ISSN 1000-9825, CODEN RUXUEW
Journal of Software, Vol.19, No.5, May 2008, pp.1099−1112
DOI: 10.3724/SP.J.1001.2008.01099

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

支持协商的网构软件体系结构行为建模与验证*

周 立[1,2], 陈湘萍[1,2], 黄 罡[1,2+], 孙艳春[1,2], 梅 宏[1,2]

[1](北京大学 信息科学技术学院 软件研究所,北京 100871)

[2](北京大学 高可信软件技术教育部重点实验室,北京 100871)

Negotiation-Enabled Modeling and Verification of Architectural Behavior of Internetware

ZHOU Li[1,2], CHEN Xiang-Ping[1,2], HUANG Gang[1,2+], SUN Yan-Chun[1,2], MEI Hong[1,2]

[1](Software Institute, School of Electronics Engineering and Computer Science, Peking University, Beijing 100871, China)

[2](Key Laboratory of High Confidence Software Technologies of Ministry of Education, Peking University, Beijing 100871, China)

+ Corresponding author: E-mail: huanggang@sei.pku.edu.cn, http://www.sei.pku.edu.cn

Zhou L, Chen XP, Huang G, Sun YC, Mei H. Negotiation-Enabled modeling and verification of architectural behavior of Internetware. *Journal of Software*, 2008,19(5):1099−1112. http://www.jos.org.cn/1000-9825/19/1099.htm

Abstract: For the uncertainty and incompleteness in the behavior of Internetware, a negotiation-enabled approach to modeling and verification of software architectural behavior is proposed in this paper. In the modeling, this approach takes the notations of UML sequence diagram and extend the model with new elements to support the uncertainty and incompleteness. In the verification, the model checker Spin is employed for checking behavior models. In addition, a negotiation-enabled solution is proposed for the verification with uncertain and incomplete model based on the counter example guided abstract refinement method.

Key words: Internetware; software architecture; model checking; behavior model

摘 要: 针对网构软件行为中的不确定性和不完整性,提出了一种支持协商的网构软件体系结构行为建模与验证方法.在建模中,该方法借鉴了 UML 时序图元素表示法,并增加了建模元素支持行为的不确定与不完整建模.在验证中,除了集成广泛应用的模型检查工具 Spin 以提供行为模型的验证能力以外,还引入了基于反例引导的抽象-精化过程思想的协商检查,以解决不确定和不完整建模所带来的正确性验证问题.

关键词: 网构软件;软件体系结构;模型检查;行为模型

中图法分类号: TP311 **文献标识码**: A

软件体系结构(software architecture,简称 SA)中的行为建模,本身就具有不确定性和不完整性的特点.不确定性指的是系统的部分行为是无法精确描述的.造成不确定的因素,有可能是设计人员缺乏对系统细节的足够了解而无法给出精确的信息,也可能是系统中一些无法预计的行为造成的.不完整性指的是其行为描述可能只包括部分行为或者对验证所需信息不够充足.而体系结构的设计往往是针对大型系统的,描述其完整的系统行

* Supported by the National Natural Science Foundation of China under Grant Nos.90412011, 90612011, 60503028 (国家自然科学基金); the National Basic Research Program of China under Grant No.2002CB312003 (国家重点基础研究发展计划(973))

Received 2007-06-10; Accepted 2007-10-15

为需要耗费大量的时间和人力,这通常是一项不可能完成的任务.

作为 Internet 环境下的一种新型的软件形态,网构软件对传统的开发理论、方法和技术提出了一系列的挑战[1].在行为建模中,其不确定和不完整的特点也更为突出.由于网构软件运行在开放、动态、多变的 Internet 环境下,其行为本身就存在很多无法预计的因素.而且其自主性、演化性、协同性等增大了设计人员对行为进行精确描述的难度.这不只增加了行为描述中的不确定性,也使设计人员给出完整行为描述的难度上升.如何有效地处理行为建模中的不确定性与不完整性,对于网构软件的行为正确性验证也更为重要.

为了支持不确定性,不确定行为的描述应该作为建模中的元素得到支持.这样不仅可以将系统中的不确定行为记录下来进行分析,而且可以减轻建模人员的工作量,使得对于与系统正确性或者可靠性影响不大的部分或者前期分析不够精确的部分,可以用不确定行为来描述甚至被忽略.然而,不确定性的描述通常隐含着系统的可达状态过多乃至无限,并可能会导致验证无法进行.此时,需要对无限的状态进行抽象,减少系统的可达状态或者使状态有限.

而不完整性是设计人员在正确性需求与模型精化的工作量之间进行权衡的结果.对软件体系结构进行完全而精确的行为建模需要大量的时间与劳动,但是对其中正确性和可靠性要求非常高的部分,只有足够的精化才能进行有效的验证.对什么部分进行精化、精化到什么程度才足够,是设计人员经常需要面对的问题.因此,行为模型中与正确性约束相关的行为很难在建模中一次性建模完全.但是,当对行为模型进行验证时,如果与正确性约束相关的系统描述不够精确,可能会影响验证的正确性.由于精确描述的困难与验证对精确性的要求相矛盾,因此对体系结构行为的验证只提供基本的验证支持是不够的.这样,需要在一个迭代的过程中,通过提示用户所需精化的部分和需要精化的程度来协助用户对模型的精化.

鉴于行为建模中的不确定性和不完整性所带来的问题,本文提出了基于协商的网构软件体系行为的建模及验证方法.本文的工作基于原有 ABC(architecture based component composition)方法对网构软件开发的研究[2],在支持构件建模与组装的基础上,提供对构件行为的建模与验证.在建模中,本方法借鉴了 UML 时序图元素表示法,并增加了建模元素支持行为的不确定与不完整描述.在验证中,除了集成广泛应用的模型检查工具 Spin[3]以提供行为模型的验证能力以外,还引入反例引导的抽象-精化过程的思想[4]的协商检查,以解决不确定和不完整建模所带来的正确性验证问题.其中,对建模中不确定性可能导致的无限状态问题,通过支持自动抽象来解决;对于用户精化过程,通过提供错误路径帮助用户定位建模中可能的精化不足,并对其精化结果的有效性进行验证,帮助用户获得有效的验证.本方法的支持工具原型已在 ABCTOOL 上实现.

本文第 1 节概述协商检查的方法.第 2 节介绍行为建模的方法.第 3 节介绍对于模型检查的支持.第 4 节介绍协商检查.第 5 节以实例来验证方法的可行性和有效性.第 6 节介绍相关工作.第 7 节是全文总结与未来工作展望.

1 方法总览

本文的支持协商的行为建模与验证方法主要包括两个模块:基本的模型检查和协商.在基本的模型检查模块中,首先通过工具将行为模型转换为 Spin 的描述语言 Promela,然后进行验证,最后再将验证的结果转换到行为模型中.协商模块建立在基本模型检查功能的基础上,支持用户对模型的迭代精化.其中,通过对行为模型进行自动抽象来解决不确定建模所可能导致的无限状态问题.在精化过程中,支持用户根据验证的结果对行为模型进行再解释(包括精化或者忽略约束),并对解释的有效性进行校验(如图 1 所示).

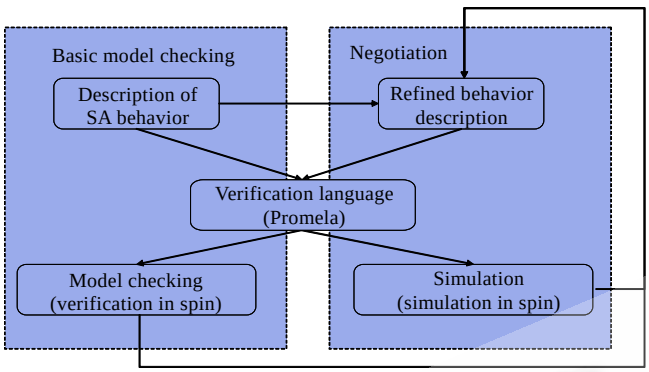


Fig.1 Approach overview

图 1 方法总览

2 行为建模

本文中的行为,指的是软件体系结构层次的行为.ABC 中的行为模型主要是基于 UML 2.0 顺序图中的常用建模元素,并增加了新的元素,以支持不确定与不完整的行为描述以及约束的定义.

2.1 元模型

ABC 行为模型基于 UML 2.0 顺序图[5],选择了其中的常用建模元素来描述软件体系结构的行为,包括生命线(lifeline)、事件(OccurrenceSpecification)和消息(message)、复合片段(CombinedFragment)等.对于复合片段,支持 5 类复合片段:选择(alternative,alt)、并行(parallel,parr)、严格顺序(strict sequencing,strict)、循环(loop,loop)和可选(option,opt),其类型由属性“InteractionOperatorKind”进行声明.而对于消息,支持同步(synchronous)和异步(asynchronous)消息,由属性“MessageSort”进行声明.在此基础上,ABC 行为模型中支持变量定义,并加入了计算(calculation)事件[6].计算事件可以通过事件声明(OccurenceSpecification)中的 Calculation 选项进行声明.其元模型使用 MOF 描述,结构如图 2 所示.

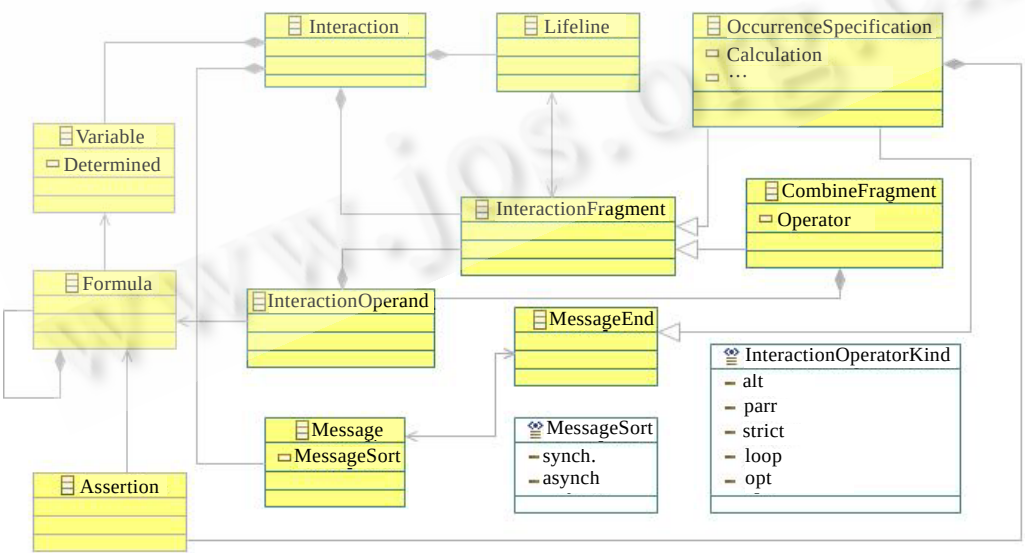


Fig.2 Meta-Model of behavior model

图 2 行为模型的元模型

对于 UML 顺序图中已有元素,其语义[7]不发生改变,而增加的元素是变量以及计算事件.

定义 1(变量). 对一个给定的顺序图 *SD*,都有一个变量集 *V*(*SD*)与之对应.*V*(*SD*)是一个二元组⟨*UV*,*DV*⟩.*UV* 和 *DV* 是两个变量集合 $UV=\{UV_1,UV_2,\ldots,UV_n\}$,$DV=\{DV_1,DV_2,\ldots,DV_n\}$.其中,*UV* 称为不确定变量(uncertain variables),*DV* 称为确定型变量(determinate variables).每一个变量都有一个类型,确定型变量的类型可以是 ANSIC 语言支持的任一数据类型,而不确定变量的类型是命题、实数或整数,即{*Proposition*,*Real*,*Int*}.每一个确定变量 DV_i 对应一个初始值 IDV_i,表示在顺序图 *SD* 开始时,变量的初始取值.而不确定变量集合 *UV* 对应一个初始取值范围 *IUV*.取值范围 *IUV* 是一组布尔表达式$\{B_1,B_2,\ldots,B_l\}$,其中的自由变元的集合是 *UV*.其语义是使得这些布尔表达式的合取为真时,所有 *UV* 中变量取值的集合.

举例说明,如有 *A*,*B* 两个不确定变量,*A* 为实数,*B* 为命题,{(*A*>0),(*A*⇐1)}代表的语义是:在顺序图的开始,变量 *A*,*B* 取值不确定,其取值范围恰为集合$\{\langle A,B\rangle|0<A\leq 1\}$.

为了描述变量随着顺序图中事件的发生而产生的变化,还需要进一步引入计算事件和不确定改变.

定义 2(计算事件). 计算事件是一种事件(occurence),可以表述为二元组 *CO*=⟨*CCode*,*ChangeUV*⟩.其中,*CCode* 是确定型变量的改变,是一段可执行的 C 代码.*ChangeUV* 是不确定变量的改变.为了避免歧义,规定计算事件的语义为:先执行 *CCode*,再执行 *ChangeUV* 的语义.

在计算事件中,*CCode* 的语义等同于执行这段 C 代码,下面给出 *ChangeUV* 的语义.

定义 3(不确定改变). 一个不确定改变 *ChangeUV* 是一个二元组⟨*Decl*,*Change*⟩,其中,*Decl* 是 *UV* 的子集,声明了 *ChangeUV* 中发生改变的变量.*Change* 则代表这些变量的不确定更改动作.其 BNF 范式如下:

Change::=*Change*‘|’*Change*|*CE*;

CE::=*BExp*‘→’*Change*’*Tuple*‘→’*BExp*.其中,*Tuole*∈*Decl* 是一组不确定变量,*BExp* 是关于 *Decl* 的布尔表达式.其中,如果 *CE* 中的布尔表达式可以略去,则表示恒真.

举例说明,*Decl* 为“*A*,*B*”,*Change* 为“1→*Change*(*A*,*B*)→(*A*>0)”就可以表示一个不确定改变,而且可以简写为“*Change*(*A*,*B*)→(*A*>0)”,其语义不发生改变.

根据上述文法,不确定改变由“|”隔开的若干个分支构成.每个分支包括前、后两个布尔表达式和中间的更改部分.前面的布尔表达式是锁(guard)条件,表示当该布尔式为真时,改变操作得以进行.后面的布尔表达式为后置条件(post-condition),表示更改发生后所更改的变量需要满足的条件.其具体语义如下:

定义 4(不确定改变的语义). 设计算事件声明改变的变量为 $V_1,V_2,\ldots,V_k$,设一个更改为 $CE_1|CE_2|\ldots|CE_m$,设 $CE_i = P_i \rightarrow Change(V_i^1,V_i^2,...,V_i^m) \rightarrow Q_i$,其中,$\{V_i^1,V_i^2,...,V_i^m\} \subseteq \{V_1,V_2,...,V_k\}$.如果$\{(V_1,V_2,\ldots,V_k)=(v_1,v_2,\ldots,v_k)\}|=P_1$,$P_2,\ldots,P_{j-1}$,但是,$\{(V_1,V_2,\ldots,V_k)=(v_1,v_2,\ldots,v_k)\}|\neq P_j$,那么,如果$(v_1,v_2,\ldots,v_k)$是计算事件发生前变量组$(V_1,V_2,\ldots,V_k)$的取值,那么在该计算事件发生后,变量组$(V_1,V_2,\ldots,V_k)$可能的取值范围为$\{(v_1,v_2,\ldots,v_k)|\{(V_1,V_2,\ldots,V_k)=(v_1,v_2,\ldots,v_k)\}|=Q_j\}$.如果$\forall i,\{(V_1,V_2,\ldots,V_k)=(v_1,v_2,\ldots,v_k)\}|\neq P_i$,那么计算事件发生后,$(V_1,V_2,\ldots,V_k)$的取值范围为全集 *I*,即可以取任意值.

图 3 是 ABCTOOL 中描述计算事件的属性栏界面,其中,*Changed* 属性声明了该计算事件改变的变量 *A*,*B*(均为)实数类型,Calculation 指明了所进行的更改.如果变量 *A*,*B* 在事件发生前的取值为 *A*=1,*B*=1,那么在该事件后的取值范围是$\{(A,B)|(A=1\wedge B>1)\}$;如果事件发生前的取值为 *A*=−1,*B*=0,则该事件发生后,(*A*,*B*)的取值范围为$\{(A,B)|(B=0)\}$.

如果一个计算事件中的不确定改变不包括任何 Calculation 语句,则称这个改变为自由的(根据上面的语义,声明改变的变量可以改变为任意值);否则,称这个改变为受限的.

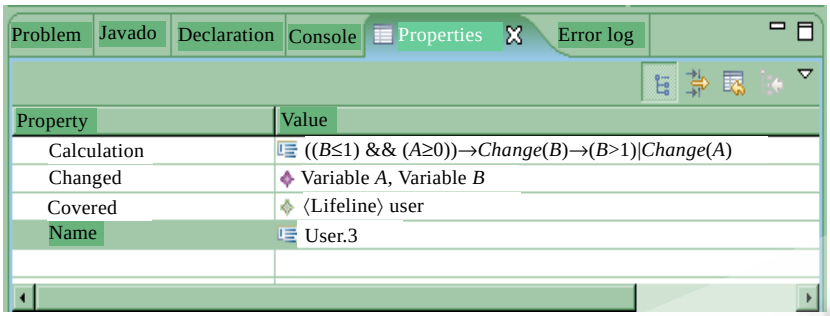


Fig.3 An example of calculation event

图 3 计算事件示例

2.2 约束建模

行为约束建模的是系统应当满足的正确性约束.在行为模型中,主要关注的是安全性的(safety)行为约束,即系统若违反某个约束,则表示系统错误.在 ABC 行为模型中,提供了两种表示方式:事件约束和状态机约束.

事件约束表示在可能的初始数据输入下,任何一条可执行路径在某个事件刚执行完的时刻,系统变量的取值应当满足的约束.事件约束表达的形式是一个以模型中若干个变量为变元的布尔表达式.当其取值为假时,表示该事件约束被违反.事件约束使行为模型具有描述不变式约束的能力.

状态机约束指的是用户画出一个状态机,以描述系统预期的执行序列.每个迁移可以是一个事件,也可以是一个同步消息.状态机约束表示系统在任何一种可能的执行方式下,发生的事件序列必须是状态机约束接受的语言.

3 模型检查

3.1 模型检查工具

模型检查器以 Spin 作为内核,由语法转换器和路径解析器两部分组成.它的输入是顺序图和正确性约束,由语法转换器将顺序图转换为 Promela,并将约束插入到模型中的对应位置.工具的输出文档是记录违反路径的事件序列的编号和不确定状态的改变方式.这是由路径解析器将 Spin 的检查结果的记录进行再解析,并标记到行为视图上去.目前,其对应的 ABCTool 行为视图正在开发中(如图 4 所示).

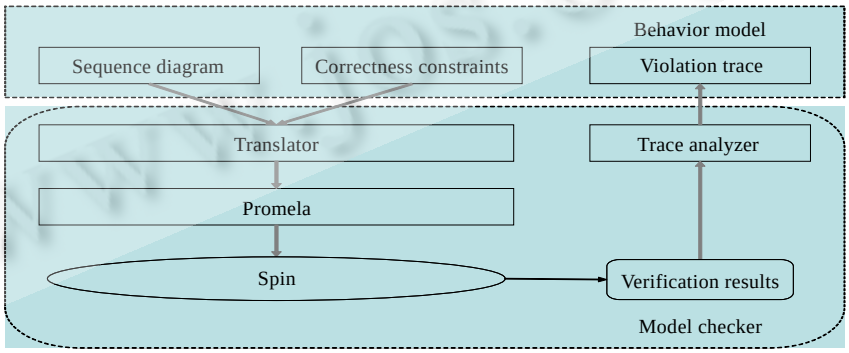


Fig.4 Structure of the model checking tool

图 4 模型检查工具的结构

3.2 模型转换

为了支持行为模型的验证,模型转换包括两个部分:从顺序图和约束转换为 Promela 和从用 Promela 表示的

验证结果转换到行为模型.

从顺序图和约束转换为 Promela,是通过在行为模型建模元素与 Promela 元素之间的对应来进行的.其主要对应关系见表 1.

Table 1 Table of elements used in the model transformation

表 1 模型转换元素对应表

Meta model elements		Promela elements
Occurrence specification		Atomic block
Lifeline		Proctype
Message		Chan
Combined fragment	Loop	do: (Exp)→seq od;
	Choice	if: (Exp)→seq fi;
	Strict sequence	Thread sych
	Parallel	*New proctype*()
Variable	Certain variable	C variables (embedded code)
	Uncertain variable	Proposition variables
Assertion		Assert (Exp)
Formula		*never*{}

行为模型元模型中选用的 UML 顺序图中的元素,是基于 UML 交互的形式化定义[7]将其转换为 Promela 的.这些基本元素包括生命线、消息、事件和复合片段.

对于复合片段,根据元模型中支持的 5 种复合操作类型(InteractionOperandKind)将其分为循环片段(loop)、选择片段(alt 和 opt)、严格顺序片段(strict)和并行片段(parr)4 类,再根据各个复合操作(CombinedOperand)的形式化定义对其进行转换.

对于变量的转换,按照确定变量和不确定变量分别进行转换.

对于确定变量,为了支持多种变量类型,而不局限于 Promela 所支持的基本类型,采用 Promela 内嵌 C 代码的方法对其进行转换.首先,将确定型变量声明成 C 变量,然后建立对改变量的地址的追踪,从而将变量作为全局变量.

对于不确定变量,为了保证其状态有限,首先对非命题型变量进行抽象.除了命题型不确定变量以外,不确定变量的表达式全部被替换为命题型不确定变量(包括该变量的前置和后置约束).具体的自动抽象方法将在协商检查部分进行介绍.抽象后,非命题型不确定变量之间的关系也转换为命题型不确定变量之间的逻辑关系,通过将其逻辑关系表示为相应的限定变量取值的内嵌 C 函数来实现.由于所有的不确定变量都转换为命题型不确定变量,因此对不确定变量的不确定改变也变成了对命题型不确定变量的真值表的遍历问题.对于 Promela 中的遍历问题,利用内嵌 C 代码的功能来解决.

对于事件约束,直接将其转换为 Promela 中的断言“*assert*()”,然后添加在相应事件之后.事件约束与事件之间的对应关系,是通过在相应事件中添加相应的输出语句,然后在路径解析器中进行解析.由于每个事件都是转换为 Promela 中的 atomic 关键字括起来的程序块,为了保证事件约束的校验在事件执行完毕后立即进行,将事件约束添加在其对应事件的 atomic 块内部,并增加相应的同步语句,就可以实现.

对于状态机约束,将状态机约束中的每个状态转成 Promela 中否定声明“*never*{}”中的标记;而每个状态迁移,其标记的事件或同步消息转换为否定声明中的分支语句“if”的条件,其对应的两个状态之间的跳转也转换为 Promela 中否定声明中标记之间的跳转.这样,对于每一条执行路径中发生的事件或者同步消息,都检查是否满足否定说明中的一个条件分支,也就是符合状态机约束中的一个状态跳转.如果一条执行路径不符合状态机约束,那么会在否定声明中由于没有可执行的条件分支而被 Spin 检查出来.

最后,需要将用 Promela 表示的验证结果转换到行为模型中.因为 Spin 检查后返回的路径是按照其自身的编码方式存储的,按照 Spin 的解析器进行解析后的事件的发生顺序是以在原 Promela 的文本文件中的行数来显示的,难以对应到行为模型的事件路径上.所以,在 Promela 中增加一个变量来记录事件.对每一个事件,都在其发生后插入对该变量的赋值以记录事件的发生.然后,在该变量赋值后插入 Promela 的输出语句“*printf*()”(仅在

Spin 模拟时生效),就可以在验证后模拟违反路径时输出这个变量的值.这样,根据该变量的取值序列,可以将违反路径解析到行为模型上.

在整个转换过程中还涉及到同步和复合片段的处理等技术问题.处理的办法涉及到锁机制以及一些状态控制的办法.具体的技术细节这里从略.

4 协商检查

为了对包含不确定性的行为模型进行验证,并帮助用户对模型进行有效的精化,本节在行为模型建模以及模型检查的基础上提出了一种支持协商的方法.协商检查借助了模型检查和程序自动化验证中反例引导的抽象-精化过程(counterexample-guided abstraction refinement,简称 CEGAR)[4]的思想,主要借鉴其中使用反例来改进抽象函数的思想,通过与用户进行交互的方式对体系结构模型进行半自动调整、修改和精化.对于用户,精化是在协商的过程中完成的.所以,本文称其为协商检查.

本节先介绍作为协商检查方法借鉴的 CEGAR 过程,之后介绍协商方法的过程以及如何保证协商过程的有效性与正确性,并且通过一个例子来解释协商检查的过程,最后讨论协商方法的不足与局限性.

4.1 方法基础

反例引导的抽象-精化过程,即 CEGAR 思想,源于 21 世纪初一些借助定理证明或者可满足性工具(SAT)与模型检查器结合的方式来进行抽象的模型检查的研究工作[8,9].它主要应用于模型检查和程序自动化验证方面,以达到算法和验证能力上的改进.在 2003 年,E.M Clarke 对这种形式的模型检查方法进行总结[4],并定义如下:

抽象-精化过程是一种使用某种自适应的算法通过分析伪反例不断改进验证中的抽象函数直至达到验证目的的方法,它一般包含如下 4 个步骤:

(1) 初始化:生成一个抽象函数.

(2) 模型检查:验证给定的抽象模型.如果检查的结果认定系统符合所声明的约束,那么算法结束,并且报告系统正确;否则,生成一个该抽象模型的反例.

(3) 健全检查:确定该反例是否是一个伪反例(即该反例是抽象模型的反例而不是真实系统的反例),如果不是伪反例,那么,该反例成为最终结果并且输出;否则,进入步骤(4).

(4) 精化:精化抽象函数,使得该反例可以避免,并且返回步骤(2).

这种思想对模型抽象和精化的支持给现有软件体系结构的分析和验证研究带来了新的启发.抽象与精化正是解决软件体系结构行为模型的不确定性和不完整性的对策.抽象方法对行为模型中的不确定变量和不确定改变进行抽象,使状态有限或大为减少,从而保证验证的可行性和有效性;精化使与约束相关而用户没有提供的行为描述不断添加到模型中,支持正确的验证.

不同于这种思想在模型检查和程序自动化验证中的应用,体系结构层次的模型不是一个精确的模型,所以抽象与精化的过程无法达到完全自动化.其中的健全检查与精化两个过程,必须加入用户的参与.用户根据系统所提供的反例,基于对系统的理解来进行精化.

4.2 协商检查的过程

借助 CEGAR 思想,本文提出了软件体系结构行为模型的协商检查方法.用户输入软件体系结构的行为模型,作为协商检查的开始.模型经过自动的抽象以避免不确定变量所可能引起的无限状态问题,之后经过转换并进行验证.模型检查证明系统的正确性或者提供其存在的错误路径.对于存在的每个错误路径,都需要用户判断是否是一个伪反例.如果是伪反例,用户需要对模型进行解释,并且解释后的模型能够避免错误路径.这时,可以认为用户提供了一个合理的精化解释,将其记录到模型中.协商检查的流程如图 5 所示,其中,浅色的部分需要用户的参与,深色的部分可以通过工具支持自动完成.

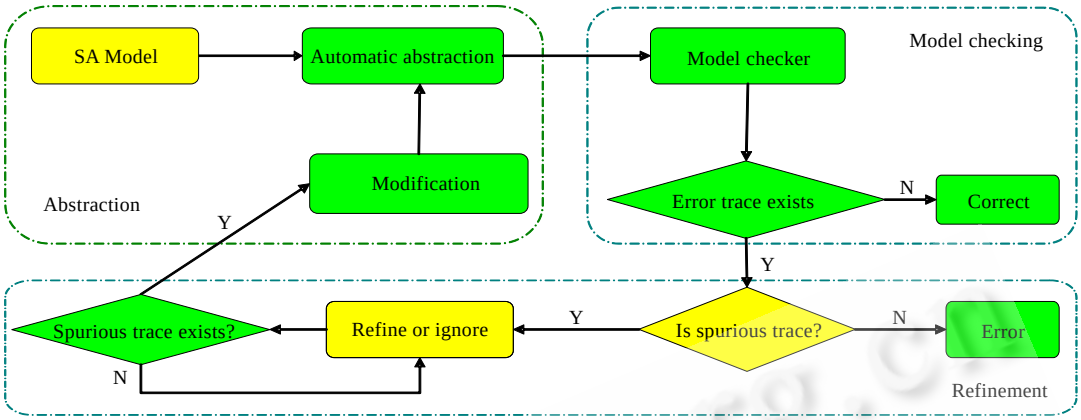


Fig.5 The process of negotiation-enabled model checking

图 5 协商检查流程

4.3 协商支持

为了支持上述协商过程,体系结构层次的模型检查器需要提供给用户尽可能多的自动化支持.其中主要包括自动抽象和帮助用户精化两个部分.

4.3.1 自动抽象

所谓抽象,就是把一个变量从取值范围较多甚至是无限的可能取值状态转化为与验证相关的有限种状态.设想一个实数型变量 A,取值范围为 $A>0$,遍历其所有取值是不现实的.因此,对于非命题型不确定变量,由于模型检查器不可能对其所有取值进行遍历,所以需要对这些变量的取值范围进行抽象.

举例说明,如果 A 是一个不确定的实数型变量,而验证关心的只是 A 是否非负,那么,A 的取值可以抽象为两种:$A\geq 0$ 和 $A<0$.而变量的状态,可能是与其他变量的状态相关的.例如,在一个顺序图中有两个不确定实数型变量,在计算事件的条件中如果出现了形如$(A<1)\rightarrow change(A)\rightarrow(A>2)|change(A)\rightarrow(A>B)$的语句,那么,$A$ 的取值还需要考虑是否符合不确定改变的锁条件与后置条件.这样,与 A 相关的不确定更改的锁条件与后置条件的布尔表达式的真值表状态均应包含在抽象后$\langle A,B\rangle$的状态中.令布尔表达式 $P1=(A<1),P2=(A<0),P3=(A>2),P4=(A>B)$,那么,布尔表达式 $P1,P2,P3,P4$ 的真值表就构成了 A,B 的取值范围的不同状态.因此在验证之前,不确定变量集 $UV=\langle A,B\rangle$的取值范围会由 $R\times R$ 被抽象为 $P1,P2,P3,P4$ 的真值表.也就是说,在验证时,不确定变量 A,B 不再存在,被替换为布尔型不确定变量 $P1,P2,P3,P4$.这样,整个验证可以通过对 $P1,P2,P3,P4$ 的取值遍历而进行.同理,对于初始取值范围,约束断言或复合片段中的交互条件含有不确定变量的情况,也需要进行类似的处理.也就是说,把用上述条件作为布尔型变量替换掉所有的不确定型整型和实型不确定变量,从而依靠对这些布尔型变量的真值表的遍历而进行验证.

如果只是进行上述的抽象,验证是无法实现的:一方面,真值表的遍历是指数级的;另一方面,一些不可满足的状态也被遍历.因此,对不可满足状态下的遍历会产生伪路径,也就是说,会检查出实际不会发生的错误.如上面例子中,$\langle P1=\text{False},P2=\text{True}\rangle$即“$A\geq 1\wedge A<0$”就是不可满足的.事实上,$P1,P2,P3$ 的真值表可以被合成为 4 个状态$\langle P1,P2,P3\rangle=\{\langle T,T,F\rangle,\langle T,F,F\rangle,\langle F,F,F\rangle,\langle F,F,T\rangle\}$,即$\{(A<0),(0\leq A<1,(1\leq A\leq 2),(A>2))$,而不是整个真值表的 8 个状态.因此,对真值表的取值需要加以限制.另一方面,由于算术运算符在模型中可以使用,因此,根据哥德尔不完备定理,整个系统的不确定变元取值范围的限制问题是不可解的.尽管如此,一些简单的限制是可以解决的.比如,可以实现对单个整型变量与常数间的“>,<,=”的逻辑关系支持.也就是说,可以将$\langle P1,P2,P3\rangle$的真值表化简为 4 个状态,而对于形如 $P4:A>B$ 或者含有算术运算的如$(A+B>0)$等就无法处理.这些命题变元之间的限制关系必须在后面的精化过程中,在用户参与下,通过声明来一步步完成.目前,对于自动抽象的研究还比较初步,所以自动化程度较低,在将来的工作中,可以通过集成第三方定理证明器来提高工具的自动化程度.这方面的研究还有待进一

步加强.

4.3.2 精化过程

为了支持协商过程中与用户的交互部分,模型检查器除了模型检查部分以外,还应该提供以下 3 方面的支持:

- 模型检查器提示用户对其模型中与约束相关却描述不清楚的地方进行解释;
- 模型检查器支持用户精化模型或者忽略其中的约束,并生成相应的文档记录下来;
- 模型检查器自动检查用户解释本身的正确性,并对不正确的解释提示用户进行重新解释.

这些目标是为了帮助用户针对其关心的系统约束进行验证以进行模型的精化,并以较小的精化代价获得所需的验证.协商过程的支持是建立在模型检查的基础上的.

首先,对用户的提示是将模型检查产生的违反正确性约束的路径显示在行为模型上.路径上所有影响约束命题状态的点都是对约束敏感的点.用户可以将这个结果与自己对系统的理解加以比对,以判断是系统错误还是由于精化不足而出现的伪错误.这个功能直接搭建于基本模型检查之上,增加用户进行判断的功能.

然后,允许用户对模型进行精化或者忽略其中的约束,实际上是允许用户建模过程的迭代.但是这个迭代必须满足覆盖条件,也就是用户精化前,可达的系统状态集一定要包含精化后可达的系统状态集.为了满足这个需求,在协商检查的支持中定义了如下 5 种精化算子:

(1) 前置/后置约束精化:用户可以追加自由改变的前置或者后置约束,或者对前置与后置约束追加限定.

(2) 追加命题追踪:要求检查器追加检查命题变元,并抽象该命题变元.

(3) 增加限定:要求增加变元之间的关系约束.由于目前 ABCTool 对于命题变元的相容性进行检查的 SAT 证明器没有集成,这导致了此方式的产生(如变元 $P1$:($a>b+1$)和 $P2$:($a>b$)的蕴涵关系尚未得到实现).

(4) 增加忽略的消息:要求在图中增加一些忽略的消息.

(5) 强制忽略约束:用户在不愿解释的情况下强制要求某个约束不检查.

上述 5 种精化算子是符合覆盖条件的,具体的证明超出本文的范围,不再进行讨论.在协商过程中,用户可以直接使用精化算子,其要求忽略的约束也由系统自动转化为强制忽略约束的精化算子.

最后,对用户解释的正确性验证,使用了模型检查工具 Spin 中提供的模拟(simulation)功能来进行.经过用户解释之后的模型仍然是一个可供验证的行为模型,这个模型经过抽象、转换之后,可以使用之前的出错路径记录进行模拟.由于精化算子本身保证了模型的覆盖条件,所以这个模拟必然是可以进行的.如果模拟的结果仍然是错误的,则证明解释是无效的,因为解释后的模型不能使这个伪错误再出现;如果模拟的结果是正确的,则可以认为解释是合理的,并且结果得以接受和记录.由于模拟消耗的时间远远小于重新进行模型检查的时间(可以在线性时间内完成),因此,对解释接受与否的判定可以迅速返回给用户.

4.4 例 子

下面就一个例子来说明协商检查的过程.其初始输入的行为模型如图 6(a)所示,包括 4 个计算事件(*Occn* 是它们的编号)和一个同步消息.如果 a,b 都是实数型不确定变量,且 a 的初值大于 0.可以看出,*Occ*4 上的事件断言是可以被满足的.但一开始,软件体系结构设计人员可能忽略了其中部分的细节,所给出的模型缺少同步消息,没有设置初始条件,并且 3 个事件只是笼统地分别写为 *Change*(a),*Change*(b)和 *Change*(a).其具体的计算事件的内容如虚线框中所示,只是用户在初始时没有明确定义出来.

Step 1. 工具将 Assert 的表达式抽象为 $P1$,3 个事件分别抽象成 *change*($P1$),*void* 和 *change*($P1$).第 1 次检查时报错,并且返回给用户错误路径:"*Occ*1,*Occ*2,*Occ*3,*Occ*4".另外,报告不确定改变中的 *Occ*1 将 $P1$ 的初值改变为 0,*Occ*3 再次将其值变为 0,导致 *Occ*4 检查报错.

Step 2. 为此,用户进行精化,他对 *Occ*1 设定了初值 $P1=1$.并且该表达式被精化为($P1=0\rightarrow$*change*($P1$)|$P1=1\rightarrow$*change*()).然而在检查后发现,原来的错误路径"*Occ*1,*Occ*2,*Occ*3,*Occ*4"仍然报出错误,用户的精化不被接受.因此,用户必须继续精化.

Step 3. 用户追加命题变量 $P2$:($b<a$)的追踪,并且给出 *Occ*2 计算事件的一个后置精化:(*change*($P2$)→

(*P*2=1))和 *Occ*3 的一个前置与后置精化(*P*2=1→*change*(*P*1)→*P*1=1),*Occ*1 的计算也被自动转为(*P*1=0)→*change*(*P*1,*P*2)|(*P*1=1)→*change*(*P*1,*P*2)→(*P*1=1).这时,路径"*Occ*1,*Occ*2, *Occ*3,*Occ*4"不再出错.这样,用户的精化被接受.

Step 4. 接受后的模型检查出另一个错误路径"*Occ*1,*Occ*3,*Occ*2,*Occ*4".因为 *b* 没有赋予初值,*Occ*3 的计算结果可能是 *P*1=0(即 *a*=*a*−*b*<0),而 *Occ*2 这时不改变 *a* 的取值,也不改变 *P*1,因此仍然报错.用户看出此路径的出现是由于缺少对应的同步消息,因此他增加了被忽略的消息.

Step 5. 检查器检查出结果正确.将精化过程 Step 4 和最终模型记录入文档,其最终模型如图 6(b)所示.可以看到,*a* 和 *b* 的无限取值可能被转化为只有 4 个可能的命题变量 *P*1 和 *P*2 的真值表.

这样,协商方法协助用户精化获得系统正确性约束验证中所需的信息并且定位了系统中约束的相关事件.

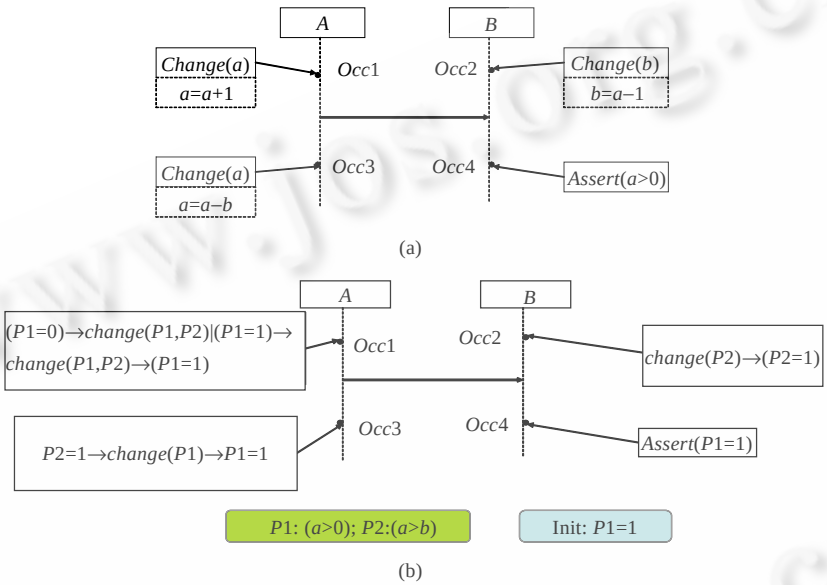


Fig.6 An example of negotiation-enabled model checking

图 6 协商检查示例

4.5 局限性

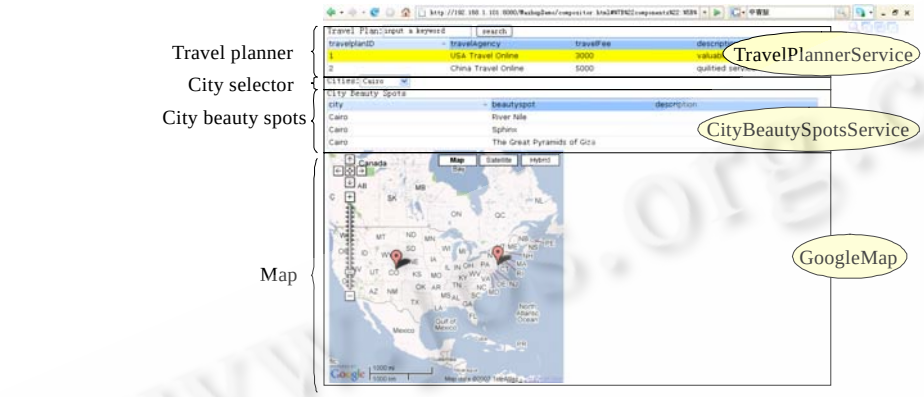
本文提出的协商方法的工具支持仍然具有局限性.用户如果对不确定性变量用算术表达式进行描述,在自动抽象过后,协商方法无法自动地将抽象后的结果精化到所需要的程度.比如上例中,即使用户一开始在 *Occ*1 中标明了 *a*=*a*+1,其部分语义也会在自动抽象中丢失.而在后面的协商过程中,用户仍然需要反复对这个表达式进行精化,这说明工具的自动化程度比较低.文献[10]表明,带有算术表达式的模型的全自动精化是一个不可解的问题.尽管如此,仍然可以通过寻找一种自动抽象-精化策略来支持含有某一类特定算术表达式的自动验证,而不需要用户对已经写明表达式的计算事件进行手动的精化.具体的精化策略选择应该使其所支持的算术表达式范围能够兼顾特定体系结构层次模型的易用性和合理性,目前仍需更深入的研究.

5 实例分析

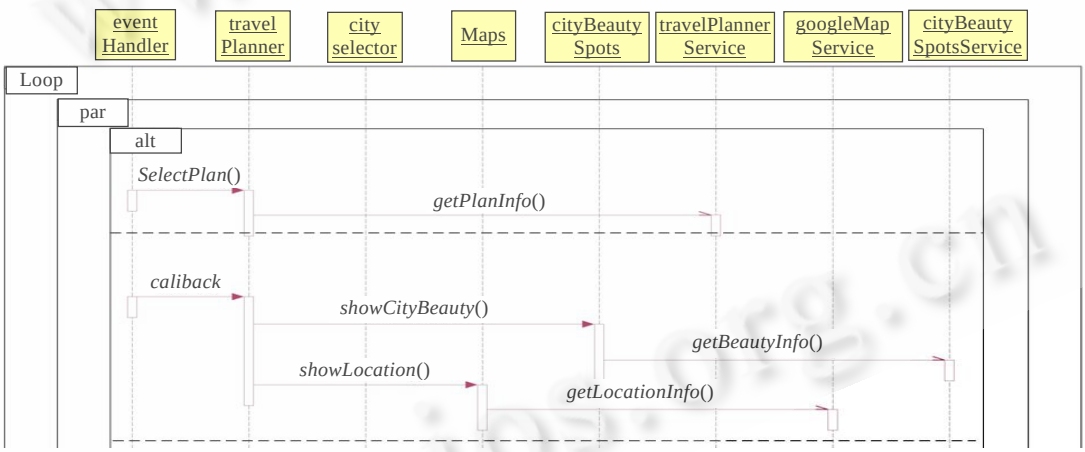
我们以一个旅游指南的 mashup 应用来评估协商检查方法的有效性.mashup 应用[11]是一种新兴的网络应用,它使用 Ajax 技术,通过将网络中第三方所提供的数据和功能集合在一起从而获得新的功能,来构建个性化的 Web 应用.mashup 应用的开发,较多地通过复用表示层构件或者服务构件将其组装起来获得目标应用,比较适合使用基于体系结构的构件组装来支持其开发.所以,我们选择一个 mashup 应用以及 mashup 应用在实际运行中

经常会出现的一个一致性问题来对我们的方法进行实例研究.

作为例子的旅游指南应用如图 7 所示,图 7(a)是应用的界面,图 7(b)是其部分的软件体系结构行为模型(例子行为模型的时序图长度为图 7(b)的 3 倍).这个应用中包含 4 个构件:(1) 一个旅游计划构件(travel planner)接受用户输入的关键字去搜索相关的旅行计划;(2) 一个目的地选择器(city selector)允许用户选择其感兴趣的城市;(3) 一个列表(city beauty spots)显示所有相关的风景名胜的详细信息;(4) 一个页面(map)显示地图.而这 4 个构件主要负责处理与其对应的后台服务相关的 GUI 层次的逻辑.



(a)



(b)

Fig.7 The views of an iMashup application

图 7 iMashup 实例视图

假设用户使用这个应用,经过搜索“New York”之后,选择页面上“USA Travel online”所提供的包括了“Denver”和“New York”的旅游路线.之后,他觉得出现在第 2 个路线中的城市“Cairo”可能也是一个不错的选择,所以他又很快地在目的搜索里面选择了“Cairo”.如果两个动作的时间间隔非常短,那么一个不一致的问题有可能出现.在列表中显示的是所有 Cairo 的风景名胜;而在地图上显示的却是前一个动作所产生的结果:旅游计划中的“Denver”和“New York”这两个城市.显示顺序的错误,导致了对于用户而言,查询的结果与其预期不一致.

这种不一致的问题在 mashup 应用中非常普遍,其出现的原因主要是因为网络延迟或者服务器反应时间的影响,导致应用接收到服务的返回结果顺序与其发出服务请求的顺序不同.

为了评估协商检查对行为模型检查的影响,对这个例子进行了对比实验:分别使用基本的模型检查和协商检查的方法来验证例子中的一致性问题.首先,我们对例子的行为进行建模,作为两种方法的输入.在行为模型

中,包括上述 4 个表示层的构件以及 3 个服务构件.其中,使用时间戳的方式来模拟用户对应用的操作,并将其作为应用中显示内容的标识.这样,当使用普通的模型检查时,一致性约束就是将要显示的内容的时间戳必须晚于之前显示的内容的时间戳.由于对时间戳的改变是无限的,普通模型检查无法进行处理,只能在对问题进行了简化后,让时间戳的取值个数限定为一个有限的数.而使用协商检查,经过自动抽象,时间戳被抽象为各个构件中内容的时间戳之间的关系来表示,因为这也是约束所关心的属性.这样,时间戳的无限改变可以得到模拟,也就是可以模拟所有用户对应用的操作.

表 2 为通过普通模型检查与协商检查之后的模型比较.

Table 2

表 2

	Traditional model checking	Negotiable check
Component number	7	7
Variable number	16	98
Changed variable number	13	13
Constraint number	7	7

可以看出,模型的复杂性变化主要是体现在变量个数的不同上,但是,由于协商方法的变量是模型通过自动抽象而获得的,所以并不需要用户另外的建模代价.而值得一提的是协商过程对于用户精化模型的辅助作用.在这个只有 7 个构件的系统中,针对一致性约束,需要用户描述的相关状态变化点就有 13 个.在基本模型检查中,用户只能手工找出这些变化点并对其进行描述,然后才能进行模型检查.而使用协商检查,变化点的描述并不需要完全精确.用户可以先给出部分的变化点,并且将不重要的变化点定义为不确定更改.协商检查器能够提示用户对不正确的结果进行解释,而用户可以迭代地对模型进行精化,直到获得这个模型为止.

在对两个模型进行验证之后,虽然两种方法都可以检查到其中不一致错误出现的路径,但是可以看出,普通模型检查不能模拟无限的用户交互的过程.为了对比其性能和可伸缩性的区别,本文通过对例子中的服务实例进行增加来进行对比实验.为了考察可伸缩性,实验在检查中强制 Spin 检查所有的可达状态.图 8 显示的是进行了服务实例个数调整之后,用不同方法进行检查的内存使用量和运行时间的情况.实验环境为 Pentium CPU 4 2.26GHz,内存为 1.5G DDR.其中,普通模型检查模拟的是用户交互次数上限为 2 的情况.图中没有显示数据的部分是由于内存溢出而无法验证的情况.可以看出,普通模型检查在增加服务构件实例之后就由于内存溢出而无法进行检查.

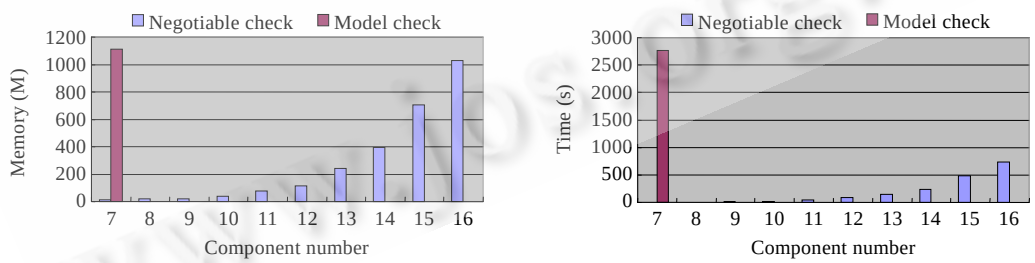


Fig.8 Experimental results

图 8 实验数据

可以看出,协商过程对行为模型的辅助,不仅在于建模能力上对不确定状态的描述、对设计人员精化模型的支持,而且也使行为模型的验证能力得到提高.

6 相关工作

目前,对体系结构层次的行为建模方法及验证的研究已有很多,代表性的工作包括 Wright[12],Rapide[13],Darwin[14]等.这些体系结构描述语言对系统行为的描述一般隐藏在连接子中,没有显式地表示出来.Rapide 采用

了事件偏序集的方式来形式化地表达系统行为,但是没有相应的工具支持该模型的自动验证.Wright 直接采用形式化语言 CSP 对系统行为进行建模,并且采用第三方 CSP 模型检查工具(如 FDR)进行正确性约束的验证.Darwin 中的行为模型采用了 FSP 进行描述.FSP 是 CSP 的简化模型,可以使用标记转移图(labeled transition system,简称 LTS)进行可视化表示.针对 FSP 和 LTS,Magee[15]等人实现了可视化的分析与验证工具 LTSA.

UML 的交互图能够对系统行为进行建模.UML2.0 出现后,针对 UML2.0 的交互图,Cengarle[7]等人通过将 UML2.0 的交互图转换到 Promela 语言上,从而支持对 UML2.0 顺序图的正确性验证.

上述工作均没有考虑行为模型的不确定性与不完整性.因此,对比上述工作,本文的主要创新点在于特定于网构环境和体系结构层面,能够对系统行为在不确定和不完整的信息下进行建模,并提出了用协商检查的方法进行验证.此外,本文提出的建模方法可以避免由 UML 不特定于体系结构层次而导致的过于庞杂,并且弥补了体系结构描述语言在学习成本上的缺陷.

7 总结与展望

网构软件运行在开放、动态、多变的 Internet 环境下,其体系结构行为建模中的不确定性与不完整性也更为突出.为了对网构软件的体系结构行为进行有效的验证,本文作为 ABC 方法的系统行为建模与验证部分,提出了基于协商的行为建模与验证方法.本方法在行为建模中借鉴 UML 顺序图的表示法,添加了不确定变量作为建模元素,并提出协商检查的方法协助用户进行有效的精化,以实现模型的正确性验证.最后,以一个 mashup 应用作为实例,验证了本文提出的协商检查方法的可行性和有效性.目前,协商检查的支持工具(包括行为模型的验证工具)的原型已经开发完毕,而行为建模的图形化界面仍在开发.

本文提出的协商检查方法虽然对不确定性和不完整性在建模和验证过程中提供了支持,但是,本文的工作仍然存在不足,其自动化支持的程度仍然不够高.在下一阶段,我们将进一步研究如何为用户的行为描述精化过程提供更多的自动化支持.具体来说,通过采用一些在模型检查和机器证明中比较成熟的方法,如组合推理等方法,来减少用户精化的工作量以及提供更多的引导信息.

在现阶段,该方法所应用的实例仍然较为简单,仅用于展示所提出的协商检查方法的可行性与有效性.在未来的工作中,作者将把协商方法运用于更多的实例,特别是更大规模的系统,并在实际使用中进一步收集用户的使用意见以及考察本方法的有效性.

另外,我们还将在现有工作的基础上考虑其与 ABC 方法中已有的体系结构静态结构检查工作的结合,并在此基础上进一步开展网构软件体系结构建模与验证及其方法学的研究.

References:

[1] Yang FQ, Mei H, Lü J, Jin Z. Some discussion on the development of software technology. Acta Electronica Sinica, 2002,30(12A): 1901−1906 (in Chinese with English abstract).

[2] Mei H, Huang G, Zhao HY, Jiao WP. A software architecture centric engineering approach for Internetware. Science in China (Series F), 2006,49(6):702−730.

[3] Holzmann GJ. The Spin Model Checker Primer and Reference Manual. Boston: Addison-Wesley Publishing Company, 2003.

[4] Clarke EM, Grumberg O, Jha S, Lu Y, Veith H. Counterexample-Guided abstraction refinement. In: Goos G, Hartmanis J, Leeuwen J, eds. Proc. of the 12th Int'l Conf. on Computer Aided Verification (CAV 2000). LNCS 1855, Heidelberg: Springer-Verlag, 2000. 154−169.

[5] Omg U. UML 2.0 superstructure spec. Object Management Group, 2003.

[6] Zhou L. The design and implementation of a model checker for the description of system behavior in software architecture [MS. Thesis]. Beijing: Peking University, 2007 (in Chinese with English abstract).

[7] Cengarle MV, Knapp A. UML 2.0 interactions: Semantics and refinement. In: Jürjens J, Fernandez EB, France R, Rumpe B, eds. Proc. of the 3rd Int'l Workshop Critical Systems Development with UML (CSDUML 2004). 2004. 85−99.

[8] Ball T, Cook B, Levin V, Rajamani SK. SLAM and static driver verifier: Technology transfer of formal methods inside microsoft. In: Goos G, Hartmanis J, Leeuwen J, eds. Proc. of the 4th Int'l Conf. on Integrated Formal Methods, IFM 2004. Heidelberg: Springer-Verlag, 2004. 1–20.

[9] Cimatti A, Clarke EM, Giunchiglia E, Giunchiglia F, Pistore M, Roveri M, Sebastiani R, Tacchella A. Nusmv 2: An opensource tool for symbolic model checking. In: Goos G, Hartmanis J, Leeuwen J, eds. Proc. of the 14th Int'l Conf. on Computer Aided Verification (CAV 2002). LNCS 2404, Heidelberg: Springer-Verlag, 2002. 359–364.

[10] Giacobazzi R, Quintarelli E. Incompleteness, counterexamples, and refinements in abstract model-checking. In: Goos G, Hartmanis J, Leeuwen J, eds. Proc. of the SAS 2001: Static Analysis. LNCS 2126, Heidelberg: Springer-Verlag, 2001. 356–373.

[11] O'Reilly T. What is Web 2.0. 2005. http://www.oreillynet.com/pub/a/oreilly/tim/news/2005/09/30/what-is-web-20.html

[12] Allen RJ. A formal approach to software architecture [Ph.D. Thesis]. Pittsburgh: Carnegie Mellon University, 1997.

[13] Luckham DC, Kenney JJ, Augustin LM, Vera J, Bryan D, Mann W. Specification and analysis of system architecture using Rapide. IEEE Trans. on Software Engineering, 1997,21(4):336–354.

[14] Magee J, Kramer J. Dynamic structure in software architectures. In: Kaiser GE, ed. Proc. of the ACM SIGSOFT'96: The 4th Symp., Foundations of Software Engineering (FSE4). New York: ACM Press, 1996.

[15] Magee J, Kramer J. Concurrency: State Models & Java Programs. New York: John Wiley & Sons, Inc., 2000.

附中文参考文献:

[1] 杨芙清,梅宏,吕建,金芝.浅论软件技术发展.电子学报,2002,30(12A):1901–1906.

[6] 周立.一种软件体系结构中系统行为描述的模型检查工具的设计与实现[硕士学位论文].北京:北京大学,2007.

周立(1982－),男,广东广州人,硕士,主要研究领域为模型检查,软件体系结构.

孙艳春(1970－),女,博士,副教授,CCF 高级会员,主要研究领域为软件工程,软件复用及构件技术,计算机支持的协同工作.

陈湘萍(1981－),女,博士生,主要研究领域为软件体系结构.

梅宏(1963－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件工程,软件复用和软件构件技术,分布对象技术.

黄罡(1975－),男,博士,副教授,CCF 会员,主要研究领域为软件工程,分布式计算,中间件.