软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(4):747−759 [doi: 10.13328/j.cnki.jos.004751]
©中国科学院软件研究所版权所有.
E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel: +86-10-62562563

一种开放环境下软件在线演化一致性验证方法*

周 宇[1,2], 黄延凯[1], 黄志球[1], 吴维刚[3]

[1](南京航空航天大学 计算机科学与技术学院,江苏 南京 210016)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

[3](中山大学 信息科学与技术学院,广东 广州 510006)

通讯作者: 周宇, E-mail: zhouyu@nuaa.edu.cn, http://cs.nuaa.edu.cn

摘 要: 一致性保障技术是开放环境下软件在线演化研究的热点问题.区别于传统的基于图文法或基于体系结构描述语言(architectural description language,简称 ADL)对结构演化进行分析的途径,提出一种从行为角度采用层次式时间自动机对软件在线演化进行分析的方法,可支持对软件的时间属性、层次特征等直接建模.提出了层次平展化算法,将层次模型等价地转化为若干并行时间自动机模型,从而可应用现有模型检测工具针对演化规约进行一致性验证,并通过实验验证了所提方法的有效性.

关键词: 在线演化;时间自动机;一致性;模型检验

中图法分类号: TP311

中文引用格式: 周宇,黄延凯,黄志球,吴维刚.一种开放环境下软件在线演化一致性验证方法.软件学报,2015,26(4):747−759. http://www.jos.org.cn/1000-9825/4751.htm

英文引用格式: Zhou Y, Huang YK, Huang ZQ, Wu WG. Towards an approach of consistency verification for online software evolution in open environments. Ruan Jian Xue Bao/Journal of Software, 2015,26(4):747−759 (in Chinese). http://www.jos.org.cn/1000-9825/4751.htm

Towards an Approach of Consistency Verification for Online Software Evolution in Open Environments

ZHOU Yu[1,2], HUANG Yan-Kai[1], HUANG Zhi-Qiu[1], WU Wei-Gang[3]

[1](College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 210016, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[3](College of Information Science and Technology, Sun Yat-Sen University, Guangzhou 510006, China)

Abstract: Consistency assurance mechanism is of particular importance for online software evolution. Different from traditional approaches based on attributed graph grammar or architectural description languages, the paper investigates the problem from the behavioral perspective and proposes a novel verification technique based on hierarchical timed automata. The approach can directly support the modeling of temporal aspects as well as the hierarchical organization of composed systems in open environments. To make it feasible for consistency verification, this paper also proposes a flattening algorithm, translating the model into a set of synchronized serial timed automata so as to be fed into third-party tool-set. An example is presented throughout the paper to illustrate the new method. Experiments are carried out to complement the discussion and demonstrate the feasibility of the proposed approach.

Key words: online evolution; timed automata; consistency; model checking

随着计算机以及 Internet 技术的不断发展,软件运行环境的特征逐步由传统的封闭、静态、可控转向开放、动态、难控,云计算、Internet 计算等新型计算范型不断涌现.这种趋势对软件开发方式和使用方式亦产生深刻影响,由分散的自主构件组装或服务组合而成的系统逐渐增多.这种软件实体以开放、自主的服务形式存在于

* 基金项目: 国家自然科学基金(61202002, 61272083, 61379157); 江苏省软件新技术与产业化协同创新中心计划
收稿时间: 2014-06-29; 修改时间: 2014-10-14; 定稿时间: 2014-11-14

© 中国科学院软件研究所 http://www.jos.org.cn

Internet 各个节点之上,各个软件实体间通过协同机制进行跨网络互连、互通、协作和联盟.从技术角度看,这种软件不再如经典软件那样一蹴而就,它应能感知外部环境的动态变化,并随之进行静态的调整和动态的演化[1];从操作角度看,用户对软件的依赖性不断增长的现实,也需要应用系统提供服务的持续可用性和较高的满意度,这一需求使得对软件在线演化技术的研究更加重要.尽管有关程序运行时刻动态演化技术研究起步较早,例如面向过程式设计语言的 PODUS 系统[2]、基于 Java 的 DUSC 工具[3]、基于类型理论的 Ginseng[4]等,但这些研究往往从特定的程序设计语言或者编译技术的角度来展开,而针对开放环境下动态演化技术的研究相对较少,系统化的方法论和支撑工具较为欠缺.

在线演化是指软件运行时刻能根据情境信息的变化动态调整自身的行为,采取相应的动作以更好地为用户提供服务这一过程.在此过程中,系统的一致性不被破坏是保证演化正确性和有效性的必要前提,这是动态演化研究的重要问题.一致性是一个内涵较广的概念,在软件系统中可从不同角度和层次来考量,例如动态更新的模块与原有模块间类型的一致性[4,5]、软件实现与模型规约之间的一致性[6,7]、演化前后系统的结构与全局约束之间的一致性等[8,9].此外,还有与具体的应用语义紧密相关,由用户定义的一致性等,例如文献[10]提出可信演化操作集合的概念以间接地蕴含了演化过程的一致性这个属性.本文讨论的一致性是指在模型层次上,软件结构动态演化过程中能满足系统的全局约束,其行为和结果符合用户预期.

由于在开放环境具有动态、难控等特征,来自于外界的不确定性成为导致软件演化的重要因素,而具有良好适应能力的软件能根据情境信息的变化在不破坏规约一致性前提下调整行为以满足用户需求.如前所述,开放环境下软件的形态往往表现为一些自主服务或构件的组合集成,这种在线演化往往表现为新的服务(或构件)的添加或者替换.正因如此,传统的基于体系结构描述语言建模方法往往从结构出发描述其动态性,虽然体系结构与需求之间存在某种因果关系,然而这种关系却是隐含而非显而易见的,因此,仅从结构特征出发难以直接描述状态及行为的转换.再者,对终端用户而言,其往往更关心与系统的直接交互感受而非系统结构的变化,因此这个鸿沟需要一种建模机制能对软件系统从行为的角度直接进行分析和度量.其次,开放环境下的服务(或构件)可以组合的形式为用户提供增值服务,这一组合实质上是一个递归过程,即,组合后的服务亦可视作为单个服务集成至其他服务之中,这种复合方式具备抽象层次化的特征,而这种特征也需要一种建模方法能够对之加以直接支持.再次,时间相关属性是软件服务质量的重要考虑因素[11],虽然开放环境下的软件系统对时间硬实时性要求要低于传统的嵌入式系统,但其时间作为一种非功能性需求往往是系统演化的主要驱动因素[12],而当前无论是基于 ADL 还是基于进程代数的方法大多不支持对时间属性进行直接建模.

鉴于以上考虑,提出一种基于层次式时间自动机的在线演化分析方法,包含了时钟变量、精化函数等元素,从而可以直接支持对时间属性、层次特征、状态变迁、演化行为等进行建模和一致性分析,以弥补现有方法的不足.本文在初步工作[13]的基础上作了进一步的扩展,补充了层次平展算法,给出了相应的伪代码实现,并结合启发案例用实验论证了所提出方法的可行性.

本文第 1 节首先简介背景知识,包括顺序时间自动机、层次式时间自动机的概念以及启发案例.第 2 节介绍在线演化行为建模的方法.第 3 节给出层次化平展算法和一致性模型验证过程.第 4 节对本文所提方法进行初步的讨论.第 5 节介绍国内外相关工作.第 6 节总结本文并展望未来工作.

1 背景及启发案例

时间自动机[14]是在经典的ω自动机[15]基础之上,为描述现实世界中的反应式系统(reactive system)所扩充的一种自动机模型,其扩展之处在于增添了时钟符号,用来记录流逝的时间,且在迁移触发后,时钟可被重置为 0 并在新状态重新计时.这样,时间自动机的状态描述扩充为一个二元组,分别是所处的位置状态和时刻状态.由于时间本身是一个连续变量,时间自动机就有无穷多个状态,通过应用区域(region)约减算法,可以将这些无穷多状态划分为有限个离散的等价状态类,从而可以应用已有的工具和算法对其进行模型检验.时间自动机在诸多领域,诸如实时 UML、Web service、物联网等有着重要应用[16,17].层次式时间自动机是在上述时间自动机的基础上进一步的扩展[18],观察到现实世界的层次模型的普遍性,应用精化函数描述状态间的层次关系,可直接对

© 中国科学院软件研究所 http://www.jos.org.cn

复杂状态——例如包含多个区域的复合状态及其转换——进行建模.考虑到现实软件系统的层次式特征,这种模型更自然地符合人的思维模式.本文提出采用层次式时间自动机作为对在线演化行为建模的基础,首先给出顺序时间自动机和层次式时间自动机的定义,然后采用一个服务集成的场景实例来解释建模过程.

定义 1. 顺序时间自动机定义为七元组$\langle S,S_0,\sigma,C,Inv,M,T\rangle$,其中,

- S 为有穷状态集合;
- S_0 为初始状态;
- $\sigma:S\to\{BASIC,COMPOSITE,INIT\}$ 是类型函数,该函数返回一个状态的类型,具体的类型包括基本状态(*BASIC*)、复合状态(*COMPOSITE*)、初始状态(*INIT*);
- C 是时钟(clocks)集合;
- Inv 是状态的不变式函数;
- 返回状态的不变式 M 是消息集合,包含两类消息,分别是同步消息(由!和?协同完成)和异步消息;
- T 是转移集合,为集合 $S\times M\times CC\times 2^C\times S$ 的子集,其元素 t 亦可标记为 $t=s\xrightarrow{m,g,r}s'$.

定义 2. 层次式时间自动机被定义为三元组$\langle F,E,\rho\rangle$,其中,

- F 是顺序时间自动机集合,其中任意两个顺序时间自动机状态集合不相交;
- E 是触发事件集合,该触发事件亦包含相应的卫式条件以及时钟重置动作,具体形式可标记为
$$E\subseteq\left(\bigcup_{A\in F}S_A\times\bigcup_{A\in F}CC_A\times\bigcup_{A\in F}2^{C_A}\times\bigcup_{A\in F}\Sigma_A\times\bigcup_{A\in F}S_A\right);$$
- $\rho:\bigcup_{A\in F}S_A\to 2^F$ 是精化函数,将一个复合状态映射到一组顺序时间自动机集合,从而在层次式时间自动机中构建了树形的层次关系.对于基本状态,该函数返回空集.

根据精化函数,可递归定义 $\rho^*(s)=\rho(s)\cup\left(\bigcup_{s_i\in S_{\rho^*(s)}}\rho(s_i)\right)$,表示状态 s 所有的后代顺序时间自动机的集合.

为了讨论方便,我们定义在后代顺序时间自动机集合中的特殊子集:叶节点顺序自动机,这类自动机位于树结构的末端,$\rho_{leaf}(s)=\{A'|A'\in\rho^*(s)\wedge(\forall s'\in A',\rho(s')=\varnothing)\}$.针对复合状态的转移,其源和目标往往牵扯到多个正交状态.为此,我们定义源限制函数(source restriction function)和目标限制函数(destination restriction function).源限制函数和目标限制函数分别定义为 $sr:t\to S$ 和 $tr:t\to S$,将转移 t 映射为一个指定的状态集合 S.由于复合状态的存在使得一个系统在某个时刻可以有多个活动状态,我们把这样的活动状态集合称为格局(configuration).

下面以一个常见的电子商务中订单处理服务及其演化为启发案例,介绍行为建模方法.该场景根据文献[10]中的应用实例调整而来,由于订单处理牵扯多方服务协作组合,又对处理时间有较高要求,因此具备一定普适性.一个完整的电子商务过程往往包含客户端服务、商品提供端服务、物流端服务,而商品提供端服务又进一步包含支付管理服务和商品管理服务.为了降低不必要的复杂性,本案例简化客户端服务,省略物流端服务,以商品提供端服务为重点,阐述基于层次式时间自动机演化行为建模过程.

2 演化行为建模

客户发送查询请求,根据反馈结果查找是否包含目标商品:如果存在,则和提供端的订单管理(*orderMgr*)交互采购;否则,更改请求返回.订单管理的支付管理子服务(*payMgr*)会经过一系列的身份认证(*auth*)、加密(*encrypt*)、支付(*paying*)等,每个过程都有一定的完成时间限制(由 *clk*1 来描述,时间单位为分钟);商品管理子服务(*itemMgr*)部分将客户感兴趣的物品在一定的时间内由可用状态(*available*)锁定为预定状态(*reserve*),并根据结果或流逝时间更改为卖出状态(*bought*)或还原为可用状态,时间属性由变量 *clk*2 来描述,单位与 *clk*1 相同.两个子系统通过管道(*channel*)机制来同步并通知客户结果.具体转换、时间约束及不变式等细节如图 1 所示,图中带点状态为初始状态;*orderMgr* 为复合状态,包含 *payMgr* 和 *itemMgr* 两个子状态;其余为基本状态.图 2 展示了该案例的层次分解部分,由于 *orderMgr* 是复合状态,根据上述定义,应用精化函数可以得到:

$$\rho(orderMgr)=\{payMgr,itemMgr\}.$$

© 中国科学院软件研究所 http://www.jos.org.cn

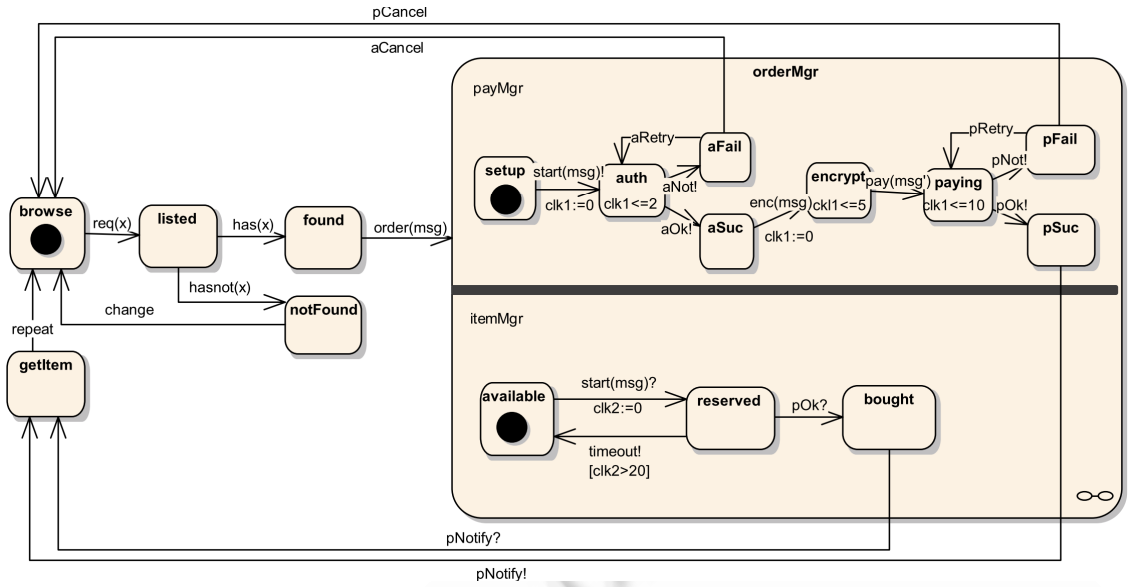


Fig.1 Order process scenario illustration

图 1 订单处理服务案例

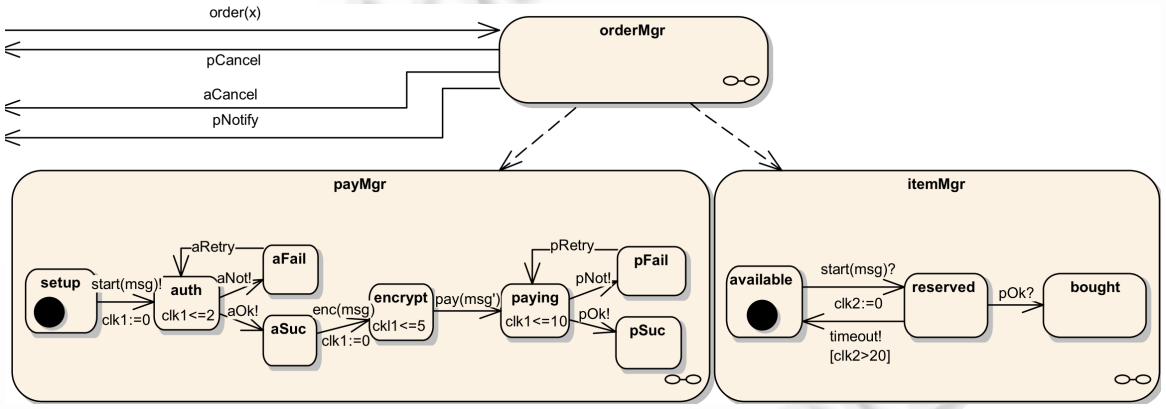


Fig.2 Hierarchy decomposition illustration (part)

图 2 层次分解示例(部分)

如前所述,开放环境下存在诸多不确定性,可能存在恶意攻击等,服务集成者意图加强认证过程,动态添加短信认证行为,这就需要对原有系统在线演化,由于新认证过程存在有效期概念,可以借助 *clk*1 对之建模,本例中设置为有效期不超过 8 分钟(*clk*1<=8).新的行为过程如图 3 所示,由于客户端及商品管理子服务无需变化,图中省略未演化部分.

图 1 和图 3 分别针对演化前后的状态行为模型(源模型和目标模型).由于在线演化过程牵扯到数据等状态信息,区别于离线模式,其演化过程受当前活动状态即格局的影响,必须在确保不破坏状态一致性的前提下进行.这方面的经典工作包括 Kramer 等人提出的静态(quiescent state)机制[19]、Vandewoude 等人提出的稳态(tranquillity status)机制[20].由于稳态机制较静态而言要求更宽松,而且对服务的扰度影响(disruption)较低,因此本文采用稳态机制作为不破坏状态一致性的基础.根据稳态的定义[20],我们给出一个在线演化的前提:当前格局中的状态元素与演化相关服务的稳态集合无关时,可实施演化过程.本实例中,由于演化相关的服务为认证服

© 中国科学院软件研究所 http://www.jos.org.cn

务,如果当前格局中包含相关状态,例如 *auth*,*aFail*,*aSuc* 等,则不可以触发演化过程;反之可以.

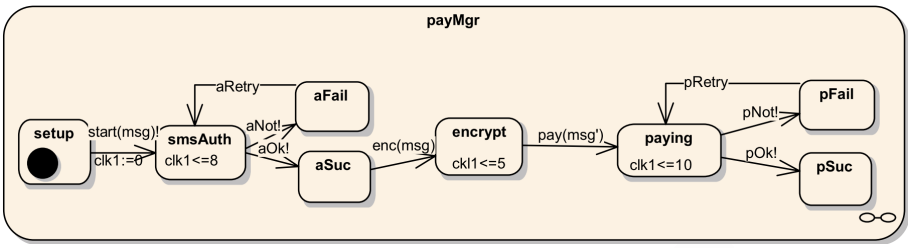


Fig.3 Partial state transition after evolution

图 3 演化后状态转换图(部分)

对于每一种满足稳态条件的合法格局都存在相关的演化模型用于关联源模型和目标模型,如图 4 所示,我们以格局〈*payMgr.setup*,*payMgr.clk*1〉,〈*itemMgr.available*,*itemMgr.clk*2〉〉为例展示演化模型,演化动作为 *evolve*,相关参数以消息的形式传递.

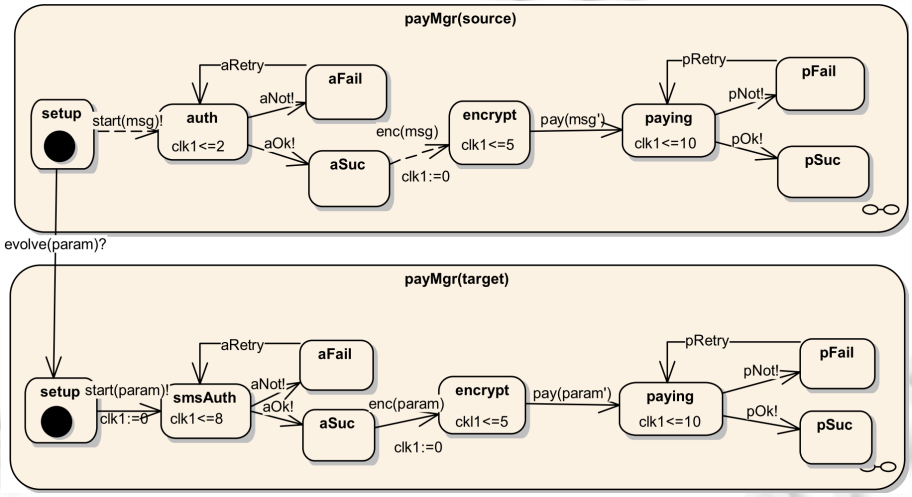


Fig.4 Partial evolution illustration

图 4 演化过程示意(部分)

由于该演化动作并未牵扯到 *itemMgr* 子服务部分,则对于〈*itemMgr.available*,*itemMgr.clk*2〉的状态不受演化影响,因此为简单起见,图 4 只展示了演化相关的部分.稳态机制要求被替换节点与环境的交互被阻塞直至替换完成,因此图中的上部相关转换 $setup \xrightarrow{start(msg)!,clk1:=0} auth$ 以及 $aSuc \xrightarrow{enc(msg),clk1:=0} encrypt$ 等标识为虚线,代表受限.

3 演化验证

下面我们对上述模型进行验证,验证过程可分为演化前验证、演化过程验证和演化后系统验证这 3 个部分,待验证属性由计算树逻辑(CTL)来描述.由于现有的模型检测工具难以直接对层次式时间自动机模型进行验证,因此在对模型进行验证之前需要对其进行转换工作,即:对层次进行平展化,将其转换为一组并行执行的顺序时间自动机集合.本节首先介绍层次平展化算法,然后展示一致性验证过程.

3.1 层次平展化方法

从实用的角度考虑,我们选取了目前主流的时间自动机模型验证工具 UPPAAL[21]的语法作为该算法输出

© 中国科学院软件研究所 http://www.jos.org.cn

的格式,在 UPPAAL 中,每个时间自动机称为 *template*,状态节点称为 *location*.该算法主要分为以下几个步骤,相应的伪代码表示如算法 1 所示.

(1) 针对层次式时间自动机 M 中的每一个顺序时间自动机 F,将其视作一个独立的时间自动机,针对每一个不是根自动机(root automaton)的 F,相应地添加一个 *location*,并将其标记为 *inactive*;

(2) 针对每一个进入复合状态 S 的转移 t,从 *inactive* 状态添加这些复合状态内部相应的默认初入状态,触发事件同于 t 的触发事件.如果复合状态 S 本身是初始状态,那么该状态中的默认初入状态被标记为初入状态;否则,将在上一步骤添加的 *inactive* 状态标记为初入状态;

(3) 为同步各顺序自动机之间的并行执行,将进入复合状态 S 的每一个转移事件扩充为广播通道(broadcast channel)的发出类型(用!标记);而在其相应的子状态顺序时间自动机中,把从 *inactive* 状态到默认初入状态的事件扩充标记为该广播通道类型的接受类型(用?标记);

(4) 针对每个非根节点顺序时间自动机,添加一个特殊状态,并从源于复合状态转移的相应的源限制函数值中的每个状态添加相应的转移到该特殊状态,并将该转移对应的事件扩充为广播通道类型,并将上面到达新状态的转移事件扩充为该广播通道类型的接受类型;

(5) 从上一步骤每个顺序时间自动机中新添加的特殊状态出发引出新的转移,到达步骤 1 中所添加的对应的顺序时间自动机中的 *inactive* 状态.

算法 1. 层次平展算法.

输入:M,层次时间自动机模型;

输出:T,顺序时间自动机集合.

```
begin
  T←∅; Hashmap map;
  forall the A_i∈M.F do
    create atemplate t; add locations and transitions based on A_i.S and A_i.Σ;
    if A_i≠Aroot then
      add A_i_inactive location in t; mark A_i_inactive as committed;
    T←T∪{t}; map.add(A_i,t);
  forall the A_i∈M.F do
    forall the s∈A_i:S∧σ(s)=COMPOSITE do
      forall the A_i∈ρ(s) do
        temp←map.get(A_i);
        forall the in_tr∈{incoming transitions to s} do
          add transition trans from A_i_inactive to the entry location in temp;
          augment the action of trans with broadcast channel;
        if s is initial then
          default entry location in A_i is marked as initial in temp;
        else
          A_i_inactive is marked as initial in temp;
      map.update;
      T.update;
  forall the A_i∈M.F∧A_i≠Aroot do
    temp←map.get(A_i);
    add join location A_i_join in temp;
    mark A_i_join as committed;
```

© 中国科学院软件研究所 http://www.jos.org.cn

forall the $t \in \bigcup_{A_i.M.F} \sum A_i$ **do**

forall the $s \in sr(t) \wedge \sigma(s)=COMPOSITE$) **do**

forall the $A_j \in \rho^*(s)$ **do**

$temp=map.get(A_j)$;

add transitions tt from each locations generated by A_j:S to A_j_join in $temp$;

add transitions tt' from A_j_join to $A_j\_inactive$;

associate tt with channels, guards and clock resets based on t;

$map.update$;

$T \leftarrow \bigcup_{A_j \in M.A} map.get(A_i)$

end

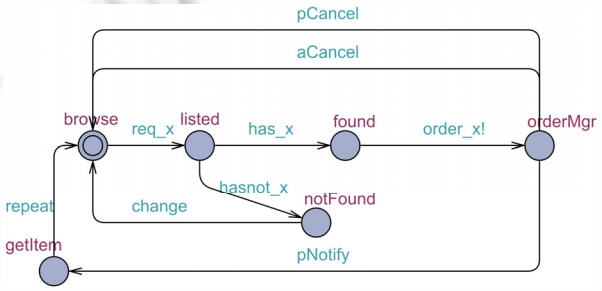
3.2 一致性验证

如前所述,将层次式时间自动机转换为系列并发的时间自动机之后,即可应用现有的模型检测工具对其进行一致性验证工作.验证可从 3 个方面开展:演化前验证、演化后验证以及演化过程验证.我们用 CTL 来描述待验证属性,采用 UPPAAL(版本号为 4.1.15,学术授权版)来检测系统设计是否与属性规约一致.我们定义两类属性规约:一类是安全性(safety)规约,一类是活性(liveness)规约[22].

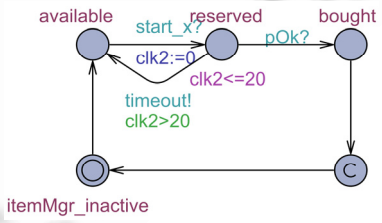
- 对于安全性规约,我们希望无论系统演化前后,都不会出现客户已成功付款而相应商品却依然处于可购买状态,即,$P1$:A[](¬($payMgr.pSuc \wedge itemMgr.available$));
- 对于活性规约,我们希望系统满足这样的性质:只要客户需求的某种商品存在,那么最终该商品或者被买走或者可再售,而不会一直处于预定状态,用 CTL 描述为

$$P2:A\langle\rangle((orderMgr.found) \rightarrow (itemMgr.bought \vee itemMgr.available)).$$

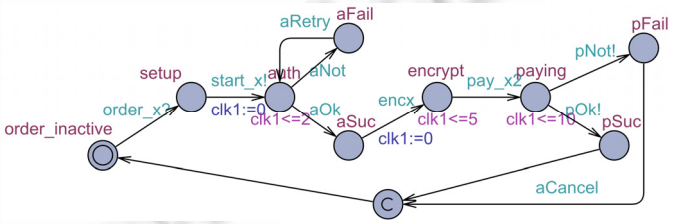
根据上一节提出的层次平展化算法,我们把演化前的系统转换为 UPPAAL 的输入,得到 3 个并行的时间自动机,如图 5 所示.由于演化状态和演化前状态图类似,限于篇幅,文中不再列出.



(a) *orderMgr* 对应 Template



(b) *itemMgr* 对应 Template



(c) *payMgr* 对应 Template

Fig.5 UPPAAL models after flattening

图 5 UPPAAL 验证平展后系统示例

© 中国科学院软件研究所 http://www.jos.org.cn

根据系统规约和所得模型,我们针对演化前的系统进行了验证,发现演化前的系统满足这两个规约,即,系统行为与该安全性和活性规约相互一致.

接下来,我们对演化后的系统模型进行验证,检测结果为模型满足第 2 个属性规约,而第 1 个属性规约不成立.通过观察反例我们发现:由于演化后的短信验证服务的验证时效要求过长,结合后续加密过程、付款过程时间不变式,会导致在最终付款之前由于商品预定状态超时(本例中预定时间控制为 20 分钟)自动回退到可售卖状态,从而会出现客户付款成功但是商品处于可再卖状态(即 $payMgr.pSuc \vee itemMgr.available$);而观察时间自动机可以发现:一旦处于这样的状态,由于 $payMgr.pSuc$ 的付款通知(信道 pOK)无法同步,进一步会发生死锁现象.在这一过程中,每个状态的时间控制是合理的,但依然会发生与需求规约不一致的情况,这就需要我们更改设计方案.我们发现,要求客户在经过认证之前就将商品状态更改为预定态的设计是不合理的,因此调整了我们的模型,将商品预定态同步过程由经过认证前改为认证成功之后.而演化前的模型之所以没有检测出这个问题,是因为原简单认证所需时间较短,包括后续的加密、付款等过程等总时间要求都小于预定态的时间上限.修改后的设计(*payMgr*)如图 6 所示(只展示改动部分),我们对该设计演化前后的方案进行了重新验证,其行为与上述两个规约一致.

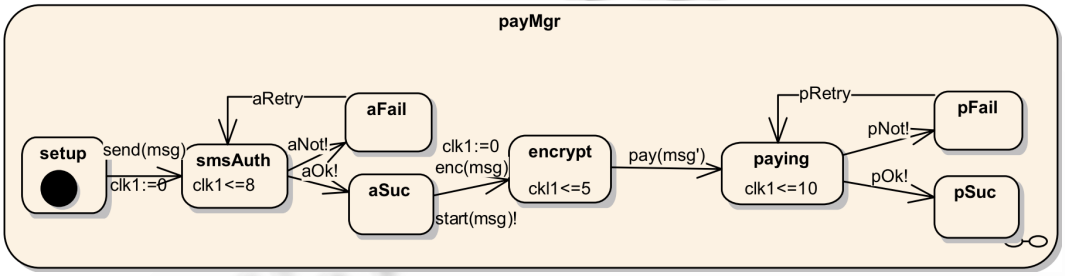


Fig.6 Revised design of *payMgr*

图 6 更改 *payMgr* 部分设计

下面我们对演化过程进行检测.由于更改了设计方案,根据图 6 我们进行了调整,将更新后的状态行为转换为 UPPAAL 模型输入,并针对上述两个全局规约进行了验证.更改设计后,系统的演化过程与规约需求一致,即,包含格局$\langle\langle payMgr.setup,payMgr.clk1\rangle,\langle itemMgr.available,itemMgr.clk2\rangle\rangle$的行为路径满足安全性和活性要求.图 7 展示了 UPPAAL 中演化过程自动机模型,由于稳态机制要求原认证服务处于非服务状态才能替换,因此与之相关的转换被设置为失活,演化前的用户数据信息作为参数通过信道传给演化后的服务.

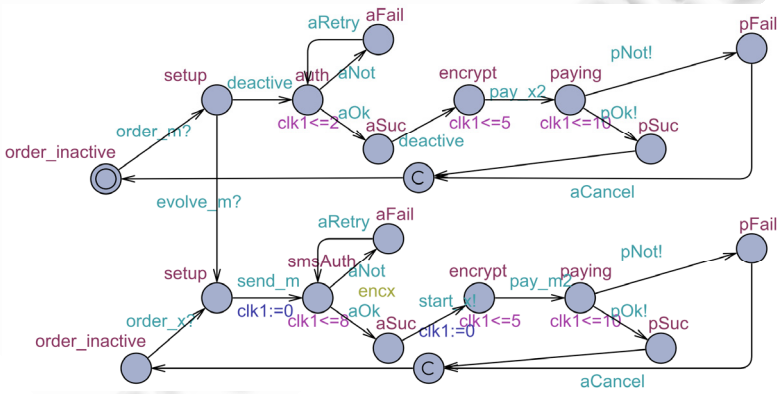


Fig.7 Evolution model after revision

图 7 更改后的演化模型

© 中国科学院软件研究所 http://www.jos.org.cn

为了考量所提方法的可扩展性,我们针对该实例模型采用参数化调节方法,将商品数量作为参数(标记为*N*),实验记录随 *N* 的增大所增加的状态、所消耗的存储空间以及验证所需时间.实验的硬件平台为 PC 机配置为 i5-2.4GHz 处理器,4G 内存,软件平台为 Windows 7 专业版,UPPAAL 版本为 4.1.15 学术版.演化前、演化后以及演化过程中的实验数据见表 1.其中,我们对更改后的演化模型实验部分列在表 1 的下半部分.由于演化后的模型不牵扯到演化前的验证,因此这部分数据不变.针对属性 *P*1,由于在初始设计的验证过程中并不成立,UPPAAL 会在找到反例之后停止验证,因此,所统计出来的数据包括状态数量、时间及空间消耗,是指在空间遍历过程中遇到(包括)反例之前的数据.待演化模型如上修改之后,我们重新进行验证,所得数据见表 1,修改前后分别由上下两部分表示(其中,*N*=number of items,R/V=residential/virtual,NC=not concluded,OM=out of memory).

Table 1 Summary of experiments data

表 1 实验数据小结

N	Before evolution			During evolution			After evolution		
	States	R/V memory (MB)	Time (s)	States	R/V memory (MB)	Time (s)	States	R/V memory (MB)	Time (s)
1: *P*1	108	9.6/29.5	0.02	26	9.6/29.7	0.01	23	9.5/29.4	0.01
1: *P*2	112	9.7/30.1	0.02	374	9.9/30.4	0.02	208	9.7/29.7	0.02
2: *P*1	204	9.6/29.6	0.02	31	9.6/29.7	0.01	28	9.5/29.5	0.01
2: *P*2	212	9.8/30.3	0.02	818	10.5/31.3	0.02	388	9.7/30.1	0.02
4: *P*1	1 068	9.7/29.7	0.03	44	9.9/30.1	0.01	41	9.6/29.6	0.01
4: *P*2	1 100	9.9/30.5	0.03	5 210	10.2/30.7	0.03	2 044	9.9/30.5	0.02
8: *P*1	58 668	13.5/36.6	0.39	82	9.8/30.2	0.01	79	9.7/29.8	0.01
8: *P*2	59 180	13.8/37.1	0.39	334 970	24.7/55.6	1.73	115 804	17.3/43.5	0.55
12: *P*1	2 174 940	204.9/417.9	16.6	136	9.9/30.3	0.01	133	9.8/30.1	0.01
12: *P*2	2 179 036	206.0/420.1	16.5	12 938 948	851/1710	92.3	4 345 780	398/803.2	30.1
14: *P*1	19 328 508	1779/3584	171.5	169	9.9/30.3	0.01	166	9.8/30.2	0.01
14: *P*2	19 344 892	1796/3605	181.7	NC	OM	NC	NC	OM	NC
N	Before evolution			During evolution revised			After evolution revised		
	States	R/V memory (MB)	Time (s)	States	R/V memory (MB)	Time (s)	States	R/V memory (MB)	Time (s)
1: *P*1	108	9.6/29.5	0.02	312	9.6/29.7	0.02	208	9.6/29.5	0.01
1: *P*2	112	9.7/30.1	0.02	326	9.9/30.5	0.02	216	9.7/29.8	0.01
2: *P*1	204	9.6/29.6	0.02	600	9.7/29.7	0.02	400	9.6/29.5	0.02
2: *P*2	212	9.8/30.3	0.02	626	9.9/30.5	0.02	416	9.8/30.3	0.02
4: *P*1	1 068	9.7/29.7	0.03	3 192	9.9/30.1	0.03	2 128	9.7/29.8	0.03
4: *P*2	1 100	9.9/30.5	0.03	3 290	10.1/30.6	0.03	2 192	9.9/30.6	0.03
8: *P*1	58 668	13.5/36.6	0.39	175 992	17.2/42.6	0.92	117 328	16.4/42.5	0.62
8: *P*2	59 180	13.8/37.1	0.39	177 530	17.5/43.1	0.91	118 352	16.6/43.0	0.64
12: *P*1	2 174 940	204.9/417.9	16.6	6 524 808	434.4/875.7	51.4	4 349 872	320.2/648.4	33.2
12: *P*2	2 179 036	206.0/420.1	16.5	6 537 098	435.4/877.6	51.9	4 358 064	321.2/650.4	33.5
14: *P*1	19 328 508	1779/3584	171.5	NC	OM	NC	NC	OM	NC
14: *P*2	19 344 892	1796/3605	181.7	NC	OM	NC	NC	OM	NC

从表 1 可以看出:

- 按照原有的设计,演化过程中不满足规约 *P*1;随着模型规模的扩大,由于反例的存在,在验证 *P*1 的过程中,模型检测工具总能在有限时间内找出该反例;
- 在模型更改之后,*P*1,*P*2 规约总能被满足;随着参数的增大,系统规模不断变大,产生的状态空间增长很快超出了系统的能力,在表 1 中,用 OM 和 NC 来表示.

4 讨 论

前文给出了基于层次式时间自动机对服务在线演化建模和一致性验证方法,需要明确的是:

- 本文所指的状态并不是服务内部算法层次的细粒度数据状态,而是在接口层次外部用户可见的粗粒度状态行为,这是由开放环境下服务特征所决定的;
- 此外,我们借鉴了稳态概念来作为演化实施的基础,但是稳态并非免费,其到达是需要一定的前提条件和时间开销的,尽管本例中的稳态条件并不会对最终待验证性质产生直接影响,因为商品管理子服务

© 中国科学院软件研究所 http://www.jos.org.cn

(*itemMgr*)中的预定状态计时是由验证服务通过后才同步触发,也就是说,其计时开始瞬时已无需保持稳态条件.有统计证明:只有在极少数的并发场合,稳态条件是不可达的.但更一般意义的研究超出本文讨论范围,关于稳态可达性的分析感兴趣的读者可参见文献[20];

- 再次,我们的启发案例中状态之间需要传递信息,例如客户请求的商品名等,但时间自动机模型检测工具 UPPAAL 本身并不支持消息的传递,我们在实现过程中采用了一个变通方案.因为消息种类是有限的,我们把这些消息设为变量,在转换的同时对这些变量进行赋值或取值操作,而这些值对应着消息的编码,从而实现消息在不同状态之间的传递.

本文重点关注演化行为一致性验证,并不讨论服务可替换的保障机制,关于行为可替换协议分析可参见文献[23,24]等.采用行为状态建模机制描述在线演化,则可能存在多种转换方案,分别对应着不同的源状态、目标状态和稳态条件.尽管这为建模带来了一定的复杂性,但是这种模型却可以深入分析内部状态的转换过程,是传统的基于图或者 ADL 等方式所不能直接表达的.

5 相关工作

鉴于一致性问题在动态演化过程中的重要性,有相当多的工作从不同角度对此展开研究,由于篇幅所限,只考察与本项目研究角度相关性较大的一些代表性工作.由于开放环境下的动态演化研究尚未达到成熟阶段,存在不少问题,我们主要从这些工作的特点和不足两方面展开讨论:文献[6]中应用反射机制在系统的结构规约(结构视图)和运行实体(运行视图)之间建立了一层因果关系,这样,体系结构方面的约束可显示化地存在于运行实体中,并指导演化过程,系统与规约间的一致性得到了较好的保证,但规约本身与演化逻辑这些模型设计阶段的产物却较少提及;文献[7]采用运行时体系结构对象作为演化过程的协同实体,进而利用面向对象思想中的继承和多态的概念来保持这种演化后的对象一致性,然而这种机制只能保证类之间的兼容性,对于演化的行为是否满足全局约束这类问题缺少模型层次的验证.在模型层次验证方面的工作较有代表性的包括文献[9,25–29],其中,

- 文献[9]采用属性图文法描述软件的逻辑结构以及用图转换规则描述演化策略,在此基础上,利用 AGG 等第三方工具对模型进行检测,具有直观、形象等特点,但可扩展性较弱,其中大量用到的图的同态映射,其本质属于成员类问题(membership problem),是 NP 完全的[30];
- 文献[25]尝试用 Petri-Net 建模组件行为,用时态逻辑描述系统规约,然后用模型检测工具检测该状态模型和规约间的一致性.尽管该方法与本文类似,也是从行为入手建模分析,但其并未考虑时间因素;其次,该方法也未考虑待验证系统的层次特征;
- 文献[26]提出一种程序融合技术,将演化前后的程序绑定在一起,并生成相应的规约.这样,将演化前后程序一致性验证问题划归为传统的规约模型验证问题,并可利用第三方工具进行验证.该工作是从程序代码级别的角度展开程序验证研究,类似的工作包括文献[29]等.这些工作依赖于特定的程序设计语言或编译器,对演化系统有着完全的掌控,这个前提假设在开放环境下并不成立,难以应用于开放环境下的软件一致性验证;
- 文献[27,28]从模型角度对 publish-subscribe 体系结构风格的软件系统相关属性验证问题展开了研究,考虑了该体系结构风格的约束及其应用领域特征,并采用定制的状态生成算法.相较于传统的验证途径[31],降低了状态数量,提高了可验证系统规模.但其目标是验证特定的体系结构风格是否与设计需求规约一致,而非本文讨论的演化过程的行为一致性;
- 文献[32]则从运行时数据监控入手,建立可靠性度量模型,采用概率模型检测方法来验证系统行为是否满足需求,该方法侧重于软件可靠性分析.

6 结 语

开放环境下的系统演化往往表现为服务组件的动态加入或替换从而引起体系结构的更新,而体系结构有

© 中国科学院软件研究所 http://www.jos.org.cn

多种视角,本文从行为的角度对软件在线演化展开研究,针对开放环境下软件的层次、时序等特征采用基于层次式时间自动机的方法对在线演化过程进行建模和一致性分析,并通过案例论证了该方法的可行性.尽管本文的工作主要集中在系统的设计验证阶段,但由于开放环境下软件的开发设计阶段和运行阶段边界已不明显,对于运行阶段动态发现的服务,可以经过一致性验证之后动态地更新现有系统[33];基于语义网络的服务描述及动态替换方法等,则在实现层面上提供了使能机制.同时,所提出的模型方法和组件状态一致性替换方法是互补的,文中采用了稳态机制作为演化可进行的基础条件,但是对其他的相关方法是开放的.例如,文献[34]提出了一个更细粒度、对服务干扰度更低的版本——一致性替换机制,我们准备在将来的工作中加以采用.此外,本文采用了一种间接转换的方式,对层次式时间自动机的验证进行支持,在未来的工作中,我们准备研究、开发可直接对层次式时间自动机进行建模和分析的工具.

References:

[1] Lü J, Ma XX, Tao XP, Xu F, Hu H. Research and progress on Internetware. Science in China (Series E), 2006,36(10):1037−1080 (in Chinese with English abstract).

[2] Frieder O, Segal ME. On dynamically updating a computer program: From concept to prototype. Journal of Systems and Software, 1991,14(2):111−128. [doi: 10.1016/0164-1212(91)90096-O]

[3] Orso A, Rao A, Harrold MJ. A technique for dynamic updating of Java software. In: Proc. of the Int'l Conf. on Software Maintenance. IEEE, 2002. 649−658. [doi: 10.1109/ICSM.2002.1167829]

[4] Hicks M, Nettles S. Dynamic software updating. ACM Trans. on Programming Languages and Systems (TOPLAS), 2005,27(6): 1049−1096. [doi: 10.1145/1108970.1108971]

[5] Stoyle G, Hicks M, Bierman G, Sewell P, Neamtiu I. Mutatis mutandis: Safe and flexible dynamic software updating. ACM Trans. on Programming Languages and Systems, 2007,29(4):1−70. [doi: 10.1145/1255450.1255455]

[6] 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).

[7] Yu P, Ma XX, Lü J, Tao XP. A dynamic software architecture oriented approach to online evolution. Ruan Jian Xue Bao/Journal of Software, 2006,17(6):1360−1371 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1360.htm

[8] Oreizy P, Medvidovic N, Taylor R. Runtime software adaptation: Framework, approaches, and styles. In: Proc. of the 30th Int'l Conf. on Software Engineering. ACM Press, 2008. 899−910. [doi: 10.1145/1370175.1370181]

[9] Xu HZ, Zeng GS, Chen B. Conditional hypergraph grammars and its analysis of dynamic evolution of software architecture. 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] Zeng J, Sun HL, Liu XD, Deng T, Huai JP. Dynamic evolution mechanism for trustworthy software based on service composition. Ruan Jian Xue Bao/Journal of Software, 2010, 21(2):261−276 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3735.htm [doi: 10.3724/SP.J.1001.2010.03735]

[11] Kazhamiakin R, Pandya P, Pistore M. Timed modelling and analysis in web service compositions. In: Proc. of the 1st Int'l Conf. on Availability, Reliability and Security (ARES 2006). IEEE, 2006. 7. [doi: 10.1109/ARES.2006.134]

[12] De Lemos R, Giese H, Müller HA, *et al*. Software engineering for self-adaptive systems: A second research roadmap. In: Proc. of the Software Engineering for Self-Adaptive Systems II. Berlin, Heidelberg: Springer-Verlag, 2013. 1−32. [doi: 10.1007/978-3-642-35813-5_1]

[13] Zhou Y, Ge JD, Zhang PC. Hierarchical timed automata based verification of dynamic evolution process in open environments. In: Proc. of the Int'l Conf. on Software and Systems Process (ICSSP). 2014. 144−148. [doi: 10.1145/2600821.2600838]

[14] Alur R, Dill D. A theory of timed automata. Theoretical Computer Science, 1994,126(2):183−235. [doi: 10.1016/0304-3975(94)90010-8]

[15] Safra S. On the comlexity of ω automata. In: Proc. of the 29th Foundations of Computer Science. IEEE, 1988. [doi: 10.1109/SFCS.1988.21948]

[16] Li LX, Jin Z, Li G. Modeling and verifying services of internet of things based on timed automata. Chinese Journal of Computers, 2011,34(8):1365−1377 (in Chinese with English abstract). [doi: 10.3724/SP.J.1016.2011.01365]

© 中国科学院软件研究所 http://www.jos.org.cn

[17] Chen W, Xue YZ, Zhao C, Li MS. A method for testing real-time system based on timed automata. Ruan Jian Xue Bao/Journal of Software, 2007,18(1):62−73 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/62.htm

[18] Zhou Y, Baresi L, Rossi M. Towards a formal semantics for UML/MARTE state machines based on hierarchical timed automata. Journal of Computer Science and Technology, 2013,28(1):188−202. [doi: 10.1007/s11390-013-1322-8]

[19] Kramer J, Magee J. The evolving philosophers problem: Dynamic change management. IEEE Trans. on Software Engineering, 1990,16(11):1293−1306. [doi: 10.1109/32.60317]

[20] Vandewoude Y, Ebraert P, Berbers Y, D'Hondt T. Tranquility: A low disruptive alternative to quiescence for ensuring safe dynamic updates. IEEE Trans. on Software Engineering, 2007,33(12):856−868. [doi: 10.1109/TSE.2007.70733]

[21] Behrmann G, David A, Larsen KG, Håkansson J, Pettersson P, Wang Y, Hendriks M. UPPAAL 4.0. In: Proc. of the 3rd Int'l Conf. on Quantitative Evaluation of Systems (QEST 2006). IEEE, 2006. 125−126. [doi: 10.1109/QEST.2006.59]

[22] Alpern B, Schneider FB. Recognizing safety and liveness. Distributed Computing, 1987,2(3):117−126. [doi: 10.1007/BF01782772]

[23] Liu Y, Zhang YC, Zhang B, Zhang MW, Zhu ZL. Analysis of service replaceability on behavior effect. Journal of Computer Research and Development, 2010,47(8):1442−1449 (in Chinese with English abstract).

[24] Song W, Tang JH, Zhang GX, Ma XX. WS-BPEL service repalceability anlayisis. Science in China (Information Science), 2012, 42(3):264−279 (in Chinese with English abstract).

[25] Zhang J, Cheng BHC. Model-Based development of dynamically adaptive software. In: Proc. of the 28th Int'l Conf. on Software Engineering. ACM Press, 2006. 371−380. [doi: 10.1145/1134285.1134337]

[26] Hayden CM, Magill S, Hicks M, Foster N, Foster JS. Specifying and verifying the correctness of dynamic software updates. In: Proc. of the Verified Software: Theories, Tools, Experiments. Berlin, Heidelberg: Springer-Verlag, 2012. 278−293. [doi: 10.1007/978-3-642-27705-4_22]

[27] Baresi L, Ghezzi C, Mottola L. On accurate automatic verification of publish-subscribe architectures. In: Proc. of the 29th Int'l Conf. on Software Engineering. 2007. 199−208. [doi: 10.1109/ICSE.2007.57]

[28] Baresi L, Ghezzi C, Mottola L. Loupe: Verifying publish-subscribe architectures with a magnifying lens. IEEE Trans. on Software Engineering, 2011,37(2):228−246. [doi: 10.1109/TSE.2010.39]

[29] Chen HB, Yu J, Hang CQ, Zang BY, Yew PC. Dynamic software updating using a relaxed consistency model. IEEE Trans. on Software Engineering. 2011,27(5):679−694. [doi: 10.1109/TSE.2010.79]

[30] Zhou Y. A runtime architecture-based approach for the dynamic evolution of distributed component-based systems. In: Proc. of the 30th Int'l Conf. on Software Engineering. ACM Press, 2008. 979−982. [doi: 10.1145/1370175.1370217]

[31] Garlan D, Khersonsky S, Kim J. Model checking publish-subscribe systems. In: Proc. of the 10th Int'l SPIN Workshop. LNCS, Springer-Verlag, 2002. 166−180. [doi: 10.1007/3-540-44829-2_11]

[32] Calinescu R, Ghezzi C, Kwiatkowska M, Mirandola R. Self-Adaptive software needs quantitative verification at runtime. Communications of the ACM, 2012,55(9):69−77. [doi: 10.1145/2330667.2330686]

[33] Wang HM, Shi PC, Ding B, Yin G, Shi DX. Online evolution of software services. Chinese Journal of Computers, 2011,34(2): 318−328 (in Chinese with English abstract).

[34] Ma XX, Baresi L, Ghezzi C, Panzica V, Manna L, Lü J. Version-Consistent dynamic reconfiguration of component-based distributed systems. In: Proc. of the 19th ACM SIGSOFT Symp. and the 13th European Conf. on Foundations of Software Engineering. ACM Press, 2011. 245−255. [doi: 10.1145/2025113.2025148]

附中文参考文献:

[1] 吕建,马晓星,陶先平,徐锋,胡昊.网构软件的研究与进展.中国科学(E 辑),2006,36(10):1037−1080.

[6] 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.

[7] 余萍,马晓星,吕建,陶先平.一种面向动态软件体系结构的在线演化方法.软件学报,2006,17(6):1360−1371. http://www.jos.org.cn/1000-9825/17/1360.htm

[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]

© 中国科学院软件研究所 http://www.jos.org.cn

[10] 曾晋,孙海龙,刘旭东,邓婷,怀进鹏.基于服务组合的可信软件动态演化机制.软件学报,2010,21(2):261−276. http://www.jos.org.cn/1000-9825/3735.htm [doi: 10.3724/SP.J.1001.2010.03735]

[16] 李力行,金芝,李戈.基于时间自动机的物联网服务建模和验证.计算机学报,2011,34(8):1365−1377. [doi: 10.3724/SP.J.1016.2011.01365]

[17] 陈伟,薛云志,赵琛,李明树.一种基于时间自动机的实时系统测试方法.软件学报,2007,18(1):62−73. http://www.jos.org.cn/1000-9825/18/62.htm

[23] 刘莹,张一川,张斌,张明卫,朱志良.基于行为效果的服务可替换性分析.计算机研究与发展,2010,47(8):1442−1449.

[24] 宋巍,唐金辉,张功萱,马晓星.WS-BPEL 服务可替换性分析.中国科学(信息科学),2012,42(3):264−279.

[33] 王怀民,史佩昌,丁博,尹刚,史殿习.软件服务的在线演化.计算机学报,2011,34(2):318−328.

周宇(1981－),男,江苏徐州人,博士,副教授,CCF 高级会员,主要研究领域为软件演化,软件验证.

黄志球(1965－),男,博士,教授,博士生导师,CCF 杰出会员,主要研究领域为云计算,软件工程.

黄延凯(1990－),男,硕士生,CCF 学生会员,主要研究领域为软件验证.

吴维刚(1976－),男,博士,副教授,CCF 会员,主要研究领域为云计算,分布式系统.

© 中国科学院软件研究所 http://www.jos.org.cn