软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2017,28(5):1167−1182 [doi: 10.13328/j.cnki.jos.005215]

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

面向动作的上下文感知应用的规约与运行时验证*

李暄松[1,2], 陶先平[2], 吕 建[2], 宋 巍[1,2]

[1](南京理工大学 计算机科学与工程学院,江苏 南京 210094)

[2](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210023)

通信作者: 陶先平, Email: txp@nju.edu.cn

摘 要: 面向动作的上下文感知(activity-oriented context-aware,简称 AOCA)应用组织环境中的资源,为用户动作的顺利进行提供支持.为应对环境和动作相关需求的开放性,这类应用采用轻量级、增量式的开发方法进行开发.相对于在开发阶段描述全局信息的开发方法,AOCA 应用的开发可能由不同开发者在不同时间共同参与,这可能会导致较多的不一致等问题,且难以在开发阶段被发现.围绕使用运行时验证手段提高 AOCA 应用可靠性这一目标展开研究.给出了对于 AOCA 应用运行状态进行形式化规约、对于系统级和应用级性质进行描述的方法.进一步地设计实现了 AOCA 应用监控器.最后,通过案例分析以及性能评估证实了该方法的有效性.

关键词: 普适计算;上下文感知;形式规约;运行时验证

中图法分类号: TP311

中文引用格式: 李暄松,陶先平,吕建,宋巍.面向动作的上下文感知应用的规约与运行时验证.软件学报,2017,28(5):1167−1182. http://www.jos.org.cn/1000-9825/5215.htm

英文引用格式: Li XS, Tao XP, Lü J, Song W. Specification and runtime verification for activity-oriented context-aware applications. Ruan Jian Xue Bao/Journal of Software, 2017,28(5):1167−1182 (in Chinese). http://www.jos.org.cn/1000-9825/5215.htm

Specification and Runtime Verification for Activity-Oriented Context-Aware Applications

LI Xuan-Song[1,2], TAO Xian-Ping[2], LÜ Jian[2], SONG Wei[1,2]

[1](School of Computer Science and Engineering, Nanjing University of Science and Technology, Nanjing 210094, China)

[2](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210023, China)

Abstract: Activity-Oriented context-aware (AOCA) applications organize environment resources to support the smooth performing of user activities. These applications are developed with a light-weight and incremental method in order to deal with the openness of the environment and requirements related to user activities. In contrast to the methods which attempt to deal with information from an overall level, AOCA applications allow different developers take part in the development in different time. However, this method may lead to more problems such as inconsistency. These problems are difficult to be detected in the development phase. This study focuses on using runtime verification to enhance the reliability of AOCA applications. In this paper, a method is first proposed for specifying AOCA application runtime status and describing system-level and application-level properties. Next, an AOCA application monitor is designed and implemented. Moreover, a case study and a performance evaluation are described to demonstrate the usability of this method.

Key words: pervasive computing; context-awareness; formal specification; runtime verification

随着计算机技术以及传感器、移动设备、网络通信、智能控制等相关软硬件技术的发展,计算系统不再采用键盘、鼠标、显示器等传统的显式输入输出手段,而是通过感知用户的需求和环境的状态,自动地控制设

* 基金项目: 国家自然科学基金(61373011, 61202003, 61502225)
Foundation item: National Natural Science Foundation of China (61373011, 61202003, 61502225)
收稿时间: 2016-07-15; 修改时间: 2016-09-25; 采用时间: 2016-12-07; jos 在线出版时间: 2017-01-20
CNKI 网络优先出版: 2017-01-20 16:06:34, http://www.cnki.net/kcms/detail/11.2560.TP.20170120.1606.007.html

备为用户提供服务成为可能.强调计算无处不在而又透明于用户的普适计算[1]这一新型计算模式应运而生.这一计算模式的关键技术之一是上下文感知[2],即系统可以感知用户的状态和环境信息,并根据这些信息调整系统的行为.

普适计算中以智能会议室、智能家居、老人看护等为代表的一类典型应用场景体现出如下特点:用户自主地进行日常的动作,动作背后隐藏着对于环境的需求(如阅读时希望光线较强,睡觉时希望温度适宜、光线较暗),计算系统组织环境中的资源,保障用户动作的顺利进行.我们将这类应用称为面向动作的上下文感知(activity-oriented context-aware,简称 AOCA)应用.由于 AOCA 应用的环境资源开放、动态,需求随丰富的动作类型随时变化、因人而异,这类应用难以利用现有工作中的通用方法(如文献[3,4])在开发阶段一次性、考虑整体地完成开发.为了应对这样的复杂性,已有工作[5]给出了针对 AOCA 应用的轻量级、增量式的开发方法以及轻量级、可插拔的运行支撑技术.其核心思想是,在开发阶段将环境资源的描述和应用对于环境的需求描述分离开,进一步地,在应用部分,以约束的形式,从用户的动作出发分别定义对于环境的需求;在运行时将分离开发的部分融合在一起,用“上下文”这一概念描述和特定动作相关的环境信息.

根据上述方法,AOCA 应用的开发采用关注分离的思想,其环境资源和动作相关的应用需求可能在不同时期由不同开发者增量式地完成开发.由于对环境的刻画和环境中应用需求的表达是分离的,而且应用需求的表达本身也基于动作进行分离,AOCA 应用开发方法并不能保证这些软件制品之间达成完全的一致.比如,用户的烹饪动作所在的位置应当部署有一氧化碳传感器,以免发生泄漏却无法监测而导致的危险.这种性质应当被精确定义并始终被保障.上述的不一致情况可能在多处发生,情形也不一而足,在一些安全攸关的场合下(如老人看护应用)可能带来严重的后果,而这些问题在开发时难以被发现,需要在运行时采用有效手段来解决因为“关注分离”而导致的一致性欠佳问题,进而提高 AOCA 应用的可靠性.

运行时验证(runtime verification)是提高系统可靠性的有效手段[6].该技术把形式化验证技术与系统的运行相结合,其一般方法是从系统需求中产生一些需要监控的性质,设计监控器(monitor)监控系统的运行状态是否满足性质,以发现系统运行中的异常行为,进一步地对系统进行维护[7].

针对 AOCA 应用的可靠性问题,本文提出了这类应用的形式化规约与运行时验证方法.具体而言,以 Ambient Calculus 为形式工具,对 AOCA 应用的运行时状态进行形式化规约,以 Ambient Logic 为逻辑基础,给出了 AOCA 应用在系统级和应用级一致性性质的形式化方法.本文还基于上述运行时验证方法,实现了对于 AOCA 应用运行时状态进行监控的监控器.

本文第 1 节首先介绍 AOCA 应用的相关背景知识与研究用例,然后介绍 Ambient Calculus 和 Ambient Logic 的基本概念.第 2 节概述 AOCA 应用的形式化规约方法.第 3 节给出 AOCA 应用监控器的实现.第 4 节通过案例讨论该形式化方法的有效性,并描述性能评估实验.第 5 节讨论相关工作.最后在第 6 节总结全文并展望下一步的研究方向.

1 背 景

本文以 Ambient Calculus/Ambient Logic 为工具,对 AOCA 应用的运行状态进行规约和验证.本节介绍这一工作的研究基础.首先,概述 AOCA 应用的运行系统;然后,通过这类应用的典型场景讨论其可能存在的可靠性问题;最后,介绍本文工作中使用的形式化工具 Ambient Calculus 和 Ambient Logic.

1.1 AOCA应用及运行

面向动作的上下文感知(AOCA)应用指的是一类可以感知用户动作,进一步地感知动作相关的环境信息,并根据感知到的这些信息反馈环境和用户,以保障动作进行的应用.

以文献[5]给出的方法支持 AOCA 应用的开发与运行,其运行时模型可归纳为图 1 的形式.其中,环境部分描述了环境中的资源以及感知、影响这些资源的能力,该部分可独立开发,可被不同应用共享;动作部分在运行时识别用户动作的类型和位置,通过开发时定义的模式生成当前动作背后的需求(对环境的约束);上下文在运行时为每个特定动作生成,描述与特定动作需求相关的环境,该部分在运行时将上述相对独立的两部分关联起来,

判断环境中的信息能否满足动作相关的约束,在不满足时进行影响环境或者影响用户的操作.

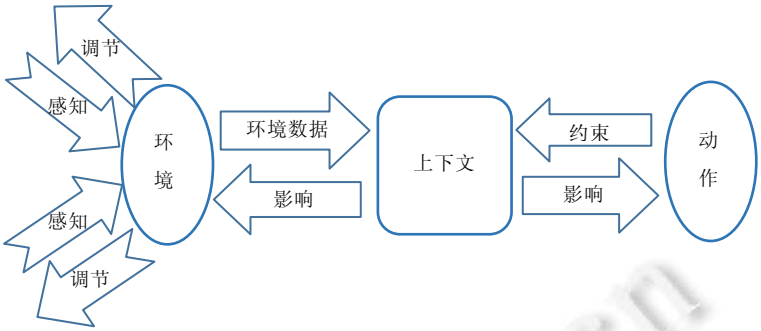


Fig.1 Runtime model of activity-oriented context-aware (AOCA) applications

图 1 AOCA 应用的运行时模型

具体而言,这 3 部分分别描述了以下内容.

- 环境部分:该部分描述了空间的结构(如 812 号房间中有一个卧室、一个客厅);空间中可被获取利用成分(如“卧室的温度”“客厅的光线强度”)以及用户的状态(如“用户 Bob 的心跳速率”),我们称其为环境要素;对于每个要素,还需将它和感知它的设备(如感知光线强度的光线传感器)以及影响它的设备(如可以影响光线强度的灯)关联起来.
- 动作部分:该部分描述了运行时某一时刻用户的动作(如 Bob 正在卧室睡觉);该动作相关的对环境的需求(如 Bob 睡觉的需求包括温度适宜、光线较暗),这样的需求可理解为对环境中部分状态的约束.
- 上下文部分:AOCA 应用中的上下文被理解为与一个特定动作相关的环境要素的集合(如 Bob 在卧室睡觉的上下文包括该卧室的温度、光线强度).上下文在运行时被支撑系统根据动作相关的约束生成,每个上下文被一个主动性的上下文管理器进行管理,可以判断约束是否满足,对环境和用户动作进行反馈.

根据这样的理解,开发人员可被分为环境开发者和应用开发者两类,环境开发者负责描述和维护环境中的资源,对环境要素进行定义,并描述每个要素相关的传感器和执行器;应用开发者配置对于动作类型以及位置进行识别的服务,通过给定各个阶段动作相关约束的方式开发应用对环境的需求.这两部分开发出的软件制品都是轻量级的,开发过程是增量式的,可方便地进行修改、扩展.AOCA 应用的运行系统支持上述方法开发出的软件制品的运行,根据动作识别技术(如文献[8])识别出的动作类型和位置,生成当前动作的上下文,并用上下文管理器这一轻量级的软件实体对上下文进行管理,当动作变化时,上下文将被重新生成.

1.2 AOCA研究用例

老人看护是 AOCA 应用的典型案例.这类应用可以通过部署在智能环境一系列的传感器(如光线传感器、温度传感器)以及获得环境信息的服务(如天气服务)获得物理环境信息以及老人的心跳、血压等身体状况,可通过控制灯、空调等设备对于环境进行影响.享受该服务的老人在这样的环境中自主地进行日常动作,如睡觉、阅读、烧饭等,这些动作在进行时对环境有着不同的要求,如睡觉时希望温度适宜、光线较暗、噪音较小,阅读时希望光线较强,烧饭时希望厨房的温度、一氧化碳浓度不能过高.应用系统的任务即是保障老人的日常动作顺利进行,其一般过程是通过感知老人当前的动作,围绕这一动作的需求组织上下文,判断动作的需求是否被满足,当需求不被满足时,通过驱动设备影响环境或者将环境不满足要求这一消息通知给老人或其家人.

由于 AOCA 应用围绕着不同用户的不同动作进行组织,动作的类型、因人而异的需求、环境中的资源经常产生扩展、变化,体现出较强的开放性,这类应用适合采用增量式的开发方法进行开发[5].这样的开发方法导致了在开发和运行过程中可能会有较多的不一致等问题.例如,这类应用运行时在报告动作类型时必须同时报告动作发生位置,但是由于这两类服务可能由不同开发者在不同时间开发,这一要求有可能无法满足,出现在运行时系统只能获得动作类型而无法获得用户位置的情况;另一方面,具体的应用也可能存在一些需要被始终保

障的性质,如对于老人看护应用而言,要求老人进行烹饪动作的位置中的一氧化碳传感器必须始终有效,以确保老人的安全.

由于 AOCA 应用的开发人员只需关注系统的部分信息、描述轻量级的软件制品,上述不一致问题以及性质无法满足的情况往往难以在开发阶段被发现,通过运行时验证的方法发现系统中的问题、提高可靠性具有研究价值.本文选择了 Ambient Calculus/Ambient Logic 为形式工具开展上述研究.

1.3 Ambient Calculus与Ambient Logic

Ambient Calculus[9]是一个描述进程和设备移动的进程演算.在这个演算中,最重要的概念是 ambient(中文中有时被译为环境,为和 environment 等近义词区分,更清晰地阐述这一概念,本文直接使用英文),它指的是计算发生的一个有界限的位置(a bounded place where computation happens).每个 ambient 存在边界,可被嵌套在其他 ambient 中,可以作为一个整体进行移动.

这个演算中,所有的实体都可被描述为进程(process).本文涉及到的进程可表示如下.

$P,Q::=$	processes
$n[P]$	ambient
$P\|Q$	composition

其中,ambient 是一种特殊的进程,表示为 $n[P]$,其中,n 是 ambient 的名,P 是这个 ambient 之中运行的进程.进程的并行执行由二元操作 $P|Q$ 表示,这个操作符合交换律和结合律.下面的表达式是一个 ambient 之间嵌套的例子:

$$n[P_1|...|P_j|m_1[...]|...|m_k[...]],$$

它表示一个名为 n 的 ambient 中有 j 个名为 $P_1,...,P_j$ 的进程以及 k 个名称为 $m_1,...,m_k$ 的 ambient.

这个演算还提供了一些其他算子,如用!描述进程的克隆(使用该算子会导致相应的检验问题不可判定[10]).该演算还提供了描述进程间操作的算子,如进入 ambient、离开 ambient、进程间消息传递等.限于篇幅和应用需求,这里不再赘述.

Ambient Logic[11]是对于时态逻辑的扩展,用以表达特定 ambient 的相关性质.该逻辑中的公式可以包括以下算子.

- 命题逻辑中的算子,如否定(¬)、析取(∧)、合取(∨).
- 全称(∀)和存在(∃)量词.
- 时态算子:终究(eventually◊)、总是(always□).
- 位置算子:某处(somewhere∇).

从语义上说,满足关系 $P\vDash A$ 指的是进程 P 满足封闭的公式 A(A 没有自由变元).其中,$P\vDash A@L$ 指的是进程 $L[P]$满足 A;$P\vDash L[A]$指的是存在一个进程 P',满足 $P\equiv L[P']\wedge P'\vDash A$;$P\vDash A1|A2$ 指的是存在 P'、P'',满足 $P\equiv P'|P''\wedge P'\vDash A1\wedge P''\vDash A2$;$P\vDash\nabla A$ 指的是 P 进程中的某个位置(可能被嵌套在 ambient 中)存在进程 P',使得 $P'\vDash A$.

为了更好地对时间相关的性质进行描述,我们采用文献[12]中讨论的方法,在算子◊中设置一个时间段,即描述为$\Diamond^{\leqslant k}$,其中,k 既可以是一个时间值(如 30s、10m)也可以是状态转换的次数(如采样 10 次).□等价于¬◊¬,所以也可相应地设置时间段.

本文基于以下理由选择 Ambient Calculus/Ambient Logic 作为 AOCA 应用的形式化规约、验证工具.

- AOCA 应用以位置、人员为依据组织设备,这类系统中广泛存在某个位置部署了一个设备、某个人员持有一个设备、某个人员位于一个位置中这类关系,对象间这样的关系可以比较容易地用 Ambient Calculus 中的 ambient 内运行进程(这些进程也可能是其他 ambient)这样的方式进行描述.
- AOCA 应用中的上下文不是预定义的,而是在运行时基于动作组织,它可被看作一个有界限的位置中的数据,这与 Ambient Calculus 中 ambient 的概念一致.
- 对于用 Ambient Calculus 进行规约的 AOCA 应用运行状态,使用 Ambient Logic 描述需要满足的性质是适合的,因为该工具提供了位置相关的算子.而其他模态逻辑,如时态逻辑中的 LTL(线性时态逻辑)[13]、CTL (树

状分支时态逻辑)[14]仅提供时态算子,不适合描述 ambient 相关的性质.

2 AOCA 应用的运行状态及其形式化

本节给出 AOCA 应用运行状态的形式化规约方法.首先,简单介绍 AOCA 应用运行状态中包含的主要概念;然后,介绍基于 Ambient Calculus 对 AOCA 应用的运行状态进行形式化的方法;进一步地,基于 Ambient Logic 定义应用需要被保障的性质.

2.1 AOCA应用的运行状态

本节介绍 AOCA 运行过程中所涉及到的各部分信息的构成.

1) 环境部分

该部分的基本单元为环境要素(environment attribute),它描述了环境中一个可被获取利用成分(如“812 房间的温度”).一个环境要素可被描述为⟨feature,owner,value⟩三元组形式,其中,feature 是环境特征(如温度 temperature、血压 blood pressure);owner 是所有者,它描述了一个可以拥有环境要素的实体,本文讨论的所有者包括位置(location,如 room812)和人员(person,如 Bob)两大类,位置之间可以嵌套,如一个房间中可以包含卧室、客厅等;value 是这个要素当前的值,当它为空(NULL)时表示要素值尚未被报告.例如,⟨temperature,room812,25⟩表示一个描述 812 房间温度的环境要素.

对于每个环境要素而言,还需描述和它相关的设备.我们将用以提供环境要素信息的传感器(如温度传感器)和服务(如天气服务)的软件封装称为探测器(probe),将用以影响环境的设备(如影响光线强度的灯)的软件封装称为执行器(actuator).这两类设备需要和环境要素关联起来.

AOCA 应用的环境由环境要素集合构成,即$\{attr_1,attr_2,...,attr_n\}$.这样的环境描述了计算系统对于物理世界的感知、影响能力.

2) 用户动作部分

该部分描述了运行时某一时刻用户的动作以及该动作相关的应用需求.

一个动作(activity)可被描述为⟨actor,perLoc,actType⟩三元组形式,其中,actor 描述了动作的行动者(系统中的一个用户,如 Bob、Alice);perLoc 描述了动作发生的位置;actType 描述了动作的类型.行动者和发生位置可以是所有者(owner)的实例(所有者类型分别是人员和位置),在给定一个行动者或者发生位置之后,可获得其拥有的环境要素.

运行时动作对于环境的约束(constraint)由一组⟨feature,predicate⟩形式的断言构成,其中,feature 是环境特征,predicate 是一个可以判断该断言是否被满足的谓词(如大于 10,“>10”).

3) 上下文部分

上下文是一组环境要素的集合,在运行时根据动作相关的约束以及系统感知环境的能力进行生成.如系统识别到⟨Bob,bedroom1,sleeping⟩(Bob 在 bedroom1 这一房间睡觉)这一动作,且该动作相关的约束包括⟨noiseIntensity,<30⟩(噪音小于 30 分贝)和⟨temperature,[23,27]⟩(温度在 23°C~27°C 之间).如果 bedroom1 这一房间存在噪音传感器和温度传感器,则该动作的上下文中应当有⟨noiseIntensity,bedroom1,value⟩,⟨temperature,bedroom1,value⟩这两个环境要素.

2.2 AOCA应用运行状态的形式化

AOCA 应用运行状态的规约见表 1.本节从环境、用户动作、上下文这 3 个部分出发进行具体的探讨.

1) 环境部分

在该部分,我们对环境中的探测器、执行器的部署进行形式化,即按照环境要素的格式描述每个设备相关的特征和所有者.

环境相关的特征被规约为一个保留使用的字符串,用一个集合描述系统可以关注的所有特征.这些特征被用于描述探测器/执行器、约束、环境要素的公式中.

每个位置、人员皆被规约为 Ambient Calculus 中的一个 ambient,位置 ambient 可以嵌套在其他位置中,这些位置构成一个树状层次结构.

每个探测器/执行器也被规约为一个 ambient.在规约中指明了其位置、相关的特征.

2) 用户动作部分

该部分对动作以及动作相关约束中的特征进行形式化.

动作类型被规约为一个保留使用的字符串,用一个集合描述系统可以感知到的所有动作类型.这些动作类型可被用于描述性质的公式中.

动作被形式化为一个 ambient,在规约中指明了其位置、行动者,并以断言的形式给出了相关的约束.

3) 上下文部分

上下文之中包括了与特定动作相关的环境要素,被形式化规约一个 ambient.上下文为动作组织,随动作识别生成,在生成后该 ambient 被嵌套入相应的动作 ambient 中,如,

sleeping1[Bob[]|as1|as2|cxt[cxtData[attr1|attr2]|P]]

where as1⊨asHasFea(temperature),as2⊨asHasFea(noiseIntensity),

attr1⊨attrHasFea(temperature),attr2⊨attrHasFea(noiseIntensity),

表示 Bob 睡觉这个动作的上下文中包括和温度和噪音相关的两个环境要素.

Table 1 Specification for AOCA applications

表 1 AOCA 应用规约

非形式描述	形式规约	样例
位置 loc	loc[ ]	room812[ ]
位置 loc1 中包含位置 loc2	loc1[loc2[ ]]	room812[bedroom1[ ]]
人员 user	user[ ]	Bob[ ]
探测器 probe	probe [P]	ts1[P1]
探测器 probe 感知 feature 特征	P⊨sensingFea(feature)@ probe	P1⊨sensingFea(temperature)@ts1
探测器的位置或持有的人员	owner[probe[P]]	bedroom1[ts1[P1]]
执行器 actuator	actuator[P]	ac1[P2]
执行器 actuator 影响 feature 特征	P⊨influencingFea(feature)@ actuator	P2⊨influencingFea(temperature)@ac1
执行器的位置或持有的人员	owner[actuator[P]]	bedroom1[ac1[P2]]
用户 user 的动作 act	act[user[P]|Constraints|Q]	sleeping1[Bob[P]| Constraints|Q]
动作 act 的位置 loc	loc[act[P]]	bedroom1[sleeping1[…]]
动作 act 的类型 actType	act[P] ⊨hasActType(actType)	sleeping1[…]⊨hasActType(sleeping)
动作的约束由断言构成	assertion1|assertion2|...	sleeping1[Bob[P]| as1|as2|as3|Q]
断言 assertion 相关的特征是 feature	assertion⊨asHasFea (feature)	as1⊨asHasFea(temperature)
动作 act 的上下文包含若干环境要素 attr	act[cxt[cxtData[attr1|attr2|…]|P]]	sleeping1[cxt[cxtData[attr1|attr2|…]|P]]
环境要素 attr 的特征 feature	attr⊨attrHasFea (feature)	attr1⊨attrHasFea(temperature)

2.3 应用性质的形式化

AOCA 应用的运行系统需要保障的性质用 Ambient Logic 的形式进行描述,这样的性质包括系统级别性质和应用级别性质两类.

为了更好地描述这样的性质,我们可以预定义一些谓词,如 P⊨hasSubAmbient(Q)表示在 P[]这个 ambient 中包含一个子 ambient Q[],在进行形式规约时,该谓词随着 ambient 之间嵌套关系的建立进行赋值.

1) 系统级别性质

这类性质描述了脱离具体应用的、普遍适用于 AOCA 应用的性质.下面给出几个例子.

“每个动作皆有一个发生位置”这一性质可被描述为

$$\forall act\in Activity,act[P]\vDash\exists loc\in Location,hasSubAmbient(act)@loc \quad (1)$$

“每个动作皆有一个行动者”这一性质可被描述为

$$\forall act\in Activity,act[P]\vDash\exists per\in Person,hasSubAmbient(per) \quad (2)$$

“上下文中的每个环境要素的特征必然和动作相关的约束中的某个断言一致”这一性质可被描述为

$$\forall act\in Activity,act[P]\models\forall fea\in Feature,act[cxt[cxtData[attrHasFea(fea)]]]\rightarrow act[asHasFea(fea)] \quad (3)$$

2) 应用级别性质

这类性质描述了特定应用在运行过程中所需保障的性质,这些性质可由开发人员编写.下面给出几个老人看护系统中可设计的应用级别性质.

“烹饪这一动作所在的位置必须有一氧化碳传感器”这一性质可被描述为

$$\forall loc\in Location,loc[P]\models loc[hasActType(cooking)]\rightarrow loc[sensingFea(CO)] \quad (4)$$

“厨房中需要部署温度传感器和一氧化碳传感器”这一性质可被描述为

$$\forall loc\in Location,loc[P]\models isKitchen\rightarrow(loc[sensingFea(CO)|sensingFea(temperature)]) \quad (5)$$

“如果一个动作是剧烈运动,它的约束一定要涉及行动者的心跳”这一性质可被描述为

$$\forall act\in Activity,act[P]\models isStenuousAct\rightarrow act[asHasFea(heartRate)] \quad (6)$$

“如果一位老人进入浴室,则在 30 分钟内他应当离开(以免洗澡时发生事故)”这一性质可被描述为

$$\forall loc\in Location,loc[P]\models isBathroom\rightarrow(loc[hasActType(entering)]\rightarrow\Diamond^{\leqslant 30min}loc[hasActType(leaving)]) \quad (7)$$

3 AOCA 应用监控器实现

本节将对于我们基于上述方法实现的 AOCA 应用监控器进行介绍,首先给出监控器的软件结构,然后讨论监控器中的两个关键部分:应用状态翻译器和性质的验证器.

3.1 监控器软件结构

AOCA 应用的监控器是一个独立运行的软件部件,用以监控 AOCA 应用的运行状态,检验其中的对象及其关联关系是否满足一些特定的性质,其功能包括将 AOCA 应用的运行状态翻译为基于 Ambient Calculus 的形式系统,在这个形式系统的基础上验证性质,当某些性质不被满足时对应用进行反馈.从监控时机角度说,该监控器主要是接收 AOCA 应用的新动作产生以及动作发生变化时推送(push)的运行状态,根据不同需求,该监控器也可以采用某个特定的时间周期获取(pull)AOCA 应用的运行状态.它包含的主要部分如下(如图 2 所示).

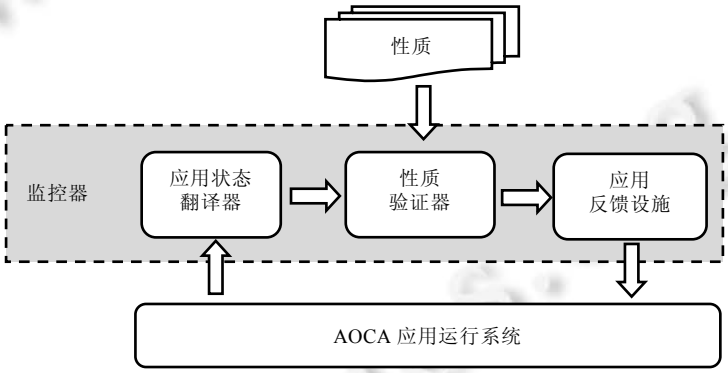


Fig.2 Structure of the monitor of AOCA applications

图 2 AOCA 应用监控器的结构

- 应用状态翻译器.该部分获取(pull)或接收推送(push)的 AOCA 应用运行状态,建立与 AOCA 应用的计算系统对应的形式系统,具体而言,是采用 Ambient Calculus 将 AOCA 应用的某个运行时(runtime)包含的用户、环境、设备、动作、上下文等对象的存在性及其关联关系进行形式规约,得到相应的形式系统.
- 性质验证器.当验证器收到翻译器给出的运行时状态,即一个 Ambient Calculus 表达式后,它将对这个表达式是否满足一些给定的性质进行验证,如果某些性质不满足,则告知反馈设施对应用进行反馈.

● 应用反馈设施.当监控器所监控的性质不满足时说明系统发生了异常情况,如信息不一致、不满足预定义的需求等,监控器通过反馈设施对 AOCA 应用进行反馈.具体的反馈方式是由具体应用类型确定的.

3.2 应用状态翻译器

AOCA 应用中的用户、环境、设备、动作、上下文等对象被规约为 Ambient Calculus 中的进程(包括 ambient),它们之间相互嵌套.这样的嵌套构成一种树状关系,AOCA 应用对应的形式系统即是由 1 个或多个这样的树构成.在系统实现过程中,这个树状关系被模拟出来,本文称其为 Ambient 树.

Ambient 树节点的数据结构如图 3 所示,每个节点描述的是一个其中不含有并行关系(即“|”符号)的进程,并行关系由一个 ambient 节点中的子节点列表进行描述..

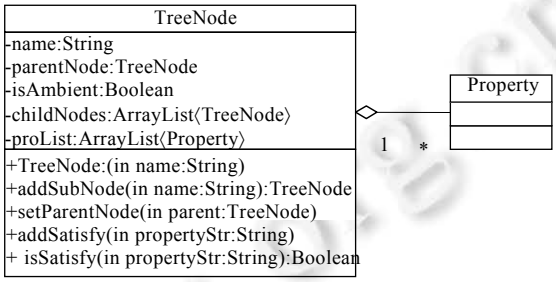


Fig.3 Data structure of Ambient tree nodes

图 3 Ambient 树节点的数据结构

对于每个节点,需描述它是否为 ambient、它的名称 name(如果是 ambient,即为 ambient 名;如果是普通进程,即为这个进程的描述)、它的父进程 parentNode(如果为顶层进程则为空)、它的子进程列表 childNodes(对于 ambient 节点)、它满足的性质.setParent 是一种用以设置某个节点的父节点的方法;addSubNode 是一种用以新建某个节点的子节点的方法;addSatisfy 是一种用以增加满足的性质的方法;isSatisfy 是一种用以判断一个性质是否满足的方法.

确定这样的树状关系后,应用状态翻译器将 AOCA 应用状态翻译为基于 Ambient Calculus 的形式系统可被理解为基于运行时一个时刻各实例的关系构建 Ambient 树的过程.算法 1~算法 4 展示了这一过程.

算法 1. 生成 Ambient 树根节点算法 generateRoots.

Input: 环境,*env*.

Output: Ambient 树根节点列表,*rootList*.

```
1: locationList:=getLocations(env);
2: for each loc in locationList do begin
3:     if (loc is not a subLocation)
4:         node:=addOwner(loc);
5:         rootList.add(node);
6:     end if
7: end for
8: return rootList;
```

算法 1 是生成 Ambient 树根节点算法.这些根节点描述了顶层的位置(即不是另一个位置的子位置)实例(行 3).对于每个位置,调用 addOwner(算法 2)生成树节点.

算法 2. 处理所有者实例的算法 addOwner.

Input: 所有者,*owner*.

Output: 生成的节点,*node*.

```
1: node:=new TreeNode(owner.ownerName);
2: node.isAmbient:=true;
```

```
3: addDeviceNodes(owner,node);
4: if (owner.type is Location
5:     subLocList:=owner. getSubLocations();
6:     for each subLoc in subLocList do begin
7:         temNode:=addOwner(subLoc);
8:         temNode. setParentNode(node);
9:     end for
10:    addActivityNodes(owner,node);
11: end if
12: return node;
```

算法 2 是处理所有者实例的算法.该算法用以添加位置或人员两类所有者.在建立所有者节点之后,调用addDeviceNodes(算法 3)生成该所有者中的设备节点;对于一个包含其他位置的位置实例,该方法递归调用自己(行 7),生成子位置的节点.对于位置节点,还需调用 addActivityNodes(算法 4)生成发生于这个位置的动作节点.

算法 3. 处理设备实例的算法 addDeviceNodes.

```
Input: 所有者,owner;所有者的节点,node.
1: attrSet:=owner.getAttrSet();
2: for each attr in attrSet do begin
3:     probe:=attr.getProbe();
4:     name:=probe.name;
5:     temNode:=node.addSubNode(name);
6:     temNode.isAmbient:=true;
7:     processNode:=temNode.addSubNode("P"+(++num));
8:     processNode.addSatisfy("sensingFea("+attr.getFeature().name+")@"+name);
9:     processNode.isAmbient:=false;
10:    actuator:=attr.getActuator();
11:    if (actuator!=NULL)
12:        name:=actuator.name;
13:        temNode:=node.addSubNode(name);
14:        temNode.isAmbient:=true;
15:        processNode:=temNode.addSubNode("P"+(++num));
16:        processNode.addSatisfy("influencingFea("+attr.getFeature().name+")@"+name);
17:        processNode.isAmbient:=false;
18:    end if
19: end for
```

算法 3 是添加一个所有者中探测器、执行器的算法.在本文的方法中,每个设备被形式化规约为一个ambient,这样的 ambient 中运行着一个进程(行 7、行 15),进程名以 P_{num} 的形式定义,num 依次递增,保证唯一性.该算法通过要素相关的特征对于探测器执行器满足的性质进行描述(行 8、行 16).

算法 4. 处理动作实例的算法 addActivityNodes.

```
Input: 位置,loc;位置节点,node.
1: actList:=loc.getActList();
2: for each act in actList do begin
3:     actNode:=node.addSubNode(act.name);
4:     actNode.isAmbient:=true;
5:     actNode.addSatisfy("hasActType("+act.Type+")");
6:     actor:=act.getActor();
7:     perNode:=addOwner(actor);
8:     perNode.setParentNode(actNode);
9:     assertionList:=act.getAssertions();
10:    for each as in assertionList do begin
```

```
11:         asNode:=actNode.addSubNode(as.name);
12:         asNode.addSatisfy("hasFea("+as.getFeature().name+")");
13:         asNode.isAmbient:=false;
14:     end for
15:     cxt:=act.getContext();
16:     cxtNode:=actNode.addSubNode("cxt");
17:     cxtNode.isAmbient:=true;
18:     cxtDataNode:=cxtNode.addSubNode("cxtData");
19:     cxtDataNode.isAmbient:=true;
20:     attrSet:=cxt.getAttrSet();
21:     for each attr in attrSet do begin
22:         attrNode:=cxtDataNode.addSubNode(attr.name);
23:         attrNode.addSatisfy("hasFea("+attr.getFeature().name+")");
24:         attrNode.isAmbient:=false;
25:     end for
26: end for
```

算法 4 是添加一个位置中正在发生的动作的算法.生成动作节点后,该算法设置动作的类型(行 5)、调用 addOwner 算法添加其中的行动者(行 7),然后添加该动作相关的断言(行 11),之后再添加该动作的上下文节点(名为 cxt,行 16).在上下文中添加环境要素作为上下文数据(行 22).断言和上下文中环境要素满足的性质分别被添加(行 12、行 23).

3.3 性质验证器

AOCA 应用的性质验证器用以对于 $P \models A$ 形式描述的性质进行验证,即考察一个进程 P 是否满足公式 A.首先需要确定进程 P 对应的 Ambient 树节点,有的性质可能涉及到多个进程,如$\forall act \in Activity, act[P] \models A$ 可能涉及到多个动作,需要分别进行验证.

这个验证器的结构如图 4 所示.在执行性质验证时,我们首先将性质中的公式 A 进行预处理.我们根据文献[10]介绍的算法设计了 AOCA 应用监控器中的性质验证器,与通用的 Ambient Logic 验证器相比,这个验证器对于部分 AOCA 应用形式化规约没有用到的语言成分做了删减.等待区用以处理性质中含有时态算子时,等待后续运行状态输入的情况.

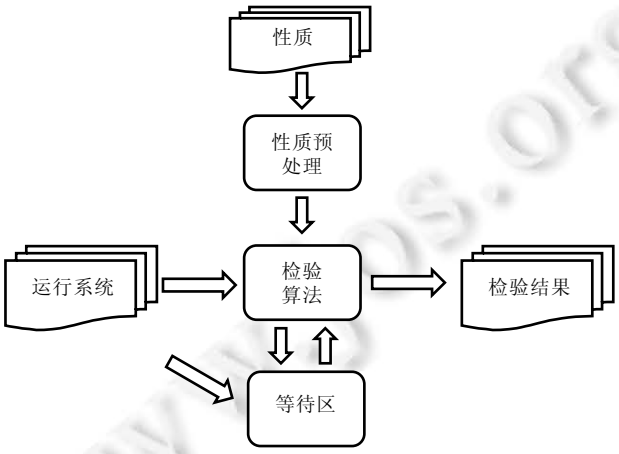


Fig.4 Structure of the property verifier

图 4 性质验证器的结构

性质的预处理过程见算法 5.

算法 5. 性质预处理 preprocess.

Input: 公式,A.

Output: 转换后的公式,A'.

1: A':=A;
2: 对于 A'中所有$\forall x.A1$ 形式的部分,转为$\neg(\exists x.\neg A1)$;
3: 对于 A'中所有$\Box A1$ 形式的部分,转为$\neg(\Diamond\neg A1)$;
4: 将 A'转为合取范式(包括将 $A1\rightarrow A2$ 形式的部分转为$\neg A1\vee A2$);
5: 将合取范式 $A1\wedge A2\wedge...\wedge An$ 转为$\neg(\neg A1\vee\neg A2\vee...\vee\neg An)$;
6: **return** A';

完成上述预处理后,系统用检验算法(算法 6)对于相应的 Ambient 树节点是否满足一个特定的公式进行检验.

算法 6. 性质检验算法 check.

Input: Ambient 树节点,*node*;公式,A.

Output: Boolean.

```
1: switch (公式 A 呈现的形式) do begin
2:     case (TRUE)
3:         return true;
4:     case (¬A')
5:         return (not check(node,A'));
6:     case (A1∨A2)
7:         return (check(node,A1) or check(node,A2));
8:     case (n[A'])
9:         if (node.name=n)
10:            if(A'是 k 个公式组成的 A1|A2|...的形式)
11:                if(在 node.childNodes 中存在 k 个节点,分别满足公式 A1|A2|...)
12:                    return true;
13:            else
14:                for node.childNodes 中的每个子节点 node' do begin
15:                    if(check(node',A')=true)
16:                        return true;
17:                end for
18:        end if
19:        return false;
20:    case (A'@n)
21:        if(node.parentNode.name=n)
22:            return check(node.parentNode,A');
23:        return false;
24:    case (∃x.A')
25:        for node 的表达式中涉及到的每个为自由变元的名 fn do begin
26:            if (check(node,A'{x←fn})=true)
27:                return true;
28:        end for
29:        return false;
30:    case(∇A')
31:        if (check(node,A')=true)
32:            return true;
33:        for each node' in node.childNodes do begin
34:            if (check(node',∇A')=true)
35:                return true;
```

```
36:         end for
37:         return false;
38:   case(◇^{≤k}A′)
39:         if(k 是采样的次数)
40:               if(在 k 个状态之内满足公式 A′)
41:                     return true;
42:               else
43:                     return false;
44:         else                                    //k 是一个时间段
45:               if(在时间段 k 内以一个较短的周期获得运行状态,如其中有状态满足公式 A′)
46:                     return true;
47:               else
48:                     return false;
49:         end if
50:   else
51:         return node.isSatisfy(A);
52: end switch
```

该算法针对性质中的各种符号分别递归地进行处理.当公式中不再含有上述符号时,则查询节点 node 是否在之前设置过满足 A 公式(行 51).

由于该算法对 $A1|A2|\ldots$形式的公式处理过程(行 11)需要从一组进程(n 个)中选取若干个(k 个)分别进行检验,这一过程在最坏情况下需要进行 $A(n,k)$的排列.该算法最坏情况下的复杂度是阶乘级的.然而,一方面限于 AOCA 应用规模,另一方面我们在实现该算法过程中进行了一些剪枝优化,实际运行过程中验证器的运行效率可以接受(参见第 4.3 节实验说明).进一步地,如文献[10]所证明的,对于 Ambient Logic 进行验证的复杂度级别与 CTL 等被常用于形式化验证的逻辑语言是一致的.

4 案例研究与性能评估

本节仍以老人看护应用为例,讨论上述规约和验证方法的有效性.我们首先介绍运行时状态的形式化规约案例;然后介绍性质验证的案例;最后,通过一个实验对本文所提出方法的运行效率进行讨论.

4.1 运行时状态规约案例

在 AOCA 应用运行时,其运行状态被规约为 Ambient Calculus 的形式,本节以老人看护应用为例介绍这一规约过程.

老人生活的智能家居环境为 812 号房间(room812),其中,包含卧室、客厅、厨房.

room812[bedroom1[ ]|livingroom2[ ]|kitchen3[ ]]

该智能环境中部署着一些设备:

room812[bedroom1[ts1[P1]]|nis1[P2]]|ta1[P3]]|livingroom2[lis1[P4]|nis2[P5]]|kitchen3[ts2[P6]]]

where P1⊨sensingFea(temperature)@ts1,P2⊨sensingFea(noiseIntensity)@nis1,
P3⊨influencingFea(temperature)@ta1,P4⊨sensingFea(lightIntensity)@lis1,
P5⊨sensingFea(noiseIntensity)@nis2,P6⊨sensingFea(temperature)@ts2.

上述 ambient 描述了 bedroom1,livingroom2,kitchen3 中的一些探测器和执行器.其中,卧室中的噪音强度和温度可被感知、温度可被调节;客厅的光线强度和噪音可被感知;厨房的温度可被感知.

同样地,可部署一些感知用户状态的设备.

Bob[bps1[P1]|hrs1[P2]]

where P1⊨sensingFea(bloodPressure)@bps1,P2⊨sensingFea(heartRate)@hrs1.

上述 ambient 描述了老人 Bob 持有可感知血压和心跳的设备.

在运行时某一时刻,系统识别到的动作是“Bob 在 livingroom2 中阅读”,可被规约为

livingroom2[reading1[Bob[...]|as1|as2|as3|as4|as5]]

where reading1⊨hasActType(reading),as1⊨asHasFea(lightIntensity),as2⊨asHasFea(temperature)

as3⊨asHasFea(noiseIntensity),as4⊨asHasFea(bloodPressure),as5⊨asHasFea (heartRate).

该动作对于光线强度、温度、噪音、心跳、血压这 5 个要素有要求.

为该动作生成的上下文如下:

reading1[...|cxt[cxtData[attr1|attr2|attr3|attr4]|P]]

where attr1⊨attrHasFea(lightIntensity),attr2⊨attrHasFea (noiseIntensity),

attr3⊨attrHasFea(bloodPressure),attr4⊨attrHasFea(heartRate).

该上下文中包含了 4 个环境要素,需要注意的是,由于该动作所在的位置 livingroom2 中没有感知温度的探测器,上下文中不包含与温度相关的要素.

4.2 性质验证案例

在本节中,我们以第 2.3 节所给出的 7 条性质为例,对于老人看护应用的运行时状态进行验证.该过程即是监控器将运行状态翻译的结果与上述 7 条性质分别作为验证器的两个输入,执行运行时验证,判断性质是否被满足.

场景 1. Bob 在客厅 livingroom2 中阅读,系统运行时状态规约为以下表达式.

livingroom2[lis1[P4]|nis2[P5]|reading1[Bob[]|as1|as2|cxt[cxtData[attr1|attr2]]

where P4⊨sensingFea(lightIntensity)@lis1,P5⊨sensingFea(noiseIntensity)@nis2,

as1⊨asHasFea(lightIntensity),as2⊨asHasFea(temperature),

attr1⊨attrHasFea(lightIntensity),attr2⊨attrHasFea(noiseIntensity).

对于这个表达式,上下文中包含噪音相关的环境要素(attr2),但是该动作的约束中并不包含噪音,该状态无法满足系统级别性质(3),监控器进行相应的提醒.

场景 2. Bob 在厨房 kitchen3 中烹饪,系统运行时状态规约为以下表达式.

kitchen3[cooking1[Bob[ ]|P]|ts2[P6]]

where cooking1[P]⊨hasActType(cooking),P6⊨sensingFea(temperature)@ts2.

由于该厨房中没有一氧化碳探测器(仅有一个温度探测器),该状态无法满足应用级别性质(4),监控器进行相应的提醒.

场景 3. Bob 在客厅 livingroom2 中阅读,系统运行时状态规约为以下表达式.

livingroom2[lis1[P4]|nis2[P5]|reading1[Bob[bps1[P1]]|as1|as2|as3|cxt[cxtData[attr1|attr2|attr3]]

where P4⊨sensingFea(lightIntensity)@lis1,P5⊨sensingFea(noiseIntensity)@nis2,

P1⊨sensingFea(bloodPressure)@bps1,as1⊨asHasFea(lightIntensity),

as2⊨asHasFea(noiseIntensity),as3⊨asHasFea(bloodPressure),

attr1⊨attrHasFea(lightIntensity),attr2⊨attrHasFea(noiseIntensity),

attr3⊨attrHasFea(bloodPressure).

该状态满足第 2.3 节中提到的所有公式,这个场景满足现有的性质.

4.3 性能评估

本节对于本文提出的 AOCA 应用规约与运行时验证方法的运行效率进行实验分析,由于该监控器是在原有 AOCA 应用的运行时增加了一个运行过程,这个实验试图回答以下研究问题.

• RQ:在系统运行过程中,相对于原有的上下文感知过程,我们实现的监控器的时间开销是怎样的?

为了对上述问题进行评估,我们建立了一个智能空间的模拟实验环境.该空间包括 30 个位置,其类型(如厨房、浴室等)随机确定,在每个位置都会有一些虚拟的探测器、执行器被随机地部署进来.我们分别测试 5 个动

作 7 条规则、5 个动作 20 条规则、20 个动作 7 条规则、20 个动作 20 条规则这 4 种情况下监控器实现应用状态翻译、性质验证的总时间开销,作为对比,同时测试这 4 种情况下原有的上下文感知过程(包括上下文生成与使用)的时间开销.

这些实验的服务器为一个 4G RAM,Intel 3.2GHz 双核 CPU 的计算机.在这个评估中,一些因素被忽略,如网络的带宽,这些因素在比较过程中保持不变.对于每个实验,模拟系统运行 20 次,时间开销被分别记录下来取平均值.

图 5 展示了实验结果,其中,VER 列表示监控器进行运行状态翻译、性质验证所消耗的时间,CA 列表示原来的 AOCA 应用系统中进行上下文生成、使用的时间.从这个结果可以看出,监控器的耗时随着动作数量、性质数量的增加而增加,与 AOCA 应用原有的时间开销(这一时间大致与动作数量成正比)相比,这一运行时验证过程并不会带来太大的延时.

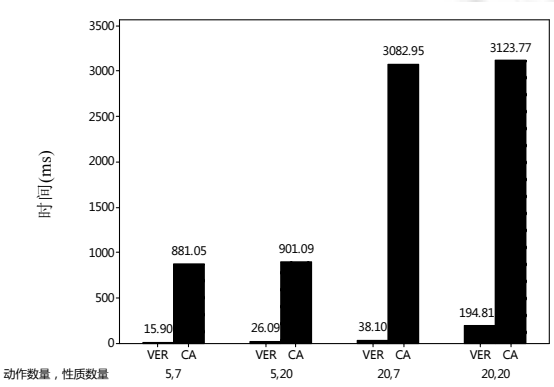


Fig.5 Results of the performance evaluation

图 5 性能评估的结果

根据这一实验结果,我们回答上述研究问题如下.

• RQ:相对于原有的 AOCA 应用上下文感知过程,我们实现的监控器并不会在运行时增加太多的时间开销,该方法以及基于此方法实现的监控器在 AOCA 应用的一般规模下,其时间开销是可接受的.

5 相关工作

运行时验证是提高软件可靠性的常用手段.针对各种类型的应用,研究者们采用适合的形式系统,给出软件运行时状态的形式化规约方法以及为运行时验证提供支持的软件组件.较为典型的工作包括了针对 CPS[6]、云计算[15]、医疗护理[16]、自治车辆[17]等系统的运行时验证.对于本文关注的 AOCA 应用而言,也同样需要给出这类应用特定的形式化规约方法和运行时验证支持.

对于上下文感知应用而言,利用形式化规约和验证提高可靠性也得到了广泛的研究.部分工作(如文献[18,19])仅关注对于环境部分的形式化规约,而不包括对于运行时应用行为的描述,对于 AOCA 应用而言,仅考虑环境部分的话无法描述动作相关的应用需求以及 AOCA 应用在运行时的状态,不能很好地解决可靠性问题.部分工作(如文献[20–22])同时考虑了上下文感知应用环境和应用行为的形式化规约,实现了对应用运行时状态的验证,但是这些针对通用性上下文感知应用工作仍然不能很好地应对 AOCA 应用开发、运行的特点.针对 AOCA 应用给出特定的运行时验证方法具有研究价值.

Ambient Calculus/Ambient Logic 被广泛用于对涉及到空间资源和人员、设备状态的应用进行描述.如 Weis 等人[23]给出了基于 Ambient Calculus 的编程风范;Ranganathan 等人[21]采用这个形式化工具验证性质、保障上下文感知应用的可靠性;Coronato 等人[24]对基于该形式化工具的运行时监控器进行了实现.这些工作对于我们选取形式化规约、验证工具有所启发.

6 总结与展望

本文针对面向动作的上下文感知(AOCA)应用开发和运行的特点,提出了提高这类应用可靠性的规约与运行时验证方法.我们给出了基于 Ambient Calculus 的 AOCA 应用运行状态形式化规约方法以及基于 Ambient Logic 的性质描述方法.进一步地,我们实现了 AOCA 应用的监控器,其中包括将应用运行状态翻译为形式系统的工具以及性质验证工具.

在未来,我们考虑进行以下后续研究:首先,设计一个包含文本提示和语法查错等功能的性质描述工具,为应用性质的描述提供更好的支持;进一步地,当前工作是从 AOCA 应用的用户动作部分出发考察这类系统,针对一个状态或一定时间段内的若干状态进行运行时验证,由于用户动作具有自主性、随时切换,其状态的轨迹难以预计,当前方法是适合的,在未来,我们考虑从应用本身出发,考察其运行过程,采用 Ambient Calculus 提供的 ambient 进入、离开、消息传递等描述手段对应用的运行状态转换进行规约,以处理应用运行过程中无限轨迹的情况(这方面的部分工作基础见文献[25]);此外,我们会在更多场景中进行实验,以验证该方法的实用性,并考虑该方法是否能够在改进后,在 AOCA 应用以外的其他类型应用(例如过程感知应用[26])中得到使用.

References:

[1] Weiser M. The computer for the 21st century. Scientific American, 1991,261(30):94−104.

[2] Satyanarayanan M. Pervasive computing: Vision and challenges. IEEE Personal Communications, 2001,8(4):10−17. [doi: 10.1109/98.943998]

[3] Dey AK, Abowd GD, Salber D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. Human-Computer Interaction, 2001,16(2):97−166. [doi: 10.1207/S15327051HCI16234_02]

[4] Gu T, Pung HK, Zhang DQ. A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications, 2005,28(1):1−18. [doi: 10.1016/j.jnca.2004.06.002]

[5] Li XS. Research on technologies of design and implementation for activity-oriented context-aware (AOCA) application systems [Ph.D. Thesis]. Nanjing: Nanjing University, 2016 (in Chinese with English abstract).

[6] Yu K, Chen Z, Dong W. A predictive runtime verification framework for cyber-physical systems. In: Proc. of the 8th IEEE Int'l Conf. on Software Security and Reliability-Companion. IEEE, 2014. 223−227. [doi: 10.1109/SERE-C.2014.43]

[7] Zhang X, Dong W, Qi ZC. Conflicts detection in runtime verification based on AOP. Ruan Jian Xue Bao/Journal of Software, 2011, 22(6):1224−1235 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4016.htm [doi: 10.3724/SP.J.1001.2011.04016]

[8] Wang L, Gu T, Tao X, Chen H, Lü J. Recognizing multi-user activities using wearable sensors in a smart home. Pervasive and Mobile Computing, 2011,7(3):287−298. [doi: 10.1016/j.pmcj.2010.11.008]

[9] Cardelli L, Gordon AD. Mobile ambients. In: Proc. of the Int'l Conf. on Foundations of Software Science and Computation Structure. Berlin, Heidelberg: Springer-Verlag, 1998. 140−155. [doi: 10.1007/BFb0053547]

[10] Charatonik W, Zilio SD, Gordon AD, Mukhopadhyay S, Talbot J. Model checking mobile ambients. Theoretical Computer Science, 2003,308(1-3):277−331. [doi: 10.1016/S0304-3975(02)00832-0]

[11] Cardelli L, Gordon AD. Ambient logic. In: Mathematical Structures in Computer Science. 2003.

[12] Baier C, Katoen JP, Larsen KG. Principles of Model Checking. MIT Press, 2008. 673−744.

[13] Pnueli A. The temporal logic of programs. In: Proc. of the 18th Annual Symp. on Foundations of Computer Science. IEEE, 1977. 46−57. [doi: 10.1109/SFCS.1977.32]

[14] Emerson EA, Halpern JY. Decision procedures and expressiveness in the temporal logic of branching time. In: Proc. of the 14th Annual ACM Symp. on Theory of Computing. ACM, 1982. 169−180. [doi: 10.1145/800070.802190]

[15] Zhou J, Chen Z, Wang J, Zheng Z, Dong W. A runtime verification based trace-oriented monitoring framework for cloud systems. In: Proc. of IEEE Int'l Symp. on Software Reliability Engineering Workshops. IEEE, 2014. 152−155. [doi: 10.1109/ISSREW.2014.84]

[16] Jiang Y, Liu H, Kong H, Wang R, Hosseini M, Sun J, Sha L. Use runtime verification to improve the quality of medical care practice. In: Proc. of the 38th IEEE/ACM Int'l Conf. on Software Engineering Companion. ACM, 2016. 112−121. [doi: 10.1145/2889160.2889233]

[17] Kane A, Chowdhury O, Datta A, Koopman P. A case study on runtime monitoring of an autonomous research vehicle (ARV) system. In: Proc. of the 6th Int'l Conf. Runtime Verification. LNCS 9333, Springer Int'l Publishing, 2015. 102−117. [doi: 10.1007/978-3-319-23820-3_7]

[18] Anagnostopoulos CB, Ntarladimas Y, Hadjiefthymiades S. Situational computing: An innovative architecture with imprecise reasoning. Journal of Systems and Software, 2007,80(12):1993−2014. [doi: 10.1016/j.jss.2007.03.003]

[19] Konur S, Fisher M, Dobson S, Knox S. Formal verification of a pervasive messaging system. Formal Aspects of Computing, 2014, 26(4):677−694. [doi: 10.1007/s00165-013-0277-4]

[20] Schmidtke HR, Woo W. Towards ontology-based formal verification methods for context aware systems. In: Proc. of the 7th Int'l Conf. on Pervasive Computing. LNCS 5538, Berlin, Heidelberg: Springer-Verlag, 2009. 309−326. [doi: 10.1007/978-3-642-01516-8_21]

[21] Ranganathan A, Campbell RH. Provably correct pervasive computing environments. In: Proc. of the 6th IEEE Int'l Conf. on Pervasive Computing and Communications. IEEE, 2008. 160−169. [doi: 10.1109/PERCOM.2008.116]

[22] Coronato A, De Pietro G. Formal specification of wireless and pervasive healthcare applications. ACM Trans. on Embedded Computing Systems (TECS), 2010,10(1):12:1−12:18. [doi: 10.1145/1814539.1814551]

[23] Weis T, Becker C, Brändle A. Towards a programming paradigm for pervasive applications based on the ambient calculus. In: Proc. of the Int'l Workshop on Combining Theory and Systems Building in Pervasive Computing (CTSB). 2006.

[24] Coronato A, De Pietro G. Tools for the rapid prototyping of provably correct ambient intelligence applications. IEEE Trans. on Software Engineering, 2012,38(4):975−991. [doi: 10.1109/TSE.2011.67]

[25] Li XS, Tao XP, Lü J. Programming method and formalization for activity-oriented context-aware applications. In: Proc. of the 12th Int'l Conf. on Ubiquitous Intelligence and Computing. IEEE, 2015. 174−181. [doi: 10.1109/UIC-ATC-ScalCom-CBDCom-IoP.2015.48]

[26] Song W, Ma XX, Hu H, Lü J. Dynamic evolution of processes in process-aware information systems. Ruan Jian Xue Bao/Journal of Software, 2011,22(3):417−438 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/3962.htm [doi: 10.3724/SP.J.1001.2011.03962]

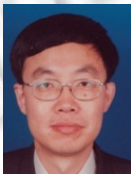
附中文参考文献:

[5] 李晅松.面向动作的上下文感知应用系统的设计与实现技术研究[博士学位论文].南京:南京大学,2016.

[7] 张献,董威,齐治昌.基于 AOP 的运行时验证中的冲突检测.软件学报,2011,22(6):1224−1235. http://www.jos.org.cn/1000-9825/4016.htm [doi: 10.3724/SP.J.1001.2011.04016]

[26] 宋巍,马晓星,胡昊,吕建.过程感知信息系统中过程的动态演化.软件学报,2011,22(3):417−438. http://www.jos.org.cn/1000-9825/3962.htm [doi: 10.3724/SP.J.1001.2011.03962]

李晅松(1985－),男,山东博兴人,博士,讲师,CCF 专业会员,主要研究领域为软件工程与方法学,形式化方法,普适计算技术.



吕建(1960－),男,博士,教授,博士生导师,CCF 会士,主要研究领域为软件形式化与自动化,面向对象方法与技术,软件工程与方法学.

陶先平(1970－),男,博士,教授,博士生导师,CCF 高级会员,主要研究领域为软件中间件技术,网构软件方法学,普适计算技术.

宋巍(1981－),男,博士,副教授,CCF 高级会员,主要研究领域为软件工程与方法学,形式化方法,服务计算.