第39卷第5期
2011年5月

同济大学学报(自然科学版)
JOURNAL OF TONGJI UNIVERSITY(NATURAL SCIENCE)

Vol.39 No.5
May. 2011

文章编号：0253-374X(2011)05-0745-06

DOI:10.3969/j.issn.0253-374x.2011.05.021

基于超图文法的软件体系结构动态演化

徐洪珍[1,2]，曾国荪[1]

(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)

摘要：提出用带约束的超图表示软件体系结构，给出基于超图态射的软件体系结构动态演化通用产生式规则的形式化语义和操作，定义类型超图作为体系结构风格，运用超图文法和体系结构风格建模软件体系结构动态演化.为了验证软件体系结构动态演化的正确性，采用模型检测技术，设计算法对软件体系结构动态演化性质进行形式化验证，并应用模型检测工具进行实验分析.该方法既提供了图形化的直观表示，又展示了基于文法的形式化理论框架.

关键词：软件演化；体系结构；形式化建模；模型检测

中图分类号：TP 311　　**文献标识码**：A

Dynamic Evolution of Software Architectures Based on Hypergraph Grammars

XU Hongzhen[1,2], *ZENG Guosun*[1]

(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)

Abstract: Hypergraphs with constraints was proposed to represent software architectures. The formal semantics and operations of dynamic evolution production rules of software architectures were presented based on hypergraph morphisms. A software architecture style was defined as a type hypergraph, and the dynamic evolution of software architectures was modeled by applying hypergraph grammars and the architecture style. Model checking technique was used to verify the correctness of dynamic evolution of software architectures, an algorithm was designed to verify the property of dynamic evolution of software architectures. Experimental analysis was made by using a model checker. The approach provides a graphical representation of dynamic evolution of software architectures, and displays a formal theoretical framework based on grammars.

Key words: software evolution; architecture; formal modeling; model checking

随着软件技术的不断发展，软件的变化性和复杂性进一步增强.为了适应用户需求、计算环境等的不断变化，软件系统必须能随时间不断改变.软件这种不断改变的过程就是软件演化.软件演化可分为静态演化和动态演化[1].软件动态演化是指软件在运行期间进行的演化.支持动态演化的软件能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.目前，软件动态演化得到学术界和工业界的极大重视，已成为软件工程领域研究的热点.

软件体系结构SA(software architecture)描述了软件系统的结构组成，构件之间的交互、连接及约束等[2].SA从全局的角度为系统提供结构组成、交互等信息，为人们宏观把握软件演化提供了一条有效途径.如何在SA层次上建模、分析动态演化并保证SA演化正确，已成为研究软件动态演化的关键问题.

当前的SA动态演化研究工作主要有3类：①采用统一建模语言UML及其扩展模型[3-4]建模SA动态演化.尽管UML具有图形化，易理解等特点，但UML主要是一种面向对象建模语言，侧重SA模型描述，缺乏形式化语义，难以精确刻画SA演化的动态特性[5].②使用ADL(architecture description language，体系结构描述语言)建模和分析SA动态

收稿日期：2010-01-25

基金项目：国家"八六三"高技术研究发展计划(2007AA01Z425，2009AA012201)；国家"九七三"重点基础研究发展规划(2007CB316502)；国家自然科学基金(90718015)；NSFC-微软亚洲研究院联合资助(60970155)；教育部高等学校博士学科点专项科研(20090072110035)；上海市优秀学科带头人计划(10XD1404400)；高效能服务器和存储技术国家重点实验室开放基金(2009HSSA06)

第一作者：徐洪珍(1976—)，男，副教授，博士生，主要研究方向为软件演化及模型检测.E-mail:xhz_97@163.com

演化.如采用动态 Wright[6],π-ADL[7]和 D-ADL[8]建模 SA 动态演化,但缺乏图形化工具显示这些模型.③使用形式化方法对 SA 动态演化建模,如采用图的方法[9-10],逻辑的方法[11],代数的方法[12].这些方法大多针对具体系统,给出 SA 动态演化的描述,缺乏通用的规则,且未提供形式化验证.

本文用带约束的超图表示 SA,给出 SA 动态演化的通用产生式规则,定义类型超图作为 SA 风格,用超图文法建模 SA 动态演化,并采用模型检测技术,对 SA 动态演化的相关性质进行形式化验证.

1 软件体系结构与超图文法

定义 1 软件体系结构 SA:是一个 3 元组 $A_s=(C_p,C_n,C)$.其中,C_p 表示软件系统中构件集合;C_n 表示连接件集合;C 表示约束集合.构件指具有一定功能、可明确识别的软件实体.连接件是指用于建立构件间的交互,以及支配这些交互的 SA 构造模块.约束则描述了 SA 配置和拓扑的相关要求.

定义 2 超图:是一个 5 元组 $H=(V,E,s,l_v,l_e)$.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;$s:E\to V^*$ 表示超边到节点的映射,$*$ 表示每条超边可以连接多个节点;l_v,l_e 分别是节点和超边上的标记函数,用于表示节点和超边的相关属性.

采用带约束的超图表示 SA,超边表示构件和连接件,其中方角矩形表示构件,圆角矩形表示连接件,构件或连接件名以及他们之间的交互标记超边.节点表示构件和连接件之间的通信端口,通信端口名标记节点.图 1 所示为用带约束的超图表示的一个 SA 实例,其中包含 3 个构件(客户 c_1,代理 b_1 和服务器 s_1),2 个连接件(客户连接件 c_{nc1} 和服务器连接件 c_{ns1}),4 个通信端口(P_{c1},P_{cb1},P_{bs1} 和 P_{s1}),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.R_{c1},R_{b1},A_{s1},A_{b1} 分别代表客户请求、代理请求、服务器响应和代理响应等交互关系.

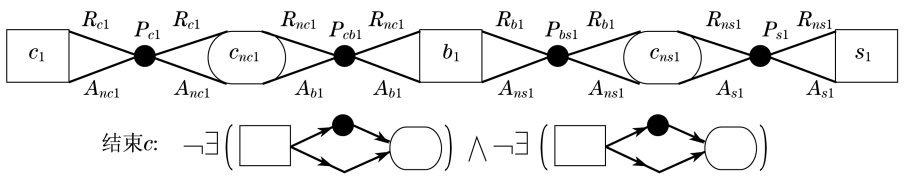

图 1 Client/Server 系统 SA 的超图表示

Fig.1 Hypergraph of a Client/Server system SA

形式上,图是形如 $R(e_1,\cdots,e_n)$的关系元组集合[11];其中,R 是一个 n 元关系,e_i 是实体名.超图作为一类特殊的图,这里只考虑其中的一元和二元关系.用一元关系 $U(e)$表示实体如构件或连接件 e,用二元关系 $L(e_1,e_2)$表示实体 e_1 和 e_2 之间的联系,如构件与连接件之间的通信端口或交互.例如,$C(c_1)$表示构件实体 c_1,$C_{nc}(c_{nc1})$表示连接件实体 c_{nc1},$P_{c1}(c_1,c_{nc1})$表示 c_1 和 c_{nc1} 之间的通信端口,$R_{c1}(c_1,c_{nc1})$表示 c_1 向 c_{nc1} 发出请求等.图 1 所示的 SA 实例可形式化定义为如下集合:$\{\{C(c_1),C_{nc}(c_{nc1}),B(b_1),C_{ns}(c_{ns1}),S(s_1)\},\{P_{c1}(c_1,c_{nc1}),P_{cb1}(c_{nc1},b_1),\cdots\},\{R_{c1}(c_1,c_{nc1}),A_{nc1}(c_{nc1},c_1),\cdots\},\cdots\}$.

定义 3 超图产生式规则:一个超图产生式规则 $p=(L,R)$,通常写成 $p:L\to R$,是指子图 L 变换成子图 R.

用超图产生式规则描述 SA 演化过程.例如,设有一 SA,对应的超图为 H,给定一超图产生式规则 $p:L\to R$,则演化过程等效于寻找 H 的一个子图 L,用子图 R 对其替换,并保留 H 的其他部分不变,得到另一个超图 H',即 SA 由超图 H 演化到新的超图 H'.图 2 给出了运用产生式规则进行 SA 演化的一个例子.

定义 4 超图文法:是指一个 4 元组 $G=(N_H,T_H,P,H_0)$;其中,N_H 代表有限的非终结超图集,T_H 代表有限的终结超图集,P 是有限的超图产生式规则集,H_0 是初始超图.

用 $H\xrightarrow[p]{}H'$表示超图 H 应用一次产生式规则 p 演化成 H',用 $H\xrightarrow[p_1,p_2,\cdots,p_n]{}H'$表示 $H_0\xrightarrow[p_1]{}H_1\xrightarrow[p_2]{}H_2\longrightarrow\cdots\xrightarrow[p_n]{}H_n$.运用超图文法,可使用基于文法的形式化方法建模 SA 动态演化,且演化过程可以转换为模型检测中的状态迁移系统,进行相关性质验证.

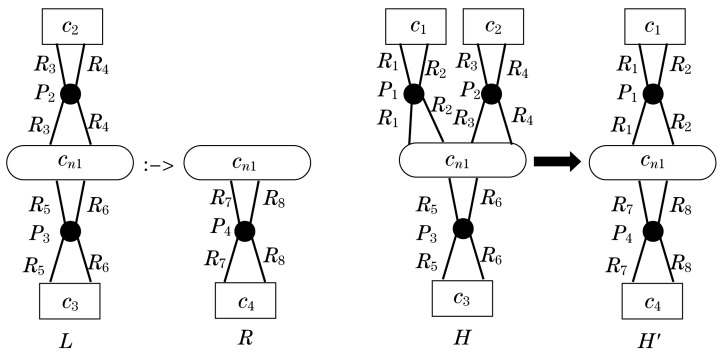

图 2　运用产生式规则进行 SA 演化

Fig. 2　SA evolution using a production rule

2　SA 动态演化

2.1　SA 动态演化产生式规则

SA 动态演化通常包括 3 类操作：增加构件或连接件，删除构件或连接件，替换构件或连接件. 为了建模 SA 动态演化，预先定义以下通用的产生式规则，且这些产生式规则必须满足一定的体系结构风格.

2.1.1　增加构件和连接件产生式规则

设系统当前 SA 超图为 H，增加一个构件和连接件的演化产生式规则形式化描述为

$$H \to H \cup \{C_p(c), R_1(c, c_n), \cdots, P_1(c, c_n), \cdots\} \tag{1}$$

$$H \to H \cup \{C_n(c_n), R_1(c, c_n), \cdots, P_1(c, c_n), \cdots\} \tag{2}$$

式中：$C_p(c)$ 为构件实体 c；$C_n(c_n)$ 为连接件实体 c_n；R_i, P_i 为交互关系和通信端口. 当系统需要增加功能时，可以使用规则(1)，(2)动态增加相应构件或连接件到 SA 中. 图 3 表示增加一个构件 c_1 时的动态演化操作的图示化过程.

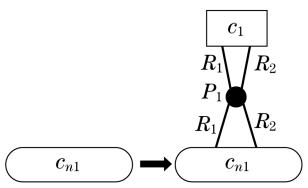

图 3　增加构件操作

Fig. 3　Operation for adding a component

2.1.2　删除构件和连接件产生式规则

删除一个构件和连接件的演化产生式规则形式化描述为

$$H \to H - \{C_p(c), R_1(c, c_n), \cdots, P_1(c, c_n), \cdots\} \tag{3}$$

$$H \to H - \{C_n(c_n), R_1(c, c_n), \cdots, P_1(c, c_n), \cdots\} \tag{4}$$

当一个构件或连接件不再被使用时，可以用这两个规则动态删除它们. 图 4 表示删除一个构件时的动态演化操作的图示化过程.

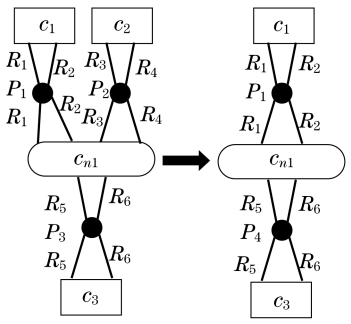

图 4　删除构件操作

Fig. 4　Operation for removing a component

2.1.3　替换构件和连接件产生式规则

替换一个构件和连接件的演化产生式规则形式化描述为

$$H \to H \cup \{C_p(c_1), R'_1(c_1, c_n), \cdots, P'_1(c_1, c_n), \cdots\} - \{C_p(c), R_1(c, c_n), \cdots, P_1(c, c_n), \cdots\} \tag{5}$$

$$H \to H \cup \{C_n(c_{n1}), R'_1(c, c_{n1}), \cdots, P'_1(c, c_{n1}), \cdots)\} - \{C_n(c_n), R_1(c, c_n), \cdots, P_1(c, c_n) \cdots\} \tag{6}$$

当系统需要对某个构件或连接件进行功能扩充时，可以用这两个规则进行相应的替换. 图 5 表示替换一个连接件的动态演化操作的图示化过程.

2.2　实例分析

本文使用一个基于 Web 的电子商务系统作为应用场景. 设系统初始 SA 如图 6 所示，其超图为 H_0. 系统包含 3 个构件(Web 客户 c_{w1}、Web 服务器 s_{w1}、SQL 数据库 d_1)，2 个连接件(Web 连接件 c_{nw1} 和数据库连接件 c_{nd1}). c_{w1} 通过 c_{nw1} 向 s_{w1} 发出访问资源的请求，d_1 响应通过 c_{nd1} 发回给 s_{w1}；c_{w1} 和 c_{nw1}

通过端口 P_{w1} 连接；R_{w1}，A_{s1} 分别代表 Web 客户请求、Web 服务器响应；其他标识含义类似.

2.2.1 SA 风格

当系统动态演化时，SA 必须遵循预先定义的风格，例如保持相应的构件类型、连接件类型，以及构件间交互类型及约束等. SA 风格是指描述 SA 中构件、连接件的组成，构件与连接件之间交互和约束的典型模式. 本文采用类型超图定义了上述应用场景的 SA 风格，该 SA 风格规定了构件和连接件类型、通信端口类型、SA 约束等. 例如定义 Web 客户的类型为 C_w，Web 服务器的类型为 S_w，Web 连接件的类型为 C_{nw}，Web 客户和 Web 连接件的通信端口类型为 P_w，定义 SA 约束：$\forall C(c), C_{nw}(c_{nw}) \Rightarrow \exists\,|\,P_w(c, c_{nw})$，其中 $\exists\,|$ 表示惟一存在，等等. 该 SA 风格如图 7 所示.

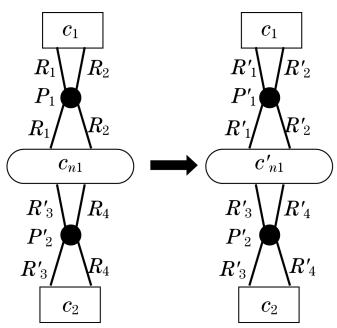

图 5 替换连接件操作

Fig. 5 Operation for replacing a connector

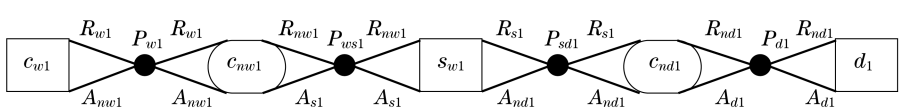

图 6 客户服务器系统例子

Fig. 6 Example of a Client/Server system

图 7 SA 风格

Fig. 7 SA style

2.2.2 SA 动态演化过程

随着无线通信和移动技术的发展，系统需要增加移动电子商务功能，面向移动用户. 此时需扩展 Web 服务器的功能. 为此设计新的构件 s_{mw1} 替换原有的服务器，同时增加移动连接件支持移动用户. 系统应用产生式规则(5)，用 s_{mw1} 替换 s_{w1} 以及相应的联系，并分别应用产生式规则(2)和(1)增加移动连接件、移动用户及相应的交互和通信端口. 其演化过程形式化描述如下：

$$H_0 \xrightarrow[5]{} H_0 \cup S_{mw}(s_{mw1}), R'_{nw1}(c_{nw1}, s_{mw1}), A_{m1}(s_{mw1}, c_{nw1}), P_{wm1}(c_{nw1}, s_{mw1}), R_{m1}(s_{mw1}, c_{nd1}), A'_{nd1}(c_{nd1}, s_{mw1}), P_{md1}(c_{nd1}, s_{mw1})\} - \{S_w(s_{w1}), R_{nw1}(c_{nw1}, s_{w1}), A_{s1}(s_{w1}, c_{nw1}), P_{ws1}(c_{nw1}, s_{w1}), R_{s1}(s_{w1}, c_{nd1}), A_{nd1}(c_{nd1}, s_{w1}), P_{sd1}(s_{w1}, c_{nd1})\} =$$

$$H_1 \xrightarrow[2]{} H_1 \cup \{C_{nm}(c_{nm1}), R_{nm1}(c_{nm1}, s_{mw1}), A_{mm1}(s_{mw1}, c_{nm1}), P_{mm1}(s_{mw1}, c_{nm1})\} =$$

$$H_2 \xrightarrow[1]{} H_2 \cup \{C_m(c_{m1}), R_{m1}(c_{m1}, c_{nm1}), A_{nm1}(c_{nm1}, c_{m1}), P_{m1}(c_{m1}, c_{nm1})\} = H_3$$

该演化过程如图 8 所示.

类似地，可用 SA 演化产生式规则描述其他 SA 动态演化，由于篇幅有限，这里不再描述.

2.2.3 SA 动态演化超图文法

根据以上分析，定义该系统 SA 动态演化的一个超图文法 $G = \{\{H_0\}, \{H_0, H_1, \cdots, H_n\}, P, H_0\}$. 其中，$H_0, H_1, \cdots, H_n$ 为如上所述的有限系统 SA 超图系列，P 表示演化产生式规则集合，H_0 为系统初始 SA 超图.

3 SA 动态演化验证

3.1 SA 动态演化验证

在 SA 动态演化中，为了保证演化正确性，SA 必须满足一定的性质，且必须保证这些性质得到满足. 为了验证满足 SA 性质，本文采用模型检测技术，将 SA 超图映射为状态，每次 SA 演化规则应用映射为转移关系，由此定义一个状态迁移系统.

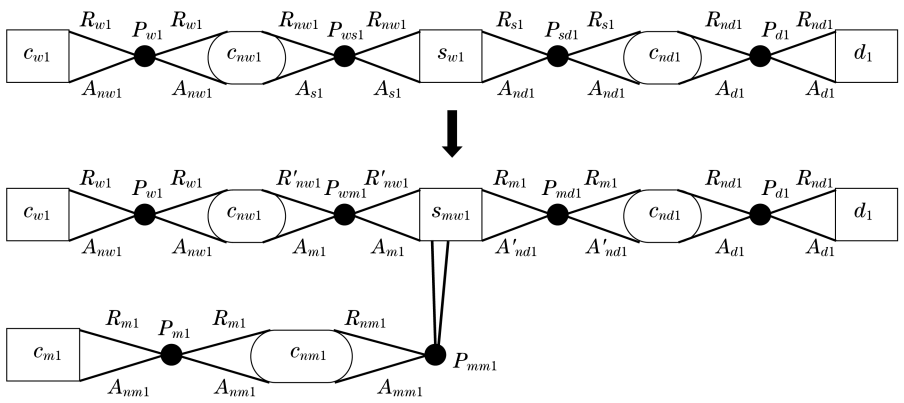

图 8　SA 动态演化过程

Fig. 8　Dynamic evolution process of SA

定义 5　SA 状态迁移系统：是一个元组 $M=(S,s_0,R,A,L)$. 其中：$S=\{H_0,H_1,\cdots,H_n\}$ 为状态集，对应超图文法中的 SA 超图集；$s_0=H_0$ 为初始 SA 超图；$R\subseteq S\times S$ 为转移关系，对应超图文法中的每次 SA 演化产生式规则应用；A 为原子命题集；L 为标记函数，用于给每个状态标记满足的原子命题集.

给定 SA 动态演化超图文法 $G=(N_H,T_H,P,H_0)$，则令 $S=N_H\cup T_H$，$R=\{s_i\rightarrow s_j\mid \exists H_i=s_i\wedge \exists H_j=s_j\wedge \exists p\in P\wedge H_i \xrightarrow[p]{} H_j\}$，$s_0=H_0$，再给定原子命题集 A 和标记函数 L，则得对应的状态迁移系统 $M=(S,s_0,R,A,L)$.

本文仅讨论系统 SA 动态演化过程具有有限个 SA 超图，即 S 为有限状态集的情形，且讨论验证 SA 动态演化的不变性(invariant)，其他性质验证将在后续工作中进行.

定义 6　SA 不变性：SA 性质 p 为不变性，是指存在一个 p 对应的逻辑公式 ϕ，对任意的 SA 演化系列 $H_0,H_1,\cdots,H_n$，其中 H_0 为初始 SA 超图，$H_i \xrightarrow[p_i]{} H_{i+1}$，$0\leqslant i<n$，$p_i$ 为演化产生式规则，n 为自然数，使得 H_i 满足公式 ϕ，即 $\forall i, H_i\mid=\phi$，$0\leqslant i<n$.

性质 1　在前文实列分析定义的场景中，每个 SA 中的每个客户必须且只能通过 1 个通信端口和连接件相连. 为了保证系统通信不出现环路，必须保证性质 1 为真，将其写成逻辑公式为

$$\Phi_1=(\forall c,c\in H_i\rightarrow \exists P_w,P_w(c,c_n)\in H_i\wedge(\forall P_{w1},P_{w2},P_{w1}(c,c_n)\in H_i\wedge P_{w2}(c,c_n)\in H_i\rightarrow P_{w1}=P_{w2})),\ 1\leqslant i\leqslant n$$

式中：c 表示客户；c_n 表示对应的连接件；P_w，P_{w1}，P_{w2} 均为通信端口；H_i 为 SA 超图.

为了验证性质 1 是 SA 不变性，必须验证在转移系统 M 中，每个由初始状态出发可达的状态上 Φ_1 为真. 为此，在 M 上设计一种深度优先遍历算法，在每步到达的状态上验证性质 1 的正确性，算法结束即可验证性质 1 的不变性. 算法如下图所示，其中 V 存放所有访问过的 SA 超图，U 为一个栈，存放所有即将访问的 SA 超图，ε 为空栈.

```
Procedure CheckingInvariant ( transition system M, state s0, property Φ1)
V := Ø;
U := ε;
bool b := true;
push (s0, U);
V := V∪{ s0};
do
    s1 := top(U);
    if post(s1)⊆V then
        pop(U);
        b := b∧(s1 | = Φ1);
    else
        let s2∈post(s1)/V
        push(s2, U);
        V := V∪{s2};
    End if
while ((U ! =ε)∧b)
if b then
    return("yes")
else
    U′ :=ε;
    while(U ! =ε)
        pop(U,e)
        push(e,U′)
```

```
        end while
        while(U′ ! =ε)
            pop(U′,e)
            print(e)
        end while
        return("no")
    end if
    end Procedure
```

该验证算法的时间复杂性主要在于深度优先遍历.设在单个状态上检测公式 Φ_1 的复杂度为 $|\Phi_1|$,用邻接表表示状态迁移系统中的后继关系,即 $p(s)$ 用 s 的邻接表表示,则该验证算法的时间复杂性即为深度优先遍历的时间复杂性 $O(n^*|\Phi_1|+e)$,其中 n 为 M 中的可达状态数,即系统演化可达的 SA 超图数,e 为 M 中的转移数.

3.2 实验分析

本文使用模型检测工具 SPIN 实现了上述不变性的验证,用 PROMELA 定义相应的 SA 动态演化模型,其中 SA 超图定义为 PROMELA 中的进程,演化产生式规则定义为 PROMELA 中的事件.定义相应的超图类型,初始 SA 超图和演化产生式规则.使用该初始 SA 超图和这些演化产生式规则可以生成 SA 动态演化的状态迁移系统.上述性质 1 表示为时序逻辑公式.使用本文的算法检验每步产生的状态上该不变性是否成立.实验结果表明,使用本文的方法进行动态演化,SA 超图均满足性质 1,即性质 1 为不变性.

4 结语

软件技术的发展使得人们在设计软件时日益关注软件演化的动态特性,尤其是 SA 演化的动态特性.本文讨论了一种形式化的 SA 动态演化建模和验证方法——应用超图文法对 SA 动态演化进行建模与验证.首先建立基于超图态射的通用 SA 演化产生式规则的形式化语义和操作;然后通过 SA 的初始模型,运用这些 SA 演化产生式规则和 SA 风格,建模 SA 动态演化过程;最后应用模型检测对 SA 动态演化的不变性进行了验证,给出了相应的验证算法和实验分析.该方法既有直观的图形化描述,又有形式化的理论基础.

进一步的工作是设计组合的产生式规则描述 SA 动态演化,并对 SA 动态演化的其他属性如活性、一致性等进行相应的验证.

参考文献:

[1] Buckley J, Mens T, Zenger M, et al. Towards a taxonomy of software change [J]. Journal of Software Maintenance and Evolution: Research and Practice, 2005, 17(5): 309.

[2] 梅宏,申峻嵘.软件体系结构研究进展[J].软件学报,2006,17(6):1257.
MEI Hong, SHEN Junrong. Progress of research on software architecture[J]. Journal of Software, 2006, 17(6): 1257.

[3] Kacem M H, Kacem A H, Jmaiel M, et al. Describing dynamic software architectures using an extended UML model[C]//The 21st Annual ACM Symposium on Applied Computing. New York: ACM Press, 2006: 1245 - 1249.

[4] Miladi M. N, Jmaiel M, Kacem M H. A UML profile and a Fujaba plugin for modelling dynamic software architectures [C]// Proceedings of the Workshop on Model - Driven Software Evolution. Amsterdam: IEEE Press, 2007: 20 - 26.

[5] Pelliccione P, Inverardi P, Muccini H. CHARMY: a framework for designing and verifying architectural specifications [J]. IEEE Transactions on Software Engineering, 2009, 35(3): 325.

[6] Allen R, Douence R, Garlan D. Specifying and analyzing dynamic software architectures [C]//Lecture Notes in Computer Science. Lisbon: Springer Press, 1998. 1382: 21 - 37.

[7] Oquendo F. π - ADL: an architecture description language based on the higher - order typed π - calculus for specifying dynamic and mobile software architectures [J]. ACM Sigsoft Software Engineering Notes, 2004, 29 (4): 1.

[8] 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,2006,17(6):1349.
LI Changyun, LI Gansheng, HE Pinjie. A formal dynamic architecture description language [J]. Journal of Software, 2006, 17(6): 1349.

[9] Métayer D. L. Describing software architecture styles using graph grammars [J]. IEEE Transactions on Software Engineering, 1998, 24(7): 521 - 533.

[10] Bruni R, Bucchiarone A, Gnesi S, et al. Modelling dynamic software architectures using typed graph grammars [J]. Electronic Notes in Theoretical Computer Science, 2008, 213 (1): 39.

[11] Aguirre, N, Maibaum T. A temporal logic approach to the specification of reconfigurable component - based systems [C]//Proceedings of the 17th IEEE international conference on automated software engineering. Washington D C: IEEE Press, 2002: 271 - 278.

[12] 王映辉,王立福.软件体系结构演化模型[J].电子学报,2005,33(8):1381.
WANG Yinghui, WANG Lifu. Research about model and ripple effect analysis of software architecture evolution [J]. Acta Electronica Sinica, 2005, 33(8): 1381.