基于π演算的动态软件架构研究

任洪敏[1] 张敬周[2] 杨志应[1]

(上海海事大学计算机系 上海 200135)[1] (复旦大学计算机科学系软件工程实验室 上海 200433)[2]

摘 要 动态软件架构(Dynamic Software Architecture)是架构领域的研究热点。分析了动态软件架构建模应关注的问题,基于π演算提出了动态软件架构形式化建模的方法,给出了动态软件架构语义的推导算法,奠定了动态架构行为分析和仿真的基础,并能对动态架构的诸多要素进行建模,包括动态演化的起因、时间、操作、非瞬时特性、断点继续执行等。

关键词 软件架构,动态软件架构,架构描述语言,π演算

中图法分类号 TP311 **文献标识码** A

Research on Dynamic Software Architecture Based on π-Calculus

REN Hong-min[1] ZHANG Jing-zhou[2] YANG Zhi-ying[1]

(Department of Computer Science and Technology, Shanghai Maritime University, Shanghai 200135, China)[1]

(Department of Computer Science and Technology, Fudan University, Shanghai 200433, China)[2]

Abstract Dynamic software architecture is one of the important research subjects in software architecture. This paper discussed issues in modeling and analyzing dynamic software architectures, proposed a modeling method of dynamic software architectures based on π-calculus, and developed an algorithm for reasoning about the semantics of dynamic architectural configuration. The π-calculus based modeling method is able to specify many aspects of dynamic architecture, including cause, time, operations and non-instantaneous change, etc.

Keywords Software architecture, Dynamic software architecture, Architecture description language, π-calculus

1 引言

面向服务的架构(Service-oriented Architecture)、移动计算、网格计算、业务关键系统(Mission critical systems)等都需在系统运行时进行动态演化。而软件架构(Software Architecture)是系统高层结构,由构件和连接件配置而成,提供了研究、设计和实现动态软件演化的良好视图。但是,现今软件架构的研究,主要关注静态架构,对动态架构支持甚少。因此,动态软件架构(Dynamic Software Architecture)成为架构领域的研究热点[1,2]。

π演算是90年代计算机并行理论领域重要的并发计算模型,能够描述结构不断变化的并发系统[3]。基于π演算,提出了软件架构描述语言πADL[4]。本文基于πADL的研究工作,分析了动态架构建模需要关注的问题,提出了动态架构的形式化规约方法和其语义的推导算法。该方法的特点是运用统一的形式化方法π演算,分别描述架构的计算行为和动态演化行为,并且这两类行为相互交互和制约,从而能够对动态架构的诸多因素建模,包括动态演化的起因、时间、操作、非瞬时特性、断点处继续执行等,以支持系统安全、完整的演化。

本文第2节是πADL架构描述语言简介;第3、4节介绍πADL动态架构建模的方法、语义推导算法及其验证方法;第5节是相关工作的介绍和比较;最后是结论并介绍下一步的研究工作。

2 πADL架构描述语言简介

πADL基于多元π演算(Polyadic π-Calculus)描述软件架构,它遵循Wright[5]描述软件架构的框架,独立描述构件、连接件和系统配置。πADL运用π演算的进程描述构件端口、连接件角色和它们内部的计算行为。πADL在π演算的语法元素和架构行为描述所关注的内容之间建立映射,提供构件、连接件的控制行为、观察行为和内部行为与π演算语法元素的对应关系,从而给π演算语法元素赋予了架构级的语义。

πADL的具体语法规定和建模方法参见文献[4],下面通过对客户-服务器架构风格中客户构件的描述,简要说明πADL描述架构元素行为的方法。其中Π为πADL引入的简便标记,表示非确定选择,其语义为 $P\Pi Q=\tau.P+\tau.Q$。

Component Client

Port Cp=$\overline{\text{request}}$. reply. CpΠ0

Computation=τ_internalCompute. $\overline{\text{Cp\_request}}$. CP_reply. ComputationΠ0

客户构件Client具有一个端口Cp,它的行为交互协议是发出服务请求$\overline{\text{request}}$,收到回答 reply 后,内部非确定地选择

到稿日期:2008-10-24 返修日期:2009-01-22 本文受上海市教委创新基金项目(09YZ224),上海市教委科学基金项目(2008096)联合资助。

任洪敏 博士,主要研究方向为软件体系结构、CBSE、形式化方法,E-mail:hmren@cie.shmtu.edu.cn;张敬周 博士生,主要研究方向为构件技术、软件复用。

继续执行进程 Cp 或决定结束服务请求而执行进程 0。Client 的计算行为是首先进行内部计算 τ_internalCompute，然后通过端口 Cp 发出服务请求$\overline{Cp_request}$，等到端口 Cp 得到回答 Cp_reply 后，Computation 继续进行该过程或决定结束。

运用端口名称修饰、限定通道名称，防止各个端口的通道同名，因此$\overline{Cp_request}$和 Cp_reply 表示端口 Cp 发出服务请求$\overline{request}$和得到回答 reply。

3 πADL 动态软件架构建模

3.1 动态架构建模的问题

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[6]。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：

(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时，方能允许体系结构动态调整。

(3) 体系结构演化的非瞬时性(Non-instantaneous Changes)。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。在这个过程中，如果相关构件和连接件继续执行，将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处继续开始执行。

(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。

3.2 πADL 动态架构建模的方法

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。

(1) 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、连接件的 Computation 进程、Glue 进程相互交互，总体控制体系结构的动态演化。

πADL 进行动态体系结构建模的具体方法如下。

(1) 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，表明表达构件或连接件内部发生了错误或异常，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。

(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时，处于一个安全状态，能够开始接受动态配置指令，进行系统演化。

(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。

(4) 对构件从断点开始继续执行的能力进行建模。根据第 4 节的 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) 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_Attach_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，故用 τ 作为前缀。

(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_Configuror，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，并通过专门通道 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。Dynamic_Configuror 进程的控制结构基本如下。

Dynamic_Configuror::=Action$^+$. Dconfiguror

DConfiguror::=$\sum$X_Config(y). Rule + En_Config(y). Rule

Rule::=($\sum$[y=controlname]. Subrule)

SubRule::=$\overline{X\_Config}$⟨BeginOk⟩$^+$. Action$^+$. $\overline{X\_Config}$⟨EndOk⟩$^+$. DConfiguor

Action::= τ_New_Iname_Tname | τ_Delete_Iname
| τ_ Attach_M_N_TO_I_J | τ_ Detach_M_N_From_I_J

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

Dynamic_Configuror 进程的直观语义是首先执行系列配置行为 Action，创建一个初始运行系统。然后通过 X_Config、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。SubRule 中，首先通过 X_Config 通道输出系列控制名字 BeginOK，发出动态配置开始指令，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。

3.3 πADL 动态架构建模的语义

3.1 节和 3.2 节给出了动态架构配置的方法和其直观语义。下面给出动态架构配置的形式语义。即根据构件、连接件类型定义和其行为描述、动态配置进程 Dynamic_Configuror，构造、推导、描述整个系统行为的 π 进程，该 π 进程按赋予的直观语义执行，协调控制系统的演化行为和构件、连接件实例的计算行为。

π 演算自身提供进程合成和动态建模的能力，奠定了动态架构配置语义推导的良好基础。因此，关键要点是如何基于 π 演算编码和表达构件、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：

(1) 构件、连接件的动态创建和删除。分析 Dynamic_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动态配置进程并发执行，实现构件的动态创建，并与动态配置进程交互，从而控制架构的动态演化。πADL 通过让删除的构件和连接件进程在系统中不能找到对偶的名字而不能继续执行，表达它们的删除。

(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 Dynamic_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，Dynamic_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，实现架构的动态变化。

下面给出动态架构行为的推导算法。该算法调用 πADL 静态架构行为的推导算法，具体参见文献[5]。动态架构的初始和演化基本都由动态配置进程控制，因此，算法的骨架是分析、操作配置进程 Dynamic_Configuror，并整合构件、连接件的实例进程，构造得出代表整个系统的 π 演算进程。

算法(动态架构行为推导算法)

输入：构件、连接件定义、动态架构配置进程 Configuror

输出：描述动态架构行为的 π 进程

步骤：

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).[z=EndOk]之间，插入连接件交互通道的输入行为 Config($\vec{x}$)，其中 $\vec{x}$ 为该连接件所有交互通道名字的序列，形成：Config(y).[y=BeginOk].Config($\vec{x}$).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。

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 Configuror::=Action$^+$.Dconfiguror 中的 Action$^+$ 部分，根据 Action$^+$ 中的构件、连接件创建、连接配置情况，调用架构静态配置行为的推导算法，得到表达初始配置的进程 Pinitial。并运用 Pinitial | Dconfiguror 替换 Dconfiguror，得到 Action$^+$.(Pinitial | Dconfiguror)。该步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。

第三步　分析 Dconfiguror 部分的动态配置行为，动态改变系统架构。即分析语法元素 SubRule::=$\overline{\text{X\_Config(BeginOk)}}^+$.Action$^+$.$\overline{\text{X_Config(EndOk)}}^+$.DConfiguor 中的 Action$^+$ 部分，具体执行如下操作：

(1) 动态创建新的构件和连接件。根据 Action$^+$ 部分中构件、连接件的创建信息、新建连接件的端口角色连接情况，调用架构静态配置行为的推导算法，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 X1，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$\overline{\text{X1_Config}}\langle\vec{y1}\rangle\cdots\overline{\text{Xn_Config}}\langle\vec{yn}\rangle$，其中$\vec{yi}$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。

(3) 连接件删除的处理。设在 Action$^+$ 部分删除连接件 C1，C2，Cm，向它们输出代表空的 Void 交互通道，得到$\overline{\text{C1_Config}}(\vec{v1})\cdots\overline{\text{Cn_Config}}(\vec{vn})$，其中$\vec{vi}$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(1)、(2)、(3)得到的信息，修改 Subrule 元素如下：在 Action$^+$ 之后插入行为：$\overline{\text{X1\_Config}}\langle\vec{y1}\rangle\cdots\overline{\text{Xn\_Config}}\langle\vec{yn}\rangle\overline{\text{C1\_Config}}(\vec{v1})\cdots\overline{\text{Cn\_Config}}(\vec{vn})$，并用(Pcreated | DConfiguror)代替 DConfiguror。结果得到的进程就是描述该动态架构行为的 π 进程。

该算法的时间花费主要是第二步、第三步中构件、连接件实例的生成和连接件初始生成时端口角色的静态连接及后来的动态连接，故其最坏时间复杂度为 $O(n + k * m)$，其中 n 为整个动态配置中生成的所有构件实例中事件个数，m 为所有连接件实例中事件的个数，k 为所有连接件的角色数目之和。

4　πADL 动态架构建模实例

设计一个动态配置、具有容错功能的客户-服务器系统，运用 πDL 描述该系统的架构行为和动态配置方案，整个行为规约采用 π 演算的分析工具 MWB(The Mobility Workbench)进行交互、仿真执行，以验证 πADL 的动态架构建模方法。

动态客户-服务器架构具有两个服务器：主服务器和备份服务器。主服务器计算能力强，它出现错误后，客户自动切换到备份服务器，主服务器的错误修复后，重新运行主服务器。该动态客户-服务器架构的 πADL 建模具体规约如下。

```
Style Fault_Tolerant_Client_Server
  //客户的规约同于第二节中 Client_Server 中的 Client。
  Component PrimaryServer
    Port Sp=Request.Reply.Sp+0
    Computation=(Sp_Request.τ_InternalCompute.Sp_reply.Com-
                putation∏Config⟨Error⟩.(0+Config(y).[y=
                FixOk]Computation))+0
  Component BackupServer
    Port Sp=Request.Reply.Sp+0
  Computation=Sp_Request.τ_InternalCompute.Sp_reply.(Config
                (y).[y=BeginOk]Config(z).[z=EndOk]Compu-
                tation+Computation)+0
  Connector FTlink
      Role C=request.reply.C∏0
      Role S=Request.Reply.S+0
      Glue=C_request.S_request.S_reply.C_reply
          .(Config(y).[y=BeginOk]Config(z).[z=EndOk]
          Glue+Glue)+0
EndStyle
```

Configuration Dynamic-Client-Server

Rule1 = $\overline{FL\_Config}$⟨BeginOk⟩. τ_Detach_PS_SP_From_FL_S. τ_Attach_BS_SP_To_FL_PS. $\overline{FL\_Config}$⟨EndOk⟩. DConfiguror

Rule2 = $\overline{FL\_Config}$⟨BeginOk⟩. $\overline{BS\_Config}$⟨BeginOk⟩. $\overline{PS\_Config}$⟨FixOk⟩. τ_Detach_BS_SP_From_FL_S. τ_Attach_PS_SP_To_FL_PS. $\overline{FL\_Config}$⟨EndOk⟩. $\overline{BS\_Config}$⟨EndOk⟩. DConfiguror

EndConfiguration

容错客户-服务器系统的定义和静态客户-服务器定义基本相同，但其间增加了用于系统动态配置的控制行为。主服务器 PrimaryServer 不能随时产生错误，只能在完成一次事务之后，决定是否发生错误，并通过专门通道输出该错误信息($\overline{Config}$⟨Error⟩)，请求动态配置，然后能够从通道 Config 等待重新启动信息(Config(y). [y=FixOk])并继续执行计算。

备份服务器 BackupServer 和容错连接件 FTLink 同样只能在完成一次事务之后，处于安全状态时，才根据外部情况决定是接受动态配置(Config(y). [y=BeginOk])，还是继续执行。

配置进程 Dynamic_Configuror 首先生成构件、连接件实例，并连接客户构件 C、主服务器构件 PS、连接件 FL，从而得到初始系统。然后等待 PS 发生错误(PS_Config(y). [y=Error])，或者等待环境给出构件 PS 修复信息(En_Config(y). [y=Ok])。当错误发生后，Configuror 向连接件发出动态配置开始指令($\overline{FL\_Config}$⟨BeginOk⟩，从连接件 FL 上撤下 PS，换上备份构件 BS(τ_Attach_BS_SP_To_FL_S)，然后向连接件 FL 发出动态配置结束指令$\overline{FL_Config}$⟨EndOk⟩，重新回到监控状态。当 Dynamic_Configuror 接受到环境给出的修复指令后，行为基本同上，只是换上了主服务器 PS。两者不同的是，在换上 BS 时，不需要对其发出动态配置开始指令，等待它处于安全状态，因为 BS 随时备用。

5 相关研究工作

动态架构的研究首先体现在将动态配置功能集成在 ADL 中。为开发能随着环境和需求变化而动态自适应的系统，Dowling 等人设计了 K-Component 框架元模型[7]，其运用配置图描述软件架构和演化，虽然直观，但并不能严格和全面地表达演化的行为和语义。Rapide[11] 基于偏序事件集(Partially Ordered Event Sets，称为 Posets)描述构件的计算行为和它们之间的交互行为，但 Rapide 把构件之间的连接机制和系统的配置融为一体，这导致它不能独立、方便地描述系统的动态配置。文献[8]基于 Wright 架构描述语言和通信顺序进程进行描述进程行为的方法，对软件架构动态演化中构件的行为进行了分析和研究，在构件替换之前分析原构件和新构件的行为特性，在演化前确认构件的行为一致性，从而保证动态升级过程的正确性和合法性，但该方法主要对基于行为协议的替换性进行分析，未对构件演化过程中多种因素进行控制和分析。

Wright[5] 运用进程代数 CSP 对架构建模，但难以规约动态架构。Robert Allen 运用标签事件(Tagged events)技术[12] 扩展 Wright 的研究工作，提供了运用 CSP 对动态架构建模的方法[14]，但因为 CSP 固有的静态特性，动态架构的形式语义变得复杂且难以理解。并且动态配置行为和计算行为分离，但又不彻底。因其端口、角色规约中出现用于动态配置的控制事件，导致兼容性检测变得复杂。

LEDA[9] 基于 π 演算描述架构，但它重在构件交互接口的兼容性检测、构件行为的扩充和精化(Refinement)研究。文献[10]同样基于高阶多型 π 演算理论，提出了动态体系结构描述语言 D-ADL，运用高阶多型 π 演算中的抽象类型、进程和通道描述构件、连接件和体系结构风格，并将计算行为和动态演化行为进行了分离。该方法重在给出架构动态演化的语义，但对架构演化的要素约束缺乏考虑。π 演算与 CSP 同为进程代数，πADL 借鉴 Wright 提供的架构建模框架和思想，具备了 Wright 的优点，它利用 π 演算的动态建模能力，克服了 Wright 描述动态架构的不足。

结束语 基于 π 演算软件架构形式化规约语言 πADL，本文分析了动态软件架构建模时应考虑的问题，提出了一个动态架构建模的形式化方法，并给出了它的形式化语义的推导算法。该方法运用专门的动态配置进程规约架构动态变化，动态配置信息局部化。并且在配置进程和系统的计算行为之间建立交互关系，从而能够系统地对动态架构的诸多要素建模，确保系统完整地演化。

进一步的研究工作包括基于 πADL 的动态架构演化框架的研究与开发、πADL 动态架构演化行为分析、仿真与监控等。

参 考 文 献

[1] 梅宏，申峻嵘. 软件体系结构研究进展[J]. 软件学报，2006，17(6)：1257-1275

[2] SUN Chang-ai, JIN Mao-zhong, LIU Chao. Overviews on Software Architecture Research[J]. Journal of Software, 2002, 13(7): 1228-1237

[3] Milner R. Communicating and Mobile Systems: the π-Calculus [M]. Cambridge: Cambridge University Press, 1999

[4] Ren Hongmin. Research on Software Architectural Formalism Based on π Calculus [D]. Fudan University, Shanghai, 2003

[5] Allen R J. A Formal Approach to Software Architecture [D]. Carnegie Mellon University, Pittsburgh, 1997

[6] Oreizy P. Issues in Modeling and Analyzing Dynamic Software Architectures[C]// Proceedings of the International Workshop on the Role of Software Architecture in Testing and Analysis. Marsala, Sicily, Italy, 1998: 98-101

[7] Dowling J, Cahill V, Clarke S. Dynamic software evolution and the k-component model [C]// Northrop L, Vlissides J, eds. Workshop on Software Evolution, Conf. on Object-oriented Programming Systems, Languages, and Applications 2001. New York: ACM Press, 2001

[8] 黄崇德，彭鑫，赵文耘. 体系结构动态演化中的构件行为分析[J]. 计算机工程与应用，2007，43(10)：87-92

[9] Canal C, Pimentel E, Troya J M. Compatibility and inheritance in software architectures[J]. Science of Computer. Programming, 2001, 41: 105-138

[10] 李长云，李赣生，何频捷. 一种形式化的动态体系结构描述语言[J]. 软件学报，17(6)：1349-135

[11] Luckham D C, Vera J. An event-based architecture definition language [J]. IEEE Transactions on Software Engineering, 1995, 21(9): 717-734

[12] Allen R J, Douence R, Garlan D. Specifying and Analyzing Dynamic Software Architectures[C]// Proceedings of the Fundamental Approaches to Software Engineering. Lisbon, Portugal, 1998: 21-37