面向网构软件的环境显式化技术

吕建^{①②*}, 马晓星^{①②}, 陶先平^{①②}, 黄宇^{①②}, 余萍^{①②}, 许畅^{①②}

① 南京大学计算机软件新技术国家重点实验室, 南京 210046

② 南京大学计算机软件研究所, 南京 210046

* 通信作者. E-mail: lj@nju.edu.cn

收稿日期: 2012-08-23; 接受日期: 2012-11-09

国家重点基础研究发展计划 (批准号: 2009CB320702) 和国家自然科学基金创新群体 (批准号: 61021062) 资助项目

摘要 网构化软件处于开放、动态、难控的环境之中, 需能感知并适应环境的变化, 以持续提供用户满意的服务. 传统软件构造技术缺乏对环境感知和适应的系统支持. 本文在前期工作基础上, 提出一种面向网构软件的结构, 通过为软件系统配备先验的环境模型和环境规约来实现环境处理的显式化. 进而, 介绍了我们使用这种方式来处理开放环境之分布异步性、感知信息的不一致性以及环境中人类用户行为特性等的若干具体技术进展. 而后, 讨论了一种先验环境模型及环境规约的开发方法原理, 给出了相应的环境驱动逻辑的静态表达与运行实现机制. 该方法和机制可较好地支持环境处理与应用业务逻辑间的关注分离.

关键词 网构软件 环境显式化 分布性 一致性 自适应

1 引言

当前越来越多的软件系统需工作于开放、动态、难控的互联网和物联网环境之中. 此类软件系统应能感知外部环境的动态变化, 并随着这种变化按照功能指标、性能指标和可靠性指标等进行静态 (离线) 的调整和动态 (在线) 的演化, 以使系统具有尽可能高的用户满意度^[1]. 网构软件 (Internetware) 是我国学者提出的一种面向开放网络环境的新型软件范型 (paradigm) 设想^[2~4]. 与传统软件系统相比, 网构化的软件系统更强调系统的自主性、协同性、适应性、演化性和可信性等开放环境所要求的能力特征.

在前期的工作^[1]中, 我们提出了一种以“环境信息显式化、互动方式层次化、系统结构可演化”为特征的面向网构软件的环境驱动模型, 并以环境上下文建模、管理与使用为核心, 研究了一系列相关的关键支撑技术. 然而, 从软件工程的角度, 如何系统地支持环境的“显式化”, 以及以何种方式在实现系统行为自动适应当前环境情况的同时, 维持环境处理¹⁾机制与应用业务逻辑间的关注分离, 仍需要进一步的研究.

在诸如结构化和面向对象这样的传统的软件开发方法中, 没有针对系统所处环境动态变化的专门支持. 使用这些方法开发软件, 通常需要对系统运行时将要面对的环境做预先的约定, 而这些约定被

1) 本文讨论的是软件方法. 在软件的范畴内, 所处理的对象是信息

作为既定的前提假设, 在需求分析、架构设计和系统实现的各个环节被隐式地使用. 这种方式对于相对稳定可控的环境下的系统是可行的, 但不利于实现网构化软件对开放环境的适应. 其原因在于, 若要使用这些传统方法来应对变化的环境, 就只能将环境事件当作系统的输入的一部分. 这原则上要求系统开发时完全事先确定其变化的范围, 这在开放环境下难以完全做到; 与此同时, 这种把环境当输入的做法容易使得环境适应逻辑与系统的业务逻辑过度耦合纠缠, 导致不必要的系统复杂性, 也不利于环境适应逻辑和应用业务逻辑各自的复用和演化.

目前已有一些工作注意到这个问题. 比如, 有学者提出了面向情境的编程方法 (context-oriented programming, COP), 即设计一种新的语法, 将情境相关的行为包含到编程语言引入的新成分中, 如 ContextL^[5] 和 ContextJ^[6,7]. ContextL 和 ContextJ 使用“层” (layer) 将情境相关的行为和属性与应用程序其他的行为隔离开来, 通过一个层激活和层抑制的机制来控制层的组合进而改变应用的行为. 但是这类工作没有提供一个完善的软件方法框架, 从而难以使用; 并且这类技术更适合处理情境的出现或消失, 而较难处理因情境值变化而需要调整行为的情况. 在感知技术和具体应用方面, 尽管在普适计算/信息物理融合系统等领域当前已有大量的研究涉及各种具体的环境上下文感知技术, 如以 TelosB^[8] 为代表的无线传感器节点集成方案、以气象数据采集^[9]、温室气体监测^[10] 为代表的大气环境感知系统, 以 TeenyLime^[11], MiLan^[12] 和 Impala^[13] 为代表的普适应用中间件软件技术等. 但在如何认识环境、感知环境的软件方法、模型和技术方面仍有待进一步研究.

本文提出一种基于先验环境认知框架的环境显式化技术. 一个具有环境适应能力的网构化软件系统除了其应用业务逻辑之外, 还需要显式地提供刻画环境特征的抽象环境模型以及表达当前应用的环境关注的规约. 各种传感设备及相应软件基础设施依据环境模型感知具体环境信息, 进而依据环境规约检测相关环境动态变化, 以此驱动系统的演化以适应环境. 我们将具体展示如何以此理念为指导, 来应对开放动态环境的分布性异步性、感知信息的不一致性以及环境中人类用户行为等开放环境的关键特性. 同时, 在环境模型和环境规约的构造方法方面, 我们给出了一种以计算实体之活动为中心、动态组织的主动上下文建模方法; 在具体的环境驱动静态表达和运行实现机制方面, 提出了一种面向环境的软件构造技术, 较好地实现了环境适应逻辑与应用业务逻辑的关注分离.

2 环境显式化软件结构

如前所述, 网构化的软件系统处于开放、动态、难控的环境之中, 并通过与其互动来实现用户的应用目标. 网构化软件系统区别于传统软件系统的一个显著特征是其不能再基于既定环境假设进行开发、部署和运行, 而需要显式地持续感知开放、动态和难控的环境, 并通过调整自身的结构和行为以适应环境的变化. 因而网构软件开发方法研究的一个难点和重点是如何系统地显式处理环境. 从软件工程的角度看, 问题不在于我们能否感知特定的环境信息, 而在于如何控制由于引入环境适应特性而导致的系统复杂性.

软件开发是构造逻辑产品的创造性活动^[14]. 为了驾驭这种创造的复杂性, 软件工程师构造软件时常常常有意地借用了或无意中暗合了哲学和逻辑的思想方法^[15,16]. 当我们说软件系统能够“感知”并“适应”环境的变化时, 实际上是将软件系统拟人化了. 因此我们这里也借用认识论的 (epistemology) 的概念和思想来讨论我们关于网构软件如何处理环境的认识和工作. 需要说明的是, 本文工作不能够也无意于成为严谨完善的哲学讨论, 我们只是借用已知的概念作为我们所讨论软件方法和思想的一种易于理解的隐喻 (metaphor).

图 1 环境显式化

Figure 1 Explicit environmental constructs

哲学家 Immanuel Kant 认为, 尽管人的一切知识始于经验, 但知识并非仅凭经验可得. 除了经验还需要与生俱来的“先验”(a priori) 的知识²⁾. 与之类比, 开放环境下网构化的软件系统, 作为一个自主的实体, 要能够适应环境的变化, 首先需要正确认识其所处的当前环境. 有关环境的信息当然来自于系统与环境交互的“经验”. 但要“正确”地解释这个“经验”有赖于“先验”的认识框架. 这个先验的认识框架决定了系统面对开发复杂难控的真实物理环境时能感知什么 (后验, a posteriori) 信息, 如何感知. 也就是说, 软件开发者需要为软件系统配备合适的先验认识框架如图 1 所示.

从实现角度说, 我们说网构化软件在静态组成结构上包含有显式的“环境”成分. 此处的“环境”并非是指其所处的物理环境, 而是软件开发者为软件系统所配备的对于物理环境的先验环境认识框架, 包括环境模型与环境规约两部分. 其中, 环境模型乃是系统开发者基于对环境本质特性的认识, 而给出的一种计算抽象. 环境模型相对独立于具体的应用. 通常一个应用系统并不需要关心当前环境模型所涉及的所有可能 (后验) 信息. 应用可通过环境规约来抽象声明其所关注的事件.

在运行时刻, 须有支持相应环境模型的环境处理基础设施来感知物理环境. 它将物理环境的运动转换到计算世界. 环境事件检测机制将从中提取环境规约所指明的应用所需事件, 以此驱动系统的演化以适应环境.

从软件工程方法的角度看, 为网构化的软件系统显式构建这种先验环境认识框架的意义在于它可从以下 3 个方面帮助我们驾驭感知和适应环境的复杂性:

1. 关注分离: 首先将环境处理从应用业务逻辑中分离出来, 其次将相对通用的环境模型与应用特定的环境规约适当分离.

2. 抽象与重用: 环境模型抽象了具体的环境感知和处理基础设施的行为, 通常针对此模型可实现较为一般的事件检测机制. 这样应用能以抽象的语言规约其关注的环境变化, 而重用整套基础设施与事件检测机制.

3. 潜在的动态扩展: 开放环境下对环境认识常常不能一次到位. 独立表达并在运行显式维护这个先验认识框架是在运行时刻扩展环境认识能力的前提和基础.

2) 严格说来, 康德的“先验”并无“与生俱来”之意, 他有意避免先天与否的争论. 本文更一般地使用“先验”这个自柏拉图以来就广为人知的概念, 把它当作一个 Metaphor, 以强调必须显式提供环境模型和环境规约

为展示这样一种“元层”结构和机制如何使用, 我们针对若干重要的环境特征, 构建了相应的先验环境认识框架, 给出了相应环境处理基础设施和高效的事件监测机制, 以更好支持具有“显式环境”的网构化软件系统的构造。

2.1 异步分布环境感知

开放环境固有其分布异步性。具体而言, 显式化环境设施天然分布的, 每个环境设施主要监控计算环境的某个局部或某个侧面的情况。其次, 各个分布的环境设施之间通过网络进行通信与协同, 环境设施间的交互具有固有的异步性。这一异步性主要源自于环境设施之间往往没有统一的全局时钟, 并且其间的通信面临不确定的延时。正如 Lamport 等人^[17]所作的类比, 开放环境感知好比对一个巨大的、动态的场景进行拍照。由于场景太大, 单个摄影师难以全面捕捉, 所以需要多个摄影师协同拍摄; 但同时由于场景处在不断动态变化之中, 而每个摄影师的拍照时间难以准确协调到同一时刻, 所以难以精确地对某个瞬间的全局场景进行拍摄。开放计算环境的上述分布异步特性对环境感知提出了挑战。

如何实现网构化软件对开放环境分布异步性的认知呢? 依据上述框架, 我们需要给出先验的环境模型和环境规约。在环境建模部分, 由于环境的分布性, 对于不同环境设施上感知到的环境事件, 我们一般无从判断其间的时序关系。但是环境设施间的交互通过消息传递来完成, 我们基于消息的发送和消息的传递之间的因果 (causality) 关系可以得到其间的先后 (happen-before) 关系^[18]。这一时间先后关系首先是一种偏序关系, 其一个重要的特征是不同的环境设施上的环境事件可能无从判断其时间先后。时间先后关系的这一偏序特性表征了分布异步环境感知所固有的不确定性, 我们可以引入格 (lattice) 理论对其进行更深入细致的刻画。其次, 我们可以进一步引入逻辑时钟机制对时间先后关系进行高效地编码与解码, 使得每个分布式的环境设施都能够基于环境时间的逻辑时间戳来有效地判断部分环境事件之间的时序关系。

在上述环境模型的基础上, 我们可以有相应的环境规约机制, 让应用可以方便、准确地表达其对分布异步环境的关注。环境特性规约的主要载体是分布谓词。分布谓词规约的基础是本地谓词, 它刻画各个环境设施本地的性质。根据上述环境建模, 我们以相对的逻辑时钟观重新审视分布异步环境的全局状态的涵义, 并基于本地谓词来规约环境全局状态谓词, 以刻画分布异步环境所具有的全局特性。基于全局谓词我们可以进一步规约时序谓词, 它刻画了分布异步环境的全局状态随时间推移的动态变化。

基于上述对环境分布异步性的先验认知, 我们可以运行时收集物理上分布、时间上异步的开放环境信息, 并有效地根据应用需求进行处理, 进而驱动软件应用进行环境感知的演化。

2.2 环境中用户行为感知

我们注意到, 在环境中存在着大量以人为代表的智能实体, 而在考虑以人为对象的环境状态时, 人的行为是一个需要重点关注的核心内容。我们发现, 以人的行为为核心的环境状态有以下突出特点:

首先, 不同的高层的环境状态表现出特定的可区分的外部状态, 具有可区分性。考虑人的行为而言, 高层的行为表现为特定的可观测的底层动作的集合或序列。举例来说, “吃饭”这一行为包括使用碗筷, 坐在桌子边等一组区分于其他行为如“看电视”的基本动作。因此, 在构建环境处理基础设施时, 就可以通过部署各类传感器来捕捉人行为的外部状态, 例如: 利用加速度传感器来感知人体各个部位的动作, 通过 RFID 设备来实现对物品使用的捕捉, 利用位置传感器来感知人体所处的位置等。同时,

在构建相应的环境模型时, 就可以利用不同行为间外部状态的可区分性, 构建不同行为的差异化模型, 由此来实现对于真实物理环境中的行为的刻画.

第二, 环境事件的发生具有交错性. 考虑人的行为, 单个人在进行活动的时候, 存在每次只完整执行一项行为的顺序情形, 但是在更多的情况下, 人会在多项行为之间进行随机的切换和并行, 体现出行为的交错性. 但是我们发现, 在真实的物理环境中, 这种交错性往往是遵循一定的规律, 例如, “吃饭”和“接电话”可能会互相切换交替执行或并行执行, 但是“洗澡”和“做饭”则很少会发生交替和并行. 由此出发, 在构建环境模型时, 我们在简单行为模型的基础上, 考虑两个简单行为通过交替或并行执行组合成复杂行为的可能性, 并依此构建出针对复杂行为的联合模型, 来描述真实用户行为的交错性.

第三, 多个环境实体之间存在复杂的交互性. 考虑人的行为时, 我们发现多名用户之间除了独立行动之外, 也可能存在着包括合作、冲突等复杂的相互影响. 而这种相互影响在真实的物理环境中, 体现为行为之间的关联关系. 因此, 我们在针对多用户行为的环境模型中, 利用条件概率来建模不同用户不同行为同时执行的概率, 由此来刻画真实用户行为之间可能存在的独立、合作和冲突的情况.

于是, 在一个针对人类用户行为识别的先验认识框架中, 环境模型应该能够表述可识别的基本动作、动作的序列关系、序列的并行与交错的现象以及行为动作的几率关联等. 而环境规约应该刻画所关注行为在这个模型下的相关特征模式.

此外, 环境事件具有突发性. 人在行动的时候并不是严格按照事先可预知的脚本来执行, 也不存在精确的规律性, 而是不断地会突发产生各种行为. 这就要求我们的环境事件监测机制能够实时的对这些突发事件进行响应. 具体而言, 要求我们实现实时的行为识别.

2.3 环境感知的一致性

最后我们讨论达到环境感知一致性的必要性及其挑战. 由于一致性问题的一般性, 我们使用一种计算机可处理的一般模型——有限元素集合上的一阶谓词逻辑公式来作为环境模型; 把用户已知的事实和规律作为环境信息必须满足的规约, 进而自动检测其一致性.

我们知道在现实中, 无论是物理环境还是用户行为的感知, 都没有完美并内在一致的技术. 这有两个层面的含义: 第一, 各种传感技术正在蓬勃发展, 它们一直处在由于不够成熟而被要求不断改进的过程中, 第一手所获取的最接近自然的环境信息反而天生不是“自然”的 (即不够真实); 第二, 即便是被感知的环境信息被各类优化技术所处理, 每个应用仍会因为自己对环境的理解不同而表现出不同的“品味”. 这就导致, 一方面被感知的环境与实际的环境总有偏差, 另一方面应用所期望的环境与实际经历的环境也有偏差, 这些偏差很难在应用设计阶段就被全面和准确地考虑, 所以极易造成应用各种不合时宜的行为甚至错误和异常.

要纠正这些偏差, 就需要在应用运行阶段达到环境感知的一致性. 然而挑战性在于, 并不存在一个神谕 (oracle) 来明确地告诉我们什么时候环境感知是错误的. 幸运的是, 也正因为环境信息来源于大自然, 它必须遵循物理规律和领域知识, 假如这些规律或知识被环境感知所违反, 则所感知到的环境信息里必然包含一致性错误 (inconsistency). 这些规律和知识包含很多例子, 比如人的步行速度不快, 所以他的位置变化必然连续 (无法大幅度跳跃); 任何事物没有分身术, 所以它无法同时出现在多处; 仓库货物必须依据流水线的设计先被登记, 然后被分派, 并且必然按照流水线的速度在规定的时间内出现在合适的地方. 这些规律和知识正好应对前述的两个层面的偏差含义, 即使用物理规律来纠正被感知的环境与实际的环境的偏差, 使用领域知识来纠正应用所期望的环境与实际经历的环境的偏差.

在我们网构软件的计算理论中, 这些物理规律和领域知识被建模为一致性约束 (consistency con-

straint), 在应用运行阶段被不断检测去发现什么时候环境的感知发生一致性错误. 这错误的检测能引发一系列研究问题, 比如开放环境动态多变, 如何做到高效检测环境信息中的一致性错误? 应用运行阶段被检测到的一致性错误无法推迟到离线状态下处理, 那么如何根据应用各自的需求去自动达到环境感知的一致性? 多个应用之间可能存在冲突的一致性策略, 该如何支持? 我们的网构软件计算理论对这些问题都进行了深入探讨, 在后面的具体技术部分我们将举例和展示说明.

下一节将分别针对这些开放环境特性讨论相应的环境感知技术. 这些特性并不局限于特定应用, 因而这些技术也是较为通用的. 而对于那些特定应用的环境感知技术, 我们也可以应用上述环境显式化结构, 但此时需要有较为系统化的办法来开发相应的先验环境模型和环境规约, 也需要有相应的软件机制来将显式化的环境和系统业务逻辑有机结合起来. 我们将在第 4 节进行有关这方面的讨论.

3 环境处理技术

在这一节里, 我们将通过回顾我们以往在分布异步环境感知、用户行为感知、一致性保障这 3 种环境感知技术方面的工作, 来展现我们的基于先验环境认识框架的环境显式化途径的应用. 这 3 种技术本身并无必然联系, 我们选择它们作为展示载体的意图在于体现开放环境所涉及问题的多样性和我们途径的可用性.

3.1 分布异步环境感知技术

环境感知的基础概念是环境事件之间的“时间先后”(happen-before)关系. 这一关系的偏序(partial order)特性有效表征了环境的分布异步性给环境感知带来的不确定性. 以时间先后关系为基础, 我们可以进一步建模分布异步环境的状态与变化, 进而提出相应的环境特性规约与检测机制

这一基于“建模—规约—检测”的分布异步环境感知框架^[19]有效地支持了环境感知应用构建过程中不同角色之间的关注分离以及环境处理机制的抽象与重用. 具体而言, 环境处理基础设施可以基于通用的环境建模机制对原始环境数据(raw environment data)进行统一地收集与处理. 环境感知应用无需知道环境信息收集与处理的细节, 它只需要根据自身对环境感知的需求依据所提供的规约机制进行环境特性描述谓词的规约. 而对于各种符合规约的谓词, 均可以统一地开发检测机制, 自动地完成环境特性谓词的检测. 同时这一分布异步环境感知框架具有较好的可扩展性(extensibility). 以时间先后关系为基础, 我们可以针对不同应用的需求, 构建不同的环境感知模型, 进而引入相应的环境特性规约及感知机制.

3.1.1 分布异步环境建模

我们将显式化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^[20]. 每个环境设施被抽象成一个进程 P_i , $1 \leq i \leq n$. 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. 进程 P_i 上的操作 a_{ij} 可分为 3 类: a) 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; b) 消息发送操作 $\text{send}(i, j, \omega)$: 进程 P_i 向进程 P_j 发送消息 ω ; c) 消息接收操作 $\text{receive}(i, j, \omega)$: 进程 P_j 从进程 P_i 接收消息 ω ^[21].

分布异步环境建模的基础是不同状态/操作间的因果联系(causality)所蕴含的时间先后关系(记为 $\rightarrow$)^[18], 其定义为: 对给定的 i 和任意 $k < l$, $s_{ik} \rightarrow s_{il}$ 如果:

图 2 分布异步环境模型

Figure 2 Distributed asynchronous environment model

消息发送操作 $\text{send}(i, j, \omega)$ 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 , 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时间先后关系 [22,23].

基于上述建模, 对于分布异步环境行为的观察可以表示为所有本地环境设施状态在时间先后关系下形成的一个偏序集. 这一偏序集可以形象地表示为环境行为时空图 (space-time diagram), 如图 2(a) 所示.

环境建模的首要核心概念是如何在相对的逻辑时钟观下定义分布异步环境的全局状态. 将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) $g = [s_1, s_2, \dots, s_n]$. 我们定义一个全局状态是一致的 (consistent global state, CGS), 当且仅当该全局状态中的任意两个本地状态之间均没有时间先后关系. 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. 例如图 2(a) 中标记的两个本地状态组成了一个一致的全局状态.

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, 相邻的 CGS 间满足偏序关系 ' $\prec$ ' 如下:

对于全局状态 $g = (s_1, s_2, \dots, s_n)$, $g' = (s'_1, s'_2, \dots, s'_n)$, 我们定义 $g \prec g'$ 当且仅当 $g \neq g'$, 且对任意 $1 \leq i \leq n$, $s_i = s'_i$ 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时间先后关系. 对于没有建立时间先后关系的两个状态, 它们的两种相对执行顺序均是可能的. 持续观察分布异步环境所得到的所有环境全局状态在全局状态间的先后关系下形成一个格. 我们用一个点表示一个全局状态, 用一条边表示全局状态间的 ' $\prec$ ' 关系, 则一个环境全局状态格可以表示为图 2(b) 所示.

上述两种建模方式体现了对分布异步环境的不同理解与环境感知的不同侧重. 当我们主要关注多个分布环境设施的本地状态的动态变化时, 可以将环境行为建模成环境设施本地状态偏序集. 类似地, 根据上面对环境全局状态的讨论, 我们以环境全局状态为核心, 则可以将环境行为建模成具有格结构的环境全局状态集合.

3.1.2 环境特性描述谓词规约

环境特性描述框架的核心是环境特性谓词的规约, 它可以从两个不同的维度来讨论. 首先, 环境特性描述谓词的规约以环境模型为基本载体. 根据前面的讨论, 如果我们侧重于描述环境全局状态的变化, 则我们可以基于环境全局状态格来规约. 类似的, 如果我们关注不同环境设施本地特性的动态

变化, 则我们可以基于环境时空图来规约.

其次, 环境特性描述谓词的规约具有从局部到全局的结构, 这与我们从局部到整体来认识分布异步环境是相吻合的. 具体而言, 环境特性描述谓词可以逐层定义如下:

本地谓词 ϕ_{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. 本地谓词是环境特性规约的原子构成.

环境全局状态谓词由多个本地谓词通过布尔运算而得到, 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^[24].

环境动态变化谓词是指定义在环境状态序列上的谓词, 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. 我们借用正则表达式、时序逻辑等各种规约机制来描述环境状态以何种模式随时间依次发生某种变化^[25,26].

由于分布异步环境所固有的不确定性, 消息传递所带来的时间先后关系仅仅是一个偏序关系. 因而在环境感知时, 一般情况下我们仅能确定系统的实际执行是一组环境状态序列中的某一条, 因而仅仅定义环境状态序列谓词是没有实际意义的. 为此我们针对所有可能的环境状态序列组成的集合引入两种模态 (modality)^[27,19]: a) $\text{Pos}(\phi)$: 在系统执行过程中存在一条环境状态序列, 使得 ϕ 成立; b) $\text{Def}(\phi)$: 对于系统执行过程中每一条环境状态序列, ϕ 均成立. 模态 $\text{Pos}()$ 和 $\text{Def}()$ 的引入解决了时间先后关系仅仅是一个偏序关系所带来的不确定性, 使得针对某条环境状态序列定义的谓词有了实际的含义.

3.1.3 环境特性描述谓词运行时检测机制的设计与实现

高效环境感知的核心内容是设计运行时、增量式的谓词检测算法, 及时准确地检测用户规约的环境特性描述谓词是否成立. 具体而言, 环境谓词检测算法的设计主要包含两部分内容: 环境全局状态格的运行时维护和环境谓词的增量式检测.

在环境感知过程中, 环境感知应用将运行时收到来自各个分布的环境设施上的环境事件. 不同环境设施上的本地状态被组合成有意义的环境全局状态. 所有全局状态被添加到既有的环境状态格中. 后续增量式的环境谓词检测算法将基于既有的环境感知结果, 仅仅处理新增的环境全局状态, 并更新环境谓词的取值.

在环境谓词检测部分, 对于每一类的规约机制我们均可以使用系统化的检测方法. 例如, 对于以正则表达式规约的环境特性谓词, 其检测方法是将环境谓词转化成等价的自动机. 首先对新增的环境全局状态, 完成全局谓词的检测, 然后讲刚刚更新的全局谓词检测的结果注入到环境特性谓词等价的自动机中, 并根据新得到的环境全局状态谓词的取值进行状态跳转. 当谓词检测算法到达自动机的接受状态即意味着应用所规约的环境谓词成立^[19]. 又例如对于以时序逻辑 CTL 规约的环境特性, 其检测主要是利用 CTL 公式的递归结构, 将其层层解析为简单的原子命题.

基于“智能办公空间”应用实例, 我们对所提出的分布异步环境感知框架进行了检验^[28,19]. 基于上述“建模—规约—检测”的感知框架, 一个面向整个智能空间的中间件平台依据先验的环境模型负责所有环境数据的收集与处理. 环境感知应用直接依据所提供的规约机制描述自身对环境行为的关注, 并向中间件进行注册 (subscribe). 对于所支持的每一类规约机制, 均实现了通用的检测机制, 中间件自动解析规约进行运行时检测, 并及时向应用返回环境感知的结果. 而插件式的 (pluggable) 环境检测机制的设计使得环境感知机制的扩充更加容易.

3.2 环境中用户行为感知技术

如前文所述, 在对研究环境中用户的行为感知展开研究时, 我们主要解决的问题包括环境状态的交错性、交互性和突发性等具体的问题, 而环境状态的可区分性则是我们进行研究的基础. 针对环境中用户行为感知的问题, 我们的工作包含以下几个主要的方面.

首先, 我们以环境状态的可区分性为基础, 提出了利用可穿戴式传感器来感知人身体各个部位的动作和物品使用情况, 以这些可感知的数据为基础, 结合人的行为模型, 来实现对人行为的感知与识别. 我们的可穿戴式传感器感知的范围包括人体各个部分 (包括, 头、躯干、四肢等) 的加速度数据、基于穿戴式 RFID 传感器的物品使用数据, 基于固定式 RFID 传感器的用户位置数据. 这些传感器构成了我们的针对用户行为识别的环境感知和处理基础设施.

在获取了基本的行为数据之后, 在针对用户行为的建模时, 我们充分考虑环境状态的可区分性, 采用了以不同行为之间的差异模式来建模人的行为的方法, 来实现准确的行为建模. 我们注意到行为之间在所涉及的基本动作和物品使用上有着明显的差异, 如“吃饭”就包括“拿起饭碗”、“提起筷子”等动作, 而“打电话”则包含“拿起/放下电话”这样的基本动作, 不同的行为之间的基本动作集合是不同的. 因此我们提出了以模式识别技术 (emerging pattern, EP) 为基础的识别框架^[29]. 通过利用 EP 强大的类别间区分能力, 我们首先实现了针对每一个基本的行为的准确识别.

更进一步地, 我们发现在识别高层的用户行为时, 行为的语义和行为被执行的方式之间有着密切的联系, 体现出较强的序列性. 而行为执行的方式, 往往就体现在执行每一个高层行为时, 其中每一个更细粒度的步骤的执行顺序. 例如:“泡茶”行为包含“放入茶叶”、“提起水壶”、“倒水”、“放下水壶”这一序列, 而且这个序列中的每一个步骤的次序不能随意更改. 因此我们在原有的 EP 的基础上, 提出了基于 (emerging sequential pattern, ESP) 的行为识别方法^[30]. 在 ESP 的方法中, 我们在对行为进行建模的基本元素就是序列而非 EP 中的集合, 由此我们能够实现对行为的更精确的建模. 除此以外, 我们针对序列性还探讨了基于扩展的 HMM (hidden Markov Models), CRF (conditional random fields) 模型的行为建模方法. 通过图模型中的状态来表示行为的发生, 通过状态的序列来建模行为的序列, 实现对真实用户行为序列性的建模.

以可区分性为出发点, 我们进一步解决了真实物理环境中, 用户行为的交错定性、交互性和突发性问题.

针对环境状态的交错性问题, 我们以单个用户行为为切入点, 首先分析了单用户在执行多个行为时可能出现的顺序、交替和并行执行等复杂的情况. 我们发现, 人的行为虽然存在个体差异, 但是在同样的行为在模式上表现出类似的特性, 并且与其他行为存在明显的差异. 同时, 在考虑用户行为的交错性时, 我们发现, 尽管单用户行为在执行时存在的交替和并发等不确定性, 但是这种交错性的行为执行能够在识别每个基本行为的基础上, 再通过分析基本行为之间的关联关系来进行识别. 因此我们就在基于 EP 的行为识别算法的基础上, 加入了对多个行为交替和并发执行的建模来实现对行为不确定性的适应能力, 对单个用户的复杂行为进行识别. 实验结果表明, 基于 EP 的算法对于单个用户的顺序、交替和并行执行的行为的识别准确率分别达到了 90.96%, 88.1% 和 82.53%.

进一步通过利用 ESP, 我们实现了针对顺序执行的行为更准确的识别. 同时, 由于 ESP 不仅包含了行为执行过程的步骤, 还能够表达每一个步骤被执行的顺序, 该方法还有潜力能够实现对于交替和并行执行等行为复杂性的更为准确和细粒度的识别. 实验结果表明, 基于 ESP 的算法对于单个用户顺序、交替和并行行为的识别准确率分别达到了 91.61%, 92.26% 和 92.32%, 该结果显示基于 ESP 的算法优于我们原先基于 EP 的识别算法.

针对多用户行为之间的交互性问题, 我们首先注意到几个事实: 第一, 在一个典型的生活场景中, 往往同时存在多个用户; 第二, 多个用户的行为之间存在复杂的相互影响, 包括协作和冲突等多种情形. 我们在研究多用户行为识别的问题时, 主要从多用户行为的互相影响出发展开研究. 我们注意到, 多个用户的行为可以分解成两个部分来考虑: 每个用户的行为, 以及这些行为之间的相互影响和关联关系. 从这个思路出发, 我们的解决方案是通过在单人的行为模型中, 加入对多个用户行为间相互影响的建模, 来实现多用户的行为识别. 具体而言, 我们采用了在单用户的 HMM 和 CRF 这两类图模型的基础上分别加入用户间相互影响的条件概率和函数, 形成了我们的 CHMM 和 FCRF (factorial conditional random field) 模型^[31,32]. 区别于图模型的方法, 我们也提出了基于 EP 的模式识别方法^[33,34]. 在传统的基于 EP 的单人行为模型中, 加入了对多个用户行为间影响的因子, 形成了我们的基于 EP 的多用户行为模型和识别算法^[33,34]. 通过分析实验结果, 我们所提出的基于图模型和基于模式识别的行为识别算法, 能够准确的捕捉多个用户行为间的相互影响, 实现了对多用户行为的准确识别. 在经过了简单的优化之后, 我们的算法都达到了 90% 的识别准确率.

针对用户行为的突发性问题, 我们的解决思路是实现实时的行为识别. 我们首先分析实现实时识别的需要解决的问题, 包含如下几个方面: 第一, 要实现对用户行为的在线的识别, 就要求我们对输入的传感器数据流只进行单遍扫描就识别出其中所包含的用户行为; 第二, 我们要尽量加快算法的运行速度, 降低时间开销. 我们注意到, 用户的行为包含明显的层次性. 高层的行为往往是有许多更细粒度的低层次行为所组成的. 为此我们提出了一种片上底层姿势识别结合服务器端高层行为识别的层次化方法^[35,36]. 我们的方法分两层执行, 首先, 在底层, 传感器获得的加速度数据, 通过一个轻量级 DTW 算法, 在片上直接识别出用户身体各个部分的姿势数据; 随后, 这些姿势数据被传输到服务器端, 通过我们提出的基于 EP 的轻量级行为识别算法, 识别出高层的用户行为. 我们的方法的优势体现在: 第一, 我们的轻量级行为识别算法, 仅对输入传感器数据流进行单次扫描, 就能够准确识别出当前的用户行为. 采用 EP 这样一种描述类别间差异的有力工具, 我们达到了 80% 以上的识别精度. 第二, 通过采用分层的识别方案, 我们所需要传输和处理的数据量减少了 60%, 大大加快了处理速度, 使得处理的平均延迟低至 5.7 个感知周期 (每个感知周期是传感器获取单位数量个读数所需要的时间, 由于其受具体的传感器设置影响, 对于识别算法而言, 采用感知周期来计算其延迟是相比物理时间更为科学的手段).

3.3 环境感知的一致性保障技术

如前所述, 环境感知需要保障环境信息的一致性. 复杂的网构软件处于开放、动态、难控的环境中, 各类分布异构的传感器以一种局部、非精确, 甚至冲突的方式来理解环境, 这很容易造成应用软件的决策困惑和表现异常. 为保障环境信息的一致性, 需要及时检测并消除环境信息中的一致性错误. 虽然一致性错误的概念及一致性保障技术在传统软件工程领域已有了一些研究, 但网构软件所面向的开放环境提出了新的挑战和需求. 我们下面列举一些挑战和需求的实例以及讨论我们的应对措施和技术.

首先, 传统一致性错误的检测技术假设了被检测实体是静态或缓慢变化的, 其检测效率不高, 并且即便是延迟检测也不会影响其检测效果. 然而网构软件所处的环境时刻变化, 这就要求一致性错误需被及时检测出来, 以避免应用的响应落后于环境的变化. 否则一致性错误会因为没被及时检测出而错过, 造成漏报, 并导致应用因使用错误的环境信息而表现出行为异常. 因此, 网构软件环境下的一致性错误检测需要高效性, 并且其与检测的效果密切相关. 为了应对这样的需求, 我们提出部分约束检

测^[37,38] (partial constraint checking) 技术. 该技术将环境的变化建模为上下文增加和上下文删除两种基本类型, 并把每种这样的变化对一致性约束的真值可能产生的影响进行预先的静态分析, 其分析结果可被分类为支持一致性约束真值评估的 4 级计算模式. 这 4 级计算模式分别为: 本公式无影响、本公式有上下文增加的影响、本公式有上下文删除的影响, 以及子公式有影响, 而其中最后一级又将在本公式的下一级递归套用这 4 级计算模式. 如此我们可以把一致性约束的检测问题进行最小粒度的分解, 并最大可能地复用先前计算并仍然有效的结果, 然后增量式地调整和添加新的计算信息以得到最新的检测结果. 该技术能大幅提高一致性错误的检测效率, 特别适用于像网构软件所处的动态环境的信息处理. 此外我们还从理论上证明了该技术的正确性, 这包括环境变化的顺序无关性 (soundness theorem), 以及增量/非增量计算的等价性 (equivalence theorem). 之后我们在一次城市交通信息处理的实际案例中, 通过实验观察到该技术比传统方法提高了近 6 倍的检测速度, 并大幅降低了一致性错误的漏检率 (从 52.2% 降至 0.1%)^[38].

其次, 被检测出的一致性错误需要被及时处理, 否则该错误的环境信息将仍旧危害应用. 然而网构软件所处的开放环境保持持续的动态变化, 这造成一致性错误的处理不能使用传统软件工程领域中的交互式修复方式, 比如处理 UML 模型中的一致性错误通常是产生一系列可能的修复方案并要求用户来选择最适合的一种. 处理环境信息中的一致性错误需要自动的修复方案, 在这方面已有的各种启发式策略可以派上用场. 它们基于各种领域知识和专家经验, 有选择地丢弃可疑的环境信息 (如 Drop-latest, Drop-random^[39], Drop-all^[40] 等启发式策略), 以达到消除一致性错误的效果. 虽然这样的策略效率比较高, 因为它们属于预先总结好的执行规则, 只关注局部数据和要求少量计算, 但它们也有两方面的局限性: 一是这样的启发式策略通常是静态的, 只关注单个一致性错误, 无法形成全局性的视角, 从而难以找到最佳的修复方案; 二是丢弃环境信息 (即便是可疑的环境信息) 仍会造成环境信息的损失, 而对应用产生未知的影响. 针对这两方面的局限性, 我们分别提出 Drop-bad 策略^[41] 和 Effect 技术^[42,43]. 前者 Drop-bad 策略着眼于多个一致性错误, 跟踪它们的生命周期并收集与之相关的所有可疑的环境信息. Drop-bad 策略使用了更具普遍意义的启发式规则, 即最近最频繁造成一致性错误的环境信息是最可疑的, 这样的环境信息被归为 Bad 类型. 该策略从全局而非单个一致性错误的角度来分析最可疑的环境信息, 并只选择这样 Bad 的环境信息进行丢弃, 从而提高了识别的准确率并降低了有用环境信息的损失. 而后者 Effect 技术, 则使用增量式计算的方式来在线预测每种一致性错误消除方案可能带来的实际影响, 然后根据应用自身的需求来选择最适合的方案, 从而更准确地减少有用环境信息的损失并保护应用原本的行为. 我们的 Drop-bad 策略和 Effect 技术分别从数据层面和应用层面两方面来保护有用的环境信息, 在处理环境信息的一致性错误方面真正考虑了应用的需求. 之后我们在基于实际场景德国 Leimen 小镇的模拟实验中, 发现使用这两项一致性错误处理策略和技术能把对应用原本行为的保护力度从 48.5%~56.0% 分别提高到 67.5% 和 81.4% (与传统方法相比)^[43].

再次, 除了前述的支持单应用的一致性错误检测与处理, 环境信息的一致性保障还需考虑多应用的需求. 当多个应用运行于同一环境处理基础设施^[44,45] 上时, 它们对一致性错误的定义、理解和处理要求可能不同甚至会相互冲突. 比如为某一应用消除一致性错误时, 修改了某条环境信息却反而对另一应用带来本不该有的新的一致性错误. 传统的做法是要么依赖于用户来制定规则并强制执行某项预定的方案^[46], 要么使用一个预定的效用函数 (utility function) 来计算出一种全局最优方案并执行之^[47]. 这些做法都有一定的局限性, 比如预定的规则或效用函数难以确保对所有应用适用, 可能需要反复调整, 容易造成无法遵循某项应用的一致性错误处理要求, 另外有些情况下可能根本无法同时满足所有应用的需求. 对此我们考虑一种新的视角, 为每个应用透明地维护一个单独的环境视图 ConsView.

图 3 环境感知的一致性保障技术

Figure 3 Consistency assurance of environmental contexts

该视图使该应用感觉自己独占此环境, 与其他应用都隔离. 如此每个环境视图可仅为自己负责的应用进行一致性保障的工作, 从而隔离多应用之间的冲突^[48]. 我们证明了该项技术的可靠性, 能保证每个应用原本的环境信息使用完全不受影响.

最后, 我们用图 3 来总结我们对环境感知的一致性保障技术的理解和处理方式, 我们展现环境信息一致性处理是如何与应用相结合, 其如何影响应用的行为和我们如何动态地约束这样的影响, 以达到既保障应用的环境感知一致性又保护应用自身行为的目标.

4 显式化环境开发与运行驱动机制

上一节我们在先验环境认知框架概念下, 讨论了环境感知中的 3 个较为重要的技术问题. 而在实际应用开发中, 仍会遇到应用特定的环境感知需求. 为此, 我们需要有一套方法来指导我们针对应用特定需求开发相应的先验认知框架 (包括环境模型和环境规约), 并配备以适当的环境设施. 具体而言, 本文提出了一种面向计算活动的显式化环境迭代式建模方法及主动式计算驱动机制. 在开发层面, 开发人员首先将分析网构软件中计算实体的活动对环境的需求, 然后在环境经验的基础上针对需求开展环境上下文建模、环境规约描述, 并得到相应的环境模型; 然而, 由于难以一步到位地进行需求获得并得到最终的环境模型, 开发人员可以采用迭代式不断进行增量式环境开发. 在环境驱动层面, 显式化环境将围绕环境模型动态组织各个活动的上下文, 主动式地进行环境事件的检测以及环境规约的检查并最终驱动计算的演化.

本节将顺序介绍上述显式化环境的开发方法、主动式计算驱动机制, 及相应的实例分析.

4.1 面向计算活动的显式化环境迭代式开发

4.1.1 面向计算活动的上下文建模

通常情况下, 环境开发人员会根据自身对环境的观察能力、抽象能力和理解环境的先验知识和经验, 采用形式化方法对网构软件环境中的所有要素进行定义, 进而在这些要素的基础上进行不同语义层面上的抽象, 建立环境模型, 从而最终为业务逻辑的执行提供服务. 这种面向整个环境的开发方法却常常难以实施或者效果欠佳: 1) 客观环境是开放、不确定和动态的, 我们只能获得不完整的、难以掌

图 4 面向计算活动的上下文概念模型

Figure 4 Computation activity-oriented context concept model

控、难以一致的环境视图, 在环境规模和复杂性上升到一定程度后, 环境视图的科学性和可用性得不到保证; 2) 上述视图极大地依赖于开发人员对于这个世界的经验知识积累和抽象能力, 具有较强的武断性和个人因素, 缺乏工程方法的指导, 可理解、可复用性不足; 3) 直接开展整个环境的要素分析和语义抽象, 往往和应用逻辑的环境需求不能很好地对应, 在效率和适应性方面均存在不足。

关注分离是软件工程中进行规模分解、复杂性控制和适应性增强的重要方法。从根本上看, 上述传统方法的不足来自于环境开发过程的抽象环节中没有寻找到一个合适的关注分离应用点, 从而导致了环境视图过于庞大、难以高效应用、难以适应性维护。

为网构软件提供高质量的环境保障并在环境变化时合理地驱动计算的演化是显式化环境本身的存在价值, 网构软件计算实体在运行时刻对环境的需求是相对独立和稳定的。以各个计算实体的活动对环境的要求为关注点, 将环境的建模分解为各个关注点的上下文建模并以此为核心开展环境的维护和计算演化的驱动, 是一种合理的关注分离思想的应用, 具有以下几个优点: 1) 从网构软件计算实体的活动出发, 便于开展针对性的环境需求分析和需求获取; 2) 从环境需求入手进行针对性的环境上下文模型的建立, 便于在经验知识基础上进行工程应用, 上下文模型及相应规约的开发复杂度得到有效控制; 3) 由针对不同需求的多个环境上下文模型组成环境模型, 环境模型的开放性、可复用性和可适应性更好。

从网构软件计算实体的动作出发, 本文提出了一个面向计算活动的上下文模型, 而一个网构软件的显式化环境即是所有计算实体对应的不同上下文模型的统称 (或者是集合)。图 4 是该模型的一个本体描述。

4.1.2 迭代式环境开发

如前所述, 潜在的动态扩展是开放环境下网构软件系统开发的必要因素。具体而言, 开放环境下对环境认识常常不能一次到位, 无论面向应用的环境规约还是相对通用的环境模型均在持续性地满足业务需求方面, 需要得到开发层面的独立表达以及运行时的在线动态扩展支持。结合前述上下文模型, 可使用一种支持迭代式的网构软件环境上下文模型和规约的开发方法, 在该方法下开发得到的若干上下文模型及相应的上下文规约共同构成显式化环境。

图 5 迭代式环境开发方法

Figure 5 Iterative development of environment framework

该方法将环境上下文模型和上下文规约的开发归结为两个基本开发动作和两个可迭代周期如图 5 所示:

面向计算活动的环境需求分析及其迭代: 网构软件的正确执行需要环境显式地提供保障, 另一方面环境的变化也需要驱动计算进行适应性变化. 因此, 开发人员应该在高层的业务逻辑中, 基于关注分离原则, 以计算活动为基础, 针对每个活动, 分析该计算动作的环境依赖性, 显式地表达环境保障需求及计算演化驱动需求 (这种表达可以在不同的语义层面上给出, 可以不直接观察和操纵客观环境). 该阶段将产生一个环境需求规格说明. 在网构软件运行过程中, 如果有新的计算实体加入或者新的计算活动产生或者发现原有需求有偏差, 开发人员可以加入新的环境需求或者修改原需求.

面向需求的上下文建模及其迭代: 环境需求分析得到的每一个环境需求规格说明都需要具体落实到一个可实现的上下文模型中去 (“可实现” 特指可以高质量地访问到该模型中的所有客观环境要素数据). 开发人员根据自身的环境经验知识以及已有的上下文模型, 分析每个上述的环境需求, 解剖其涉及到的客观环境要素, 分析这些要素之间的语义关联, 开发不同层次上的数据融合规则, 建立起对应的可实现上下文模型及上下文规约. 同样如此, 在新环境要素、新环境知识或者新环境需求加入的情况下, 开发人员可以迭代式开发新的上下文模型和规约.

4.1.3 实例分析

为展示上述迭代式环境开发方法的可用性, 本文设计了以下案例.

考虑一个简化的商务场景: 旅行商务系统集成了自主运行的、分布在 Internet 上的多个节点的往返机票预定、酒店/餐馆预定及在线支付等服务. 然而在运行时刻, 分布、自治、难控的运行环境将对该系统的正常运行产生影响: 比如某些原因导致服务的不可用、服务所处节点崩溃或网络连接质量下降、服务所处节点负载过大、服务的信誉度发生变化等.

在前述网构软件环境开发方法指导下, 环境开发人员首先围绕各个服务的运行环境保障需求, 提出了两个环境需求: 1) 所有服务都是可用的, 2) 在线支付服务是高可信的. 针对可用性 (availability) 的环境需求, 开发人员在对环境认识理解的基础上, 分析可获取环境因素, 建立以下可实现的上下文模型:

环境要素: 网络可达性、服务有效性、服务器系统负载和网络可用带宽. 其中, 网络可达性可实现为服务所在节点之间的网络可达率 r_{NA} (r_{NA} 表示最近 n 次 ping 中成功的比率); 服务有效性可以利用服务有效率 $r_{SV} = t_{run}/t_{server}$ 来实现 (t_{run} 表示服务正常部署运行的时间, t_{server} 则表示从该服务首次部署到调用时刻这段期间内服务器运转时间总和³⁾); 而服务器系统负载则可用 CPU 的平均空闲率 R_{CPU} 来实现.

针对一个具体的服务构件 SC_1 , 我们可以依据上述环境因素进一步对 SC_1 的可用性定义一系列效用函数, 例如:

$$\text{Overload}(SC_1) = GT(R_{CPU}, T_{CPU}), \quad (1)$$

$$\text{Availability}(SC_1) = (r_{NA} * (r_{NA}, T_{NA})) * (r_{SV} * GT(r_{SV}, T_{SV})) * \text{Overload}(SC_1) * GT(BW, T_{BW}), \quad (2)$$

其中 $T_{NA}, T_{SV}, T_{CPU}, T_{BW}$ 为事先设定的几个阈值, 分别表示应用对当前的网络可达率、服务有效率、可用 CPU 以及可用带宽的最低要求 ($GT(x, y) = 1$, 当且仅当 $x > y$, 否则 $GT(x, y) = 0$).

上面针对服务可用性的各个相关的环境因素以及我们针对服务可用性的效用函数共同构成了我们针对有效性的上下文模型. 在此模型之上, 环境开发人员可以进一步通过环境规约来定义所关注的、导致系统进行演化或调节的环境事件 (如 $\text{Availability}(SC_1) \geq 0.8$). 类似的, 针对另一条可信性的需求, 我们也可以找到与之相关的各个环境因素 (例如服务提供商的声誉、网络传输是否安全等) 以及相应的效用函数, 从而建立一个与可信性对应的上下文模型以及在此基础上的一系列的环境规约.

然而, 随着系统的运行, 用户对整个系统 (及其环境) 的认识逐步深入, 可能会发现现有的环境模型的缺陷或不足, 而需要对现有环境模型进行重定义或修改. 例如, 仅仅依据 CPU 空闲率来衡量服务器负载并不充分, 而要求将服务器的可用内存 R_{Mem} 纳入可用性的考虑因素. 为满足这一新需求, 我们需要重新开发针对可用性的上下文模型. 而这一模型的开发可以在原有的上下文模型基础上简单的添加 (或删除环境因素以及相应的效用函数) 进行迭代式增量的实现: 将可用内存这一环境因素加入原有的环境模型之中, 并在此基础上重新定义服务器系统负载效用函数并替换原有效用函数 (见公式 (1)):

$$\text{Overload}(SC_1) = GT(R_{CPU}, T_{CPU}) * (R_{Mem}, T_{Mem}), \quad (3)$$

其中 T_{Mem} 表示应用对当前的服务有效率的最低要求.

4.2 环境驱动的软件运行机制

环境需求反映了软件对环境的需求, 比如上述案例中的服务 SC_1 可用性. 环境需求按照前文所述迭代式环境开发方法转换为环境模型和环境规约, 前者反应了环境中各个环境要素之间的关系和值的计算方法, 比如 $\text{Availability}(SC_1)$ 的效用函数, 后者给出了环境中各个要素需要满足的性质, 比如 $\text{Availability}(SC_1) \geq 0.8$. 结合应用需求规约和环境规约, 给出应用自适应的规则, 从而在运行时刻建立起环境和应用系统之间交互的 ECA (event-condition-action) 模式. 其中 Event 包括: ContextChangedEvent, ContextAddedEvent 和 ContextDeletedEvent, 分别表示上下文值变化、新增上下

3) 此处要求服务所部属的节点提供相应统计数据

图 6 环境驱动的软件运行机制

Figure 6 Environment-driven mechanism

文和删除上下文这 3 类事件, 开发者可在这 3 类事件的基础上扩展出更具体的上下文事件; Condition 定义了运行时刻可能出现的一些上下文因素变化条件; Action 则对应于为了保障应用系统满足环境规约而需要采取的一些自适应动作. 根据粒度不同可以分为两类自适应行为: 一类不需要修改已有应用系统的行为, 仅需要根据环境规约对应用系统的行为进行组合/重组, 形成环境驱动的动态协同. 另一类则需要修改应用系统的行为, 主要体现为由于需求变化或非预期环境因素的出现而导致环境规约发生变化, 系统既有行为已经不能满足规约中的约束, 必须通过修改行为而达到, 我们称之为操作变体 (operation variants). 值得一提的是, 我们假定应用系统功能实现采用松散耦合的体系结构, 比如支持动态协同的 Web 服务或基于构件的架构, 各个服务或构件相互间通过协同机制进行跨网络的互连、互通、协作和联盟来实现目标系统的业务功能, 并允许在运行时刻改变协同的方式以便适应环境的变化, 具体可参考文献 [49 ~ 52]. 如果既有软件系统并非采用松耦合结构, 或者耦合度不够松散, 需要采用更为细粒度的程序级自适应机制加以应对, 相对地操作变体也应有所调整, 但本文提出的驱动机制也具有一定的参考价值.

如图 6 所示, 在运行时刻, 环境模型经过解析生成环境上下文模型、给出环境上下文事件的定义, 通过环境基础设施监控环境中的上下文, 当发生所定义的上下文事件时通知上下文模型, 模型按照效用函数计算上下文的值, 并将计算结果发送给规则引擎. 规则引擎中加载了一组适应性规则和操作变体. 由于应用系统在实现业务功能时与环境分离, 为了使得应用系统能够在运行时刻具有环境感知和自适应能力, 需要通过编织器将环境与应用的行为进行编织. 编织器根据适应性规则编制出应用行为协同的流程, 驱动应用系统中各个功能部件 (web 服务或构件) 所提供的操作, 流程执行的模式包括顺序执行 (SEQ)、并行执行 (PAL) 和选择执行 (SEL), 该流程采用解释执行的方式, 可以根据环境的变化而做适应性调整. 流程中的各个活动可以来自于应用系统的既有功能或服务, 亦可以是新引入的操作变体. 操作变体与 COP^[6,7] 中的 partial method 类似, 代表了在特殊上下文中可能被激活的操作. 操作变体可以根据其被激活的位置组织成为 3 种类型的子操作链, 即 BeforeChain, AroundChain 和 AfterChain. 从每个子操作链的名字不难看出, 各操作变体分别对应于编织类型为 before, around 和

after 的适应行为, 依次表示先于指定方法、替代指定方法或晚于指定方法执行.

适应性规则的示例如下:

```
<rule name="rule1">
  <event>
    <type>ContextChangedEvent</type>
    <context>sc1.availability</context>
  </event>
  <condition>
    less sc1.availability 0.8
  </condition>
  <action-set type="SEQ" >
    <action name="ChangeServiceAction"
      endpoint="http://moon.nju.edu.cn/services/archserv/"
      method="changeService">
      <argument name="service" type="string" value="sc2" />
    </action>
    <action name="AddFeedbackAction"
      endpoint="http://moon.nju.edu.cn/services/feedback/"
      method="feedback" type="after"
      interceptService= "http://moon.nju.edu.cn/services/calculate/"
      interceptMethod= "calc" >
    </action>
  </action-set>
</rule>
```

(1) 每个 rule 都含有一个 name 作为 ID, 是唯一的. Rule 中包含 3 个部分, 一部分为 event, 一部分为 condition, 另一部分是 action set. 只有当指定 event 发生时检测 condition 是否满足, 若 condition 满足则执行 action set 中的 action.

(2) Action set 中包含了一系列的 action, 它还有个 type 属性, 用来标识 action 的执行方式, 即 SEQ, PAL 和 SEL, 默认为 SEQ.

(3) 在 action 节点中, endpoint 是将要执行的网构应用既有功能实现中的服务访问入口或新增操作变体的访问入口, method 是指要调用的方法, 如果这个方法含有参数的话, 可以使用 argument 子节点指出参数的名称, 参数的类型, 比如 int, float 或者 string, 以及参数的值, interceptService 指出操作变体需要截获的服务名称, interceptMethod 则给出要拦截的服务具体方法, type 则是其拦截的方式, 如 before, around 或者 after.

上述示例表示, 当服务的可用性低于阈值时, 触发自适应行为. 首先协同既有系统所提供的服务, 比如更换一个备用服务, 然后执行操作变体提供的服务, 比如增加环境反馈, 由于其类型为 "after", 因此该操作将在指定服务的方法执行结束后进行.

本节对显式化环境的迭代式开发和相应的自适应行为环境驱动机制进行了阐述. 尽管这方面的工作尚处于探索阶段, 但根据以上分析, 我们认为较之传统方法, 该方法中上下文模型的建立因关注分离、粒度适宜而可有较好的实用性, 上下文实例的组织管理可动态进行, 可有较好的适应性, 自适应行为驱动也相应地可有更好的针对性和适应性. 我们将在未来工作中通过详细的案例研究来评估和验证其实际使用效果.

5 相关工作

网构软件在“环境显式化”和“软件系统动态应对环境变化”这两个方面, 缺乏适合直接刻画环境驱动的网络软件系统特点的开发模型和支撑技术成为一个日趋突出的问题. 在这两方面可以借鉴上下文感知计算领域的相关工作. 一类是提供编程工具包, 代表项目有 Context-Toolkit^[53], 它使用基于构件的软件开发方法, 提供了一个编程框架结构和一组可复用的组件用于支持上下文感知应用的快速原型开发. 另外一类工作主要关注构建用于上下文感知应用的基础结构或中间件, 比如 Solar 系统^[54]和 SOCAM 系统^[55]. 虽然在中间件的层次上处理上下文使得上层应用可以不用关心环境信息的收集、管理等工作, 但是已有的大多数工作中, 难以对上下文感知相关的行为和决策等进行显式的定义和开发, 即如何利用上下文自适应地调整自身的行为仍然包含于应用业务逻辑中. 这样便导致了如果出现了新的上下文, 或者如果软件的上下文适应策略发生了变化, 软件就需要停止运行并重新编程. 本文所提出的面向上下文的软件构造方法的优点体现为两个方面: 一方面, 环境可以影响应用系统的行为, 通过环境的变化在运行时刻驱动应用系统业务逻辑的执行或改变协同逻辑, 环境与应用系统的隔离可以使得应用系统相对稳定, 不会因为环境的变化而导致需要重新编程, 提高了各自的复用性和可维护性; 另一方面, 软件在环境中运行过程亦会影响环境, 比如改变环境上下文的值. 由于在同一个环境中可以运行多个软件, 通过这种动态“互动”的方式亦可以实现环境信息及环境基础设施的复用.

从环境模型的建立和具体上下文建模和处理技术上看, 传统的工作可以理解为“属性描述上下文”的建立和处理^[53,56,57]. 通常情况下, 开发人员根据自己的知识积累、观察能力和抽象能力, 尝试去定义应用所处环境中的所有要素及其关系, 建立一个统一的上下文应用模型, 并在这个模型上开展各类应用服务^[55,58]. 上下文成为了一种预定义范围的计算成分, 尽管对于上下文建模和使用的方法各不相同, 但在这些工具中, 上下文的范围是静态的, 不随计算系统的动作的改变而改变, 显得臃肿和缺乏适应性. 随着研究者对上下文理解的不断深入, 普适计算领域中的部分研究者开始关注于基于用户的动作的上下文感知系统设计, 并取得了一些进展. Dourish^[59]给出了对于上下文涌现特性的一些理解. Chalmers^[60]提出上下文的范围不能被预定义, 不应当由系统设计者决定上下文, 而是用户在设计者的影响下决定. Greenberg^[61]把上下文看作一个动态的结构体, 认为设计者如果轻易地确定上下文, 容易导致建立不合适的上下文感知应用. Rehman^[62]提出了一个给用户控制权和反馈的交互模型. 另有一些研究者给出了动作驱动模型^[63]以及从用户角度设计上下文感知计算^[64]的计划. 然而这些工作要么仅仅给出了理念级上的新理解, 要么就是在某些设计技术上有所尝试, 并没有将“以计算实体的动作为核心开展环境分析、建模和维护”作为一般意义上的网络化应用软件开发方法加以深入和系统的研究, 更没有将环境或者上下文作为显式的主动式计算成份加以凸显和独立开发.

在针对特定开放环境特性的感知技术研究方面, 如何应对开放环境所固有的分布异步性这一问题已受到较多关注^[65]. 但是在具体应对分布异步性方面, 既有工作主要是基于绝对的物理时钟观, 采用首先在分布式计算实体间进行时钟同步然后再进行同步协同的做法. 例如 Anid Dey 等指出, 不同环境设施之间需要有相同的时间 (the same notion of time), 因而需要预先 (尽量) 在环境设施间做同步, 以此为基础再进行后续环境感知的工作^[66]. 与既有工作所不同的是我们的工作直接应对开放环境的分布异步性, 基于相对的逻辑时钟进行环境感知的工作. 这主要由于随着新型计算通信技术以及应用模式的出现, 环境的分布异步性在较为显著地增加, 而与之相对的是用户对软件应用容错性、可靠性等非功能性需求同样在提高. 在这一背景下, 同步时钟假设成为了系统容错性可靠性保障的瓶颈, 所需开销在许多场合下显著增加. 与之相对的是, 我们所采用的逻辑时钟观与异步计算模型更贴近真实物理环境, 对计算环境的前提假设最少, 因而具有相对更高的通用性, 是同步环境感知机制的有效

补充.

在环境感知的一致性保障技术方面, 有许多在传统软件工程领域做出的相关工作, 比如为 UML 模型 [67~69]、XML 文档 [70,71] 和数据结构 [72] 等软件实体的一致性检测, 这些软件实体通常为静态或者变化缓慢. 而我们网构软件的开放环境动态多变, 其中的环境信息多而杂, 因而需要高效的一致性检测算法. 此外, 检测到的一致性错误, 传统的做法是使用交互式的修复 [73,74], 由用户来选择最合适的方案, 这带来效率上的问题, 也不能解决动态网构软件环境的需求. 有些工作使用启发式策略 [39,40] 等, 通过快速执行预先总结的规则来进行一致性错误的消除, 但这些工作多是基于数据处理的视角, 没有考虑环境信息的改变也会带来应用软件行为的变化, 而是否这样的变化是所需的或者可接受的, 仍需要进一步考虑. 这些已有工作不足促成了我们网构软件环境感知新技术的发展, 这包括了前述的部分约束检测、Drop-bad 和 Effect 技术, 以达到“又快又好”感知环境的目标, 为应用软件正确理解环境奠定基础.

6 小结

环境驱动的网络化软件系统的构造是一个重要而复杂的问题. 究其原因, 除了开放环境所固有的复杂性外, 还因为当前实践中缺乏合理的软件方法与技术来系统地处理软件所面对的环境感知和使用需求. 本文提出为系统配备先验的环境模型和环境规约来显式化环境感知逻辑, 以实现环境处理与业务逻辑的关注分离、通用环境模型与应用环境规约的关注分离, 促进环境感知基础设施和环境检测机制等的软件复用. 我们讨论了如何用这个框架来组织分布异步环境感知、用户行为感知、环境一致性保障等具体开放环境感知技术.

在本文中我们对环境的感知更多地采用所谓的演绎 (deduction) 的方式, 但我们相信, 显式化表述并在运行时刻维护环境模型和环境规约也为我们采用归纳 (induction) 的途径 (如数据挖掘) 来认识环境提供了依托. 这也是我们下一步要研究的方向. 另一方面, 本文主要考虑环境的感知, 关于对环境的控制部分, 实际上也应通过环境模型和环境规约来实施. 此时模型和规约将需要有操作性 (operational) 的解释, 而这也我们未来工作需要补充的部分. 同时, 本文中讨论的环境模型和环境规约的开发思路和环境驱动机制仍然较为初步, 在未来工作中我们将进一步具体化相关的开发方法, 设计实现相应的语言工具和支撑平台.

致谢 感谢汪亮、马骏等对本文所作的贡献.

参考文献

- 1 Lv J, Ma X X, Tao X P, et al. On environment-driven software model for Internetwork. *Sci China Ser F-Inf Sci*, 2008, 51: 683-721
- 2 Yang F Q, Mei H, Lv J, et al. Some discussion on the development of software technology. *Chin J Electron*, 2002, 30: 1901-1906 [杨芙清, 梅宏, 吕建, 等. 浅论软件技术发展. *电子学报*, 2002, 30: 1901-1906]
- 3 Lv J, Ma X, Huang Y, et al. Internetwork: a shift of software paradigm. In: *Proceedings of the First Asia-Pacific Symposium on Internetwork*. Beijing: ACM, 2009
- 4 Mei H, Huang G, Xie T. Internetwork: a software paradigm for Internet computing. *Computer*, 2012, 45: 26-31
- 5 Costanza P, Hirschfeld R. Language constructs for context-oriented programming: an overview of ContextL. In: Wuyts R, ed. *Proceedings of the 2005 Dynamic Languages Symposium*. San Diego: ACM, 2005

- 6 Keays R, Rakotonirainy A. Context-oriented programming. In: Proceedings of the Third ACM International Workshop on Data Engineering for Wireless and Mobile Access. San Diego: ACM, 2003. 9–16
- 7 Hirschfeld R, Costanza P, Nierstrasz O. Context-oriented programming. *J Object Technol*, 2008, 7: 125–151
- 8 Polastre J, Szewczyk R, Culler D E. Telos: enabling ultra-low power wireless research. In: Proceedings of the 4th International Symposium on Information Processing in Sensor Networks (IPSN). Los Angeles: IEEE Press, 2005. 364–369
- 9 Lundquist J D, Cayan D R, Dettner M D. Meteorology and hydrology in Yosemite National Park: A sensor network application. In: Proceedings of the 2nd International Symposium on Information Processing in Sensor Networks. Palo Alto: Springer-Verlag, 2003. 518–528
- 10 Ahonen T, Virrankoski R, Elmusrati M. Greenhouse monitoring with wireless sensor network. In: *Mechronic and Embedded Systems and Applications*, Beijing, 2008. 403–408
- 11 Costa P, Mottola L, Murphy A L, et al. Programming wireless sensor networks with the TeenyLime middleware. In: *Middleware 2007*. Newport Beach: Springer, 2007. 429–449
- 12 Heinzlman W, Murphy A, Carvalho H, et al. Middleware to support sensor network applications. *IEEE Network*, 2004, 18: 6–14
- 13 Liu T, Martonosi M. Impala: a middleware system for managing autonomic, parallel sensor systems. *ACM SIGPLAN Notices*, 2003, 38: 107–118
- 14 Frederick P B Jr. *The Mythical Man-Month*. Boston: Addison-Wesley Longman Publishing Co., Inc., 1995. 57–60
- 15 Giguette R. Building objects out of Plato: applying philosophy, symbolism, and analogy to software design. *Commun ACM*, 2006, 49: 66–71
- 16 Rayside D, Campbell G T. An Aristotelian understanding of object-oriented programming. In: *Proceedings of the 15th ACM SIGPLAN Conference on Object-oriented Programming, Systems, Languages, and Applications*. Minneapolis: ACM, 2000. 337–353
- 17 Chandy K M, Lamport L. Distributed snapshots: determining global states of distributed systems. *ACM Trans Comput Syst*, 1985, 3: 63–75
- 18 Lamport L. Time, clocks, and the ordering of events in a distributed system. *Commun ACM*, 1978, 21: 558–565
- 19 Yang Y L, Huang Y, Cao J, et al. Formal specification and runtime detection of dynamic properties in asynchronous pervasive computing environments. *IEEE Trans Parallel Distrib Syst*, 2012, in press
- 20 Huang Y, Yu J P, Ma X X, et al. Monitoring properties of open environments. *J Softw*, 2011, 22: 865–876 [黄宇, 余建平, 马晓星, 等. 开放环境特性感知技术. *软件学报*, 2011, 22: 865–876]
- 21 Schwarz R, Mattern F. Detecting causal relationships in distributed computations: in search of the holy grail. *Distrib Comput*, 1994, 7: 149–174
- 22 Fidge C J. Partial orders for parallel debugging. In: *Proceedings of ACM SIGPLAN and SIGOPS Workshop on Parallel and Distributed Debugging*. Madison: ACM, 1988. 183–194
- 23 Mattern F. Virtual time and global states of distributed systems. In: *Proceedings of International Workshop on Parallel and Distributed Algorithms*. Holland: Elsevier Science Publishers B.V., 1989. 215–226
- 24 Garg V K, Waldecker B. Detection of strong unstable predicates, in distributed programs. *IEEE Trans Parallel Distrib Syst*, 1996, 7: 1323–1333
- 25 Yang Y L, Huang Y. Formal specification and runtime detection of dynamic properties in asynchronous pervasive computing environments. Technical Report ICS-NJU-121311. Institute of Computer Software, Nanjing University, Dec. 2011
- 26 Wei H F, Huang Y, Cao J N, et al. Formal specification and runtime detection of temporal properties for asynchronous context. In: *Proceedings of the International Conference on Pervasive Computing and Communications (PerCom)*. Lugano, 2012. 30–38
- 27 Huang Y, Yang Y L, Cao J N, et al. Runtime detection of the concurrency property in asynchronous pervasive computing environments. *IEEE Trans Parallel Distrib Syst*, 2012, 23: 744–750
- 28 Middleware Infrastructure for Predicate detection in Asynchronous environments. <http://mipa.googlecode.com>
- 29 Gu T, Wang L, Wu Z, et al. A pattern mining approach to sensor-based human activity recognition. *IEEE Trans Knowl Data Engineer (TKDE)*, 2011, 23: 1359–1372

- 30 Gu T, Wang L, Chen H, et al. Mining emerging sequential patterns for activity recognition in body sensor networks. In: *Proceedings of the 7th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services (MobiQuitous 2010)*. Sydney: Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, 2012. 102–113
- 31 Wang L, Gu T, Tao X P, et al. Sensor-based human activity recognition in a multi-user scenario. In: *Proceedings of the European Conference on Ambient Intelligence (AmI'09)*. Salzburg: Springer-Verlag Berlin Heidelberg, 2009. 78–87
- 32 Wang L, Gu T, Tao X P, et al. Recognizing multi-user activities using wearable sensors in a smart home. *Elsev J Pervas Mobile Comput (PMC)*, 2011, 7: 287–298
- 33 Gu T, Wu Z, Wang L, et al. Mining emerging patterns for recognizing activities of multiple users in pervasive computing. In: *Proceedings of the 6th International Conference on Mobile and Ubiquitous Systems: Computing, Networking and Services*. Toronto, 2009. 1–10
- 34 Gu T, Wang L, Chen H, et al. Recognizing multi-user activities using body sensor networks. *IEEE Trans Mobile Comput (TMC)*, 2011, 7: 287–298
- 35 Wang L, Gu T, Chen H, et al. Real-time human activity recognition using a body sensor network. In: *Proceedings of the 16th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications (RTCSA 2010)*. Macau, 2010
- 36 Wang L, Gu T, Tao X P, et al. A hierarchical approach to real-time activity recognition in body sensor networks. *Elsev J Pervas Mobile Comput (PMC)*, 2012, 8: 115–130
- 37 Xu C, Cheung S C, Chan W K. Incremental consistency checking for pervasive context. In: *Proceedings of the 28th International Conference on Software Engineering*. Shanghai: ACM, 2006. 292–301
- 38 Xu C, Cheung S C, Chan W K, et al. Partial constraint checking for context consistency in pervasive computing. *ACM Trans Softw Eng Meth*, 2010, 19: 1–61
- 39 Chomicki J, Lobo J, Naqvi S. Conflict resolution using logic programming. *IEEE Trans Knowl Data Eng*, 2003, 15: 244–249
- 40 Bu Y, Gu T, Tao X, et al. Managing quality of context in pervasive computing. In: *Proceedings of the 6th International Conference on Quality Software*. Washington: IEEE Computer Society, 2006. 193–200
- 41 Xu C, Cheung S C, Chan W K, et al. Heuristics-based strategies for resolving context inconsistencies in pervasive computing applications. In: *Proceedings of the 28th International Conference on Distributed Computing Systems*. Washington: IEEE Computer Society, 2008. 713–721
- 42 Xu C, Cheung S C, Chan W K, et al. On impact-oriented automatic resolution of pervasive context inconsistency. In: *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. New York: ACM, 2007. 569–572
- 43 Xu C, Ma X, Cao C, et al. Minimizing the side effect of context inconsistency resolution for ubiquitous computing. In: *Proceedings of the 8th ICST International Conference on Mobile and Ubiquitous Systems*. Berlin: Springer, 2011. 285–297
- 44 Xu C, Cheung S C. Inconsistency detection and resolution for context-aware middleware support. In: *Proceedings of the Joint 10th European Software Engineering Conference and 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering*. ACM SIGSOFT Softw Eng Notes, 2005, 3: 336–345
- 45 Xu C, Cheung S C, Lo C, et al. Cabot: on the ontology for the middleware support of context-aware pervasive applications. In: *Proceedings of the NPC Workshop on Building Intelligent Sensor Networks*. Berlin: Springer, 2004. 568–575
- 46 Park I, Lee D, Hyun S J. A dynamic context-conflict management scheme for group-aware ubiquitous computing environments. In: *Proceedings of the 29th Annual International Computer Software and Applications Conference*. Washington: IEEE Computer Society, 2005. 359–364
- 47 Capra L, Emmerich W, Mascolo C. CARISMA: context-aware reflective middleware system for mobile applications. *IEEE Trans Softw Eng*, 2003, 29: 929–945
- 48 Yang H, Xu C, Ma X, et al. ConsView: towards application-specific consistent context views. In: *Proceedings of the IEEE International Computer Software and Applications Conference*. Washington: IEEE Computer Society, 2012. 632–637

- 49 David G, Cheng S, Huang A, et al. Rainbow: architecture-based self-adaptation with reusable infrastructure. *IEEE Comput*, 2004, 37: 46–54
- 50 Ma X X, Yu P, Tao X P, et al. A service-oriented dynamic coordination architecture and its supporting system. *Chin J Comput*, 2005, 4: 467–477 [马晓星, 余萍, 陶先平, 等. 一种面向服务的动态协同架构及其支撑平台. *计算机学报*, 2005, 4: 467–477]
- 51 Yu P, Ma X X, Lv J, et al. A dynamic software architecture oriented approach to online evolution. *J Softw*, 2006, 17: 1360–1371 [余萍, 马晓星, 吕建, 等. 一种面向动态软件体系结构的在线演化方法. *软件学报*, 2006, 17: 1360–1371]
- 52 Ma X, Zhou Y, Pan J, et al. Constructing self-adaptive systems with polymorphic software architecture. In: Chang S, ed. *Proceedings of the 19th International Conference on Software Engineering and Knowledge Engineering*. Illinois: Knowledge System Institute, 2007. 2–8
- 53 Dey A K, Salber D, Abowd G D. A conceptual framework and a toolkit for supporting the rapid prototyping of context-aware applications. *Hum-Comput Interact (HCI) J*, 2001, 16: 97–166
- 54 Chen G. Solar: building a context fusion network for pervasive computing. PhD Thesis. Dartmouth : Dartmouth University, 2004
- 55 Gu T, Pung H K, Zhang D Q. Toward an OSGi-based infrastructure for context-aware applications. *IEEE Pervas Comput*, 2004, 3: 66–74
- 56 Chen H, Finin T, and Joshi A. An ontology for context-aware pervasive computing environments. *The Knowl Eng Rev*, 2003, 18: 197–207
- 57 Dey A, Abowd G, Salber, D. A conceptual framework and a toolkit for supporting therapid prototyping of context-aware applications. *Hum-Comput Interact*, 2001, 16: 97–166
- 58 Román M, Hess C, Cerqueira R, et al. Gaia: amiddleware platform for active spaces. *ACM SIGMOBILE Mob Comput Commun Rev*, 2002, 6: 65–67
- 59 Dourish P. What we talk about when we talk about context. *Pers ubiquit comput*, 2004, 8: 19–30
- 60 Chalmers M. A historical view of context. *Comput Supp Cooper Work (CSCW)*, 2004, 13: 223–247
- 61 Greenberg S. Context as a dynamic construct. *Hum-Comput Interact*, 2001, 16: 257–268
- 62 Rehman K, Stajano F, Coulouris G. An architecture for interactive context-aware applications. *IEEE Pervas Comput*, 2007, 6: 73–80
- 63 Teo H. An activity-driven model for context-awareness in mobile computing. In: *Proceedings of the 10th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI 2008)*. New York: ACM , 2008. 545–546
- 64 Zhang X. A users perspective of design for context-awareness. In: *Proceedings of the 13th International Conference on Ubiquitous Computing (UbiComp 2011)*. New York: ACM, 2011. 531–534
- 65 Gratner F C. Fundamentals of fault-tolerant distributed computing in asynchronous environments. *ACM Comput Surv*, 1999, 31: 1–26
- 66 Dey A. Providing architectural support for building contextaware applications. PhD Thesis. Georgia: Georgia Institute of Technology, 2000
- 67 Egyed A. Instant consistency checking for the UML. In: *Proceedings of the 28th International Conference on Software Engineering*. New York: ACM, 2006. 381–390
- 68 Reiss S P. Incremental maintenance of software artifacts. *IEEE Trans Softw Eng*, 2006, 32: 682–697
- 69 Xiong Y, Hu Z, Zhao H, et al. Supporting automatic model inconsistency fixing. In: *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering*. New York: ACM, 2009. 315–324
- 70 Nentwich C, Capra L, Emmerich W, et al. Xlinkit: a consistency checking and smart link generation service. *ACM Trans Internet Technol*, 2002, 2: 151–185
- 71 Nentwich C, Emmerich W, Finkelstein A, et al. Flexible consistency checking. *ACM Trans Softw Eng Meth*, 2003, 12: 28–63
- 72 Demsky B, Rinard M C. Goal-directed reasoning for specification-based data structure repair. *IEEE Trans Softw Eng*, 2006, 32: 931–951
- 73 Egyed A. Fixing inconsistencies in UML design models. In: *Proceedings of the 29th International Conference on Software Engineering*. Washington: IEEE Computer Society, 2007. 292–301

- 74 Nentwich C, Emmerich W, Finkelstein A. Consistency management with repair actions. In: Proceedings of the 25th International Conference on Software Engineering. Washington: IEEE Computer Society, 2003. 455-464

Explicit environmental constructs for Internetware

LV Jian^{1,2*}, MA XiaoXing^{1,2}, TAO XianPing^{1,2}, HUANG Yu^{1,2}, YU Ping^{1,2}, XU Chang^{1,2}

1 State Key Laboratory for Novel Software Technology, Nanjing University, Nanjing 210046, China;

2 Institute of Computer Software, Nanjing University, Nanjing 210046, China

*E-mail: lj@nju.edu.cn

Abstract Situated in open, dynamic and uncertain network environments, Internetware systems must be able to sense the changes in their running environments and adapt themselves accordingly. This paper proposes to encode environmental knowledge a priori with environment models and environment specifications, and to include them as explicit parts of Internetware systems. We discuss how to use this approach to address some import issues of open environments: understanding the distributed and asynchronous nature of the environments, detecting and resolving context inconsistencies and recognizing human activities. We also come up with a preliminary methodology for the development of environment models and specifications and the corresponding implementation mechanism, which provides a good concern separation between environment handling and business logic.

Keywords Internetware, explicit environment constructs, distributed environment, consistency, self-adaptation

LV Jian is a Professor in the Department of Computer Science and Technology and the Director of the State Key Laboratory for Novel Software Technology at Nanjing University. His research interests include software methodologies, software automation, software agents, and middleware systems. Lu received his B.S., M.S., and Ph.D. in Computer Science from Nanjing University. He also serves as the director of the Software Engineering Technical Committee of the China Computer Federation (CCF).

MA XiaoXing is a Professor in the Department of Computer Science and Technology at Nanjing University. His current research interests are self-adaptive software systems, middleware systems, and cloud computing. Ma received his Ph.D. in computer science from Nanjing University.

TAO XianPing is a Professor in the Department of Computer Science and Technology at Nanjing University. His current research interests include software agents, middleware systems, pervasive computing and software methodology. He is a member of the IEEE. Tao received his Ph.D. in computer science from Nanjing University.

HUANG Yu received the B.S. and Ph.D. degrees in computer science from the University of Science and Technology of China, Hefei, in 2002 and 2007, respectively. He is currently an Associate Professor in the Department of Computer Science and Technology, Nanjing University. His research interests include distributed computing theory, formal specification and verification, and pervasive context-aware computing. He is a member of the IEEE and the China Computer Federation (CCF).