软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(6):1210−1223 [doi: 10.3724/SP.J.1001.2011.04017]

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

软件体系结构动态演化的条件超图文法及分析*

徐洪珍[1,2,3+], 曾国荪[1,3], 陈 波[1,3]

[1](同济大学 计算机科学与技术系,上海 201804)

[2](东华理工大学 计算机科学与技术系,江西 抚州 344000)

[3](嵌入式系统与服务计算教育部重点实验室,上海 201804)

Conditional Hypergraph Grammars and Its Analysis of Dynamic Evolution of Software Architectures

XU Hong-Zhen[1,2,3+], ZENG Guo-Sun[1,3], CHEN Bo[1,3]

[1](Department of Computer Science and Technology, Tongji University, Shanghai 201804, China)

[2](Department of Computer Science and Technology, East China Institute of Technology, Fuzhou 344000, China)

[3](Key Laboratory of Embedded System and Service Computing, Ministry of Education, Shanghai 201804, China)

+ Corresponding author: E-mail: hongzhenxu@gmail.com

Xu HZ, Zeng GS, Chen B. Conditional hypergraph grammars and its analysis of dynamic evolution of software architectures. *Journal of Software*, 2011,22(6):1210−1223. http://www.jos.org.cn/1000-9825/4017.htm

Abstract: This paper proposes to represent software architectures with constraint hypergraphs, depict pre-and post-assertions of dynamic evolution of software architectures with left and right application conditions, and model the dynamic evolution process of software architectures with conditional hypergraph grammars. Firstly, how to construct conditional hypergraph grammars and how to apply to dynamic evolution of software architectures through a case study are discussed. Secondly, the consistency condition definition and the corresponding consistency decision method of dynamic evolution of software architectures are given out on this basis. Finally, an experiment is desisgned over analysis for dynamic evolution of software architectures to show the effectiveness of the proposed method.

Key words: software evolution; architecture; constraint hypergraph; conditional hypergraph grammar; consistency

摘 要: 针对目前.软件体系结构动态演化描述方法的不足,提出用约束超图表示软件体系结构,用左右应用条件刻画软件体系结构动态演化的前断言和后断言,用条件超图文法建模软件体系结构动态演化过程.通过案例分析,讨论了如何构建条件超图文法并应用于软件体系结构动态演化.在此基础上,建立软件体系结构动态演化的一致性条件定义,给出动态演化的一致性判定方法.最后,设计实验进行分析,验证了方法的有效性.

关键词: 软件演化;体系结构;约束超图;条件超图文法;一致性

中图法分类号: TP311 **文献标识码**: A

* 基金项目: 国家自然科学基金(90718015); 国家高技术研究发展计划(863)(2009AA012201); 国家重点基础研究发展计划(973)(2007CB316502); 国家教育部博士点基金(20090072110035); 上海市优秀学科带头人计划(10XD1404400); NSFC-微软亚洲研究院联合资助项目(60970155); 高效能服务器和存储技术国家重点实验室开放基金(2009HSSA06)

收稿时间: 2010-07-09; 定稿时间: 2011-03-29

现实世界的软件要么越来越没有价值,要么持续改变以适应环境变化[1].特别是随着普适计算、移动计算、云计算以及网络技术的不断发展,网络的开放性、异构性和动态性使得用户需求、计算环境频繁变化,软件的变化性、复杂性也进一步增强.为了适应这些变化,软件必须能够随时间不断改变.软件的这种不断被改变、调整、加强等以达到所期望状态的过程称为软件演化[2].软件演化可分为静态演化和动态演化[3].支持动态演化的软件能够在运行时改变系统的实现,包括对系统进行功能完善、扩充,改变体系结构等,而不需重启或重编译系统.由于具有持续可用等优点,软件动态演化已逐渐成为软件工程领域研究的热点.

软件的动态可演化性是目前软件工程领域面临的一项挑战.研究人员尝试从软件需求、体系结构、代码复用等方面揭示软件动态演化的规则或规律,并探索动态演化的描述和分析方法.软件体系结构 SA(software architecture)描述了软件系统的组成元素、构件之间的交互、连接及其所遵循的约束或规约等[4].SA 从全局的角度为系统提供结构组成及其相互关系等信息,为人们宏观把握软件的整体结构和动态演化提供了一条有效途径.如何在 SA 层次上刻画和分析演化,已成为研究软件动态演化的关键问题.

SA 动态演化主要涉及运行时软件构件/连接件的增加、删除、替换,以及构件间交互关系、拓扑结构的改变等[5].目前的 SA 动态演化研究工作主要集中在两方面:(1) SA 动态演化描述.研究者尝试通过各种方法制定一些规则或操作,指导和描述 SA 动态演化.例如:Miladi 等人[6]利用 UML(unified modeling language,统一建模语言)profile 建立构件/连接件的创建、删除规则,描述 SA 动态演化;Kacem 等人[7]尝试扩展 UML,通过动态元类建立 SA 的增加、删除、保留规则和操作,描述 SA 动态演化;Oquendo[8],Abi-Antoun[9],梅宏[10]和李长云[11]等人使用不同的 ADL(architecture description language,体系结构描述语言),结合实例系统建立 SA 对象的增加、删除、替换等操作或规则,描述和分析 SA 动态演化;Métayer[12]使用图文法(graph grammar),以节点表示构件,边表示连接件,给出了一个 C/S 系统的增加、删除规则,描述该系统 SA 及其演化;Bruni 等人[13]针对应用实例“Automotive Software System”,采用类型图文法(typed graph grammar)建立相关重配置规则,建模该系统 SA 动态演化;马晓星等人[14]使用类型化图文法刻画 SA 及其风格,给出了基于“Master/Slaver”风格的系统 SA 动态演化的一些规则,并构建了一个动态体系结构支撑环境.(2) 支持 SA 动态演化的软件框架或模型.研究者设计或开发一些以 SA 为核心、支持 SA 动态演化的软件框架或模型.例如:Dowling 等人[15]设计 K-Component 框架元模型,通过提供体系结构元模型和适配契约(adaptation contract)来支持 SA 动态演化;黄罡等人[16]提出了基于体系结构反射的中间件系统,即 PKUAS 系统,支持运行时 SA 演化;马晓星等人[14,17]提出一种支持运行时 SA 对象演化的支撑环境.

虽然研究者在 SA 动态演化方面做了较多的工作,也提出了各种方式的 SA 演化规则或操作,但只有少数学者提及 SA 动态演化的约束或条件.如 Bruni 等人[13]、马晓星等人[14]在他们研究工作的案例分析中提到负应用条件(negative application conditions)[18]在 SA 动态演化中的运用,但没有提供相关的理论依据和深入研究.Tibermacine 等人[19]提出用对象约束语言(OCL)描述全局 SA 约束,并确保这些 SA 约束在软件开发的每个过程中都被遵守,但没有涉及 SA 动态演化的约束.Zhang 等人[20]提出用时态逻辑描述动态自适应软件的语义,用线性时态逻辑 LTL 刻画其中的限制条件,但不是从 SA 的角度来考虑.然而,确定 SA 动态演化的相关约束以及在什么条件下才能进行演化,是保障 SA 动态演化正确的重要手段,对保证 SA 动态演化的正确性具有重要意义.

本文在前期工作的基础上[21],将约束引入到超图,把应用条件扩展到超图文法,提出用约束超图表示 SA,建立 SA 演化规则的应用条件,讨论 SA 动态演化的条件超图文法构建及其应用.在此基础上,给出 SA 动态演化的一致性条件定义及一致性判定方法,并结合实例对 SA 动态演化的一致性进行了讨论.最后,根据所提出的方法,利用 AGG 工具进行了 SA 动态演化实验分析.

1 基本概念

定义 1.1(超图 Hypergraph). 设 $L=(L_V,L_E)$是一对符号集,L 上的超图 H 定义为六元组 $H=(V,E,s,t,vl,\ el)$.其中:V 是节点集合;E 是超边集合,超边是指可以连接任意多个节点的边;$s,t:E\to V^*$分别表示超边到入节点和出节点的映射.这里,*表示每条超边可连接多个入节点和出节点;$vl:V\to L_V,el:E\to L_E$ 分别是节点和超边上的标记函

数,表示节点和超边的属性.

图 1 所示是一个超图举例,其中,$V=\{v_1,v_2,v_3,v_4,v_5,v_6,v_7\}$,$E=\{e_1,e_2,e_3,e_4\}$,$s(e_1)=\{v_1,v_3\}$,$t(e_1)=\{v_2\}$,$s(e_2)=\{v_2\}$,$t(e_2)=\{v_3,v_5\}$,$s(e_3)=\{v_5\}$,$t(e_3)=\{v_3,v_4\}$,$s(e_4)=\{v_4,v_7\}$,$t(e_4)=\{v_5,v_6\}$.节点名和超边名分别作为标记函数,超边上连接节点的部分称为触须(tentacle).

定义 1.2(完全超图态射). 设两超图 $H_1=(V_1,E_1,s_1,t_1,vl_1,el_1)$,$H_2=(V_2,E_2,s_2,t_2,vl_2,el_2)$,$H_1$ 到 H_2 上的一个完全超图态射(total hypergraph morphism)$f:H_1\to H_2$ 是一对映射 $f=(f_V,f_E)$,使得:(1) $f_V:V_1\to V_2$,且 $f_E:E_1\to E_2$;(2) $s_2\circ f_E=f_V^*\circ s_1$;(3) $t_2\circ f_E=f_V^*\circ t_1$;(4) $vl_2\circ f_V=vl_1$; (5) $el_2\circ f_E=el_1$.其中, f_V^* 是 f_V 在多个节点上的扩展,"$\circ$"表示映射的组合.

完全超图态射保持入节点、出节点和标记之间的对应关系.在下文中,除非特别说明,均将完全超图态射简称为超图态射.如果映射 f_V 和 f_E 均为单射(满射),则称 f 为超图单射(超图满射).如果 f 既为超图单射,又为超图满射,则称 f 为同构.此时,称超图 H_1 和 H_2 是同构的,记为 $H_1\cong H_2$.

定义 1.3(部分超图态射). 超图 H_1 到超图 H_2 上的一个部分超图态射(partial hypergraph morphism)$f:H_1\to H_2$ 是指存在 H_1 的一个子图 $H_{1f}\subseteq H_1$ 和一个超图态射 $f':H_{1f}\to H_2$,使得 f' 为完全超图态射.

定义 1.4(超图产生式规则). 一个超图产生式规则 $p=(L,R)$,通常写成 $p:L\to R$,是指超图 L 到超图 R 的一个部分超图态射.其中,L 称为左手边(left-hand side),R 称为右手边(right-hand side).

超图产生式规则可用于描述超图间的变换.给定一个超图 H,一个超图产生式规则 $p:L\to R$,运用 p 对 H 进行变换的实施过程是:(1) 在 H 中查找到其子图 L;(2) 将所有在 L 中但不在 R 中的节点和超边从 H 中删除;(3) 在 H 中增加所有在 R 中但不在 L 中的节点和超边;(4) 保留 L 和 R 中的公共节点和超边,同时用这些公共部分连接已有部分和新添加部分,并保留 H 的其他部分不变,得另一个超图 H',即为变换后的新超图.该变换过程由 4 个(部分)超图态射构成:$p:L\to R$,$m:L\to H$,$p^*:H\to H'$,$m^*:R\to H'$,如图 2 所示.该过程称为 p 和 m 的一个单推出 SPO(single pushout)[22].

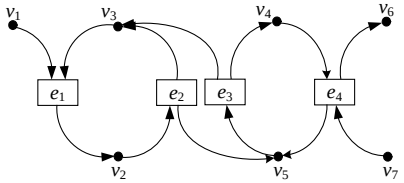

Fig.1 Hypergraph example

图 1 一个超图举例

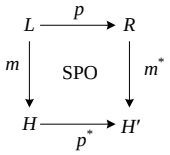

Fig.2 Hypergraph production rule

图 2 超图产生式规则

定义 1.5(超图文法). 一个超图文法是指一个三元组 $G=(\mathcal{H},P,H_0)$,其中,$\mathcal{H}$为有限的超图集,P 为有限的超图产生式规则集,H_0 为初始超图.

本文用超图约束刻画 SA 的结构约束以及 SA 动态演化的应用条件.超图约束表达的是超图中的结构限制,其基本思想是限制在给定的超图上必须存在或禁止存在一定的超图结构.为了描述在 C/S 系统中如果存在客户则必存在服务器之类的复杂约束,本文定义:

定义 1.6(原子超图约束). 一个原子超图约束是指一个超图态射 $c:G\to C$,其中,G,C 均为超图.

定义 1.7(超图约束). 超图约束是按如下方式定义的逻辑公式:(1) 每个原子超图约束均是超图约束;(2) 如果 c 是超图约束,则$\neg c$ 也是超图约束;(3) 如果 c_1 和 c_2 是超图约束,则 $c_1\wedge c_2$ 也是超图约束.其中,$\neg$表示逻辑联结词非,$\wedge$表示逻辑联结词与.

定义 1.8(超图约束满足性). (1) 给定一个原子超图约束 $c:G\to C$,一个超图 H.如果对每个超图单射 $g:G\to H$,存在一个超图单射 $q:C\to H$,使得 $q\circ c=g$,则称 H 满足该超图约束 c,记为 $H\models c$.当 $H\models c$ 时,表示当超图 H 包含超图 G 时,则必包含超图 C,记为$\forall(c:G\to C)$.当 $G=\varnothing$时,表示 H 中必须包含 C,记为$\exists(c:C)$;(2) $H\models\neg c$ 当且仅当 H 不满足 c;(3) $H\models c_1\wedge c_2$ 当且仅当 $H\models c_1$ 且 $H\models c_2$.其中,$\forall$表示逻辑量词任意,$\exists$表示逻辑量词存在.

超图约束$\forall(c:G\to C)$实际上是逻辑公式$\forall G\Rightarrow\exists C$ 的图形表示.图 3 所示的超图约束表明,如果超图 H 满足该

约束,则在 H 中,如果节点 v_1 和 v_2 有超边相连,节点 v_2 和 v_3 也有超边相连,则 v_1 和 v_3 必有超边相连.其对应的逻辑公式为$\forall e_i\wedge\forall e_j\wedge v_1\in s(e_i)\wedge v_2\in t(e_i)\wedge v_2\in s(e_j)\wedge v_3\in t(e_j)\Rightarrow\exists e\wedge v_1\in s(e)\wedge v_3\in t(e)$,其中,为 e_i,e_j,e 为超边,$s(e_i)$为 e_i 的入节点集合,$t(e_i)$为 e_i 的出节点集合,其他记法类似.

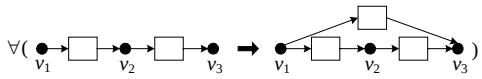

Fig.3 Hypergraph constraint example

图 3 超图约束举例

2 SA 动态演化的条件超图文法

SA 一般采用图形的方式表示,这种方法具有直观、形象等特点.但一般的图在描述构件之间、构件与连接件之间的交互方面存在不足,另外,单纯的图无法描述 SA 的约束,也无法刻画 SA 动态演化的条件.本文提出用带约束的超图(简称约束超图)表示 SA,用基于超图态射的产生式规则表示 SA 演化规则,通过构建条件超图文法建模 SA 动态演化过程.

2.1 SA的约束超图表示

定义 2.1(约束超图 SA). 一个约束超图 SA 是指一个七元组 $SA=(E_{SA},V_{SA},s_{SA},t_{SA},vl_{SA},el_{SA},C_{SA})$,其中:$E_{SA}$ 是超边集合,代表 SA 中的所有构件和连接件;V_{SA} 是节点集合,代表 SA 中所有构件和连接件之间的通信端口,用于表示构件端口和连接件角色之间的绑定,即构件与连接件之间的连接关系;$s_{SA},t_{SA}:E_{SA}\rightarrow V_{SA}^{*}$ 分别是超边的入节点和出节点映射,代表 SA 中构件与连接件之间的交互关系;vl_{SA},el_{SA} 分别是超边和节点上的标记函数,代表 SA 中构件、连接件和通信端口的相关属性;C_{SA} 是超图约束集合,代表 SA 中的所有约束.

本文用方角矩形表示构件,圆角矩形表示连接件,构件/连接件名及其交互关系标记超边,通信端口名标记节点,SA 对应的超图称为 SA 超图.图 4 所示为用约束超图表示的一个 SA 实例,其中包含 3 个构件:$client_1$, $broker_1$ 和 $server_1$、2 个连接件:$client\text{-}connector_1$ 和 $server\text{-}connector_1$、4 个通信端口:$Pc_1,Pcb_1,Pbs_1$ 和 Ps_1,分别对应构件和连接件间的通信端口、1 个约束 c,表示系统中任何构件和连接件只能通过一个通信端口相连,即禁止构件和连接件通过两个或更多的端口相连.CR,CA,SR,SA 分别代表客户请求(client request)、客户响应(client answer)、服务器请求(server request)和服务器响应(server answer)等交互关系.

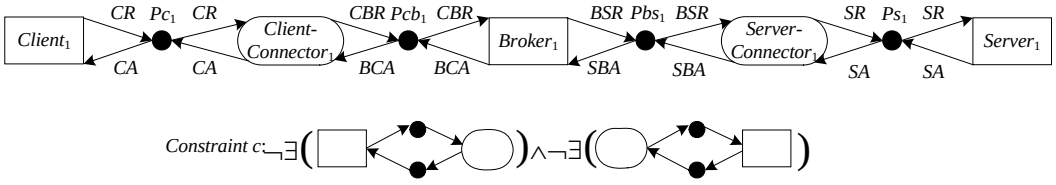

Fig.4 Constraint hypergraph of Client/Server system SA

图 4 Client/Server 系统 SA 的约束超图表示

2.2 SA的行为描述

除了可直观地表示 SA 的结构组成、约束外,超图也可表示 SA 的可观行为.SA 的可观行为是指软件系统中构件与连接件之间外部可见的交互序列,例如构件间通信时发生的交互动作系列.

本文采用类似 CSP(communicating sequential process,通信顺序进程)[23]的记法,用 $p.\bar{a}$ 表示构件/连接件 p 是信息 a 的发送者,$p.a$ 表示构件/连接件 p 是信息 a 的接收者.图 5 所示描述了图 4 表示的软件系统中,客户通过连接件和代理服务器进行通信时的部分可观行为.首先,客户 c_1 通过端口 Pc_1 发出客户请求 CR,连接件 cc_1 接收到该请求,将其封装成新请求 CBR,然后通过端口 Pcb_1 向代理服务器 b_1 转发,b_1 收到该请求 CBR 后,继续向服

务器转发;如果 b_1 收到服务器的响应,则通过端口 Pcb_1 将响应 BCA 返回给 cc_1,最后,cc_1 将相应的响应 CA 返回给 c_1,c_1 通过端口 Pc_1 接收到该响应后,本次通信结束.该部分 SA 可观行为序列可描述为

$$c_1.\overline{CR} \to cc_1.CR \to cc_1.\overline{CBR} \to b_1.CBR \to ... \to b_1.\overline{BCA} \to cc_1.BCA \to cc_1.\overline{CA} \to c_1.CA.$$

显然,相对于一般的图而言,采用约束超图不仅可以直观地刻画 SA 的结构组成、拓扑及约束,并且可以很好地表示构件与构件、构件与连接件之间的交互行为.

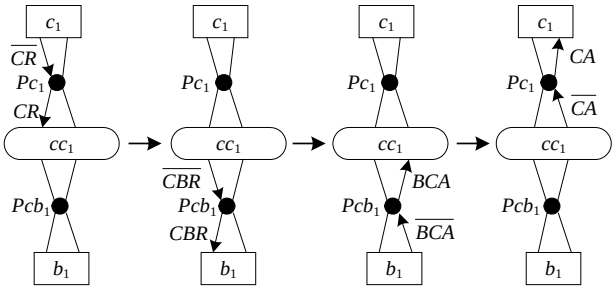

Fig.5 Behavior description of the system SA

图 5 系统 SA 行为描述

2.3 SA动态演化的条件超图文法

定义 2.2(SA 演化规则). 设有一个 SA,它对应的超图为 H,给定一个超图产生式规则 p:$L \to R$,若运用 p 可由 H 变换得到 H',H'为另一个 SA 超图,则称 p 为一个 SA 演化规则.

运用演化规则,开展 SA 动态演化过程可阐述为:假设有一个 SA,它对应的超图为 H,给定一个 SA 演化规则 p:$L \to R$,则运用 p 对 H 进行 SA 动态演化的过程等效于寻找 H 的一个子图 L,用超图 R 对其替换,并保留 H 中其他部分不变,得到另一个超图 H',即系统 SA 由原始的超图 H 演化到新的超图 H'.为了避免出现悬挂的通信端口和触须,本文规定,当删除构件或连接件时,其连接的通信端口以及其他构件或连接件上指向该端口的触须也一并删除.图 6 给出了运用演化规则删除构件 c_2,并用构件 c_4 替换 c_3 的 SA 动态演化的一个例子.

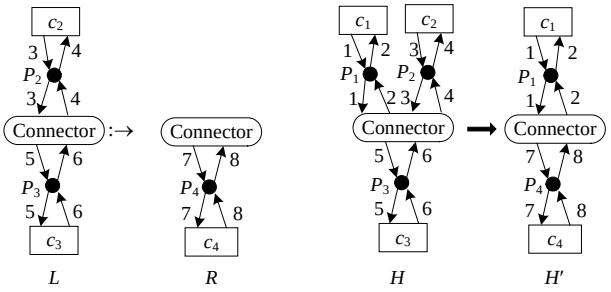

Fig.6 SA dynamic evolution using evolution rule

图 6 运用演化规则进行 SA 动态演化

由 SA 演化规则的定义和实施过程,显然有以下结论:

命题 2.1(SA 演化规则可用性). 设 p:$L \to R$ 是一个 SA 演化规则,H 是一个 SA 超图,若 p 的左手边 L 是 H 的一个子图,则演化规则 p 可运用于 SA 超图 H.

我们称 L 是 H 的一个子图为运用演化规则 p 于 SA 超图 H 的必然条件.除了该必然条件外,在特定的环境下,运用演化规则进行 SA 动态演化还可能存在一些上下文条件.本文主要讨论这类上下文条件,简称为 SA 动态演化的应用条件.为了刻画 SA 动态演化的应用条件,我们将条件引入到 SA 演化规则,用左右应用条件刻画 SA 动态演化的前断言和后断言,用条件超图文法建模 SA 动态演化过程,为此定义:

定义 2.3(SA 演化规则左应用条件). 设 $p:L\to R$ 是一个 SA 演化规则,H 是一个 SA 超图,$c:L\to L'$是一个超图约束,且 L 是 H 的一个子图,即存在一个超图单射 $m:L\to H$.如果存在一个超图单射 $n:L'\to H$,使得 $n\circ c=m$,则称 c 是 p 的一个正左应用条件(positive left application condition).此时也称 m *PL*-满足 c,记为 $m\models_{PL}c$.$m\models_{PL}c$ 表明,只有当 SA 超图 H 上存在超图结构 L'时,才能运用规则 p 进行 SA 动态演化.即正左应用条件刻画的是运用 SA 演化规则前必须具有的条件.

如果不存在一个超图单射 $n:L'\to H$,使得 $n\circ c=m$,则称 c 是 p 的一个负左应用条件(negative left application condition).此时也称 m *NL*-满足 c,记为 $m\models_{NL}c$,即 $m\models_{NL}c\Leftrightarrow m\not\models_{PL}c.m\models_{NL}c$ 表明,如果 SA 超图 H 上存在超图结构 L'时,则禁止运用 p 进行动态演化.即负左应用条件刻画的是运用 SA 演化规则前不能出现的条件.

p 的正左应用条件和负左应用条件通称为 p 的左应用条件.SA 演化规则的左应用条件刻画的是 SA 动态演化的前断言,即 SA 动态演化前必须满足的条件.类似地,可定义:

定义 2.4(SA 演化规则右应用条件). 设 $p:L\to R$ 是一个 SA 演化规则,H'是一个 SA 超图,$c^*:R\to R'$是一个超图约束,且 R 是 H'的一个子图,即存在一个超图单射 $m^*:R\to H'$.如果存在一个超图单射 $n^*:R'\to H'$,使得 $n^*\circ c^*=m^*$,则称 c^*是 p 的一个正右应用条件(positive right application condition).此时也称 m^* *PR*-满足 c^*,记为 $m^*\models_{PR}c^*.m^*\models_{PR}c^*$ 表明,只有当运用规则 p 动态演化后的 SA 超图 H'上存在超图结构 R'时,才能使用 p 进行动态演化.即,正右应用条件刻画的是运用 SA 演化规则后必须具有的条件.

如果不存在一个超图单射 $n^*:R'\to H'$,使得 $n^*\circ c^*=m^*$,则称 c^*是 p 的一个负右应用条件(negative right application condition).此时也称 m^* *NR*-满足 c^*,记为 $m^*\models_{NR}c^*$,即 $m^*\models_{NR}c^*\Leftrightarrow m^*\not\models_{PR}c^*.m^*\models_{NR}c^*$表明,如果运用规则 p 演化后的 SA 超图 H'上存在超图结构 R'时,则禁止使用 p 进行动态演化.即,负右应用条件刻画的是运用 SA 演化规则后不能出现的条件.

p 的正右应用条件和负右应用条件通称为 p 的右应用条件.SA 演化规则的右应用条件刻画的是 SA 动态演化的后断言,即 SA 动态演化后必须满足的条件.

定义 2.5(SA 演化规则应用条件). 设 H,H'是两个 SA 超图,$p:L\to R$ 是一个 SA 演化规则,$\alpha=(\alpha_L,\alpha_R)$称为 p 的一个应用条件,其中,α_L 是 p 的左应用条件,α_R 是 p 的右应用条件.运用规则 p 从 H 到 H'的 SA 动态演化满足应用条件$\alpha=(\alpha_L,\alpha_R)$,是指存在一个如图 2 所示的单推出 SPO,使得 $m\models\alpha_L$ 且 $m^*\models\alpha_R$.当α_L 恒满足时,记$\alpha=\alpha_R$;当α_R 恒满足时,记$\alpha=\alpha_L$.

为了简便和直观,本文采用图形方式表示应用条件.SA 演化规则 p 的左边加上一个带虚线框的超图态射表示 p 的正左应用条件;若在虚线框上有一个“×”,则表示 p 的负左应用条件;类似地也可以表示右应用条件.例如,图 7(a)、图 7(b)分别表示运用演化规则增加构件 c_1 时的两个左应用条件.即增加构件 c_1 时,系统 SA 必须存在和禁止存在另一个构件 c.

相应地,运用带应用条件的演化规则进行 SA 动态演化的过程可描述为:假设有一个 SA,它对应的超图为 H,给定一个 SA 演化规则 $p:L\to R,\alpha=(\alpha_L,\alpha_R)$是 p 的一个应用条件,则运用 p 对 H 进行动态演化的步骤是:(1) 在 H 中查找子图 L,若不存在,则不能运用 p 进行 SA 动态演化,否则进入第 2 步;(2) 检测α是否满足,即检测动态演化前的 SA 超图 H 是否满足左应用条件α_L,且运用 p 动态演化后的 SA 超图 H'是否满足右应用条件α_R;(3) 若满足α,则用 R 对 L 进行替换,并保留 H 中其他部分不变,得到另一个超图 H',即系统 SA 由原始的超图 H 演化到新的超图 H'.若不满足α,则禁止运用 p 进行 SA 动态演化.

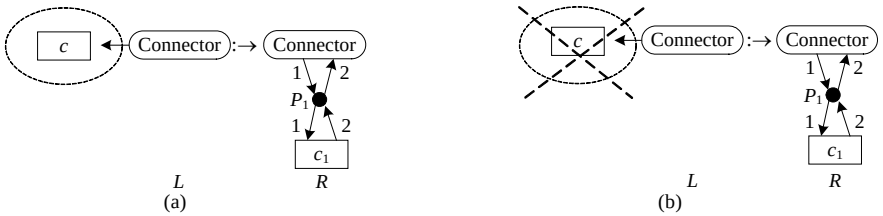

Fig.7 Left application conditions of SA evolution rule

图 7 SA 演化规则的左应用条件

定义 2.6(SA 动态演化条件超图文法). 一个 SA 动态演化条件超图文法,也称为带应用条件的 SA 动态演化超图文法,是指一个四元组 $G=(\mathcal{H},P,H_0,AC)$,其中,$\mathcal{H}$为有限的 SA 超图集,P 为有限的 SA 演化规则集,H_0 为初始 SA 超图,AC 为 P 上的应用条件集合.

本文用 $H \xrightarrow[p]{} H'$ 表示 SA 超图 H 运用演化规则 p 一步演化成 SA 超图 H',用 $H_0 \xrightarrow[p_1p_2\cdots p_n]{} H_n$ 表示 $H_0 \xrightarrow[p_1]{} H_1 \xrightarrow[p_2]{} H_2 \to \ldots \xrightarrow[p_n]{} H_n, H \xrightarrow[*]{} H'$ 表示存在一个 SA 演化规则序列(可能为空)$s \in P^*$,使得 $H \xrightarrow[s]{} H'$.

利用条件超图文法可以建模 SA 动态演化过程,其中,所有动态演化过程中的 SA 超图集合可以表示为该文法生成的一个语言 $L(G)=\{H \mid H_0 \xrightarrow[*]{} H\}$.该方法不仅可以使用基于文法的形式化方法描述 SA 动态演化,而且可以直观、有效地刻画 SA 动态演化及其前、后断言.

3 案例分析

下面使用一个基于 C/S 风格的应用系统作为案例,讨论如何构建条件超图文法并应用于 SA 动态演化.如图 8 所示,该系统包含 3 类构件:客户 *c*、控制服务器 *cs*、服务器 *s*;两类连接件:客户连接件 *cc*、服务器连接件 *sc*.客户通过控制服务器向服务器组发出访问资源的请求,通过一定的机制,服务器组中的某个服务器负责对该请求进行响应,控制服务器则负责将服务器的响应返回给该客户.其中,客户和客户连接件通过通信端口 *Pc* 连接,客户连接件和控制服务器通过通信端口 *Pcs* 连接,控制服务器和服务器连接件通过通信端口 *Pss* 连接,服务器连接件和服务器通过通信端口 *Ps* 连接.*CR*,*CA*,*SR*,*SA*,…分别代表客户请求、客户响应、服务器请求、服务器响应等交互行为.

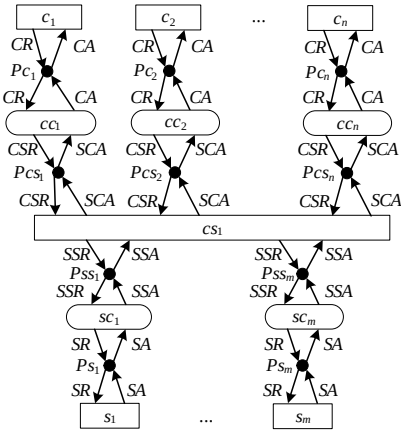

Fig.8 Client/Server system example

图 8 客户/服务器系统例子

为了提高运行效率和减少成本,系统可根据运行情况动态调整在线服务器的数量.同时,系统 SA 动态演化还必须满足一定的约束,本文约定:1) 系统运行时只能包含一个控制服务器;2) 系统运行时最多只能有 *m* 个服务器,不妨设 *m*=2;3) 每个服务器同时最多只能接受 5 个客户的连接请求;4) 每个客户某一时刻只能连接到一个服务器;5) 为了保证通信时不出现环路,每个构件和连接件只能通过一个通信端口相连;6) 系统初始化仅激活控制服务器;7) 系统可在客户连接请求数过多或过轻时,激活或休眠服务器组中服务器;8) 系统可将正在等待响应的客户请求由一个服务器转给另一个服务器.

3.1 案例中的SA演化规则

由于本文的目的不是讨论建立 SA 演化规则,所以这里仅以增加构件和连接件演化为例定义以下 SA 演化规则,其他 SA 演化规则定义可参考前期工作[21]:

(1) 初始化演化规则:假设系统在运行前没有任何实体是激活的,系统初始化后仅激活控制服务器.令系统初始 SA 超图 $H_0=\varnothing$,则初始化演化规则如图 9 所示.这里假设控制服务器暂时没有连接件相连,故没有画出其上的触须;

(2) 类似可以定义增加服务器连接件演化规则、增加服务器演化规则、增加客户连接件演化规则、增加客户演化规则,分别如图 10~图 13 所示.

Fig.9 Initial evolution rule

图 9 初始化演化规则

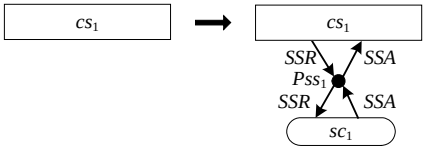

Fig.10 Adding server connector evolution rule

图 10 增加服务器连接件演化规则

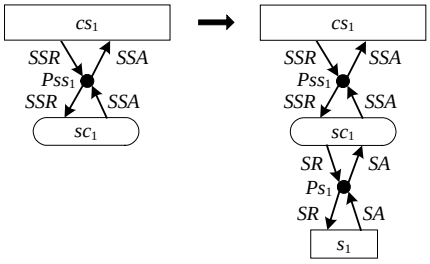

Fig.11 Adding server evolution rule

图 11 增加服务器演化规则

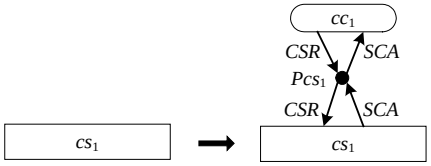

Fig.12 Adding client connector evolution rule

图 12 增加客户连接件演化规则

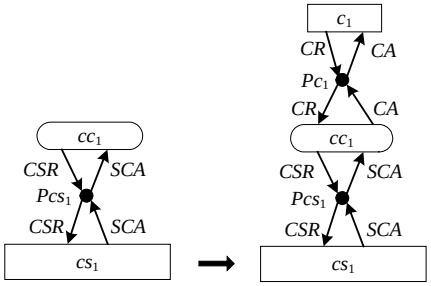

Fig.13 Adding client evolution rule

图 13 增加客户演化规则

3.2 案例中的SA动态演化应用条件

为了刻画 SA 动态演化的上下文条件,本文定义以下应用条件:

(1) 对如图 9 所示的系统 SA 初始化演化规则.由于该规则的左手边为空,故可以任意匹配.为了保证系统运行时只有一个控制服务器存在,本文对该规则增加一个负左应用条件——系统 SA 中不能存在控制服务器.即,当运用初始化演化规则前,首先判断系统中是否存在一个控制服务器.如果不存在,则可运用该规则进行 SA 动态演化;否则,则禁止运用该规则进行 SA 动态演化.该初始化演化规则的应用条件如图 14 所示;

(2) 当运用增加服务器演化规则增加服务器时,为了保证系统运行时最多只有两个服务器存在,此时系统中不能存在两个服务器.该增加服务器演化规则的应用条件如图 15 所示;

(3) 当运用增加客户演化规则增加客户时,系统必须有服务器存在.即系统只能先增加服务器,后增加客户.另外,根据系统约束 3),当增加一个客户后,该客户连接的服务器所连接的客户总数不能超过 5 个.该增加客户演化规则的应用条件如图 16 所示,这里用 *Count*(*c*)表示系统 SA 中单个服务器所连接的

客户个数.

类似地可以定义其他演化规则的应用条件,限于篇幅,这里不再描述.

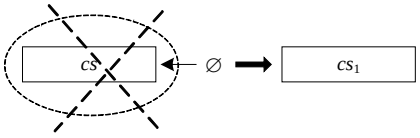

Fig.14 Application condition of initial evolution rule

图 14 初始化演化规则的应用条件

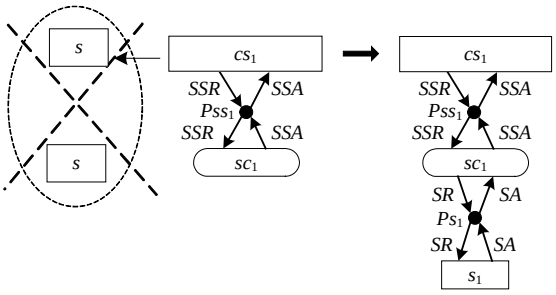

Fig.15 Application condition of adding server evolution rule

图 15 增加服务器演化规则的应用条件

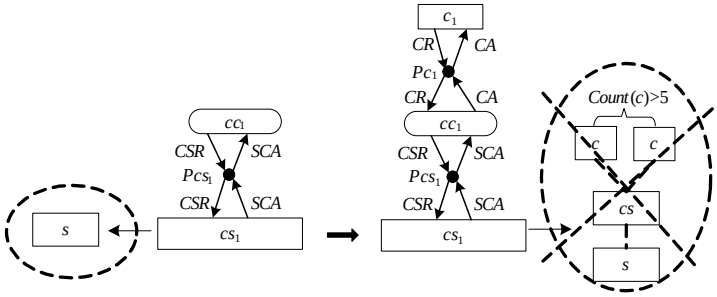

Fig.16 Application condition of adding client evolution rule

图 16 增加客户演化规则的应用条件

3.3 案例中的SA动态演化条件超图文法

根据以上分析,定义该系统 SA 动态演化的一个条件超图文法 $G=\{\{H_0,H_1,\ldots,H_n\},P,H_0,AC\}$.其中,$H_0$ 为系统初始 SA 超图,$H_1,\ldots,H_n$ 为系统动态演化过程中的有限 SA 超图系列,P 为第 3.1 节所定义的演化规则集合,AC 为第 3.2 节所定义的应用条件集合.按照该条件超图文法,系统 SA 则可按照定义的演化规则正确地进行动态演化.限于篇幅,这里不再描述具体演化过程,可参考文献[21].只是在运用相应演化规则进行 SA 动态演化时,必须先进行应用条件判断.如果满足应用条件,则可以使用相应的演化规则进行 SA 动态演化;否则,禁止使用该规则进行 SA 动态演化.

4 SA 动态演化的一致性分析

一致性通常用于描述动态演化过程中所有 SA 都必须拥有的性质,例如存在或唯一存在一定的 SA 组成元素或连接关系等.这些 SA 性质与系统演化过程无关.为保证 SA 动态演化过程的一致性,给出以下定义:

定义 4.1(SA 动态演化一致性条件). 设$\mathcal{H}$为系统动态演化过程中的 SA 超图集,给定超图约束 c,若对任意的 SA 超图 $H\in\mathcal{H}$,有 $H\vDash c$,则称 c 为该 SA 动态演化的一个一致性约束.SA 动态演化中定义的所有一致性约束集合称为该 SA 动态演化的一致性条件.如果系统动态演化过程中的所有 SA 均满足一致性条件,则称该系统 SA 动态演化是一致的.

命题 4.1(SA 动态演化约束可满足性). 给定两个 SA 超图 H 和 H',一个 SA 演化规则 $p:L\to R$,一个超图约束 $c:G\to C$, $H\xrightarrow[p]{}H'$ 表示 H 可运用演化规则 p 一步动态演化到 H'.若 $H\vDash c$ 且 $R\vDash c$,则 $H'\vDash c$.

证明:如图 17 所示,设 $K=H-L$,则 K 是 H 的子超图,显然 $K\vDash c$.由超图约束的定义可知,若 $K\vDash c$,则对每一个超

图单射 g_k:$G\to K$,存在一个超图单射 q_k:$C\to K$,使得 $q_k\circ c=g_k$.又因为 $R\vDash c$,故对每一个超图单射 g_r:$G\to R$,存在一个超图单射 q_r:$C\to R$,使得 $q_r\circ c=g_r$.由 SA 演化规则的应用过程可知,$K\cap R=\varnothing$.对于每一个超图单射 $g'=g_k\cup g_r$:$G\to K\cup R$,构造超图态射 q':$C\to K\cup R$,使得

$$q'=\begin{cases}q_k, \text{ if } q_k\circ c=g'\\ q_r, \text{ if } q_r\circ c=g'\end{cases}.$$

显然,q'为超图单射,且满足 $q'\circ c=g'$.由超图约束的定义,可得 $K\cup R\vDash c$.根据运用演化规则进行 SA 动态演化的过程,则有 $H'=(H-L)\cup R=K\cup R\vDash c$,故命题成立.

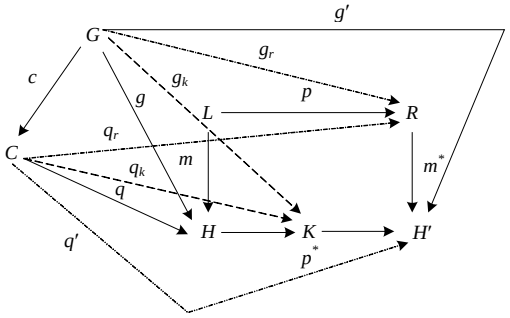

Fig.17 Constraint satisfiability of dynamic evolution of software architectures

图 17 SA 动态演化的约束可满足性

命题 4.2(SA 动态演化一致性判定). 给定系统 SA 动态演化的一个超图约束集合 C,一个条件超图文法 $G=(\mathcal{H},P,H_0,AC)$,如果初始 SA 超图 H_0 满足 C 中的每个约束,且 P 中的每个演化规则的右手边均满足 C 中的每个约束,则按照该条件超图文法进行动态演化的所有系统 SA 均满足 C 中的每个约束,即 C 中的所有超图约束均是一致性约束.若 C 是 SA 动态演化中定义的所有一致性约束集合,即 C 是一致性条件,则该系统 SA 动态演化是一致性的.

证明:系统初始 SA 超图为 H_0,则按照该条件超图文法进行的系统 SA 动态演化过程可描述为

$$H_0\xrightarrow[p_1]{}H_1\xrightarrow[p_2]{}H_2\to\ldots\xrightarrow[p_n]{}H_n,$$

其中:p_i:$L_i\to R_i\in P$;H_i 为系统动态演化过程中的 SA 超图,i=1,2,…,n,n 为自然数.下面对 n 进行归纳证明:

(1) 由命题假设可知,对任意超图约束 $c\in C$,有 $H_0\vDash c$ 且 $R_1\vDash c$,则由命题 4.1 可知 $H_1\vDash c$;

(2) 设 $k<n$ 时命题成立,则有 $H_{n-1}\vDash c$,又由命题假设可知 $R_n\vDash c$,故 $H_n\vDash c$.

由归纳假设可知,按照该条件超图文法进行动态演化的所有系统 SA 均满足 c,即 c 为一致性约束.由 c 的任意性可知,C 中的所有超图约束均是一致性约束,即命题成立.

在第 3 节定义的案例系统中,为了刻画动态演化过程中系统 SA 必须具有的性质,以系统约束 1)、约束 2)、约束 5)为例,定义如图 18 所示的 SA 超图约束,其对应的逻辑公式分别表示为:

(1) $\neg\exists(cs_1\in H\wedge cs_2\in H)$;

(2) $\neg\exists(s_1\in H\wedge s_2\in H\wedge s_3\in H)$;

(3) $\neg\exists(Pc_1(c,cc)\in H\wedge Pc_2(c,cc)\in H)$;

(4) $\neg\exists(Pcs_1(cc,cs)\in H\wedge Pcs_2(cc,cs)\in H)$;

(5) $\neg\exists(Pss_1(cs,sc)\in H\wedge Pss_2(cs,sc)\in H)$;

(6) $\neg\exists(Ps_1(sc,s)\in H\wedge Ps_2(sc,s)\in H)$.

这里,cs_1 表示一个控制服务器,s_1 表示一个服务器,$Pc_1(c,cc)$表示任意客户 c 和客户连接件 cc 间的通信端口,H 表示动态演化过程中的任意 SA 超图,其他记法类似,关于 SA 超图的相关记法见文献[21].其他的 SA 约束可类似定义,限于篇幅,这里不再给出.

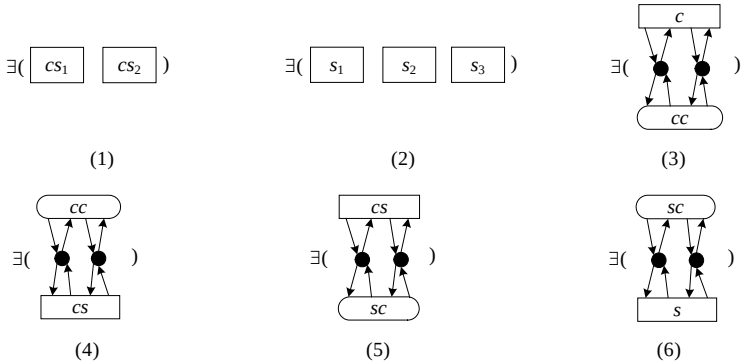

Fig.18 Hypergraph constraints of dynamic evolution of software architectures

图 18 系统 SA 动态演化的超图约束

以上定义的 SA 超图约束均为一致性约束,下面以超图约束(1)为例证明其是一致性约束,其他超图约束是一致性约束的证明完全类似:系统初始 SA 超图 $H_0=\varnothing$,满足任何约束,故满足超图约束(1).显然,第 3.1 节定义的所有演化规则的右手边均满足超图约束(1),则由命题 4.1 可知,超图约束(1)为一致性约束.

定义上述一致性约束为系统 SA 动态演化过程中的一致性条件,则由命题 4.2 可知,按照第 3 节定义的条件超图文法进行的 SA 动态演化满足该一致性条件,即该系统 SA 动态演化是一致的.

5 实验分析

为进一步评估本文所提出的基于条件超图文法 SA 动态演化的正确性和一致性,我们利用图变换(graph transformation)工具 AGG[24]进行实验.AGG 是一个基于代数方法的图变换系统仿真环境,它不仅可以图形方式模拟图变换系统的变换过程,而且还可检查图变换系统的一致性.

按照我们提出的方法,利用 AGG 实现了第 3 节中案例系统的一个 SA 动态演化条件超图文法,并进行实验分析.具体步骤如下:

第 1 步,将我们的超边、超图等记法转换为 AGG 中的记法,在 AGG 中定义相应的节点类型、边类型和一个起始图,该起始图相当于案例系统中的初始 SA 超图;

第 2 步,将案例系统中定义的 SA 演化规则及其应用条件转换为 AGG 中的图变换规则,并定义相应的左应用条件.由于 AGG 不显式支持右应用条件,将演化规则的正右应用条件和负右应用条件分别转换为 AGG 中图变换规则的正左应用条件和负左应用条件,分别对应 AGG 中规则的 NAC 和 PAC.例如第 3.2 节中增加客户演化规则中的负右应用条件(如图 16 所示),即增加一个客户后,该客户连接的服务器所连接的客户数不能超过 5 个,可转换为负左应用条件:增加客户前该服务器所连接的总客户数不能为 5 个.相应地,该带应用条件的演化规则在 AGG 中的构建如图 19 所示,这里用虚线表示客户和服务器通过其他连接件与构件相连.实验中,我们定义了初始化演化规则、增加/删除/替换服务器连接件演化规则等 13 个规则以及它们的应用条件;

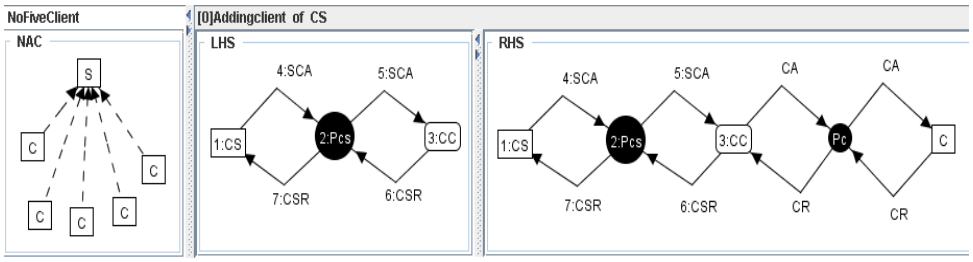

Fig.19 Adding client evolution rule with negative left application condition in AGG

图 19 AGG 中带负左应用条件的增加客户演化规则

第 3 步,在 AGG 中定义案例系统中的原子超图约束和超图约束,对应于第 4 节中的超图约束.本文分别定义了 9 个原子超图约束和超图约束,例如,为了构建图 18 中的超图约束(3),先在 AGG 中定义其原子超图约束,然后再定义其逻辑非“¬”,得到该超图约束.

最后,利用前面定义的起始图和演化规则,在 AGG 仿真本系统的 SA 动态演化过程.进一步地,利用 AGG 的一致性检测功能,可以检测按照我们方法进行的 SA 动态演化的一致性.实验结果如图 20 所示.

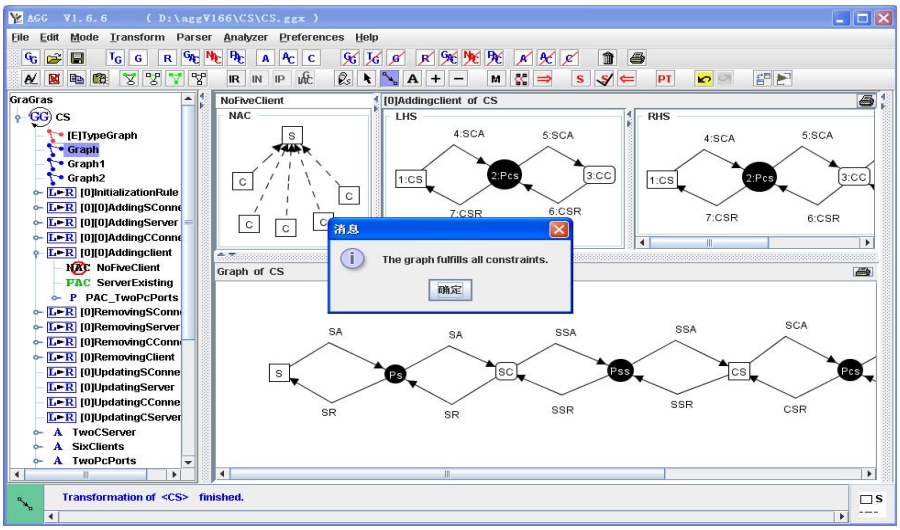

Fig.20 Consistency checking result

图 20 一致性检测结果

实验结果表明,按照我们提出方法定义的条件超图文法和超图约束进行 SA 动态演化,不仅可以保证 SA 动态演化的正确性,还能够保证其一致性.

6 结束语

软件技术发展和环境的变化使得人们在设计软件时日益关注软件动态演化的特性,尤其是 SA 动态演化的特性.其中,刻画 SA 动态演化的相关约束和应用条件是保证 SA 动态演化正确的重要手段.针对目前 SA 动态演化研究缺乏结构约束和演化应用条件分析等问题,本文提出了将约束引入到超图,用约束超图表示 SA,建立 SA 演化规则的左、右应用条件,讨论了如何构建 SA 动态演化的条件超图文法及其在 SA 动态演化中的应用,并在此基础上给出了 SA 动态演化的一致性相关定义和结论,对 SA 动态演化的一致性进行了讨论分析.最后,设计实验分析并验证了所提出方法的可行性和正确性.实验结果表明,我们的方法可以有效地保证 SA 动态演化的正

确性和一致性.另外,我们的方法不仅有直观的图形化描述,而且有基于文法的形式化理论基础.

进一步的工作是结合 UML 和超图文法描述 SA 动态演化的行为语义,并对 SA 动态演化的行为一致性等进行相关分析.

References:

[1] Lehman MM, Ramil JF, Wernick PD, Perry DE, Turski WM. Metrics and laws of software evolution-the nineties views. In: Proc. of the 4th Int'l Software Metrics Symp. Washington: IEEE Press, 1997. 20−32. [doi: 10.1109/METRIC.1997.637156]

[2] Godfrey MW, German DM. The past, present, and future of software evolution. In: Proc. of the 24th IEEE Int'l Conf. on Software Maintenance. Washington: IEEE Press, 2008. 129−138. [doi: 10.1109/FOSM.2008.4659256]

[3] Buckley J, Mens T, Zenger M, Rashid A, Kniesel G. Towards a taxonomy of software change. Journal of Software Maintenance and Evolution: Research and Practice, 2005,17(5):309−332. [doi: 10.1002/smr.319]

[4] Mei H, Shen JR. Progress of research on software architecture. Journal of Software, 2006,17(6):1257−1275 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[5] Halima RB, Jmaiel M, Drira K. Graphical simulation of the dynamic evolution of the software architectures specified in Z. In: Proc. of the 8th Int'l Workshop on Principles of Software Evolution. Washington: IEEE Press, 2005. 45−48. [doi: 10.1109/IWPSE.2005.20]

[6] Miladi MN, Jmaiel M, Kacem MH. A UML profile and a FUJABA plugin for modelling dynamic software architectures. In: Proc. of the Workshop on Model-Driven Software Evolution. Washington: IEEE Press, 2007. 20−26. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.125.5815

[7] Kacem MH, Kacem AH, Jmaiel M, Drira K. Describing dynamic software architectures using an extended UML model. In: Proc. of the Symp. on Applied Computing. New York: ACM Press, 2006. 1245−1249. [doi: 10.1145/1141277.1141569]

[8] Oquendo F. π-ADL: An architecture description language based on the higher-order typed π-calculus for specifying dynamic and mobile software architectures. ACM Sigsoft Software Engineering Notes, 2004,29(4):1−14. [doi: 10.1145/1141277.1141569]

[9] Abi-Antoun M, Aldrich J, Garlan D, Schmerl B, Nahas N, Tseng T. Modeling and implementing software architecture with acme and archJava. In: Proc. of the 27th Int'l Conf. on Software Engineering. New York: ACM Press, 2005. 676−677. [doi: 10.1145/1028664.1028668]

[10] Mei H, Chen F, Wang QX, Feng YD. ABC/ADL: An ADL supporting component composition. LNCS 2495, 2002. 38−47. [doi: 10.1007/3-540-36103-0_6]

[11] Li CY, Li GS, He PJ. A formal dynamic architecture description language. Journal of Software, 2006,17(6):1349−1359 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1349.htm [doi: 10.1360/jos171349]

[12] Métayer DL. Describing software architecture styles using graph grammars. IEEE Trans. on Software Engineering, 1998, 24(7): 521−533. [doi: 10.1109/32.708567]

[13] Bruni R, Bucchiarone A, Gnesi S, Melgratti H. Modelling dynamic software architectures using typed graph grammars. Electronic Notes in Theoretical Computer Science, 2008,213(1):39−53. [doi: 10.1016/j.entcs.2008.04.073]

[14] Ma XX, Cao C, Yu P, Zhou Y. A supporting environment based on graph grammar for dynamic software architectures. Journal of Software, 2008,19(8):1881−1892 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/19/1881.htm [doi: 10.3724/SP.J.1001.2008.01881]

[15] Dowling J, Cahill V. Dynamic software evolution and the *k*-component model. In: Proc. of the OOPSLA 2001 Workshop on Software Evolution. Florida: ACM Press, 2001. http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.25.4179

[16] Huang G, Mei H, Yang FQ. Runtime software architecture based on reflective middleware. Science in China (Series E), 2004,34(2):121−138 (in Chinese with English abstract).

[17] Ma XX, Yu P, Tao XP, Lü J. A service-oriented dynamic coordination architecture and its supporting system. Chinese Journal of Computers, 2005,28(4):467−477 (in Chinese with English abstract).

[18] Habel A, Heckel R, Taentzer G. Graph grammars with negative application conditions. Fundamenta Informaticae, 1996,26(3-4):287−313. [doi: 10.3233/FI-1996-263404]

[19] Tibermacine C, Fleurquin R, Sadou S. Preserving architectural choices throughout the component-based software development process. In: Proc. of the 5th Working IEEE/IFIP Conf. on Software Architecture. Washington: IEEE Press, 2005. 121−130. [doi: 10.1109/WICSA.2005.52]

[20] Zhang J, Cheng BHC. Using temporal logic to specify adaptive program semantics. Journal of Systems and Software, 2006,79(10): 1361−1369. [doi: 10.1016/j.jss.2006.02.062]

[21] Xu HZ, Zeng GS. Specification and verification of dynamic evolution of software architectures. Journal of Systems Architecture, 2010,56(10):523−533. [doi: 10.1016/j.sysarc.2010.08.005]

[22] Ehrig H, Heckel R, Korff M, Löwe M, Ribeiro L, Wagner A, Corradini A. Algebraic Approaches to Graph Transformation, Part II: Single Pushout Approach and Comparison with Double Pushout Approach. Handbook of Graph Grammars and Computing by Graph Transformation: Vol.I. Foundations: World Scientific Publishing Co., 1997.

[23] Hoare CAR. Communicating Sequential Processes. Prentice Hall, 1985.

[24] Taentzer G. AGG: A graph transformation environment for modeling and validation of software. LNCS 3062, 2004. 446−453. [doi: 10.1007/978-3-540-25959-6_35]

附中文参考文献:

[4] 梅宏,申峻嵘.软件体系结构研究进展.软件学报,2006,17(6):1257−1275. http://www.jos.org.cn/1000-9825/17/1257.htm [doi: 10.1360/jos171257]

[11] 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.软件学报,2006,17(6):1349−1359. http://www.jos.org.cn/1000-9825/17/1349.htm [doi: 10.1360/jos171349]

[14] 马晓星,曹春,余萍,周宇.基于图文法的动态软件体系结构支撑环境.软件学报,2008,19(8):1881−1892. http://www.jos.org.cn/1000-9825/19/1881.htm [doi: 10.3724/SP.J.1001.2008.01881]

[16] 黄罡,梅宏,杨芙清.基于反射式软件中间件的运行时软件体系结构.中国科学(E 辑),2004,34(2):121−138.

[17] 马晓星,余萍,陶先平,吕建.一种面向服务的动态协同架构及其支撑平台.计算机学报,2005,28(4):467−477.

徐洪珍(1976－),男,江西临川人,博士生,副教授,CCF 会员,主要研究领域为软件体系结构,软件演化.

陈波(1963－),男,博士生,副教授,主要研究领域为模型检测,可信软件.

曾国荪(1964－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为并行分布处理,可信软件.