软件体系结构

第五章 动态软件体系结构

5.1 动态软件体系结构概述
5.2 软件体系结构动态模型
5.3 动态体系结构的描述
5.4 建模工具及应用
5.5 动态体系结构特征

第五章 动态软件体系结构

传统的SA研究设想体系结构总是静态的，即软件的体系结构一旦建立，就不会在运行时刻发生变动。但人们在实践中发现，现实中的软件往往具有动态性，即它们的体系结构会在运行时发生改变。

第五章 动态软件体系结构

SA在运行时发生的变化包括两类：

一类是软件内部执行所导致的体系结构改变。比如，很多服务器端软件会在客户请求到达时创建新的构件来响应用户的需求。某个自适应的软件系统可能根据不同的配置状况采用不同的连接子来传送数据；

另一类变化是软件系统外部的请求对软件进行的重配置。比如，有很多高安全性的软件系统，这些系统在升级或进行其他修改时不能停机。因为修改是在运行时刻进行的，体系结构也就动态地发生了变化。在高安全性系统之外也有很多软件需要进行动态修改，比如很多操作系统期望能够在升级时无须重新启动系统，在运行过程中就完成对体系结构的修改。

第五章 动态软件体系结构

通过以上的描述，可以看出，静态的SA，由于缺少表示动态的变化机制，不能用来指导系统进行动态演化，所以，它不适应分析描述这类系统。因此对动态软件体系结构DSA(Dynamic Software Architecture)的研究应运而生。

Perry在IFIP2000年的世界计算机大会主题演讲中，重点指出，在软件体系结构中，主要的三个研究方向是：第一，体系结构风格；第二，体系结构连接件；第三，动态软件体系结构。由此可见，动态软件体系结构的重要性是不容忽视的。

5.1 动态软件体系结构概述

1、动态软件体系结构概念

软件体系结构的演化是指由于系统技术、需求、环境、分布等因素的改变从而引起软件体系结构的最终演变。

通常，软件体系结构在软件生命周期除了运行阶段以外所发生的变化定义为软件体系结构的扩展；而当软件系统在其运行时，软件体系结构发生的变化定义为软件体系结构的动态性。

5.1 动态软件体系结构概述

1、动态软件体系结构概念

综合以上两种情况，动态软件体系结构既能支持体系结构的扩展，又能适应体系结构的动态性，它是边界开放的。在软件体系结构演化的不同阶段，体系结构应能随着需求变更和对系统性能评价的要求而发生改变；在应用系统的运行中，体系结构的组成部分即组件、连接件还能够增加、删除、修改和替换；软件体系结构构成的规则也可以变化。

5.1 动态软件体系结构概述

2、研究内容

现阶段，动态软件体系结构研究可分为两个部分：

1)体系结构设计阶段的支持。主要包括变化的描述、根据变化如何生成修改策略、描述修改过程、在高抽象层次保证修改的可行性以及分析、推理修改所带来的影响等；

2)运行时刻基础设施的支持。主要包括系统体系结构的维护、保证体系结构修改在约束范围内、提供系统的运行时刻信息、分析修改后的体系结构符合指定的属性、正确映射体系结构构造元素的变化到实现模块、保证系统的重要子系统的连续执行并保持状态、分析和测试运行系统等。

5.2 软件体系结构动态模型

软件体系结构动态演化，不是简单地进行构件、连接件的创建和删除，它需要确定体系结构变化的起因，根据系统运行的状态决定体系结构变化的时间，给出体系结构变化方案，从而确保系统正确、完整地进行动态演化。因此，动态体系结构建模的核心问题就是提供系统化的方法，描述体系结构动态演化的诸多要素，从而能够全面地对动态体系结构建模。动态体系结构建模时，具体需要考虑的问题包括：

5.2 软件体系结构动态模型

(1)体系结构动态演化的起因。体系结构动态演化的起因能够分为两类：一类是系统内部的原因，即构件或连接件内部出错或发生异常。另一类是系统外部原因，如客户指令、负载动态平衡调整等。

(2)体系结构动态演化的时间。系统运行中，不能随时、随意进行系统的动态调整，否则可能造成数据丢失或系统异常。只有当相关构件、连接件处于某一安全状态的时候，方能允许体系结构动态调整。

5.2 软件体系结构动态模型

(3)体系结构演化的非瞬时性。体系结构从某一安全的时刻开始演化，到演化结束进入一个新的完整状态，需要执行多个动态配置动作，经过系列中间状态。其间如果相关构件和连接件继续执行，可能导致错误或系统死锁。

(4)构件从断点开始继续执行的能力。构件执行到某一安全点时，从系统配置上撤换下来，它保持一定的状态信息。当该构件重新连接进入系统时，很多时候需要从撤换时的断点继续开始执行。

5.2 软件体系结构动态模型

(5)体系结构动态演化的基本操作。体系结构动态演化的基本操作是动态调整体系结构的基本命令，组合运用这些基本命令，实现体系结构动态演化。常见的基本命令包括构件、连接件的创建和删除、端口角色连接关系的建立和撤消。

(6)体系结构动态演化的完整方案。综合考虑和协调体系结构动态演化的诸多因素，给出体系结构动态配置的完整方案，用来执行和控制体系结构的动态演化，保障演化完整进行。

(7)动态体系结构规约的形式语义。不仅需要提供体系结构动态演化的描述方法，用以完整地描述体系结构动态演化，而且需要给出描述方法的形式语义，从而能够支持体系结构动态演化的分析和仿真。

5.2.1 基于构件的动态系统结构模型

1、模型介绍

基于构件的动态体系结构模型CBDA(Component Based Dynamic system Architecture model)支持运行系统的动态更新，它分为三层：应用层、中间层和体系结构层。

5.2.1 基于构件的动态系统结构模型

1、模型

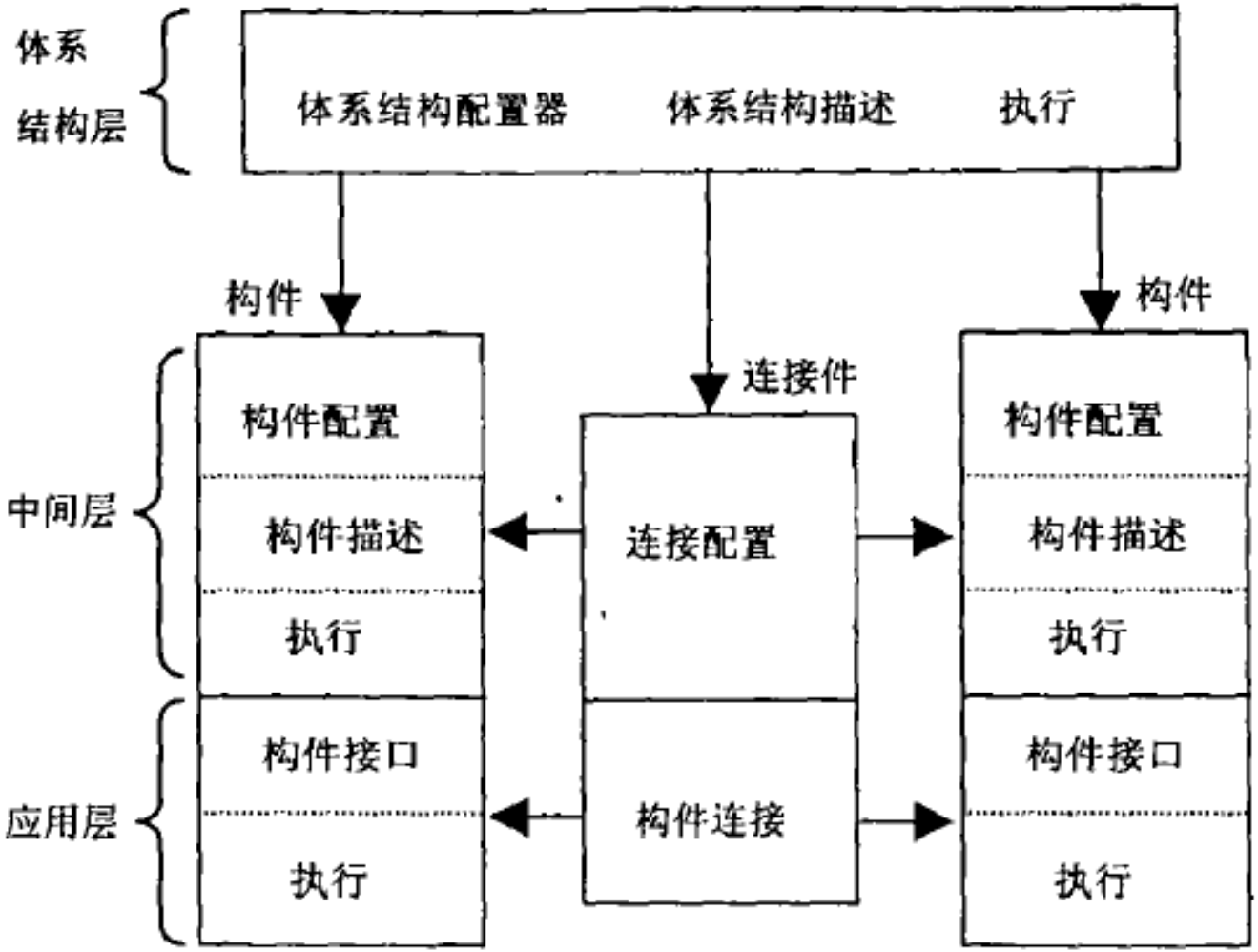


图 3 1 三层结构模型图

5.2.1 基于构件的动态系统结构模型

1、模型介绍

应用层处于最底层，包括构件连接、构件接口和执行。构件连接定义了连接件如何与构件相连接；构件接口说明了构件提供的服务，例如消息、操作和变量等等。在这一层，可以添加新的构件、删除或更新已存在的构件。

中间层包括连接件配置、构件配置、构件描述及执行。连接件配置主要是连接件及接口的通信配置；构件配置管理构件的所有行为；构件描述对构件的内部结构、行为、功能和版本等信息加以描述。在这一层，可以添加版本控制机制和不同的构件装载方法。

5.2.1 基于构件的动态系统结构模型

1、模型介绍

体系结构层位于最上层，控制和管理整个体系结构，包括体系结构配置器、体系结构描述和执行。其中，体系结构描述主要是描述构件以及它们相联系的连接件的数据；体系结构配置控制整个分布式系统的执行，并且管理配置层；体系结构描述主要是对体系结构层的行为进行描述。在这一层，可以更改和扩展更新限制，更改系统的拓扑结构，更改构件到处理元素之间的映射。

5.2.1 基于构件的动态系统结构模型

1、模型介绍

在每一层都有一个执行部分，主要是对相应层的操作进行执行。

在更新时，必要情况下将会临时孤立所涉及的构件。在更新执行之前，要确保：

(1)所涉及的构件停止发送新的请求；

(2)在更新开始之前，连接件的请求队列中的请求全部已被执行。

而且，模型封装了连接件的所有通信，这样可以很好的解决动态更新时产生的不一致性问题。

5.2.1 基于构件的动态系统结构模型

2、更新请求描述

更新可以由用户提出，也可以由系统自身发出请求。一般来说，一个更新描述包括以下几个部分：

(l)更新类型(update type)：更新类型包括添加、删除和更新一个新的构件；

(2)更新对象列表(list of updated objects)：需要更新的对象类的ID号；

(3)对象的新版本说明(new versions of the objects)：对象的新版本执行情况；

(4)对象更新方法(update method)：更替、动态及静态；

(5)更新函数(update function)：用来更新一个执行对象进程的状态转换函数；

(6)更新限制(update constraints)：描述更新(包括子更新)和它们之间的关系的序列，例如只有对象A的版本〉=2.0时，对象A才能被更新。

5.2.1 基于构件的动态系统结构模型

2、更新请求描述

```
<update_deseriptor>
  <add_obj to="server01//compl">
    <object name="C">
      <implementation>…</>
    </object>
  </add>
  <remove_obj from="server01//compl">
    <object name="D">
    </object>
  </remove>
  <update_obj in="server01//compl" >
    <object name="A" method="replaee">
      <old_version>1.0</>
      <new_version>1.1</>
      <implementation>…</>
    </objeet>
  </update>
  <update_obj in="server01//eompl">
    <object name="B" method="dynamie">
      <old_version>1.1</>
      <new_version>1.2</>
      <update_function>...</>
      <implementation>…</>
    </object>
  </update>
</update_descriptor>
```

5.2.1 基于构件的动态系统结构模型

3、更新执行步骤

按照CBDA模型的结构，对系统进行更新，一般来说，有以下几个步骤：

(l)检测更新的范围。在更新执行之前，首先要判断是局部更新还是全局更新，局部更新作用于需更新构件的内部而不影响系统的其他部分，全局更新影响系统的其他部分，全局更新需要发送请求到更高的抽象层。

(2)更新准备工作。如果更新发生在应用层，构件配置器等待参与的进程(或线程)发出信号，以表明它们已处于可安全执行更新的状态；如果更新发生在配置层，就需要等待连接件中断通信和其他构件配置器已完成它们的更新；如果更新发生在体系结构层，就直接执行。

(3)执行更新。执行更新，并告知更新发起者更新的结果。

(4)存储更新。将构件或体系结构所作的更新存储到构件或体系结构描述中。

5.2.1 基于构件的动态系统结构模型

4、实例

下面用两种更新（局部更新和全局更新）来分析CBDA模型如何支持体系结构动态更新。

（1）局部更新

局部更新由于只作用于需要更新的构件内部，不影响系统的其他部分，因此比全局更新要简单，步骤如图所示。

5.2.1 基于构件的动态系统结构模型

4、实例

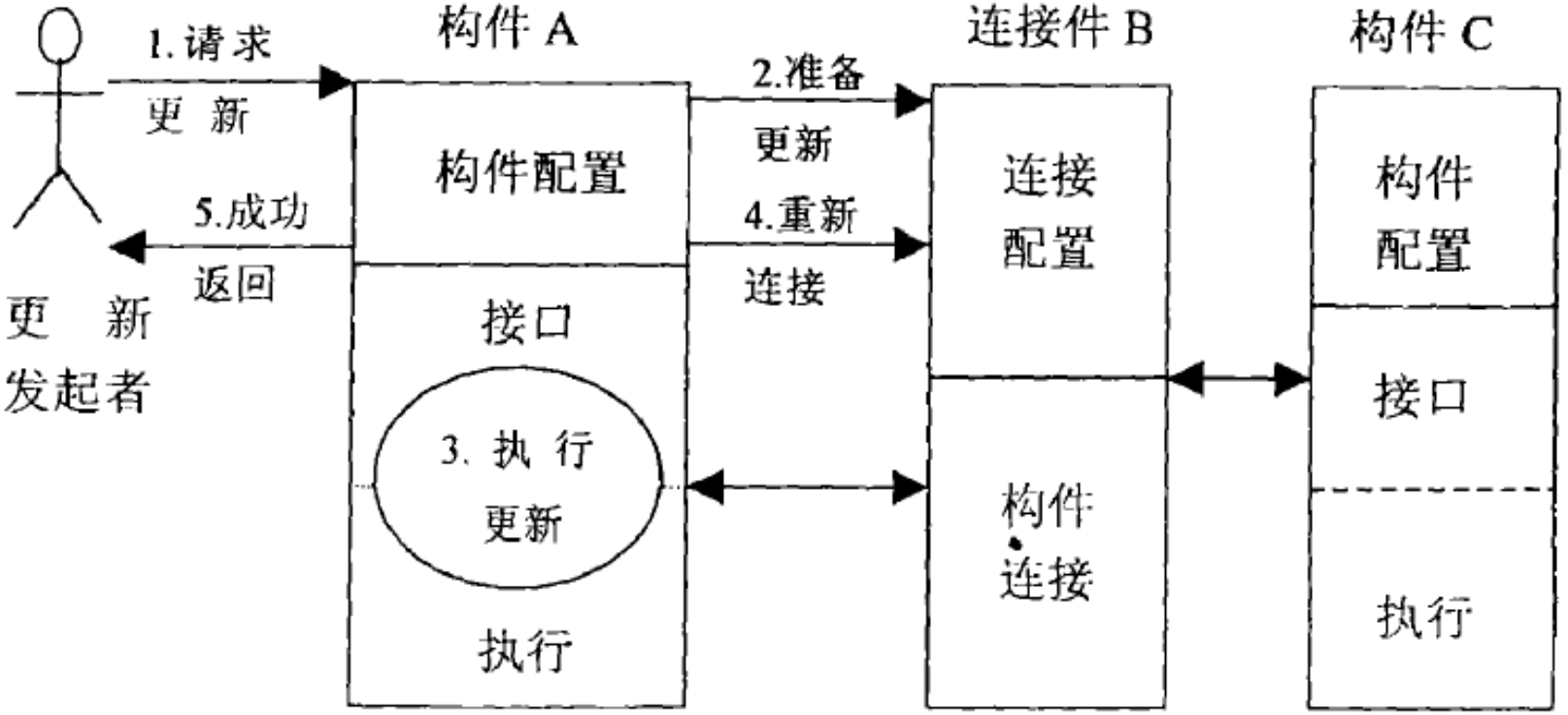


图 3.3 局部更新

5.2.1 基于构件的动态系统结构模型

4、实例

（1）局部更新

图中线条人表示更新发起者，更新发起者可以是系统的某一状态，也可以是系统用户。构件A和构件C通过连接件B连接，构件A内部需要更新。局部更新请求可按照下列步骤执行：(步骤序号与图中箭头符号相对应，表示更新中操作的顺序)

1）更新发起者发出一个更新请求，这个请求被送到构件A的配置器中，构件配置器将分析更新的类型，从而判断它是接口对象的局部更新；

2）由于更新为局部更新，构件A的配置器发出一个信号给连接件以隔离构件A的通信，准备执行更新；

5.2.1 基于构件的动态系统结构模型

4、实例

（1）局部更新

3）构件A的配置器开始执行更新。

4）更新执行完毕后，构件A的构件描述被更新，并且构件A发送一个消息给连接件B，两者间的连接被重新存储起来；

5）将更新结果返回给更新发起者。

由上述分析可知，在整个更新过程中，构件C都没有受到影响，这说明按照CBDA模型的方法，不会影响系统的其他部分的运行。

5.2.1 基于构件的动态系统结构模型

4、实例

（2）全局更新

下面以一个Client-Server系统动态更新实例来说明CBDA模型在全局更新中的应用，在本例中，要求更新某一Server构件。按照CBDA模型，用UML的时序图来描述动态更新过程，如图所示。

5.2.1 型

4、实
（2）

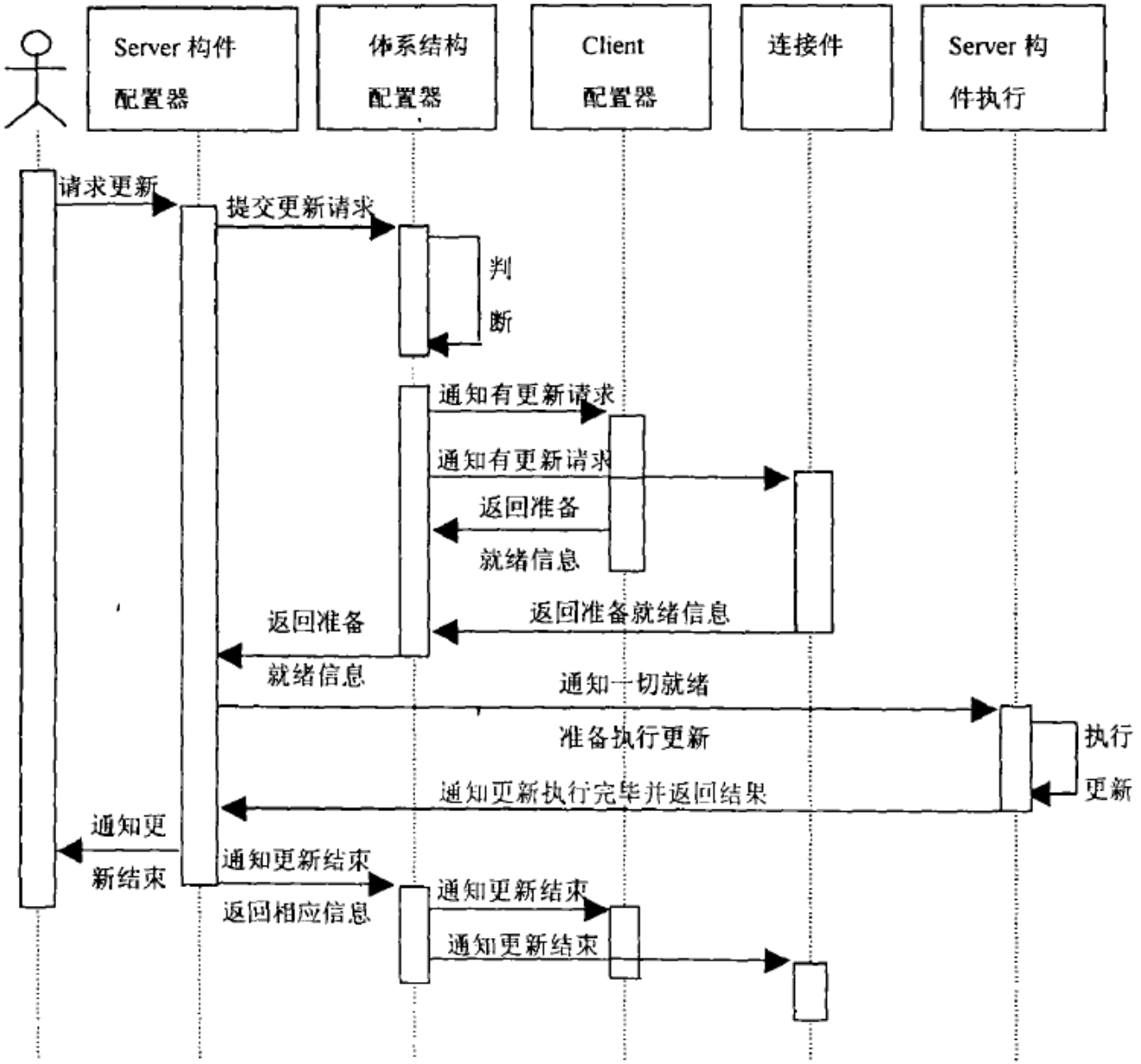


图 3.4 Client-Server 系统更新时序图

5.2.1 基于构件的动态系统结构模型

4、实例

（2）全局更新

步骤如下：

1）Server构件配置器接收到更新发起者提出的更新请求后，向体系结构配置器提交更新请求；

2）体系结构配置器对更新请求的类型进行分析，判断是否在更新限制（属于全局更新还是局部更新）范围内，不在更新限制范围内的更新不予执行；如果更新在限制范围内，体系结构配置器对更新所涉及的连接件和构件（本例中为Client构件和连接件）发出消息，要求它们做好更新准备工作；

5.2.1 基于构件的动态系统结构模型

4、实例

（2）全局更新

3）准备工作完成后，Client构件配置器和连接件向体系结构配置器返回就绪信息；

4）一切准备就绪后，体系结构配置器通知Server构件执行更新；

5）更新执行完毕后，向Server构件配置器、体系结构配置器和更新发起者通知更新执行完毕并返回更新结果；同时，体系结构配置器通知Client构件和连接件更新结束，可继续正常的运行。

这样，在没有影响系统的运行情况下，按照更新发起者的要求对系统进行了更新，并且维护了系统的一致性。

5.2.2 πADL动态体系结构

πADL借鉴、遵循ACME、Wright等给出的已被广泛认同的体系结构描述框架，提供专门的标记符号，围绕体系结构抽象级别的实体如构件、连接件、系统配置、体系结构风格等进行体系结构建模。

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

运用扩充的BNF范式给出πADL描述体系结构的框架。其中，运用“//”标记注释，运用黑体标记关键字，运用大写单词标记此处不再定义的非终结符，或者不再给出。[1…]表示其中的项出现一次，[01…]表示其中的项出现0次或1次，[1+…]表示其中的项出现1次或多次，[0+…]表示其中的项出现0次或多次。

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

πADL既能描述单个系统的体系结构，又能描述代表一类系统的体系结构风格，并且支持体系结构的层次配置，支持动态体系结构和体系结构精化。

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

单个系统体系结构描述方法主要包括三个部分：类型、配置和约束。

//单个系统的体系结构描述：

πADL Architecture Specification::=System

System_Name

[1 Type_Specification] //类型

[1 Configuration_Specification] //配置

[0+ Constraint_Specification] //约束

End System_Name

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

类型主要包括二个部分：构件和连接件。

//类型定义

Type_Specification::=Type:

[l+ Component_Specification] //构件

[l+ Connector_Specification] //连接件

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

构件包括原子构件和复合构件。

Component_Specification::=Atomie_Component|COMPOSITE_COMPONENT

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

构件是独立的功能计算单位。πADL中，软件构件具有两个重要的组成部分：接口部分和计算(Computation)部分。

//构件定义时，可以规定它的参数

Atomic_Component::=Component Com_Name

[01 Parameter--Specification] //参数

[l+ Port_Specification] //端口

[1 Computation_Specification] //计算

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

构件接口由若干端口(Port)组成，端口代表构件的逻辑交互功能，是构件计算行为的部分规约。

//端口的行为用扩充的π进程、接口类型或参数定义

Port_Specification::=Port PName= EXTENDED_π_PROCESS|Interface_Name |Parameter_Name

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

计算部分描述构件的整体计算功能，它执行各个端口规约的交互行为和内部计算行为，并把它们结合成为一个有机计算整体，是构件计算功能的完整规约。

//计算用扩充的π进程定义

Computation_Specification::=Computation

=EXTENDED_π _PROCESS

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

连接件是构件之间连接、交互和通信的机制。πADL显式、独立地规约连接件，能够让构件之间的交互模式局部化、独立化，并且增强构件的独立性，从而增加构件复用机会。

//连接件包括原子连接件和复合连接件，复合连接件的描述方法略。

Connector_Specification::=Atomie_Connector|COMPOSITE_CONNECTOR

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

πADL连接件具有两个重要组成部分：角色部分和粘接(Glue)规约。

//连接件定义时，可以规定它的参数

Connector--Specification::=Connector Con_Name

[01 Parameter_Specification] //参数

[l+ Role_Specification] //角色

[1 Glue_Specification] //胶合

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

连接件接口由若干角色(Role)组成，角色定义参与连接件交互机制的构件的必须遵守的行为。

//角色的行为用扩充的π进程、接口类型或参数定义

Role_Specification::=Role RName=
EXTENDED_π_PROCESS|Interface_Name|
Parameter_Name

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

粘接规约具体描述连接件的连接、交互协议，协调参与交互的多个角色协同工作。

//胶合用扩充的π进程定义

Glue_specification::=Glue EXTENDED_π_ROCESS

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

系统配置是构件、连接件按照一定方式组装而成的软件拓扑结构。

//配置定义

Configuration_Specification::=

[1 Instance_Specification]//实例

[1 Assembly_Specification]//组装

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

//实例规约

Instance_Specification::=Instances:

[1+ Instance_Definition]

Instance_Definition::=

[1 Component_Definition|Connetor_Definition]

//实例定义时，可能需要设定相关参数

Component_Definition::=[l+ Com_Name [01 (Actual_Parameter)]]:Com_Type

Conneetor_Definition::=[l+ Con_Name [01 (Aetual_Parameter)]]:Cone_Type

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

1）单个体系结构描述

系统常见组装方式是构件端口、连接件角色进行连接而组装成为一个整体，πADL运用Attach To语句进行描述。aCom.aPort AttachTO aCon.aRole表示构件aCom的端口aPort连接到连接件aCon的角色aRole。

//组装规约

Assembly_Specification::=Assembly:

[1 Connection_Assembly|SIMPLE_ASSEMBLY

|REPLICATE_ASSEMBLY]

Connection_Assembly::=[l+ Com_Name.Pname

AttaehTO Con_Name.Rname]

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

2）体系结构风格描述

πADL主要从两个方面规约体系结构风格：风格的词汇表(Vocabulary)和配置约束。根据需要，同样可以规约风格的配置拓扑结构。

//体系结构风格描述：

πADL Architectural Style Specification::=Style

Style_Name

[1 Idiom_Specification] //风格术语

[01 Configuration_Specification] //配置

[0+ Constraint_Specification] //约束

EndStyleesName

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

2）体系结构风格描述

风格的词汇表定义该风格的名词术语，常运用构件和连接件类型进行规约。

//风格术语规约:

Idiom_Specification::=[l+ Interface_Specification |Component_Specification|Connector_Specification]

Interface_Specification::=Interface_Nmae=EXTENDED_π_ PROCESS

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

2）体系结构风格描述

配置约束定义风格所有成员配置必须遵守的条件，πADL采用基于一阶谓词逻辑的方法进行约束规约。

//约束规约，采用一阶谓词逻辑进行约束描述。

Constraint_Specification::=NOTATIONS_BASED_ON_PREDICATE LOGIC

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

2）体系结构风格描述

此外，πADL提供接口规约和参数化两种机制方便体系结构风格规约。

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

3）体系结构动态性描述

//体系结构动态配置描述，具体见后面。

πADL Dynamic Architecture Specification::=DYNAMIC_CONFIGUROR

4）体系结构精化描述

//体系结构精化描述，略。

πADL Architecture Refinement Specification::= [1 ARCHITECTURE_REFINEMENT]

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

5）参数描述

//参数可以是进程、整数、行为名字或其它，π ADL规约的任意部分可为参数

Parameter_Specifieation::=(Parameter_Name:Parameter_Type [0+, Parameter_Name:Parameter_Type])

Aetual_Parameter::=Parameter_Value[0+， Parameter_Value]

Paramete_Value::=EXTENDED_π_PROCESS|INT|Action_Name|OTHER

5.2.2 πADL动态体系结构

1、πADL体系结构描述框架定义

6）名字描述

//名字规约

Com_Name::=IDENTIFIER
Com_Type::=IDENTIFIER
Con_Name::=IDENTIFIER
Con_Type::=IDENTIFIER
PName::=IDENTIFIER
RName::=IDENTIFIER
Config_Name::=IDENTIFIER
Style_Name::=IDENTIFIER
Interface_Narne::=IDENTIFIER
Parameter_Name::=IDENTIFIER
Parameter_Type::=IDENTIFIER
Action_Name::=IDENTIFIER

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

针对动态体系结构建模的诸多问题，πADL动态体系结构建模的基本思路如下：

(l)构件的computation、连接件的Glue进程中，插入用于体系结构动态演化的特定控制名字，表达体系结构动态演化的起因和安全地进行动态演化的时间。

(2)运用π演算作为统一的形式语义基础，建立单独的动态配置进程，形式化描述体系结构动态演化的方案。它与构件的Computation进程、连接件的Glue进程相互交互和作用，总体控制体系结构的动态演化。

(3)运用π演算固有的动态建模能力，通过连接件动态输入不同构件的交互行为名字，实现与构件的动态连接和交互，并据此给出体系结构动态配置的形式语义。

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL进行动态体系结构建模的具体方法如下：

(l)对体系结构动态配置的起因建模。

πADL在构件的Computation进程、连接件的Glue进程中向配置进程输出特定的控制名字，表达引发动态配置的内部原因，如输出Error、Exception，表达构件或连接件内部发生错误或异常，要求配置进程进行动态配置。而在配置进程中从外部环境输入特定的控制名字，表达体系结构动态配置的外部原因。如RequestUpgrade表示用户要求服务升级。

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL进行动态体系结构建模的具体方法如下：

(2)对体系结构动态配置的时间建模。

πADL在构件的Computation进程、连接件的Glue进程中插入特定的控制名字BeginOk，表达它们运行到该处的时候，处于一个安全状态，能够开始接受接受动态配置指令，进行系统演化。

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL进行动态体系结构建模的具体方法如下：

(3)对体系结构动态配置的非瞬时性建模。

πADL根据体系结构配置方案，在构件的Computation、连接件的Glue进程中引入特定的控制名字EndOk，与配置进程的相关行为同步，控制构件或连接件，使它们只有在动态配置结束后，才能重新启动执行。

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL进行动态体系结构建模的具体方法如下：

(4)对构件从断点开始继续执行的能力进行建模。

根据πADL动态体系结构建模语义，构件从连接件上撤换下来时，系统中不再存在能够与之交互的进程，它的交互要求不能得到满足，故不能够继续运行。当它再次与连接件连接时，连接件输入它的交互行为名字，从而能够与它交互，构件得以在断点开始继续执行。

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL进行动态体系结构建模的具体方法如下：

(5)对体系结构动态变化的基本操作建模。

πADL运用特定动作名字表示动态配置的基本操作。τ_New_Iname_Tname表示生成类型Tname的实例Iname，τ_Delete_Iname表示删除构件或连接件实例Iname。τ_Attaeh_M_N_TO_I_J、τ_Detach_M_N_From_I_J分别表示在端口M.N和角色I.J之间建立连接和撤消连接。它们都是配置进程的内部行为。

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL进行动态体系结构建模的具体方法如下：

(6)对体系结构动态演化的方案建模。

πADL运用π演算作为统一的语义基础，建立专门的动态配置进程Dynamic_Configulor，描述体系结构动态演化的方案。Dynamic_Configuror模拟一个监控进程，根据系统运行情况和环境变化因素，实施相应动态变化方案。Dynamic_Configuror通过专门通道X_Config、En_Config与实例X、外部环境进行通信，协调体系结构动态变化的诸多因素，控制体系结构动态变化。运用BNF范式，定义Dynamic_Configuror的语法和结构如下。其中，单字母的终结符运用双引号标记，如“+”表示π演算的“和”运算符。

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL的BNF定义中，DYNAMIC_CONFIGUROR的定义如下：

//动态配置进程包括两个部分：系统初始配置和动态配置
DYNAMIC_CONFIGUROR::=Dynamic_Configuror=
[1 Initial_Configuration_Instruetion] //系统初始配置指令
[l Dynamic_configuration_Program] //系统动态配置方案

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL的BNF定义中，DYNAMIC_CONFIGUROR的定义如下：

//执行系列配置动作，创建初始系统，并执行动态监控进程
Dconfiguror
Initial_Configuration_Instruetion::=[1+ Action “.”]
Dconfiguror

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL的BNF定义中，DYNAMIC_CONFIGUROR的定义如下：

//动态配置方案给出Dconfigurour的定义，它监控系统和环境，并执行动态配置

Dynamic_Configuration_Program::=Dconfiguror=

[01 System_Monitor]　　//系统监控部分

[01 “+” Envirenment_Monitor]　　//环境监控部分

//从多个构件或连接件实例接受动态配置的控制信息并执行动态配置规则

System_Monitor::=[1 X_Config(y).Rule][0+ “+” X_Config(y).Rule]

//环境监控部分接受环境的动态配置信息并执行动态配置规则

Envirenment_Monitor::=En_Config(y).Rule

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL的BNF定义中，DYNAMIC_CONFIGUROR的定义如下：

//根据收到的各种控制信息，执行具体的动态配置方案

Rule::=[l ([y=controlname].Subrule[0+ “+”[y=controlname].Subrule])]

//动态配置方案包括：发布动态配置开始、进行动态配置和发布动态配置结束

SubRule::=[1 Begin_Config] [I Dynamic_Config] [1 End_Config]

//向多个相关实例发布动态配置开始指令

Begin_Config::=[1+ X_Config<BeginOk> “.”]

//执行多个动态配置行为

Dynamic_Config::=[l+ Action “.”]

//向多个相关实例发布动态配置结束指令，并循环执行动态配置方案Dconfiguror

End_Config::=[l+ X_Config<EndOk> “.”] Dconfiguror

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

πADL的BNF定义中，DYNAMIC_CONFIGUROR的定义如下：

//配置基本命令包括实例的生成和删除，连接关系的建立和撤消

Action:=τ_New_Iname_Tname | τ_Delete_Iname | τ_Attach_M_N_TO_I_J | τ_Detach_M_N _From_I_J

//动态配置控制名字，动态体系结构建模人员能够定义自己的控制名字

Controlname::=Error | Exception | FixOk | RequestUpdate | …

X::=IDENTIFIER **y::=IDENTIFIER**

Iname::=IDENTIFIER **Tname::= IDENTIFIER**

M::=IDENTIFIER **N::=IDENTIFIER**

5.2.2 πADL动态体系结构

2、πADL动态体系结构建模方法

Dynamic_Configuror首先执行系列配置动作，创建一个运行系统。

然后通过X_Configuror、En_Config通道监听各个实例X和外部环境。

当收到表示动态配置起因的控制名字时，根据不同的控制名字，执行不同的配制规则SubRule。

在SubRule中，首先通过系列X_Config通道向实例X输出控制名字BeginOK，发出动态配置开始指令，等待和控制相关构件、连接件处于安全状态.

然后执行系列配置行为，最后输出系列控制名字EndOK，告诉相关构件、连接件配置结束，重新开始执行。

SubRule执行完动态配置后，执行DConfiguror，继续处于监控状态。

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

πADL动态体系结构运用π进程作为体系结构动态配置的形式语义，即根据构件、连接件和动态配置进程Dynamic_Configuror，推导得出一个π进程，该π进程按体系结构动态配置的直观语义执行，形式化描述了动态配置系统的整体行为。

基本思想是：动态配置软件系统的整体行为是所有构件实例的进程、连接件实例的进程、动态配置进程Configuror的并发运行。但关键要点是表达构件、连接件的动态创建和删除，表达构件、连接件之间的动态连接关系。πADL的方法是：

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

(1)构件、连接件的动态创建和删除。分析Configuror程序，在创建初始系统和每次动态配置结束的地方，插入代表新创建的构件、连接件的实例进程并与余下的Configuror进程并发执行，实现构件的动态创建。

π演算并不直接提供进程删除的手段，故只能运用等价的方法对构件、连接件的删除建模，即使它们不能再执行。当构件不再与任何连接件连接时，系统中不能找到对偶名字，它的行为不能执行，故效果等价于构件删除。连接件删除时，通过输入行为，替换连接件的所有通道名字为Void，从而它也不能找到对偶的通道名字，不能再执行，等价于连接件删除。

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

(2)运用π演算的动态建模能力，借助连接件实现构件、连接件的动态连接关系。πADL在连接件的Glue进程和配置进程configuror中自动插入连接件交互通道的输入、输出行为。动态配置中，Configuror根据配置变化情况，向连接件输出当前的连接通道，连接件接受该输出，得到新的连接通道，从而动态改变与构件的连接关系，实现体系结构的动态变化。借助连接件实现体系结构的动态变化，构件的Computation进程不受动态变化的影响，便于动态变化的分析和实现。

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

下面给出动态体系结构行为推导算法。算法的骨架是分析动态配置进程**Dynamic_Configuror**，构造描述动态体系结构行为的π进程。

算法1（动态体系结构行为推导算法）

输入：构件、连接件定义、动态体系结构配置进程**Dynamic_Configuror**

输出：描述动态体系结构行为的π进程

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

第一步：改造连接件，在连接件中插入交互通道输入行为。在所有连接件Glue代码中，在动态配置开始和结束之间，即在Config(y).[y=BeginOK]和Config(z).[y=EndOK]之间，插入用于输入连接件交互通道的行为Config(x)，其中x为该连接件所有交互行为名字构成的矢量序列，于是形成：Config(y).[y=BeginOK].Config(x).Config(z).[Z=EndOK]。

它的作用是动态配置结束之前，连接件重新得到交互通道，从而动态改变构件、连接件之间的交互关系，实现动态配置。

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

第二步：分析配置进程Dynamic_configuror的初始行为，动态创建系统的初始配置。即分析Initial_Configuration_Instruction::=[l+ Action “·”] Dconfiguror，根据其中系列构件、连接件创建动作和它们之间的连接配置信息，调用静态配置的组装推导算法，得到表达初始配置的进程Pinitial。并运用Pinitial | Dconfiguror替换Dconfiguror，得到[l+ Action “.”] (pinitial | Dconfiguror)。该步骤的作用是动态创建初始系统，该初始系统与动态配置进程并发执行。

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

第三步：分析Dconfiguror的动态配置行为，动态改变体系结构。即分析语法元素SubRule::=[1 Begin_Config][1 Dynamie_Config][1 End_Config]中的动态配置部分Dynamic_Config，具体执行如下操作：

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

(l)动态创建新的构件和连接件。根据动态配置行为中构件、连接件的创建信息、新建连接件的端口角色连接情况，调用静态配置的组装推导算法，得到表达它们行为的进程Pcreated。

(2)动态改变端口、角色的连接关系。对已经存在、但所连接构件发生变化的连接件Xl、X2…Xn，根据新的端口、角色连接关系，向它们输出新的交互通道，得到X1_Config<y1>…Xn_Config<yn>，其中yi是交互通道的名字序列，它的元素形如M_N_A，表示构件M的端口N连接到该连接件上，连接件接受M_N_A，从而与构件M建立新的连接交互关系。

5.2.2 πADL动态体系结构

3、πADL动态体系结构建模语义

(3)连接件删除的处理。设在动态配置行为中，删除连接件Cl、C2、Cm，向它们输出代表空的void交互通道，得到Cl_Config<vl>.....Cn_Config<Vn>，其中vi是元素ci_void构成的元组，它们得到该交互通道后，因为系统中其它进程皆不具有该交互通道，因此它们不能执行，从而等价于连接件删除。

(4)综合。根据(l)、(2)、(3)的结果，修改Subrule如下：在动态配置行为后插入

:Xl_Config<yl>…Xn_Config<yn>.Cl_Conflg<vl>…Cn_Config<vn>，并用(pcreated | DConfiguror)代替End_Config::=[l+ X_Config<EndOK> “.”]Dconfiguror中的Dconfiguror。

5.3 动态体系结构的描述

动态ADL分为非形式化和形式化两种，非形式化方法主要是采用XML技术。当前，在动态软件体系结构研究过程中，是以形式化的描述语言为主流的研究趋势。

动态软件体系结构下形式化描述语言所必须满足的条件有以下几点：

1）能够描述组件和连接件等一些独立的实体；并能够刻画组件、连接件属性。如：组件的功能、质量等。

2）能够描述系统SA的动态性。如：运行时刻增减组件。

3）能够检查组件、连接件交互间的一致性。

5.3 动态体系结构的描述

当前在动态软件体系结构研究过程中，形式化描述语言主要表现在对已存在的一些ADLs的扩展上。其所采用的技术主要包含以下四类：

第一，图文法，组件用图的节点来表示，连接件用图的边来表示，利用协同机制管理体系结构，由图的重写(Graph Rewriting)规则实现其动态性。

第二，进程代数法，常用来研究并发系统通信，是一种基于代数以及微分知识为语义的，表示体系结构的重配置。能够表示并发系统的进程代数主要包含有：通信系统演算(CCS，calculus of communication systems)、通信顺序进程(CSP，communicating sequential processes)和π演算。

第三，时序逻辑的方法，它是基于逻辑重写，以时序逻辑为语义，表示体系结构的重配置。

第四，其它方法，指不能严格划分到以上三类的方法。

5.3.1 图文法Graph

1、Le Metayer和Hirsh方法

Le Metayer方法采用的是体系结构风格由上下文无关的图文法来描述的，而体系结构实例采用图形化元素来表示，组件是由图的节点来表示，连接件是由边来表示。基于图形化重写(Graph Rewriting)语义规则来描述体系结构的重配置(Reconfiguration)。

与 Le Metayer不同，Hirsh方法虽然仍用图来描述体系结构的动态性，但是连接件是由图的节点来表示，组件是由边来表示，同时引入了协同机制来对体系结构进行管理，利用图的重写规则来实现体系结构的动态性。

5.3.1 图文法Graph

2、Taentzer方法

在Taentzer方法中，以分布图和分布图的转化的形式来描述软件体系结构。其中分布图包含网络图和本地图。

3、COMMUNITY

COMMUNITY方法创建了一种基于范畴理论语义(category theory semantics)的形式化体系结构描述语言，采用标签图描述该体系结构，其重配置过程是采用double-pushout图形转换来描述的。

5.3.1 图文法Graph

4、CHAM(Chemical Abstract Machine)方法

Berry等用相互反应的化学物质来形容软件系统，并根据此想法，提出了化学抽象机模型CHAM。Inverardi和wolf提倡采用CHAM模型描述软件体系结构。如：实际应用此方法描述了编译器的体系结构。依据CHAM的语义规则，在体系结构描述和重配置过程中，应用了类似化学反应原理的反应规则及图重写规则。

5.3.2 进程代数法 (Process Algebra)

1、Dynamic Wright

Dynamic Wright是 Robert Allen等开发的体系结构描述语言。 Dynamic Wright扩展了Wright语言，以通信顺序进程 CSP(Communication sequence Process)为形式化语义基础，通过引入负责连接组件和连接件的配置器来描述体系结构的动态性，并利用基于CSP的验证工具进行验证。

5.3.2 进程代数法 (Process Algebra)

2、Darwin方法

Darwin是Magee和Kramer开发的体系结构描述语言。Darwin组件类型是采用接口定义的，而接口包括了服务提供接口和服务请求接口，组件实例、服务提供接口和服务请求接口间的绑定关系在系统配置中定义。惰性实例化 (Lazy instantiation)和直接动态实例化 (Direct dynamic instantiation)两种特殊机制用以支持体系结构动态性，但存在局限。

5.3.2 进程代数法 (Process Algebra)

2、Darwin方法

基于π演算，Darwin提供语义规则，把组件提供的服务作为一个名字，服务提供组件与服务请求组件间的绑定关系作为一个进程，这个进程传递该服务名字到服务请求组件。Darwin没有独立的连接件元素。它的提供或请求连接模型必须在组件内进行描述，仅支持非对称的交互模式。一般情况，在定义体系结构风格时，只定义它的交互模型，而把组件定义留给体系结构师(Architect)。另外，Darwin运用参数化配置方法描述相似系统的规约。以上两种情况存在着局限，都不能很好支持体系结构风格规约。

5.3.2 进程代数法 (Process Algebra)

2、Darwin方法

Magee和Kramer应用π演算语义模型分析实现体系结构配置的分布式算法，然而，它未提供描述组件及相关服务属性的方法和技术。底层平台的实现方法决定了服务类型的语义。因此，Darwin自身并没有提出体系结构行为分析的基础和方法。

5.3.2 进程代数法 (Process Algebra)

3、LEDA方法

LEDA是Canal等开发的体系结构描述语言。与Darwin相同，都是基于π演算作为形式化语义规约动态软件体系结构。组件规约为接口、复合、附属关系。接口使用角色实例定义，角色实例定义了组件与其它组件的行为。特别的，行为表示了组件提供给系统的和要求连接到那个组件。LEDA中的attachments关系定义组件实例间的连接。顶层组件的附属关系即为连接件。附属关系可能为静态或可重配置的。LEDA也提供了验证以保证相互连接的角色间的正确性。

5.3.2 进程代数法 (Process Algebra)

3、LEDA方法

LEDA除了规约软件体系结构以外，还支持原型的执行。例如MWB(一种分析并发系统的π演算工具)可用于执行LEDA规约的原型系统。

5.3.2 进程代数法 (Process Algebra)

4、Pilar方法

Pilar方法是Cuesta等开发的体系结构描述语言。Pilar可表示层次化系统的动态重配置。系统中有两类组件：单一组件、复合组件。单一组件具有一个或多个接口，定义了交互点。复合组件包含端口和单一或其它复合组件。除了接口和组件复合以外，组件也具有CCS表示的约束集。在此约束下，重配置得以定义。Pilar具有两种活动：operators(组件)和operations(执行重配置)。

近来Pilar语言利用π演算扩展了它的语义，以程序语言的风格来描述体系结构。

5.3.2 进程代数法 (Process Algebra)

5、ArchWare

ArchWare是欧盟在以体系结构模型为中心构造的软件系统。ArchWare提出了动态体系结构描述语言π-ADL。而π-ADL是一个支持动态体系结构建模、精化、分析的基于高阶π演算的形式化语言。

5.3.2 进程代数法 (Process Algebra)

6、D-ADL

D-ADL是国内学者李长云提出的，基于高阶多型π演算为语义的体系结构描述语言。在D-ADL中，组件、连接件和体系结构风格定义为抽象(abstraction)类型；系统行为定义为进程 (process)；组件和连接件的交互点定义为通道(channel)。为方便系统变更逻辑的编写、修改和理解，D-ADL将动态行为从计算行为中分离出来，独立、集中地表达。由于动态行为可形式化为高阶π演算进程，其结果能够被预先推导。可用于系统联机演化和体系结构模型求精的规则。

5.3.3 逻辑法(Logic)

1、Gerel(Generic Reconfiguration Language)方法

Gerel起初作为重配置和并行分布项目的一部分，Gerel的描述作为一种通用语言，这种能力来自于它能表达特定和程序化的动态性。Gerel以一阶逻辑为语义。Gerel包含配置语言和程序语言。程序语言用于程序化的组件，配置语言用于配置组件，这些组件是由程序组件和其它配置组件构成的。Gerel的动态变化使用 keyword change作为变化的脚本在配置组件中规约。

5.3.3 逻辑法(Logic)

2、Aguirre-Maibaum方法

Aguirre和Maibaum开发了一种以一阶时序逻辑为语义的动态软件体系结构重配置语言。这种语言表示组件类型为类，类包含类符号C和C上的有限公理集；连接件类型表示为联系，联系识别了交互中参与者的组件类型和组件间的连接，这种连接由一个动作同步来定义；整个架构作为一个子系统，子系统用于定义拓扑的初始状态和应用到已创建状态的重配置操作。初始状态包含组件和连接件的实例。连接件接受组件实例名为参数。公理用于子系统中管理重配置的操作行为，包含架构中元素的增加和删除及拓扑的重配置。Aguirre-Maibaum方法利用了时序逻辑验证规约。类和子系统的属性都可通过使用Kripke结构来证明。

由于Aguirre-Maibaum方法中子系统仅仅由组件类型而不是其它子系统构成，所以无法实现层次化组件复合的系统。

5.3.3 逻辑法(Logic)

3、ZCL Framework方法

ZCL方法是Virginia等开发的基于CL模型，利用Z语言规约的框架描述语言。Z语言基于谓词逻辑和集理论。在ZCL框架中，Z计划用于规约CL模型的抽象，例如模块(组件)、端口、实例、连接和配置。Z在动态结构规约中区分两种类型的计划：状态计划，表示架构的拓扑结构；操作计划，表示配置和重配置操作。

ZCL Framework不仅仅能对动态软件体系结构规约，而且能对基于状态机的运行模型进行规约。运行模型和配置命令集用于管理动态变化。ZCL规约可用于模拟运行时刻行为和分析系统配置的不变条件和约束。当前模拟和分析可用Z-EVES定理证明工具来实现。

5.3.4 其它类型(Others)

1、C2SADEL(Software Architecture Description & Evolution Language)方法

C2是一种由Taylor等开发的体系结构描述语言。基于消息总线结构，C2用于刻画图形化用户界面的软件体系结构。它的组件模型包含消息发送和消息接收两个端口。依赖于消息总线结构的组件连接机制用于对消息的过滤和转发。C2只适合于特定领域，是一种简单的体系结构描述语言，它只提供了用于体系结构规约的语法规则，没有相关的语义规约，不支持体系结构风格的规约和分析，只支持消息总线风格的用户接口系统描述。

5.3.4 其它类型(Others)

2、Rapide方法

Rapide是一种Luckham等开发的体系结构描述语言。Rapide对组件的计算行为和交互行为，采用偏序事件集进行建模。与Darwin不同，Rapide是运用通信事件序列描述组件的行为，并独立实现的。其通信事件可分为外部动作和公共动作。在外部动作和公共动作之间通过观察外部动作而建立关联，从而达到描述组件和系统的计算行为。组件间的交互通过连接机制实现，连接机制定义了组件之间事件发生的次序。其连接结果是启动连接模式左边的事件，右边事件随之发生。Rapide与Darwin的绑定相似，也只能描述组件之间非对称的交互关系。其连接机制定义在配置当中，二者缺一不可，被Medividovic称为嵌入配置ADL。与动态wright相比，Rapide没有对连接件独立建模，且多个连接机制没有组合成一个整体，构成复杂的交互模式。因此Rapide在描述组件交互模式的能力方面存在局限性。

5.4 建模工具及应用

1、ArchStudio

ArchStudio是由加州大学Irwine分校提出的，一种支持C2体系结构的动态修改的工具。它有三个工具作为体系结构更改的源：Argo、ArchShell和Extension Wizard。Argo提供了一个图形化的体系结构表示，通过它可以方便地对体系结构模型进行操作。ArchShell采用一种文本的方式，利用命令行界面来指明运行时的更改，命令包括删除构件和连接件，体系结构重新配置及将体系结构以文本方式显示出来。Extension Wizard工具用来将更改发布给用户。但是ArchStudio采用的软件体系结构仅仅是设计阶段的制品，无法刻画真实系统的运行状态和行为，并且它仅限于处理遵循C2体系结构风格的软件动态修改，变化的合理性与正确性也难以确保。

5.4 建模工具及应用

2、SAA

SAA(Software Architecture Assistant)由伦敦皇家学院研制，它是一种用来描述、分析和构建动态体系结构的交互式图形工具集。特定的SAA只能用图形化描述Darwin系统结构模型，并利用外部工具分析结构，生成框架代码。虽然SAA提供了智能化图形工具，可是它仍不能操控运行系统。

5.4 建模工具及应用

3、K-Component

J.Dowling等人设计了K-Component元模型。K-Component元模型是通过提供体系结构元模型和适配契约来支持系统的动态重配置。适配契约显式地表达适配逻辑。软件体系结构由一个具有类型的连接图来表示，其中的图节点表示构件接口，类型标签表示构件，有向边则表示构件之间的连接。这个连接图和运行系统之间建立因果连接由一个反射机制来实现的，然而它主要是靠静态代码的映射，缺乏灵活性。

5.4 建模工具及应用

4、PKUAS系统

国内北京大学开发的PKUAS系统是基于运行时（Runtime Software Architecture,简称RSA）体系结构和J2EE兼容的一个反射构件操作平台。PKUAS采用构件化的平台内部体系结构，并引入运行时软件体系结构作为全局视图，用以实现反射体系对系统整体的表示和控制。它对反射的实现主要通过元模型、元协议和元数据。PKUAS采用扩展传统体系结构描述语言ABC/ADL描述运行时体系结构，使之具备能够继承设计阶段体系结构所含语义能力。但由于ABC/ADL缺乏构件行为和交互的形式化描述，PKUAS没有解决构件替换时的可观察行为的一致性问题。

5.4 建模工具及应用

5、Artemis-ARC

为了能够让应用系统以灵活的适应底层因特网计算环境和用户需求的变化，南京大学马晓星等提出了一种面向服务的动态协同架构，引入内置的运行时体系结构对象来解耦系统中的各个服务构件，并通过该对象，从体系结构的视角来重新解释服务构件之间的引用和交互。把体系结构从抽象的概念具体化为能够直接操作的对象。这样就可以利用面向对象程序设计语言的继承和多态等机制，实现面向体系结构的系统动态演化。并且以此开发了一个支撑平台Artemis-ARC，为具有动态演化能力的面向服务应用系统的开发、运行和监控提供了一套集成环境。

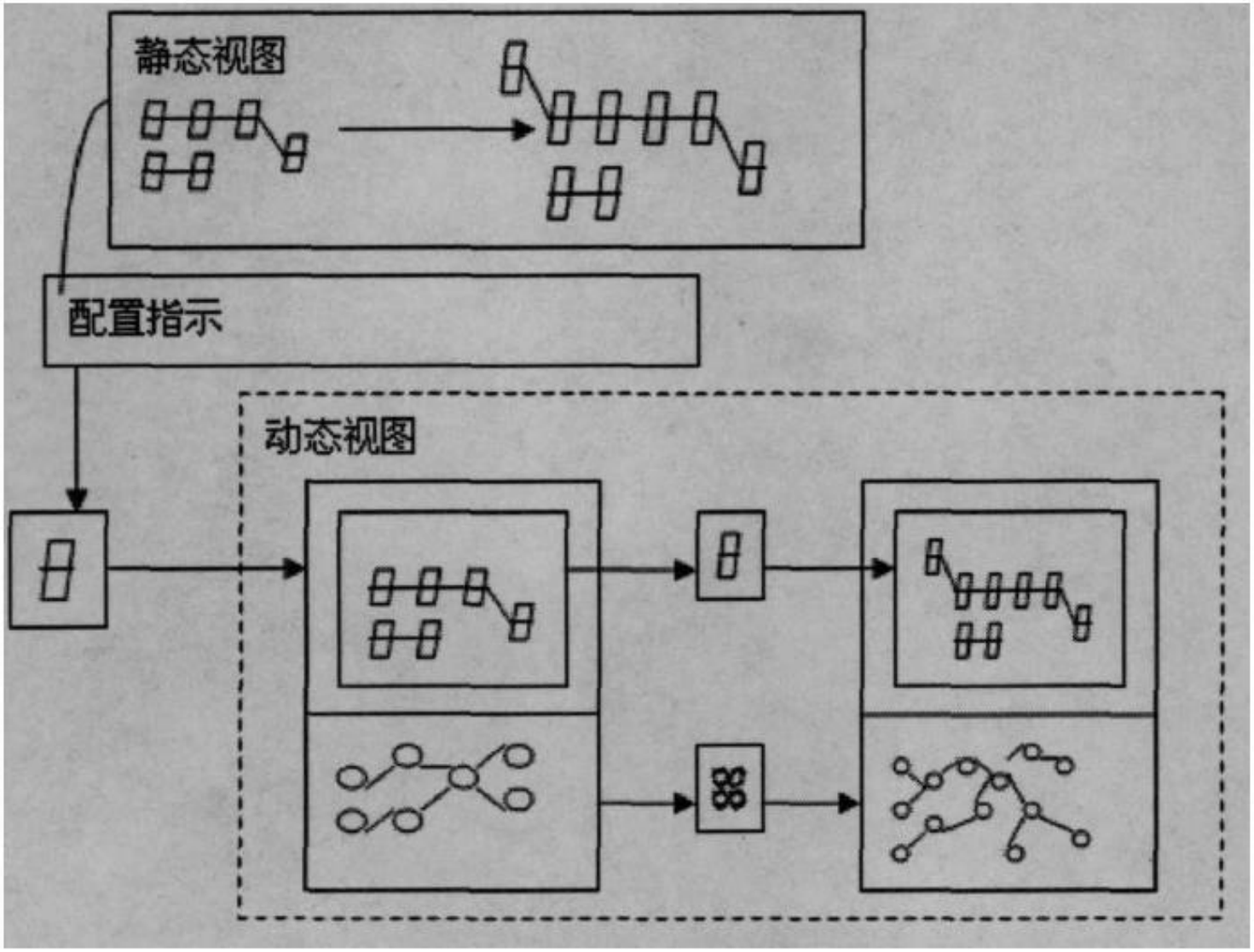
5.5 动态体系结构特征

动态软件体系结构的特征主要有三个特征：可构造性动态特征、适应性动态特征和智能型动态特征。

1、可构造性动态特征

可构造性动态特征通常可以通过结合动态描述语言、动态修改语言和一个动态更新系统来实现。动态描述语言主要是描述应用系统软件体系结构的初始配置；动态修改语言主要是描述体系结构发生改变时的改变，该语言支持增加或删除、激活或取消体系结构元素和系统遗留元素;动态变更可以通过体系结构框架或者中间件实现。

5.5 动态体系结构特征

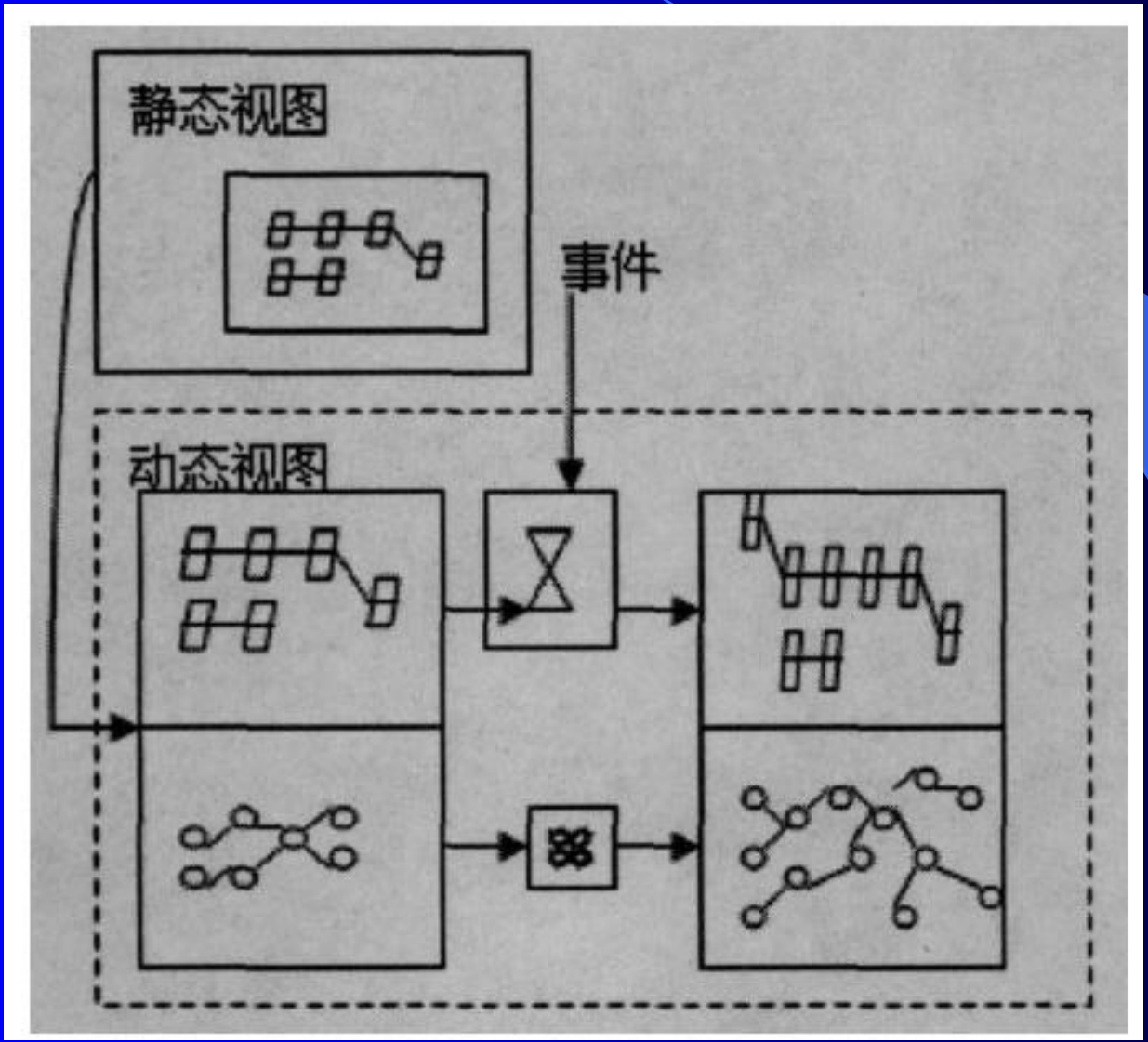


5.5 动态体系结构特征

2、适应性动态特征

适应性动态特征要求应用程序立即反映当前事件的能力，此时程序不能进行等待，必须把初始化、选择和执行整合到体系结构框架或中间件里面。适应性动态特征是所有事件在开发期间就已经进行了评估，然后在执行期间，动态改变通过一系列预定义的事件集触发，最后体系结构选择一个合适的配置来执行系统的重构。它是基于一些预定义的配置来进行实施的。下图所示描述了由事件触发改变的适应性动态特征。

5.5 动态体系结构特征

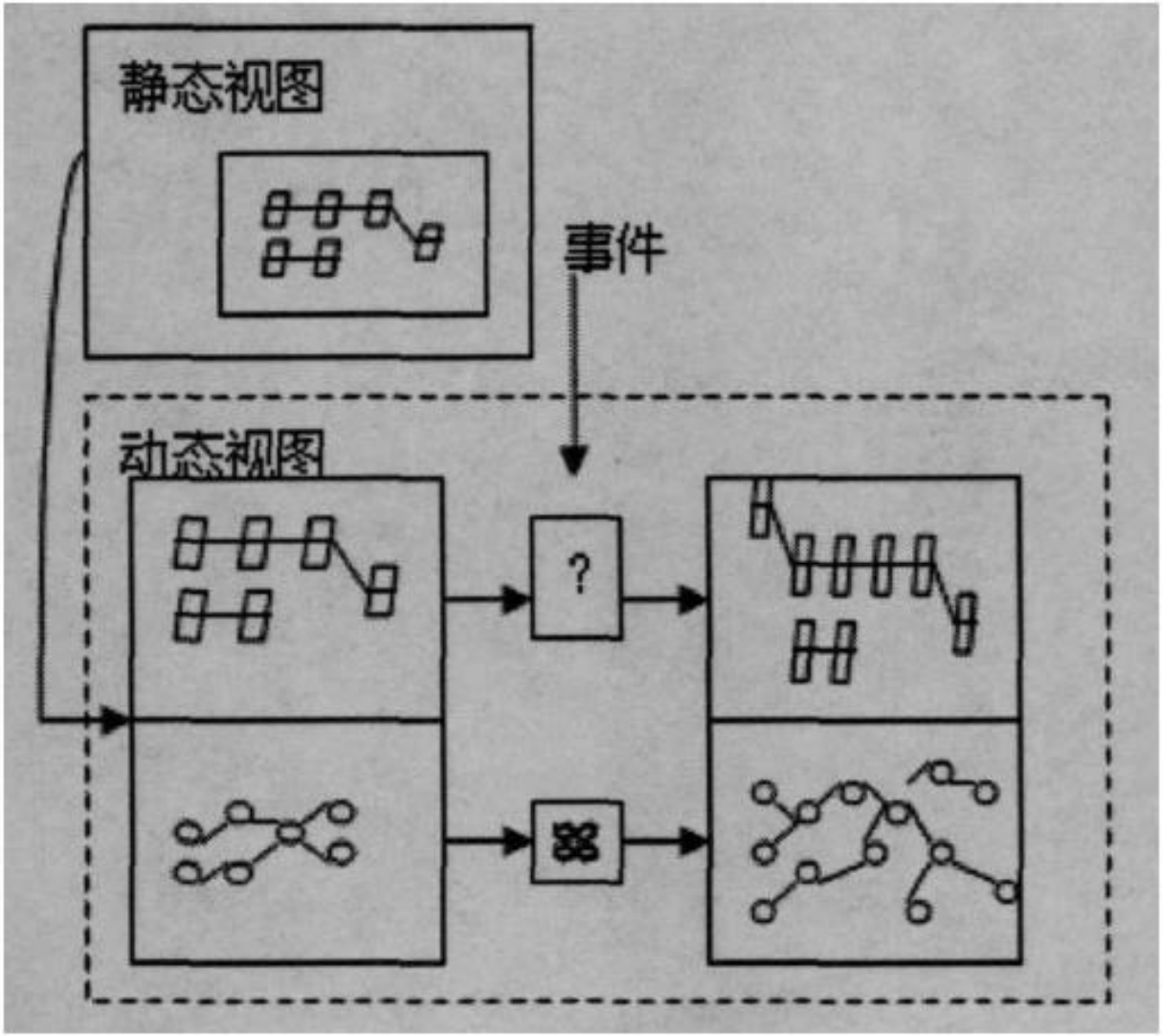


5.5 动态体系结构特征

3、智能型动态特征

智能型动态特征是用一个有限的预配置集来移除约束。它描述的是一个具有智能型动态特征的应用程序体系结构。相比适应性体系结构特征而言，智能性体系结构特征改善了配置选择转变的功能。智能型动态特征是从动态构造候选配置中选择，与适应性特征（从一系列固定的配置中选择一个适应体系结构的配置）不同。由于智能型特征的复杂性，在实际的软件体系结构中并没有太多的系统能够用到这种方法。

5.5 动态体系结构特征



谢谢大家！

再见！