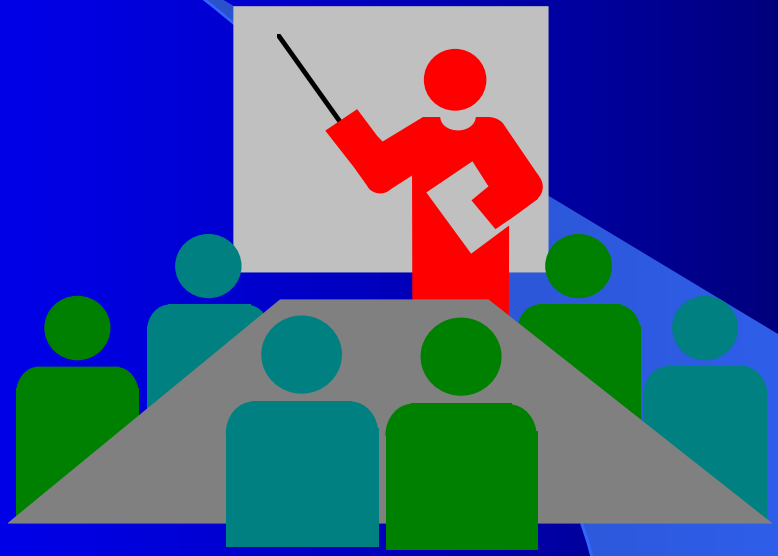
软件体系结构



二0一六年二月

第二章 软件体系结构建模

不管是在软件行业还是在其他各行各业，人们都广泛使用各种各样的模型。模型是对现实原型的抽象或模拟，这种抽象或模拟不是简单的复制，而是强调原型的本质，抛弃原型中的次要因素。

第二章 软件体系结构建模

在软件开发的各个阶段，都需要运用不同的方法对系统建立各种各样的模型，比如需求模型、功能模型、数据模型和物理模型等等，可以说整个软件开发的过程就是一个模型不断建立和不断转化的过程。

软件体系结构的建模方法是由建模语言和建模过程两部分组成。其中，建模语言是用来表述设计方法的表示法，建模过程是对设计中所应采取的步骤的描述。

第二章 软件体系结构建模

2.1 模型

2.2 元模型

2.3 建模方法

2.4 软件体系结构的生命周期模型

2.1 模型

1、结构化模型

2、框架模型

3、动态模型

4、过程模型

5、功能模型

6、4+1模型

2.1 模型

1、结构化模型

结构化模型是最常见的体系结构模型，综合软件体系结构的概念，软件体系结构的结构化模型由5种元素组成：构件（component）、连接件（connector）、配置（configuration）、端口（port）和角色（role），其中构件、连接件和配置是最基本的元素。

2.1 模型

1、结构化模型—构件

构件是具有某种功能可重用的软件模板单元，表示了系统中主要的计算元素和数据存储。

构件有两种：复合构件和原子构件，复合构件由其它复合构件和原子构件通过连接构成；原子构件是不可再分的构件，底层由实现该构件的类组成，这种关于构件的划分方法提供了体系结构的分层表示能力，有助于简化体系结构的设计。

2.1 模型

1、结构化模型—连接件

连接件表示了构件之间的交互，简单的连接件如：管道(pipes)、过程调用(procedure call)、事件广播(event broadcast)等，更为复杂的交互如：客户-服务器(c1ient-server)通讯协议、数据库和应用之间的SQL连接等。

2.1 模型

1、结构化模型—配置

配置表示了构件和连接件的拓扑逻辑和约束。

2.1 模型

1、结构化模型—端口

另外，构件作为一个封装的实体，只能通过其接口与外部环境交互，构件的接口由一组端口组成，每个端口表示了构件和外部环境的交互点。通过不同的端口类型，一个构件可以提供多重接口。一个端口可以非常简单，如过程调用，也可以表示更为复杂的界面(包含一些约束)，如必须以某种顺序调用的一组过程调用。

2.1 模型

1、结构化模型—角色

连接件作为建模软件体系结构的主要实体，同样也有接口，连接件的接口由一组角色组成，连接件的每一个角色定义了该连接件表示的交互的参与者，二元连接件有两个角色，如RPC的角色为Caller和Called，pipe的角色是reading和writing，消息传递连接件的角色是sender和receiver。有的连接件有多于两个的角色，如事件广播有一个事件发布者角色和任意多个事件接受者角色。

2.1 模型

2、框架模型

与结构模型类似，但它不太侧重描述结构的细节而更侧重于整体的结构。框架模型主要以一些特殊的问题为目标，建立只针对和适应该问题的结构。

2.1 模型

3、动态模型

是对结构或框架模型的补充，研究系统的“大颗粒”的行为性质。例如，描述系统的重新配置或演化。

具体内容在第五章介绍。

2.1 模型

4、过程模型

研究构造系统的步骤和过程，体系结构是遵循某些过程脚本的结果。

2.1 模型

5、功能模型

认为体系结构是由一组功能构件按层次组成，下层向上层提供服务。功能模型可以看作是一种特殊的框架模型。

2.1 模型

6、4+1模型

从多个视图描述软件体系结构，每一个视图描述软件体系结构的不同特征，这样有助于减少体系结构建模的复杂度，有助于设计人员对体系结构的理解。

2.1 模型

6、4+1模型

Kruehten在1995提出了软件体系结构的4十l视图描述，show在1996中提出了多维设计空间的概念用来描述软件体系结构的不同特征，分为功能设计空间(表示功能和性能需求)和结构设计空间(表示系统的初始分解结果)，相比较而言，4十l视图描述较为全面地描述了软件体系结构，它是以Booch方法的图形描述为基础，建立在对象模型之上，但缺少形式化描述基础。下面我们详细介绍“4+1”视图模型，通过它更好地理解软件体系结构概念。

2.1 模型

6、4+1模型

“4+1”视图模型从5个不同的视角来描述软件体系结构。

- 逻辑视图
- 进程视图
- 物理视图
- 开发视图
- 场景视图

2.1 模型

6、4+1模型

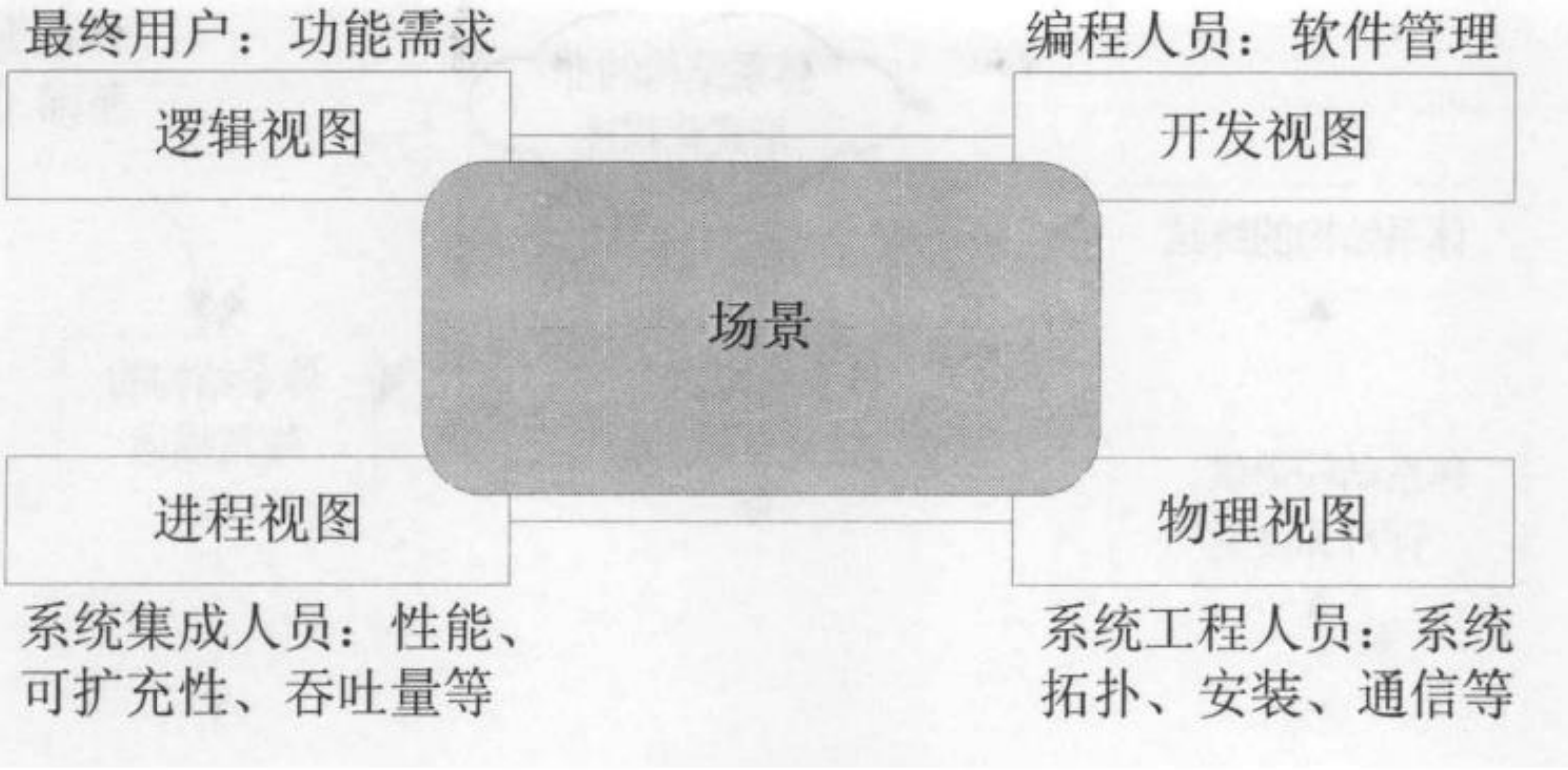


图 1-1 “4+1”视图模型

2.1 模型

6、4+1模型

1）逻辑视图

逻辑视图（logic view），也称概念视图，主要支持对系统功能方面需求的抽象描述，即系统最终将提供给用户什么样的服务。

在逻辑视图中，系统分解成一些列的功能抽象，这些抽象主要来自问题领域。这种分解不但可以用来进行功能分析，而且可用作标识在整个系统的各个不同部分的通用机制和设计元素，是系统工程师与领域专家交流的有效媒介。

2.1 模型

6、4+1模型

1）逻辑视图

逻辑视图强调问题空间中各实体间的相互作用，用实体一关系图来描述，若采用面向对象技术，则可以用类图来描述。其中的构件主要是类，风格通常是面向对象风格。

2.1 模型

6、4+1模型

2）开发视图

开发视图（development view），也称模块视图，主要侧重于系统模块之间的组织和管理。

根据系统模块的组织方式，这个视图可以有不同的形式。通常是根据分配给项目组的开发和维护工作来组织，例如：可根据信息隐蔽原则来组织模块，也可把系统运行时关系紧密或执行相关任务的模块组织在一起，也可把模块组织成层次结构。

2.1 模型

6、4+1模型

2）开发视图

该视图同逻辑视图不同，它与实现紧密相连，通常用模块和子系统图来表示接口输人输出。该视图的风格主要是层次结构风格，通常将层次限制在4—6层左右。在同一个系统中，子图中的模块之间可以相互作用，同一层中不同子图间的模块之间也可以相互作用，也可与相邻层的模块之间相互作用。这样可使每个层次的接口既完备又精练，减少各模块之间的复杂关系。此外，对于各个层次，越是底层通用性越强，当应用系统的需求发生变化时，所需的改变越少。

2.1 模型

6、4+1模型

3）进程视图

进程视图（process view）主要侧重于描述系统的动态属性，即系统运行时的特性。

该视图着重解决系统的并发和分布，及系统的完整性和容错性，同时也定义在概念视图中的各个类中的操作是在哪一个控制线索中被执行的。该角度的部件通常是进程，进程是一个有其自己控制线索的命令语句序列。当系统运行时，一个进程可被执行、挂起、唤醒等，同时可与其它进程通讯、同步等，并且通过进程间的通讯模式可对系统性能进行评估。进程视图有许多风格，如数据流风格、客户/服务器风格等

2.1 模型

6、4+1模型

4）物理视图

物理视图主要描述如何把软件映射到硬件上，通常要考虑系统的性能、规模、容错等。

当软件运行于不同的节点上时，各视图中的部件都直接或间接地对应在系统的不同节点上。因此，从软件到节点的映射要有较高的灵活性，当环境改变时，对系统其它视图的影响才比较小。此外，这种映射关系直接影响到系统的性能。

2.1 模型

6、4+1模型

从以上介绍可知，逻辑和开发视图描述的是系统的静态结构，进程和物理视图描述的是系统的动态结构。逻辑视图与开发视图虽密切相关，但它们的侧重点不同，系统规模越大，它们的差别也越明显。此外，值得注意的是，系统的拓扑结构在不同的视图下保持不变。

2.1 模型

6、4+1模型

5）场景

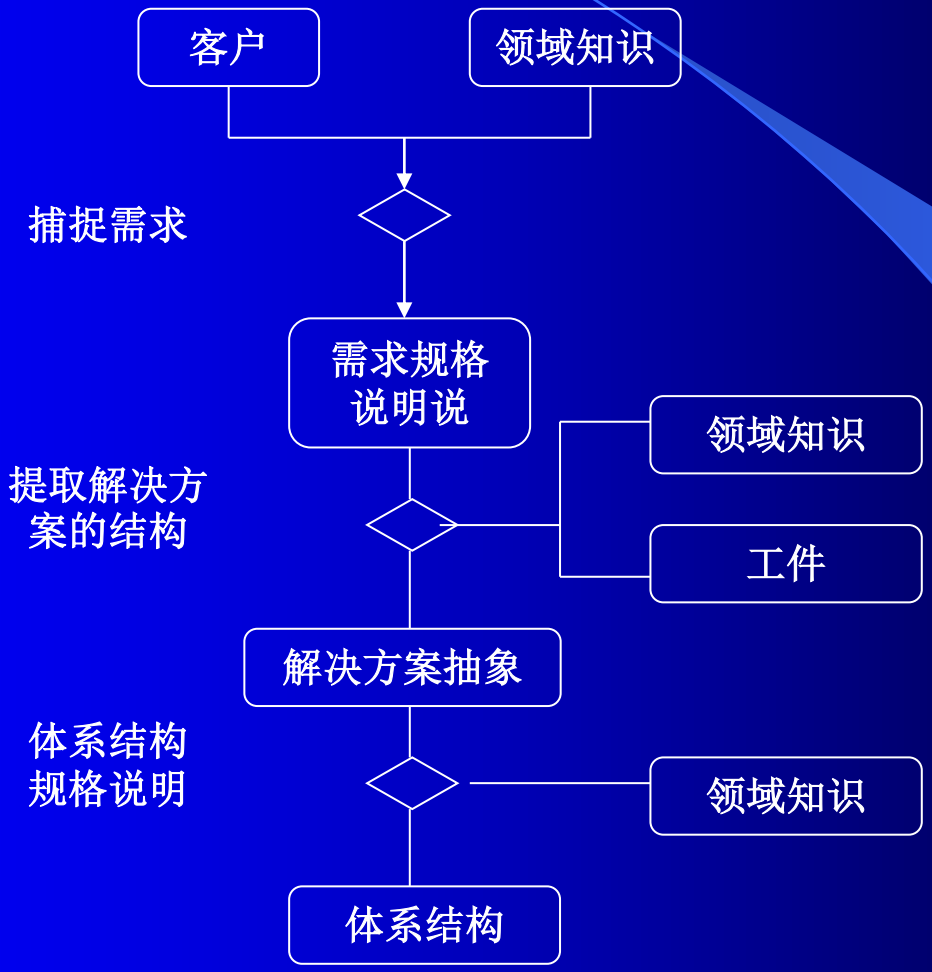
不同的视图描述了同一系统的不同侧面，所以它们之间并不是相互独立的。通过“场景”可将不同的视图联系起来。

所谓“场景”，可看作是重要的系统活动的抽象，在开发系统体系结构时，它可帮助设计人员找到体系结构的构件及构件间的相互作用关系。通过场景，可以分析系统的体系结构或某个特定视图，场景还可以描述不同视图的构件如何相互作用。场景可用文本表示，也可用图形表示，如OID图。

2.2 体系结构设计元模型

元模型是对各种体系结构设计模型的抽象。使用这个模型对当前的各种体系结构设计方法进行分析和比较。各种体系结构设计方法都可以描述成元模型的实例，每种方法在过程的顺序上、在概念的特定内容上有所不同。

2.2 体系结构设计元模型



体系结构设计的元模型

2.3 建模方法

为了获取对体系结构设计的抽象，人们已经提出了许多方法。我们把这些体系结构设计方法分类为：

- 工件驱动(Artifact-Driver)的方法
- 用例驱动(Use-Case-Driven)的方法
- 模式驱动(Pattern-Driven)的方法
- 领域驱动(Domain-Driven)的方法

下面所介绍的每一种方法都可视为元模型的一种实现。

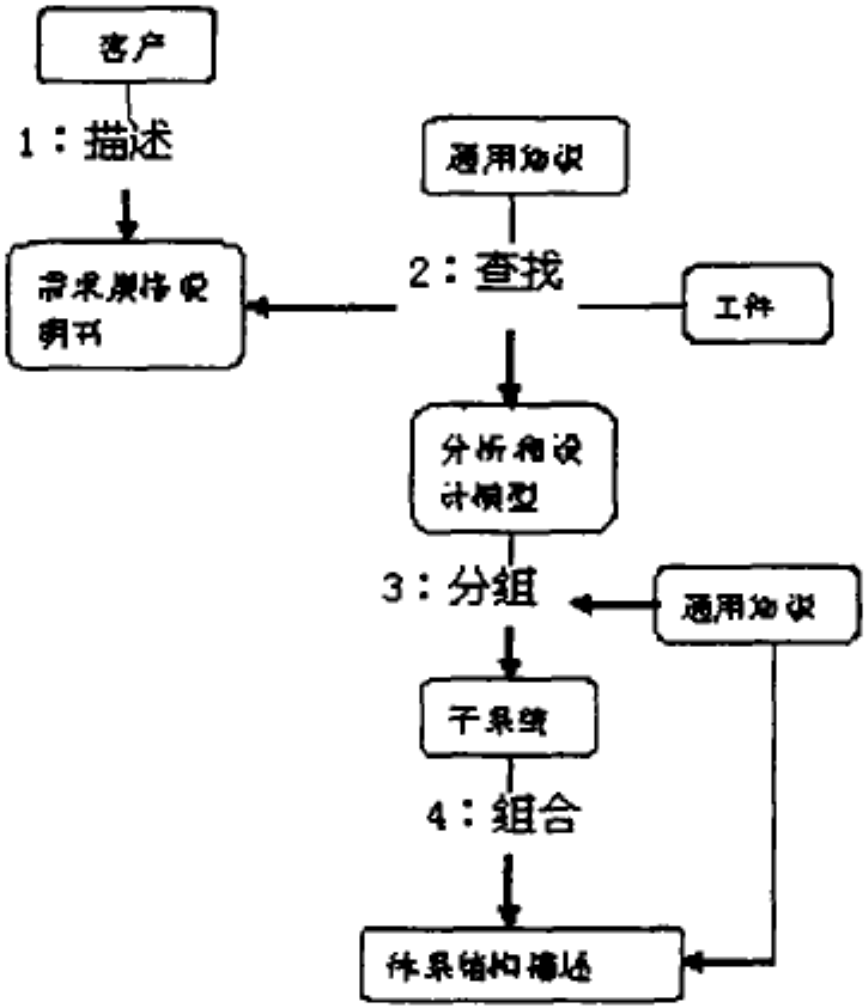
2.3 建模方法

- 工件驱动(Artifact-Driver)的方法

  工件驱动的体系结构设计方法从工件描述中提取体系结构描述。工件驱动的体系结构设计方法的例子包括广为流行的面向对象分析和设计方法OMT(Object Modeling Technology)和OAD(Object orient Analysis & Design)。

2.3 建模方法

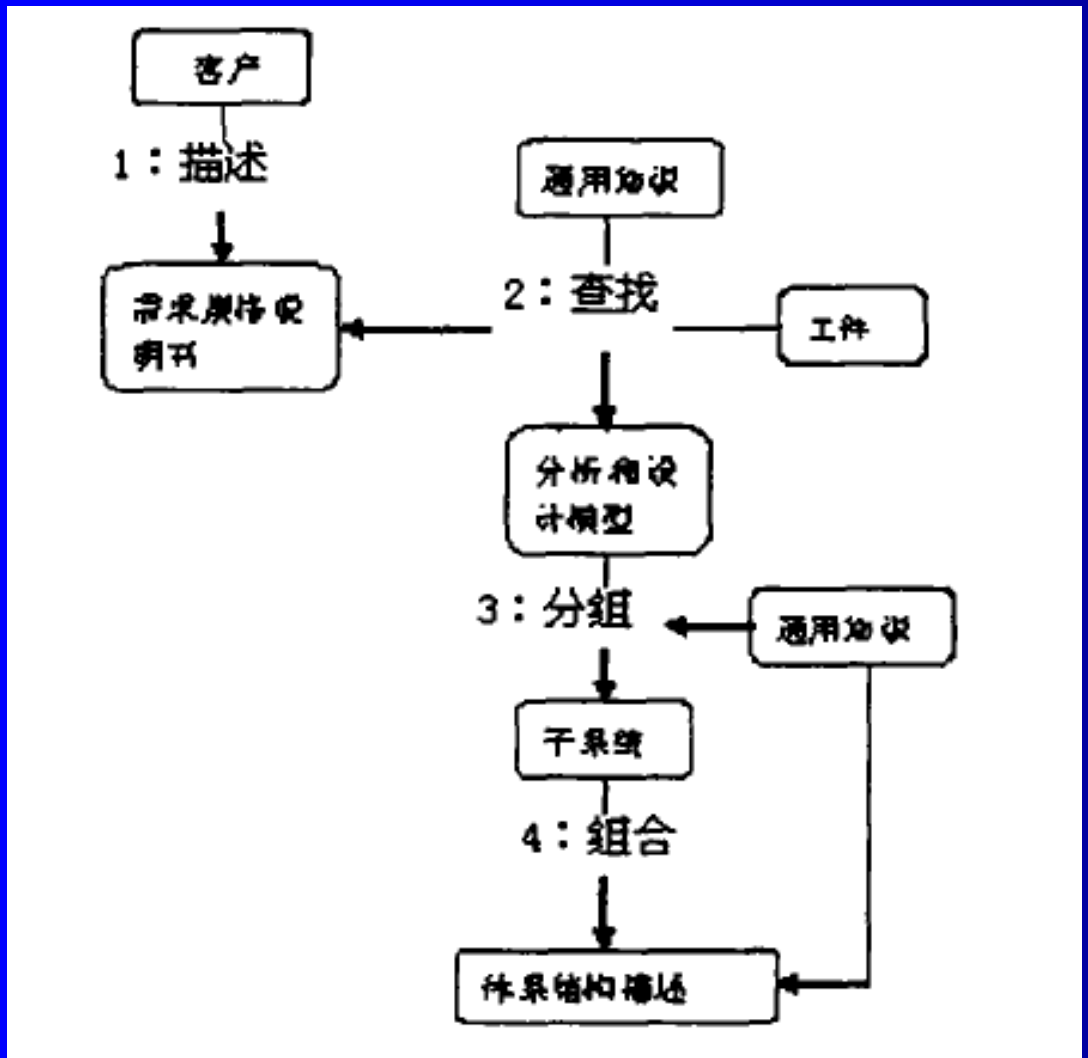
- 工件驱动(Artifact-Driver)的方法



加以标号的箭头表示体系结构设计步骤的过程顺序：“分析和设计模型”和“子系统”的概念共同表示了元模型中的“解决方案抽象”概念；“通用知识”概念表示了元模型中“领域知识”概念的特殊化。

2.3 建模方法

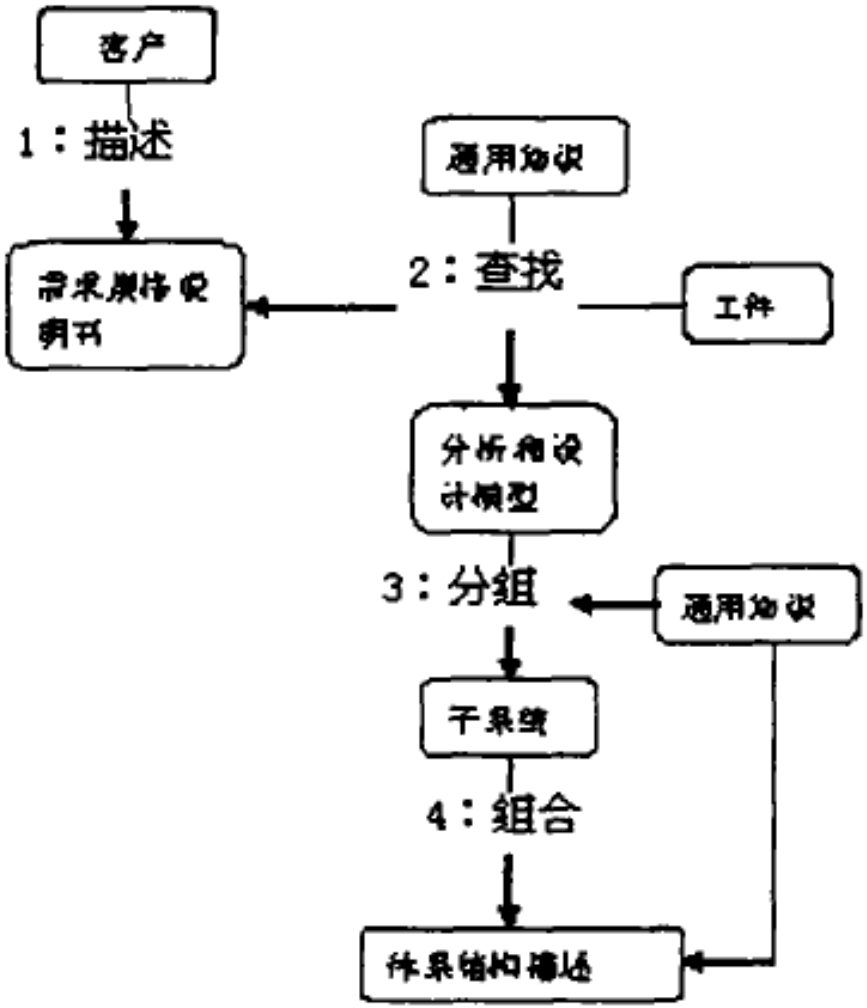
- 工件驱动(Artifact-Driver)的方法



用OMT解释这一模型，可以把OMT认为是这一策略的适当代表。在OMT中，体系结构设计并不是软件开发过程中的一个明确阶段，而是设计阶段的一个隐含部分。

2.3 建模方法

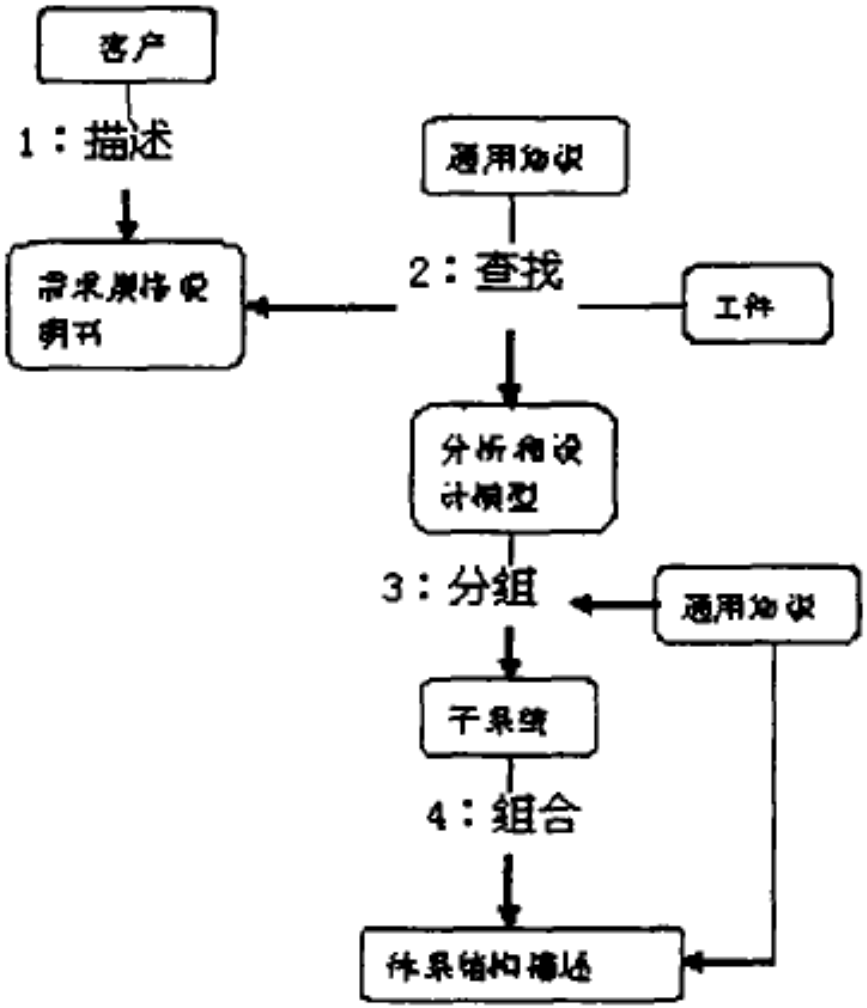
- 工件驱动(Artifact-Driver)的方法



OMT方法主要由以下阶段组成：分析、系统设计、对象设计。箭头线“1：描述”表示需求规格说明书的描述；箭头线“2：查找”表示对工件的查找，如系统分析阶段中需求规格说明的类。

2.3 建模方法

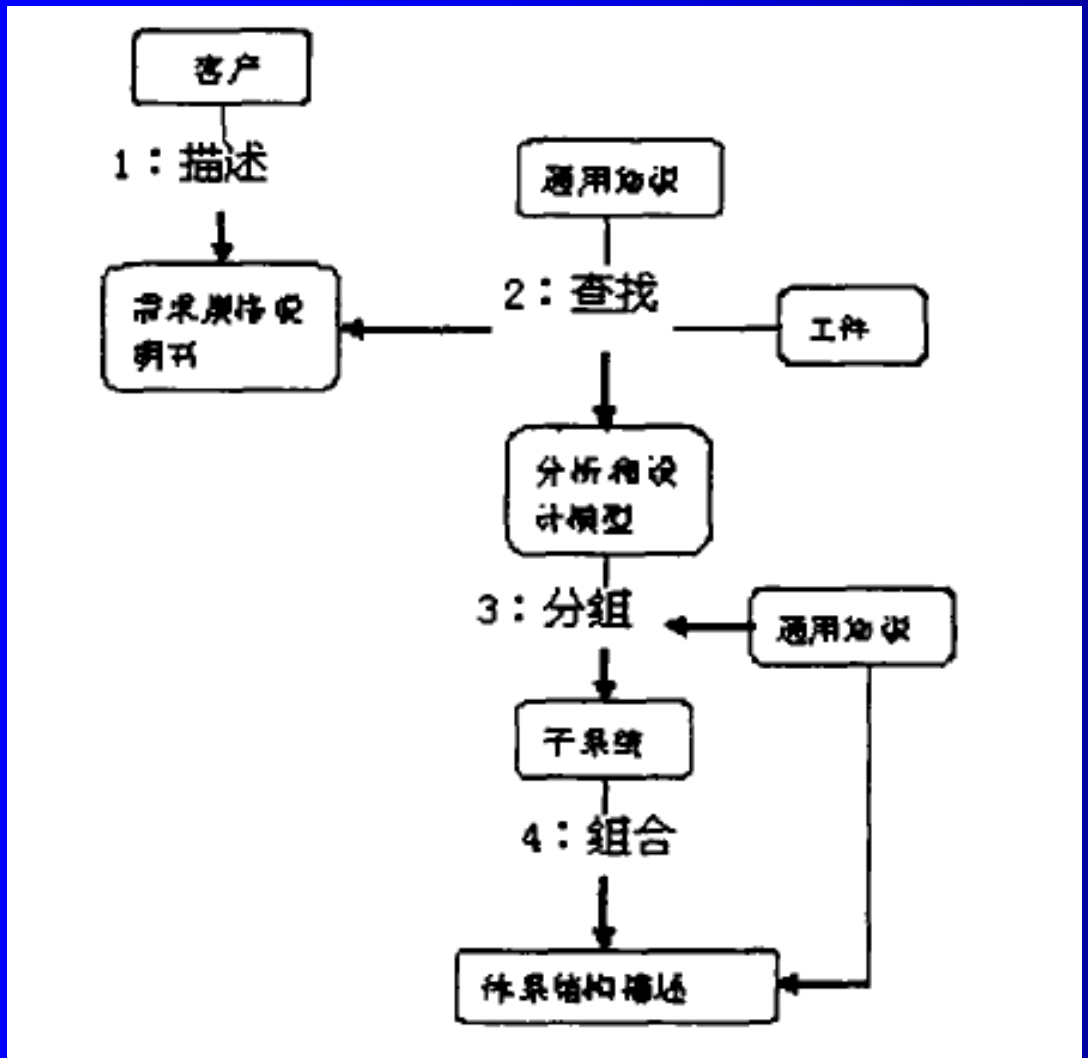
- 工件驱动(Artifact-Driver)的方法



查找过程得到了软件工程师的通用知识的支持，也得到了构成工件的启发式规则的支持，这些工件是该方法的重要构成部分。“2：查找”的结果是一组工件实例，在元模型中用“分析和设计模型”的概念来表示。

2.3 建模方法

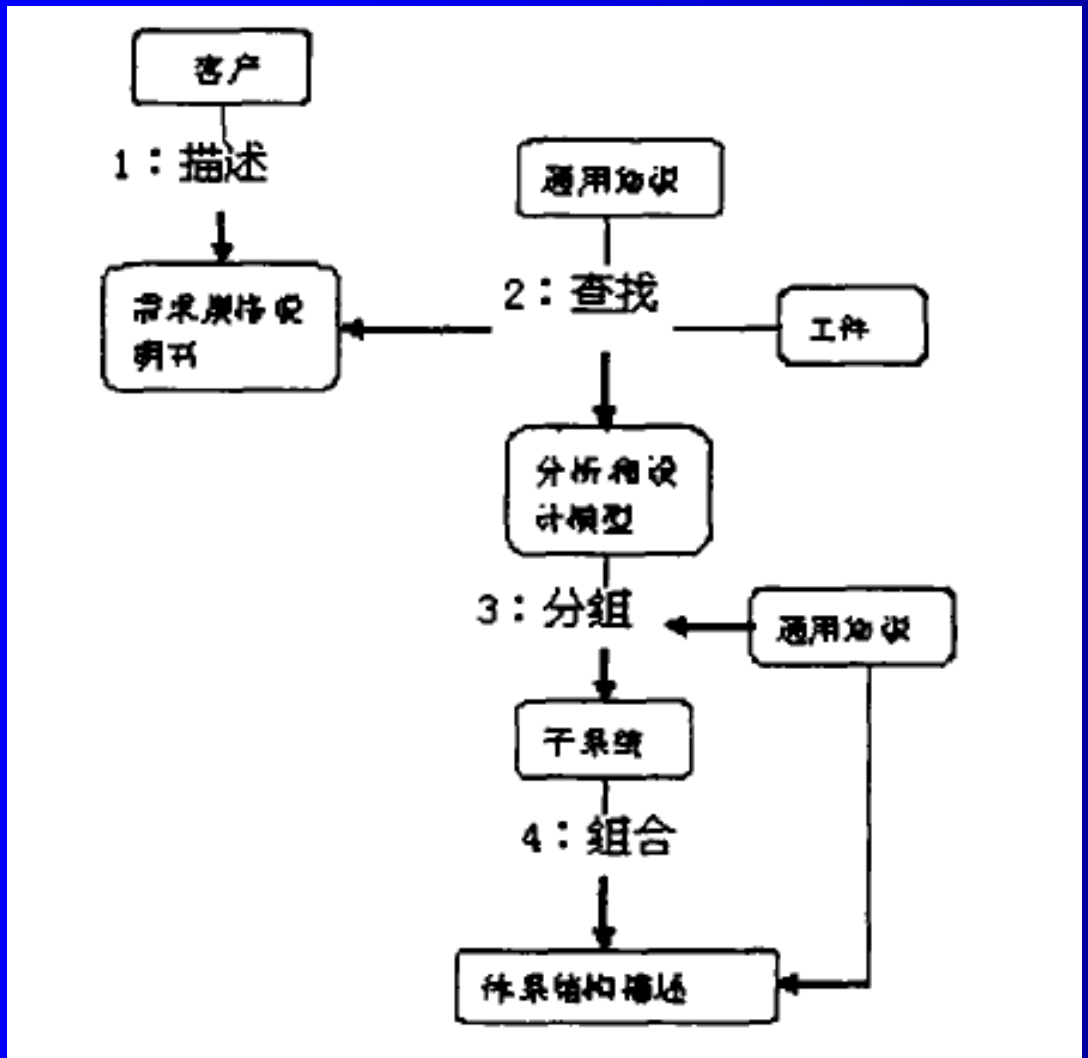
- 工件驱动(Artifact-Driver)的方法



在OMT方法中，接下来是系统设计阶段。该阶段将工件分组为子系统，为单个软件系统的全局结构的开发定义整体体系结构。这一功能被表示为“3：分组”。

2.3 建模方法

- 工件驱动(Artifact-Driver)的方法



软件体系结构由子系统组合而成，被表示成“4：组合”。这一功能也用到了“通用知识”概念提供的支持。

2.3 建模方法

- 工件驱动(Artifact-Driver)的方法

在体系结构开发方面，该方法存在着以下问题：

(l)文本形式的系统需求含混不清，不够精确、不够完整，因此，将它作为体系结构抽象的来源作用是不够的。

(2)子系统的语义过于简单，难以作为体系结构构件。

(3)对子系统的组合支持不足 。

2.3 建模方法

- 用例驱动(Use-Case-Driven)的方法

用例驱动的体系结构设计方法主要从用例导出体系结构抽象。

一个用例，是指系统进行的一个活动系列，它为参与者提供一些结果值。参与者通过用例使用系统。参与者和用例共同构成了用例模型。用例模型的目的是作为系统预期功能及其环境的模型，并在客户和开发者之间起到合约的作用。

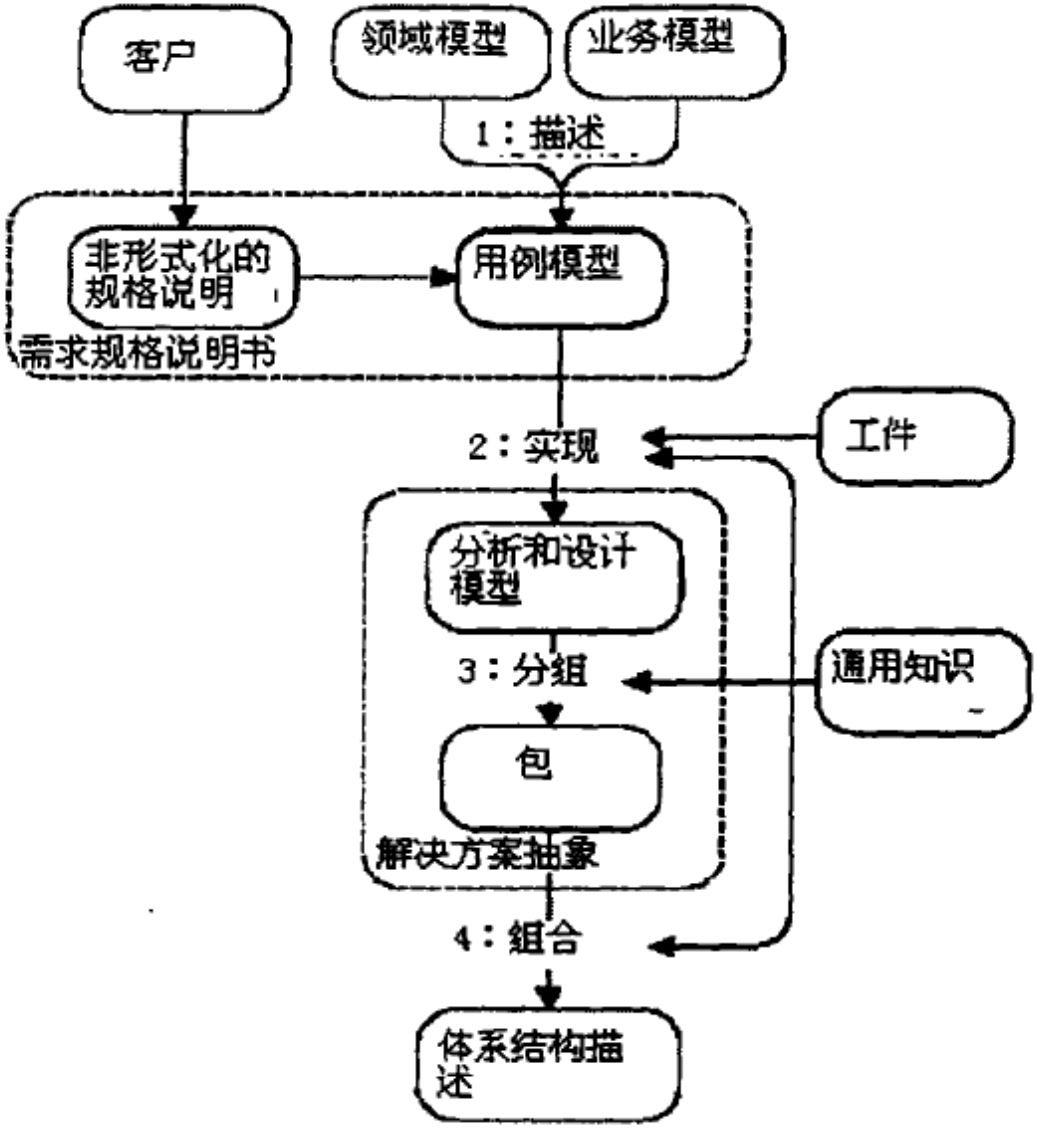
2.3 建模方法

- 用例驱动(Use-Case-Driven)的方法

统一过程使用的是一种用例驱动的体系结构设计方法。它由核心工作流(core workflows)组成。

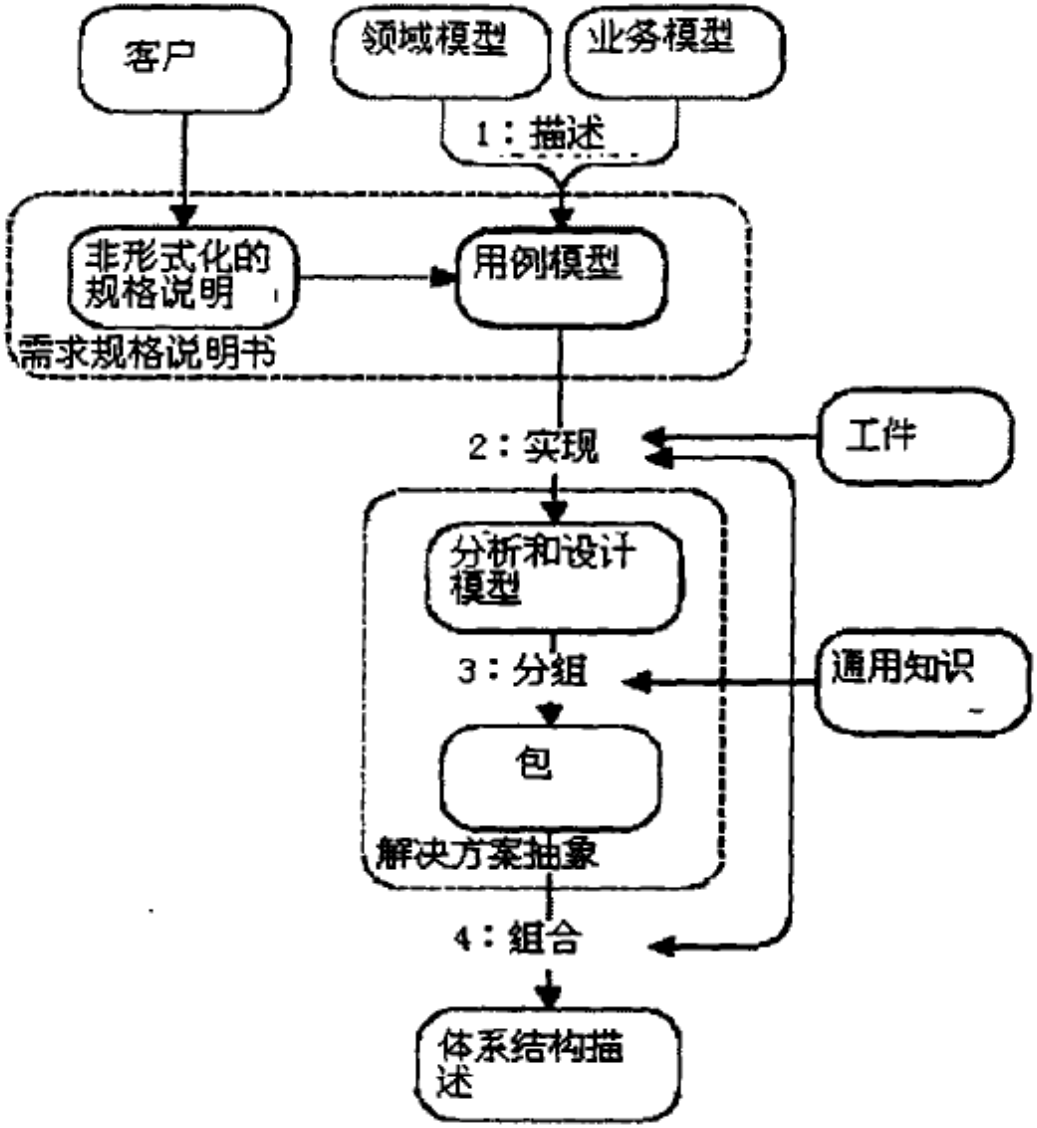
核心工作流定义了过程的静态内容，用活动、工人和工件描述了过程。随时间变化的过程的组织被定义为阶段。下图给出了用统一过程描述的用例驱动的体系结构设计方法的概念模型。

2.3 建模方法



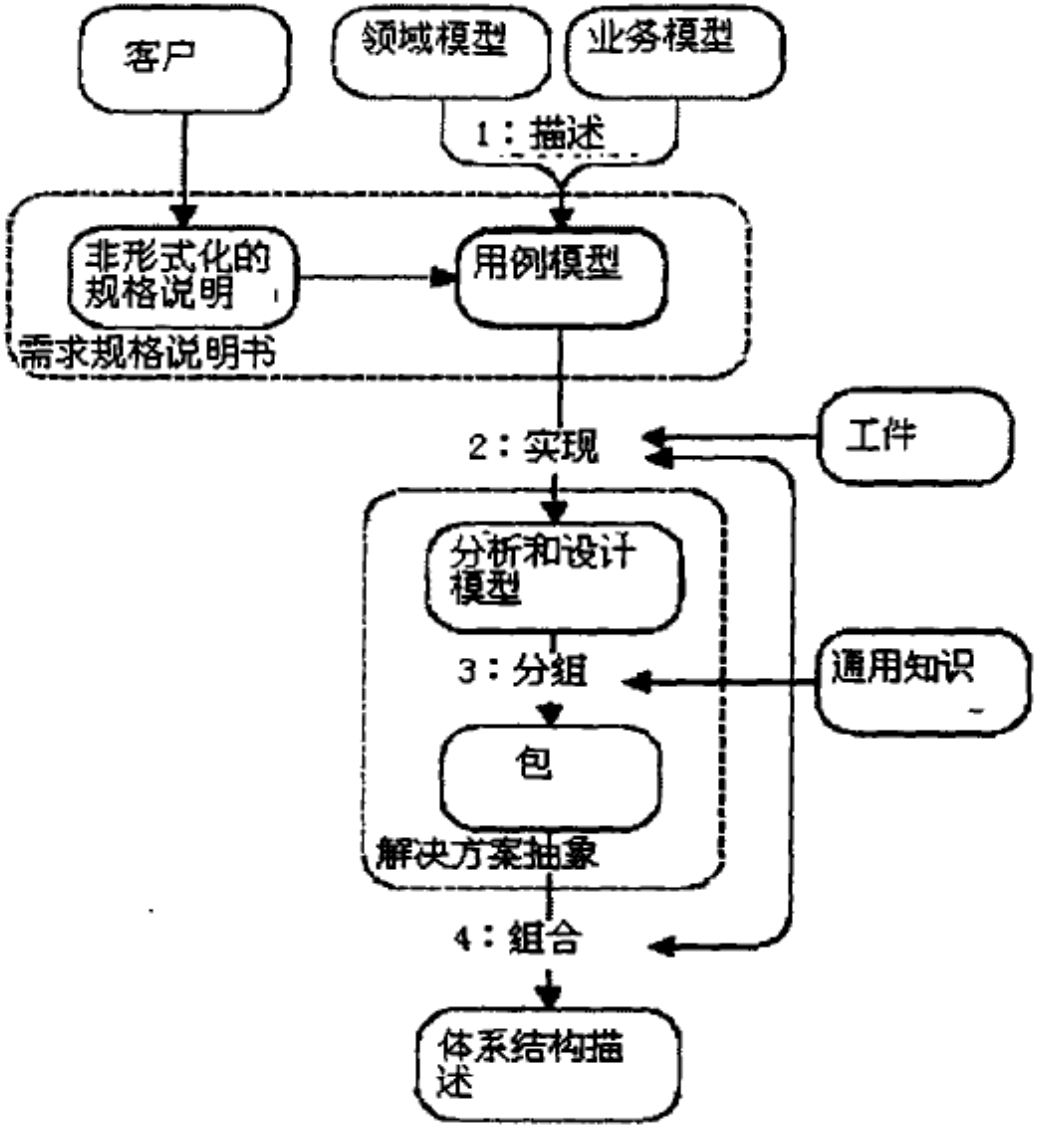
统一过程由六个核心工作流组成：商业模型、需求、分析、设计、实现和测试。这些核心工作流的结果分别是下列模型：商业和领域模型、用例模型、分析模型、设计模型、实现模型和测试模型。

2.3 建模方法



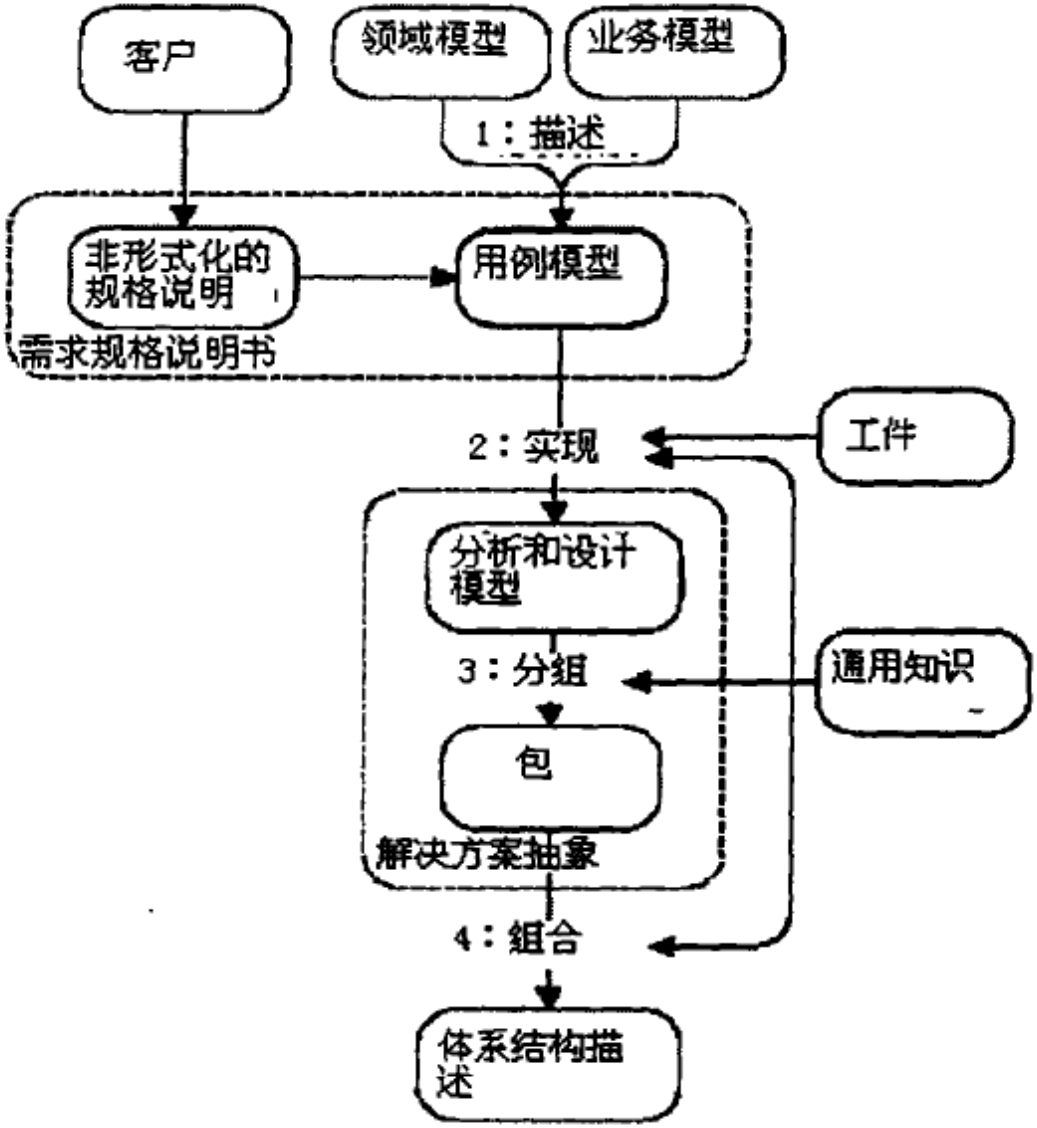
在需求工作流中，以用例的形式捕捉客户的需求，构成用例模型。这一过程在上图中被定义为“1：描述”。用例模型和非形式化的需求规格说明共同构成了系统的需求规格说明。

2.3 建模方法



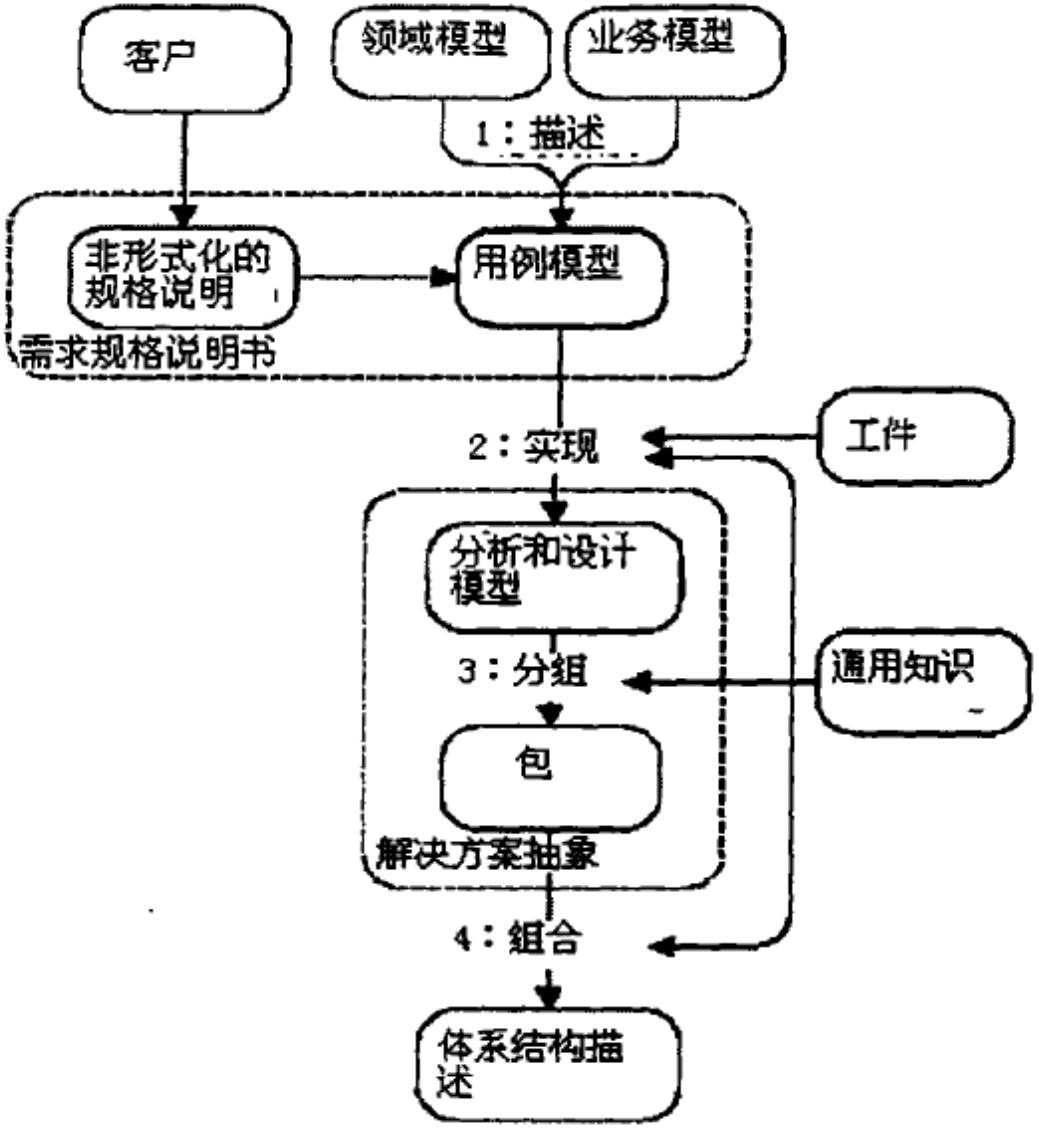
用例模型的开发得到了“非形式化的规格说明”、“域模型”、“商业模型”等概念的支持，在设置系统的上下文时这些概念是必需的。如前所述，“非形式化的规格说明”表示文本形式的需求规格说明。“商业模型”描述一个组织的商业过程。“域模型”描述上下文中最重要的类。

2.3 建模方法



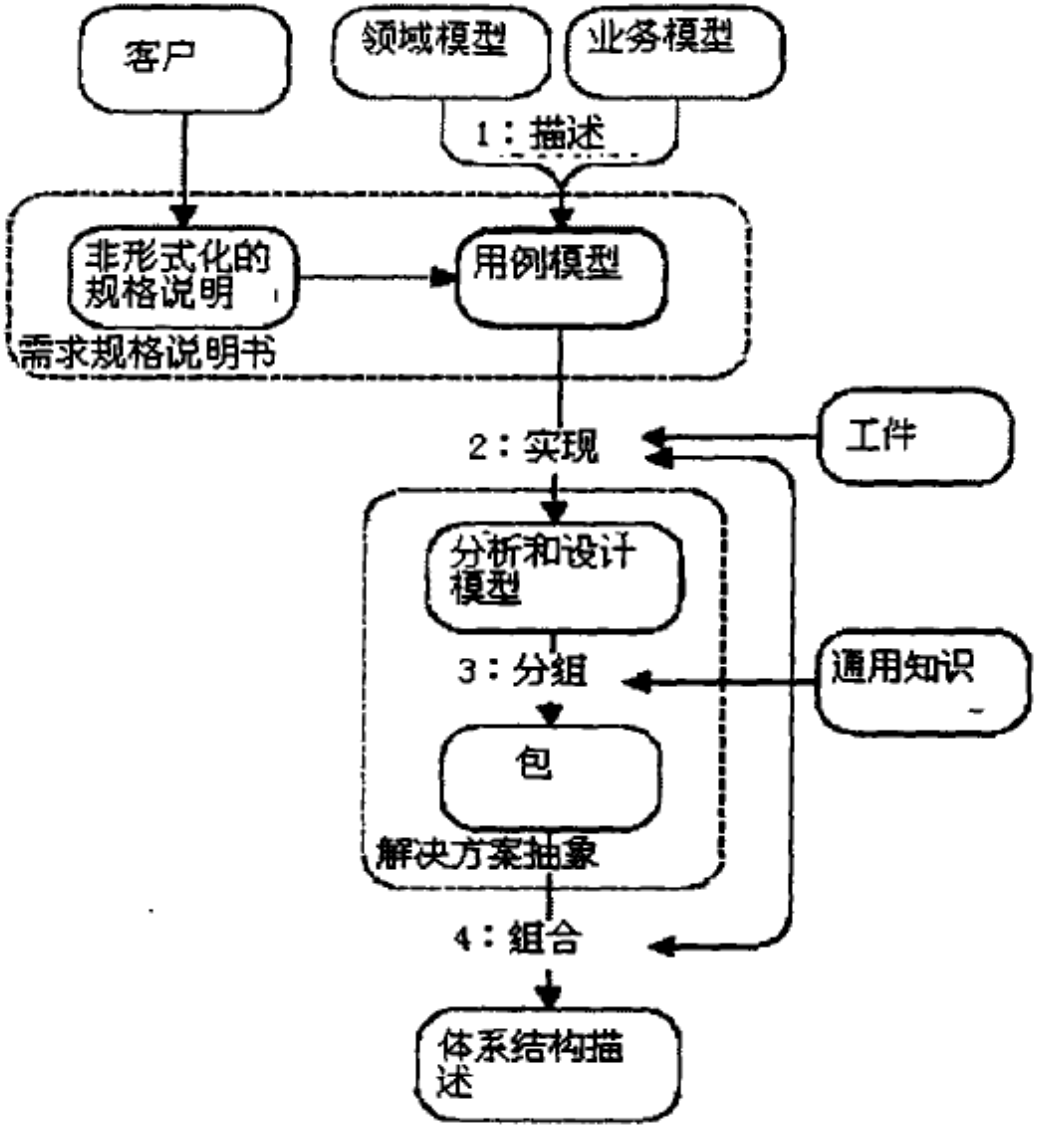
从用例模型可以选择出对于体系结构有重要意义的用例，并创建“用例实现”，如图中“2：实现”所述。用例实现决定了任务在系统内部是怎样进行的。

2.3 建模方法



用例实现受到相关工件的知识和通用知识的支持。这在图中被表示为分别从“工件”和“通用知识”引出的指向“2：实现”的箭头线。这一功能的输出是“分析和设计模型”概念，它表示在用例实现之后标识出的工件。

2.3 建模方法



然后，分析和设计模型被分组为包(packages)，这在图中被表示为“3：分组”。图中的“4：组合”代表定义这些包之间的接口，其结果是“体系结构描述”的概念。“3：分组”和“4：组合”这两个功能受到“通用知识”概念的支持。

2.3 建模方法

- 用例驱动(Use-Case-Driven)的方法

  在统一过程中，为了理解上下文，首先要开发商业模型和域模型。然后，主要从非形式化的规格说明、商业模型和域模型中导出用例模型。从用例模型中选择用例实现，导出体系结构抽象。

2.3 建模方法

- 用例驱动(Use-Case-Driven)的方法

在使用这一方法标识体系结构抽象时，必须处理下面几个问题。

(l)难以适度把握域模型和商业模型的细节

(2)对于如何选择与体系结构相关的用例没有提供系统的支持

(3)用例没有为体系结构抽象提供坚实的基础

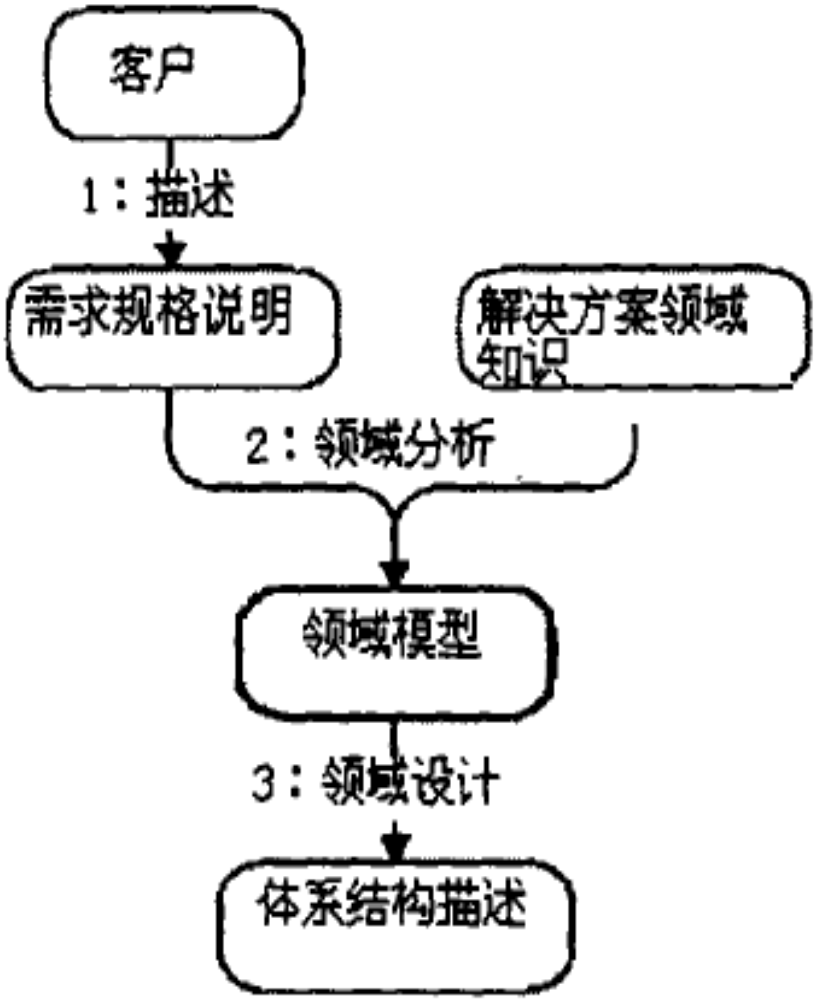
(4)包的语义过于简单，难以作为体系结构构件

2.3 建模方法

- 领域驱动(Domain-Driven)的方法

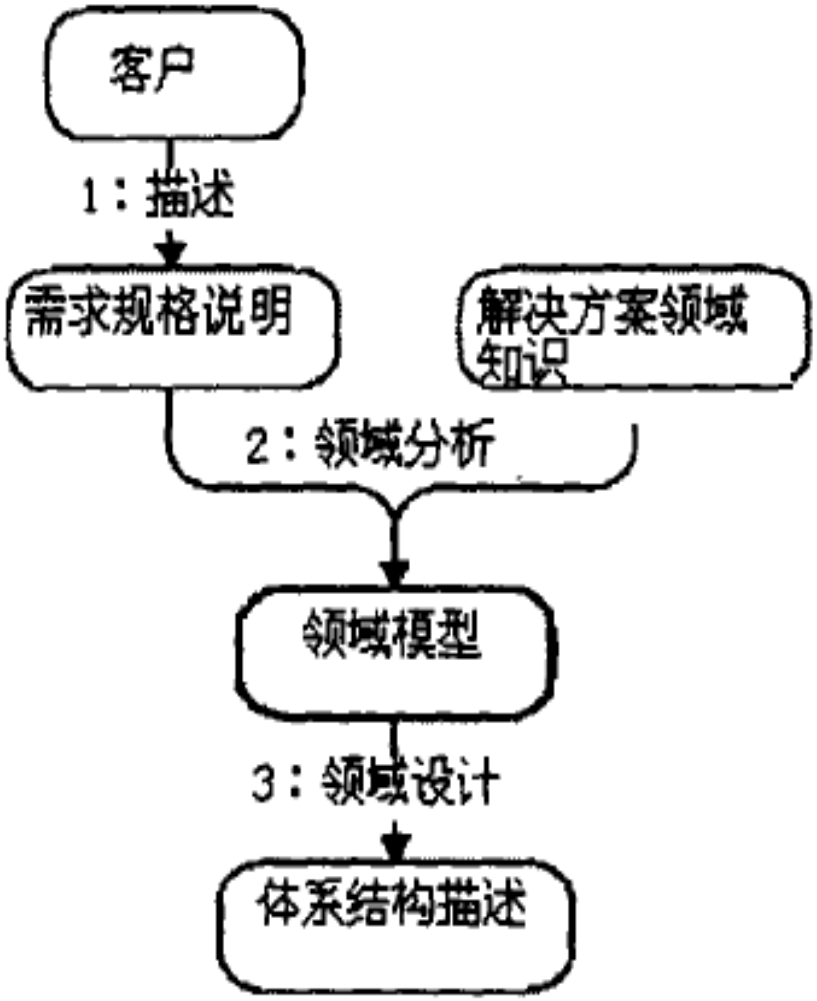
  在领域驱动的体系结构设计方法中，体系结构抽象是从领域模型导出的。这一方法的概念模型如下图所示。

2.3 建模方法



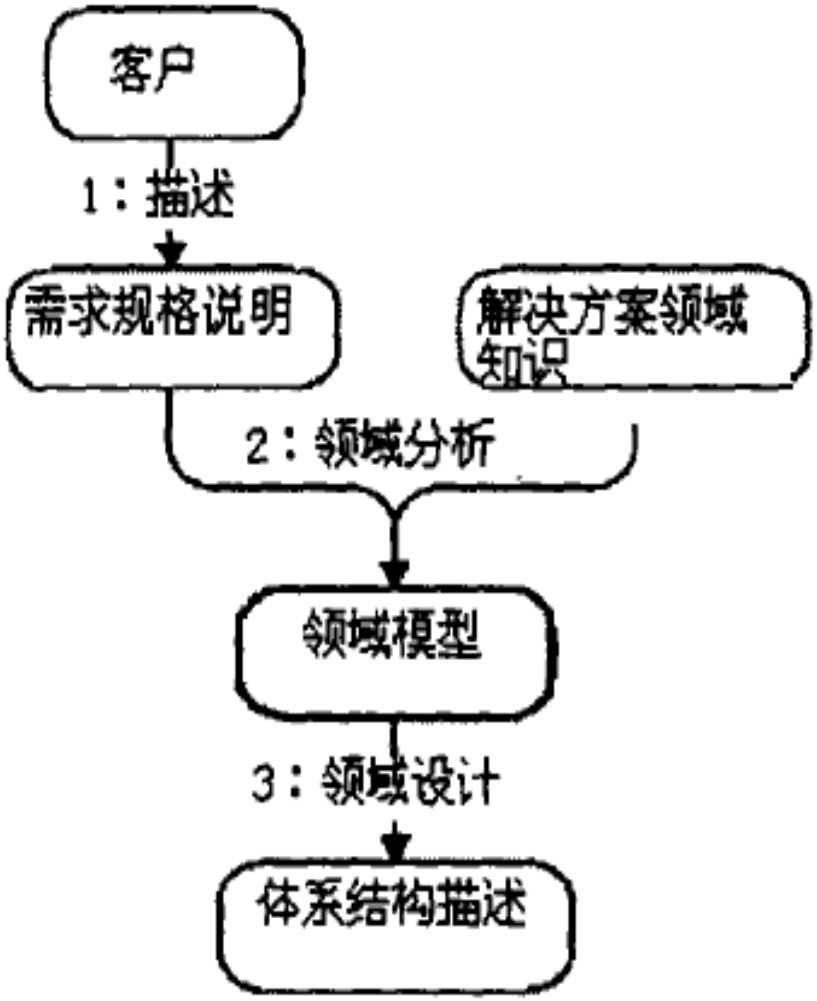
领域模型是在领域分析阶段开发的，这在图中被表示为“2：领域分析”。领域分析可以被定义成一个重用为目标的，确认、捕捉和组织问题领域的领域知识的过程。

2.3 建模方法



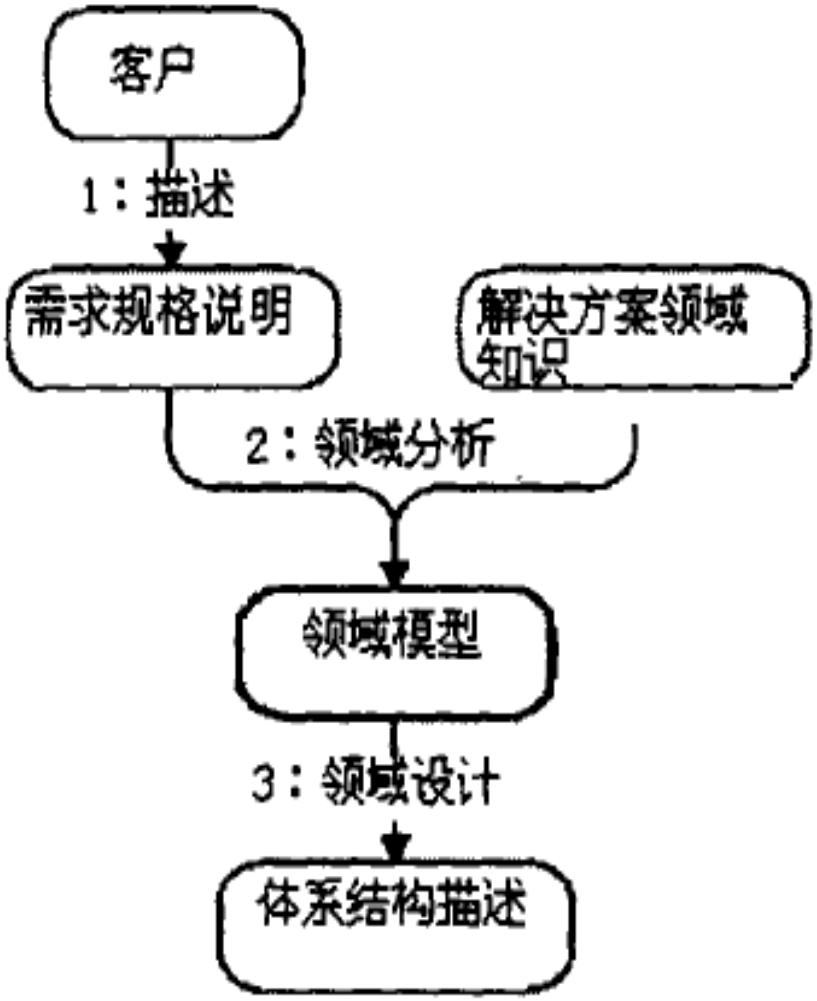
图中“2：领域分析”以“需求规格说明”和“解决方案领域知识”的概念为前提，产生“领域模型”的概念作为结果。需要注意的是，图中的“解决方案领域知识”和“领域模型”都属于元模型的“领域知识”的概念。

2.3 建模方法



领域模型可以有多种不同的表示方法，比如类、实体关系图、框架、语义网络、规则等。与此相应，领域分析的方法也有多种。

2.3 建模方法



在这里，主要关注用领域模型导出体系结构抽象的方法。在图中，这被表示为“3：领域设计”。下面，考虑两种领域驱动的方法，它们从领域模型到体系结构设计抽象。

2.3 建模方法

- 领域驱动(Domain-Driven)的方法

(l)产品线体系结构设计

在产品线体系结构设计方法中，软件体系结构是为一个软件产品线而开发的。

可以这样定义软件产品线：它是一组以软件为主的产品，具有多种共同的、受控的特点，能够满足特定市场或任务领域的需求。

2.3 建模方法

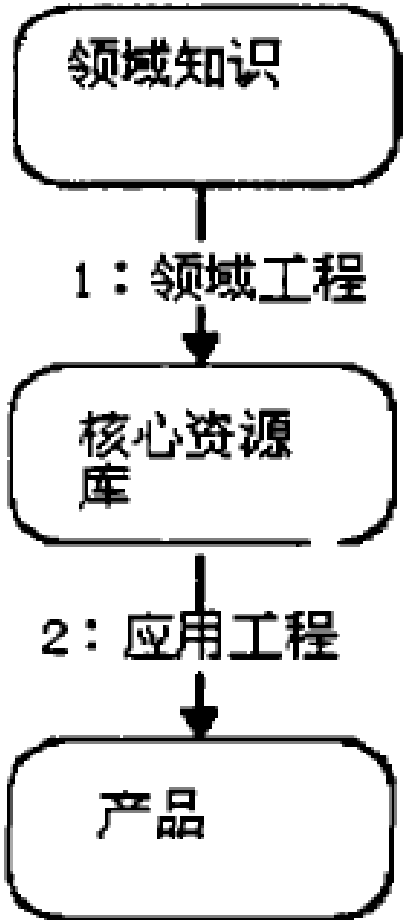
- 领域驱动(Domain-Driven)的方法

(l)产品线体系结构设计

软件产品线体系结构是对一组相关产品的体系结构的抽象。产品线体系结构设计方法主要关注组织内部的重用，基本是由核心资源开发和产品开发这两个部分组成的。核心资源库通常包括体系结构、可重用软件构件、需求、文档和规格说明，性能模型，方案，预算，以及测试计划和用例。核心资源库用于从产品线生成或集成产品。

2.3 建模方法

- 领域驱动(Domain-Driven)的方法



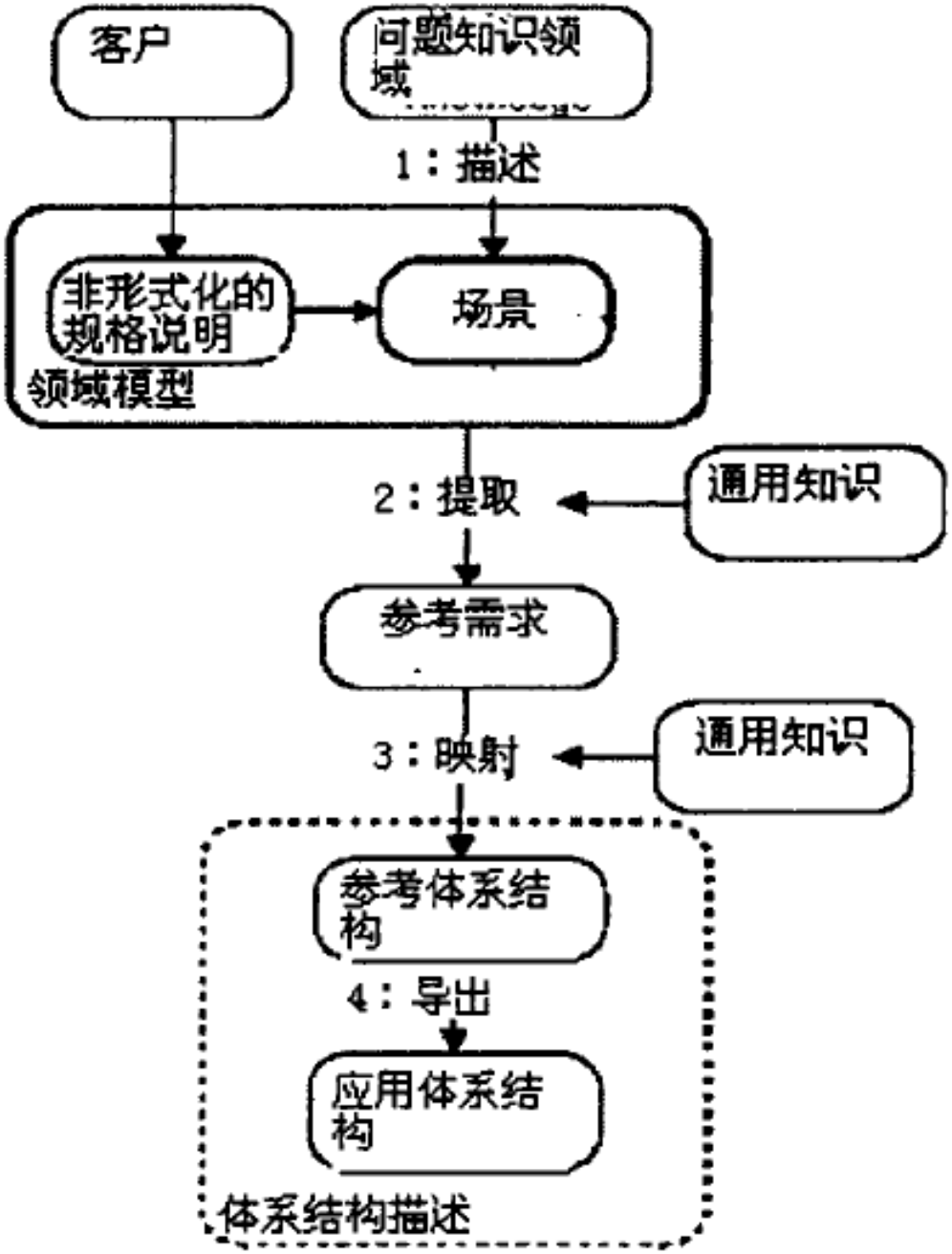
产品线体系结构设计的概念模型如下图所示。其中，“1：领域工程”表示核心资源库的开发，“2：应用工程”表示从核心资源库开发产品。

2.3 建模方法

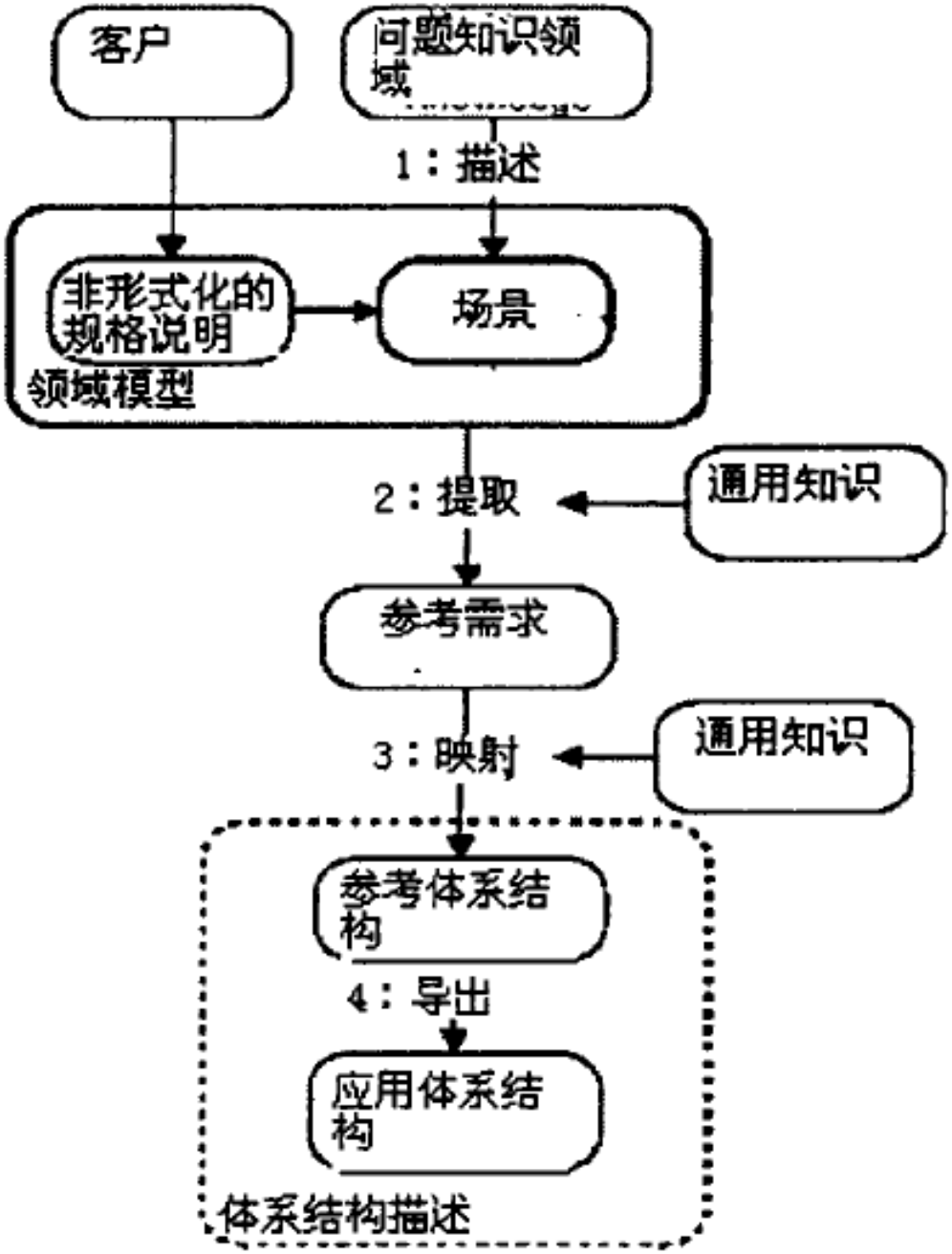
- 领域驱动(Domain-Driven)的方法

(2)特定领域的软件体系结构设计

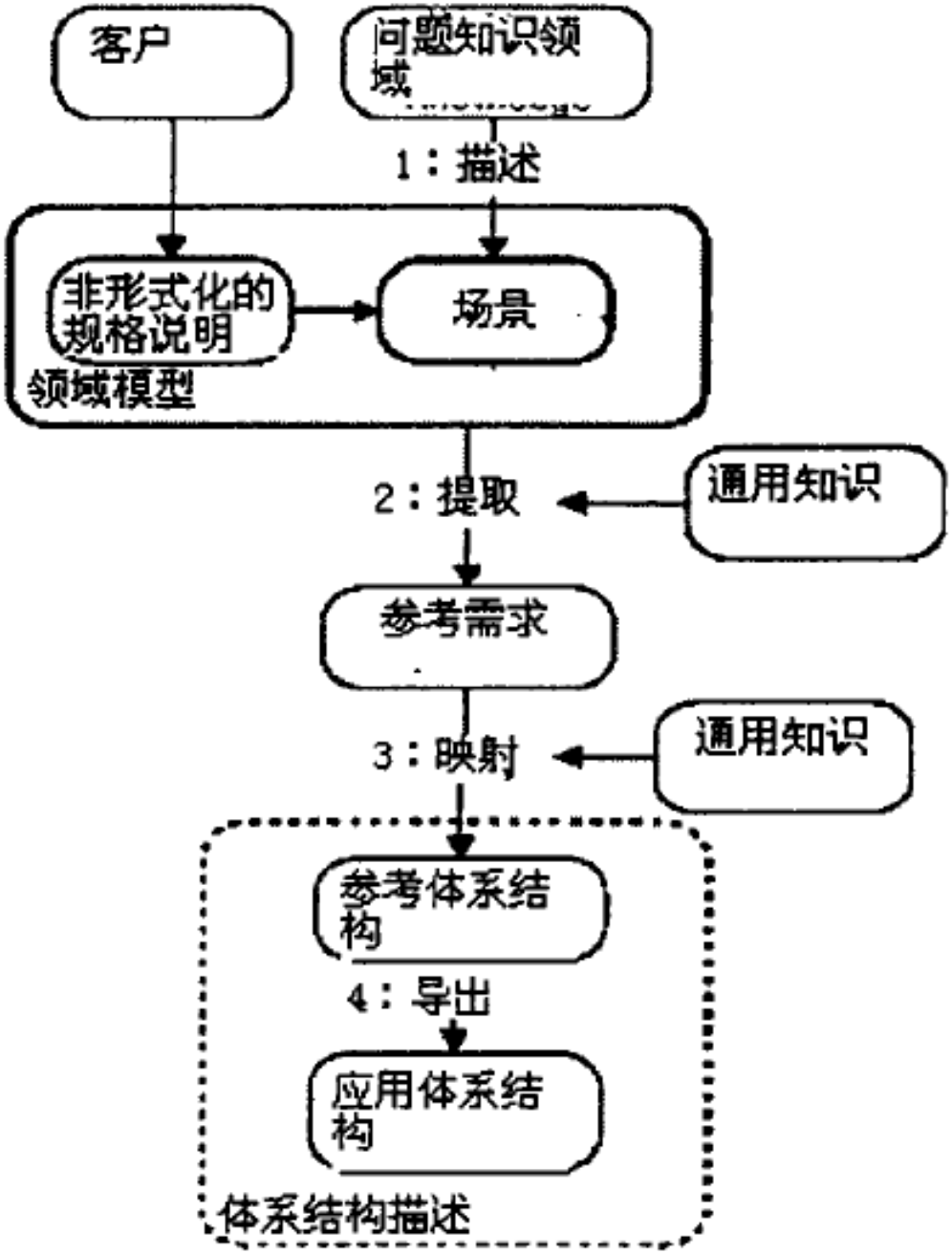
可以把特定领域的软件体系结构看成是多系统范围内的体系结构，即它是从一组系统中导出的，而不是某一单独的系统。下图表示了DSSA方法的概念模型。



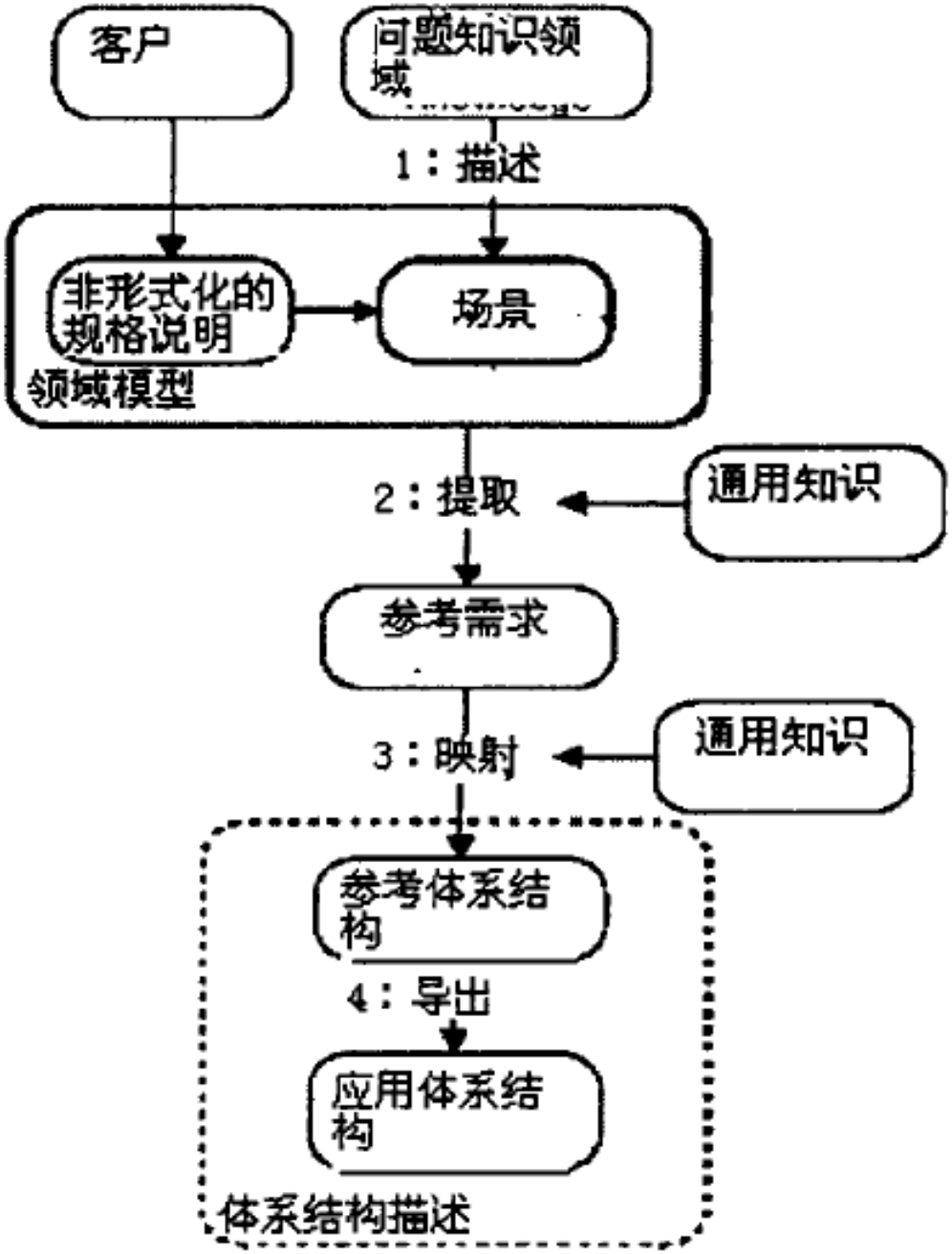
DSSA方法的基本工件是领域模型、参考需求和参考体系结构。DSSA方法从领域分析阶段开始，面向一组有共同问题或功能的应用程序。



这种分析以场景为基础，从中导出功能需求、数据流和控制流等信息。



领域模型包括场景、领域字典、上下文图、实体关系图、数据流模型、状态转换图以及对象图。



从图中可以看出，除了领域模型之外，还定义了参考需求，它包括功能需求、非功能需求、设计需求、实现需求，而且它主要关注解决方案空间。领域模型和参考需求被用于导出参考体系结构。

2.3 建模方法

- 领域驱动(Domain-Driven)的方法

(2)特定领域的软件体系结构设计

DSSA过程明确区别参考体系结构和应用体系结构。

参考体系结构被定义为用于一个应用系统族的体系结构，应用体系结构被定义为用于一个单一应用系统的体系结构。

应用体系结构是从参考体系结构实例化或求精而来的。实例化/求精的过程和对参考体系结构进行扩展的过程被称为应用工程。

2.3 建模方法

领域驱动(Domain-Driven)的方法

由于对“领域”这一术语有着不同的解释，领域驱动的体系结构设计方法也有多种，其中存在的一些问题如下。

(1)问题领域分析在导出体系结构抽象方面效果较差;

(2)解决方案领域分析不够充分。

2.3 建模方法

- 模式驱动(Pattern-Driven)的方法

软件工业界已经广泛接受了软件设计模式的概念。软件设计模式的目的在于编制一套可重用的基本原则，用于开发高质量的软件系统。软件设计模式常常用在设计阶段，但是，人们已经开始在软件开发过程中的其他阶段定义并使用设计模式。比如，在实现阶段，可以定义从面向对象的设计到面向对象的语言构造的设计模式；在分析阶段，可以使用设计模式导出分析模型。

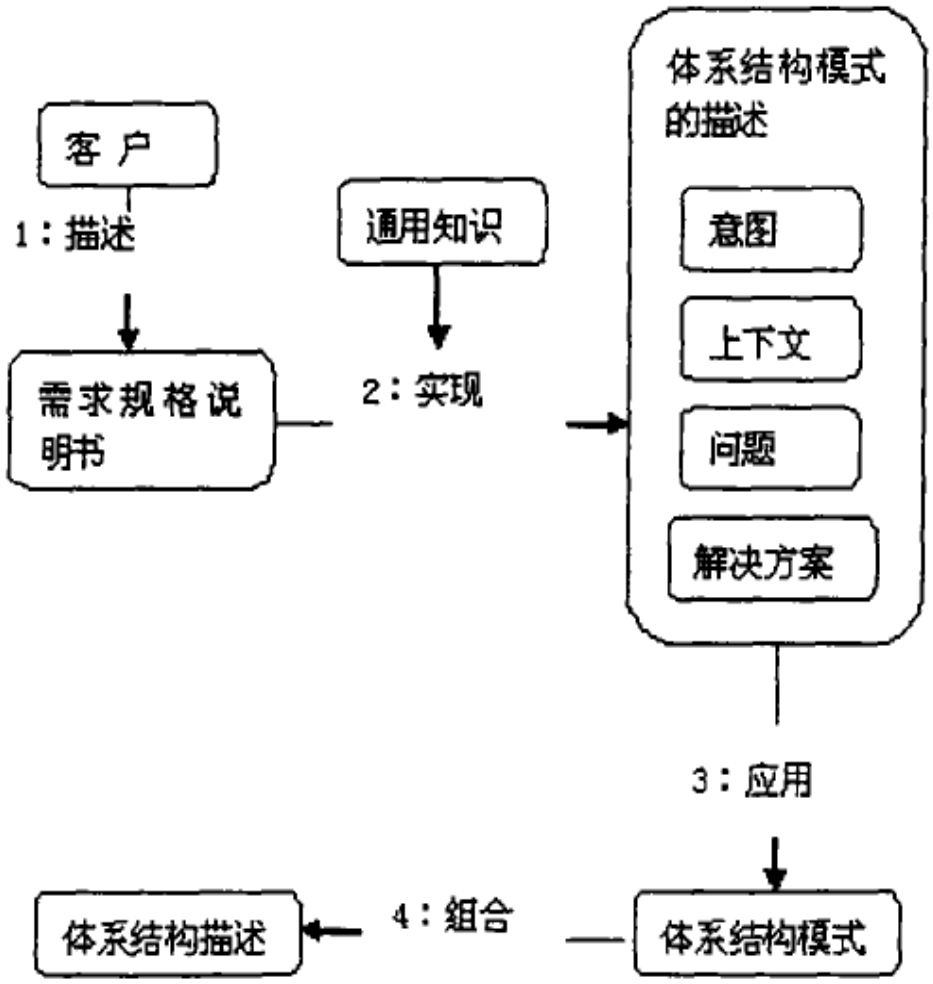
2.3 建模方法

- 模式驱动(Pattern-Driven)的方法

近年来，也有研究者在软件开发过程中的体系结构分析阶段应用设计模式。体系结构模式类似于设计模式，但它关心的是更粗粒度的系统结构及其交互。实际上，它也就是体系结构风格的另一种名称。体系结构设计模式是体系结构层次的一种抽象表示。

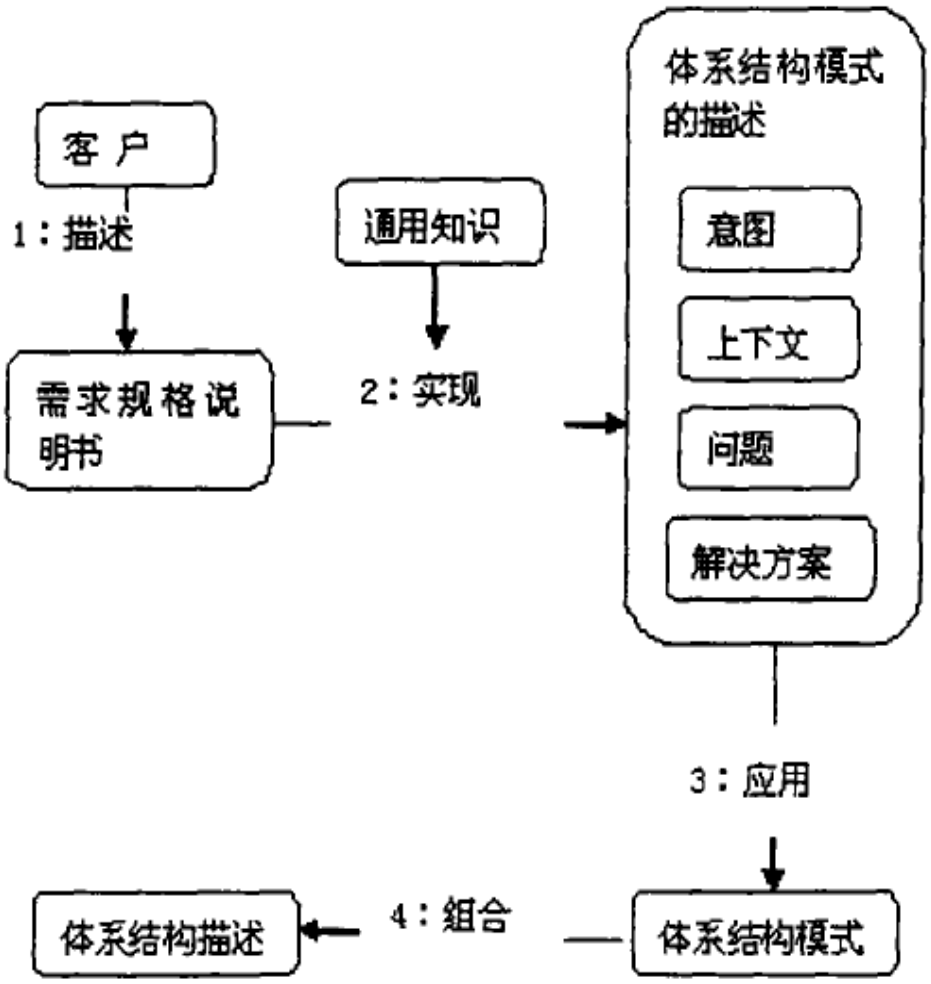
模式驱动的体系结构设计方法从模式导出体系结构抽象。下图描述了这一方法的概念模型。

2.3 建模方法



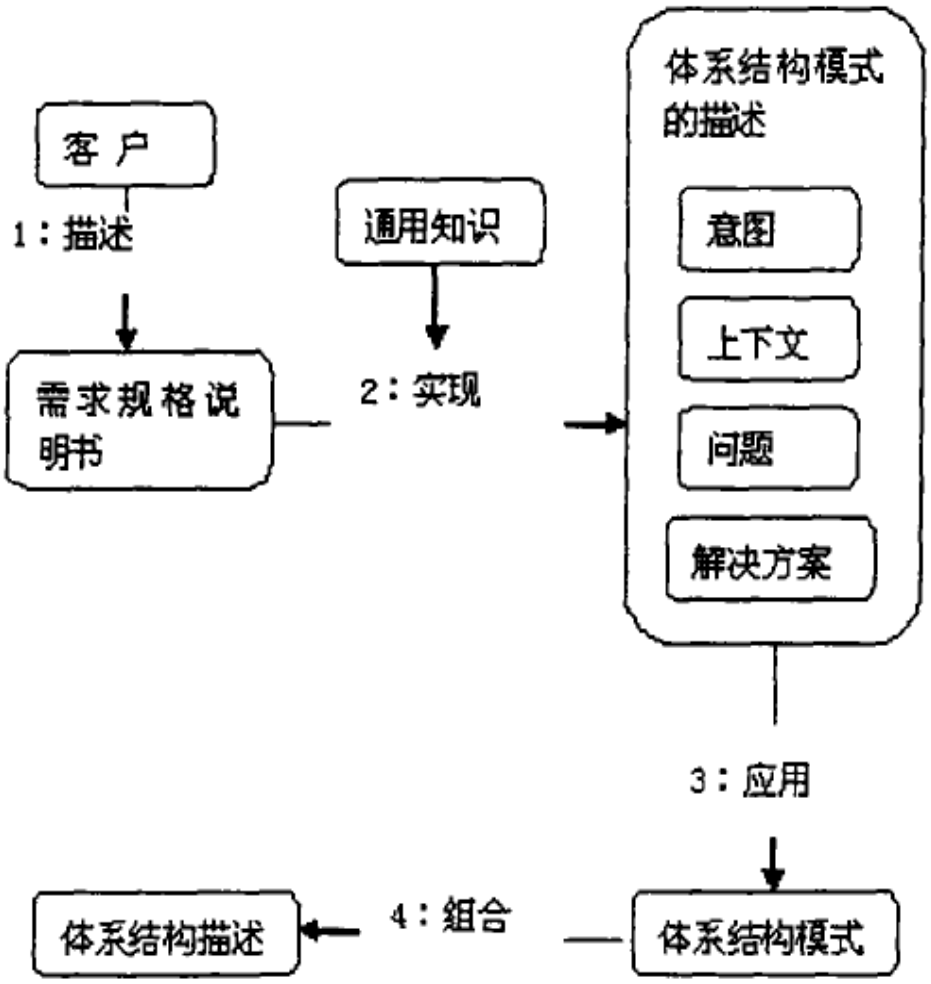
“需求规格说明”的概念表示对问题的规格说明，该问题可能通过模式得以解决。图中的“实现”表示了为给出的问题描述查找适当的模式的过程，它受到“通用知识”概念的支持。

2.3 建模方法



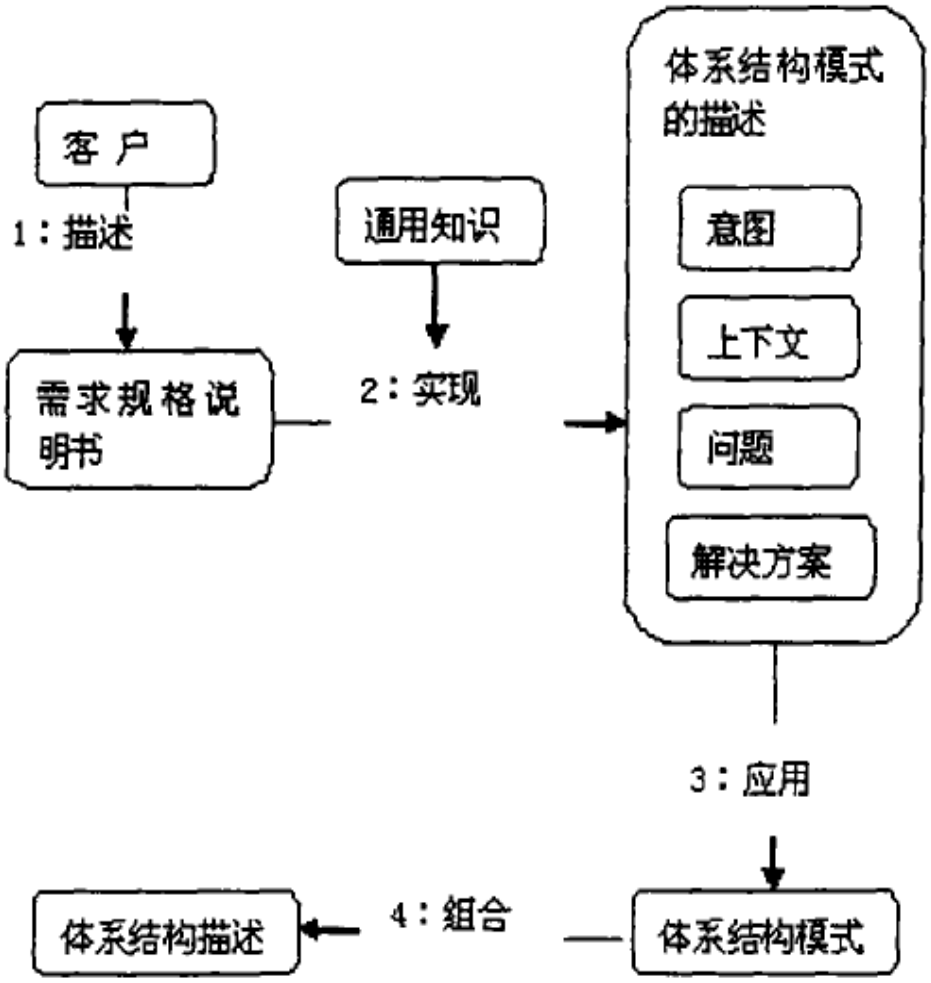
“体系结构模式描述”的概念指的是对体系结构模式的描述。它主要由4个概念组成：意图、上下文、问题和解决方案。

2.3 建模方法



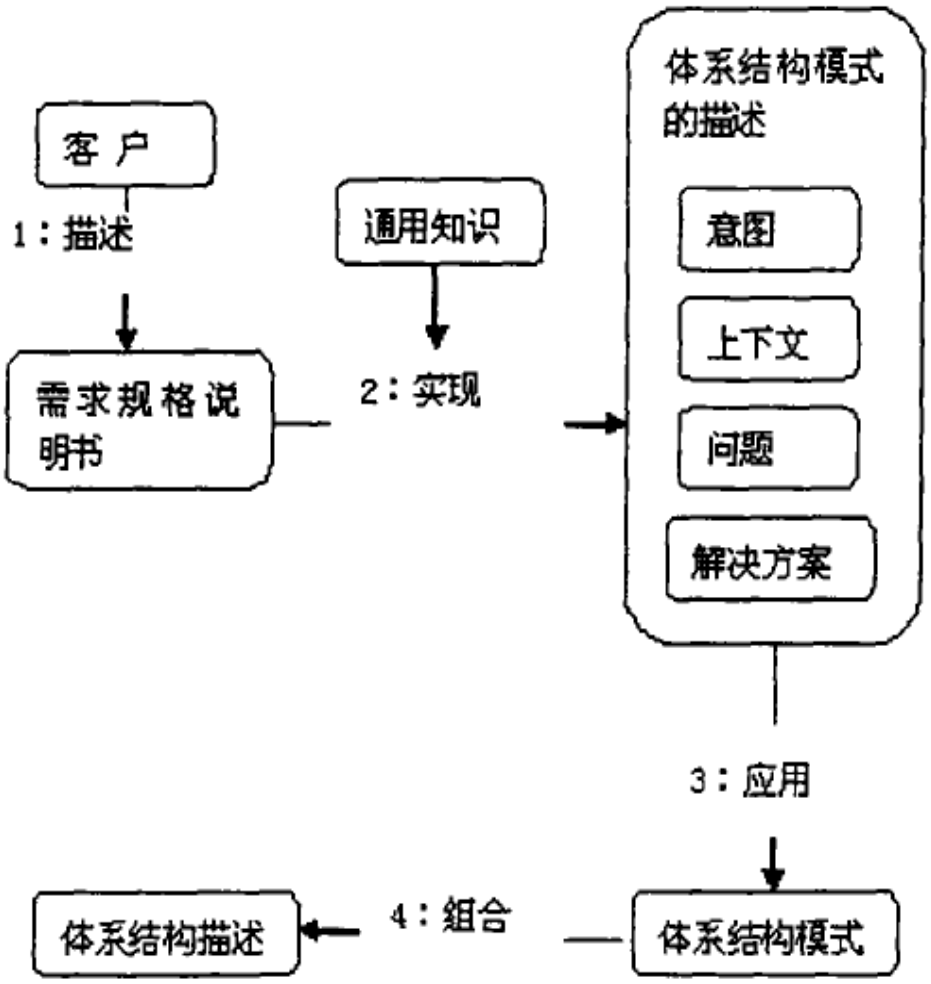
意图(intent)表示使用模式的基本原则；上下文(context)表示问题的产生环境；问题表示上下文环境中经常出现的问题；解决方案是以元素及其关系的抽象描述的形式来表示对问题的解决方案。

2.3 建模方法



为了确认模式，要对各个可用模式的意图进行扫描。如果发现一个模式的意图和给出的问题相关，那么就分析它的上下文描述。这时，如果上下文描述仍然能够和给出的问题相匹配，则处理过程进入图中的“3：应用”。

2.3 建模方法



进而，用解决方案这一子概念来提供所给出问题的解决方案。概念“体系结构模式”表示了“3：应用”的结果。最后，“4：组合”表示在导出体系结构描述时，体系结构模式之间的相互协作。

2.3 建模方法

- 模式驱动(Pattern-Driven)的方法

在许多体系结构设计方法中，都包括作为一个子过程的模式驱动的体系结构设计方法。尽管体系结构模式在构建软件体系结构时能够起到一定的作用，但是在选择模式、应用，以及把模式组成体系结构等问题上，当前的方法并不能提供足够的支持。下面更详细地介绍了这些问题。

(1)在处理范围广泛的体系结构问题时，模式库可能不够充足

(2)对模式的选择仅依靠通用知识和软件工程师的经验

(3)模式的应用并不是一个简单直接的过程，它需要对问题进行全面的分析

(4)对于模式的组合没有提供很好的支持

2.4 软件体系结构的生命周期模型

软件体系结构生命周期模型是对软件体系结构在整个生存期间所需经历的所有阶段和步骤的描述，这种描述独立于具体的体系结构，使得体系结构的设计遵循一定的理论基础和工程原则。

一个软件体系结构生命周期模型主要由以下几个阶段组成：

2.4 软件体系结构的生命周期模型

(1)软件体系结构的非形式化描述
(2)软件体系结构的规范描述和分析
(3)软件体系结构的求精及其验证
(4)软件体系结构的实施
(5)软件体系结构的演化和扩展
(6)软件体系结构的提供、评价和度量
(7)软件体系结构的终结

2.4 软件体系结构的生命周期模型

(1)软件体系结构的非形式化描述(Software Architecture Informal Description)

一种软件体系结构在其产生时，其思想通常是简单的，并常常由软件设计师用非形式化的自然语言表示概念、原则。例如：客户机-服务器体系结构就是为适应分布式系统的要求，从主从式演变而来的一种软件体系结构。

尽管该阶段的描述常是用自然语言描述的，但是该阶段的工作却是创造性和开拓性的。

2.4 软件体系结构的生命周期模型

(2)软件体系结构的规范描述和分析(Software Architecture Specification and Analysis)

这一阶段通过运用合适的形式化数学理论模型对第1阶段的体系结构的非形式化描述进行规范定义，从而得到软件体系结构的形式化规范描述，以使软件体系结构的描述精确、无歧义，并进而分析软件体系结构的性质，如无死锁性、安全性、活性等。分析软件体系结构的性质有利于在系统设计时选择合适的软件体系结构，从而对软件体系结构的选择起指导作用，避免盲目选择。

2.4 软件体系结构的生命周期模型

(3)软件体系结构的求精及其验证(Software Architecture Refinement and Its Verification)

大型系统的软件体系结构总是通过从抽象到具体，逐步求精而达到的，因为一般说来，由于系统的复杂性，抽象是人们在处理复杂问题和对象时必不可少的思维方式，软件体系结构也不例外。但是过高的抽象却使软件体系结构难以真正在系统设计中实施。因而，如果软件体系结构的抽象粒度过大，就需要对体系结构进行求精、细化，直至能够在系统设计中实施为止。

2.4 软件体系结构的生命周期模型

(3)软件体系结构的求精及其验证(Software Architecture Refinement and Its Verification)

在软件体系结构的每一步求精过程中，需要对不同抽象层次的软件体系结构进行验证，以判断较具体的软件体系结构是否与较抽象的软件体系结构的语义一致，并能实现抽象的软件体系结构。我们这里并不排斥在不求精的情况下对软件体系结构的验证，而只是侧重于研究和讨论软件体系结构的求精及其验证。

2.4 软件体系结构的生命周期模型

(4)软件体系结构的实施(Software Architecture Enactment)

这一阶段将求精后的软件体系结构实施于系统的设计中，并将软件体系结构的构件和连接件等有机地组织在一起，形成系统设计的框架，以便据此实施于软件设计和构造中。

2.4 软件体系结构的生命周期模型

(5)软件体系结构的演化和扩展(Software Architecture Evolution and Extension)

在实施软件体系结构时，根据系统的需求，常常是非功能需求，如性能、容错、安全性、互操作性、自适应性等非功能性质影响软件体系结构的扩展和改动，这称为软件体系结构的演化。由于对软件体系结构的演化常常由非功能性质的非形式化需求描述引起，因而需要重复第1步，如果由于功能和非功能性质对以前的软件体系结构进行演化，就要涉及软件体系结构的理解，需要进行软件体系结构的逆向工程和再造工程。

2.4 软件体系结构的生命周期模型

(6)软件体系结构的提供、评价和度量(Software Architecture Provision, Evaluation and Metric)

这一阶段通过将软件体系结构实施于系统设计后系统实际的运行情况，对软件体系结构进行定性的评价和定量的度量，以利于对软件体系结构的重用，并取得经验教训。

2.4 软件体系结构的生命周期模型

(7)软件体系结构的终结(Software Architecture Termination)

如果一个软件系统的软件体系结构进行多次演化和修改，软件体系结构已变得难以理解，更重要的是不能达到系统设计的要求，不能适应系统的发展。这时，对该软件体系结构的再造工程即不必要、也不可行，说明该软件体系结构已经过时，应该摈弃，以全新的满足系统设计要求的软件体系结构取而代之。

谢谢大家！

再见！