一种网构软件体系结构中的纵横验证机制 *

陈 暄[1], 高 俊[1], 李长云[2]
(1. 浙江工业职业技术学院, 浙江 绍兴 312000; 2. 湖南工业大学 计算机与通信学院, 湖南 株洲 412008)

摘 要: 如何在开放、动态、复杂的 Internet 环境下开发网构软件是软件技术领域一个挑战性课题。从网构软件整个生命周期入手,对网构软件的形式化模型,在简单介绍抽象状态机(ASM)的基础理论之后,刻画了网构软件的构件模型,并对构件模型进行了基于 ASM 的形式化描述,在此基础上,将粗粒度抽象构件的精化问题转换为求解构件组合方案的问题,并在体系结构元层,提出一种双向验证方法对不同抽象程度的组合方案进行横向和纵向的验证,以保证目标系统的正确性和求精过程的正确性。以上形式化建模和双向验证方法尽可能地避免和消除了软件设计早期的错误。通过系统实验验证可以看出,该方法对网构软件的开发具有一定指导意义。

关键词: 形式化模型; 形式化验证; 软件体系结构; 抽象状态机
中图分类号: TP311 **文献标志码**: A **文章编号**: 1001-3695(2012)02-0601-05
doi:10.3969/j.issn.1001-3695.2012.02.053

Approach of vertical and horizontal verification in internetware architecture

CHEN Xuan[1], GAO Jun[1], LI Chang-yun[2]
(1. *Zhejiang Industry Polytechnic College*, *Shaoxing Zhejiang* 312000, *China*; 2. *School of Computer & Communication*, *Hunan University of Technology*, *Zhuzhou Hunan* 412008, *China*)

Abstract: It is a topic of challenge for how to develop internetware in an open, dynamic, complex Internet environment. Formal model of internetware, refinement theory, and formal verification are researched from the life cycle of the internetware. First of all introduced basic theory of ASM, and then proposed the model of component that oriented internetware, and formally described the model of component with abstract state machine. On that basis, translated the problem of coarse-grained abstract component mapping to entity component into the refinement of connector. It proposed a bidirectional verification approach of composition scheme, which made the final system correct, in meta level of architecture. On the whole, the research mentioned above can help to find and fix the errors at as early stage as possible. It is the right way to develop internetware.

Key words: formal model; formal verification; software architecture; abstract state machine(ASM)

0 引言

Internet 的开放性和动态性以及用户个性化要求,使得软件出现一种新的形态——网构软件。目前针对网构软件开发的研究已经广泛展开。与传统软件开发不同的是,网构软件开发不再是一次成型式,给定用户需求后,通过组装 Internet 中已有的软件实体来完成[1]。组装过程实际是理解用户需求,将抽象的粗粒度服务逐层分解为多个细粒度服务的组合,直到每个服务对应到实体服务为止。因此亟需一套贯穿整个软件生命周期的形式化方法来指导网构软件的开发[2]。

网构软件的开发需要从需求逐步求精,直到每个细化的服务映射到一个实体构件。从最终软件形态来看,实体构件间通过协作组装在一起,但各构件均是异构的实体,这时需要一种充当协作功能的连接件将各异构实体连接起来,因而体系结构的求精可看做连接件的求精。构造一个可信的网构软件,首先应保证抽象构件映射到实体构件和连接件的过程是正确的,同时在软件体系结构的各层组合方案是无死锁的和可行的。抽象状态机(abstract state machine, ASM)能够有效准确地刻画用户需求,并且能够进行理论分析,在系统实现之前通过模拟执行,验证软件各目标和属性的正确性。

本文在简单介绍 ASM 后,介绍了基于 ASM 的构件模型,对于网构软件体系结构的各抽象层的精化过程,将问题转换为完成协作的连接件求精,详述了如何由连接件求精得到组合方案,定义了验证组合方案正确性的规则,最后在正确性验证方面,提出了横向和纵向的软件形式化正确性验证方法,并说明了验证的具体实施过程。

1 相关工作

通用的形式化方法对软件体系结构规格进行说明,近年来出现了许多的研究工作。目前这方面的工作大致分为四类[3]:基于模型的方法有 VDM,VDM ++ 是 VDM 结合面向对象概念的扩展,Z 和面向对象方法的结合,典型的有 Object-Z、Z ++;基于逻辑的方法有 Wolper 等人的基于模态逻辑的方法;Heljanko 等人的基于计算树逻辑(computation tree logic, CTL)

收稿日期: 2011-08-16; **修回日期**: 2011-09-23 **基金项目**: 湖南省自然科学基金资助项目(09JJ6087);中国博士后科学基金资助项目(20080440216);浙江省教育技术研究规划课题(JB036);绍兴市教育科学规划课题(SGJ11061)

作者简介:陈暄(1978-),男,江西南昌人,讲师,硕士,CCF 会员,主要研究方向为可信计算(langyi 0512@163.com);高俊(1984-),男,讲师,硕士,主要研究方向为软件自适应;李长云(1971-),男,教授,博士,主要研究方向为可信软件、软件自动化.

的方法;国内唐稚松教授等人的基于时序逻辑 XYZ/E 的方法等。Fisher 等人定义了一种时序信念逻辑 TBL 用于分布式多代理系统的描述与验证;基于进程演算的方法有通信顺序进程(communication sequential processes,CSP)、通信系统演算(calculus of communicating systems,CCS)、π 演算等;专门的体系结构描述语言方面,国外具有一定影响力的 ADLs 有 Aesop、ACME、Rapide、C2、Darwin、SADL、Unicon 和 Wright,目前在 ADL 方面还没有形成一致的观点,国内研究方向也提出几种较有代表性的描述语言,如支持面向构件软件开发方法的 ABC/ADL、基于框架和角色模型体系结构规约语言 FRADL、基于层次消息总线的 JB/ADL、基于时序逻辑的 XYZ/ADL、基于主动连接件的体系结构描述语 Tracer 等。

运用 ASM 对软件进行形式化验证的研究目前国内外也有不少,具有典型代表的有文献[4]提出一种扩展的 ASM 执行引擎,具有良定义接口,可嵌入到其他开发工具对软件验证;文献[5]将模型求精视做由抽象系统解构和实例化为具体系统;文献[6]提出带时间轴的抽象状态机,国内上海交通大学,中国科学院的研究等均有涉及到,文献[7]对动态重构软件重构前后运用 ASM 的规则检验其一致性,文献[5]提出运用 ASM 对网构软件进行形式化描述,提出基础构件模型,给出了基于 OSGI 平台的精化案例,但是它们主要针对运行时系统动态演化前后的一致性,本文将软件需求转换到具体系统过程视做软件体系结构中连接件的求精,研究的对象是验证构件组合方案的正确性,明确提出了双向验证形式和验证规则,对于软件设计和开发具有一定借鉴意义。

2 抽象状态机简介

2.1 ASM 的发展、特点及应用

ASM 是 Dijkstra 抽象机与 Tarski 结构(structure)描述抽象状态两个概念的结合。ASM 又叫演化代数(evolving algebras),是可运行的形式化描述语言。ASM 模型是 Gurevich 提出的一种具有严格数学基础的、用于系统建模和分析的形式化方法。在 ASM 模型中,状态用泛代数中的结构来表示,即状态是由一些基本元素的集合和作用在该集合上的函数和关系构成,状态转换通过转换规则进行[8]。

抽象状态机具有以下特点:

a)具有简单性和通用性。它通过简易伪代码的方式进行表达,能够有效地运用到各种抽象度和复杂度分布式系统。

b)表达能力强,能够清晰、准确地描述问题空间在不同抽象层次的模型。

c)特别适合动态系统的建模,能表示模型中元素的变更,并且可以执行,用于确定规约的正确性。

鉴于 ASM 具有上述特点,它被广泛应用于各种系统的形式化建模和模型分析[9]。在复杂系统的分析和设计中,基于 ASM 的系统构造方法是一种具有良好数学基础的方法。借助 ASM,目标系统的非形式化需求可以逐步细化,并能够转换为具有更严格形式的基础模型。本文将采用该方法建立网构软件的形式化抽象模型,利用 ASM 的可抽象执行机制验证求精过程和目标系统的正确性,以此指导网构软件的开发。

2.2 ASM 的定义和基本规则

下面是 ASM 的一些重要定义和基本规则[10]:

定义 1 基调。ASM 定义在词汇表 $\mathscr{R}$ 上,其中 $\mathscr{R}$ 为一系列动态和静态的函数名的集合。每个函数名都有属于自己的变元数量,其中 0 元静态函数为常量,0 元动态函数为变量。一个 ASM 由四部分组成:词汇表、满足词汇表的初始状态集合、触发规则集合、唯一的 0 元规则符号。

定义 2 状态。在 $\mathscr{R}$ 中,一个状态 ∞ 包括一个非空集 κ,即 ∞ 的超低集,以及对 Ψ 中函数名的解释。如果 f 是 $\mathscr{R}$ 中的一个 n 元(nArg)函数名,则它的解释 f^{∞} 是一个从 κ_n 到 κ 的函数;如果 l 是 $\mathscr{R}$ 中的常量,则它的解释 l^{∞} 是 κ 中的一个元素。状态 ∞ 的超底集 κ 用 $|\infty|$ 表示。状态的超底集也被称为状态的基础集。状态中的元素就是其超底集中的元素。

定义 3 位置。状态 ∞ 的位置是个偶对 $f^{\infty}(a_1,\cdots,a_n)$,其中 $a_1,\cdots,a_n$ 是 $|\infty|$ 的元素,值 $f^{\infty}(a_1,\cdots,a_n)$ 为 ∞ 中位置的内容。

定义 4 更新。状态 ∞ 的更新是一个偶对 $u=w$,更新的含义为在 ∞ 中位置 l 的内容变换为值 v。如果两个更新引用同一个位置,则更新冲突。

定义 5 相容更新。如果更新集 U 不包含冲突更新,则称它是相容的,也就是对任何位置 l 和元素 v,都存在 $(l,v)\in U$ 和 $(l,w)\in U$,则 $u=w$。

抽象状态机的各种状态之间的转换通过如下状态转移规则描述,基本规则如表 1 所示。

表 1 ASM 的状态转移规则

规则名称	规则表示	规则含义
Conditional 规则	if condition then P else Q	Condition 是一阶谓词表达式,表示当条件为真的情况下,执行 P,否则执行 Q
Update 规则	$f(t_1,\cdots,t_n):t$	抽象状态 s 的函数 f 在参数(term)为 $t_1,\cdots,t_n$ 时的值更新为 t
Choose 规则	choose x with φ do P	选择一个满足 φ 的 x,然后执行 P
Forall 规则	forall x with φ do P	对于所有满足 φ 的 x,并行地执行 P
Let 规则	let $x:=t$ in P	把 t 的值赋予 x,然后执行 P
Sequence 规则	P seq Q	顺序地执行 P 和 Q,先 P 后 Q
Block 规则	P par Q $a\in \mathrm{Res}(\infty)\backslash\mathrm{ran}(\zeta)$	并行地执行 P 和 Q
Call 规则	$r(t_1,\cdots,t_n)$	调用转移规则 r,其参数为 $t_1,\cdots,t_n$

3 构件模型及组装

3.1 构件模型

在软件工程领域,构件的提出主要是为提高软件的生产效率,通过多个构件组装成目标软件系统,由此构件之间的兼容性、构件之间行为的安全性和可达性等问题暴露了出来。形式化方法可以辅助软件设计、分析和验证,从而提高软件正确性和可靠性。构件是一组相关服务的封装,是一个黑箱实体,这些服务只能通过构件的输入/输出接口访问。构件模型描述如图 1 所示。

所有构件集合 universe:COMPONENT,universe 表示元素的集合。本文统一将构件定义为一个六元组 COMPONENT(NAME, EXPORTS, IMPORTS, SPEC, STATE)。

1)名字 Name:COMPONENT→STRING。

2)构件的输出接口和输入接口用 Exports 和 Imports 函数表示,Exports: COMPONENT → powerset(EXPSERVICE);Imports: COMPONENT→powerset(IMPSERVICE)。例如构件 component 的输出服务 s_{exp1}、s_{exp2} 和输入服务 s_{imp1}、s_{imp2},输出接口函数和输入接口函数分别为 Export(comp) = $\{s_{\mathrm{exp1}},s_{\mathrm{exp2}}\}$,Import

(comp) = $\{s_{imp1}, s_{imp2}\}$。

3)输入/输出服务规约(universe:SPEC),构件的输出服务和输入服务满足特定的规约。它又分为输入数据约束(ImportConstraint)和输出数据结构(ExportStructure)。例如 A 构件调用 B 构件,A 构件的输出结构应满足 B 构件的输入约束,它们分别定义为 ImportConstraint: IMPSERVICE → EXPCONSTRAINT,ExportStructure:EXPSERVICE→IMPSTRUCTURE。

4)状态(STATE)State:COMPONENT→STATE。状态有四种:构件可被调用 free、正被调用 busy、不可用 failur 和被占用 block 状态。

网构软件由部署在 Internet 上的不同构件组装而成,而网构软件的组成元素构件,它们的接口并不互相匹配,因此需要连接件能以协调的模式将构件按照一定的序列组装,从而形成抽象服务的功能。分布式软件的任何服务均可认为是一种计算,因而连接件也是一种特殊的构件。连接件模型如图 2 所示。

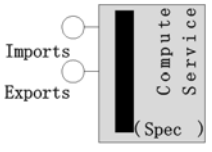


图1　构件模型

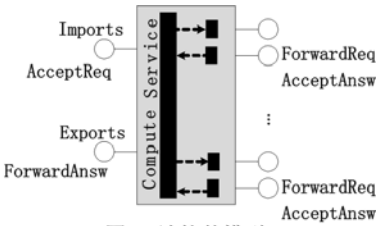


图2　连接件模型

对连接件的定义,在构件的基础上增加了行为(universe:BEHAVIOR),Behavior: COMPONENT→BEHAVIOR。连接件行为在服务提供者和服务调用者两个视图下分成四个动作:BEHAVIOR = {AcceptReq, ForwardAnsw, ForwardReq, AcceptAnsw},它们表述如下:

a)AcceptReq 表示从用户或客户构件接受服务调用请求;

b)ForwardAnsw 表示响应用户或客户构件的业务请求;

c)ForwardReq 表示转发分解后的子服务调用请求;

d)AcceptAnsw 表示接受子服务的处理结果。

上述四个行为其实是针对构件间交互信息流向的抽象集合 In 和 Out。

3.2　构件及其行为的 ASM 表示

基于 ASM 的形式化方法不仅可以为软件体系结构提供准确的表示方法和工程化的指导规则,还能够分析和推理系统的演化,有助于辅助开发网构软件。从上述描述构件和连接件模型的角度看,构件由输入/输出接口函数模块和服务处理模块构成。本文用高抽象 ASM 表示构件,用三个子状态机(sub_ASM)表示三个子模块。连接件在构件基础上增加两个行为子状态机,用五个 sub_ASM 表示。连接件是一种特殊的构件,本文统称为构件。构件和连接件的定义如下:

Component: = choose M with {AccepReq_ASM, ForwardAnsw_ASM, Service_ASM} do M

Connector: = {AccepReq_ASM, ForwardAnsw_ASM, Service_ASM, AcceptAnsw_ASM, ForwardReq_ASM} do M

构件的输入和输出动作用 accepter 和 sender 状态机表示。对于传输的信息,无论本文所研究的构件作为调用者还是被调用者,均抽象为 In_Message 和 Out_Message。对于 VirtualService 状态机来说,外界服务请求用 ResultSet 表示,而网构软件的高抽象层次的构件往往由多个子构件组成,因此一个服务请求(RequestObject)可细化为子服务请求序列 Sub_Seq_Request。连接件虚计算服务(VirtualService)的功能是通过求解函数 GetSubRequest 求解子服务请求序列,这个求解过程可以嵌套。连接件的服务请求需要两个谓词:NeededAck 是否需要确认,isBlocking 数据是否阻塞;连接件的服务接受需要四个谓词:isReadyToAccept、isNeedBuffered、NeededAck 和 isDiscard。构件的三个 Sub_ASM 表示方法如下:

a)cceptReq 抽象状态机

AcceptReq_ASM(in_Request_Message, RequestObject, r_q) = if AcceptedReq(in_Request_Message) then if isReadyToAccpt(in_Request_Message) then NewReq(in_Request_Message, RequestObject) if NeededAck(in_Request_Message) then Send(Ack(in_Request_Message)) else if isDiscarded(in_Request_Message) then DisCard(in_Request_Message) else Buffer(in_Request_Message, r_q)

b)ForwardAnsw 抽象机

ForwardAnsw_ASM(out_Answer_Message, ForwAnswToComPlat) = if ForAnswToComPlat(out_Answer_Message) then Send(out_Answer_Message) if NeededAck(out_Answer_Message) then SetWaitCondition(out_Answer_Message) if isBlocking(out_Answer_Message) then status: = blocked(out_Answer_Message)

c)虚计算服务的抽象状态机 VirtualService

VirtualService(Request) = if status(Request) = started then GetRequestObject(Request) status(Request): = Computing if status(Request) = Computing then PrepareResult(Request status(Request) = Deliver

ForwardReq 和 AcceptAnsw 的行为规则与 AcceptReq 和 ForwardAnsw 类似,不再做详细介绍。

3.3　构件组装方案求精

网构软件开发过程中的一个关键问题是如何求解出构件的组装方案。从上往下:在软件体系结构的抽象层,粗粒度构件所完成的服务,往往需要划分成多个子任务,子任务又可再分解,最终对应到实体构件;从下往上:实体构件按照一定的规则向外提供服务,从而构成粗粒度构件。由哪些构件按照怎样的规则组合在一起呢?本文通过连接件的求精能很好地回答。连接件的虚计算模块的求精如图 3 所示。

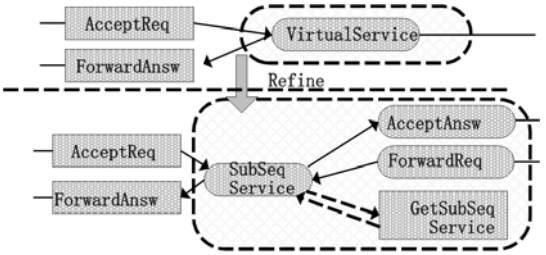


图3　连接件求精示例

连接件和构件的区别在于连接件负责协调各服务请求,通过求精计算出子构件组合方案,构件中 VirtualService 完成业务逻辑。连接件中的 VirtualService 主要完成各子构件协调,该模块求精过程如下:

VirtualService_ASM(Request) = if status(Request) = started then GetSubRequestObject(Request) GetSubRequestObject(req) = let r = InitCureentSubReq(GetSubseqRequst(req)) subReq: = r subSubSeq(r): = InitCrrentSubSubReqSeq(r) status(req): = subComputeService GetNextSubRequestObject(req) = let x = GetNextSubSeq(GetSubSeqRequest(req), subReq, AnswerSet(req)) if x = = null then FinishedSubService: = true else subReq: = x subSubSeq(x): = GetSubSubSeq(x, GetAnswer(req)) status(req): = subSubVirtualService

通过任务分解,连接件可逐步精化到子构件或者子连接件,事实上,连接件本身就是一个组装方案。

4　双向验证

4.1　双向验证内容和规则

通过连接件求精得出组装方案,但如何保证方案的正确

性？求精的过程由多个步骤组成，最后得到一个由不同抽象程度的组合方案的叠加体（图 4）。如果层内组装方案正确和层间求精步骤均无误，则可选底层组装方案构造软件，而事实上这两个步骤均可能出错。因此需要横向和纵向的验证。双向验证具体内容包括：a）在求精过程中，每个抽象层次是上一个相邻层次的精化，验证下层是否保留上层系统属性和规约；b）非形式化的需求描述往往存在需求模糊、终止条件不确定等问题，通过可执行规约在体系结构元层模拟目标系统执行，从而使问题明确，验证各抽象层系统属性和目标的正确性。双向验证的实现需要辅助工具，同一层的模拟执行检验：抽象状态机的设计原则是通过构建系统的设计模型，在抽象层模拟系统执行，CoreASM[4] 环境可以执行 ASM 描述的系统模型各种属性；不同层的精化正确性方面：通过辅助工具 KIV[7] 的模态推演，从而验证系统精化前后的各属性和规约的正确性。

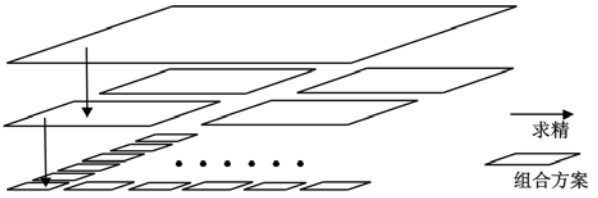


图4 不同抽象程度的组合方案叠加体

对于组装方案的有效性验证，暂先讨论四个规定：构件规约相同、连接匹配、构件状态可用和底层为构件实体。两个构件规约相同的条件：

$SPEC(Comp) = SPEC(Comp')$ iff

1. $ImportConstrait(Comp) = ImportConstrait(Comp')$
2. $ExportStructure(Comp) = ExportStructure(Comp')$
3. $Imp(Comp) = Imp(Comp'), Exp(Comp) = Exp(Comp')$
4. $|Import_{Comp}| = |Import_{Comp'}|, |Export_{Comp}| = |Export_{Comp'}|$
5. $Type(Comp[i]^{exp}_{imp}) = Type(Comp'[i]^{exp}_{imp})$

规约包括输入约束和输出结构，两者均需相同，$|Import_{Comp}|$表示参数个数，$TypeArg(Comp[i])$（其中，$i = 1, 2, \cdots, n$）表示参数类型。连接匹配必须满足：

$Connection(Comp) = Connection(Comp')$ iff

1. $State(Comp) \&\& State(Comp') = free$
2. $Imp(Comp) = Exp(Comp'), Exp(Comp) = Imp(Comp')$
3. $Import_{Comp} = Export_{Comp'}, Import_{Comp} = Export_{Comp'}$
4. $ArgType(Comp[i]^{exp}_{imp}) = ArgType(Comp'[i]^{imp}_{exp})$

对某抽象层的系统进行有效性验证，需要定义构件可用状态：每个构件只能与一个连接件连接，这样可避免响应错误。规则如下：

Forall ImpComponent, ExpComponent

if $Connect(Imp(Comp), Exp(Comp')) \neq$ undef then

$|Comp_{Imp}| = 1 \;\&\&\; |Comp_{Exp}| = 1$

最终的求精方案节点都应映射到实体构件，判断是否为实体构件的规则：

Forall comp with $\in$ COMPONENT do

if($\exists$ comp $\in$ COMPONENT):

let comp(SubRequest) := null

let GetNextSubRequest($Request_{comp}$) := null

let State(comp) := free

ArgImport $\in$ Import(comp)

至此，构件的求精方法和求精后组装方案的验证规则俱已说明，接下来可通过辅助验证工具 CoreASM 和 KIV 对体系结构进行横向同层和纵向层间的验证。

4.2 双向验证实现

纵向验证方面：KIV 是用来验证软件正确性的辅助工具，在新版 KIV 中提供了支持 ASM 描述和分析的 ASM Specification，尤其是 KIV 中的模态推演规则，给 ASM 的精化提供支持。在 KIV 中证明抽象状态机的精化正确性需要两个步骤：首先行为的 ASM 描述和约束谓词；其次装载精化理论规约 Proof 检验 ASM，得到一个谓词结构，再根据谓词结构判断精化是否正确。相关谓词有 IN\OR\IR\INV，这四个谓词描述的是两个状态机之间的关系。以计算服务抽象状态机 VirtualServiceASM 为例，检验其请求状态信息是否完整。首先给出 VirtualService 精化前后状态机 VirtualServiceASM 和 R_VirtualServiceASM，再根据 Proof 进行验证。KIV 验证工具运行界面如图 5 所示。

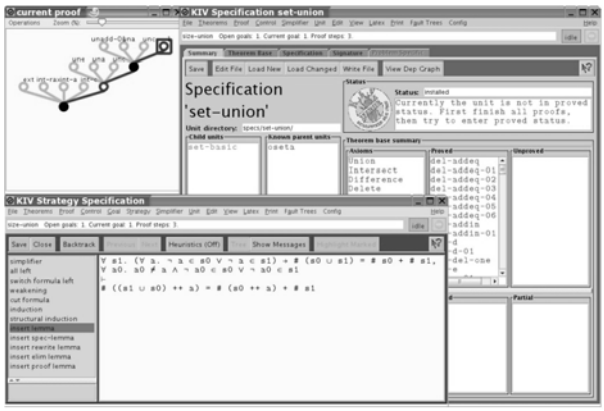
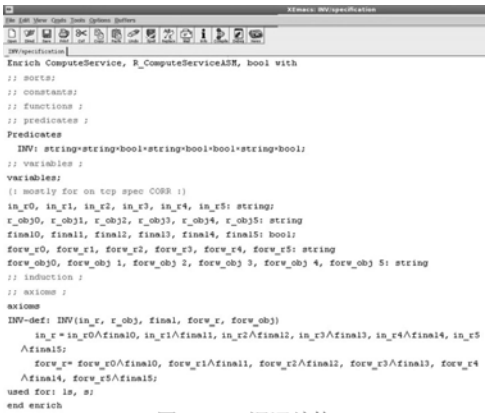


图5 KIV验证工具

INV 谓词包含了前后状态机的变量是否合法。具体 INV 谓词的结构具有如下一般形式：

INV(var1, …, varn, r_var1, …, r_varn) legal(r_var1, …, r_varn) and var1 = f1(r_var1, …, r_varn) and var2 = f2(r_var1, …, r_varn)。上例得到的 INV 谓词结构如图 6 所示。



```
Enrich ComputeService, R_ComputeServiceASM, bool with
;; sorts:
;; constants:
;; functions :
;; predicates :
Predicates
  INV: string*string*bool*string*bool*bool*string*bool;
;; variables :
variables:
(: mostly for on tcp spec CORR :)
in_r0, in_r1, in_r2, in_r3, in_r4, in_r5: string;
r_obj0, r_obj1, r_obj2, r_obj3, r_obj4, r_obj5: string
final0, final1, final2, final3, final4, final5: bool;
forw_r0, forw_r1, forw_r2, forw_r3, forw_r4, forw_r5: string
forw_obj0, forw_obj 1, forw_obj 2, forw_obj 3, forw_obj 4, forw_obj 5: string
;; induction :
;; axiome :
axiome
INV-def: INV(in_r, r_obj, final, forw_r, forw_obj)
     in_r = in_r0∧final0, in_r1∧final1, in_r2∧final2, in_r3∧final3, in_r4∧final4, in_r5
  ∧final5;
     forw_r= forw_r0∧final0, forw_r1∧final1, forw_r2∧final2, forw_r3∧final3, forw_r4
  ∧final4, forw_r5∧final5;
used for: is, s;
end enrich
```

图6 INV谓词结构

横向验证方面：通过 ASM 引擎模拟执行基于 ASM 的形式化系统模型，从而验证本层相应模型的各种属性及目标。CoreASM 环境提供了可视化的图形界面，同时也是一个开放的执行环境，能进行扩展，并可作为插件嵌入到 Eclipse 中。CoreASM 的执行过程以单步执行为例，步骤如下：a）装载需要验证的 ASM；b）解释器执行，通过产生更新集的方式对规约进行计数，直至没有下一个待执行的子 ASM；c）检查更新规则是否有冲突。CoreASM 的规约程序段如下所示：

```
CoreASM Component
use StandardPlugins
enum Status = {free, busy, failur, block}
enum BOOL = {true,false}
function AcceptedReq:STRING -> BOOLEAN
function isReadyToAccept:STRING -> BOOLEAN
```

```
function NeededAck:STRING -〉BOOLEAN
universe Components = {ACCEPTREQ,FORWARDANSW,
  FORWARDREQ,ACCEPTANSW,COMPUTESERVICE}
init InitRule
rule InitRule = par
program(ACCEPTREQ) := @ ACCEPTREQ_ASM
program(FORWARDANSW) := @ FORWARDANSW_ASM
rule ACCEPTREQ_ASM(in_Request_Message,
  RequestObject,
if AcceptedReq(in_Request_Message) then
if isReadyToAccept(in_Request_Message) then
seq par
NewReq(in_Request_Message,RequestObject)
if NeededAck(in_Request_Message) then
seq par
…
```

5　结束语

本文从基于 ASM 理论构建网构软件系统模型,进而对其进行验证等方面展开研究。首先提出了网构软件的构件模型,将软件需求描述逐渐向目标系统精化的过程视同构件组装方案的精化,意即连接件的求精;其次,建立基于 ASM 软件形式化模型,在得到抽象程度不同的构件组装方案叠加体后,利用辅助验证工具验证精化过程和精化方案的正确性,一方面用 KIV 工具验证精化前后的一致性,另一方面利用 CoreASM 对组装方案验证其安全性和可达性,从而保证网构软件各个生命周期内目标系统正确性;最后给出了求精和验证的具体实现。基于 ASM 的软件形式化建模和双向验证分析方法具有可行性,对在早期发现软件设计错误,降低软件生产成本有指导意义。

参考文献:

[1] 高俊,沈才梁,陈暄.一种面向服务体系结构的服务组合方案求解方法[J].计算机应用研究,2011,28(11):4184-4187.

[2] 周立,陈湘萍,黄罡,等.支持协商的网构软件体系结构行为建模与验证[J].软件学报,2008,19(5):1099-1112.

[3] 高俊,沈才梁,郑美芳,等. 一种面向自适应软件系统的体系结构描述语言[J].计算机应用研究,2010,27(5):1796-1801.

[4] FARAHBOD R, GERVASI V, GLÄSSER U, *et al*. CoreASM: an extensible ASM execution engine[J]. Fundamenta Information, 2007,77(1-2):71-103.

[5] 伍建焜.网构软件系统构建的形式化分析研究[D].上海:上海交通大学,2010.

[6] TEICH J, WEPER R, FISCHER D, *et al*. A joint architecture compiler design environment for ASI[C]//Proc of Conference on Compilers, Architectures and Synthesis for Embedded Systems. San Jose, CA: ACM Press,2008:1126-1133.

[7] 王源.动态重构网格应用系统的鲁棒一致性研究[D].北京:中国科学院研究生院,2004.

[8] BÖRGER E. Abstract state machines: a method for high-level system designed analysis[M]. Berlin: Springer-Verlag, 2003.

[9] GLÄSSER U, FARAHBOD R, VAJIHOLLAHI M. An abstract machine architecture for Web service based business process management [J]. International Journal on Business Process Integration and Management,2006,1(4):279-291.

[10] GARGANTINI A, RICCOBENE E, SCANDURRA P. A meta model based language and a simulation engine for abstract state machines [J]. Journal Universal Computer Science,2008,14(12):1949-1983.

[11] SCHELLHORN G. Completeness of ASM refinement[J]. Electronic Notes in Theoretical Computer Science,2008,214(7):25-49.

[12] BÖRGER E, Del CASTILLO G. A formal method for provably correct composition of a real-life processor out of basic components[C]//Proc of the 1st IEEE International Conference on Engineering of Complex Computer Systems. New York: B Werner,2005:145-148.

(上接第 600 页)理论的正确性,并且体现了其在评估过程中的优越性。但本文所提出的信任云合成算法仍需要进一步扩展,使得可以针对更加复杂系统结构进行可信性评估。

参考文献:

[1] WANG Wen-li, PAN Dai, CHEN M H. Architecture-based software reliability modeling[J]. Syst Software,2006,79(1):132-146.

[2] WANG Jun, CHEN Wei-ru, LIU Jun. A modeling of software architecture reliability[C]//Proc of IFIP International Conference on Network and Parallel Computing Workshops. Washington DC: IEEE Computer Society,2007:983-986.

[3] DOLBEC J, SHEPARD T. A component based software reliability model[C]//Proc of Conference of the Centre for Advanced Studies on Collaborative Research. Toronto:IBM Canada, 1995:19.

[4] MASON D. Probabilistic analysis for component reliability composition[C]//Proc of the 5th ICSE Workshop Component-based Software Engineering. New York: ACM Press, 2002.

[5] GUTJAHR W J. Software dependability evaluation based on Markov usage models[J]. Performance Evaluation, 2000, 40(4): 199-222.

[6] YU Ben-hai, WANG Qing, YANG Ye. Research on a software trustworthy measure model[C]//Proc of the 2nd International Conference on Networks Security, Wireless Communications and Trusted Computing. Washington DC:IEEE Computer Society, 2010:518-521.

[7] HOU Hong, SONG Qin-bao, YANG Jing, *et al*. The research of BPM software trustworthy evaluation model[C]//Proc of the 1st International Workshop on Education Technology and Computer Science. Washington DC: IEEE Computer Society,2009:815-823.

[8] GRASSI V, PATELLA S. Reliability prediction for service-oriented computing environments[J]. IEEE Internet Computing,2006,10(3):43-49.

[9] 黄茗云. 贝叶斯网络在软件可信性评估指标体系中的应用[D].济南:山东轻工业学院,2008.

[10] 路峰,吴慧中.基于云模型的信任评估研究[J].中国工程科学,2008,10(10):84-90.

[11] WANG Shou-xin, ZHANG Li, MA Na, *et al*. An evaluation approach of subjective trust based on cloud model[J]. Journal of Software Engineering & Applications,2008,1(1): 44-52.

[12] 向楷.软件构件的可信评价及组装方法的研究[D].大连:大连理工大学,2010.

[13] FIELDING, ROY T, TAYLOR, *et al*. Principled design of the modern Web architecture[J]. ACM Trans on Internet Technology,2002,2(2):115-150.

[14] IEEE Std. 982.1 - 1988, IEEE standard dictionary of measures to produce reliable software[S]. Washington DC:IEEE Press, 1988.

[15] ISO/IEC 9126, Information technology-product quality-part1: quality model[S]. Geneva: ISO/IEC, 2001.