软件学报 ISSN 1000-9825, CODEN RUXUEW
*Journal of Software*,2011,22(5):865−876 [doi: 10.3724/SP.J.1001.2011.03800]

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

# 开放环境特性感知技术*

黄 宇[1,2+], 余建平[1,2], 马晓星[1,2], 陶先平[1,2], 吕 建[1,2]

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

[2](南京大学 计算机软件研究所,江苏 南京 210093)

## Monitoring Properties of Open Environments

HUANG Yu[1,2+], YU Jian-Ping[1,2], MA Xiao-Xing[1,2], TAO Xian-Ping[1,2], LÜ Jian[1,2]

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

[2](Institute of Computer Software, Nanjing University, Nanjing 210093, China)

+ Corresponding author: E-mail: yuhuang@nju.edu.cn, http://cs.nju.edu.cn/yuhuang/

**Huang Y, Yu JP, Ma XX, Tao XP, Lü J. Monitoring properties of open environments. *Journal of Software*, 2011,22(5):865−876.** http://www.jos.org.cn/1000-9825/3800.htm

**Abstract**: This paper proposes a framework for the specification of open environment properties. This framework enables convenient and formal specification of properties of asynchronous environments, including temporal properties which cannot be handled in the existing work. This framework also supports implementation of effective schemes for monitoring environment properties based on distributed predicate detection. A middleware infrastructure is designed and implemented, which supports efficient monitoring of environment properties for building high-confidence systems in open environments. This paper also evaluates design of the proposed infrastructure with a case study.

**Key words**: predicate detection; environment property; high-confidence software; open environment

**摘 要**: 提出了一个开放环境特性描述框架.该框架支持便捷地、形式化地描述异步环境的各种特性,包括那些既有技术不能处理的时序特性.该框架还引入了谓词检测技术,支持高效的环境特性感知机制的实现.开发了一个开放环境特性感知中间件平台,并通过详细的案例分析展示了如何基于所提出的环境特性描述框架与中间件平台,高效地感知环境特性,支持可信软件系统的构建.

**关键词**: 谓词检测;环境特性;高可信软件;开放环境

**中图法分类号**: TP311 **文献标识码**: A

一般说来,软件系统总是在一定的环境中运行.软件系统利用环境中的各种资源,也通过自身的行为改变环境.Internet 的出现和普及,使计算机软件所面临的运行环境从封闭、静态、可控逐步走向开放、动态、难控,在此开放环境下构建行为可理解、结果可预期的可信软件系统的一个基本要求是准确感知并恰当应对环境状态及其变化[1,2].

* 基金项目: 国家自然科学基金(60903024, 60736015, 60721002); 国家重点基础研究发展计划(973)(2009CB320702); 江苏省攀登计划(BK2008017)

收稿时间: 2009-06-15; 修改时间: 2009-09-11; 定稿时间: 2009-12-07; jos 在线出版时间: 2010-05-28

在传统软件开发技术中,环境设定通常被当作一个固定的、全局性的假设体现在软件的需求分析、设计实现、部署运行等各个阶段[3].在此过程中的各种验证和确认(verification and validation)技术也以此环境设定为隐含前提.环境设定的遵循主要通过对软件过程,特别是部署过程的管理来实现.这种技术在封闭环境中是可行的,但若要应用于开放环境,则只能将环境状态当作输入的一部分,并在应用逻辑之中予以特别的处理.由于开放环境所固有的复杂性和多变性,这种处理方式往往使得应用逻辑复杂凌乱、难以形式地验证、难以适应环境的变化,最终损害系统的可信性.因而,我们需要遵循"关注分离"这一软件工程基本原则,将对环境的感知和应对从应用逻辑中尽可能地分离出来,予以显式地、系统地处理.即将隐式为主的环境和静态为主的应变转变成显式化环境与动态化应变[2].本文的工作主要研究如何准确感知环境的状态及其变化,虽然这方面的已有许多研究工作[4,5],但其对于构建开放环境下具有容变能力的可信软件尚有一些重要的不足:

- 缺乏通用、高效而又可靠的环境感知技术框架.从软件技术的角度看,对环境感知就是在基本的环境探测器所给出的原始信息的基础上,通过环境特性建模来建立这些原始信息与高层环境特性语义的联系,进而自动地从原始信息获知环境特性,亦即环境特性感知.尽管人们已提出多种环境模型,但要么形式化程度较弱,难以支持可靠的环境特性感知,如用简单名-值对来刻画环境;要么过于复杂,导致使用困难,实际运行效率低下,如各种环境本体模型[2].
- 既有环境特性感知技术未充分考虑开放环境固有的分布性.开放环境本质上是一个"真"分布环境,环境设施间的协同是异步的,环境设施之间往往没有同步的全局时钟.环境设施间的通信会面临不确定的延迟[6],并且受资源的限制,环境收集设备经常需要将收集到的环境数据缓存一定的时间再作分发[7].因而,在开放环境下时间的概念需要被重新审视,基于统一全局时钟假设的传统技术难以感知开放环境的时序特性,例如并发、先后等.

针对开放环境对构建可信软件系统提出的挑战与已有环境感知技术的不足,本文提出一种基于形式逻辑的环境特性描述与感知方法,其核心是谓词检测技术.具体而言,本文的贡献在于:

- 提出了一个开放环境特性描述框架,该框架可以便捷地、形式化地描述"真"分布环境的各种特性,包括那些既有技术不能处理的时序特性.引入谓词检测技术以支持高效的环境特性感知机制的实现,有效应对开放环境动态、难控的特点.
- 开发了一个支持可信应用构建与运行的开放环境特性感知中间件平台 Middleware Infrastructure for Predicate Detection in Asynchronous Environments(MIPA),通过 MIPA 平台提供的谓词检测服务,应用可自动感知其关注的环境特性.
- 通过一个案例分析展示了如何基于所提出的环境特性描述框架与 MIPA 平台,有效地感知用户任意指定的环境特性.

本文第 1 节介绍环境特性描述框架.第 2 节介绍 MIPA 中间件平台的设计.第 3 节是案例分析.第 4 节回顾相关工作.第 5 节总结全文并展望未来的工作.

## 1 面向异步环境的特性描述框架

本节提出一个基于形式逻辑的开放环境特性描述框架.在描述能力方面,该描述框架支持异步环境下时序特性的描述,有效地弥补了传统环境规约技术在"真"分布开放环境下的不足;在效率方面,该描述框架以谓词作为环境特性规约的基础,较好地保留了传统非形式化环境规约技术的简洁易用性;同时,基于该描述框架的环境特性感知机制具有较高的实际运行效率.我们的描述框架的核心是产生于分布程序调试的谓词检测技术[8−10],并将该技术运用到开放环境特性感知中来.下面,我们首先给出系统模型,然后给出描述框架.在第 2 节与第 3 节中,我们将详细讨论如何基于我们的描述框架构建环境感知应用.

### 1.1 系统模型

为了实现显式化环境与动态化应变,开放环境下有多个环境设施负责环境信息的收集、分发与处理,其间通过异步的消息传递进行通信.我们将环境设施建模成一个松耦合的基于消息传递的分布式系统,其间没有统

一的全局时钟,并且通信面临不确定的延迟.每个环境设施被抽象成一个进程 $P_i$,$1\leqslant i\leqslant n$,每个 $P_i$ 的执行过程可以表示为该进程的状态(state)$s_{ij}$ 经过操作(action)$a_{ij}$ 后不断转换的过程,即

$$Trace(P_i)=s_{i0}a_{i0},s_{i1}a_{i1},\ldots,s_{iu}a_{iu}.$$

$P_i$ 上的操作 $a_{ij}$ 可分为 3 类:

i) 本地操作:仅涉及一个进程的本地信息而不与外界交互的操作.

ii) 消息发送操作 $send(i,j,\omega)$:进程 $P_i$ 向进程 $P_j$ 发送消息 $\omega$.

iii) 消息接收操作 $receive(i,j,\omega)$:进程 $P_j$ 从进程 $P_i$ 接收消息 $\omega$.

Happen-Before 关系:异步环境特性描述的核心是不同状态/操作间的因果联系(causality)所蕴含的 happen-before 关系(记为$\rightarrow$)[6],其定义为(为了表述方便,下面的讨论中仅考虑进程的状态 $s_{ik}$,省略其操作 $a_{ik}$):

- 对给定的 $i$ 和任意 $k<l$,$s_{ik}\rightarrow s_{il}$;
- 消息发送操作 $send(i,j,\omega)$前后,进程 $P_i$ 和 $P_j$ 的状态分别为 $s_{it}$ 和 $s_{jr}$,则 $s_{it}\rightarrow s_{jr}$;
- 对于任意两个状态 $s_1$ 和 $s_3$,如果存在状态 $s_2$ 满足:$s_1\rightarrow s_2$ 且 $s_2\rightarrow s_3$,则 $s_1\rightarrow s_3$.

我们可以通过逻辑向量时钟来记录并方便地判断系统中状态间的 happen-before 关系[10].

#### 1.1.1 一致全局状态

系统的一次执行(记为 RUN)由每个进程的 $Trace(P_i)$,$1\leqslant i\leqslant n$ 组成,每个进程的本地状态 $s_i$ 组成的向量记为一个全局状态(global state),$g=[s_1,s_2,\ldots,s_n]$,我们定义一个全局状态是一致的(consistent global state,简称 CGS),当且仅当:任取 $s_i$ 与 $s_j$ 属于 $g$,$s_i\rightarrow s_j$ 与 $s_j\rightarrow s_i$ 均不成立.

全局状态的一致性的直观物理含义是:一个 CGS 是在系统执行过程中可能出现的全局状态;相反,一个不一致的全局状态则是在实际系统执行过程中不可能出现的.

#### 1.1.2 全局序列

基于 CGS 的定义,一个系统的执行则可以看成是系统从一个 CGS 推进到下一个 CGS,系统执行过程中先后经历的若干 CGS 形成的序列,我们定义为一个全局序列(global sequence,简称 GSE).一个 GSE 中的 CGS 之间满足偏序关系$\prec$:对于全局状态 $g=(s_1,s_2,\ldots,s_n)$,$g'=(s'_1,s'_2,\ldots,s'_n)$,我们定义 $g\prec g'$ 当且仅当 $g\neq g'$,且对任意 $1\leqslant i\leqslant n$,$s_i=s'_i$ 或者 $s_i\rightarrow s'_i$.

对于异步系统的一次执行,往往由于缺乏充分的消息传递,系统中部分状态之间并不能建立 happen-before 关系.对于没有建立 happen-before 关系的两个状态,它们的两种相对执行顺序都是可能的.系统的一次执行 RUN 中所有的 CGS,在$\prec$关系下形成一个格[9,10],其几何表示如图 1 所示.我们观察到的系统执行并不是一条确定的 GSE,而是一组可能的 GSE 组成的集合.

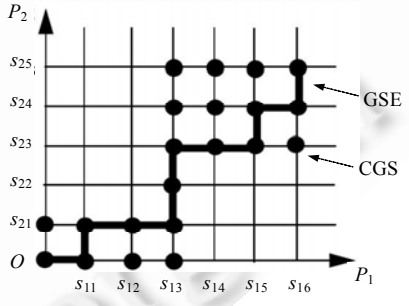



Fig.1 Lattice of consistent global states

图 1 一致全局状态组成的格

### 1.2 环境特性谓词

环境特性描述框架的核心是环境特性谓词的定义,环境特性谓词的结构与上述系统模型是密切对应的.具体而言,我们可以采用自顶向下的方式来看待一个异步系统的构成:系统的一次执行 RUN 包含多条可能的 GSE;每个 GSE 由若干 CGS 构成;每个 CGS 由各个进程的本地状态构成,与系统的结构相对应,一个环境特性谓词 $\Phi$ 是一个逻辑表达式,其构成可以逐层定义如下:

- 环境特性描述谓词:$\Phi::=Pos(\Phi_{GSE})|Def(\Phi_{GSE})$.
- GSE 谓词:$\Phi_{GSE}::=\Phi^1_{CGS}(g_{i_1})\wedge\Phi^2_{CGS}(g_{i_2})\wedge\ldots\wedge\Phi^m_{CGS}(g_{i_m}),g_{i_1}\prec g_{i_2}\prec\ldots\prec g_{i_m}$.
- CGS 谓词:$\Phi_{CGS}::=\Phi^1_{LP}(s_{k_1})\wedge\Phi^2_{LP}(s_{k_2})\wedge\ldots\wedge\Phi^n_{LP}(s_{k_n})$.
- 本地谓词:$\Phi_{LP}$::=local predicate over some $P_i$.

下面我们自底向上地、逐层地讨论一个环境特性谓词的构成.

1.2.1 本地谓词$\Phi_{LP}$

本地谓词$\Phi_{LP}$是仅涉及某个进程本地信息,不用与其他进程交互即可确定其取值的谓词.在我们的描述框架中,本地谓词是原子构成模块.虽然本地谓词的取值仅仅依赖环境设施本地的信息,但这并不意味着获取本地谓词的取值是简单的.用户可能设定复杂的本地谓词,它的取值需要通过对各种本地的环境信息进行运算而得到.并且,本地谓词的取值往往是不稳定的,因而环境设施需要持续监控各种环境信息,获得本地谓词不同时间不同取值构成的流.我们将在第 2.2 节的系统设计中详细讨论如何获取本地谓词的值.

1.2.2 CGS 谓词$\Phi_{CGS}$

CGS 谓词是指定义于一个 CGS 之上的谓词,它是多个本地谓词之间通过布尔运算而得到的.

例如,$\Phi_{CGS}=\Phi_{LP}^1\wedge\Phi_{LP}^2$是一个 CGS 谓词,它由本地谓词$\Phi_{LP}^1$="房间 $A$ 的温度高于 10°C"和$\Phi_{LP}^2$="房间 $B$ 的温度低于 20°C"通过合取运算构成.构成 CGS 谓词的布尔运算包括合取、析取、非、蕴含等,其中,合取运算是核心,即通过其他布尔运算获得的 CGS 谓词的检测均可以化归为合取 CGS 谓词检测问题[11].为了便于讨论,本文仅考虑合取 CGS 谓词.

1.2.3 GSE 谓词$\Phi_{GSE}$

虽然 CGS 谓词可以描述环境在某个状态下的特性,但是它无法描述涉及到环境在某段时间内动态变化的特性.例如,基于 CGS 谓词我们可以描述环境在某一状态下的特性"用户 1 在房间 $A$∧用户 2 在房间 $B$",但是无法描述环境在一段时间内的变化"用户 1 进入了房间 $A$,又进入房间 $B$,再进入房间 $C$".而这类环境特性的刻画对于开放环境下的软件系统非常重要.为此,我们以 CGS 谓词为基础来定义 GSE 谓词,其含义是随着时间的推移,环境在不同的状态下表现出不同的特性.其中,我们用 CGS 谓词描述环境在不同状态下的特性,用 CGS 间的偏序关系$\prec$来描述不同 CGS 间的先后顺序.

1.2.4 模态 $Pos(\Phi_{GSE})$和 $Def(\Phi_{GSE})$

根据第 1.1.3 节的讨论,一般情况下,我们仅能确定系统的实际执行是一组 GSE 集合中的某一条 GSE,因而仅仅定义 GSE 谓词是没有实际意义的.为此,我们针对所有可能的 GSE 组成的集合引入两种模态(modality)[11]:

- $Pos(\Phi_{GSE})$:在系统执行过程中存在一条 GSE,使得$\Phi_{GSE}$成立.
- $Def(\Phi_{GSE})$:对于系统执行过程中每一条 GSE,$\Phi_{GSE}$均成立.

模态 $Pos(\Phi_{GSE})$和 $Def(\Phi_{GSE})$的引入,解决了 happen-before 关系仅仅是一个偏序关系所带来的不确定性,使得针对某条 GSE 设定的谓词有了实际的含义.

### 1.3 讨 论

在我们所提出的环境特性描述框架中,本地谓词$\Phi_{LP}$是设定一个环境特性谓词的基础,但并不是所有谓词都可以基于$\Phi_{LP}$而设定.例如"房间 $A$ 中人的个数比房间 $B$ 中人的个数多 2 个"这样的谓词并不能直接在我们的框架下描述,但是很多情况下,如果我们知道到谓词的取值满足的一定的限制,我们仍然可以将上述谓词化归为本地谓词的布尔运算,进而纳入到我们的描述框架中.在上面的例子中,如果我们知道房间中人的数目最多为 3 个,则可以将谓词"房间 $A$ 中人的个数比房间 $B$ 中多两个"等价地转化为"(房间 $B$ 中没有人∧房间 $A$ 中有两个人)∨(房间 $B$ 中有一个人∧房间 $A$ 中有 3 个人)".

## 2 异步环境特性感知中间件平台

上述异步环境特性描述框架讨论如何刻画应用关心的环境特性.当应用通过该框架设定其关注的环境特性后,随之而来的关键问题是如何在异步环境下实现用户设定的各种环境特性谓词的检测.为此,本节给出面向开放环境的环境特性谓词检测中间件平台 MIPA.我们首先概述 MIPA 的系统结构,其次讨论 MIPA 各部分的详细设计,最后给出基于 MIPA 进行环境特性谓词检测的基本流程(本文的工作对应于 MIPA 系统 0.2 版,其源码参见 http://mipa.googlecode.com).在第 3 节,我们通过详细的案例分析来考察 MIPA 平台的设计.

### 2.1 MIPA系统结构概述

MIPA的总体结构如图2所示,该中间件平台的核心是环境特性谓词检测.根据第1节提出的描述框架,环境特性谓词检测主要包括两个部分:

- 本地谓词值的获取.这部分是谓词检测的基础,我们采用 Event-Condition-Action(ECA)机制来持续地监控本地谓词的取值,其详细设计见第2.2节.
- 谓词检测算法的设计与实现.这部分是谓词检测的核心,我们根据环境特性描述框架所揭示的谓词结构给出谓词检测算法框架,并针对不同的场景讨论谓词检测算法框架的具体实现与性能优化,其详细设计见第2.3节.

谓词检测的实际部署与运行还依赖于若干系统构件的支持,包括 Predicate broker,Resource broker,Event register,Context collector,Networking 等.我们将在第 2.4 节讨论各个系统构件的设计,并在第 2.5 节给出基于MIPA平台进行环境特性谓词检测的基本流程.

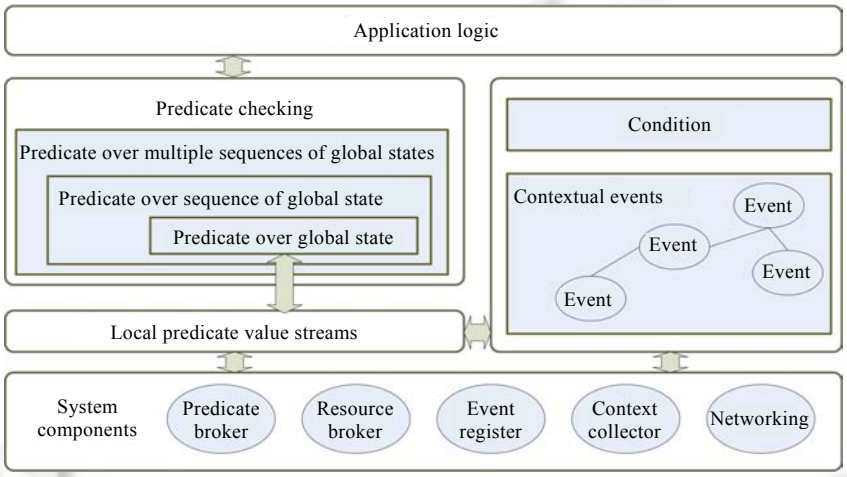



Fig.2 System architecture of the middleware infrastructure MIPA

图2 MIPA中间件平台系统结构

### 2.2 基于ECA机制的环境上下文获取

环境设施通过 ECA 机制获得本地谓词取值流,并将本地谓词取值变化的时刻所对应的逻辑时间戳通知谓词检测模块.ECA机制主要包括3个部分:

- Event listener.该模块由谓词检测模块充当.为了监听到谓词检测所必须的本地谓词取值流,Event listener模块向Event register系统构件查询与本地谓词相关的环境上下文事件,并根据Event register返回的结果向Event source模块注册与自己相关的事件.
- Event source.该模块周期性地(通过context collector系统构件)获取各种环境上下文事件,并将这些事件传递给Event condition.
- Event condition.该模块接收Event source输入的环境上下文事件,并计算本地谓词的取值,过滤出Event listener关注的环境上下文事件.

### 2.3 环境特性谓词检测算法设计框架

在获得本地谓词的取值流之后,谓词检测可以看成是在系统执行所形成的格上匹配上层应用指定模式的问题.系统执行的格由系统中各个环境设施上状态的逻辑时钟时间戳来描述,待匹配的模式由环境特性描述谓词描述.

对于 $Pos(\Phi_{GSE})$的检测,根据第 1.2 节的环境特性描述框架,检测到 $Pos(\Phi_{GSE})$成立的关键是在系统执行的格上找到一条 GSE,使得构成该 GSE 的每个 CGS 均满足指定的 CGS 谓词.对于谓词 $Def(\Phi_{GSE})$,其检测的关键是遍历系统执行的格中所有可能的 GSE,检查其是否满足 GSE 谓词 $\Phi_{GSE}$.为了检测上述两类模态下的各种谓词,我们都需要枚举系统执行所有可能的 GSE.不同的是,对于 $Pos(\Phi_{GSE})$谓词,我们仅需发现应用指定的谓词满足即可;而对于 $Def(\Phi_{GSE})$谓词,我们则需要覆盖所有可能的 GSE,并且定其均满足应用指定的谓词.

一般来讲,谓词的检测问题是 NP-hard 的[10],具体而言,其时间复杂度是 $O(kn)$.这里,$k$ 是单个进程上状态数的上界,$n$ 是环境设施的数目.但是在实际的应用中,我们往往可以从多个维度对谓词检测算法进行优化.谓词检测算法优化的本质是在检测的效果和检测开销之间作权衡,具体而言:

- 谓词描述能力 vs. 谓词检测开销.我们可以通过限制谓词的描述能力,并在检测过程中充分利用谓词的结构特征来降低谓词检测的开销.例如,我们将环境特性谓词限定于单个 CGS,考察 $Pos(\Phi_{LP}^1 \wedge ... \wedge \Phi_{LP}^n)$ 和 $Def(\Phi_{LP}^1 \wedge ... \wedge \Phi_{LP}^n)$ 谓词,则可以在遍历系统执行格的过程中避免回溯,因而可以将算法的时间复杂度降低到多项式的[11,12].我们将在第 3 节的案例分析中给出针对 $Def(\Phi_{LP}^1 \wedge ... \wedge \Phi_{LP}^n)$ 的一个具体检测算法的实现.
- 谓词检测准确性 vs. 谓词检测开销.在许多应用场景中,用户往往愿意接受一定程度的不精确性,以换取检测开销的明显降低[13,14].因而,另一种谓词检测优化方法是在应用可接受的范围内,通过放弃遍历系统执行的格中的某些部分来降低谓词检测开销[15].

### 2.4 系统构件设计

为了支持谓词检测的实现,MIPA 通过一组系统构件(system component)提供必要的功能支撑:

- Predicate broker.谓词检测所涉及的显式化环境设施通过异步消息传递进行通信,Predicate broker 屏蔽了显式化环境设施与 MIPA 中间件平台间交互的细节.
- Resource broker.谓词检测最终要依赖 ECA 机制中 Event source 模块输入的上下文事件,而上下文事件是由各类 Context collector 产生的.Resource broker 构件完成上下文事件与 Context collector 间的映射,我们用 XML 文件描述系统中的 Context collector.Context collector 描述文件的 DTD 如图 3 所示.
- Event register.该构件完成了本地谓词到环境上下文事件的映射.基于该构件,ECA 机制中的 Event listener 模块可以查询本地谓词所涉及的事件,并根据该构件返回的结果向 Event source 注册.
- Context collector.该构件封装了各种环境上下文收集设备,屏蔽设备之间的差异,向 ECA 机制中的 Event source 模块提供了环境上下文原始数据.
- Networking.该构件封装了有线/无线网络通信的细节,是各个计算实体间交互的基础.

```
〈!ELEMENT sensor(name,id,class,location,SensorType,ValueType)〉
〈!ELEMENT name(#PCDATA)〉
〈!ELEMENT id(#PCDATA)〉
〈!ELEMENT class(#PCDATA)〉
〈!ELEMENT location(#PCDATA)〉
〈!ELEMENT SensorType(#PCDATA)〉
```

Fig.3 DTD of resource description

图 3 资源描述的 DTD

### 2.5 基于MIPA平台进行环境特性谓词检测的基本流程

环境感知应用通过 MIPA 平台进行异步环境特性检测的基本流程是:首先,应用层根据自身所关注的环境特性设定环境特性描述谓词,并使用 XML 表述.环境特性谓词 XML 的 DTD 如图 4 和图 5 所示;MIPA 平台接收到用户设定的环境特性描述谓词后,解析该谓词,并通过 Predicate broker 找到检测该谓词所涉及的环境设施,并向其发送相应的本地谓词;每个环境设施拿到本地谓词后,首先通过 Event register 部件注册相关的环境上下文事件, ECA 模块通过 Resource broker 上的资源信息将环境上下文事件映射到相应的 Context collector.这

样,Event source 模块即可获得环境上下文事件;谓词检测模块通过 ECA 机制持续监控本地谓词的取值,并根据本地谓词的取值进行谓词检测.

```
〈!ELEMENT predicate(prefix,GSE)〉
〈!ELEMENT prefix EMPTY〉
〈!ATTLIST prefix value (def|pos) #REQUIRED〉

〈!ELEMENT GSE(CGS,global)+〉
〈!ATTLIST GSE value (conjunction|disjunction) #REQUIRED〉

〈!ELEMENT CGS(LP)+〉
〈!ATTLIST CGS value (conjunction|disjunction) #REQUIRED〉

〈!ELEMENT global(#PCDATA)〉
```

Fig.4 DTD of environment predicate (global predicate)

图 4 环境特性谓词 XML 描述 DTD(全局谓词部分)

```
〈!--local predicate definition--〉
〈!ELEMENT LP(formula)〉
〈!ELEMENT formula((quantifier,formula)|(formula,binary,formula)|(unary,formula)|atom)〉
〈!--quantifier contains symbol and variable--〉
〈!ELEMENT quantifier(#PCDATA)〉
〈!ATTLIST quantifier value (universal|existential) #REQUIRED〉

〈!ELEMENT binary EMPTY〉
〈!ATTLIST binary value (conjunction|disjunction|impliction) #REQUIRED〉

〈!ELEMENT unary EMPTY〉
〈!ATTLIST unary value (not) #IMPLIED〉

〈!--operator:==, great-than, not-great-than, less-than, not-less-than, etc. --〉
〈!ELEMENT atom EMPTY〉
〈!ATTLIST atom
   operator CDATA #REQUIRED
   name CDATA #REQUIRED
   value CDATA #REQUIRED〉
```

Fig.5 DTD of environment predicate (local predicate)

图 5 环境特性谓词 XML 描述 DTD(本地谓词部分)

## 3 案例分析——基于 MIPA 平台的异步环境并发上下文活动检测

本节通过一个案例分析来考察 MIPA 中间件平台的设计.我们考察“并发”这一重要的环境特性,研究如何基于我们所提出的环境特性描述框架来刻画这一特性,并研究如何基于 MIPA 平台实现相应的谓词检测算法.

### 3.1 应用场景

环境上下文活动(contextual activity)之间的并发关系是开放环境下应用关心的重要环境特性之一.例如,在很多关键任务(mission-critical)应用中,系统需要时刻监控环境的变化.系统关心的上下文活动往往不是本地的,而是多个并发的本地活动组成的全局活动,例如:

- 在机房监控应用中,独立地看,“机房中没有人”和“机房温度超过 36°C”这两个活动都是正常的,应用并不关心,但是,当这两个活动并发时,很可能意味着机房中出现了意外.因而应用关心“机房中没有人,且(同时)机房温度超过 30°C”这一全局上下文活动.
- 在基于 RFID 的用户位置识别中,“用户在房间 *A*”和“用户在房间 *B*”这两个活动独立地看都是正常的,但是,如果这两个活动同时发生,则意味着 RFID 读取的数据产生了错误,因为同一个人不可能同时在两个地方出现.

虽然在同步的分布式系统中,或者当系统中有统一的全局时钟时,我们可以很容易地检测两个上下文活动是否并发,但是在“真”分布的开放环境下,因为环境设施间没有统一的全局时钟,环境设施间基于异步消息传递进行交互,并且环境设施间往往有各不相同的数据更新频率,所以我们需要重新考察如何检查上下文活动之间的并发.开放环境的分布性与异步性是问题的关键所在,感知设备制造技术与网络通信技术的改进无助于解决异步环境下的并发活动检测问题.

基于上面的讨论,我们需要显式地要求环境设施之间进行消息传递,并基于消息传递建立起来的 happen-before 关系来检测上下文活动之间的并发关系.在上面的例子中,当我们需要检测“机房中没有人,且(同时)机房温度超过 30°C”这一全局活动时,为了检测两个本地活动间的并发关系,我们需要环境设施在检测到本地活动发生时,向全局活动中涉及到的其他环境设施发送消息,以建立检测全局活动所必需的 happen-before 关系.

我们首先基于所提出的环境特性描述框架设定并发谓词来刻画开放环境下上下文活动之间的并发关系,然后改进 Strong Conjunctive Predicate(SCP)算法[11],使其主动在多个环境设施之间进行消息传递,以建立后续谓词检测所必需的 happen-before 关系.最后,我们基于 MIPA 平台实现改进后的 SCP 算法,以验证 MIPA 平台的设计.

### 3.2 并发活动描述与资源配置

基于我们的描述框架,$n$ 个上下文活动并发这一环境特性可以表示为

$$\Phi = Def(\Phi_{LP}^{1}(s_{k_1}) \wedge \Phi_{LP}^{2}(s_{k_2}) \wedge ... \wedge \Phi_{LP}^{n}(s_{k_n})),$$

其中,当某个本地活动在环境设施 $P_k$ 上发生时,其对应的本地谓词 $\Phi_{LP}^{k}$ 值为 True.并发活动的描述不涉及多个 CGS 组成的 GSE,因而描述框架中的 GSE 谓词退化成单个 CGS 上的谓词.在上面的例子中,“机房中没有人,且机房温度超过 30°C”这一环境特性谓词,基于我们的描述框架可表示为

$$\Phi^{cur} = Def(\Phi_{LP}^{1} \wedge \Phi_{LP}^{2}),$$

$$\Phi_{LP}^{1} = Greater(temperature, 30),$$

$$\Phi_{LP}^{2} = Contain(RFID - Reader, tag\_00101).$$

按照图 4 和图 5 定义的 DTD,上述谓词的 XML 描述如图 6 所示.其中,本地谓词 $\Phi_{LP}^{1}$ 和 $\Phi_{LP}^{2}$ 的取值由部署于房间中的温度传感器和 RFID 阅读器获取的环境数据来获得.在 MIAP 平台的 Resource broker 上,$\Phi_{LP}^{2}$ 和房间中 RFID 阅读器的映射关系如图 7 所示(温度传感器的情况与之类似).

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈!DOCTYPE predicate SYSTEM "predicate.dtd"〉
〈predicate type="SCP"〉
  〈prefix value="def"/〉
  〈GSE value="conjunction"〉
    〈CGS value="conjunction"〉
      〈LP〉
        〈formula〉
          〈atom operator="great-than" name="temperature" value="30"/〉
        〈/formula〉
      〈/LP〉
      〈LP〉
        〈formula〉
          〈atom operator="contain" name="RFID" value="tag_00001"/〉
        〈/formula〉
      〈/LP〉
    〈/CGS〉
    〈global〉g〈/global〉
  〈/GSE〉
〈/predicate〉
```

Fig.6 XML of a concurrency predicate

图 6 一个并发谓词的 XML 描述

```
〈?xml version="1.0" encoding="UTF-8"?〉
〈!DOCTYPE sensor SYSTEM "sensor.dtd"〉
〈sensor〉
   〈name〉RFID_A〈/name〉
   〈id〉RFID_ROOM_A〈/id〉
   〈class〉cn.edu.nju.ics.mopec.sensors.RFID〈/class〉
   〈location〉ROOM_A〈/location〉
   〈SensorType〉RFID〈/SensorType〉
   〈ValueType〉String〈/ValueType〉
〈/sensor〉
```

Fig.7 Resource description of RFID reader in the room

图 7 房间中 RFID 阅读器的资源描述

### 3.3 谓词检测

为了在异步环境下检测并发的上下文活动,我们使用逻辑向量时钟[10,11]记录系统的执行.首先,我们考察本地谓词的取值.我们将本地谓词取值为 *True* 的间隔记为一个时间段(interval),时间段的初始和终止的向量时钟时间戳分别记为 *Interval.lo* 和 *Interval.hi*.并发活动谓词 $\Phi^{ur}$ 为 *True*,即本地谓词同时为 *True*,等价于本地谓词对应的区间在时间轴上有交叠.多个时间段的交叠等价于其头尾之间的 happen-before 关系:

$$(I_j.lo \rightarrow I_k.hi) \wedge (I_k.lo \rightarrow I_j.hi), \forall 1 \leqslant j < k \leqslant n \quad (1)$$

举例而言,对于两个进程上的本地谓词对应的时间段 $I_1$ 和 $I_2$,其必然重叠等价于$(I_1.lo \rightarrow I_2.hi) \wedge (I_2.lo \rightarrow I_1.hi)$,如图 8 所示.

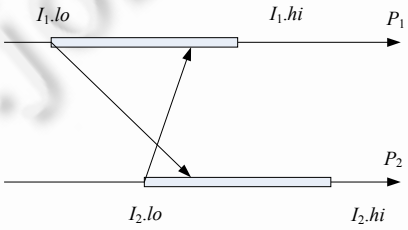



Fig.8 Overlapping intervals

图 8 重叠的时间段

为了检测到本地谓词对应的时间段之间的重叠,SCP 算法[11]分为环境设施部分和谓词检测部分.在各个环境设施 $P_i(1 \leqslant i \leqslant n)$上,为了建立各个时间段的头尾之间的 happen-before 关系,我们让每个 $P_i$ 显式地在 Interval 的开头向其他进程发送消息.谓词检测模块为每个环境设施 $P_i$ 建立时间戳队列,用于存储各个 $P_i$ 持续发送来的向量时钟时间戳 *Interval.lo* 和 *Interval.hi*.每当某个队列头部的时间戳发生变化时,检测算法则比较所有队列头部的时间戳.若每个队列头部的时间戳满足公式(1),则成功检测到并发活动;若公式(1)不能满足,则将不能与最近更新的时间戳建立公式(1)所要求的 happen-before 关系的时间戳删除.若这一删除过程导致某些队列头部的时间戳被更新,则检测算法递归地进行上述的检测与删除时间戳的过程.

并发谓词检测过程不断删除无用的时间戳,以降低遍历系统执行的格的开销.因而,上述谓词检测的时间复杂度是 $O(n^2p)$.其中,$n$ 是环境设施的数目,$p$ 是每个时间戳队列长度的上限.每个环境设施的消息复杂度是 $O(p)$.

### 3.4 实验结果与分析

在我们的实验验证中,我们模拟第 3.1 节讨论的机房监控的场景,具体谓词及其 XML 描述见第 3.2 节.本地活动的发生符合泊松过程,其平均间隔为 15 分钟.每个本地活动的持续间隔符合指数分布,平均为 10 分钟.我们首先考察环境的异步性的影响,具体而言,传感器节点更新环境数据的时间间隔以及无线通信的延迟是决定环境异步性的主要因素.其次,我们考察上下文活动的持续时间的影响.

在第 1 个实验中我们发现,环境的异步性对并发活动检测的准确率有较大的影响.随着更新延迟的增加,并

发活动检测的准确率近似线性地降低,如图 9 和图 10 所示.

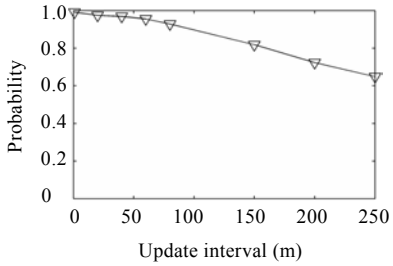



Fig.9 Effect of tuning context update interval

图 9 调整上下文更新间隔的影响

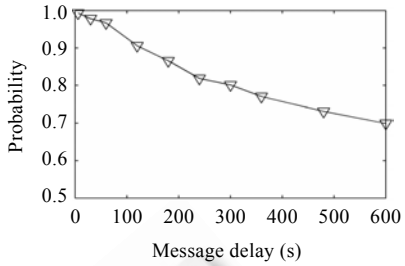



Fig.10 Effect of tuning message delay

图 10 调整消息延迟的影响

在第 2 个实验中我们发现,随着上下文活动持续时间的变化,并发活动检测的准确率基本保持平稳,在持续时间为 20 分钟时略有抖动,如图 11 所示.这说明在环境的异步性基本保持不变的情况下,检测的准确率受上下文活动发生的频率影响较小;并且当环境的异步性不是特别突出时,并发活动的检测准确率较高.

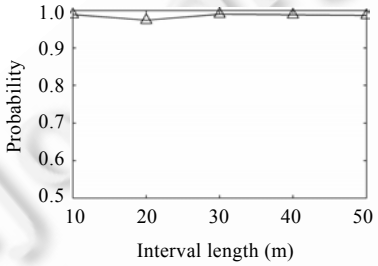



Fig.11 Effect of tuning length of local contextual activity

图 11 调整本地上下文活动持续时间的影响

总体而言,通过上面的实验,一方面我们验证了计算环境的异步性对并发上下文活动检测准确率的影响,而环境的异步性正是本文工作的基本出发点;另一方面,我们也验证了当环境的异步性处于一个合理的范围之内时,并发上下文活动检测算法的准确率较高.

## 4 相关工作

如何应对开放、动态、难控的网络环境,构建可信的软件系统,是传统软件技术面临的挑战.网构软件是应对这一挑战的技术框架之一,它提出了显式化环境与动态化应变的途径.但该技术框架对开放环境的显式化建模与处理尚停留在理念阶段,主要依赖传统软件技术来实现[1,2].在环境显式化技术方面,普适计算领域尤其是上下文感知计算领域有较多的研究.这方面的研究主要采取两种技术途径:一种是以 Lime[16],Egospaces[17]为代表的,支持上下文感知应用开发与运行的上下文感知中间件技术;另一种是以 CARISMA[18],CARMEN[19],MobiPADS[20]等为代表的自省(reflective)技术.面对开放环境,上述环境显式化技术的主要不足是,其非形式化的环境处理方式以及上述技术基于同一全局时钟的假设无法应对开放环境“真”分布性的挑战.

产生于分布程序调试的谓词检测技术是应对开放环境“真”分布性的挑战的可行途径之一.具体而言,文献[21]采用了一种蛮力(brute force)搜索机制,遍历系统执行的格以检测谓词是否成立,其时间复杂度是指数级的.Garg 等人着重研究单个 CGS 上的谓词检测,证明了合取谓词检测是关键问题,并提出相应的检测算法[11,12].针对 CGS 谓词将谓词检测的时间复杂度降低到多项式时间,Babaoglu 等人提出 GSE 上的谓词[8],Kshemkalyani 对系统执行的 GSE 集合的模态进行了更细致的划分[22].上述工作并未讨论其在构建环境感知系统中可能的应用,并且缺乏一个通用的平台支持各种谓词检测机制的实现.而本文的工作正是着眼于提供这样一个中间件支撑

平台,支持环境感知应用的构建.

## 5 结论与未来的工作

本文着重研究如何在开放环境下准确感知并恰当应对环境及其变化,支持可信软件系统的构建.我们首先提出了一个开放环境特性描述框架.该框架的核心是谓词检测技术.它支持便捷地、形式化地描述开放环境的各种特性,包括那些既有技术不能处理的时序特性,并支持高效的环境特性感知机制的实现.其次,我们开发了开放环境特性感知中间件平台 MIAP,并通过案例分析展示了如何基于 MIPA 平台高效地感知环境特性,支持可信软件系统的构建.

目前,我们的研究主要集中在环境特性的检测上.今后我们将研究如何在检测到环境特性不被满足时,触发环境的改变,主动保证环境特性被满足.我们还将通过更丰富的案例分析,全面考察 MIPA 平台的实现.

**References**:

[1] Yang FQ, Lu J, Mei H. Technical framework for Internetware: An architecture centric approach. Science in China Series F: Information Sciences, 2008, 51(6):610−622. [doi: 10.1007/s11432-008-0051-z]

[2] Lu J, Ma XX, Tao XP, Cao C, Huang Y, Yu P. On environment-driven software model for Internetware. Science in China Series F: Information Sciences, 2008,51(6):683−721. [doi: 10.1007/s11432-008-0057-6]

[3] Lamsweerde AV. Requirements engineering: From craft to discipline. In: Harrold MJ, Murphy GC, eds. Proc. of the Foundation of Software Engineering. New York: ACM Press, 2008. 238−249.

[4] Xu C, Cheung SC. Inconsistency detection and resolution for context-aware middleware support. In: Wermelinger M, Gall H, eds. Proc. of the Foundation of Software Engineering. New York: ACM Press, 2005. 336−345.

[5] Xu C, Cheung SC, Chan WK. Incremental consistency checking for pervasive context. In: Osterweil LJ, Rombach HD, Soffa ML, eds. Proc. of the Int'l Conf. on Software Engineering. New York: ACM Press, 2006. 292−301.

[6] Lamport L. Time, clocks and the ordering of events in a distributed system. Communications of the ACM, 1978,21(7):558−565. [doi: 10.1145/359545.359563]

[7] Sama M, Rosenblum DS, Wang Z, Elbaum S. Model-Based fault detection in context-aware adaptive applications. In: Harrold MJ, Murphy GC, eds. Proc. of the Foundation of Software Engineering. New York: ACM Press, 2008. 261−271.

[8] Babaoglu O, Raynal M. Specification and verification of dynamic properties in distributed computations. Journal of Parallel and Distributed Computing, 1995,28(2):173−185.

[9] Babaoglu O, Marzullo K. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. 2nd ed., New York: ACM Press, 1993. 55−96.

[10] Schwarz R, Mattern F. Detecting causal relationships in distributed computations: In search of the holy grail. Distributed Computing, 1994,7(3):149−174. [doi: 10.1007/BF02277859]

[11] Garg VK, Waldecker B. Detection of strong unstable predicates in distributed programs. IEEE Trans. on Parallel and Distributed Systems, 1996,7(12):1323−1333. [doi: 10.1109/71.553309]

[12] Garg VK, Waldecker B. Detection of weak unstable predicates in distributed programs. IEEE Trans. on Parallel and Distributed Systems, 1994,5(3):299−307. [doi: 10.1109/71.277788]

[13] Huang Y, Cao JN, Wang ZJ, Jin BH, Feng YL. Achieving flexible cache consistency for pervasive Internet access. In: Porta PL, ed. Proc. of the Int'l Conf. on Pervasive Computing and Communications. Washington: IEEE Computer Society Press, 2007. 239−250.

[14] Huang Y, Cao JN, Jin BH. A predictive approach to achieving consistency in cooperative caching in MANET. In: Jia XH, ed. Proc. of the Int'l Conf. on Scalable Information Systems. New York: ACM Press, 2006. 1−8.

[15] Huang Y, Ma X, Tao XP, Cao JN, Lu J. A probabilistic approach to consistency checking for pervasive context. In: Guo ML, ed. Proc. of the Int'l Conf. on Embedded and Ubiquitous Computing. Washington: IEEE Computer Society Press, 2008. 387−393.

[16] Murphy AL, Picco GP, Roman GC. Lime: A coordination model and middleware supporting mobility of hosts and agents. ACM Trans. on Software Engineering and Methodology, 2006,15(3):279−328. [doi: 10.1145/1151695.1151698]

[17] Julien C, Roman GC. Egospaces: Facilitating rapid development of context-aware mobile applications. IEEE Trans. on Software Engineering, 2006,32(5):281−298. [doi: 10.1109/TSE.2006.47]

[18] Capra L, Emmerich W, Mascolo C. Carisma: Context-Aware reflective middleware system for mobile applications. IEEE Trans. on Software Engineering, 2003,29(10):929−945. [doi: 10.1109/TSE.2003.1237173]

[19] Bellavista P, Corradi A, Montanari R, Stefanelli C. Context-Aware middleware for resource management in the wireless Internet. IEEE Trans. on Software Engineering, 2003,29(12):1086−1099. [doi: 10.1109/TSE.2003.1265523]

[20] Chan ATS, Chuang SN. Mobipads: A reflective middleware for context-aware mobile computing. IEEE Trans. on Software Engineering, 2003,29(12):1072−1085. [doi: 10.1109/TSE.2003.1265522]

[21] Cooper R, Marzullo K. Consistent detection of global predicates. In: Miller B, McDowell C, eds. Proc. of the ACM Workshop on Parallel and Distributed Debugging. New York: ACM Press, 1991. 167−174.

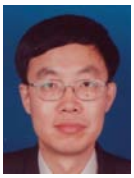
[22] Kshemkalyani AD. A fine-grained modality classification for global predicates. IEEE Trans. on Parallel and Distributed Systems, 2003,14(8):807−816. [doi: 10.1109/TPDS.2003.1225059]

**黄宇**(1982－),男,江苏淮安人,博士,讲师,CCF 会员,主要研究领域为软件方法学,分布式计算理论,移动普适计算.

**余建平**(1986－),男,硕士生,主要研究领域为软件工程.

**马晓星**(1975－),男,博士,教授,主要研究领域为软件工程,软件方法学,软件体系结构.

**陶先平**(1970－),男,博士,教授,CCF 高级会员,主要研究领域为移动普适计算,软件工程,软件方法学.



**吕建**(1960－),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,软件方法学.