软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2011,22(4):659−675 [doi: 10.3724/SP.J.1001.2011.03776]

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

模态顺序图 uMSD 的形式语义*

李雯睿[1,2,3+], 王志坚[1], 张鹏程[1,3]

[1](河海大学 计算机及信息工程学院,江苏 南京 210098)

[2](南京晓庄学院 数学与信息技术学院,江苏 南京 211171)

[3](武汉大学 软件工程国家重点实验室,湖北 武汉 430072)

Formal Semantics of Universal Modal Sequence Diagram

LI Wen-Rui[1,2,3+], WANG Zhi-Jian[1], ZHANG Peng-Cheng[1,3]

[1](College of Computer and Information Engineering, Hohai University, Nanjing 210098, China)

[2](School of Mathematics and Information Technology, Nanjing Xiaozhuang University, Nanjing 211171, China)

[3](State Key Laboratory of Software Engineering, Wuhan University, Wuhan 430072, China)

+ Corresponding author: E-mail: wenrui_li@163.com

Li WR, Wang ZJ, Zhang PC. Formal semantics of universal modal sequence diagram. *Journal of Software*, 2011,22(4):659−675. http://www.jos.org.cn/1000-9825/3776.htm

Abstract: The UML 2.0 Sequence Diagram has been extensively applied in industry. However, the vague semantics of UML 2.0 Sequence Diagram prevent it from being applied effectively. Modal Sequence Diagram is the modal extension of UML 2.0 Sequence Diagram, which distinguishes mandatory scenarios (described by universal MSD, denoted as uMSD) from possible scenarios (described by existential MSD, denoted as eMSD). uMSD is more expressive than eMSD and can represent the temporal properties of concurrent systems. Therefore, the main work of the paper is on uMSD. In order to make uMSD extensively used for formal analysis, verification, and monitoring, the formal semantics of uMSD, based on the Weak Alternating Büchi automaton, are represented, and the transformation algorithms of various operators are given in detail. Next, the expressiveness of uMSD is measured by the well known property specification patterns. Finally, an example is studied, and its future applications are discussed.

Key words: modal sequence diagram; linear weak alternating Büchi automaton; property specification pattern

摘 要: UML 2.0 顺序图已广泛应用于业界,但其语义模糊,以至于不能有效地加以使用.模态顺序图(modal sequence diagram,简称 MSD)是对 UML 2.0 顺序图的模态扩展,区分了强制场景(用 universal MSD 表示,简称 uMSD)和可能场景(用 existential MSD 表示,简称 eMSD).其中,uMSD 具有较强的表达能力,能够用于表示并发系统的时态性质,故主要工作围绕 uMSD 展开.为了使 uMSD 用于形式化分析、验证和监控,给出基于自动机的 uMSD 语义解释,并给出各种操作符的算法,用性质规约模式度量 uMSD 的表达能力.最后进行了实例研究,并讨论了其应用前景.

关键词: 模态顺序图;弱交换 Büchi 自动机;性质规约模式

* 基金项目: 国家高技术研究发展计划(863)(2007AA01Z178); 中央高校基本科研业务费专项资金(2009B04314); 武汉大学软件工程国家重点实验室开放基金(2010-08-01)

收稿时间: 2009-03-25; 定稿时间: 2009-10-23

中图法分类号: TP311 文献标识码: A

顺序图(sequence diagram,简称 SD)是一种基于场景的语言,用来描述系统需求,能够图形化地表示系统实例之间事件交换的时态关系.顺序图包括消息顺序图(message sequence chart,简称 MSC)[1]、活性顺序图(live sequence chart,简称 LSC)[2]、UML 2.0 顺序图[3]以及其他变种.其中,MSC 是由国际电信联盟提倡的,已在业界得到广泛应用,但其有限的表达能力只能描述可能发生的场景,不能描述系统必须满足的场景.于是,有研究者提出用 liveness(活性是指好的事情最终会发生)对 MSC 进行扩展,由此得到 LSC 用 universal 和 existential 两种模态来区分强制场景和可能场景.此后,UML 2.0 顺序图吸收了 MSC 和 LSC 的众多优点,如引入结构化操作符 *par*,*loop*,*alt*,并借鉴 LSC 中 universal 模态的思想,引入新操作符 *assert* 和 *negate* 来表示活性和安全性.

通过比较,这 3 种语言的特点分别是:

(1) MSC 的语义相当弱,只能描述交互样例,也就是可能场景,而不能规定系统必须满足所有运行的场景;LSC 则用 universal LSC 和 existential LSC 来描述强制场景和可能场景;而 UML 2.0 顺序图未明确规定强制场景和可能场景;

(2) MSC 不能区分触发场景的事件和响应事件;在 LSC 中,用构造子 *prechart* 表示触发场景的事件和构造子 *main chart* 表示响应事件,如果 *prechart* 成功执行,则 *main chart* 必须执行;而在 UML 2.0 顺序图中默认的是触发场景的事件,应用 *assert* 操作符的事件表示响应事件;

(3) MSC 未区分强制发生的消息和可能发生的消息;LSC 则用 hot(universal)和 cold(existential)两种模态区分强制消息和可能消息;在 UML 2.0 顺序图中则用 *assert* 操作符表示强制消息,无操作符的消息则默认为可能发生的消息;

(4) MSC 没有区分必须满足的条件和可能满足的条件;而 LSC 用 hot 和 cold 模态分别规定了强制条件和可能条件;UML 2.0 顺序图用 *assert* 操作符规定强制条件;

(5) MSC 不能规定禁止场景;LSC 则用值为 FALSE 的 hot 条件作为封装片段的最大要素来规定一个禁止场景;而 UML 2.0 顺序图是用 *negate* 操作符表示禁止场景;

(6) MSC 缺少构造子来表示场景的范围,不能规定未出现在该场景的事件是否允许任意发生或禁止;LSC 也没有相应的构造子;UML 2.0 顺序图则用 *consider*/*ignore* 操作符表示在一个图中应额外考虑/忽略的事件.

虽然 UML 2.0 顺序图添加了一些新的操作符,如 *assert*,*negate*,*critical* 和 *consider*/*ignore* 等,增强了其表达能力,但其非形式化的语义解释也限制了其应用,且基于有效和无效的迹集操作符语义是不充分的.Harel 等人[4]提出的模态顺序图(modal sequence diagram,简称 MSD)是根据 LSC 的 universal/existential 模态语义扩展 UML2.0 顺序图而来的,重新定义 *assert* 和 *negate* 操作符语义来描述强制场景和禁止场景,以表示活性和安全性,然而却没有给出组合操作符的形式语义,如 *seq*,*par*,*loop*,*alt*,*negate*,*critical* 和 *consider*/*ignore*.MSD 用 existential 和 universal 两种模态区分了可能场景和强制场景.由于 eMSD(existential MSD)是对需求不充分的描述,其表达能力相当弱,只定义了一组交互样例集,适用于系统生命周期的早期.当收集到更多的系统信息时,可将 eMSD 精化为 uMSD(universal MSD).但如何将 eMSD 精化为 uMSD,本文不作深入探讨,本文的研究主要是针对 uMSD 展开的.

为了使 uMSD 能够用于形式化分析、验证和监控,本文第 1 节介绍 MSD 的基本概念.第 2 节给出基于自动机理论的 uMSD 的精确形式语义,用弱交换 Büchi 自动机(weak alternating Büchi automaton,简称 WABA)[5]来解释 uMSD 的基本语义规则,并根据基本规则给出各种操作符的算法.第 3 节采用性质规约模式度量 uMSD 的表达能力.第 4 节给出实例研究以表明 uMSD 的可用性.第 5 节介绍相关工作.第 6 节是全文总结与未来工作展望.

1 MSD 基本概念

MSD 是 UML 2.0 顺序图的模态扩展.一个 MSD 图是由生命线(lifeline)和事件构成的,每条垂线表示构件实例的生命线,连接生命线的箭头表示事件,且事件沿着生命线自顶向下是有序的.事件包括发送消息、接收消息

或状态不变式(stateInvariant).状态不变式即条件规定了运行时一个或多个构件实例的约束,覆盖一条或多条生命线,与 MSC 和 LSC 中的条件构造子类似,条件是构件实例属性的布尔表达式.共享条件的实例需同步执行.

从场景的角度来讲,MSD 区分为描述可能场景的 eMSD 和描述强制场景的 uMSD.

eMSD 规定了系统和环境之间的交互样例,抽象地表示为 eMSD $E=\Diamond(B,\Sigma)$,其中,B 是一个基本图,Σ是 B 中事件标签的超集.其语义简单地解释为,至少有一个系统运行(run)满足该 eMSD 图,可用来对系统测试;或是简单地描述不受限制的场景,此场景描述了系统可能的行为.如图 1 所示,eMSD 的边框是虚线,图的左上角标为 esd,其中,cold 消息用虚线箭头表示.该图的含义是,系统允许 m_1,m_2,m_3 按顺序发生,但系统不强制这些消息发生.

uMSD 规定所有可能系统运行的限制,抽象地表示为 uMSD $U=\Box(C,H,\Sigma)$.其中:C 表示 *cold* 片段,相当于 LSC 中的 *prechart*;而 H 表示 *hot* 片段,相当于 LSC 中的 *main chart*.其语义简单地解释为,一旦满足 *cold* 片段,则 *hot* 片段必须发生.在 uMSD 中,消息分为两类:cold(existential)消息表示可能发生的消息,hot(universal)消息表示必须发生的消息.状态不变式也分为 *hot* 状态不变式,表示条件必须成立;*cold* 状态不变式,表示不强制条件成立.如图 2 所示,uMSD 的边框是实线,图的左上角标为 usd.其中,*cold* 条件是用虚线菱形表示,而 *hot* 消息用实线箭头表示.该图的含义是,如果条件 cond 成立,则必须顺序执行 m_1 和 m_2.

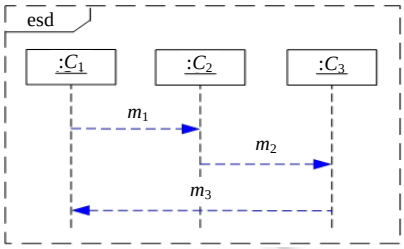


Fig.1 Existential MSD

图 1 eMSD 图

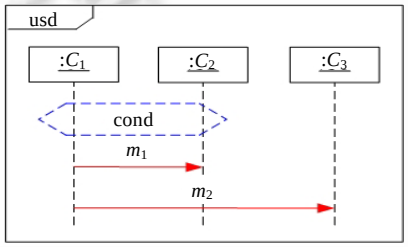


Fig.2 Universal MSD

图 2 uMSD 图

uMSD 图可以是基本交互片段,其中,事件是由 *seq* 操作符连接的,但 *seq* 操作符未显示地表示,也可以是由其他操作符和操作体组成的组合片段,操作符如 *par*,*loop*,*alt*,*consider*/*ignore*,*critical* 和 *negate*.其中,操作体又可能是组合片段.

2 uMSD 的形式语法和语义

本文的主要研究工作是针对 uMSD 展开的,首先给出 uMSD 图的形式语法描述,定义了 uMSD 图的一致性,并提出将单个事件转换为 WABA 的基本规则,再根据基本规则得出各种操作符的算法.

2.1 uMSD的语法

定义(uMSD). 一个 uMSD 图是一个七元组 $U=\langle I,E,\prec,L,e2l,mode,O\rangle$,其中:

- $I=\{i_1,i_2,\ldots,i_k\}$是有限的构件实例集;
- E 是系统中待交换的有限事件标签集,将 E 分为消息标签集 M 和条件标签集 $Cond$,即 $E=M\cup Cond$;
- $\prec$规定事件之间是偏序关系,定义事件序列 $w^{1,p}=\langle e_1e_2\ldots e_p\rangle$,有 $e_1\prec e_2\prec\ldots\prec e_p$;
- L 是构件实例发生事件的位置集;
- $e2l:E\rightarrow L$ 是映射函数将每个事件映射到位置上;
- $mode:E\rightarrow\{cold,hot\}$是将每个事件映射为 *cold* 或 *hot* 事件来表示可能或强制发生的事件.由于每个事件映射到位置上,所以将函数 *mode* 扩展到位置,即 $mode:L\rightarrow\{cold,hot\}$;
- O 是有限的操作符集.针对任意的操作符$\forall op\in O$,有如下形式 $op=\langle type,operand,n\rangle$,其中,*type* 是指操作符的类型,$type\in\{seq,par,loop,alt,consider,ignore,critical,negate\}$.其中,*seq* 操作符无需显示地表示.操作符作用在事件序列 $w^{r,u}(r<u)$上,得到操作体集 $operand=\{w^{r1,u1},w^{r2,u2},\ldots,w^{rn,un}\}(r_k\leqslant u_k,1\leqslant k\leqslant n,u_l+1=r_{l+1},1\leqslant$

$l<n$).其中:$w^{r1,u1}=\langle e_{r1}\ldots e_{u1}\rangle, w^{r2,u2}=\langle e_{r2}\ldots e_{u2}\rangle,\ldots, w^{rn,un}=\langle e_{rn}\ldots e_{un}\rangle; n\in N(n\geq 1)$表示操作体的个数.

2.2 uMSD的一致性

uMSD 的一致性体现在以下两个方面:基本交互片段的一致性和组合片段的一致性.

基本交互片段的一致性是指单个事件和多个连续事件(应用 *seq* 操作符)的一致性,单个事件的一致性是指基本交互片段的第 1 个事件通常为 *cold* 事件,多个连续事件的一致性可先通过定义两个连续事件的一致性来获得.两个连续事件 e_i 和 e_j(令 $j=i+1, 1\leq i<l\leq p$)的一致性是指:

(a) 两个连续 *cold* 事件 e_i 和 e_j 的一致性是指:如果 *cold* 事件 e_i 发生后没有发生 *cold* 事件 e_j,则是 *cold* 违反,安全退出;

(b) 两个连续 *hot* 事件 e_i 和 e_j 的一致性是指:*hot* 事件 e_i 必须发生,接着 *hot* 事件 e_j 也必须发生.即如果 e_i 发生后 e_j 没有发生,而发生了该基本交互片段中其他事件 $M\backslash e_j$,则引发错误;

(c) 两个连续 *cold* 事件 e_i 和 *hot* 事件 e_j 的一致性是指:如果 *cold* 事件 e_i 成功执行,则 *hot* 事件 e_j 必须发生;如果 *cold* 事件 e_i 执行后发生了该基本交互片段中其他事件 $M\backslash e_j$,则表示发生错误.其中,两个连续的事件是指这两个连续事件必须至少与一个构件实例相关联.

因此,两个连续事件的一致性可以推广到多个连续事件的一致性.

为了描述更复杂的场景,需用组合操作符来表示.组合片段的一致性是指操作符的一致性,所以下面定义操作符一致性来保证构建的组合片段是正确的.

1) $alt(w^{r,u})=\langle alt,\{w^{r1,u1},w^{r2,u2},\ldots,w^{rn,un}\},n\geq 2\rangle$.操作符 *alt* 将 $w^{r,u}$ 划分为 n 个操作体,操作体集
$$oprand=\{w^{r1,u1},w^{r2,u2},\ldots,w^{rn,un}\},$$
其中,$alt_1(w^{r1,u1})=alt_1(e_{r1}\ldots e_{u1}),\ldots,alt_n(w^{rn,un})=alt_n(e_{rn}\ldots e_{un}), mode(e_{rk})=cold(k=1,\ldots,n)$表示每个操作体 alt_k 中第 1 个事件 e_{rk} 的模态都为 *cold*,$n\geq 2$ 表示至少有两个操作体;

2) $par(w^{r,u})=\langle par,\{w^{r1,u1},w^{r2,u2},\ldots,w^{rn,un}\},n\geq 2\rangle$.操作符 *par* 将 $w^{r,u}$ 划分为 n 个操作体,操作体集
$$oprand=\{w^{r1,u1},w^{r2,u2},\ldots,w^{rn,un}\},$$
其中,$par_1(w^{r1,u1})=par_1(e_{r1}\ldots e_{u1}),\ldots,par_n(w^{rn,un})=par_n(e_{rn}\ldots e_{un}), mode(par_k(w^{rk,uk}))=cold(k=1,\ldots,n)$表示每个操作体 par_k 中所有事件的模态都为 *cold*,$n\geq 2$ 表示至少有两个操作体;

3) $loop(w^{r,u},h,p)=\langle loop,\{w^{r,u}\},n=1\rangle$.操作符 *loop* 应用到 $w^{r,u}$ 上只有一个操作体.其中,h 和 $p(1\leq h\leq p)$分别表示循环执行操作体次数的下界和上界,且 $mode(e_r)=cold$ 表示 *loop* 操作体中第 1 个事件的模态为 *cold*;

4) $cons(w^{r,u})=\langle cons,\{w^{r1,u1},w^{r2,u2},\ldots,w^{rn,un}\},n\geq 2\rangle$.操作符 *consider* 将 $w^{r,u}$ 划分为 n 个操作体,操作体集 $oprand=\{w^{r1,u1},w^{r2,u2},\ldots,w^{rn,un}\}$.其中,$cons_1(w^{r1,u1})=cons_1(e_{r1}\ldots e_{u1})$是主操作体,其余是 *considered* 操作体集$\{cons_2,\ldots,cons_n\}, cons_2(w^{r2,u2})=cons_2(e_{r2}\ldots e_{u2}),\ldots,cons_n(w^{rn,un})=cons_n(e_{rn}\ldots e_{un}). mode(cons_k(w^{rk,uk}))=hot$ 或者 $mode(cons_k(w^{rk,uk}))=cold(k=2,...,n)$表示每个 *considered* 操作体 $cons_k$ 中事件的模态相同,即为 *hot* 或 *cold*,$n\geq 2$ 表示至少有一个操作体;

5) $critical(w^{r,u})=\langle critical,\{w^{r,u}\},n=1\rangle$.未限制 *critical* 操作体中事件序列 $w^{r,u}$ 的模态,$n=1$ 表示只有一个操作体;

6) $negate(w^{r,u})=\langle negate,\{w^{r,u}\},n=1\rangle$.操作符 *negate* 应用到 $w^{r,u}$ 上,$n=1$ 表示只有一个操作体,$mode(negate(w^{r,u}))=cold$ 表示 *negate* 操作体中事件序列的模态都为 *cold*,不允许 *negate* 操作符自嵌套,也不允许其他操作符嵌套 *negate* 操作符.

定义(良构的 uMSD). 如果一个 uMSD 图满足基本交互片段的一致性和组合片段的一致性,则称该 uMSD 图是良构的.

2.3 基于WABA的uMSD语义

为了使得 uMSD 能够用于形式化分析、验证和监控,通常可以用 Büchi 自动机来表示语义.本文采用 WABA

而非直接采用 Büchi 自动机(Büchi automaton,简称 BA).首先,因为 WABA 比 BA 更简洁,WABA 具有 existential 和 universal 结构,适用于规约程序.WABA 的状态空间被划分为偏序集合,可以从某个状态集合迁移到更小的状态集合,将 uMSD 转换为 WABA 比将 uMSD 转换为 BA 要容易;其次,WABA 能够在 existential 和 universal 结构间进行交换,这样求得 WABA 的补非常容易,因此可以提出更为简洁和有效的验证算法.

在给出基于 WABA 的 uMSD 语义之前,首先定义 WABA.对于一个给定集合 Q,$B^+(Q)$是用∧或∨对 Q 中元素构建的正布尔公式集,布尔公式的值为 TRUE 或 FALSE.对于 $Y\subseteq Q$,当且仅当将 TRUE 赋给 Y 中的元素,则 Y 满足公式$\theta\in B^+(Q)$,并将 FALSE 赋给 $Q\backslash Y$ 中的元素.

定义(弱交换 Büchi 自动机). 一个弱交换 Büchi 自动机是一个五元组 $A=\langle\Sigma,Q,q_0,\delta,\rho\rangle$,

- Σ是一个有限的字母表;
- Q 是一个有限的状态集;
- q_0 是初始状态;
- $\delta:Q\to B^+(Q\cup\Sigma)$是转换函数,每个状态 $q\in Q$ 关联一个命题公式$\delta(q)$;
- ρ是接受条件;
- 将 Q 划分成 k 个不相交集合($k\in\mathbb{N}$),对于所有 $i,j\in\{1,\ldots,k\}$,$q\in Q_i$,$q'\in Q_j$,$q'\in\delta(q)$,有 $Q_i\leqslant Q_j$,即状态集之间存在偏序关系.也就是说,状态集经转换迁移到自身或是一个更小的状态集,且对于 $1\leqslant i\leqslant k$,$Q_i\subseteq\rho$或 $Q_i\cap\rho=\varnothing$.

为了解释 uMSD 的语义,需要构建 uMSD 的自动机.首先考虑如何将单个消息或条件转换成 WABA,下一节将讨论如何将操作符应用到交互片段中,从而得到操作符算法,同时并不会改变构建自动机的本质.

定义(uMSD 的自动机). 给定一个 uMSD U,可以构建一个 WABA $A_U=\langle\Sigma,Q,s_0,T,\rho\rangle$,

- Σ是系统待交换消息和条件的字母表;
- $Q=S\cup\{s_{rej},s_{acc}\}$,其中,$S$ 是系统所有构件实例的当前位置对应的状态集,s_{rej} 表示拒绝状态,s_{acc} 表示接受状态;
- s_0 是初始状态;
- $T:Q\to B^+(Q\cup\Sigma)$是转换函数,$t=\langle s,\Gamma,s'\rangle\in\delta(s\in Q,\Gamma\subseteq\Sigma,s'\in Q)$表示从状态 s 迁移到 s'的转换标签为Γ;
- $\rho=\{s|mode(s)=cold\}\cup\{s_{acc}\}$是接受条件.

定义(运行). 在无限字 $w=w_0w_1w_2\ldots(w_i\in\Sigma$是 w 的第 i 个字)上,WABA A_U 的一个运行σ定义为一个有向无环图 DAG $\sigma=(V,v_0,\to)$:

- $V\subseteq Q\times\mathbb{N}$是一个有限的顶点集,顶点$(s,i)$表示 A_U 读入字 w_i 后的状态 s;
- 初始顶点 $v_0=(s_0,0)\in V$;
- $\to\subseteq V\times V$,① 如果$(s,i)\to(s',i')$,则 $i'=i+1$;② 对于每个$(s,i)\in V$,集合$\{s'|(s,i)\to(s',i+1)\}$满足$\delta(s,w_i)$.如果有$(s,i)\to(s',i+1)$,则$(s',i+1)$是(s,i)的后继,并且当$(s,i)\to^*(s',i+1)$时,从(s,i)到(s',i')是可达的.

在运行$\sigma=(V,v_0,\to)$上,路径$\pi=v_0v_1v_2\ldots$是一个无限的顶点序列,对于所有的 $i\geqslant 0$,有$(v_i,v_{i+1})\in E$.$Inf(\pi)$是由路径π中出现无限次的状态组成.如果$\rho\cap Inf(\pi)\neq\varnothing$,则路径$\pi$是可接受的.如果运行$\sigma$的每条路径是可接受的,则运行$\sigma$是可接受的.如果在字 $w\in\Sigma^\omega$上有一个可接受的运行σ,则 WABA A_U 接受字 w.uMSD U 的语言定义为其构建 WABA A_U 所接受的迹(trace)语言.

定义(uMSD 的语言). uMSD U 的迹语言可定义为

$$L(A_U)=\{w\in\Sigma^\omega|w|=A_U\}.$$

2.3.1 uMSD 基本规则

基本规则讨论了如何将一个 uMSD U 中单个消息或条件转换成 WABA,如图 3 所示.消息是以同步或异步形式来通信的,本文考虑同步通信方式,将发送和接收消息看作一条消息.首先解释图 3 中转换标签的含义,Σ是系统待交换消息和条件的字母表,M 是 U 中的消息集,C 是在 U 中的条件集,$\Sigma\backslash(M\cup C)$表示发生了不在 uMSD U 中的消息或条件;$(M\cup C)\backslash m$ 表示发生了 uMSD U 中除了消息 m 之外的消息或条件;not *cond* 表示条件 *cond* 不成

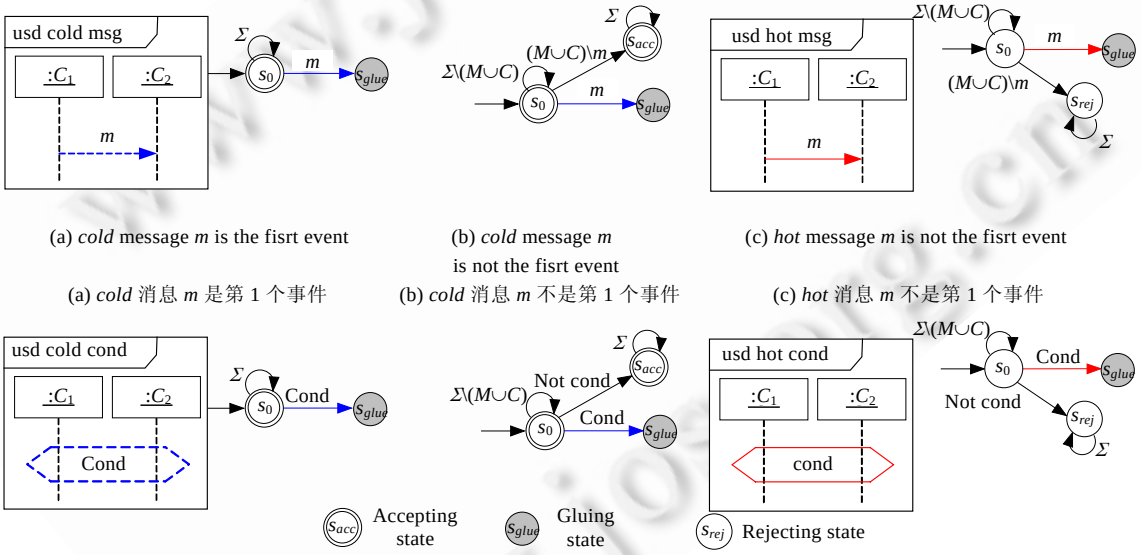
立.另外,粘合状态 $s_{glue}\in G(G\subseteq Q)$用来与下一个 WABA 的初始状态进行粘合而生成更为复杂的 WABA.

1) *cold* 消息转换成 WABA.分以下两种情况:(a) *cold* 消息 *m* 是 uMSD *U* 中的第 1 个事件:因为 *cold* 消息 *m* 可能发生,所以初始状态 s_0 为接受状态(双圈),s_0 上自转换标签Σ表示等待 *m* 发生,即系统中任意消息或条件可在 *m* 之前发生或成立,*m* 不一定是第 1 次出现;如果 *m* 发生,则从 s_0 迁移到粘合状态 s_{glue},如图 3(a)所示;(b) *cold* 消息 *m* 不是 uMSD *U* 中的第 1 个事件:状态 s_0 上自转换标签$\Sigma\backslash(M\cup C)$表示等待 *m* 发生.同样,如果 *m* 发生,则从 s_0 迁移到 s_{glue};如果发生了除 *m* 外的其他事件$(M\cup C)\backslash m$,则是 *cold* 违反,即从 s_0 迁移到接受状态 s_{acc},s_{acc} 上自转换标签Σ表示对于任意输入仍停留在 s_{acc},如图 3(b)所示.

2) *hot* 消息转换成 WABA.在 uMSD *U* 中,一般是 *cold* 消息或 *cold* 条件作为触发,引起 *hot* 消息的执行,*hot* 消息通常不作为 uMSD *U* 中第 1 个事件出现,所以 *hot* 消息转换成 WABA 只有一种情形,如图 3(c)所示.因为强制 *m* 发生,状态 s_0(单圈)上自转换标签$\Sigma\backslash(M\cup C)$表示等待 *m* 必须发生,如果 *m* 发生,则从 s_0 迁移到粘合状态 s_{glue};如果 *m* 没有发生,即发生了除 *m* 外的其他事件$(M\cup C)\backslash m$,则是 *hot* 违反,即从 s_0 迁移到拒绝状态 s_{rej},s_{rej} 上自转换标签Σ表示对于任意输入仍停留在 s_{rej}.

3) *cold* 条件转换成 WABA.与 *cold* 消息类似,也分为两种情况:(a) *cold* 条件是 uMSD *U* 中第 1 个事件,因为 *cold* 条件表示不必强制该条件成立,所以初始状态 s_0 为接受状态,s_0 上自转换标签Σ表示接受任意输入,等待条件值为 TRUE.但条件值为 TRUE 不一定是第 1 次成立,当条件值为 TRUE 时,则从 s_0 迁移到粘合状态 s_{glue},如图 3(d)所示;(b) *cold* 条件不是 uMSD *U* 中第 1 个事件:状态 s_0 上自转换标签$\Sigma\backslash(M\cup C)$表示等待 *cold* 条件成立.同样,如果条件成立,则从 s_0 迁移到 s_{glue};如果条件值为 FALSE,则从 s_0 迁移到接受状态 s_{acc},如图 3(e)所示.

4) *hot* 条件转换成 WABA.与 *hot* 消息类似,*hot* 条件转换成 WABA 也只有一种情形,如图 3(f)所示.因为 *hot* 条件表示该条件必须成立,状态 s_0 上自转换标签$\Sigma\backslash(M\cup C)$表示等待 *hot* 条件值为 TRUE.如果 *hot* 条件成立,则从 s_0 迁移到粘合状态 s_{glue};如果条件不成立,表示 *hot* 条件违反,则从 s_0 迁移到拒绝状态 s_{rej}.



(a) *cold* message *m* is the fisrt event (b) *cold* message *m* is not the fisrt event (c) *hot* message *m* is not the fisrt event

(a) *cold* 消息 *m* 是第 1 个事件 (b) *cold* 消息 *m* 不是第 1 个事件 (c) *hot* 消息 *m* 不是第 1 个事件

(d) *cold* cond is the fisrt even (e) *cold* cond is not the fisrt event (f) *hot* cond is not the fisrt event

(d) *cold* cond 是第 1 个事件 (e) *cold* cond 不是第 1 个事件 (f) *hot* cond 不是第 1 个事件

Fig.3 Basic rules

图 3 基本规则

2.3.2 uMSD 操作符算法

基本交互片段只能描述简单的场景,而复杂的场景需要用操作符将基本交互片段构成组合片段.组合操作

符包括 *seq*,*alt*,*par*,*loop*,*consider*/*ignore*,*negate* 和 *critical*,增强了 uMSD 的表达能力.本节给出以上操作符转换成 WABA 的算法.

首先来看 *merge* 算法.如果将两个 WABAs A_1 和 A_2 顺序组合,该算法添加了从 A_1 的粘合状态 $s_{glue}\in G_1$ 到 A_2 的初始状态 s_0^2 的 ε转换,A_2 中所有状态 s 到接受状态 s_{acc}^2 或拒绝状态 s_{rej}^2 的转换都替换为 A_2 中所有状态 s 到 A_1 的接受状态 s_{acc}^1 或拒绝状态 s_{rej}^1 的转换.*compose* 算法是求多个 WABAs 的并.*compose* 算法用 ε标签将新的初始状态 s_0,新的接受状态 s_{acc} 和新的拒绝状态分别与 n 个 WABAs 的初始状态 s_0^i、接受状态 s_{acc}^i 和拒绝状态 s_{acc}^i 连接.

算法 1. *merge*(A_1,A_2).

输入:WABAs $A_1=\langle \Sigma_1,Q_1,s_0^1,T_1,\rho_1\rangle$ 和 $A_2\langle \Sigma_2,Q_2,s_0^2,T_2,\rho_2\rangle$;

输出:顺序组合 A_1 和 A_2 得到的 WABA A'.

1. **begin**
2. $\Sigma'=\Sigma_1\cup\Sigma_2$;
3. $Q'=Q_1\cup Q_2\setminus\{s_{rej}^2,s_{acc}^2\}$;
4. $\rho'=(\rho_1\cup\rho_2)\setminus\{s_{acc}^2\}$;
5. **for each** $s_{glue}\in G_1$ **do**
6. $T'=T'\cup\{(s_{glue},\varepsilon,s_0^2)\}$;
7. **end for**
8. **for each** $t=\langle s,\Gamma,s'\rangle\in T'$ **do**
9. **if** $s'=s_{acc}^2$ **then**
10. $t'=\langle s,\Gamma,s_{acc}^1\rangle$;
11. $T'=(T'\setminus\{t\})\cup\{t'\}$;
12. **else if** $s'=s_{rej}^2$ **then**
13. $t'=\langle s,\Gamma,s_{rej}^1\rangle$;
14. $T'=(T'\setminus\{t\})\cup\{t'\}$;
15. **end if**
16. **end if**
17. **end for**
18. $A'=\langle \Sigma',Q',s_0^1,T',\rho'\rangle$;
19. **return** (A');
20. **end**.

算法 2. *compose*(A_1,…,A_n).

输入:n 个 WABA $A_1=\langle \Sigma_1,Q_1,s_0^1,T_1,\rho_1\rangle$,…,$A_n=\langle \Sigma_n,Q_n,s_0^n,T_n,\rho_n\rangle$;

输出:n 个 WABAs 组合而成一个 WABA $A'=\langle \Sigma',Q',s_0,T',\rho'\rangle$.

1. **begin**
2. $\Sigma':=\bigcup_{i=1}^n\Sigma_i$;
3. $Q':=(\bigcup_{i=1}^n Q_i)\cup\{s_0,s_{acc},s_{rej}\}$;
4. **for each** i:=1 **to** n **do**
5. $\rho':=\bigcup_{i=1}^n\rho_i\cup\{s_{acc}\}$;
6. **end for**
7. $T'=\varnothing$;

8. **for each** i:=1 **to** n **do**
9. T':=$T'\cup T_i$;
10. **end for**
11. **for each** i:=1 **to** n **do**
12. $T'=T'\cup\{(s_0,\varepsilon,s_0^i)\}$;
13. **end for**
14. **for each** i:=1 **to** n **do**
15. $T'=T'\cup\{(s_{acc}^i,\varepsilon,s_{acc})\}$;
16. **end for**
17. **for each** i:=1 to n **do**
18. $T'=T'\cup\{(s_{rej}^i,\varepsilon,s_{rej})\}$;
19. **end for**
20. A'=$\langle\Sigma',Q',s_0,T',\rho'\rangle$;
21. **return** (A');
22. **end**.

(1) *seq* 操作符

从 r 到 u 个事件 $w^{r,u}$ 顺序组合,首先将每个事件作为 *BasicRules*()的输入,通过基本规则生成相应的 WABA,然后用 *merge* 算法将从 r 到 u 个事件相应的 WABAs 两两顺序组合.

算法 3. *seq*($w^{r,u}$).

输入:事件序列 $w^{r,u}$=$\langle e_r e_{r+1}\ldots e_u\rangle$;

输出:从 r 到 u 个事件顺序组合的 WABA A'.

1. **begin**
2. **for each** j:=r **to** u **do**
3. A_j :=*BasicRules*(e_j);
4. **end for**
5. A'=A_r;
6. **for each** k:=r+1 **to** u **do**
7. A':=*merge*(A',A_k);
8. **end for**
9. **return** (A');
10. **end**.

(2) *alt* 操作符

alt 操作符的迹集定义为所有操作体迹的并.*alt* 操作符应用到 $w^{r,u}$ 上分成 n 个操作体 alt_1,alt_2,…,alt_n,对每个操作体应用 *seq* 算法生成相应的 WABA,然后用 *compose* 算法将这 n 个操作体的 WABAs 组合成一个 WABA A'.

算法 4. *alt*($w^{r,u}$).

输入:n 个操作体 $alt_1(w^{r1,u1})$,$alt_2(w^{r2,u2})$,…,$alt_n(w^{rn,un})$;

输出:*alt* 操作符的 n 个操作体组合而成的 WABA A'.

1. **begin**
2. **for each** k:=1 **to** n **do**
3. A_k:=*seq*(alt_k);
4. **end for**
5. A':=*compose*(A_1,…,A_n);

6. **return** (A');
7. **end**.

(3) *par* 操作符

par 操作符应用到事件 $w^{r,u}$ 上划分成 n 个操作体 $par_1(w^{r1,u1}),par_2(w^{r2,u2}),\ldots,par_n(w^{rn,un})$,在保证每个操作体中事件顺序不变的前提下,每个操作体中事件与其他操作体中事件以任意顺序交错执行,得到含有 h 个元素的 *par* 集$\{par_1(w^{a1,b1}),par_2(w^{a2,b2}),\ldots,par_h(w^{ah,bh})\}$.对于每个并行组合后的 $par_i(1\leqslant i\leqslant h)$,应用 *seq* 算法生成相应的 WABA,然后用 *compose* 算法将这些 WABAs 组合成一个 WABA A'.

算法 5. $par(w^{r,u})$.

输入:*par* 集$\{par_1(w^{a1,b1}),par_2(w^{a2,b2}),\ldots,par_h(w^{ah,bh})\}$;

输出:*par* 操作符的 n 个操作体组合而成相应的 WABA A'.

1. **begin**
2. **for each** i:=1 **to** h **do**
3. A_i:=*seq*(par_i);
4. **end for**
5. A':=*compose*($A_1,\ldots,A_h$);
6. **return** (A');
7. **end**.

(4) *loop* 操作符

loop 操作符应用到事件 $w^{r,u}$ 上,规定 *loop* 循环的下界为 h,上界为 $p(1\leqslant h\leqslant p)$,其含义是反复执行操作体至少 h 次,至多 p 次,用 *loop**表示无限次循环.*loop* 集是由 $loop_1=(w^{r,u})^h,loop_2=(w^{r,u})^{h+1},\ldots,loop_{p-h+1}=(w^{r,u})^p$ 组成,其中,$(w^{r,u})^l$ 表示自身连接 $l(h\leqslant l\leqslant p)$次.转换成相应的自动机,然后组合成一个 WABA A'.

算法 6. $loop(w^{r,u})$.

输入:事件序列 $w^{r,u}$;

输出:*loop* 循环相应的 WABA A'.

1. **begin**
2. **for each** l:=h **to** p **do**
3. A_l:=*seq*($(w^{r,u})^l$);
4. **end for**
5. A':=*compose*($A_h,\ldots,A_p$);
6. **return** (A');
7. **end**.

(5) *critical* 操作符

critical 操作符用来规定 *critical* 操作体中事件执行的原子性,只要 *critical* 操作体中第 1 个事件发生,则期望其余事件也按顺序紧接着发生,不期望 *critical* 操作体中事件之间发生其他事件,从而保证该操作体执行的原子性.*critical* 操作符的语义是指该操作体的迹不能被其他事件交错执行.例如,当封装组合片段中含有 *par* 操作符时,可根据需求在 *par* 操作体中嵌套 *critical* 操作符来阻止事件之间交错执行.先用 *seq* 算法求得事件序列 $w^{r,u}$ 的 WABA,然后对该 WABA 的转换标签进行修改.除了 *critical* 操作体中第 1 个事件的状态自转换标签为Σ以外,其余事件的相应状态上原自转换标签$\Sigma\backslash(M\cup C)$均去除.当第 1 个事件发生后,如果其余事件 $e_i(i=r+1,\ldots,u)$没有发生,则需根据事件的模态修改其余事件的相应状态到接受状态或拒绝状态的转换标签.当事件 e_i 的模态为 *cold* 时,转换 $t=\langle s_i,M\backslash e_i,s_{acc}\rangle$ 修改为 $t'=\langle s_i,\Sigma\backslash e_i,s_{acc}\rangle$,或者当事件 e_i 的模态为 *hot* 时,转换 $t=\langle s_i,M\backslash e_i,s_{rej}\rangle$ 修改为 $t'=\langle s_i,\Sigma\backslash e_i,s_{rej}\rangle$.

算法 7. *critical* ($w^{r,u}$).

输入:事件序列 $w^{r,u}=\langle e_r\ldots e_u\rangle$;

输出:*critical* 操作符对应的 WABA $A'=\langle \Sigma,Q,s_0,T',\rho\rangle$.

1. **begin**
2. A:=*seq*($w^{r,u}$);
3. $A=\langle \Sigma,Q,s_0,T,\rho\rangle$;
4. $T'=T$;
5. **for each** i:=r+1 **to** u **do**
6. $T'=T'\backslash\{t=\langle s_i,\Sigma\backslash(M\cup C),s_i\rangle\}$;
7. **end for**
8. **for each** i:=r+1 **to** u **do**
9. **if** *mode*(e_i)=*cold* and $t=\langle s_i,M\backslash e_i,s_{acc}\rangle$ **then**
10. $T'=T'\backslash\{t\}$;
11. $T'=T'\cup\{t'=\langle s_i,\Sigma\backslash e_i,s_{acc}\rangle\}$;
12. **else if** *mode*(e_i)=*hot* and $t=\langle s_i,M\backslash e_i,s_{re}\rangle$ **then**
13. $T'=T'\backslash\{t\}$;
14. $T'=T'\cup\{t'=\langle s_i,\Sigma\backslash e_i,s_{rej}\rangle\}$;
15. **end if**
16. **end if**
17. **end for**
18. $A'=\langle \Sigma,Q,s_0,T',\rho\rangle$;
19. **end**.

(6) *negate* 操作符

在 uMSD 中,例举出所有正确的场景通常是非常困难的,所以,用 *negate* 操作符来规定不期望发生的场景. *negate* 操作体隐含了值为 FALSE 的 *hot* 条件作为该操作体的最大要素,即不期望的事件发生后不会再有其他事件发生,所以不允许 *negate* 操作符自嵌套,也不允许其他操作符嵌套 *negate* 操作符.

算法 8. *negate*($w^{r,u}$).

输入:事件序列 $w^{r,u}=\langle e_r\ldots e_u\rangle$;

输出:*negate* 操作符对应的 WABA A'.

1. **begin**
2. **for** i:=r **to** u **do**
3. A_i:=*BasicRules*(e_i);
4. **end for**
5. A_{u+1}:=*basicRules*(*hot cond*=FALSE);
6. $A'=A_r$;
7. **for each** k:=r+1 to u+1 **do**
8. A':=*merge*(A',A_k);
9. **end for**
10. **return** (A');
11. **end**.

(7) *consider*/*ignore* 操作符

在 LSC 中,一个交互片段显示地规定了该片段中事件的顺序,而未出现在该片段中的事件与该片段的迹语言是无关的.而 uMSD 考虑了未出现在交互片段中的事件,这样的事件被认为是违反了已规定事件的偏序关系,

从而影响到该交互片段的迹语言.*consider*/*ignore* 操作符是用来规定在交互片段中应考虑/忽略的事件.*consider* 操作符应用到 $w^{r,u}$ 分为主操作体 $cons_1(w^{r1,u1})$和 *considered* 操作体集$\{cons_2(w^{r2,u2}),\ldots,cons_n(w^{rn,un})\}$,*considered* 操作体集作为额外需考虑的操作体出现在主操作体之后,规定每个 *considered* 操作体的模态是 *hot* 或 *cold*.每个 *considered* 操作体与主操作体中事件交错执行,至多选择一个分支执行,会影响封装组合片段的迹语言:若选择一个模态为 *cold* 的 *considered* 操作体执行,则发生 *cold* 违反,使得该迹包含在封装组合片段的迹语言中;若选择一个模态为 *hot* 的 *considered* 操作体执行,则发生 *hot* 违反,使得该迹不能包含在封装组合片段的迹语言中.另外,将 *considered* 操作体集中所有事件 E 从主操作体中事件相应状态上的自转换标签$\Sigma\backslash(M\cup C)$中移除.而对于 *ignore* 操作符,则是将 *ignored* 操作体集中所有事件 E 添加到主操作体中事件相应状态上的自转换标签$\Sigma\backslash(M\cup C)$中.由于篇幅所限,*ignore* 操作符的算法可类似地给出,这里不再赘述.

算法 9. *consider*($w^{r,u}$).

输入:主操作体 $cons_1(w^{r1,u1})$和 *considered* 操作体集$\{cons_2(w^{r2,u2}),\ldots,cons_n(w^{rn,un})\}$;

输出:应用 *consider* 操作符生成相应的 WABA A'.

```
1.  begin
2.    A:=seq(cons1(w^{r1,u1}));
3.    A=⟨Σ,Q,s0,T,ρ⟩;
4.    for each k:=2 to n do
5.      for each j:=r1+1 to u1 do
6.        if t=⟨sj,Σ\(M∪C),sj⟩ then
7.          T′=T′\{t};
8.          T′=T′∪{t′=⟨sj,Σ\(M∪C∪E),sj⟩};
9.        end if
10.       if mode(ej)=cold and mode(consk(w^{rk,uk}))=cold and t=⟨sj,M\ej,sacc⟩ then
11.         T′=T′\{t};
12.         T′=T′∪{t′=⟨sj,(M\ej)∪Ek,sacc⟩};
13.       else if mode(ej)=hot and mode(consk(w^{rk,uk}))=hot and t=⟨sj,M\ej,srej⟩ then
14.           T′=T′\{t};
15.           T′=T′∪{t′=⟨si,(M\ej)∪Ek,srej⟩};
16.         end if
17.       end if
18.     end for
19.   end for
20.   A′:=compose(A2,…,An)
21.   return (A′);
22. end.
```

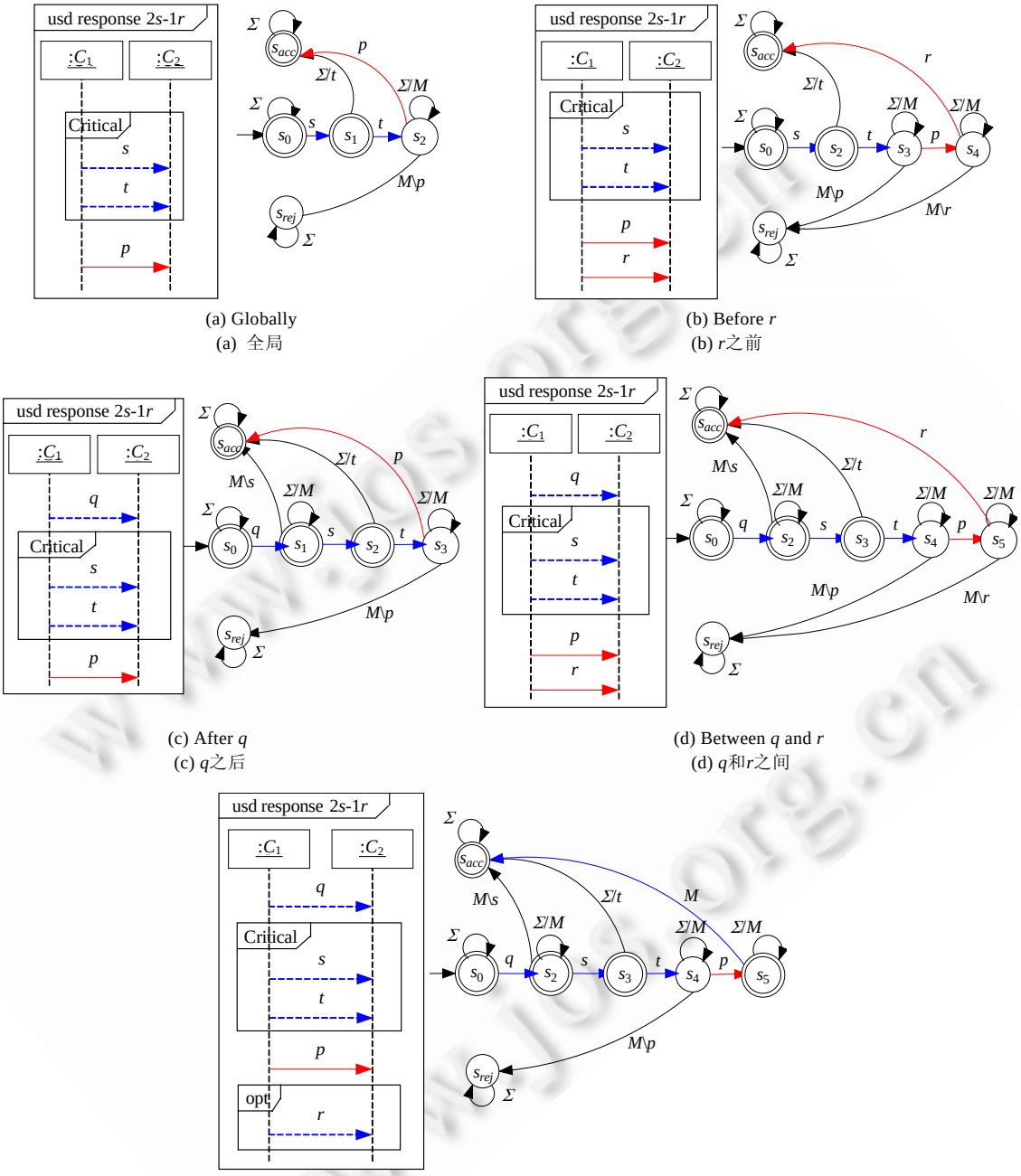
3 uMSD 表达能力

uMSD 刻画了系统中事件之间的时态关系,能够表示系统的时态性质,可用于验证或监控等,如指定 *hot* 事件表示活性,或者用 *negate* 操作符表示禁止场景,即安全性.为了有效地使用 uMSD,其表达能力需进一步加以评估,这里使用 Dwyer 等人[6]提出的性质规约模式(property specification patterns,简称 PSPs)来度量 uMSD 的表达能力,并提供 uMSD 模板表示性质规约模式.

PSPs 用来帮助用户表示系统需求,容易表示如“事件 *p* 发生在事件 *r* 之前”或“*s* 响应于 *p*,并且发生在事件 *q* 和 *r* 之间”,能够重用性质规约模式.PSP 分为 Occurrence 模式和 Order 模式.Occurrence 模式是指在系统执行过程中发生一个给定的事件,包括 Absence,Existence,Bounded Existence 和 Universality;Order 模式规定在系统执行过程中多个事件按某种顺序发生,包括 Precedence,Precedence Chain,Response 和 Response Chain.每个模式对应

于 5 种范围 Globally,Before r,After q,Between q and r 和 After q until r,即模式必须成立时程序执行的范围.

我们可以用 uMSD 表示 PSP 中的性质规约模式.下面以响应链 2s-1r(2 stimulus-1 response chain)模式为例来说明 uMSD 的表达能力.响应链 2s-1r 模式是指,在给定范围内,响应事件 p 必须在 stimulus 事件 s,t 之后发生.响应链 2s-1r 模式的 uMSD 模板和相应的 WABA,分别对应 5 种范围,如图 4 所示.



(a) Globally
(a) 全局

(b) Before r
(b) r之前

(c) After q
(c) q之后

(d) Between q and r
(d) q和r之间

(e) After q until r
(e) q 之后直到 r

Fig.4 uMSD templates for property specification patterns

图 4 性质规约模式的 uMSD 模板

stimulus 事件 *s*,*t* 的模态为 *cold*,响应事件 *p* 的模态为 *hot*.也就是说,当 *cold* 事件序列 *s*,*t* 发生后,*hot* 事件 *p* 必须发生.其中,*critical* 操作符用来封装事件序列 *s*,*t*,以保证该序列的原子性,表示 *s*,*t* 不一定发生,但如果 *s* 发生,则期望 *t* 紧接着发生,*s* 和 *t* 之间不允许发生其他事件.

响应链 2*s*-1*r* 模式的 Globally 范围表示响应链 2*s*-1*r* 在全局成立,其 uMSD 模板和相应的 WABA 如图 4(a)所示.基于 WABA 的语义解释是,初始状态 s_0 上自转换标签 Σ 表示等待 *cold* 事件 *s* 发生,为了保证事件序列 *s*,*t* 的原子性,*s* 发生后期望发生 *t*,如果发生了其他事件 $\Sigma\backslash t$,是一个 *cold* 违反,则从 s_1 迁移到接受状态 s_{acc};如果 *cold* 事件序列 *s*,*t* 发生,*hot* 事件 *p* 也发生,则从 s_2 迁移到接受状态 s_{acc},否则,从 s_2 迁移到拒绝状态 s_{rej}.

响应链 2*s*-1*r* 模式的 Before *r* 范围表示响应链 2*s*-1*r* 在事件 *r* 之前执行,其 uMSD 模板和相应的 WABA 如图 4(b)所示.基于 WABA 的语义解释是,初始状态 s_0 等待 *cold* 事件 *s* 发生,如果 *cold* 事件序列 *s*,*t* 发生,且 *hot* 事件 *p* 也发生,这时期望 *hot* 事件 *r* 发生以迁移到接受状态 s_{acc},否则,迁移到拒绝状态 s_{rej}.

响应链 2*s*-1*r* 模式的 After *q* 范围表示响应链 2*s*-1*r* 在事件 *q* 之后执行,其 uMSD 模板和相应的 WABA 如图 4(c)所示.基于 WABA 的语义解释是,初始状态 s_0 等待 *cold* 事件 *q* 发生,如果 *cold* 事件 *q* 发生,且 *cold* 事件序列 *s*,*t* 和 *hot* 事件 *p* 依次发生,则迁移到接受状态 s_{acc}.

响应链 2*s*-1*r* 模式的 Between *q* and *r* 范围表示响应链 2*s*-1*r* 在事件 *q* 和 *r* 之间执行,其 uMSD 模板和相应的 WABA 如图 4(d)所示.基于 WABA 的语义解释是,初始状态 s_0 等待 *cold* 事件 *q* 发生,如果 *cold* 事件 *q* 发生,期望事件序列 *s*,*t* 和 *hot* 事件 *p* 依次发生.这时,*hot* 事件 *r* 必须发生,则迁移到接受状态 s_{acc},否则,迁移到拒绝状态 s_{rej}.

响应链 2*s*-1*r* 模式的 After *q* until *r* 范围表示响应链 2*s*-1*r* 在事件 *q* 和 *r* 之间执行,与 Between *q* and *r* 范围不同的是,*r* 不一定发生.其 uMSD 模板和相应的 WABA 如图 4(e)所示.用 *opt* 操作符表示 *r* 不一定发生,相应的转换标签 *M* 表示无论 *r* 是否发生都迁移到接受状态 s_{acc}.

4 实例分析

本节我们以一个工作助理(on-the-job assistant,简称 OJA)为实例来评估 uMSD 的可用性.OJA 是一个 Web 服务组合,用以提供给专业人员各种视频帮助.它提供以下 4 项服务:(1) 知识库(knowledge base,简称 KB)存储各种专业的视频信息,并提供查询结果;(2) 虚拟助理(virtual assistant,简称 VA)向 KB 提交用户的查询请求,是用户和 KB 的中介;(3) 银行(bank)负责验证用户的信用卡帐户是否有效,并从用户的信用卡扣除观看视频的费用;(4) 其他服务提供者(other providers)是指,当 KB 中没有满足用户需求的视频信息时,VA 则向其他服务提供者提交用户的查询请求.用户通过手机上网请求视频帮助,并提出需求:高质量的视频流(即在线实时观看视频,而不是将视频完全下载后再观看)和中等质量的音频,预定支付金额不超过 20 元,且信用卡支付.

使用 uMSD 来描述该系统的需求,并将 uMSD 描述的场景转换成 WABA,可用于分析、验证和监控组合服务行为的正确性,这是后续工作,本文关注的是如何将 uMSD 转换成 WABA.下面考虑了 OJA 的两个性质:

性质 1(P1). 如果 KB 中没有满足用户需求的视频,则 VA 向其他服务提供者发出查询请求,并且有一些提供者响应该请求.收集所有的响应并周期性地将视频列表(对服务相关信息的文本描述)提交给用户,用户浏览视频列表,从中选择一个满足其需求的服务提供者.用户与服务提供者交互,用户获取视频并观看.当观看视频结束后,用户向 KB 发送该视频的反馈信息,最后,服务提供者将该视频的相关信息注册在 KB 中.

性质 1 的 uMSD 图和相应的 WABA 如图 5(a)和图 5(b)所示,每个状态上的自转换标签 $\Sigma\backslash M$ 未显示地标出.

性质 2(P2). 当用户观看视频时,用户不满意当前的视频质量,可取消当前视频,或当用户观看视频时,KB 应是空闲的.描述该性质使用了 *consider* 操作符,规定了在主操作体执行过程中,若在 User 和 VA 之间发生了消息 *cancel*()是一个 *cold* 违反,则封装交互片段接受该迹,如取消用户向 KB 请求获取视频 *getVideo*();规定了在主操作体执行过程中,KB 必须是空闲的.若 KB 是繁忙的,则是一个 *hot* 违反,导致最终的迹不包含在该交互片段的迹语言中.注意,将在 *considered* 操作体集中列出的事件 *E* 从主操作体中所有事件的自转换标签 $\Sigma\backslash(M\cup C)$ 中移除.

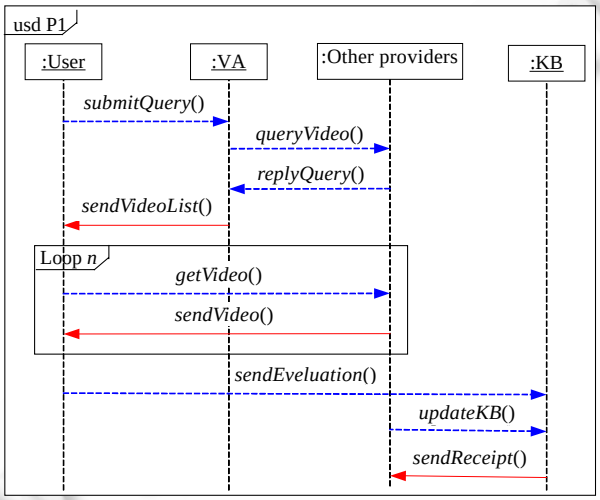
性质 2 的 uMSD 图和相应的 WABA 如图 6(a)和图 6(b)所示.

表 1 从各个性质的类型、构件实例个数、事件个数、生成的 WABA 的状态个数和转换个数作为参数,比较了两个性质及其相应的 WABA.比如,对于性质 2,需要 3 个参与者、10 个事件、生成的 WABA 有 10 个状态和 25 个转换.

Table 1 Properties and the corresponding WABA

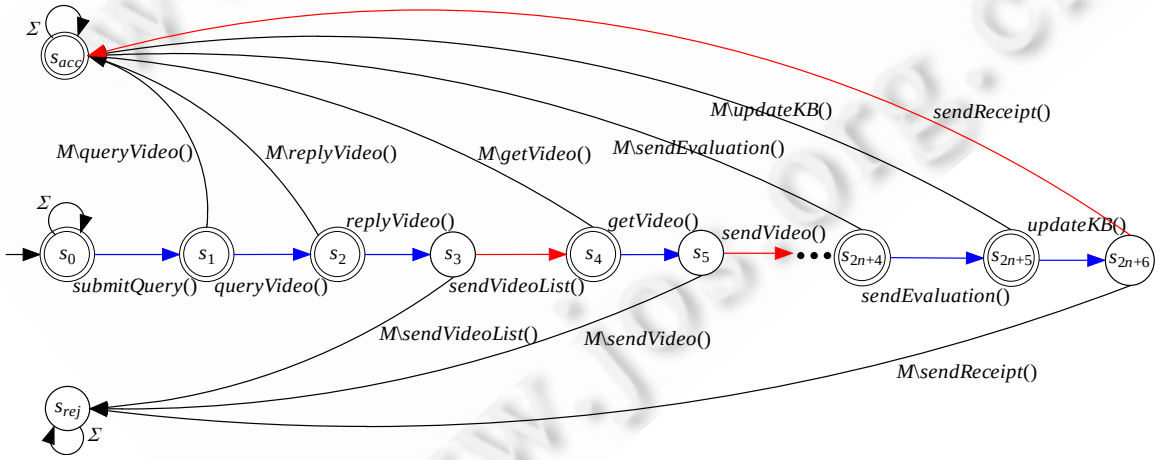
表 1 性质和相应的 WABA

Properties	Property types	# Component instances	# Events	# States	# Transitions
P1	Liveness	4	$2n+7$	$2n+9$	$6n+22$
P2	Safety	3	10	10	25



(a) uMSD for P1

(a) P1 的 uMSD 图

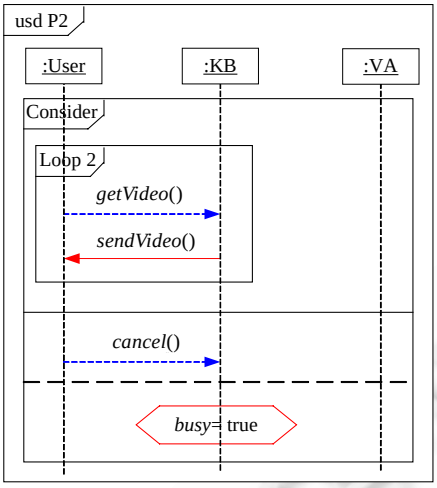


(b) WABA for P1

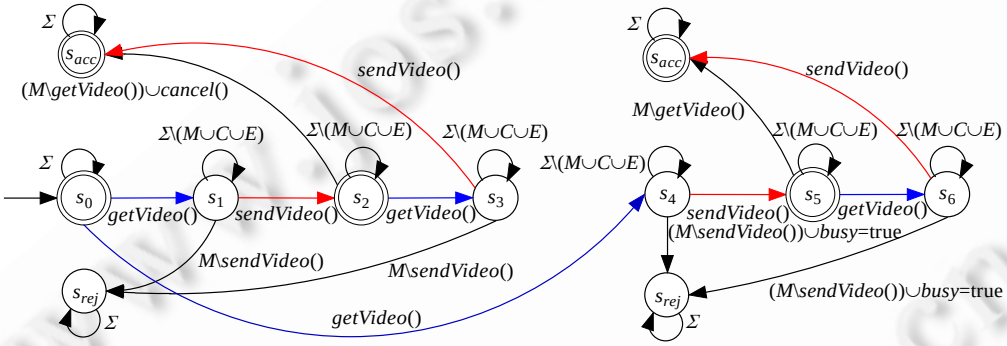
(b) P1 的 WABA

Fig.5 uMSD and the corresponding WABA for P1

图 5 P1 的 uMSD 图和相应的 WABA



(a) uMSD for P2
(a) P2 的 uMSD 图



(b) WABA for P2
(b) P2 的 WABA

Fig.6 uMSD and the corresponding WABA for P2

图 6 P2 的 uMSD 图和相应的 WABA

5 相关工作

目前,在基于场景的需求工程领域已有一些代表性的语言,如 MSC,LSC 和 UML 2.0 顺序图来表示系统的需求或性质,当前的许多研究工作都是围绕这 3 种语言展开的.

由于 MSC 缺乏描述行为触发的构造子,Sengupta 和 Cleaveland[7]提出 Triggered Message Sequence Charts (TMSC)是构件级规约而不是系统级规约,触发子(trigger)表示条件来扩展 MSC,用接受树(acceptance tree)解释了一组操作符的语义,并用精化概念来确定 TMSC 是否具有一致性.然而,TMSC 没有明确区分 universal 场景和 existential 场景,其表达力是有限的.

为了形式化 LSC 的语义,Klose 等人[8]用时间 Büchi 自动机来解释 LSC 的语义,并将 LSC 集成到验证工具 STVE 中,用于验证时间性质.然而,他们并未给出带有操作符的 LSC 图算法,所以不能组合基本的 LSC 图.类似地, Zanolin 等人[9]提出用 UML 状态图描述系统模型,用 LSC 表示时态属性,并转换成自动机,再将自动机转换成 Promela 代码,作为模型检验工具 SPIN 的输入,用于验证系统模型是否满足某种属性.该方法未讨论用 LSC 表示

的时态属性与用 LTL 表示的时态属性是否等价.

在文献[10]中,STAIRS 是基于迹的需求规约方法学,是由扩展 UML 2.0 顺序图得来的,交互语义是用正迹集和反迹集来定义的,正迹集表示强制的或可能的迹集,反迹集表示不期望发生的迹集,而既不属于正迹集又不属于反迹集的迹称为 inconclusive 迹.STAIRS 定义了一个新的强制选择操作符 *xalt* 以表示强制选择行为,而 *alt* 只在不充分描述时才会用到,这体现了增量开发的思想.在 STAIRS 中,*assert* 操作符使得所有的 inconclusive 迹为反迹;而在 uMSD 中,*assert* 操作符仅限制出现在该图中的事件序列,而没有限制未出现在该图中事件的顺序.如果需要改变事件之间的偏序关系,可以使用 *consider*/*ignore* 操作符.

性质顺序图(property sequence chart,简称 PSC)也是一种基于场景的图形化语言[11−13],是由 UML 2.0 顺序图的子集扩展而来的,用于表示并发构件之间的时态性质.在 PSC 中,regular 消息作为前置条件来触发 required 消息的执行,这与 uMSD 中 *cold* 事件作为条件触发 *hot* 事件执行是类似的;用 fail 消息表示安全性,这与将 *negate* 操作符应用到一个交互片段上是等价的.然后,PSC 是 UML 2.0 顺序图的子集的扩展,该规约具有 unwanted 消息限制和链限制,与 uMSD 相比,依然不够直观.另外,PSC 只能表示消息限制,不能表示条件限制.

6 结论语

MSD 是一种基于场景的图形化语言,借鉴了 LSC 的思想,并由 UML 2.0 顺序图扩展而来.本文简单介绍了 MSD 的基本概念,定义 uMSD 的形式语法、语义和一致性,并使用性质规约模式证明了 uMSD 具有较强的表达能力.最后,以 Web 服务 OJA 为实例展示了 uMSD 的可行性.

在下一阶段,我们将开发支持 uMSD 分析和验证的自动化工具,以帮助软件工程师来表示系统的时态性质.通过对系统建立形式化模型,将 uMSD 描述的性质和系统模型作为某个模型检验器的输入验证系统模型是否满足给定的性质.进一步研究运行时如何用 uMSD 表示系统运行时的需求,并持续监控系统行为是否满足给定的性质,初步的研究成果可参见文献[14].另外,对于一类以时间为关键性质的系统[13],考虑对 uMSD 添加时间约束来表示实时需求.本文未讨论 eMSD 的使用,下一步还将考虑如何使用 eMSD 进行测试以及如何将 eMSD 精化为 uMSD.

References:

[1] ITU-T. Recommendation Z. 120: Message Sequence Charts. Geneva: ITU-T, 1999.

[2] Damm W, Harel D. LSCs: Breathing life into message sequence charts. Formal Methods in System Design, 2001,19(1):45−80. [doi: 10.1023/A:1011227529550]

[3] Object Management Group (OMG). UML: Superstructure Version 2.2. 2009.

[4] Harel D, Maoz S. Assert and negate revisited: Modal semantics for UML sequence diagrams. Software and Systems Modeling, 2008,7(2):237−252. [doi: 10.1007/s10270-007-0054-z]

[5] Kupferman O, Vardi M. Weak alternating automata are not that weak. ACM Trans. on Computational Logic, 2001,2(3):408−429. [doi: 10.1145/377978.377993]

[6] Dwyer MB, Avrunin GS, Corbett JC. Patterns in property specifications for finite-state verification. In: Boehm B, Garlan D, Kramer J, eds. Proc. of the 21th Int'l Conf. on Software Engineering. CA: IEEE Press, 1999. 411−431. [doi: 10.1145/302405.302672]

[7] Sengupta B, Cleaveland R. Triggered message sequence charts. IEEE Trans. on Software Engineering, 2006,32(8):587−607. [doi: 10.1109/TSE.2006.82]

[8] Klose J, Wittke H. An automata based interpretation of live sequence charts. In: Margaria T, Yi W, eds. Proc. of the ETAPS 2001 and TACAS 2001. LNCS 2031, Heidelberg: Springer-Verlag, 2001. 512−527. [doi: 10.1007/3-540-45319-9_35]

[9] Zanolin L, Ghezzi C, Baresi L. An approach to model and validate publish/subscribe architectures. In: Edwards SH, ed. Proc. of the SAVCBS 2003 Workshop at ESEC/FSE. Ames, 2003. 35−41.

[10] Haugen Ø, Husa KE, Runde RK, Stølen K. STAIRS towards formal design with sequence diagrams. Software and Systems Modeling, 2005,4(4):355−367. [doi: 10.1007/s10270-005-0087-0]

[11] Autili M, Inverardi P, Pelliccione P. A scenario based notation for specifying temporal properties. In: Proc. of the 5th Int'l Workshop on Scenarios and State Machines: Models, Algorithms and Tools. 2006. 21−28. [doi: 10.1145/1138953.1138959]

[12] Zhang PC, Zhou Y, Li B X, Xu BW. Property sequence charts: Formal syntax and semantic. Journal of Computer Research and Development, 2008,45(2):318−328 (in Chinese with English abstract).

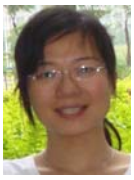
[13] Zhang PC, Li BX, Li WR. Syntax and semantics of timed property sequence chart. Journal of Software, 2010,21(11):2752−2767 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3711.htm [doi: 10.3724/SP.J.1001.2010.03711]

[14] Li WR, Wang ZJ. Monitoring composite services with universal modal sequence diagrams. In: Proc. of the 16th Asia-Specific Software Engineering Conf. (APESC 2009). Penang: IEEE Computer Society Press, 2009. 69−76. [doi: 10.1109/APSEC.2009.65]

附中文参考文献:

[12] 张鹏程,周宇,李必信,徐宝文.属性序列图:形式语法和语义.计算机研究与发展,2008,45(2):318−328.

[13] 张鹏程,李必信,李雯睿.时间属性序列图:语法和语义研究.软件学报,2010,21(11):2752−2767. http://www.jos.org.cn/1000-9825/3711.htm [doi: 10.3724/SP.J.1001.2010.03711]



李雯睿(1981－),女,河南开封人,博士,讲师,CCF 会员,主要研究领域为形式化方法,Web 服务建模,分析和验证,Web 服务组合.

张鹏程(1981－),男,博士,讲师,CCF 会员,主要研究领域为软件建模、分析和验证.

王志坚(1958－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件复用技术,构件技术,分布式计算.