（监控器，传感器，电动机）彼此分离，并且可以独立更换。必须在模块内部进行更精细的调整，详细程度未显示体系结构。

总而言之，闭环范例似乎最适合仅需要处理少量外部事件并且其任务不涉及复杂分解的简单机器人系统。

* Elfes的[分层组织](http://www.cs.cmu.edu/~ModProb/MRsol2.html)[Elfes87]，

该图显示了Alberto Elfes对理想化分层体系结构[Elfes87]的定义，该体系结构影响了在Terregator和Neptune移动机器人[Champeny93，Podnar84]上实现的海豚声纳和导航系统的设计。

  
*分层架构*

在最低级别1，驻留机器人控制例程（电机，关节等）。

第2级和第3级处理来自现实世界的输入。它们执行传感器解释（对来自一个传感器的数据的分析）和传感器集成（对不同传感器输入的组合分析）。

级别4与维护机器人的世界模型有关。

级别5管理机器人的导航。

接下来的两个级别6和7，计划并计划机器人的动作。处理问题和重新计划也是7级职责的一部分。

顶层提供用户界面和总体监控功能。

（R1）Elfes的模型通过定义更多可以委派所需任务的组件，从而避免了控制循环遇到的一些问题。由于专门针对自主机器人，它指出了必须解决的问题（例如，传感器集成）。此外，它定义了抽象级别（例如，机器人控制与导航）来指导设计。

虽然它很好地组织了协调机器人操作所需的组件，但分层体系结构却不适合实际的数据和控制流模式。这些层建议在相邻组件之间传递服务和请求。实际上，正如Elfes容易接受的那样，信息交换并不那么简单。例如，必须将需要快速反应的数据直接从传感器发送到级别7的问题处理代理，并且相应的命令可能必须跳过级别以及时到达电动机。

该模型中的另一个不精确之处是它没有将体系结构中实际存在的两个抽象层次结构分开：

* 具有原始传感器输入（级别1），已解释和集成的结果（2和3）以及最终的世界模型（4）的数据层次结构。
* 具有电机控制（级别1），导航（5），调度（6），计划（7）和用户级控制（8）的控制层次。

结论中提到的NASREM体系结构在这方面更为精确。

（R2）抽象层的存在解决了管理不确定性的需求：最低层的不确定性可能会随着更高层中可用知识的增加而变得清晰。例如，世界模型中包含的上下文可以提供线索，以消除冲突的传感器数据的歧义。

（R3）抽象机制也可以提供容错能力和被动安全性（当您努力不做任何事情时）。数据和命令从不同的角度进行了分析。可以将许多检查和平衡合并到系统中。

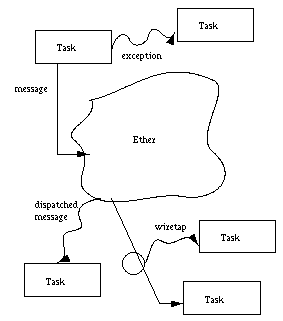
如前所述，性能和主动安全性（当您必须做某事而不是避免做某事时）可能需要使通信模式短路。

（R4）模糊的依存关系是易于替换和添加组件的障碍。随着每次更改，各层之间的脆弱关系可能变得更加难以解密。

总而言之，分层体系结构定义的抽象级别提供了构成软件体系结构总体目标的要素：组织组件的框架。它通过精确地了解不同层的作用来实现此目标。

该模型的主要缺点是，当将其应用于实际实现所要求的更详细的细节时，它就会崩溃。机器人中的通信模式很可能不会遵循该体系结构所暗示的非常有序的方案。

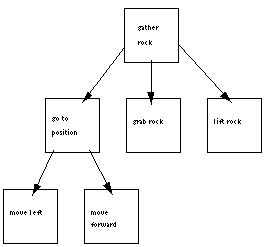
* Simmons的[任务控制架构](http://www.cs.cmu.edu/~ModProb/MRsol3.html) [Simmons92]，以及

该图总结了使用隐式调用的任务控制体系结构（TCA）[Simmons92]。除其他外，它还应用于Ambler机器人[Simmons90]。  
  
*隐式调用架构*

TCA不仅是一种体系结构；它还提供了一个用于构建机器人的精密工具箱：一个实现TCA原理的通信和控制例程库。下面的讨论集中在任务树和隐式调用功能上。有关完整的概述，请参见参考资料。

TCA体系结构基于任务的层次结构，即任务树。下图显示了一个示例任务树。父级任务启动子级任务。软件设计人员可以定义任务对之间的时间依赖性。时间限制的一个示例是：“ A必须在B开始之前完成”。这些功能允许指定选择性并发。

TCA的例程在任务树上包含许多操作，可在运行时动态重新配置它们。

  
*任务树*

在TCA中，任务通过将消息发送到中央服务器进行通信，该服务器将消息重定向到已注册以处理它们的任务。这种方案在发送方不需要知道接收方的情况下，是隐式调用的基本特征。

TCA的功能还包括另外三种隐式调用机制：

* 异常：某些条件导致执行关联的异常处理程序。异常将覆盖导致异常的子程序中当前正在执行的任务。它们快速改变了机器人的处理模式，因此比纯分层结构的反馈回路或漫长的通信路径更适合于管理自发事件（例如危险的地形变化）。

异常处理程序拥有处理任务树的所有操作：例如，它们可以中止或重试任务。

* 窃听：消息可以被叠加在现有体系结构（即任务树）上的例程拦截。例如，安全检查程序可以使用此功能来验证所有传出的运动命令。
* 监控器：监控读取的信息并在数据满足特定条件时执行某些操作。TCA手册中的一个示例是电池检查：如果电池电量低于给定的电量，则会调用为电池充电所需的操作。通过预留代理来监督系统，此功能提供了一种解决容错问题的便捷方法。

（R1）一方面是任务树，而另一方面是异常，窃听和监视器，因此可以将动作（任务树中体现的名义行为）和反应（由外部事件和环境所决定的行为）明确区分。

通过将并发代理纳入其模型，TCA也将自己与以前的范例区分开来。在TCA中，很明显，多个动作可以或多或少地独立地同时进行。其他两个模型没有显示并发的存在。

并发量受中央服务器功能的限制。通常，其对中央控制点的依赖可能是三氯乙酸的弱点。

（R2）TCA如何解决不确定性尚不清楚。如果存在不可思议的事物，那么可以建立一个暂定任务树，以供异常处理程序进行修改，前提是该假设基于错误的假设。

（R3）如上面的示例所示，TCA异常，窃听和监视功能考虑了对性能，安全性和容错性的需求。

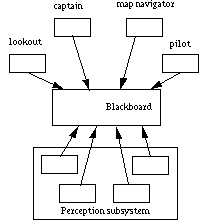
当多个处理程序注册同一信号时，将通过冗余实现容错。如果其中一个不可用，TCA仍可以通过将请求路由到另一个来提供服务。由于多个处理程序可以并发处理同一请求的多次出现，因此性能也会受益。

（R4）隐式调用的使用使组件的增量开发和替换变得简单：通常在中央服务器上注册新的处理程序，异常，窃听或监视程序就足够了；没有现有的组件会受到影响。

总而言之，TCA提供了一套全面的功能来协调机器人的任务，同时尊重质量和简化开发要求。该方案的丰富性使其最适合更复杂的机器人项目。

* Shafer在[黑板上](http://www.cs.cmu.edu/~ModProb/MRsol4.html)的应用 [Shafer86]。

该图描述了用于移动机器人的黑板架构。该范例已在NAVLAB项目中用作CODGER系统[Shafer86]的一部分。

  
< *黑板架构*

如[Shafer86]中所命名的“白板”架构，其抽象使人联想到分层架构中遇到的那些抽象。CODGER的组件为：

* “队长”，总督导。
* “地图导航器”，高级路径规划器。
* “监视”模块，用于监视环境中的地标。
* “飞行员”，低级路径规划器和电动机控制器。
* 感知子系统，这些模块接受来自多个传感器的原始输入并将其集成到一个连贯的解释中。

（R1）组件（包括感知子系统内部的模块）通过黑板系统的特征性中央数据库进行通信。模块表明他们对某些类型的信息感兴趣。数据库立即返回这些数据，也可以在其他模块将其插入数据库时​​返回。

例如，the望台可能会监视某些地理特征；例如，当感知子系统存储与描述匹配的图像时，数据库会通知它。

CODGER体系结构的一个困难是，即使在组件之间直接交互会更加自然的情况下，也必须强制所有控制流以适应数据库机制。

（R2）黑板也是解决机器人世界观中的冲突或不确定性的手段。例如，监视点的地标检测通过死区推算为距离估计提供了真实性检查，两者均存储在数据库中。负责不确定性解决方案的模块在数据库中注册以获得必要的数据。

此活动的主要示例是传感器融合，由感知子系统执行以协调来自其各种传感器的输入。

（R3）通过数据库的通信类似于通过TCA中央消息服务器的通信。异常机制，窃听和监视-反应速度，安全性和可靠性的保证者-可以在CODGER中实现，方法是定义单独的模块，这些模块监视数据库以发现意外情况或麻烦开始的迹象。TCA通过窃听两次检查消息的安全机制无法完全复制，因为一旦它在数据库中显示出来就无法阻止某个操作，可能为时已晚。（在窃听处理过程中，TCA保留该消息。）

（R4）与TCA一样，黑板体系结构提供了对并发的支持，并使发送者与接收者解耦，从而获得了维护的灵活性。

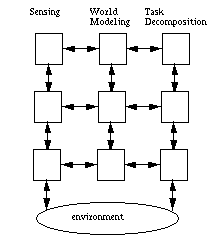
总之，由于基于数据库内容的隐式调用机制，黑板体系结构能够以非常灵活的方式为协调和不确定性解决方案的任务协作建模。这些功能仅比TCA的等效功能稍差一些。

## 总结与比较

我们已经看到了四种体系结构，其中两种（分层体系结构和黑板）非常具体，并给出了机器人所需组件的精确指示。其他两个（控制循环和隐式调用）没有定义功能组件，而是专注于机制。

特定性有助于掌握自动机器人所涉及的基本抽象和任务。研究TCA架构（其机制中最强大的）的价值以及结合机器人任务的功能分解（计划，传感器集成等），将是很有趣的。

已经提出了其他混合架构。可以将NASA / NBS远程机器人标准参考模型（NASREM）[Lumia90]视为控制环和分层体系结构的组合。

  
NASREM体系结构

从上到下的层由它们执行任务的时间范围定义。从这个角度来看，该体系结构是具有越来越严格的响应时间约束的控制回路的层次结构。

从左到右的层代表功能抽象。

总而言之，该表总结了所审查软件体系结构的优缺点。

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **控制回路** | **层数** | **没错 呼吁。** | **黑板** |
| **任务协调** | +- | -- | ++ | + |
| **处理不确定性** | -- | +- | +- | + |
| **容错能力** | +- | +- | ++ | + |
| **安全** | +- | +- | ++ | + |
| **表现** | +- | +- | ++ | + |
| **灵活性** | +- | -- | + | + |

机器人架构的优缺点

1. **[The idea](http://www.cs.cmu.edu/~ModProb/idea.html) of model problems**

对于一门学科，尤其是仅仅发挥自己才智的学科，在教学和学习中采用一些共同的，定义明确的问题是很常见的。通常被称为*模型系统*或*类型问题*，它们提供了一种比较方法和结果，在标准示例上制定新技术并为新参与者设置最低功能标准的方法。随着时间的流逝，解决其中一些问题的合理方法成为了认真考虑新技术的代价。模型问题还提供了预先调试的教育练习源。

例如，生物学有

* *果蝇*（*果蝇*）
* *褐家鼠*（*Rattus rattus Norwegicus*）
* *大肠杆菌*（消化细菌）

这些都是本领域话语通用语言的一部分。每个实例都提供了一个熟悉的具体实例，说明了一系列重要问题。这使得讨论可以从基本示例的共享知识开始，并迅速进行到当前感兴趣的结果，理论或技术。

言归正传，计算机科学在许多领域都存在模型问题。熟悉的例子包括

* *算法和数据结构：* 排序，搜索，最大公约数，素数整数，集合，堆栈，队列
* *同步：*读者/作家，生产者/消费者，饮食哲学家，吸烟者
* *编程方法：*八皇后，河内塔
* *正式规格：*电报机，电梯（位于大西洋西侧的电梯），图书馆
* *组合优化：*旅行推销员

在此站点上，我们提出了软件体系结构的几个模型问题，讨论了它们提出的有趣的设计问题，并展示了该小组中的某些工作是如何解决它们的。

我们的目的是激发对这些问题，潜在的其他问题以及选择问题以及评估或比较解决方案的标准的讨论。为此，这是一个生活场所。我们将尝试结合评论和建议以及简短的解决方案。我们愿意就应解决多长时间的解决方案或替代解决方案的比较提出建议。

1. **[The utility](http://www.cs.cmu.edu/~ModProb/benefits.html) of model problems**

软件体系结构的模型问题应有助于我们专注于特定的体系结构问题。这些问题包括

* 描述系统组织，并描述特定类型的系统组织（体系结构样式）
* 区分模板，实例和调用
* 区分不同类型的系统组织-不仅是结构上的差异，而且是这些差异的含义
* 在不同的架构选择中进行选择
* 同时使用不同的模型或对设计进行不同的改进；在不同观点之间建立一致性
* 定义系统族
* 定义建筑的家庭或风格
* 描述具有固定结构的系统的动态行为并描述系统结构的动态变化
* 测量，评估或测试系统的属性，例如总体性能，可靠性或安全性
* 测量，评估或测试设计的属性，例如扩展或子集的难易程度

当然，可以响应于不同的问题而选择不同的问题。我们尚未尝试使问题独立或正交。如果它们重叠，则很好，但是随着集合的完善，每个集合都应包括有助于澄清的特定问题的描述。