g_{lhs} 中的变量 $x_0,x_1,\ldots,x_n$,它们对应的谓词命题为真,且满足($p_{pro}\wedge\neg p_{neg}$).又因为 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$,且 ASM_{SA} 中的状态与 ASM_{STS} 中的状态具有一一对应关系,所以对 ASM_{SA} 中的相应变量值 $x_0,x_1,\ldots,x_n$,它们对应的谓词命题也为真.

因为 t 是由算法 2 生成的条件状态转移关系,故存在 ASM_{SA} 中的一个 SA 演化规则 p,由 p 可生成 t,即,$t=F(p)$.又因为 $\mathcal{A}_{STS}=F(\mathcal{A}_{SA})$,所以对 ASM_{SA} 中的相应变量 $x_1,x_2,\ldots,x_n$,其 g_{lhs} 也为真,且满足($p_{pro}\wedge\neg p_{neg}$).因此,SA 演化规则 p 可运用于 ASM_{SA} 中的状态 $\mathcal{A}_{SA}$,即:存在 $\mathcal{B}_{SA}$,使得 $\mathcal{B}_{SA}=next_p(\mathcal{A}_{SA})$.

又由于算法 2 中的赋值语句(第 6 行~第 8 行)完全对应于算法 1 中相应的赋值语句(第 3 行~第 57 行),故运用演化规则 p 后,状态 $\mathcal{B}_{SA}$ 的相应赋值完全对应于运用条件状态转移关系 t 后状态 $\mathcal{B}_{STS}$ 的相应赋值,故有:

$$\mathcal{B}_{STS}=F(\mathcal{B}_{SA}).$$

故命题成立. □

5 案例分析

为了便于描述,下面使用一个基于 C/S 风格的应用系统[9]作为案例,讨论如何将 SA 动态演化的条件超图文法映射为条件状态转移系统,并验证 SA 动态演化的相关性质.如图 4 所示,该系统包含 3 类构件:客户 c_i、控制服务器 cs_1、服务器 s_i;两类连接件:客户连接件 cc_i、服务器连接件 sc_i;客户 c_i 和客户连接件 cc_i 之间的通信端口为 Pc_i,客户连接件 cc_i 和控制服务器 cs_1 之间的通信端口为 Pcs_i;Rc_i 表示客户 c_i 向客户连接件 cc_i 发出的客户请求,Ac_i 表示客户连接件 cc_i 向客户 c_i 发回的客户应答,等等.另外,本文规定系统 SA 动态演化还必须满足如下条件:(1) 系统演化过程中只能包含一个控制服务器;(2) 系统演化过程中最多只能有 m 个服务器,不妨设 m=5;(3) 系统最多只能接受 n 个客户的连接请求,不妨设 n=100;(4) 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.

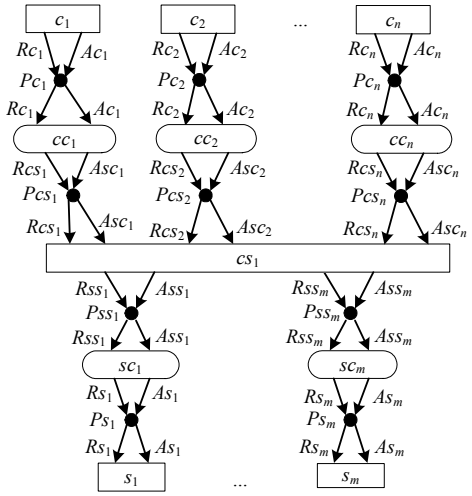


Fig.4 Client/Server system example

图 4 客户/服务器系统例子

5.1 案例系统SA动态演化的条件超图文法

由于本文的目的不是讨论如何建立 SA 演化规则及其应用条件,所以这里仅以增加服务器和增加客户为例定义以下带应用条件的演化规则:带应用条件的增加服务器演化规则(如图 3 所示)和增加客户演化规则(如图 5 所示),其他的 SA 演化规则及其应用条件定义可参考我们的前期工作[9].

假设系统初始 SA 超图为 H_0(空集),则可以定义该系统 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,$H_1,\ldots,H_n$ 为系统动态演化过程中的有限 SA 超图系列,P 为演化规则集,AC 为演化规则的应用条件集.

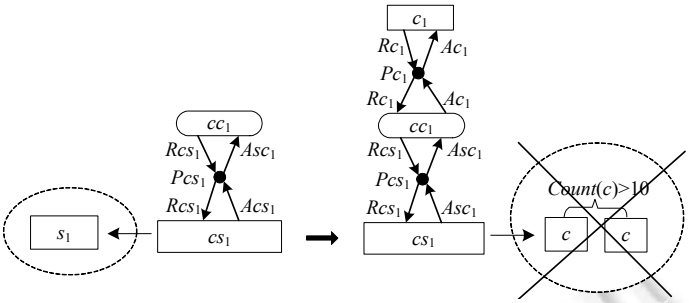


Fig.5 Adding client evolution rule with application conditions

图 5 带应用条件的增加客户演化规则

5.2 案例系统SA动态演化条件超图文法到条件状态转移系统的映射

5.2.1 案例系统 SA 超图到状态的映射

根据第 3 节的方法,将上述 SA 动态演化条件超图文法 G 中的 SA 超图映射为状态.为表示这些状态,在 PROMELA 中定义以下布尔变量或布尔变量数组:

```
bool cs1;          //标识每个状态中的控制服务器 cs1
bool C[100];       //标识每个状态中的各个客户构件
bool Cc[100];
bool S[5];
bool Sc[5];
bool Pc[100];
bool Pcs[100];
bool Pss[5];
…
```

因为在案例系统中,客户构件 c_i 只可能和客户连接件 cc_i 之间存在通信端口,与其他连接件和构件之间不可能存在通信端口,所以这里只用一维数组 Pc[100]表示客户构件和客户连接件之间的通信端口,其他变量数组的定义类似.那么,在初始状态 s_0 中,所有变量或变量数组元素的值均为 false,在其他状态上,所有变量或变量数组元素赋予相应的值.

5.2.2 案例系统 SA 演化规则运用到条件状态转移关系的映射

根据第 3 节所提方法,可将每次 SA 演化规则运用映射为一个条件状态转移关系.例如,设系统当前 SA 超图 H_1 如图 6 所示,如果将图 3 所示的增加服务器演化规则 p 运用于 H_1 进行动态演化,则存在两种可能的匹配,即:既可和连接件 sc_1 匹配,也可和连接件 sc_2 匹配,则运用演化规则 p 后,H_1 可能演化成如图 7 所示的 SA 超图 H_2,也可能演化成如图 8 所示的 SA 超图 H_3.于是,p 的两次匹配运用可映射为两个条件状态转移关系,如图 9 所示,其中,状态 s_1 对应于 SA 超图 H_1,s_2 对应于 H_2,s_3 对应于 H_3.

根据算法 2,演化规则 p 可能的运用映射成条件状态转移关系对应的 PROMELA 伪代码如下:

```
if
::(cs1==true&&Sc[1]==true&&Pss[1]==true&&Rss[1]==true&&Ass[1]==true&&S[1]==false)→S[1]=true;
Ps[1]=true; Rs[1]=true; As[1]=true;
::(cs1==true&&Sc[2]==true&&Pss[2]==true&&Rss[2]==true&&Ass[2]==true&&S[2]==false)→S[2]=true;
Ps[2]=true; Rs[2]=true; As[2]=true;
fi
```

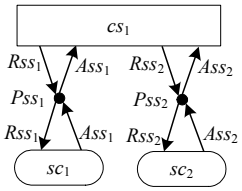


Fig.6 Current SA hypergraph H_1 of the system

图 6 系统当前 SA 超图 H_1

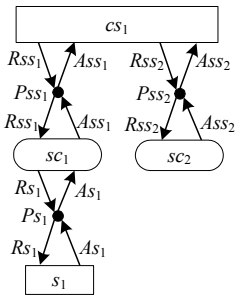


Fig.7 SA hypergraph H_2 after dynamic evolution

图 7 动态演化后的 SA 超图 H_2

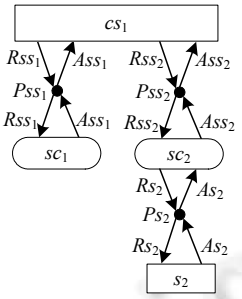


Fig.8 SA hypergraph H_3 after dynamic evolution

图 8 动态演化后的 SA 超图 H_3

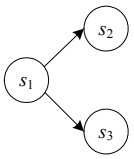


Fig.9 State transition relation of the mapping

图 9 映射成的状态转移关系

5.2.3 案例系统 SA 动态演化的条件状态转移系统

根据前面两小节,可建立该案例系统 SA 动态演化的条件状态转移系统 $M=(S,s_0,R,C,AP,L)$,其中,$S=\{s_0,s_1,\ldots,s_n\}$,$s_0=H_0,s_1=H_1,\ldots,s_n=H_n$,$C=AC$,$R\subseteq S\times C\times S$ 为如图 9 所示的状态转移关系集,AP 为给定原子命题集,$L:S\to 2^{AP}$ 为标记函数.这里,AP 通常采用以下形式:根据实际情况对每个状态中的变量或变量数组元素赋值为 true 或 false.

5.2.4 案例系统 SA 动态演化的活性验证

为了验证本文方法的有效性,这里以案例系统 SA 动态演化的活性为例,运用模型检测进行验证.SA 动态演化性质 p 是一个活性[10],是指存在一个 p 对应的逻辑命题公式 ϕ,使得:

$$\exists r\in P^*,\exists n\in\mathbb{N},H_0\xrightarrow[r]{}H_n\wedge H_n\models\phi,$$

其中,P 为 SA 演化规则集,r 为一个 SA 演化规则序列,H_0 为系统初始 SA 超图,$\mathbb{N}$为自然数集.上式也可用一个计算树逻辑 CTL(computation tree logic)公式表示:$EFH_n,H_n\models\phi$.即:存在一个将来演化到的 SA 实例 H_n,使得 ϕ 满足,其中,E,F 均为时态逻辑算子,E 表示存在一条路径,F 表示将来.例如在上述应用场景中,如果一个客户接入本系统,要求使用系统服务,则最终必须有一个服务器提供服务.

为了保证系统的可用性,必须保证该性质被满足.该性质用逻辑公式可表示为:假设系统当前 SA 超图为 H_0,对任意客户 $c_i\in H_0$,有:

$$\exists r\in P^*,\exists m\in\mathbb{N},H_0\xrightarrow[r]{}H_m\wedge H_m\models(\exists(cc_i\wedge Pc_i\wedge Pcs_i),(Pc_i(c_i,cc_i)\wedge Pcs_i(cc_i,cs_1))\in H_m)\wedge$$
$$(\exists(s_j\wedge sc_j\wedge Ps_j\wedge Pss_j),(Ps_j(sc_j,s_j)\wedge Pss_j(sc_j,cs_j))\in H_m),$$

其中,cc_i 表示一个客户连接件实体,s_j 表示一个服务器实体,Pc_i,Ps_j 为通信端口,P 为上述场景所定义的 SA 演化规则集,r 为一个 SA 演化规则序列,H_m 为系统动态演化过程中的某个 SA 超图实例.

该公式改写为 CTL 公式如下:

$$EF((\exists(cc_i \wedge Pc_i \wedge Pcs_i),(Pc_i(c_i,cc_i) \wedge Pcs_i(cc_i,cs_1)) \in H_m) \wedge$$
$$(\exists(s_j \wedge sc_j \wedge Ps_j \wedge Pss_j),(Ps_j(sc_j,s_j) \wedge Pss_j(sc_j,cs_j)) \in H_m)) = EFf.$$

为了运用模型检测技术验证该性质是SA动态演化活性,本文借助模型检测工具SPIN进行验证.因为SPIN主要检测线性时态逻辑LTL(linear temporal logic)公式,为了运用SPIN实现上述公式的验证,本文对上述公式进行了微调,将任意客户 c_i 加入到案例系统时的SA超图假定为系统的初始SA超图 H_0,则上述公式可用LTL公式表示如下:

$$F((\exists(cc_i \wedge Pc_i \wedge Pcs_i),(Pc_i(c_i,cc_i) \wedge Pcs_i(cc_i,cs_1)) \in H_m) \wedge$$
$$(\exists(s_j \wedge sc_j \wedge Ps_j \wedge Pss_j),(Ps_j(sc_j,s_j) \wedge Pss_j(sc_j,cs_j)) \in H_m)) = Ff.$$

即:在系统当前的SA超图 H_0 上,最终必须有一个服务器 s_j 为 c_i 提供服务.

本文用SPIN对上述公式进行了验证,表1为部分实验结果,其中,实验机器为Dell OptiPlex320,双CPU,主频1.60GHz,内存2GB.例如,当在SA动态演化过程中生成的状态数为40、状态转移数为81时,我们用SPIN验证上述活性公式所需的时间为0.315s;当生成的状态数为100、状态转移数为120时,用SPIN验证所需的时间为1.033s.两种情况下,上述活性的验证结果均为真.实验结果表明,在使用本文方法生成的条件状态系统上,上述活性性质始终成立.

Table 1 Some experiment results

表1 部分实验结果

状态数	转移数	验证时间(s)	结果
40	81	0.315	True
100	120	1.033	True

6 结束语

SA作为复杂软件系统设计中的核心部分,从较高的抽象层描述了系统的多个方面,从系统全局的组织结构到构件、通信、拓扑等.SA动态演化已经成为软件演化研究的关键问题,然而,如何验证动态演化的正确性是目前SA动态演化研究面临的更大挑战.因为随着外部环境和软件系统本身复杂度的不断增加,概念上的设计错误可能出现在任何高层模型上,即使用形式化的建模技术,也不能完全保证SA动态演化的正确性.为了确保软件系统及其动态演化的可靠性,形式化验证将成为SA动态演化研究中必不可少的组成部分.模型检测作为目前主流的形式化验证技术,已被逐渐应用于验证SA规范是否满足所期望的性质,但目前该方面的研究尚未考虑SA动态演化时的相关条件.模型检测SA动态演化的一个重要组成部分是建立SA动态演化的状态转移系统.本文着重研究如何建立SA动态演化的条件状态转移系统,首先讨论了用ASM对SA动态演化条件超图文法和条件状态转移系统进行统一的语义表示,给出了SA动态演化条件超图文法到条件状态转移系统的映射方法;接着证明了在该映射方法下,SA动态演化条件超图文法和条件状态转移系统的互模拟等价;最后,通过案例分析,讨论了如何运用本文提出的映射方法构建SA动态演化的条件状态转移系统,并运用模型检测技术验证了SA动态演化的活性.

本文提出的SA动态演化条件超图文法到条件状态转移系统的映射方法不仅可以实现与特定的模型检测工具无关,而且还能从理论上证明映射后的SA动态演化条件状态转移系统与映射前的SA动态演化条件超图文法是互模拟等价的.进一步的工作是运用所建立的条件状态转移系统,结合模型检测的其他技术,如符号模型检测等,验证带条件的SA动态演化的其他性质,并对SA动态演化的状态空间进行约简.

References:

[1] 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]

[2] Boehm B. The changing nature of software evolution. IEEE Software, 2010,27(4):26−28. [doi: 10.1109/MS.2010.103]

[3] Shaw M, Clements P. The golden age of software architecture. IEEE Software, 2006,23(2):31−39. [doi: 10.1109/MS.2006.58]

[4] Gomaa H, Hussein M. Software reconfiguration patterns for dynamic evolution of software architectures. In: Proc. of the 4th Working IEEE/IFIP Conf. on Software Architecture (WICSA 2004). Norway: IEEE CS Press, 2004. 79−88. [doi: 10.1109/WICSA.2004.1310692]

[5] Mens T, Magee J, Rumpe B. Evolving software architecture descriptions of critical systems. IEEE Computer, 2010,43(5):42−48. [doi: 10.1109/MC.2010.136]

[6] Zhang P, Muccini H, Li B. A classification and comparison of model checking software architecture techniques. Journal of Systems and Software, 2010,83(5):723−744. [doi: 10.1016/j.jss.2009.11.709]

[7] Clarke E, Grumberg O, Peled D. Model Checking. Cambridge: MIT Press, 2000.

[8] Xu HZ, Zeng GS. Dynamic evolution of software architectures based on hypergraph grammars. Journal of Tongji University (Natural Science), 2011,39(5):745−750 (in Chinese with English abstract). [doi: 10.3969/j.issn.0253-374x.2011.05.021]

[9] Xu HZ, Zeng GS, Chen B. Conditional hypergraph grammars and its analysis of dynamic evolution of software architectures. Ruan Jian Xue Bao/Journal of Software, 2011,22(6):1210−1223 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4017.htm [doi: 10.3724/SP.J.1001.2011.04017]

[10] Xu HZ, Zeng GS. Modeling and verifying composite dynamic evolution of software architectures using hypergraph grammars. Int'l Journal of Software Engineering and Knowledge Engineering, 2013,23(6):775−799. [doi: 10.1142/S0218194013500204]

[11] Börger E. The ASM refinement method. Formal Aspects of Computing, 2003,15(2-3):237−257. [doi: 10.1007/s00165-003-0012-7]

[12] 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(3):1−14. [doi: 10.1145/986710.986728]

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

[14] 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]

[15] 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]

[16] Ma XX, Cao C, Yu P, Zhou Y. A supporting environment based on graph grammar for dynamic software architectures. Ruan Jian Xue Bao/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]

[17] Chang ZM, Mao XJ, Qi ZC. Applying bigraph theory to self-adaptive software architecture. Chinese Journal of Computers, 2009, 32(1):97−106 (in Chinese with English abstract). [doi: 10.3724/SP.J. 1016.2009.00097]

[18] Dormoy J, Kouchnarenko O, Lanoix A. Using temporal logic for dynamic reconfigurations of components. LNCS, 2012,6921: 200−217. [doi: 10.1007/978-3-642-27269-1_12]

[19] Fiadeiro JL, Lopes A. A model for dynamic reconfiguration in service-oriented architectures. Software & Systems Modeling, 2013, 12(2):349−367. [doi: 10.1007/s10270-012-0236-1]

[20] Pelliccione P, Inverardi P, Muccini H. CHARMY: A framework for designing and verifying architectural specifications. IEEE Trans. on Software Engineering, 2009,35(3):325−346. [doi: 10.1109/TSE.2008.104]

[21] Lanoix A, Dormoy J, Kouchnarenko O. Combining proof and model-checking to validate reconfigurable architectures. Electronic Notes in Theoretical Computer Science, 2011 279(2):43−57. [doi: 10.1016/j.entcs.2011.11.011]

[22] Eckardt T, Heinzemann C, Henkler S, Hirsch M, Priesterjahn C, Schäfer W. Modeling and verifying dynamic communication structures based on graph transformations. Computer Science—Research and Development, 2013,28(1):3−22. [doi: 10.1007/s00450-011-0184-y]

[23] Rensink A. The GROOVE simulator: A tool for state space generation. LNCS, 2004,3062:479−485. [doi: 10.1007/978-3-540-25959-6_40]

[24] Hermann F, Kastenberg H, Modica1 T. Towards translating graph transformation approaches by model transformations. Electronic Communications of the EASST, 2006. http://journal.ub.tu-berlin.de/eceasst/article/view/20

[25] Lin HM, Zhang WH. Model checking: Theories, techniques and applications. Acta Electronica Sinica, 2002,30(S1):1907−1912 (in Chinese with English abstract). http://www.ejournal.org.cn/CN/Y2002/V30/IS1/1907

[26] Baier C, Katoen JP. Principles of Model Checking. Cambridge: MIT Press, 2008.

[27] Holzmann G. The model checker SPIN. IEEE Trans. on Software Engineering, 1997,23(5):279−295. [doi: 10.1109/32.588521]

附中文参考文献:

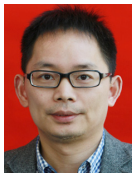
[8] 徐洪珍,曾国荪.基于超图文法的软件体系结构动态演化.同济大学学报(自然科学版),2011,39(5):45−750. [doi: 10.3969/j.issn.0253-374x.2011.05.021]

[9] 徐洪珍,曾国荪,陈波.软件体系结构动态演化的条件超图文法及分析.软件学报,2011,22(6):1210−1223. http://www.jos.org.cn/1000-9825/4017.htm [doi: 10.3724/SP.J.1001.2011.04017]

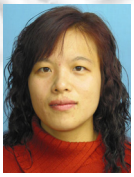
[16] 马晓星,曹春,余萍,周宇.基于图文法的动态软件体系结构支撑环境.软件学报,2008,19(8):1881−1892. http://www.jos.org.cn/1000-9825/19/1881.htm [doi: 10.3724/SP.J.1001.2008.01881]

[17] 常志明,毛新军,齐治昌.Bigraph 理论在自适应软件体系结构上的应用.计算机学报,2009,32(1):97−106. [doi: 10.3724/SP.J.1016.2009.00097]

[25] 林惠民,张文辉.模型检测:理论、方法与应用.电子学报,2002,30(S1):1907−1912. http://www.ejournal.org.cn/CN/Y2002/V30/IS1/1907



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



王晓燕(1980－),女,讲师,主要研究领域为软件体系结构,软件演化.

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