

# Bigraph 理论在自适应软件体系结构上的应用

常志明 毛新军 齐治昌

(国防科学技术大学计算机学院 长沙 410073)

**摘 要** 现有的软件体系结构形式化方法对体系结构的动态性、自适应性支持有限,并不能很好地验证系统演化过程中的一致性、完整性等动态特征. Bigraph 理论融合了  $\pi$  演算和移动 Ambient 演算的优势,重点强调计算的位置和连接两方面因素,具有较为完整、可扩展的理论框架. 这使得 Bigraph 不仅在概念上能够满足现有自适应软件对结构和行为的需求,而且还提供了直观、普适的表达能力. 文中简要介绍了 Bigraph 的基本概念和现状,利用 Bigraph 理论对自适应软件体系结构进行了形式化规约,分析和验证了系统动态演化的性质,并探讨了 Bigraph 理论在自适应软件体系结构形式化方面的优势和拓展方向.

**关键词** Bigraph; Bigraph 反应系统; 自适应软件; 软件体系结构; 形式化方法

**中图法分类号** TP311

**DOI 号**: 10.3724/SP.J.1016.2009.00097

## Applying Bigraph Theory to Self-Adaptive Software Architecture

CHANG Zhi-Ming MAO Xin-Jun QI Zhi-Chang

(School of Computer Science, National University of Defense Technology, Changsha 410073)

**Abstract** Existing theories of mobile and concurrency calculi for dynamic software architecture can not provide powerful support for evolutionary properties of self-adaptive software. Under this circumstance, Bigraph is based on a graphical model of mobile computation that emphasizes both locality and connectivity and has a complete and extensible theory framework. Therefore, Bigraph can provide a sound concept and intuitive, pervasive expression for self-adaptive software architecture. This paper introduces Bigraphical theory and current research, describes self-adaptive software architecture formally, analyzes and verifies some properties of dynamic evolution, and then discusses some promising directions of formal self-adaptive software architecture.

**Keywords** Bigraph; BRS; self-adaptive software; software architecture; formalization

## 1 引 言

近年来,随着 Internet 向社会各角落的渗透式扩张,以普适计算、网格计算、网构软件等为典型代表的计算模式迅速发展,软件系统所面临的计算环境变得开放、多元和易变. 为适应运行时刻计算环境中网络、设备、资源等的变化和用户需求的改变,人们

对自适应软件的需求日益增长. 自适应软件通常包括软件本身和外部环境,并能够在运行时刻通过调整自身结构或者行为以适应需求和环境的变化. 同时,自适应软件的复杂性也带来了越来越多的挑战<sup>[1]</sup>; 如何理解、描述和建模这类应用,什么样的理论适合分析和验证这类演化系统的性质,现有的理论需要扩展哪些要素以支持这类复杂系统的开发等等.

当前,一些研究者开始从软件体系结构的角度

对系统的自适应演化提供支持,更加关注在运行时刻显式地维护体系结构信息的重要性.相比其它的自适应方法,基于体系结构的自适应软件具有一些显著的优势:(1)体系结构作为抽象模型可以为运行期间的自适应软件提供全局视点,这有助于对系统的理解和分析;(2)由于自适应软件在运行时刻会根据环境和用户需求的变化做出相应的调整,这些调整很可能会违背设计原则,破坏系统的一致性和完整性,而体系结构模型有助于显式地刻画这些约束条件,有助于分析系统在演化过程中的性质<sup>[2]</sup>;(3)基于体系结构的设计可以使自适应软件具有松耦合性,这使得系统中的一部分更改不会影响其他部分,这可以对系统进行灵活的配置,有利于系统的自身调节<sup>[3]</sup>.

在传统软件体系结构技术的基础上,自适应软件体系结构更加关注环境变化所引起的系统结构和行为的改变.自适应软件体系结构同时强调环境、结构和行为以及这三者之间的关系,而且需要确保系统在演化过程中的有效性.形式化方法可以为自适应软件体系结构提供准确的表示方法和工程化规则的指导,能够分析和推理系统的演化,有助于开发自适应软件的辅助工具.目前,自适应软件体系结构的形式化方法大致可分为两类:体系结构描述语言(ADL)和通用的形式化方法,其中 ADL 包括 Wright、Darwin、 $\pi$ -ADL 等,通用形式化方法包括 CHAM、图等.虽然这些方法取得了一些成果,可以验证体系结构中的一些性质,但是这些形式化方法虽然强调体系结构的动态性,但不能很好地检验体系结构演化过程中的一致性、完整性等动态特征.而且由于它们各自表达能力所限,在理论方面和描述方面都面临着一些问题:

(1)理论方面.较为合用的形式化理论基础是一种软件技术达到学术成熟的标志之一,但已有的形式化方法一般只注重自适应软件的某一方面,或者是交互或者是拓扑结构.而这些形式化方法都有各自的特点,难以对自适应软件的性质提供完整的分析和验证.

(2)描述方面.自适应软件体系结构需要对环境、结构和行为及三者的变化提供形式化规约.对环境进行感知并根据环境的变化做出演变是自适应软件的基础,而现有 ADL 和通用方法很少对环境及环境的变化进行描述,因此需要引入环境模型表示空间、连接、事件等基本因素.此外,这些形式化方法的描述都过于符号化,缺乏直观表达,并将结构、行为和演化视图混杂在一起,难以描述系统的动态

演化.

Bigraph 则融合了  $\pi$  演算和移动 Ambient 演算的优势,在概念上能够满足自适应软件对并发交互和拓扑结构变化的需求.利用 Bigraph 理论及其应用可以对自适应软件体系结构的性质进行分析和验证,更适于开发出安全的自适应软件.此外,Bigraph 提供了图形化的表示方法,而且还具有完备的公理系统.我们在研究中发现 Bigraph 可以很好地与自适应软件体系结构中的概念相匹配.因此,Bigraph 理论可以为自适应软件体系结构的形式化规约和性质验证提供有效的解决途径.

本文简单介绍 Bigraph 的概念和应用现状,研究 Bigraph 理论在自适应软件体系结构上的应用.第 2 节介绍 Bigraph 的基本概念,分析 Bigraph 应用现状;第 3 节通过 Bigraph 理论及应用为自适应软件体系结构提供形式化规约;第 4 节利用 Bigraph 理论对自适应软件体系结构中的性质进行分析和验证;第 5 节与其它形式化方法进行比较;最后总结全文,展望未来的工作.

## 2 Bigraph 的简介

Bigraph 是由 Milner 等学者在 2001 年提出的一种基于图形的形式化理论工具,旨在强调计算(物理或虚拟)的位置和连接. Bigraph 理论最主要的两个目标是:(1)对普适计算系统进行建模;(2)为现有的移动和并发理论建立统一的元模型<sup>[4]</sup>.

在静态结构方面,一个 Bigraph 可以分解为两个(Bi)子图(graph):位置图(place graph)和连接图(link graph),由此得名 Bigraph.位置图用以表示各个计算节点的所在位置,节点之间允许相互嵌套;连接图则忽略节点之间的嵌套关系,只表示节点之间的连接关系.位置图和连接图是从两个不同的角度对同一个 Bigraph 观察所得到的结果,因此它们具有相对的独立性.我们结合图 1 对相关概念加以介绍:每个节点都分配了一个控制(如 PC, PDA, ROOM),控制可以描述该节点拥有多少个端口(port),每个端口可对应一个连接;控制还规定了该节点是原子节点还是复合节点,如果是复合节点,还需说明它是活跃的(active)还是不活跃的(passive).如果是活跃的,则节点内部允许施加反应规则;否则不允许内部具有反应.端口也可以通过名字来表示,比如外部名  $\{y_0, y_1, y_2\}$  和内部名  $\{x_0, x_1\}$ . 一个 Bigraph 可以包含一定数目的区域(region,最外层的虚线矩形)和地点(site,灰矩形).

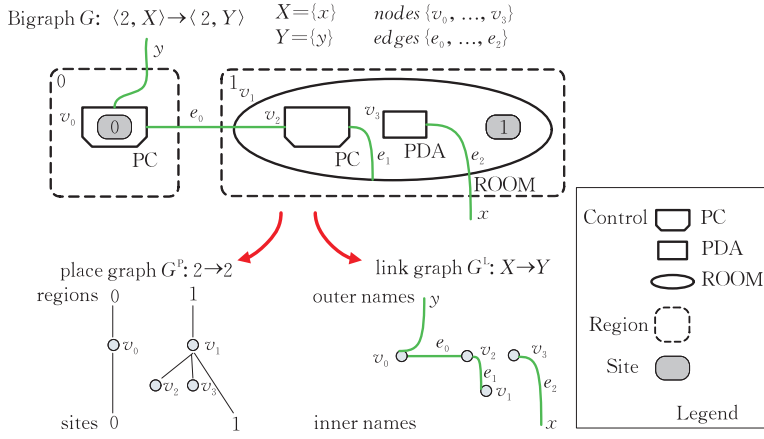
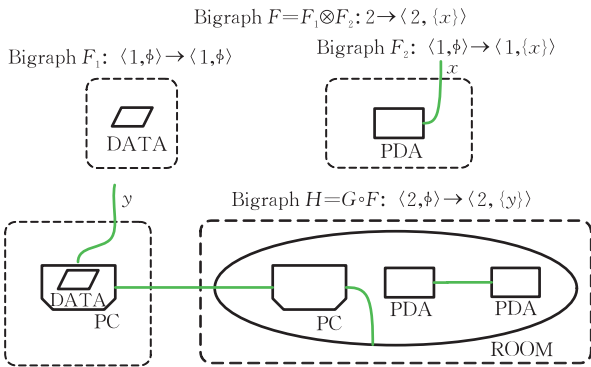


图 1 Bigraph 分解为位置图和连接图

在数学基础方面,位置图可视为态射,其对象是序数集.位置图  $G^P: m \rightarrow n$  表示  $G^P$  具有  $m$  个地点和  $n$  个区域.同样,连接图则是对象为名字集的态射.连接图  $G^L: X \rightarrow Y$  表示  $G^L$  具有内部名  $X$  和外部名  $Y$ .因此,Bigraph 是位置图和连接图的合成图,其对象是二元组:序数集和名字集,即 Bigraph  $G: \langle m, X \rangle \rightarrow \langle n, Y \rangle$ . Bigraph  $F$  和  $G$  的合成操作  $F \circ G$  的前提是  $G$  的值域和  $F$  的定域是相同的,相当于先将  $G$  的区域驻留到  $F$  的地点中,然后将  $G$  的外部名与  $F$  的内部名连接到一起,如图 2 中  $H = G \circ F$ . 扩展乘积操作  $F \otimes G$ ,是将  $F$  和  $G$  “从左到右”并列在一起,如图 2 中  $F = F_1 \otimes F_2$ .

图 2 Bigraph 的扩展乘积  $\otimes$  和合成  $\circ$  操作

在动态性方面,Bigraph 可以通过反应规则  $(r, r')$  对自身进行重配,即从一个 Bigraph 变化到另一个 Bigraph.一条反应规则包括了反应物和生成物两个部分,并可带有任意多个参数,其中反应物、生成物都是 Bigraph.生成物和反应物的参数具有映射关系,这使得反应物中的参数可以被复制或丢弃.反应规则可以根据具体的应用自由地加以定义.如果 Bigraph 与反应物匹配,则该 Bigraph 就将其内部的反应物替换为生成物.例如,图 3 的反应规

则表示在某个 ROOM 中的 PC 可以和一台远程的 PC 相连,在该 ROOM 中的 PDA 可以将内部 PC 作为网关,将远程 PC 中的 DATA 拷贝到 PDA 内部.

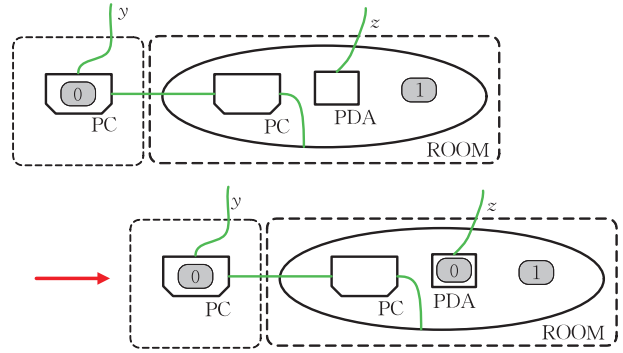


图 3 反应规则

每个 Bigraph 都可以利用基本的图元(placings、wirings 和 ions),合成或扩展乘积操作的组合形成 Bigraph 范式<sup>[5]</sup>.这些基本的元素以及操作如表 1 所示.我们可以将 Bigraph  $G, F, H$  以及应用反应规则后的  $H'$  分别表示为

$$\begin{aligned}
 G &= /m./n./z.(PC_{ym} \parallel \\
 &\quad ROOM_n(PC_{mn} | PDA_z | -_1) | z/x), \\
 F &= DATA \parallel PDA_x, \\
 H &= /m./n./z.(PC_{ym}(DATA) \parallel \\
 &\quad ROOM_n(PC_{mn} | PDA_z | PDA_z)), \\
 H' &= /m./n./z.(PC_{ym}(DATA) \parallel \\
 &\quad ROOM_n(PC_{mn} | PDA_z(DATA) | PDA_z)).
 \end{aligned}$$

一个 Bigraph 反应系统(Bigraphical Reactive System, BRS)则包含了 Bigraph 以及一组反应规则  $\mathcal{R}$ ,其静态和动态性质的数学基础是范畴论,利用其中重要的数学性质 RPO(Relative PushOuts)可以证明:在 Bigraph 反应系统中 Bigraph 之间的互模拟(bisimilarity)关系实际上是同余的(congru-

表 1 Bigraph 表达式中项的涵义

项	涵义
$U \parallel V$	并列区域
$U   V$	合并区域
$U \circ V$	合成
$U \otimes V$	扩展乘积
$U(V)$	嵌套, $U$ 包含 $V$
$K_{\vec{x}}$	离子, 节点的控制 $K$ 有外部名 $\vec{x}$
$1$	空的区域
$-i$	地点, 索引为 $i$
$/x.U$	将 $U$ 的外部名 $x$ 改为一条边
$x/y$	换名, 用外部名 $x$ 替换内部名 $y$

ence)<sup>[4]</sup>. 因此, Bigraph 不仅具有图形化的表现形式、灵活的语义定义方式, 还具备良好的数学基础和可扩展能力. 上面所介绍的 Bigraph 是纯 (pure) Bigraph, 类型 (sorting) Bigraph<sup>[6]</sup>、绑定 (binding) Bigraph<sup>[4]</sup>、局部 (local) Bigraph<sup>[7]</sup> 等是对纯 Bigraph 的扩展, 比如类型 Bigraph 用以限定节点之间的嵌套关系和连接关系, 绑定 Bigraph 可以表示  $\pi$  演算, 局部 Bigraph 可以表示  $\lambda$  演算. 目前,  $\pi$  演算、 $\lambda$  演算、CCS、Petri 网、移动 Ambient 和 HOMER 已经被描述成 BRS<sup>[4,7-9]</sup>. 此外, Bigraph 理论也可以描述上下文感知 (context-aware) 系统<sup>[10]</sup>, 提供逻辑系统 BiLog<sup>[11]</sup>. 丹麦的 IT 大学正在开发基于 Bigraph 的普适计算语言 (Bigraphical Programming Language, BPL)<sup>[12]</sup>, 该语言很有可能代替现有的 UML 等建模语言, 成为普适计算时代主流的建模及开发语言.

### 3 自适应软件体系结构的 Bigraph 模型

本节结合我们的研究工作, 利用 Bigraph 理论及现有研究成果对自适应软件体系结构的形式化方面加以探讨. 文献[13]指出, 对于动态体系结构的形式化应该包括结构、行为及其变化. 因此, 自适应软件体系结构的形式化方法应该涵盖环境、结构、行为以及三者之间的变化和关系, 因此需要利用 Bigraph 描述外部环境和体系结构自身, 其中体系结构还包括了结构和行为两个方面.

#### 3.1 结构方面

综合现有 ADL 中的概念, 体系结构在结构方面的核心概念包括构件、连接子、配置、端口、角色、内部表示和映射关系. 软件体系结构风格定义了一个系统家族, 即风格包括词汇表和一组约束. 词汇表中包含一些构件、连接子及其交互点的类型, 而约束

则表明了系统是如何将这些构件和连接子组合起来的. 我们利用类型 BRS 中的概念对体系结构及风格进行描述, 如表 2 所示. 从中可以看出体系结构与 Bigraph 在概念级别上可以很好地匹配.

表 2 体系结构与类型 BRS 概念的映射关系

自适应体系结构		扩展的类型 BRS	
结构	构件	扩展的 Bigraph	节点
	连接子		节点
	配置		边
	端口		端口
	角色		端口
	内部表示		地点
	映射关系		边
风格	构件类型	扩展的控制	控制
	连接子类型		控制
	端口类型		类型
	角色类型		类型
约束		类型约束条件	
体系结构操作		Bigraph 扩展操作	
重配		反应规则	

下面以三层 Client-Server 的体系结构风格为例进行说明. 考虑一个网络信息服务系统, 用户可以通过 *Core* 向分布的服务器发出服务请求. 在动态的网络环境中, 用户端和服务器的数目是不断发生变化的, 比如用户可以加入或离开, 服务器在没有请求服务的情况下可以断开与 *Core* 的连接. 因此, 这样的体系结构能够根据用户的请求数目而自适应地进行调整. 图 4 给出了该体系结构风格的一个具体的运行状态.

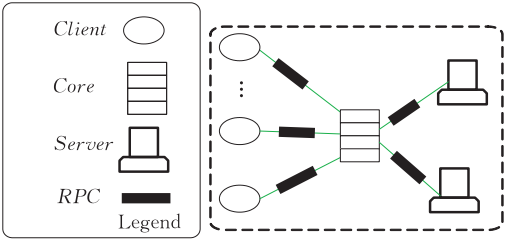


图 4 三层 Client-Server 风格实例的 Bigraph 描述

该例子的构件和连接子包括  $Client \{port \ request\}$ ,  $Server \{port \ reply\}$ ,  $RPC \{role \ caller, \ callee\}$ ,  $Core \{port \ in, out\}$ .

我们将具体的体系结构视为一个 Bigraph, 体系结构的变化操作包括增加、删除、替换构件或连接子, 配置端口和角色的连接等. 我们可以通过对 Bigraph 进行修改以实现这些体系结构操作, 其中  $A$  表示体系结构,  $U$  和  $V$  是  $A$  中的项,  $P$  表示构件或连接子,  $p$  表示端口,  $r$  是角色.

**定义 1(代换).** 令  $U, V$  和  $F$  是 Bigraph 元素,  $S_V^U F$  指用  $U$  替换任何在  $F$  中出现的  $V$ .

$Add(P, U(V), A) = S_{U(V)}^{U(V|P)} A$  // 将  $P$  添加到  $U$  中  
 $Remove(P, A) = S_P^1 A$  // 删除  $A$  中的  $P$   
 $Replace(P', P, A) = S_P^{P'} A$  // 用  $P'$  替换  $A$  中的  $P$   
 $Connect(p, r, A) = /x.^x/_p.r.A$  // 连接  $p$  和  $r$   
 $Disconnect(p, r, A) = S_{/x.^x/_p.r}^\epsilon A$  // 断开  $p$  和  $r$  的连接  
 $Rely(t, t', A) = /x.^x/_t.t'.A$  // 映射  $t$  和  $t'$   
 $Disrely(t, t', A) = S_{/x.^x/_t.t'}^\epsilon A$  // 断开  $t$  和  $t'$  的映射

比如,在这个例子中增加一个 Client,我们可以通过下面的操作序列加以描述:

```

Add a Client {
  Add(Client_request, S)
  Add(RPC_caller.caller, S)
  Connect(request, caller, S)
  Connect(in, callee, S)
}
  
```

此外,我们可以利用 BRS 中的反应规则表示结构变化,这样可以直观地表示体系结构操作的效果.例如,我们用图 5 的反应规则表示增加了一个 Client.

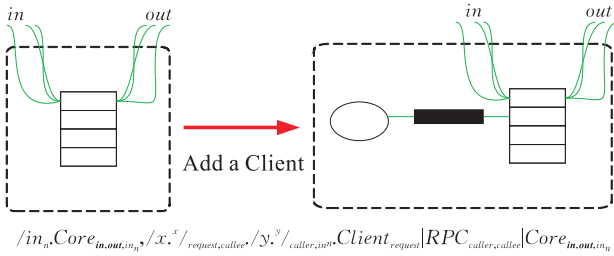


图 5 增加一个 Client 的反应规则

### 3.2 行为方面

在体系结构行为方面,现有的形式化方法大多利用  $\pi$  演算、图论等理论,正如前面所说 Bigraph 的目标之一就是为移动和并发理论提供统一的框架.因此,这些形式化方法可以通过 Bigraph 加以描述,即直接将这些行为规约转换为 Bigraph 模型.下面,我们将  $\pi$  演算的语法转换为绑定 Bigraph.

$behavior ::= prefix, behavior$

$| \text{if } boolean \text{ then } \{ behavior_1 \} [else \{ behavior_2 \}]$   
 $| \text{choose } \{ behavior_1 \text{ or } behavior_2 \text{ or } \dots behavior_n \}$   
 $| \text{variant assignment}$   
 $| \text{replicate } behavior$   
 $| \text{inaction}$

$prefix ::= \text{via face send value} | \text{via face receive value}$

语义解释,其中  $if, then, else, choose, inaction, replicate, send, get$  为不活跃控制.

$[ \text{if } boolean \text{ then } \{ behavior_1 \} \text{ else } \{ behavior_2 \} ]$   
 $= if \text{ boolean } (then ([behavior_1]) | else ([behavior_2]))$

$= \begin{cases} [behavior_1], & \text{if } boolean = true \\ [behavior_2], & \text{if } boolean = false \end{cases}$

$[ \text{choose } \{ behavior_1 \text{ or } behavior_2 \text{ or } \dots behavior_n \} ]$

$= choose ([behavior_1] | \dots | [behavior_n])$

$= [behavior_i] \quad i \in \{1, 2, \dots, n\}$

$[ \text{inaction} ] = inaction$

$[ \text{replicate } behavior ] = replicate [behavior]$

$= [behavior] | replicate [behavior]$

$[ \text{via } n \text{ send } v ] = send_{nv}$

$[ \text{via } n \text{ receive } v ] = get_{n(v)}$

对于构件或者连接子行为的变化,我们可以通过动态绑定机制中的  $join, quit, activate$  和  $deactivate$  操作加以表示<sup>[14]</sup>. 这样,构件和连接子可以动态地加入或者退出某些行为规约,并可以将行为规约置于活跃或者不活跃状态,从而在变化的环境中展示出不同的行为.动态绑定机制中行为规约的状态及操作如图 6 所示.

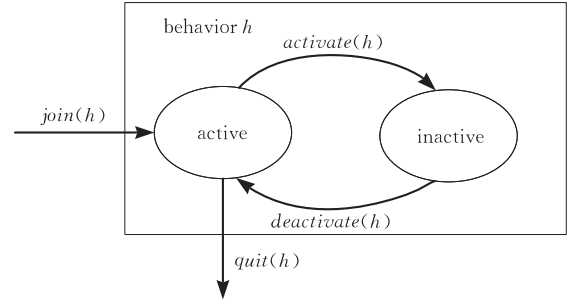


图 6 动态绑定机制

例如,如果我们在  $Core$  中增加了  $Authentication$  构件,那么  $Core$  的初始的行为规约是 Ordinarybehavior,认证行为规约为 Safetybehavior,具体描述如下:

Ordinarybehavior =  $\text{via in receive } n; \text{ inaction};$

$\text{via out send } n;$

Safetybehavior =  $\text{via in receive } n; \text{ if authentication } (n)$   
 $\text{then } \{ \text{via out send } n \} \text{ else } \{ \text{via in send FAILURE} \};$

而行为改变可以通过动态绑定操作加以说明,即  $quit(\text{Ordinarybehavior}); join(\text{Safetybehavior});$

$H' = join(behaviour) = H | behaviour,$

$H' = quit(behaviour) = S_{inactive(behaviour)}^1 H,$

$H' = activate(behaviour) = S_{inactive(behaviour)}^{behaviour} H,$

$H' = deactivate(behaviour) = S_{behaviour}^{inactive(behaviour)} H.$

### 3.3 环境方面

如 2.2 节介绍, Bigraph 具有普适的表达能力.因此,用户可以按照自己的方式给出环境的定义,比如 Bigraph 中的节点可以表示位置、实体、事件等,边可以表示连接、关系、触发条件等.我们也可以为



该例子引入环境信息:利用 *Container* 表示当前系统还可以接纳的 Client 数目(对应于内部圆圈的个数). 增加一个 Client 的反应规则变为图 7 所示的形式.

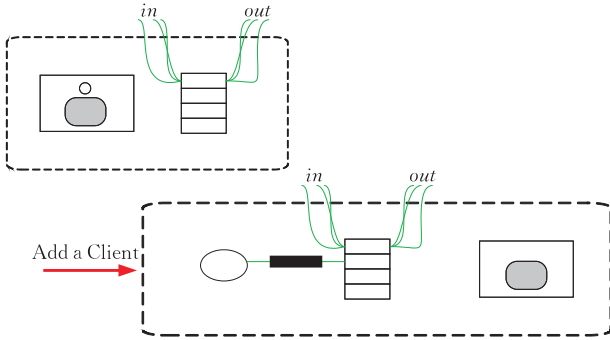


图 7 具有环境信息的反应规则

对于自适应软件而言,环境的变化会对体系结构产生影响,从而导致结构和行为的变化. 环境、结构和行为之间的变化关系可以表示为

//环境变化引起结构和行为变化  
 $\text{EnvChange} \rightarrow (\text{StructuralChange} \mid \text{BehavioralChange}) +$   
 //结构变化,其中  $op$  是体系结构操作  
 $\text{StructuralChange} ::= op + \mid \text{reaction rule}$   
 //行为变化,使用动态绑定操作  
 $\text{BehavioralChange} ::= (\text{join}(h) \mid \text{quit}(h) \mid \text{activate}(h) \mid \text{deactivate}(h)) +$

## 4 性质验证

在运行时刻实施这些变化的时候,需要保证这些变化不会破坏系统结构和行为的一致性和完整性,这是自适应演化能否实施的必要条件.

### 4.1 一致性

**定义 2.** 对于行为表达式  $BE_1$  和  $BE_2$ ,则称行为  $BE_2$  模拟行为  $BE_1$  (简记为  $BE_1 \leq BE_2$ ),如果下列条件满足其一:

- (1)  $BE_1 = BE_2$ ;
- (2) 对于反应规则  $(r, r')$ ,如果  $BE_1 \rightarrow BE'_1$ ,则存在  $BE'_2$ ,使得  $BE_2 \rightarrow BE'_2$  并且  $BE'_1 \leq BE'_2$ .

从定义 3 中,我们可以认为  $BE_1 \leq BE_2$  表示  $BE_2$  的行为能力或者与  $BE_1$  相同,或者强于  $BE_1$ . 也可以认为  $BE_2$  是对  $BE_1$  的扩展.

**定义 3.** 对于行为表达式  $BE_1$  和  $BE_2$ ,如果一定存在一个反应规则构成的序列  $tr$ ,使得  $BE_1 \mid BE_2 \rightarrow \text{inaction}$ ,则称  $BE_1$  和  $BE_2$  是相容的(简记为  $BE_1 \cong BE_2$ ).

**定义 4.** 对于端口  $p$  和角色  $r$ ,其行为表达式分别为  $BE_p$  和  $BE_r$ ,如果满足  $BE_r \cong p/r.BE_r$ ,则称

$p$  和  $r$  是相容的(简记为  $p \cong r$ ).

该定义表示它们的行为必须是相容的,即必须能够完成交互,不能出现类似死锁的情况.

**定义 5.** 对于端口  $p_1$  和  $p_2$ ,其行为表达式分别为  $BE_{p_1}$  和  $BE_{p_2}$ ,我们称  $p_2$  是对  $p_1$  的精化(简记为  $p_1 \leq p_2$ ),如果满足  $BE_{p_1} \leq p_1/p_2.BE_{p_2}$ .

为了使演化结束后新体系结构各元素间依然保持一致,我们对演化过程加以限制,使体系结构在保证一致性的基础上进行演化. 体系结构按一定操作进行演化,这些操作包括体系结构元素的增加、删除或更新,而由这些简单的操作组成复杂的演化. 如果对每个操作都保持了体系结构的一致性,则整个演化过程就保持了一致性,所以在执行每个基本操作时都要以某种条件为前提.

**定义 6.** 构件  $Comp$  是一致的,必须满足下列条件:

- (1) 对于  $Comp$  中任意的配置连接  $(p, r)$ ,满足  $p \cong r$ ;
- (2) 对于  $Comp$  中任意的映射  $(p_1, p_2)$ ,满足  $p_1 \leq p_2$ ;
- (3)  $Comp$  中的任意子构件都是一致的;
- (4)  $Comp$  中的任意子连接器都是一致的.

类似地,连接器的一致性也如定义 6 所示. 此外,我们可以认为体系结构整体是一个复合构件. 通过使用这些基本操作,我们可以修改整个体系结构的结构和行为,但同时需要保证体系结构的一致性. 因此,我们通过前置条件、结构和行为三个方面对体系结构的演化进行说明. 这里,我们假定所有的构件和连接器都是一致的. 例如

$\text{Add}(P, A)$   
 $S = \text{Add}(P, A, S)$   
 $H = H \mid BE_C$   
 $\text{Remove}(P, A)$   
 $\forall t \in ar(P) \cdot fn(t) \quad (\text{precondition})$   
 $S = \text{Remove}(P, A, S)$   
 $H = S_{BE_P}^1 H$   
 $\text{Replace}(P', P)$   
 $\forall t \in ar(P), \exists t' \in ar(P') \cdot t \leq t' \quad (\text{precondition})$   
 $S = \text{Replace}(P', P, S)$   
 $H = S_{BE_P}^{BE_{P'}} H$   
 $\text{Connect}(p, r)$   
 $t \leq t' \quad (\text{precondition})$   
 $S = \text{Rel}_y(t, t', S)$   
 $H = t/t'.H$   
 $\text{Disconnect}(p, r)$

$$\begin{aligned}
S &= \text{Disconnect}(p, r, S) \\
H &= S'_{f/pr} H \\
\text{Rely}(t, t') \\
p &\cong r \quad (\text{precondition}) \\
S &= \text{Connect}(p, r, S) \\
H &= f/pr.H \\
\text{Disrely}(t, t') \\
S &= \text{Disrely}(t, t', S) \\
H &= t'/t.H
\end{aligned}$$

**定理 1.** 令体系结构 *Arch* 是一致的, *Arch* 在应用体系结构基本操作之后仍然是一致的。

证明. 根据一致性的 4 个条件, 可以直接由定义 4~6 推导出来. 证毕.

## 4.2 完整性

完整性意味着系统的演化不能破坏体系结构规约中的约束条件. 完整性还意味着演化前后系统的状态不会丢失, 否则系统将变得不“安全”, 甚至不能正确运行. 由于演化是由运行系统根据自适应规则决定的, 因此需要对完整性进行验证.

比如对于三层 Client-Server 风格来说, 客户端和服务器的数目是不受限制的, 但是必须有一个 *Core*. 因此, 三层 Client-Server 风格中的这条约束条件可以用 BiLog 表示为  $A = \Diamond \text{Core}$ . 再如为了给用户提供安全机制而增加了 *Authentication* 构件, 但需要保证 *Authentication* 构件嵌入在 *Core* 中. 该约束可以用 BiLog 表示为  $A = \text{Authentication} - \circ \text{Core}$ .

对于完整性的判断, 我们针对体系结构的 Bigraph 表达式进行逐一检验, 具体过程见图 8. 该方法的核心是将前面提到的约束概念映射成基本布尔函数; 由概念模型构成的不变式则转换为判断表达式; 最后, 通过相应的检测函数对体系结构的所有不变式逐一进行验证, 只有合法的修改才会真正被作用于体系结构模型, 破坏了约束的修改将报错并被阻止. 例如, 对于 BiLog 公式  $\Diamond \text{Core}$ , 我们可以对

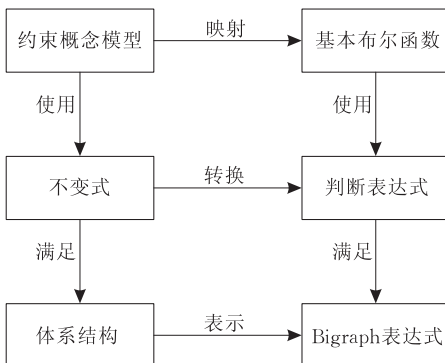


图 8 体系结构和约束的映射图

Bigraph 表达式进行检测, 判断系统在演化后是否存在 *Core* 节点. 对于公式  $\text{Authentication} - \circ \text{Core}$ , 可以通过判断 *Core* 是否为 *Authentication* 的父节点.

## 4.3 遵从风格

在实际中, 我们经常遇到这样的问题: 一个给定的系统是否满足其风格的约束, 如何保证体系结构的演变始终遵循其风格等等.

**定理 2.** 如果初始 Bigraph(体系结构)和反应规则(变化)都满足  $\Sigma$ -BRS 约束条件(风格), 那么由它们生成的 Bigraph(变化后的体系结构)都满足这些约束.

证明. 我们用归纳法加以证明, 假设 Bigraph 集合为  $\{B_0, B_1, B_2, \dots\}$ , 其中  $B_0$  是初始 Bigraph 用于表示初始体系结构实例,  $B_1, B_2, \dots$  是由  $B_0$  和反应规则生成的序列.

令  $B_0$  和任意  $(r, r') \in \mathcal{R}$  都满足约束条件  $\Phi$ , 假设  $B_i$  满足约束. 对任何  $(B_i, B_k)$ , 其中  $i < k$ ,  $B_k$  是由  $B_i$  和  $(r, r')$  生成的 Bigraph.

因此, 存在满足约束条件的上下文  $D$ , 使得  $B_i = D \circ r$  和  $B_k = D \circ r'$ . 由于  $D$  和  $r'$  满足约束,  $B_k$  一定满足约束(类型 Bigraph 的合成仍然满足该类型的约束条件<sup>[6]</sup>), 则可以断定  $\{B_0, B_1, B_2, \dots\}$  中的任何  $B$  都是满足约束的. 证毕.

为了验证该案例, 我们给出该体系结构风格、初始体系结构及反应规则的形式, 具体为

约束条件: (1) 端口 *request* 必须与角色 *caller* 连接, 端口 *reply* 必须与角色 *callee* 连接(我们可以将端口和角色设置为特定的类型, 保证连接方式满足多对一准则); (2) 系统必须包含 *Core* 节点;

初始 Bigraph 仅为一个 *Core* 节点;

反应规则如下:

```

// Add a Client
(inn.Corein.outin.inn / x.x/request.callee / y.y/caller.inn.Clientrequest |
  RPCcaller, callee | Corein.outin.inn)
// Add a Server
(outn.Corein.outout.outn / x.x/reply.callee / y.y/caller.outn.Serverreply |
  RPCcaller, callee | Corein.outout.outn)
// Remove a Client
(x.x/request.callee / y.y/caller.inn.Clientrequest | RPCcaller, callee |
  Corein.outin.inn / inn.Corein.outin.inn)
// Remove a Server
(x.x/reply.callee / y.y/caller.outn.Serverreply | RPCcaller, callee |
  Corein.outout.outn / outn.Corein.outout.outn)

```

根据定理 2 可知, 由初始体系结构和反应规则

所产生的新体系结构是符合三层 Client-Server 风格的. 这样产生的体系结构是满足系统在行为和风格约束的要求. 例如, 对于该风格来说, 客户端和服务器的数目是不受限制的, 但是必须有一个 *Core* 节点. 对于该性质的验证可利用定理 2, 首先初始体系结构 Bigraph 中存在一个 *Core* 节点, 因此  $A$  为真. 第二, 在每个反应规则中, 其反应物和生成物都包括 *Core*. 因此, 无论体系结构如何演化,  $A$  始终为真.

## 5 相关工作

现有的自适应软件体系结构形式化方面的工作可以大致分为两个方面: ADL 和通用形式化方法. 它们在自适应软件体系结构规约、分析和验证方面各有特点, 代表性工作包括:

动态 ACME 可用于描述体系结构中可选或者多重的体系结构元素, 进而达到一定程度的动态特性<sup>[15]</sup>. 同时利用 Armani 规约语言对体系结构的拓扑结构和元素进行了基于不变式的约束. C2 使用专门的体系结构变更语言 AML (Architecture Modification Language)<sup>[16]</sup>. 在 AML 中, 定义了一组在运行时可插入、删除和重新关联体系结构元素的操作, 如 *addcomponent*, *weld* 等. ABC/ADL 强调体系结构模型到程序设计语言的转换<sup>[17]</sup>. ABC/ADL 包含以体系结构元素的对象的增加、删除、替换等操作, 可以为运行实体提供内部状态和外部环境的信息, 比如线程数目、构件实例的个数等. 但这些 ADL 没有严密的理论基础, 不能对体系结构的动态行为进行严格的分析和推演.

Rapide 基于偏序事件集对构件的计算行为和交互行为进行建模<sup>[18]</sup>, 允许在条件语句中通过 *link* 和 *unlink* 操作符重新建立结构关联. 但是 Rapide 不允许单独对连接子进行描述和分析.

Dynamic Wright 使用标签事件技术对 Wright 进行扩展<sup>[19]</sup>, 从而支持对动态体系结构建模和分析. Dynamic Wright 虽然长于进行诸如死锁检测、模型一致性验证等工作, 但对动态的行为变化表达力不足. 同时, 基于 CSP 的特点, 难以进行行为模拟和等价性判定工作.

Darwin 提供延迟实例化和直接动态实例化两种技术<sup>[20]</sup>, 支持动态体系结构建模, 允许事先规划好的运行时构件复制、删除和重新绑定. Darwin 的动态机制并不提供任何  $\pi$  演算语义.

欧盟的 ArchWare 项目提出了一个动态体系结

构描述语言  $\pi$ -ADL<sup>[21]</sup>.  $\pi$ -ADL 是一个基于高阶  $\pi$  演算的形式化语言, 支持动态体系结构建模、体系结构分析和约束检测.  $\pi$ -ADL 具有形式化、可执行、可演化和抽象性等特点. 然而,  $\pi$ -ADL 太过抽象和形式化, 难以理解和使用.

文献[22]通过 CHAM 验证某个体系结构实例是否遵从其风格的约束等. 文献[23]介绍了用图来描述体系结构动态性的方法, 用节点表示构件, 边表示连接子, 并引入协调机制管理体系结构, 动态性则体现在图的重写规则上. 但它们都无法表示上下文信息, 无法处理带参数的变化形式.

我们可以发现这些形式化方法大多注重体系结构的动态性, 而不是自适应性, 因此这些方法很少关注对环境的描述. 其次, 这些形式化方法要么只关注连接, 难以表示结构的嵌套关系以及结构的变化; 要么只关注位置, 难以对行为进行合适的描述. 第三, 这些形式化方法很少关注体系结构在演化中性质的分析和验证. 第四, 大多形式化方法过于抽象, 缺乏图形化支持, 难以理解. 而 Bigraph 融合了  $\pi$  演算和移动 Ambient 演算的优势, 可以有效地解决这些问题.

## 6 总结和进一步工作

在 Internet 环境下, 软件从封闭、静态、可控逐步走向开放、动态、难以预测. 如何为这样的自适应软件提出合适的软件理论已经成为计算机科学与技术面临的挑战性问题. 较为适用的形式化理论基础是一种软件技术达到成熟的标志之一, 而已有的移动和并发理论对自适应软件体系结构的规约、分析和验证都缺乏足够的支持. 虽然软件体系结构技术目前已经进入到黄金发展的时代, 但仍然有诸多问题有待解决, 其中之一是需要有效的机制描述、分析和验证软件体系结构<sup>[24]</sup>. Bigraph 在现有的理论上重点强调计算的位置和连接两方面因素, 具有较为完整并可扩展的理论框架. 现今 Bigraph 理论已经开始走向应用, 关于 Bigraph 理论基础的扩展、移动和并发理论的描述、Bigraph 逻辑、普适计算系统的建模、Bigraph 程序设计语言 BPL 的研究也逐渐开展. 因此, Bigraph 理论可以为自适应软件体系结构的形式化方法提供坚实的基石, 但目前仍有一些问题有待解决:

(1) 上下文信息的指导方法. 现有的系统越来越多地要求持续不断地运行, 而且经常需要对不断



变化的资源和内部状态做出响应,这就需要增加上下文信息用于描述引发这种变化的条件. 因此需要研究如何对上下文信息进行规约,采用什么方法指导这样的规约;

(2) 自适应体系结构描述语言. 现有 ADL 缺乏对环境的描述,其理论基础也不足以对自适应软件的演化性质进行验证. 我们希望利用 Bigraph 理论设计出面向自适应软件的 ADL;

(3) 辅助工具的支持. 与其它的形式化方法一样,体系结构的设计、分析和验证需要大量的辅助工具加以支持,包括设计工具、检验工具和代码生成工具等.

### 参 考 文 献

- [1] Hoare Tony, Milner Robin. Grand challenges for computing research. *The Computer Journal*, 2005, 48(1): 49-52
- [2] Cheng Shang-Wen, Garlan David, Schmerl Bradley, Sousa João Pedro, Spitznagel Bridget, Steenkiste Peter, Hu Ning-Ning. Software architecture-based adaptation for pervasive systems//*Proceedings of the ARCS: Trends in Network and Pervasive Computing*. LNCS 2299. Berlin: Springer-Verlag, 2002: 67-82
- [3] Cheng Shang-Wen, Garlan David, Schmerl Bradley. Architecture-based self-adaptation in the presence of multiple objectives//*Proceedings of the ICSE SEAMS*. Shanghai, China, 2006: 2-8
- [4] Jensen O-H, Milner R. Bigraphs and mobile processes (revised). Computer Laboratory, University of Cambridge, Cambridge: Technical Report UCAM-CL-TR-580, 2003
- [5] Milner Robin. Axioms for bigraphical structure. *Journal of Mathematical Structures in Computer Science*, 2005, 15(6): 1005-1032
- [6] Birkedal L, Debois S, Hildebrandt T T. Sortings for reactive systems. IT University of Copenhagen, Copenhagen, Denmark: Technical Report TR-2006-84, 2006
- [7] Milner Robin. Bigraphs whose names have multiple locality. Computer Laboratory, University of Cambridge, Cambridge: Technical Report: UCAM-CL-TR-603, 2004
- [8] Milner Robin. Pure bigraphs. Computer Laboratory, University of Cambridge, Cambridge: Technical Report UCAM-CL-TR-614, 2005
- [9] Milner Robin. Bigraphs for petri nets//*Lectures on Concurrency and Petri Nets: Advances in Petri Nets*. LNCS 3098. Berlin: Springer-Verlag, 2004: 686-701
- [10] Birkedal L, Debois S, Elsborg E, Hildebrandt T T, Niss H. Bigraphical models of context-aware systems. IT University of Copenhagen, Copenhagen, Denmark: Technical Report: TR-2005-74, 2005
- [11] Conforti G, Macedonio D, Sassone V. BiLog: Spatial logics for bigraphs. University of Sussex, Brighton, UK: Computer Science Report, 2006
- [12] Bundgaard Mikkel N, Birkedal Lars, Debois Søren et al. Bigraphical programming languages for pervasive computing//*Proceedings of the Pervasive 2006 International Workshop on Combining Theory and Systems Building in Pervasive Computing*. 2006: 653-658
- [13] Bradbury J S. Organizing definitions and formalisms of dynamic software architectures. Kingston Queen's University, Kingston, Ontario: Technical Report 2004-477, 2004
- [14] Mao Xin-Jun, Chang Zhi-Ming, Shang Li-Jun, Zhu Hong, Wang Ji. The dynamic castship mechanism for modeling and designing adaptive agents//*Proceedings of the SEKE 2006*. San Francisco, CA, USA, 2006: 639-644
- [15] Wile David S. Using dynamic acme//*Proceedings of a Working Conference on Complex and Dynamic Systems Architecture*. Brisbane, Australia, 2001
- [16] Oreizy P, Medvidovic N, Taylor R N. Architecture-based runtime software evolution//Nuseibeh B ed. *Proceedings of the 2002 International Conference on Software Engineering*. Washington: IEEE Computer Society Press, 1998: 177-186
- [17] Mei H, Chen F, Wang Q X, Feng Y D. ABC/ADL: An ADL supporting component composition//LNCS 2495. Berlin: Springer-Verlag, 2002: 38-47
- [18] Luckham D C, Vera J. An event-based architecture definition language. *IEEE Transactions on Software Engineering*, 1995, 2(9): 717-734
- [19] Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures//*Proceedings of the Fundamental Approaches to Software Engineering*. LNCS 1382. Berlin: Springer-Verlag, 1998: 21-37
- [20] Magee J, Kramer J, Giannakopoulou D. Behaviour analysis of software architectures//*Proceedings of the 1st Working IFIP Conference on Software Architecture*. Boston: Kluwer Academic Publishers, 1999: 35-50
- [21] Oquendo F.  $\pi$ -ADL: An architecture description language based on the higher-order typed  $\pi$ -calculus for specifying dynamic and mobile software architectures. *ACM SIGSOFT Software Engineering Notes*, 2004, 29(3): 1-14
- [22] Wermelinger M. Towards a chemical model for software architecture reconfiguration. *IEEE Proceedings-Software*, 1998, 145(5): 130-136
- [23] Métayer D L. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 1998, 24(7): 521-533
- [24] Shaw M, Clements P. The golden age of software architecture: A comprehensive survey. Institute for Software Research International, Carnegie Mellon University, Pittsburgh, PA, USA: Technical Report CMU-ISRI-06-101, 2006



**CHANG Zhi-Ming**, born in 1979, Ph. D. candidate. His current research interests focus on software architectue.

**MAO Xin-Jun**, born in 1970, Ph. D. , professor, Ph. D. supervisor. His research interests include agent theory and technology, software engineering.

**QI Zhi-Chang**, born in 1942, professor, Ph. D. supervisor. His research interests focus on software engineering.

**Background**

This research is supported by the National Natural Science Foundation of China under grant No. 60773018, the National High Technology Research and Development Program (863 Program) of China under grant No. 2007AA01Z135, the Postgraduate Innovation Fund of National University of Defense Technology under grant No. 070604. These projects aim to build a solid semantic foundation for self-adaptive software, and provide a methodology to validate and check some important properties in the procedure of architectural evolution, such as structural and behavioral consistency and integrity.

With the spread of the Internet and software evolution in complex intensive systems, software architecture often need adapt variable environments and design objectives during runtime. To deal with self-adaptive software architectures, the formal method should be presented to describe software architectures and express their changes so that these changes on the evolutions of software architectures could be reasoned about. However, existing theories of mobile and concurrency calculi focus on dynamic software architecture, and can't

provide powerful support for evolutionary properties of self-adaptive software.

The theory of Bigraphical reactive systems (BRS), due to Milner and co-workers, is based on a graphical model of mobile computation that emphasizes both locality and connectivity. Informally, a BRS consists of Bigraphs and a set of reaction rules, where Bigraphs can be allowed to reconfigure themselves by reaction rules. Therefore, the theory of BRS can provide a sound concept and intuitive, pervasive expression for self-adaptive software architecture.

In this research, the authors used extended BRS as a formal method to describe self-adaptive software architecture. By providing graphic elements and term languages, extended BRS can survey static and dynamic architectures easily. Then they formalized self-adaptive software architecture on structure, behavior and reconfiguration base on extended BRS. By the model based on BRS, they educe a theorem that can ensure architecture consistent, validate the evolving architecture integrity, and check whether the architecture conforms to its style.