中国科学 E 辑：信息科学 2008 年 第 38 卷 第 6 期：864 ~ 900
www.scichina.com info.scichina.com
《中国科学》杂志社
SCIENCE IN CHINA PRESS

面向网构软件的环境驱动模型与支撑技术研究

吕建[①②*]，马晓星[①②]，陶先平[①②]，曹春[①②]，黄宇[①②]，余萍[①②]

① 南京大学计算机软件新技术国家重点实验室，南京 210093;
② 南京大学计算机软件研究所，南京 210093
* E-mail: lj@nju.edu.cn

收稿日期: 2008-01-03; 接受日期: 2008-02-19
国家重点基础研究发展规划(批准号: 2002CB312002)、国家自然科学基金(批准号: 60403014, 60736015)、国家高技术研究发展计划(批准号: 2007AA01Z178, 2007AA01Z140, 2006AA01Z159)、江苏省自然科学基金、教育部新世纪人才基金和国家基金委创新研究群体资助项目

摘要 以开放网络环境下的网构软件需求为切入点，在前期基于 Agent 的网构软件模型等工作的基础上，分析了环境驱动的网构软件系统的行为模式和技术需求，提出了一个面向网构软件的环境驱动结构模型；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，并构建了若干实验原型系统；初步形成了一种可适用于构建可自主感知并自动适应开放网络环境的软件系统需要的技术框架体系，从而在网构软件模型的研究上取得了进一步的进展.

关键词
软件工程
网构软件
环境
上下文
自适应

Internet的出现和普及使计算机软件所面临的运行环境开始从封闭、静态、可控逐步走向开放、动态、难控. 如何在开放、动态和难控的网络环境下实现各类资源的共享和集成已经成为计算机软件技术面临的重要挑战之一. 为适应这样一种发展趋势，软件系统开始呈现出一种柔性可演化、连续反应式、多目标适应的新系统形态. 从技术的角度看，在软件构件技术等支持下的软件实体以开放、自主的软件服务形式存在于Internet的各个节点之上，各个软件实体相互间通过协同机制进行跨网络的互连、互通、协作和联盟，从而形成一种与信息Web相类似的软件Web(Software Web). 网络环境的开放、动态和多变性，以及用户使用方式的个性化要求决定了这样一种软件Web不再像经典软件那样一蹴而就，它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. 我们将这样一种新的软件形态称为网构软件

(Internetware), 它具有自主性、协同性、反应性、演化性和多目标性等特征[1].

反观以面向对象方法与技术为基础的经典软件模型[2~5], 由于其产生于静态、封闭、可控的环境, 如果从直接性和自然性的角度考虑其对开放、动态、难控环境下软件开发、运行与维护的支撑, 则可发现其在基础软件模型、软件方法与技术、基本的支撑机制等方面存在一定的限制. 在微观上, 它具有“有限自主性、固定封装性、交互单调性”等特征; 在宏观上, 它具有“预假定的环境、紧耦合的结构、集中式的开发和重编程的应变方式”等特征. 为对开放、动态、难控环境下的软件系统的开发、运行和维护提供直接、自然和有效的支持, 许多软件新理念、新概念、新模型、新方法和新技术应运而生. 如果从软件方法学及其技术支撑的角度对上述工作加以归纳, 其发展趋势是, 应用资源联盟化、基础平台网络化、外部环境显式化、软件实体主体化、开发方式群体化、结构模型协同化、异构处理中件化、软件协同分离化、运行机制自适应、系统维护自治化、系统保障可信化等[1].

一般说来, 软件结构模型决定了软件的基本形态, 它是软件方法学的核心. 为建立面向Internet的网构软件方法学框架与技术体系, 需针对经典软件结构模型存在的“预假定的环境、紧耦合的结构、集中式的开发和重编程的应变方式”等限制, 提出以“显式化环境、松耦合结构、分布式聚合、动态化应变、可信化保障、智能化支撑”等为特征的网构软件模型. 将上述特征适当分解与组合, 本文作者提出了系统研究网构软件方法与技术体系的技术途径[6], 即“开放协同模型 ⇒环境驱动模型 ⇒智能可信模型”. 其中开放协同模型是基础, 主要特征是“松耦合结构与分布式聚合”, 它将突破经典封闭可控模型的限制, 使得软件在结构上能够适应Internet开放环境对各类资源的多模式协同与动态可演化的要求. 环境驱动模型的特征是“显式化环境与动态化应变”, 它是在开放协同模型的基础上, 探讨外部环境的特征及其各种的变化模式, 建立环境模型的框架结构, 在此基础上, 建立开放协同模型与环境模型的交互计算模式和自适应演化模式, 从而形成环境驱动模型. 更进一步, 智能可信模型的主要特征是“可信化保障与智能化支撑”, 它是在环境驱动模型的基础上, 将可信计算框架和机器学习技术等引入, 解决开放环境下软件的可信性与个性化等方面的问题, 从而最终形成一个具体可用的网构软件模型, 以及与之配套的原理、方法、技术、系统和示范应用.

在文献[6]中, 我们针对网构软件的需要, 基于软件Agent的原理、方法和技术, 以开放网络环境下资源共享与集成为切入点, 系统分析了基于面向对象方法学的经典软件结构模型及其支撑技术的限制, 提出了一种以“服务实体与协同部分分离(结构特征)、基于第三方服务实体的协同聚合(开发特征), 以及协同模式的设计与演化适应环境变化(应变特征)”为技术特征的开放协同软件模型作为网构软件的基础模型, 它可支持松耦合结构和分布式聚合. 结合其关键支撑技术的研究, 提出了基于移动Agent的协同程序设计方法、基于Agent的多模式交互中间件模型和面向体系结构的协同程序设计机制, 并探讨其在主流软件技术中的应用, 初步形成了一种适合于开放网络环境需求的开放协同模型, 为进一步的研究工作奠定了基础.

本文的工作就是在开放协同模型的基础上, 以开放、动态、难控的环境为切入点, 在系统分析网构软件结构模型的基础上, 分析了环境驱动的网构软件系统的行为模式和技术需求, 提出了以“环境信息显式化、互动方式层次化、系统结构可演化”为特征的环境驱动模型, 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, 并构建了若干实验原型系统, 从而在网构软件模型的研究上取得了进一步的进展. 本文下面部分的组织如下: 首先在第 1 节讨论环境驱动软件的宏观交互模式和结构模型; 第 2 节介绍环境模型及其关键支撑技术; 第 3 节给出一种沟通环境模型和软件协同系统的目标驱动技术; 第 4 节是为验证相关关键技术而开发的若干原型支撑系统和应用, 在最后总结全文之前, 我们在第 5 节简要讨论了相关工作及与之相比本文工作的特色.

1 环境驱动模型与框架的设计途径

一般说来, 软件系统总是在一定的环境中运行, 其主旨是服务于用户; 因此, 用户、软件系统、环境三要素之间的互动成为软件系统运行的核心. 在经典的软件结构模型中, 用户这一要素主要通过需求定义来体现, 运行环境这一要素则主要通过各种辅助文档来说明, 而相对固定的用户需求和环境支撑等要素则是作为一个全局性的前提和假设体现在软件开发、运行等各个阶段之中; 为了使得经典的软件模型及其开发与运行技术具有一定的灵活性, 研究人员从不同的角度, 提出了能够适应用户需求与环境变化的方法与技术来提升软件开发与软件系统的适应能力. 例如, 面向对象技术中的继承和多态机制就是一种提高软件系统应变能力的重要手段; XP 等敏捷开发方法学就是从开发方法的角度来适应变化; 由于经典软件结构模型所面临的环境主要是封闭、静态与可控的, 因此, 其对外部环境的应变主要采取了一种以“隐式为主的环境和静态为主的应变”为特征的处理方式.

然而, 由于网构软件处在开放、动态、难控的环境之中, 用户需求和运行环境等要素的变化是经常发生的, 且其发生的时间通常是在软件系统运行的过程之中; 因此, 要应对这样一种类型的变化, 就需要将以“隐式为主的环境和静态为主的应变”为特征的经典软件模型转变成以“显式化环境与动态化应变”为特征的环境驱动模型. 为完成这样一种转变, 本文从环境驱动模型的宏观交互模式、环境驱动模型的软件结构模型、以及环境驱动模型的关键支撑技术 3 个层面对其进行了探讨.

1.1 环境驱动模型的宏观交互模式

如何从宏观的角度来理解软件系统与外部环境的交互模式、进而进一步理解显式的外部环境和具有应变能力的软件系统, 这对于建立面向网构软件的环境驱动模型具有重要的指导意义. 一种可行的途径就是借鉴软件 Agent 领域与上下文感知计算领域所提出理论、方法和技术, 并将其应用于软件领域, 以此来尝试建立环境驱动模型的宏观交互模式.

在智能Agent领域, 常将外部环境从易存取性/不易存取性、确定性/非确定性、静态性/动态性、离散性/连续性等角度加以刻画[7]. 在上下文感知计算领域, 有 3 种有代表性的上下文定义与理解途径. 早期的研究工作主要采取举例的方式来给出对上下文的理解. Dey[8,9]从开发者的角度给出了上下文的定义: “上下文是任何可以用来刻画一个实体状况的信息”. 这里的实体是指用户与系统交互过程相关的人、地方, 或者对象, 它可包括用户及应用自身. 更进一步, Dourish[10]区分和提出了理解上下文的两种观点; 第 1 种观点是从表示性的角度来理解上下文, 第 2 种观点则是从互动性的角度来理解上下文. 按照第 1 种观点, 上下文是一种形式的信息,

可以被了解、编码和表示; 具有可界定性、稳定性和分离性等特点; 在指出第 1 种观点的局限性的基础上, 第 2 种观点认为上下文是互动过程所产生的涌现性质, 是活动与对象之间关系, 具有动态性、偶发性、关联性等特征.

综合分析上述各种观点并将其应用于环境驱动模型的宏观交互模式, 我们认为, 可从基本要素与应用场景、用户与系统的互动、环境驱动模型的模式等多个侧面入手来进行分析与提炼. 从抽象的角度, 环境驱动模型基本要素有三: 系统、用户和一组存在于环境中的第 3 方实体. 其所形成的宏观的应用场景是, 用户使用系统, 系统服务用户, 并在此过程中采取各种方式利用外部环境(主要包括第 3 方实体及其相关信息)来提高交互与服务的质量. 其宏观结构如图 1 所示.

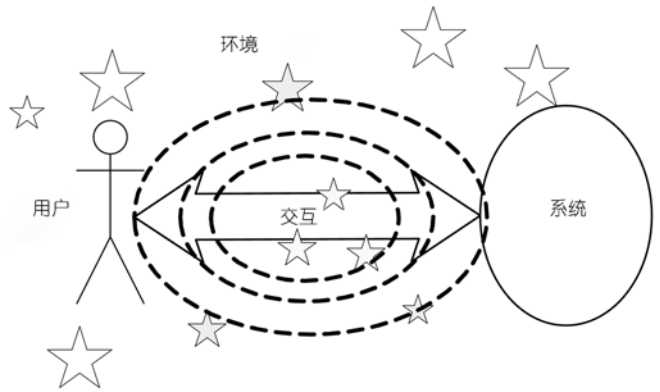


图 1 用户-环境-系统三者交互

具体说来, 以上述基本要素与应用场景作为背景, 用户与系统不断进行互动, 在互动过程中开始逐步利用第 3 方实体(可包括人、地方或对象等)来尝试改善互动方式与服务效果; 随着互动过程的发展和互动效果的改善, 逐步形成了用户与系统、相关的第 3 方实体集之间相对稳定的关联关系, 并以此来指导用户与系统的互动过程, 从而使用户与系统两者之间的互动过程演变为用户、系统与外部环境(即相关的第 3 方实体集)三者之间的互动过程与关联关系, 从而表现出一种涌现性质, 它具有动态性、偶发性、关联性等特征. 在实际的应用中, 如果这样一种涌现性质经常性的被复用, 并具有一般性的指导意义, 那么, 可从环境的角度将其提炼成一种具有可界定性、稳定性和分离性特点的上下文模式, 以一种信息的形式加以编码和表示, 从而可完成从涌现性质到固化性质的转变.

以上述讨论为基础, 可以大致区分 3 类环境驱动模型的互动模式: 第 1 种是涌现式互动模式: 此模式的主要特征是(用户、系统)与第 3 方实体集的关联关系在系统运行过程中不断发生变化, 从环境的角度, 它表现出“涌现性质⇒新的涌现性质”的循环; 第 2 种模式是固定式模式: 此模式的主要特征是(用户、系统)与第 3 方实体集的关联关系在系统运行过程中通常不发生变化, 从环境的角度, 它表现出的是单一的固化性质; 第 3 种是混合式模式: 此模式的主要特征是(用户、系统)与第 3 方实体集的关联关系在运行过程中发生间歇性的变化, 从环境的角度, 两种典型的变化模式是“涌现性质⇒固化性质⇒新涌现性质”的循环; 或者是“固化性质⇒涌现

性质⇒新固化性质”的循环. 总体而言, 目前工作重点主要集中于固定式模式系统的研究, 并逐步向混合式和涌现式的系统过渡.

1.2 环境驱动模型的软件结构模型

面向网构软件的环境驱动模型是开放、动态和难控网络环境下的分布式软件系统的一种抽象. 根据上述讨论, 其软件结构模型主要包括以下 3 个主要部分:

● 环境设施部分: 系统运行所涉及环境之相关状况及其变化规律的规约性刻画和操作性设施;

● 协同系统部分: 应用运行所需的功能部件及其协同架构, 以及支撑系统自我调整以应对环境变化的设施;

● 目标驱动部分: 依据领域知识和应用目标, 为上述环境与系统提供统一的语义解释框架, 具体化的(reified)沟通设施和模式化、可定制的驱动规则.

其宏观结构如图 2 所示.

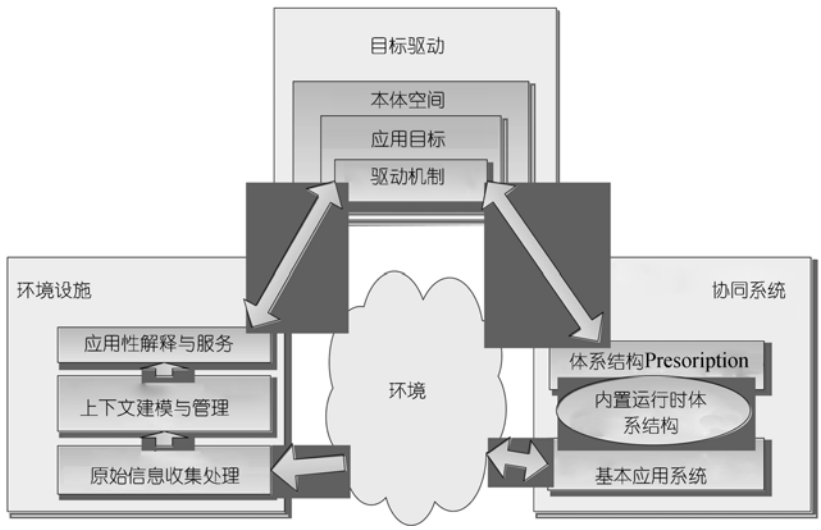


图 2 环境驱动模型的宏观结构

从软件构造技术的角度看, 环境驱动软件系统的核心挑战在于用户、环境和系统这三者之间巨大的语义鸿沟. 用户应用目标处于问题空间, 而实现系统处于解空间, 环境实体相关信息本身缺乏面向问题的解释, 更难以直接指导系统的行为. 传统软件开发过程中通过逐步分解和精化的技术来实现这些模型间单向的转换, 但这些技术已不再完全适用, 因为在环境驱动的软件系统中必须在运行时刻实现三者之间的相互沟通.

在我们所提出的模型中, 通过三重交互来实现这种沟通: 一是基本系统使用环境中的资源, 同时作用于环境, 以类似于一般分布应用系统的方式, 达成应用系统当前阶段的应用目标; 二是基本系统与内置运行时软件体系结构的交互, 通过自省计算技术来支持协同系统的动态重配置; 三是在应用目标模型和领域模型定义的语义框架下, 内置软件体系结构模型与环境

上下文模型通过一个本体(ontology)空间进行交互, 借助一组规则引擎和用户主动决策的驱动, 沟通环境运动和系统适应行为. 通过这三重交互, 可支持当前阶段环境固定模式的环境驱动模型.

进一步, 随着系统、用户以及环境三者交互的发展, 人们可能逐渐认识到新环境要素及其对系统行为的新作用. 上述模型的各主要部分均支持在线的扩展: 可以部署新的环境信息探测器, 可以增加新的上下文类型和实例以扩展上下文模型; 内置运行时体系结构可以多态地替换升级以支持非预设的动态重配置; 目标驱动本体空间松耦合的工作方式亦便于相关本体的增删和规则引擎的升级. 这种开放式的模型构造可支持环境驱动应用潜在的涌现性质及其固化.

1.2.1 环境设施

一般说来, 可从两个侧面来认识软件的外部环境, 其一是它的构成, 其二是其变化方式[1]. 从构成的角度, 外部环境实际上是指影响软件系统开发、部署、运行与维护的外部环境要素及其相互关系; 从变化方式的角度, 外部环境的要素个数及其关系是否会发生变化、如何发生变化是我们所关心的内容, 例如, 要素的个数及其关联关系在软件系统的运行过程中是否会发生改变、改变的方式是否按预想的方式进行、变化的速率如何等. 在此意义下, 所谓一个环境是封闭、静态、可控是指可对软件开发、部署、运行和维护的外部环境做出明确的界定, 构成外部环境的要素及其关系在软件生存周期中通常是固定不变的, 构成外部环境的各个要素及其关系的内容的变化是按照可控方式在预想的框架内进行变化, 从而导致软件系统可用缺省的方式来应对外部环境. 概括说来, 封闭是对整个外部环境总体特征的概括; 静态是指外部环境构成方面的特征; 可控是指在构成框架不变前提下内容变化的特征. 例如, 经典软件主要面向单机系统, 构成外部环境的要素主要包括存储大小、计算速度、输入输出、使用方法等, 在其生命周期中, 这些要素的变化有限, 并可预测控制; 因此, 在相应的软件中, 对确定环境的处理通常采取的是一种隐式缺省的方式. 在上述意义下, 开放、动态、难控则与封闭、静态、可控形成较为鲜明的对比: 开放是指对软件开发、部署、运行和维护的外部环境在通常只能做较少的界定或难以明确界定, 从而导致构成外部环境的要素及其关系在软件生存周期中是动态可变的, 构成外部空间的各个要素及其关系的内容的变化方式常常是难以控制的, 从而导致软件系统用一种非确定的显式的方式来应对外部环境的变化.

从弥合环境信息与用户目标需求以及系统实现之间语义鸿沟的角度看, 外部环境的显式化包括 3 个层面: 原始环境信息的收集与处理, 环境上下文建模与管理, 面向应用的解释与应用. 环境信息的收集与处理主要是通过一系列的探测器和量度表来感知环境, 度量感兴趣的环境实体相关属性值, 产生感兴趣的环境变动事件, 并以特定的数据结构表达之. 此层面的工作或可比拟为程序设计中的语法处理. 在此基础上, 应用本体技术, 建立环境上下文模型. 所谓本体, 是指对某一给定领域的知识实现共同理解并予以概念化, 然后给出显式的形式化的规约说明[11]. 在这个本体模型上, 进行环境上下文的冲突消解、融合、推理等工作, 同时恰当地存储和维护这些上下文本体信息, 从而达到增强环境信息的一致性、完整性和易用性的目的. 此层面的工作或可比拟为程序设计中的语义处理. 最后, 依据领域知识和应用目标, 将上述以

本体表述的上下文信息转换为当前应用问题空间下的环境因素, 并封装为应用可直接利用的服务. 此层面的工作或可比拟为程序设计中的语用考虑.

通过这种对环境的多层面的显式化处理, 我们可得到相对一致、完整、易于自动处理的环境上下文表示及其对于当前应用目标下的意义的解释, 为指导系统适应行为提供了依据. 相关关键技术将在本文第 2 节详细讨论.

1.2.2 协同系统

在网构软件的背景下, 环境驱动模型中的软件协同系统通常包括一组分布于Internet环境下各个节点的、具有主体化特征的软件实体, 以及一组用于支撑这些软件实体以各种交互方式进行协同的连接子; 整体而言, 它能够感知外部环境的变化, 通过体系结构演化的方法(主要包括软件实体与连接子的增加、减少与演化, 以及系统拓扑结构的变化等)来适应外部环境的变化, 展示上下文适应的行为, 从而使系统能够以足够满意度来满足用户的多样性目标. 我们通过将软件体系结构做为运行时刻显示表述的对象, 对传统面向对象核心协同机制(对象引用及其上的方法调用, 表述为“O.m()”)进行了重新解释, 从而扩展了分布对象模型以支持显式的灵活的协同逻辑程序设计; 并利用软件Agent技术来承载相关的协同逻辑, 从而得到一种基于Agent的开放协同软件模型, 可较好地支持多模式、可配置的软件协同及其动态演化乃至非预设的动态演化. 相关具体内容可参见文献[12].

从弥合用户目标需求和具体系统实现之间的语义鸿沟的角度看, 上述以内置运行时软件体系结构为核心的开放协同模型实际上给出了一个自省计算框架, 将抽象的软件体系结构描述规约具体化为运行系统中可见可控的对象, 并在该对象和系统之间建立了本质的因果互联, 从而沟通了具体的系统实现和抽象的软件体系结构规约. 但体系结构描述和目标需求之间仍有很大的距离. 为此, 我们进一步通过本体技术来刻画体系结构, 不但表述体系结构的事实描述(architecture description[13]), 也表述它所遵从的价值动机(architecture prescription[14]); 不但表述体系结构的静态配置, 也表达它的动态演化行为. 对于同一个系统, 其软件体系结构的这种本体表达与上述运行时实现表达具有实质上的同一性, 因而本体层面表达的演化行为可直接由内置运行时体系结构对象来实施. 上述体系结构本体的关键技术将在本文第 5 节中进一步讨论.

这样我们以软件体系结构为中心, 向下连通解空间的运行中的系统实现, 向上提供面向问题空间的抽象本体. 这种开放协同系统构造方式在为环境驱动的网构软件系统提供灵活的系统动态适应行为执行能力的同时, 也提供了进行适应决策所需的系统状态及其行为规律的抽象表述, 为目标驱动机制的设计提供了基础.

1.2.3 目标驱动

通过前述两方面的努力, 我们可将环境因素和协同系统行为在一个更为接近问题空间的抽象层面上表达. 进而, 通过对领域知识和应用目标的概念化, 可以初步形成一个沟通用户、系统和环境三方的一致的语义框架; 在此框架上施以自动或人工的推理, 有望最终完成所需的环境驱动交互. 为此, 我们一方面需要一个公共的本体表述框架, 用于维护和管理与自适应相关的用户需求、环境、软件体系结构这三方知识; 另一方面需要一套合理、高效的转换设施, 通过融合三方知识, 及时评估当前系统运行状态, 并给出恰当的系统演化指令. 此外, 如前所

述, 为了支持环境驱动系统之涌现性质及其固化, 此部分必须采用一种通用和开放式的设计.

为此, 我们采用一种基于本体空间的途径. 该本体空间包含用于描述需求知识、环境知识、软件体系结构知识的一系列的本体模型, 既有问题空间子模型, 也有解空间子模型; 既有用于刻画运行系统的子模型, 也有描述运行环境的子模型. 除前面所说的环境本体模型和体系结构本体模型之外, 还有刻画应用目标的需求本体模型. 该本体基于面向目标的需求工程[15]思想, 将用户需求表述为一棵目标精化树. 这三者的概念和语义还可能存在一定的差异, 为此我们还需要用于沟通彼此的转换本体, 其实现的功能类似本体映射[16]. 本文将在第 3 节中进一步讨论这些本体模型.

在软件支撑机制上, 上述本体使用OWL/RDF[17]来描述. 本体实例存放在一个本体空间中. 该空间及其上的一组推理引擎共同组成一个黑板系统. 这组推理引擎包括标准的OWL描述逻辑推理引擎, 各种自定义的适应规则处理引擎, 以及用户直接驱动的接口. 通过这个机制, 环境探测的信息, 比如某领域无关信息(在线服务系统的当前平均响应时间为 5 s), 可以转化为应用相关信息(系统响应不够及时), 进而转换为应用目标需求(提高系统性能), 再转换为软件体系结构操作(增加主从服务器架构中的从服务器数目); 此操作命令将递交给内置运行时体系结构对象在运行中的系统上在线的实施. 这种支撑机制也保证了新的领域知识和驱动规则能够在系统运行时刻动态地增添到本体空间中去, 新的推理引擎可以动态加入; 前述新的环境认知和新的重配置行为能力亦可以通过本体扩展的方式添加进来.

系统组织上述部件, 可为环境驱动的网构软件提供一个初步的结构模型, 如图 3 所示.

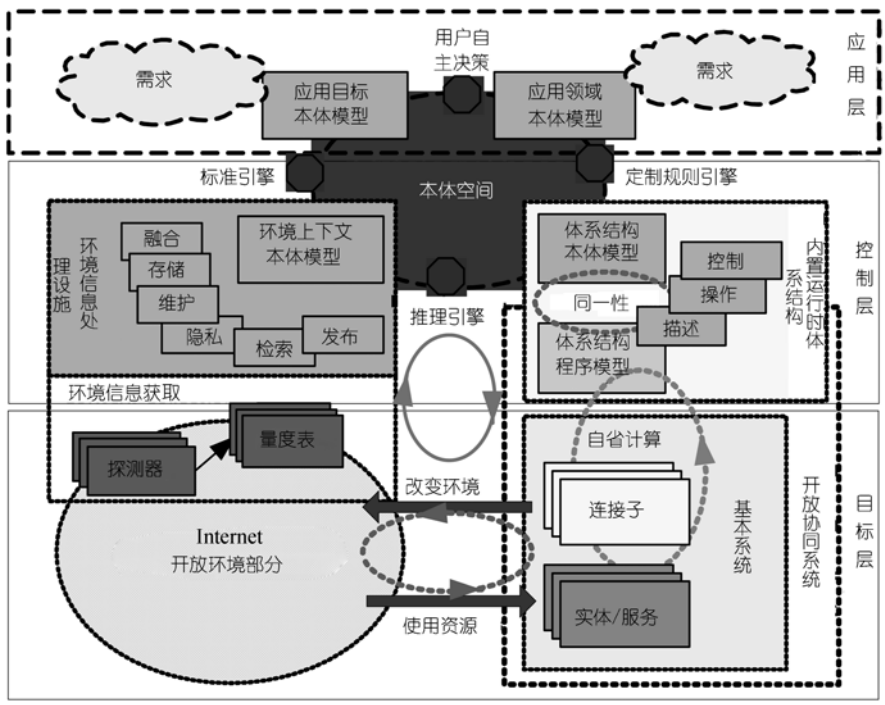


图 3 环境驱动的软件结构模型

1.3 模型与系统特征

由于需要适应显式、开放、动态、难控的外部环境, 基于环境驱动模型的软件系统在其构成、运行、目标、开发、生存周期等侧面与经典软件系统相比均存在差异.

从系统组成的角度, 基于环境驱动模型的软件系统与经典软件系统的主要不同之处可从纵横两个方面来看. 从横向的层面看, 基于环境驱动模型的软件系统是一种由分布在广域范围内的、通常由第 3 方提供的、具有主体化服务特征的一组软件实体, 通过各类风格各异的协同连接子加以互连互通而形成的联盟. 从纵向的层面看, 基于环境驱动模型的软件系统通常包括 3 个层次的内容, 即目标层、控制层及应用层. 目标层主要涉及横向层面的软件系统; 控制层则主要包括用于反映环境变化的感知部分和用于调节系统行为的演化部分; 环境与系统在应用层的应用目标和领域知识的框架下进行互动实现环境驱动的行为.

从系统运行的角度, 基于环境驱动模型的软件系统与经典软件系统的主要不同之处在于其强调了环境驱动的特性. 它的运行有两条主线, 其一是目标层系统的运行, 主要用于满足某一时段内用户的需求; 其二是控制层的运行, 主要根据环境的变化对运行系统按照所需的指标加以调整与演化, 从而使目标层系统在下一个时段能更好地满足用户的需求. 因此, 宏观上, 基于环境驱动模型的软件系统的运行是一种“满足用户阶段性目标⇒与环境进行交互⇒系统进行演化与调整⇒再满足用户新的阶段性目标”的循环过程.

从系统目标的角度, 基于环境驱动模型的软件系统与经典软件系统的主要不同之处在于其强调了时段演变特性. 目标层系统的某一方面的属性(如正确性等)只是目标层系统在一个时段内所追求目标的一个方面; 而控制层的主要作用就是通过对系统行为的调节来不断满足用户在不同时段的不同目标要求; 因此, 与强调环境驱动的系统运行相适应, 基于环境驱动模型的软件系统所追求目标的重点开始从经典软件系统的单一属性刚性追求逐步转向随时段变化而变化的用户综合柔性满意度.

从系统开发的角度, 与经典软件开发面向程序员或软件开发人员的一蹴而就特性不同, 程序设计、软件开发的细节在面向最终用户的服务抽象下被适当屏蔽, 软件开发过程就是不断获取最佳的服务与增值服务的过程, 在此过程中, 由主体化的软件实体与分离的协同机制所支持的面向用户的成长式开发方式将基于构件组装的软件复用技术等各种软件开发技术隐含其中.

从生存周期的角度, 与经典软件以开发为主的静态模型不同, 基于环境驱动模型的软件系统的生存周期强调的是一种演化适应为主的动态模型. 一种可能的模型是“需求分析及其变化预测+系统环境分析及其规约⇒目标系统设计+环境程序设计+感知系统设计+演化控制设计⇒基于环境驱动模型的系统实现⇒系统与环境的交互与演化 1⇒……⇒系统与环境的交互与演化 n”.

为了提供能够支持上述环境驱动模型所需的支撑技术和中间件平台, 我们在基于本体的环境模型关键技术和支撑具有环境适应能力的软件系统开发的中间件原型系统开发方面进行了一系列的工作. 这些工作将分别在下面第 2 节和第 3 节中讨论.

2 环境模型及其关键技术

如前所述, 环境驱动模型的基本特征为“环境信息显式化、互动方式层次化、系统结构可演化”. 环境信息显式化包括 3 个层面: 原始环境信息的感知与采集, 环境上下文建模与管理, 面向应用的解释与应用. 以环境上下文建模与管理为核心, 以环境信息的生存周期为线索, 环境上下文的处理和应用流程可以简单表述为图 4.

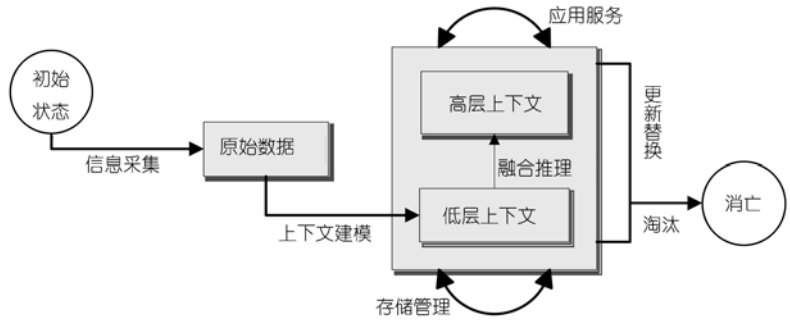


图 4 环境上下文的处理和应用流程

其中, “初始状态”环境信息数据被各类软、硬件感知器感知、获取后, 成为“原始数据”. 在“原始数据”基础上, 我们对之进行上下文建模, 确定数据的语法结构和语义信息, 成为显式的“上下文信息”. 一般来讲, 需要融合多个信息源并进行显式的逻辑推理而获得的上下文称为“高层上下文”, 否则为“低层上下文”. 上下文信息在计算机系统中需要进行有效的存储管理, 需要为应用提供高效、灵活、实用、准确的上下文信息服务. 随着应用的不断进行、环境的不断变化, 上下文信息可能不断更新和替换; 如果环境上下文信息不再为任何应用系统所使用, 它便进入消亡状态.

2.1 上下文关键技术框架

从软件的角度分析上述环境上下文的处理和应用流程可以看出, 环境信息显式化的核心需求在于上下文建模、上下文管理和上下文应用; 以此为切入点, 一种上下文关键技术框架可如图 5 所示.

图 5 上下文关键技术框架

在上述技术框架中, 上下文建模是上下文信息的管理和应用的基础, 通常情形下, 它至少包括描述框架、语义模型和建模方法 3 个方面的技术内容. 在上下文建模基础上的是上下文管理层, 它将涉及到上下文信息的存储、管理和信息融合等诸多方面的技术内容, 为上下文信息的使用提供良好的技术支撑. 技术框架的最上层是上下文应用层, 它在上下文管理层的支撑下, 直接面向应用, 提供统一、高效、便捷、准确、个性化上下文信息服务和与应用模型相关的技术.

为了探讨环境模型及其关键技术, 我们以上述技术框架为指导, 对上下文的建模、管理和应用等关键技术进行了系统的研究. 在建模层, 我们对上下文描述框架进行研究, 提出了一个基于本体的动、静态上下文统一描述模型, 并就本体建模工程进行了初步尝试; 在管理层, 我们结合上述模型, 对上下文存储模型、上下文融合/推理技术、一致性保障等关键技术进行研究, 提出了本体推理和规则推理相结合的推理引擎[18]和上下文冲突的回溯消解方法[19], 并对上下文质量保障进行了探讨[20]; 在应用层, 我们结合智能空间中间件FollowMe[18]的研制, 在上下文检索技术、上下文发布/订阅技术、个人隐私保护技术、应用中间件技术、应用编程模式等方面开展研究并取得了一定的进展; 例如, 在检索方面, 提出一个基于距离的上下文查询优化算法[21]; 在隐私保护方面, 提出基于模糊度计算的隐私保护方法[22]; 在上下文发布方面, 给出一个基于协商的上下文选取方法[23]; 在应用编程方面, 给出一个可插拔应用编程模型[18]等. 上述研究成果基本贯穿了上述技术框架的各个层面, 初步形成了可支持环境显式化的技术支撑体系. 下文将简要介绍我们在本体建模、查询优化、隐私保护和上下文选取等方面的相应工作.

2.2 基于本体的动态上下文建模

上下文建模是表达、理解和利用上下文的关键所在. 就建模内容而言, 通常有两类上下文, 一类是静态上下文, 另一类是动态上下文. 静态上下文一般指生存周期较长的上下文, 如“甲是教师”, 由于职业一般不会频繁变化, 这条上下文的生存周期通常会比较长; 与之相对, 有些上下文生存周期比较短, 如“甲在房间里”, 则称之为动态上下文. 目前, 静态上下文的建模工作已有不少, 但涉及到动态上下文, 尤其是形式化动态上下文建模方面的工作却不多见. 在形式化上下文建模方面, 基于本体的建模方法相比其他方法而言更加符合上下文建模方法所应具有的特性[24]. 本体是共享的概念模型的形式化规约, 基于本体的上下文模型使得我们能够更好地进行上下文共享、逻辑推理和知识重用[25,26].

基于上述分析, 围绕动态上下文建模问题, 本文分别采取了在静态本体基础上增加时间信息以及直接建立动态本体两种方法对其进行探讨, 提出了增强型本体上下文建模方法以及基于动态本体的上下文建模方法.

2.2.1 增强型本体上下文建模

如前所述, 环境上下文包括静态上下文和动态上下文两类. 就动态上下文而言, 时间信息是其关键, 然而, 静态本体建模技术及其工具(如 Jena 等)缺乏对其有效的描述、管理和应用的支持, 较难支持含有时间信息的动态上下文建模和管理. 因此, 对静态本体模型进行时间信息描述、管理方面的增强成为一种有效的动态上下文建模途径. 基于以上分析, 本文提出了一种

增强型本体建模方法, 该方法采用静态本体进行静态上下文建模, 在此基础上扩充了时间戳、生存期等描述和管理手段进行动态上下文建模.

细言之, 静态本体模型采用传统做法, 由高层本体和领域本体两层构成, 主要定义了公共共享概念、领域相关概念及其关联关系. 静态本体采用 RDF 三元组(subject, predicate, object)表示, 如图 6 所示.

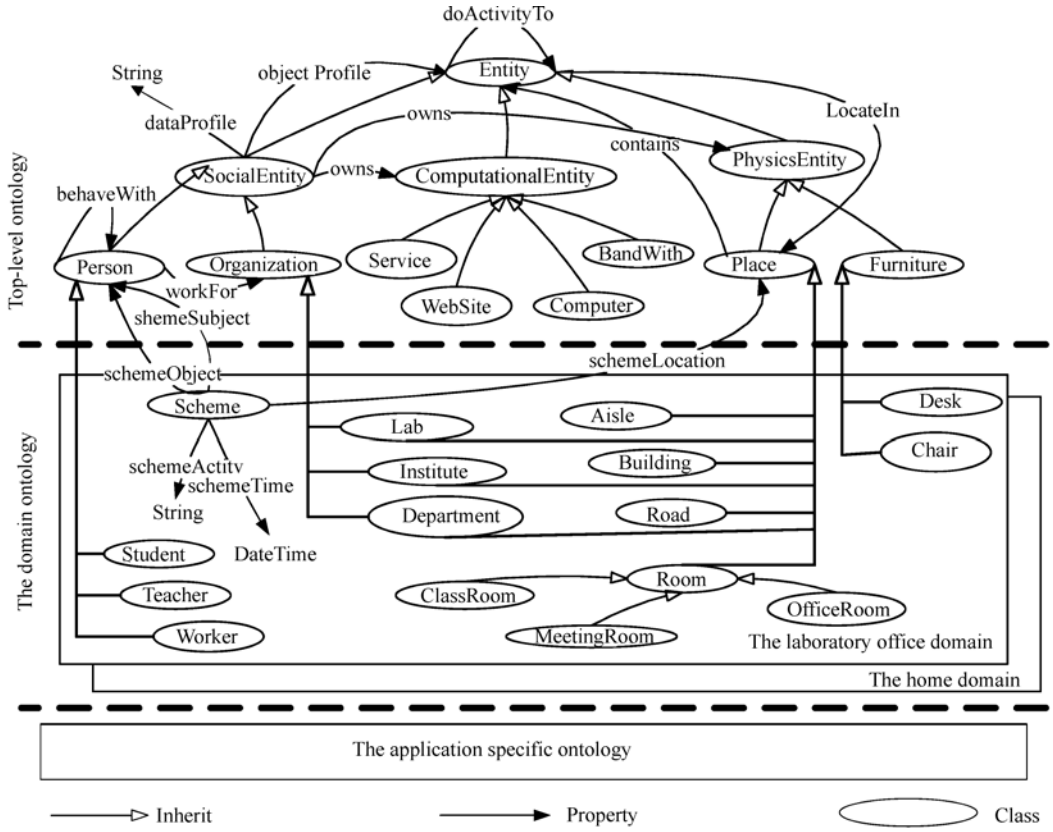


图 6 双层本体结构

在上述模型基础上, 静态上下文被组织为上述静态本体的实例化集合, 如(Tom, type, Teacher). 在动态上下文建模方面, 我们扩充了传统的 RDF 三元组表现形式为五元组(subject, predicate, object, ttl, timestamp), 增加了上下文生存期 ttl 以及生成时间戳 timestamp 等描述分量. 同样, 动态上下文也是上述增强型本体的实例化集合, 如(Tom, LocateIn, RoomA, 1200, 20071025034524).

在上述增强型本体建模基础上, 我们在智能空间支撑中间件FollowMe[18]上对上述模型进行了应用尝试, 并通过增加的时间信息开展了动态上下文生命周期、冲突消解和应用方面的研究[20], 实验结果表明上述模型在静态上下文处理方面具有良好的表现, 但在动态上下文处理方面尚有可改进之处. 其中突出表现在动态上下文的融合、推理等方面.

2.2.2 基于动态本体的上下文模型

开放、动态计算环境中的环境上下文除了刻画动态属性的时间分量之外, 尚有很多状态变化之类的信息需要获取和刻画. 比如“Tom 离开 RoomA”. 由于各种环境设备只能感知到客体所处的状态, 而难以感知到状态的变化, 比如位置感知设备能感知到人所处的空间, 而无法感知空间信息的变化. 因此, 此类信息的获取必须依赖描述状态的上下文进行推导而获得. 当然, 这样的推理工作可以在应用层面上利用时间信息等方法进行处理, 但是这种时间、状态相关的推理需求普遍存在于环境驱动应用, 因此我们希望能在本体层对时间和动态上下文进行建模, 使得使用这个模型的应用能够从中获取便利.

出于这样的考虑, 我们在原有工作的基础上, 结合W3C的工作草案OWL-Time[1)]和UML状态图[27]理论, 提出了一个基于动态本体的上下文模型. 该模型仍然由本体层和实例层构成, 在本体层, 尤其是高层本体中, 我们除了沿用了经典高层本体中的ConcreteEntity及其子类外, 引进了OWL-Time中的TemporalEntity及其子类, 自行设计了包含State, Event, Transition, Condition和Operation等及其子类的DynamicEntity, 其结构示意如图 7.

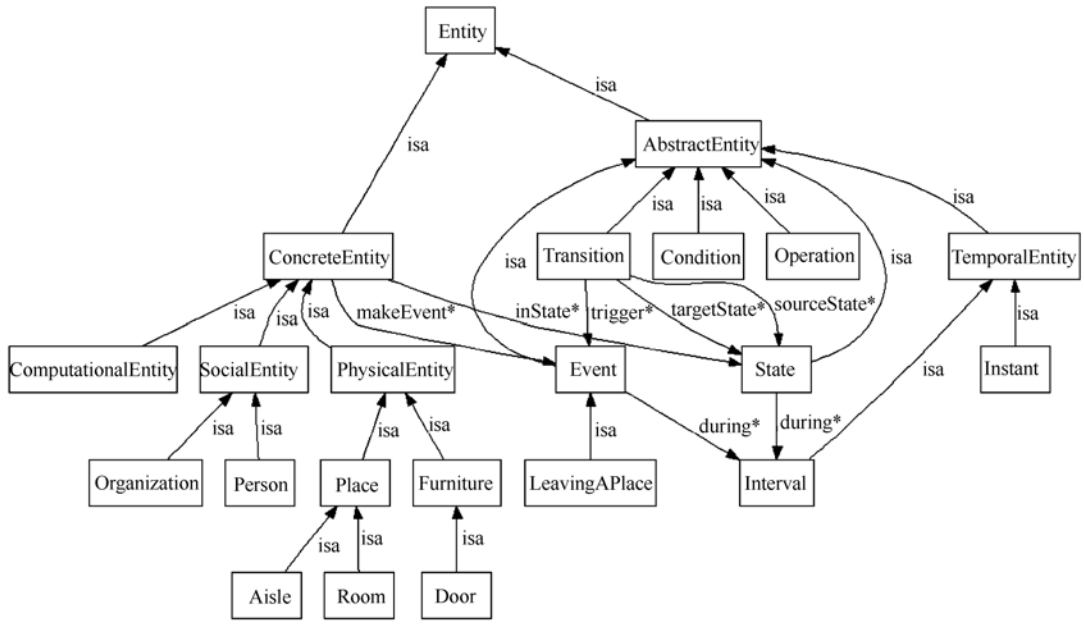


图 7 基于动态本体的上下文模型

在上述本体模型中, 我们还针对新引入和设计的 TemporalEntity 和 DynamicEntity 给出了相应的语义说明和推导规则, 并在 FollowMe 系统中借助“环境驱动的自动门禁”需求开展了相关实验, 实验结果表明, 该模型及其推理机制能够较好的满足时间相关、状态相关的环境信息表示、融合和推理的需求.

1) http://www.w3.org/TR/owl-time/

2.3 基于距离的上下文聚类查询

描述开放、动态环境的上下文具有鲜明的分布性和海量特性. 在分布的环境中如何对海量的上下文数据进行高效的查询, 成为我们面临的一个迫切的问题.

已有工作在处理这个问题时, 主要采用了对上下文的本体语义进行索引分类的方法来进行查询加速[28,29]. 但是, 本体的定义缺乏统一的标准, 使得这种方法难以实用化. 针对这个问题, 我们首次从上下文查询规律分析的角度定义了上下文间的逻辑距离, 并在此基础上提出了一种新型的上下文索引方法[21]. 我们的工作的基本思路是首先给出一个上下文形式化标记规则, 然后在此标记基础上给出两个上下文的"查询距离"定义, 当我们判定出上下文之间的距离之后, 采用聚合的方法建立上下文之间的逻辑簇, 最后利用聚类结果进行上下文查询优化.

2.3.1 上下文形式化标记

设共有m种上下文描述环境, 记为$\{C_1, C_2, \cdots, C_m\}$, 环境中有$q$个上下文感知的应用, 记为$\{A_1, A_2, \cdots, A_q\}$, 其中$C_i$ $(1 \leqslant i \leqslant m)$定义为$q$维向量,

$$C_i = (p_1, p_2, \cdots, p_q), \tag{1}$$

其中p_j $(1 \leqslant j \leqslant q)$ 是A_j在单位时间内访问C_i的次数. 这个向量形式地定义了C_i的查询模式.

2.3.2 上下文间的距离

给定两个上下文S和T(例如, S为光照强度, T是温度), 记$S = (s_1, s_2, \cdots, s_q)$, $T=(t_1, t_2, \cdots, t_q)$, 首先计算$S$和$T$的标准化内积.

$$\cos(S,T) = \frac{\sum_{j=1}^{q} s_j t_j}{\sqrt{\sum_{j=1}^{q} s_j^2}\sqrt{\sum_{j=1}^{q} t_j^2}}. \tag{2}$$

上式从查询模式的角度给出了 S 和 T 的相似度. 接着, 我们定义 S 和 T 的距离如下:

$$D(S,T) = \sin(S,T) = \sqrt{1-\cos^2(S,T)}. \tag{3}$$

可以证明, 此定义下的 D 满足非负($D(A, B)\geqslant 0$)、自反($D(A, A)=0$)、对称($D(A, B)=D(B, A)$)和三角关系($D(A, B)+D(B, C)\geqslant D(A, C)$), 所以这是一个良定义的距离度量. 不严格地说, 如果两个上下文经常被应用同时查询, 那么在上述定义下它们倾向于具有较小逻辑距离.

2.3.3 上下文聚类

基于公式(3)定义的距离度量, 我们将彼此"临近"的上下文聚类到一个簇里. 即同一个簇里的上下文有较为相似的查询频率和查询局部性. 聚类算法基于 k-means, 描述如下.

1) 任意选出 k 个上下文作为簇的中心;

2) 对于余下的每一个上下文, 将它放入距其中心最近的那个簇里. 距离以(3)式计算;

3) 重新计算每个簇的中心, 以距所有成员的平均值最近的那个上下文作为中心;

4) 如果中心都没有改变, 算法终止, 否则转到步骤 2).

这个算法可以证明是收敛的. k 需要预先确定.

2.3.4 应用聚类结果提高查询效率

无结构的 P2P 网络结构是一种典型的分布数据存储结构, 也是一种重要的海量上下文存储和管理环境. 我们选择无结构 P2P 网 Gnutella 开展了上述工作的应用, 对其泛洪查询方式进行了优化. 优化方法将在包含属于同一个簇的上下文节点之间建立捷径, 即缓存对方的节点名和 IP 地址. 这样, 每当查询到其中一个时, 若要再访问被缓存的节点只需一跳, 无需洪泛. 可以看出, 在前文的聚类算法中, k 越小, 则建立的捷径越多, 平均访问跳数越小; 但同时缓存开销越大, 网络存储结构变化时维护开销越大. 在实际使用中, 要根据具体情形对 k 作权衡.

2.3.5 性能分析和比较

在上述实验过程中, 我们对查询效率和距离度量的有效性进行了分析.

在和 Gnutella 的比较中, 当 k=10, 节点数为 1000 时, 上下文查询的平均搜索路径缩短了 20%, 在节点数为 10000 时, 平均搜索路径缩短了约 40%, 性能改善很显著. 而且节点数越多, 我们的方法性能提高越明显. 当固定节点数时, k 越小, 平均搜索路径越短, 但缓存开销越大, 并且当 k 越来越小时, 搜索性能提高幅度明显下降. 实验证明, 在应用数为 30, 上下文的种类为 128 时, k=10 在性能提高和存储空间开销间取得了较好的平衡.

我们对公式(3)中提出的上下文距离的度量与常见的 Euclid 距离作了对比. 性能评估结果显示, 对于相同的聚类算法, 在节点数超过 100 后, 以公式(3)作为距离度量的性能始终领先使用 Euclid 距离作为度量时的性能.

提高上下文的查询效率是上下文聚类的一个重要应用, 但是后者的作用并不仅限于此. 举例来说, 如果我们采集大量的现实世界中的上下文感知应用的查询请求, 将其作为我们聚类算法的输入, 我们可以不涉及人为定义的本体而得到不同上下文间的语义联系. 如同经常同时出现在同一个网页上的两个词语间有一定程度的语义联系[30], 可以想象, 两种经常被同时查询的上下文间也应有语义关联. 上下文聚类还有很多未被重视的用途, 这也是我们的后续工作之一.

2.4 基于模糊度计算的隐私保护

运行于开放、动态环境中的软件系统将会不断地主动收集各类环境信息尤其是用户信息, 并开展自适应的服务以满足用户需求. 在这个过程中, 对用户的各类私密信息的保护成为必要. 隐私是数据拥有者在个体信息被收集、处理和使用过程中对其信息拥有的特定暴露权力[31]. 计算过程中针对环境信息的隐私保护则是为保障隐私权利的行使而提供的各类基础设施、基本方法、支撑工具等.

计算环境中的上下文发布过程涉及以下 3 种最基本的参与者(如图 8): 数据拥有者和数据收集者以及数据使用者.

数据收集者可以自动从数据拥有者处收集上下文信息; 数据收集者将这些上下文信息整理并保存于数据库中; 数据使用者通过查询数据收集者获取并使用这些上下文信息. 拥有者在整个上下文发布中对这些信息拥有特定的隐私权利. 满足隐私保护的上下文发布只能将信息在合适的场合发布给合适的人. 也就是说, 数据拥有者可能希望在隐私的暴露尺度方面找

到更多的选择余地, 以便在不同的应用场景下、针对不同的使用者提供不同尺度的暴露. 另一方面, 数据的使用者也可能有诸如“数据使用匿名”等方面的隐私需求.

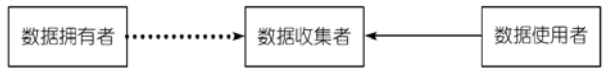


图 8 上下文发布过程参与者

基于上述考虑, 我们考虑了不同的数据加工方式[32~34], 设计了一个基于模糊度计算的隐私保护方法——Shadow方法, 并在FollowMe系统的基础上, 部分实现了该方法[22].

针对数据拥有者的隐私保护往往以制定访问控制策略的方式进行. 数据拥有者可以制定上下文信息的发布策略, 数据收集者负责根据这些策略对使用者的访问进行控制, 并对上下文信息进行处理. 策略 p 采用如下表示形式:

$$p = (subject, target, action, condition, dd).$$

策略p_1=(*student*, *FollowMeGroup*, *faculty.getLocation*(), *rest_time*, 0.3)表示: 如果“*student*”在*rest_time*时, 查询“*FollowMeGroup*”中的“*faculty*”位置信息, 则暴露度为 0.3. 在一次新的查询中可能没有适用的已制定策略, 则需要系统可以自动生成, 换言之, 需要根据历史策略信息, 制定新的策略a= (*subject*, *target*, *action*, *condition*, v_a): 其中v_a计算如下:

$$v_a = k \sum_{p_i \in SP} v_i \frac{t(d(a, p_i))}{\sum t(d(a, p_i))}, \tag{4}$$

其中*SP*表示具有相似*subject*, *target*, *action*以及*condition*的历史策略集, k为标准化参数, t()表示阈值函数. $d(a, pi)$表示策略a与策略p_i的近似程度.

通过策略得到暴露度*dd*后, 数据收集者需要根据该*dd*值对它保管的私密信息进行模糊加工并对外提供. 加工方法分两步, 首先计算出所有私密信息的“模糊度”, 然后和策略中的“暴露度”进行比较, 选择合适信息对外发布. 我们的模糊算法是基于上下文本体模型设计的. 以图 9 列出的某医院空间位置本体图(片断)为例.

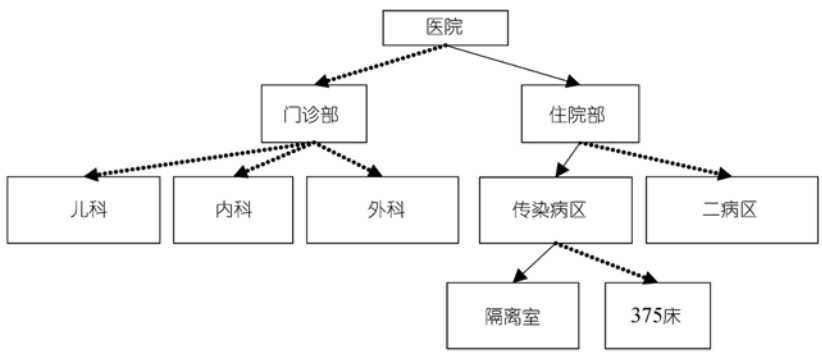


图 9 某医院空间位置本体图(片断)

对于任意本体 x, 我们定义下 *x.from* 和 *x.to* 来作为 x 的定义域与值域. 若 *a*=*x.from* 并且

$b=x.to$, 则 $b=a.child$ 以及 $a=b.parent$. 根据下式, 我们可以计算出上述模型中任意本体的向下模糊度 $BD(x)$:

$$\begin{cases} BD(x)=\begin{cases}(ACC(x))^{-1}+\omega(x)DIFF(x), & \text{当}\left|x.child\right|\neq 0,\\ 1, & \text{当}\left|x.child\right|=0,\end{cases}\\ \omega(x)=\sum\limits_{x_i\in x.child} p_i BD(x_i)\Big/\sum\limits_{x_i\in x.child} BD(x_i),\\ ACC(x)=MAXHEIGHT(x)+1-IMP(x),\\ DIFF(x)=(ACC(x)-1)^{-1}-(ACC(x))^{-1}, \end{cases} \tag{5}$$

其中, $IMP(x)$表示从根节点到 x 的最长路径长度, $MAXHEIGHT(x)$表示子树 x 的高度.

在图 9 中, $MAXHEIGHT$=4, IMP(隔离室)=4, IMP(医院)=1. 由上式可以得出, “隔离室”和“375 床”的 BD 值都是 1, “住院部”的 BD 值是 0.42.

如此, 若有患者目前处于隔离状态, 该患者对自身位置信息的暴露度定义为 0.5, 则任何查询者均只能得知该患者处于“住院部”. 位置信息“住院部”即“隔离室”的一种模糊方式.

此外, 在Shadow方法中我们考虑了两种不同的匿名: ID匿名与上下文匿名. ID匿名指数据使用者在发布其上下文信息时使用假名来实现条件匿名. 上下文匿名指对查询的上下文信息使用匿名方法来达到保护隐私的目的: 匿名方法可以将某条上下文信息隐藏于其余信息(等价类)中, 如K-anonymity算法[35]、L-diversity算法[36]以及T-closeness算法[37].

传统的匿名算法仍有不足, 其用以匿名的等价类所包含的非必要信息可能导致新的隐私泄漏隐患:

● 传统匿名算法使用混淆的方法来保护目标记录, 但是目标记录本身仍然在等价类中, 如果攻击者通过背景知识可以忽略等价类中的一些候选目标, 则混淆方法对目标记录的保护将显得不够.

● 传统匿名算法中牵涉到的其他记录都没有经过匿名外的其他数据加工. 攻击者在查询时往往可以获得意外的信息.

● 传统匿名算法中牵涉到的记录都真实存在于数据库中, 那么一个包含记录 X 的等价类可以推出“X 真实”, 这种信息很可能是一种隐私泄漏.

● 传统匿名算法保证分组是固定的, 但是当属性变化时, 攻击者仍然可以因为等价类的变化察觉到属性的变化. 属性变化的趋势可能成为一种隐私泄漏.

综上考虑, 在Shadow隐私保护方法中, 我们改进了T-closeness算法, 并在该算法返回的等价类中加入冗余信息, 以更有效的保护隐私[22].

2.5 基于协商的上下文信息选取机制

环境的开放性, 会使得不同的上下文信息提供者提供的上下文信息有所冲突[38], 在特定的应用中选取适当的上下文信息也就显得尤为重要. 此时, 当上下文信息查询者不能自行决定如何选取时, 提供者之间的协商成为主要手段.

针对上述问题, 我们提出了一种带有奖赏的协商机制来进行上下文信息选取. 在这种机

制下, 上下文信息提供者们互相协商, 它们对谁提供上下文信息和如何分配收益这两个问题达成一致. 其基本流程如图 10 所示.

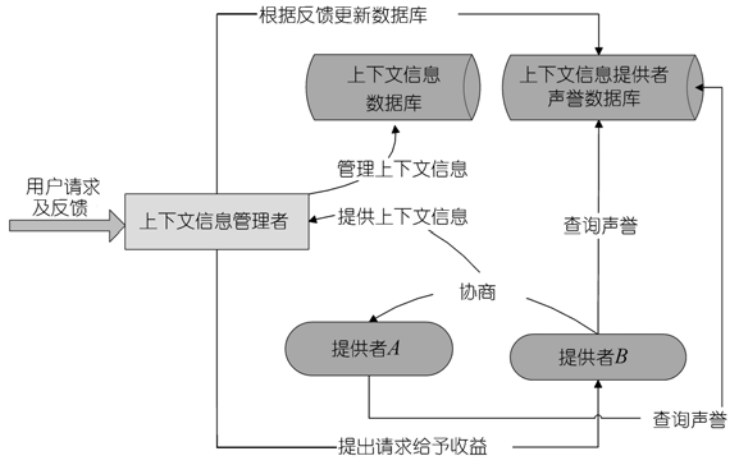


图 10 提供合适上下文信息的协商机制

应用向上下文信息管理者请求信息, 管理者发现上下文信息提供者 *A* 和 *B* 都提供符合要求的信息. 此时, 管理者通知 *A* 和 *B* 从数据库中获取自己的声誉信息, 然后两者进行协商以决定谁提供上下文信息. 信息管理者在把得到的上下文信息提交给应用的同时, 将其存入上下文信息数据库. 应用根据上下文信息的质量给出反馈, 管理者根据反馈更新提供者的声誉并分别给予两个提供者以不同的收益.

在上下文信息提供者*A*和*B*具体协商时, 协商双方以给出下次协商时的奖励方式说服对方[39], 提供者利用评估函数对当前利益和将来利益进行决策.

在 *t* 时刻上下文信息提供者 *A* 关于提供信息收益及奖励的评估函数 *U* 如下:

$$U_A(o,ep,t)=(w_1^A\cdot U_c^A(c)+w_2^A\cdot U_{ep}^A(ep))\cdot\delta_A(t), \tag{6}$$

其中, *o*表示*A*提供的offer, *o*=(*c*, *p*), *c*代表提供的上下文信息, *p*代表收益可能获得所占百分比; *ep*表示承诺对方下次协商所支付的奖励或者收到对方的奖励所占比率, 支出为负收入为正; *t* 表示当前时刻; w_1^A 和 w_2^A 代表收益和奖励的权值, $w_1^A+w_2^A=1$; $\delta_A(t)$ 是*t*时间的一个处理函数. 详情可参考文献[23].

A 被选中的可能性, 也就是 *A* 获得收益所占百分比

$$U_c^A(c)=w_{c1}^A\cdot fd(d)+w_{c2}^A\cdot f_{ref}(ref), \tag{7}$$

其中, 函数 *fd*(*d*)表示量化后的信息提供者的距离值; 函数 *frep*(*rep*)表示量化后的信息提供者的声誉值; w_{c1}^A 和 w_{c2}^A 代表距离和声誉值的权值, $w_{c1}^A+w_{c2}^A=1$.

在具体环境中, 很多协商都是会重复出现的, 根据实验, 我们发现:

1) 一般情形下, 近距离和高声誉的提供者将会提供上下文信息, 失败的一方得到较少的收益;

2) 即使提供者距离稍远或是声誉稍低, 但若其愿意在下次协商中提供给对方奖励, 它仍有可能提供上下文;

3) 一般说来, 在其他条件相同, 声誉较高者有更多的机会提供上下文信息, 同时, 在完成任务后, 他的声誉将更进一步的增加.

3 目标驱动与 Rearon 本体

面对开放、动态、难控的环境, 为维持服务质量, 保证应用目标的达成, 基于环境驱动模型的软件系统需依据上述显式化的环境上下文模型所提供的信息来动态调整协同系统的结构和行为. 此处的关键在于如何依据领域知识和应用需求来解读环境上下文信息, 将其转化为应用目标相关事件; 进而依据应用目标的体系结构指示(prescription)规则, 得到体系结构行为命令, 从而可以驱动协同系统的动态重配置行为. 为此首先需要有一个面向用户的、较为通用而又便于一定程度自动推理的表达机制来描述应用目标、环境上下文和协同系统体系结构指示. 其次在此描述机制基础上需要一个易定制、易扩展、合理高效的转换设施, 通过融合三方知识, 及时评估当前系统运行状态, 并给出恰当的系统演化指令.

作为给定领域内知识之共同理解的概念化, 本体这种显式的、形式化的规约技术可为在一个较高的抽象层面上克服不同实体(包括人、计算部件)之间的交互障碍提供了有效支撑[11,17]. 恰当选择本体描述语言(如OWL-DL), 还可利用较为成熟的逻辑工具(如描述逻辑)来实现高效的自动推理. 我们基于OWL设计了一套用于描述需求目标、环境上下文和软件体系结构的本体, 称为Rearon(requirement and architecture ontologies). 本节将首先讨论Rearon本体, 然后介绍基于Rearon所设计的转换驱动设施.

3.1 Rearon 本体

如图 11 所示, Rearon 包含一系列的本体模型, 既有问题空间子模型, 也有解空间子模型; 既有用于刻画运行系统的子模型, 也有描述运行环境的子模型. Rearon 可以分为 3 类子模型: 需求本体(requirement ontology, RO), 体系结构本体(architecture ontology, AO), 环境本体(context ontology, CO). 在这 3 个子模型的边界上, 是用于沟通彼此的转换本体(transformation ontologies, TO), 包括需求-体系结构转换本体(requirement-architecture transformation ontology, RATO)和环境-系统转换本体(context-system transformation ontology, CSTO).

3.1.1 需求本体

需求本体基于面向目标的需求工程[15,40]思想, 分 3 个层次来组织需求本体:

1) 元层(meta level)本体是与领域无关的抽象概念集, 包括了 Goal, Object, Agent, Action 等概念;

2) 领域层(domain level)本体是某个应用领域的公共概念集, 比如对于“在线旅行服务”这个应用领域来说, 有“订票要求满足”, “旅游出行”, “计划”等概念;

3) 实例层(instance level)本体[1)]用于描述某个应用领域的特定实例, 比如“2008 年元旦北

1) 我们使用OWL来表述本体. 在OWL中“本体”这一术语的内涵被扩展了, 它包括实例数据.

京一日游”是“在线旅行服务”这个应用领域中“旅游出行”本体的实例.

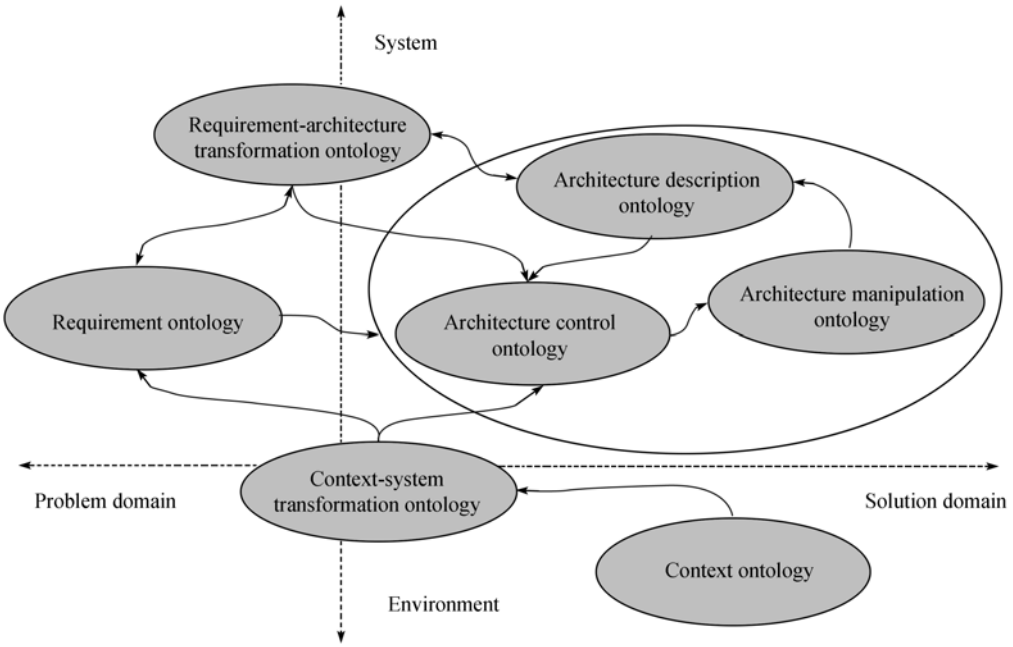


图 11 Rearon 系列本体

虽然该需求本体是比较通用的, 但是我们不期望它成为面向目标的需求工程的另一个描述工具, 而仅用其描述环境驱动软件适应行为相关的需求信息. 因此这里存在一个从中选取与适应环境相关的需求信息的问题. 需求工程中的目标精化树(goal refinement tree)[41]可以帮助我们解决这个问题. 在目标精化树中, 层次越高的结点越抽象, 可操作性越差; 层次越低的结点越具体, 可操作性越强. 我们并不需要表达整个目标精化树, 我们只需要选取某些合适的结点即可. 例如, 对于一个在线票务系统来说, 它的功能性目标包括票务查询、票务预定等, 非功能性目标包括浏览速度、安全性等. 一般而言, 这个系统的功能性目标短期内不会变动, 而易受环境变化影响的是非功能性目标中的浏览速度. 对于浏览速度这个非功能性目标来说, 它受到网络状况和系统响应速度两方面的影响, 因此它又可以分为两个子目标: 网络时延和系统响应速度. 同目标系统自身相关的是系统响应速度这个子目标, 因此我们就从在线票务系统的目标精化树中选择系统响应速度这个非功能性目标作为环境驱动的软件自适应开发中的需求元素. 相关具体描述语法可参见文献[42].

3.1.2 体系结构本体

如第 1.2.2 节所述, 我们的体系结构本体不仅需要描述体系结构的静态配置, 也要描述它的动态演化行为, 还要描述依据上述需求目标对其演化行为的控制. 据此我们将体系结构本体分为 3 类: 体系结构描述本体(architecture description ontology, ADO); 体系结构操纵本体(architecture manipulation ontology, AMO); 体系结构控制本体(architecture control ontology,

ACO). ADO 和一般意义上的体系结构描述语言(ADL)相对应, 主要用于描述静态的软件体系结构知识; AMO 定义操纵软件体系结构的动作; ACO 把 AMO 和 ADO 联系起来, 描述如何利用 AMO 定义的动作调整 ADO 定义的软件体系结构. AMO 和 ACO 结合实现软件体系结构的动态管理.

此外, 在软件体系结构的研究中发现, 对特定应用领域来说, 其系统组织方式往往存在着某些共性, 通过总结这些共性, 可以得到一些惯用的模式, 称为体系结构风格(architecture style)[43]. 体系结构风格为设计人员的交流提供了公共的术语空间, 促进了设计复用和代码复用. ADO同样也应支持体系结构风格, 故它也可以分为 3 层: 元层(meta level), 风格层(style level), 实例层(instance level).

● ADO

ADO的设计以ACME[44]为参考模型, 3 个层次包含的内容如下:

1) 元层本体包括构件(component), 连接子(connector), 系统(system), 端口(Port), 角色(Role), 表示(Representation), 表示-映射(Rep-map)等 7 个核心概念;

2) 风格层本体包含专属于某个体系结构风格的概念集. 比如 Master/Slave 体系结构风格本体, 其包含的专有概念包括 Master, Slave 等, 它们除了具备一般构件的属性外, 还具有某些只属于 Master/Slave 体系结构风格的特殊属性;

3) 实例层是对某个特定应用的体系结构的描述, 它是本体的实例. 比如在我们使用 Master/Slave 体系结构风格搭建的在线票务系统中, 存在着处理公共汽车票务的“aBusTicket-Service”、处理飞机票务的“aPlaneTicketService”以及处理火车票务的“aTrainTicketService”等“Slave”本体的实例.

● AMO

AMO 用于定义操纵软件体系结构的动作集. 与 ADO 对应, AMO 也分为元层, 风格层和实例层 3 个层次. 此外, AMO 各层包含的动作根据其作用范围, 可以分为体系结构级动作和局部动作.

1) 元层本体中, 体系结构级的动作是体系结构升级动作 UpgradeArch, 这是一个通用的父概念, 在风格层会具体化为各个体系结构风格中对应的子概念. 元层的局部动作包括基本元素的增删, 接口之间的绑定与松绑, 逻辑部件与物理部件之间的映射及其解除等基本动作;

2) 风格层的动作与 ADO 的风格层概念对应. 对于 Master/Slave 体系结构风格来说, 体系结构级动作有 UpgradeToEMS(由常用的 Master/Slave 体系结构风格升级为扩展的 Master/Slave 体系结构风格)等, 局部动作有 AddMaster(增加一个 Master), AddSlave(增加一个 Slave)等;

3) 实例层的动作和 ADO 的实例层对应, 主要是针对某个具体应用系统的特定调整动作.

图 12 展示了一个 Master/Slave 风格体系结构的 ADO 与 AMO 部分本体.

● ACO

ACO 用于说明何时采用何种动作对软件体系结构进行调整, 它是体系结构管理知识的重要组成部分. 对于一些成熟的软件体系结构风格来说, 它们都包含一些体系结构管理知识. 比如对于 Master/Slave 体系结构风格来说, 系统的性能和 Master 和 Slave 的数目有较大的关系, 增加 Master 或 Slave 的数目, 可以有效提高系统的性能.

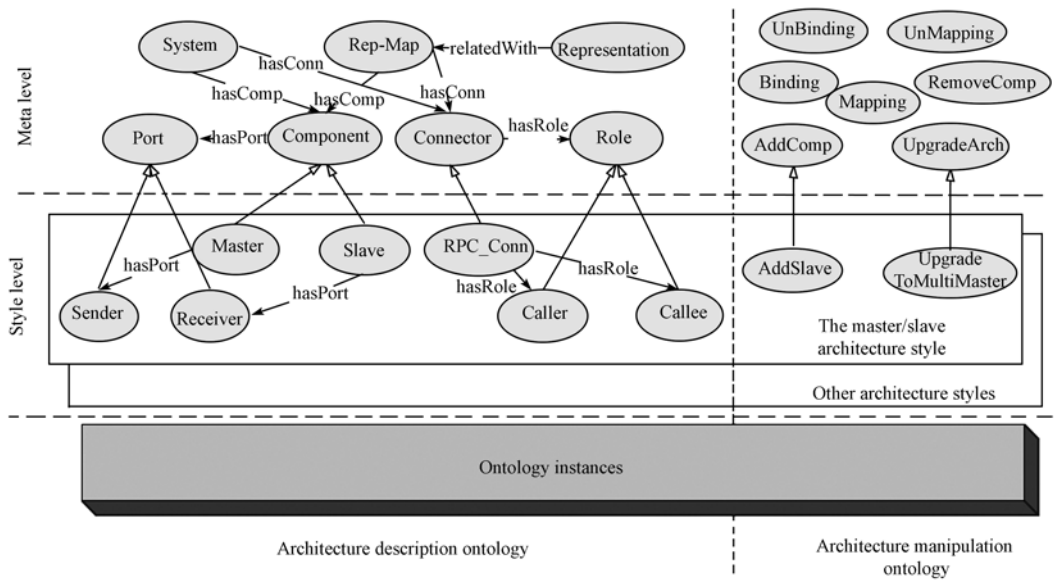


图 12 Master/Slave 风格体系结构的 ADO 与 AMO

我们采用“Condition-Operation-Consequence”的形式来组织 ACO. 其中, Condition 说明实施 Operation 的前提条件; Operation 说明采用何种动作调整软件体系结构, 它可能是 AMO 中的单个动作, 也可能是几个动作的组合; Consequence 说明 Operation 成功执行后产生的效果.

对于 ACO 来说, 不管采用 OWL/RDF 还是图示的方法描述, 其可读性都不是很好, 因此这里采用如下的简化形式来描述 ACO 的例子.

Condition: ¬satisfied(System.performance) ∧ full (Slave.capacity)
Operation: AddSlave
Consequence: ¬full (Slave.capacity)

上面的这个 ACO 是一个局部动作, 描述了 Master/Slave 体系结构风格的管理知识: 在系统的性能无法满足需求, 且所有 Slave 的容量(capacity)都已经到达极限的情形下, 我们需要增加新的 Slave 以缓解这个情形. 新的 Slave 成功添加后, Slave 的容量就不再是满负荷状态了. 下面这个 ACO 是一个体系结构级的动作和一个局部动作的组合, 表达的是在系统性能不佳且 Master 的容量又达到了极限的情形下的体系结构调整方式: 把当前的常用的 Master/Slave 体系结构风格升级为扩展的 Master/Slave 体系结构风格. 此扩展的 Master/Slave 体系结构风格在形态上比原来的增加了一个 Master, 但是由于多了 Master 后, Master 和 Slave 之间的关系以及其他系统级的属性需要调整, 因此除了执行 AddMaster 这样一个动作外, 还需要执行体系结构级的 UpgradeToEMS 动作.

Condition: ¬satisfied(System.performance) ∧ full (Master.capacity)
Operation: UpgradeToEMS, AddMaster
Consequence: ¬full (Master.capacity)

3.1.3 环境本体

有关环境的本体我们已经在第 2 节进行了讨论. 比如, 对于在线票务系统这个应用场景来说, 我们关注的是系统响应速度这个非功能性目标, 与此目标相关的情境信息有 2 个: 用户体验 UserExperience, 用于表征用户发出请求到获得结果的时间间隔; 网络时延 NetworkLatency, 用于表征借助某些工具探测到的用户和应用系统之间的网络状况. 借助这 2 个情境信息, 我们就可以判断当前系统是否满足用户需求.

3.1.4 转换本体

设计良好的RO, AO和CO可以有效地表达各自领域的概念, 同时也为连通软件体系结构和需求/情境打下了良好的基础. 但是RO, AO, CO三者的语义还可能存在一定的差异, 所以我们设计了转换本体(transformation ontologies, TO)来解决这个问题. TO实现的功能类似本体映射[16], 它包括初级转换和高级转换 2 个层面的内容. 初级转换是简单的本体映射, 这主要针对那些概念本质相同, 表述形式不同的本体; 高级转换则包含了一定的处理过程, 比如数值计算, 这主要针对那些概念本质类似, 但是存在差异的本体.

我们设计了 2 类 TO: 需求-体系结构转换本体(requirement-architecture transformation ontology, RATO)和情境-系统转换本体(context-system transformation ontology, CSTO). 其中 RATO 用于需求概念和体系结构概念之间的转换, CSTO 则用于环境概念和系统概念之间的转换. 比如对于在线票务系统来说, RO 中 Responsivity_Goal 以其属性 rate 来表达用户对系统响应速度的要求; AO 是用 System 的 performance 属性来表征运行系统的性能. 在这个应用场景下, Responsivity_Goal.rate 和 System.performance 表达的是同一个概念本质, 因此需要借助 RATO 建立两者之间的映射关系. 而在线票务系统的真实响应速度, 是通过用户体验到的时间开销和网络时延的差来体现的, 因此这里需要在 RO 和 CO 之间借助 CSTO 把 UserExperience.interval 和 NetworkLatency.value 转换为 Responsivity_Goal.rate.

3.2 转换驱动设施

通过 Rearon 本体, 可为环境驱动的自适应软件系统各方知识建立一个模型. 而要使用这些知识, 使其达到在应用目标制导下软件适应环境的目的, 就需要转换设施的支持. Rearon 是以 OWL/RDF 为描述语言的, 因此转换设施是基于 OWL/RDF 构建的一组推理引擎. 虽然描述逻辑的表达能力较强, 但是在具体领域、具体应用中, 还是需要定义一些附加规则来实现特定的处理. 因而转换驱动设施不仅包括标准的描述逻辑本体推理引擎, 它还可包括多种定制的规则驱动引擎. 基于描述逻辑的标准推理可用于本体中的冲突检测和本体的融合以及某些简单的目标驱动行为. 而对于具体领域和具体应用中复杂的场景, 则需要在基本的本体推理的基础上, 支持用户自定义规则的推理. 通过结合本体推理和规则推理, 可以完整表达用户在特定应用场景下的复杂的自适应策略.

如图 13 所示, Rearon本体模型及其本体实例均以RDF元组的形式存放在一个本体空间中, 在此空间上有一组驱动引擎观察着这些RDF信息. 每个引擎将可能产生新的RDF信息, 并在一个控制器的协调之下放入到本体空间之中. 这些引擎亦有可能要求控制器从空间中删除某些信息, 这实际上构成一种基于本体的黑板系统, 其一种基于Jena的实现方式可参考文献[45].

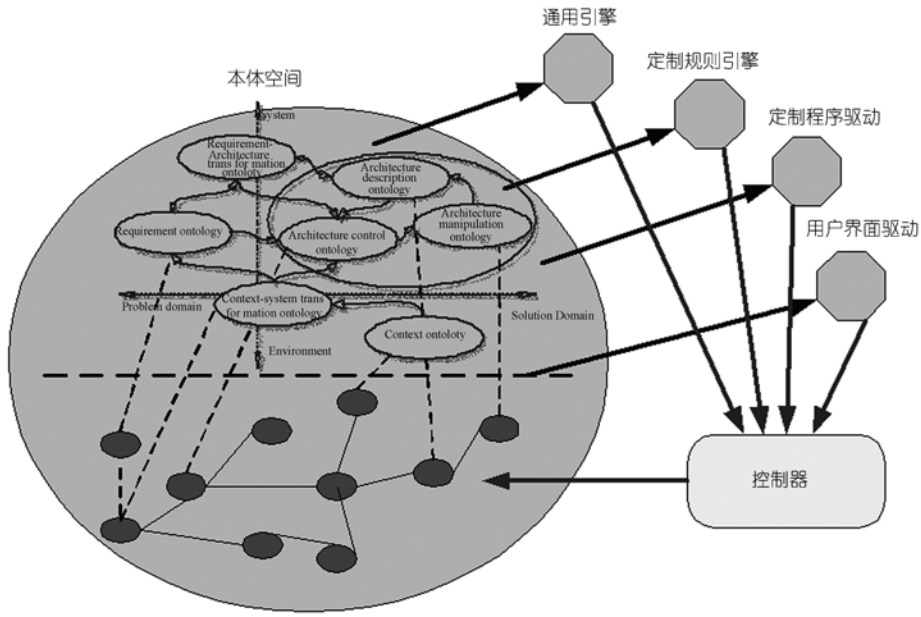


图 13 本体空间与驱动引擎

这些驱动引擎可分为 4 类: 一是通用OWL本体推理引擎, 如FaCT, RACER等. 二是自定义规则驱动引擎. 这些规则可不局限于描述逻辑甚至一阶逻辑, 而采用诸如Event-Condition-Action甚至某些特定的Term Rewriting技术, 以更直接有效地刻画环境适应行为, 表达某些复杂的、特别是操作性的处理需要. 第 3 种是通过DIG接口, 直接使用定制的程序来驱动演化行为, 主要用于满足某些无法自动推理或性能敏感的驱动需要. 它亦可用来作为系统与外部世界的接口, 支持系统本体的扩展. 最后一种是直接提供给用户手工干预本体空间及适应行为的界面. 在文献[42, 46]中我们定义了一种适应驱动规则语言, 并给出了相应的引擎.

通过Rearon本体和上述转换驱动实施, 我们在概念上以一种面向问题空间的方式沟通了环境上下文模型和协同系统演化行为; 技术上将基于本体的推理和基于定制规则的推理有机结合, 实现了应用目标制导下的环境驱动系统演化行为触发. 再通过我们在文献[6]所述的开放协同模型, 而这些体系结构演化命令可由内置运行时体系结构实现, 从而可以在线地得以实施. 这样, 我们就初步得到了一个可支持环境驱动的网构软件之构建的、具有一定一般性且易于扩展到系统的框架.

4 若干实验原型系统

可适应开放动态环境的应用的开发经历了数年的发展, 逐渐从无组织和随意的开发方式

(如Active Badge[47], Cooltown[48]), 到可复用的工具包的使用(如Context Toolkit[9]), 转向依赖于支撑应用开发的基础架构或中间件(如Solar[49], SOCAM[26]). 应用的支撑系统负责屏蔽硬件平台和操作系统的异构性, 为应用提供运行与开发的环境, 帮助用户灵活高效地开发和集成复杂的应用软件. 根据前文提出的环境驱动模型, 协同系统、环境模型和目标驱动是设计环境驱动应用支撑系统的 3 个关键要素, 为此, 针对这 3 方面的关键技术, 我们先后设计并实现了若干原型系统, 包括Artemis-ARC, Artemis-MAC, Artemis-FollowMeLite和Artemis-FollowMe-Agent等, 并在这些原型系统的基础上开发了环境驱动应用(见图 14), 这些原型系统及其应用为我们进一步研究环境驱动模型及其中间件提供了基础.

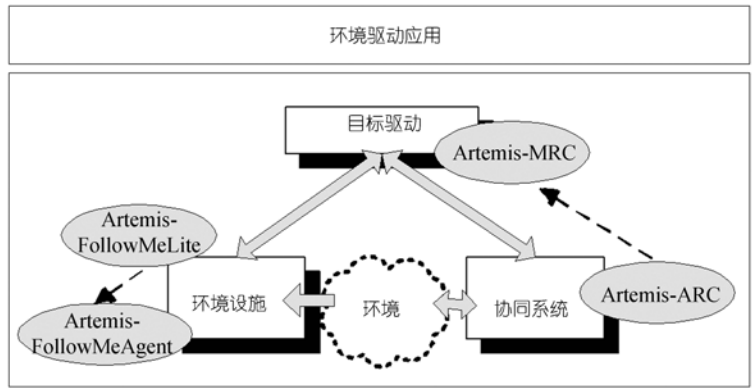


图 14　环境驱动模型的若干实验系统

需要说明的是, 开发这些原型系统的目的在于初步验证前文所述的若干新概念, 评估相关的新技术. 尽管这些系统乃是针对环境驱动网构软件的部分特性而设计的, 但它们并不是完整的支撑平台或完整的环境驱动网构软件应用. 当前网构软件作为一种技术发展趋势已经明确地呈现出来, 但完全意义上网构软件应用其真正实现尚有赖于两方面条件的成熟: 一方面, 网络上需存在有海量的可共享的自治软件服务资源; 另一方面, 以“足够正确”为目标的开放资源联盟型应用应能有可接受的大规模应用业务模式.

4.1　Artemis-ARC

协同机制是环境驱动模型支撑系统的核心, 是实现网构软件运行时刻根据环境变化进行自适应修改的基础. Artemis-ARC系统围绕这一问题, 设计并实现了一种以内置式运行时刻软件体系结构为核心的协同机制. Artemis-ARC中引入了内置的运行时体系结构对象来解耦系统中的各个服务构件, 并通过该对象以体系结构的视角重新解释服务部件之间的交互, 把体系结构这一抽象概念直接实现为具体的对象, 从而可以利用面向对象程序设计语言的继承和多态等整套机制, 导出一种面向体系结构的系统动态演化技术. 图 15 给出了运行时刻体系结构与软件体系结构规约和系统实现, 及外部环境之间的关系. 内置运行时刻体系结构在规约与实现之间建立起因果关联, 为系统的动态在线演化提供了一个面向动态软件体系结构的“反

射”架构, 通过对内置式体系结构的修改完成规约的更新和实现的演化[50].

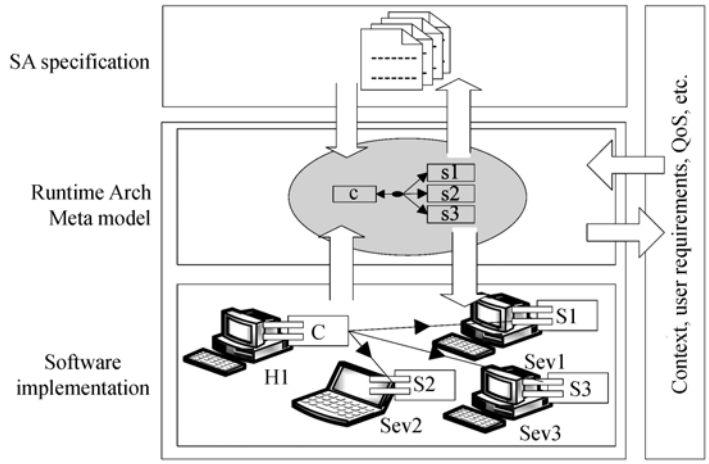


图 15 面向 DSA 的网构软件协同架构

面向动态软件体系结构的网构软件协同机制使得网构软件在线演化成为可能, 为进一步实现自适应, 及环境驱动的在线演化提供了支撑, 是后续系统开发工作的基础.

4.2 Artemis-MAC

作为 Artemis-ARC 的延续, Artemis-MAC 中重点实践了 1.2 节中提出的基于本体的软件自适应机制, 通过本体建模来认识和表达分布在问题空间和解空间中的与软件自适应相关的要素, 并采用推理规则来操纵决策要素, 从而实现了问题空间和解空间在软件自适应中的有机结合. 基于本体的自适应软件总体结构如图 16 所示.

图 16 基于本体的自适应软件总体结构图

整个系统由 5 个部分组成, 分别是: 运行系统层、体系结构层、自适应决策层、监控机制和公共服务. 底层支撑平台是通用的软件运行基础设施, 包括硬件、操作系统、网络设施等. 体系结构层复用了 Artemis-ARC 中提供的基础设施, 从系统全局的角度维护当前系统配置状态, 并对系统级属性的监控提供支持. 体系结构层由 2 个部分组成: 体系结构模型和体系结构管理器. 体系结构模型维护系统实时的体系结构信息, 包括系统的拓扑结构、构件的属性等. 体系

结构管理器就是负责根据自适应决策把对系统的动态调整表达在体系结构层面上, 从而间接达到调整运行系统结构和行为的目的. 体系结构管理器实现了基于本体的自适应软件的自适应执行引擎的功能, 此外它还为监控机制提供信息采集支持. 基于本体的自适应软件和其他基于体系结构的自适应软件的不同点在于, 决策要素的存储和使用从它的生成者和收集者中分离出来, 由系统中新增的自适应决策层来统一管理和操纵.

自适应决策层负责维护和管理自适应软件的决策要素和自适应逻辑, 并依据系统的运行状况给出合理的系统调整策略. 该层的结构如图 17 所示.

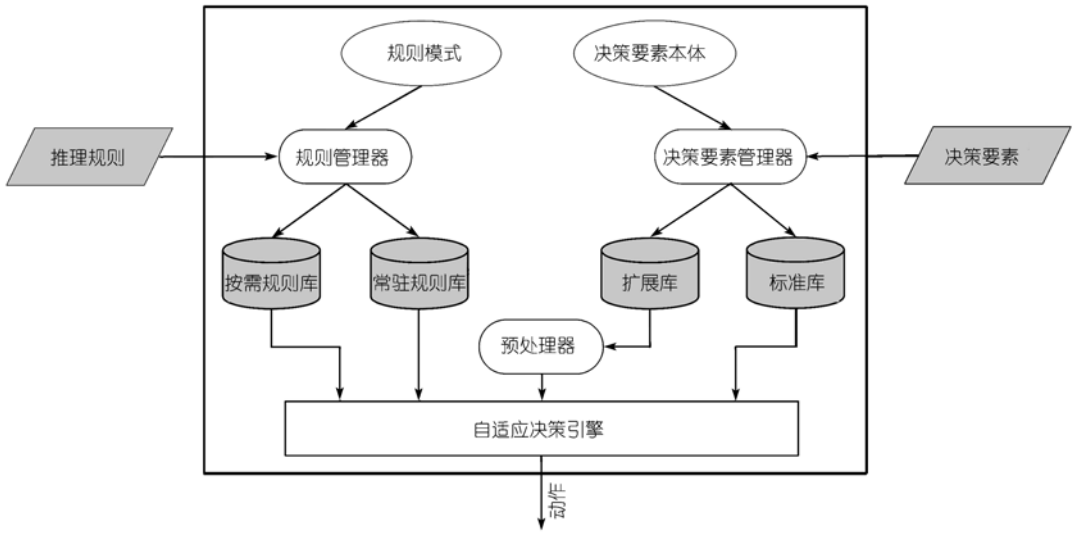


图 17　自适应决策层结构图

该层以决策要素和推理规则为输入, 系统调整动作为输出. 从图 17 可以看出, 自适应决策层由 3 个主要模块构成: 决策要素管理模块, 自适应逻辑管理模块和自适应决策模块.

● 决策要素管理模块维护整个系统的决策要素. 决策要素管理器接收来自监控机制的决策要素数据, 用决策要素本体规定的术语集描述决策要素, 并根据该决策要素的特点把它存入相应的数据库. 这里我们设计了 2 类决策要素数据库: 标准库和扩展库. 标准库存放采用标准 RDF 三元组表示的决策要素, 扩展库存放采用扩展的 RDF 元组表示的决策要素. 扩展库的存在主要是为了处理一些 RDF 三元组无法或者难以表示的决策要素, 增加系统的灵活性.

● 自适应逻辑管理模块维护自适应决策所需的推理规则. 规则管理器接收来自用户的需求定义, 同时采用规则模式对它进行校验. 规则模式用于保证输入的规则符合一定的语法. 我们把推理规则分为 2 类: 常驻规则和按需规则. 常驻规则参与每一次自适应决策; 按需规则只在与其相关的要素发生变化时参与自适应决策. 这样可以有效减小自适应决策过程中用到的规则集的规模, 从而提升自适应决策的性能.

● 自适应决策模块包括预处理器和自适应决策引擎. 为了保证自适应决策引擎的通用性和高效性, 它仅支持对标准 RDF 三元组的推理. 为了能够在自适应决策时使用扩展库中的数

据, 我们设计了预处理器. 预处理器根据用户指定的处理算法把扩展库中的数据处理成标准的 RDF 三元组, 然后和标准库中的数据一起交给自适应决策引擎作推理.

4.3 Artemis-FollowMeLite

通过轻量级运行平台Artemis-FollowMeLite, 我们试验了前述环境设施中的部分关键技术. 该平台以OSGi为底层平台来灵活地和安全地管理服务, 使用语义Web语言建立了一种Ontology上下文模型去支持上下文共享和推理, 选择RDQL[1)]作为上下文查询语言. 对于上下文应用的开发人员, 通过引入工作流模型而大大简化了应用的开发. 针对移动设备网络连接的多样性和不稳定性, Artemis-FollowMeLite使用代理连接服务技术去适应多种网络连接的通讯服务. 通过部署不同的组件模块, 它可以适用于多种应用环境.

Artemis-FollowMeLite 在语义 Web 语言的基础上建立一种形式化的上下文模型, 并设计了一种从低层上下文到高层上下文的融合机制. 动态上下文用五元组(subject, predicate, object, ttl, timestamp)来表示. Ontology 的实现采用 OWL, 持久化上下文通过 RDF 文件储存, 动态上下文表示成 RDF 消息加上时间信息. Ontology 以一种层次结构的方式建立, 从上到下分别是: 顶层 Ontology、特定领域 Ontology 和特定应用 Ontology.

Artemis-FollowMeLite 的主要目标是建立一种新型的上下文感知应用的开发方式, 使得在一个环境中的多个应用的开发成为一个整体而不再相互孤立, 使它们之间的信息共享和协商变得清晰和结构化、软件功能模块的复用最大化, 并减少应用开发者的工作量. 为此, 我们定义了专门描述上下文感知应用的工作流定义语言 esPDL(environment-sensitive process definition language), 设计并实现了能够解析 esPDL, 并支持多个上下文感知应用协同工作的工作流引擎 esWE (environment-sensitive workflow engine).

- esPDL

esPDL 用于描述上下文感知应用的事务逻辑. 每个应用被拆分成多个原子功能模块(processUnit), 如开门、拉上窗帘、读取温度信息等, 不同应用之间 processUnit 可以共享, 使得软件复用度大大提高. ProcessUnit 之间的执行顺序和数据交互由 esPDL 描述. esPDL 是一种以 XML 书写的语言, 所以原来的硬编码确定应用程序逻辑的方式变为由描述性文本定义. 编写一个新的上下文感知应用只需要编写所缺少的 processUnits 和一个 esPDL 文件. 修改一个应用的事务逻辑只需要修改相应的 esPDL 文件, 软件的易维护性大大提高.

esPDL是对XPDL的裁剪和改造. 我们去掉了XPDL中极少被使用的功能, 删去了过于复杂、易引起混淆的部分以及一些冗余的选择, 保留了定义事务过程所必需的元素. 经过裁减的esPDL仍然是一个全功能的工作流定义语言, 描述能力和XPDL等价, 但比XPDL简明清晰得多, 书写也更为简便. 为适应上下文感知的需要, 我们增加了esPDL对RDQL的支持. 在esPDL中上下文的表达以及查询均以RDQL描述. esPDL的严格语法定义参见文献[51]. 为进一步简化应用开发, 我们提供了上下文感知应用的图形化开发方式. 用户只需在FollowMeBuilder中拉拽图形元素组合成一张上下文感知应用的事务流程图, 系统将自动将其翻译为esPDL文本.

1) http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/

● esWE

esWE 解释执行由 esPDL 定义的上下文感知应用的事务流程. esWE 读取 esPDL 文件, 调配各个 processUnit 完成相应工作. 多个应用可在 esWE 上并行完成, 它们之间的交互亦由 esPDL 定义, 由 esWE 指挥.

esWE 支持事件机制. 由事件触发是上下文感知应用的一大特点. 例如智能接待系统由访客光临触发, 门禁系统由人员非法出入触发, 老人照看/报警系统由老人摔倒或者脉搏等生命体征异常触发等等. 本引擎直接支持时间机制, 大大简化了事务流程定义并且提高了应用执行的效率. 此外, esWE 还支持 RDQL 解析. esWE 通过解析 RDQL 而支持多个应用间的上下文通讯和共享, 成为上下文数据库、上下文推理机和应用间的逻辑桥梁.

● 可插拔编程模型

在 esWE 和 esPDL 的支持下, 开发上下文感知应用程序简化为开发这个应用独有的 processUnit 和编写一个描述其事务流程的 esPDL 文本. 在 Artemis-FollowMeLite 系统中, 每个 processUnit 都是 OSGi 平台上的一个可热插拔组件(bundle), 而 esPDL 文件(XML 文件)也是可以动态部署的. 所以, 整个应用开发过程成为一个可动态部署(插拔)的过程. 我们称之为可插拔编程模型. 在此模型下, 应用开发的复杂度大大降低, 可维护性和部署的灵活性大大提高.

4.4 Artemis-FollowMeAgent

我们在Artemis-FollowMeLite的基础上, 进一步研究了普适计算环境中用户计算任务的Follow-Me迁移. 所谓“应用的Follow-Me迁移”是指用户的计算任务能够随着用户在物理空间的移动在信息空间里发生相应的迁移, 与该任务执行相关的应用程序状态、属性、情境信息可以同时移动, 并且能够适应新场景下可用的计算资源, 用户可以在新的位置继续以他所喜爱的方式执行移动前的计算任务[52].

FollowMeAgent 系统基于 FollowMeLite 的面向服务基础架构, 复用了 FollowMeLite 中提供的若干基于 OSGi 的服务, 其体系结构如图 18 所示, 包含物理层、设备访问层、服务层、代理层和应用层. 物理层包括了各种各样的硬件设备, 包括用于通讯的蓝牙、GPRS 和 WLAN 设备, 用于感应和控制的传感器和激励器等设备, 比如用于定位和感光的传感器, 还有投影仪、打印机等设备, 它主要转化物理信号到计算机可读的形式并发向设备访问层. 设备访问层集成了物理层的多种设备, 以 OSGi bundle 的形式提供对外可访问的操作接口, 成为服务层中的服务. 设备访问层的目的是支持新设备的热插拔, 在需要时下载和安装设备驱动, 以及自动发现和附加已有设备到 FollowMeAgent 上.

代理(Agent)层是 FollowMeAgent 系统的核心层. Agent 容器以 bundle 形式运行在 OSGi 框架中, Agent 容器中运行着多个 Agent. 系统将为每一个用户分配一个可以定制的 User Agent, 它将负责响应用户的需求和感知用户的上下文, 捕获用户的当前任务, Follow-me Agent 是其中一种特殊的 User Agent, 它对用户的位置保持敏感, 并具有移动能力. Admin Agent 提供用户应用的管理服务, 包括生命周期的管理、目录服务、上下文管理服务等, 上下文管理服务是实现 Follow-Me 迁移的关键, 它需要通过对来自于物理环境的上下文信息和来自于用户的上下文信息进行融合、分类和推理等. System Service Agent 提供一些可插拔的服务或类库. 应用层

则可以通过代理层提供的服务和类库, 实现应用的迁移和重配置.

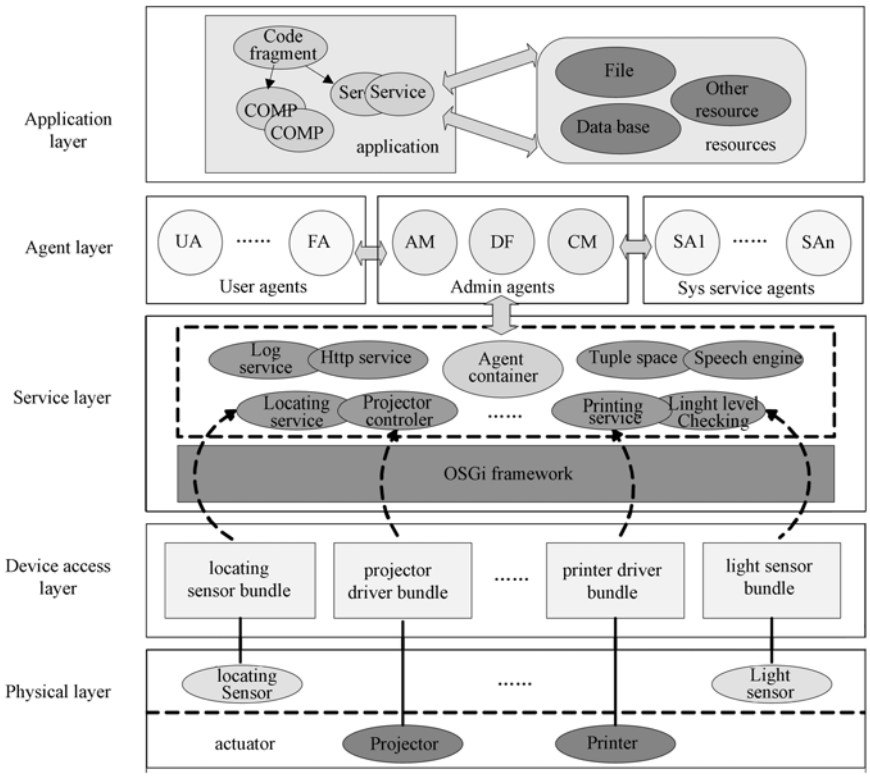


图 18 Artemis-FollowMeAgent 体系结构

4.5 环境驱动应用

我们在 Artemis-FollowMeLite 原型系统上实现了一个智能地图的应用, 它使用 RFID 技术定位环境中的物体, 并实时地动态构建在地图上, 当有物体移动非正常时(比如此物体被偷窃时), 应用会产生报警. 图 19 描述了系统的物理部署, 三个 RFID 读卡器分别放置在走廊和两个监控的房间里, 定时读取相应范围内 RFID 标签的信息. RFID 读卡器和 Context 提供者计算机相连, Context 提供者以特定的协议格式向 Context 服务器发送上下文信息. Context 服务器对环境中所有的上下文数据进行生命周期的管理, 并且可以根据 Ontology 和规则推理产生出高层的上下文数据. 当某一应用感兴趣的上下文数据产生时, 应用可以通过 RDQL 语句查询或者事件的方式得到具体的数据. 这里, 为了进行准确定位, 我们在监控房间内使用带有全向天线的 RFID 读卡器, 在走廊里使用带有定向天线的 RFID 读卡器, 应用设备选择的是 Dell Axim X51v, Artemis-FollowMeLite 平台和智能地图应用就运行于其上.

为便于把应用的逻辑和上下文信息结合, 我们使用 esPDL 文件描述智能地图应用. 应用的开始和结束由人工触发, 当 RFID 标签被检测到或者消失的时候, 应用根据相应的上下文信息刷新地图上的标签显示. Context 服务器上部署着 Ontology 信息和用户定义的规则集合. 系

统中所有的上下文信息会传递给 Context 服务器进行推理, 当推理器推理出物体移动非正常时, 应用会立即调用用于显示告警信息的过程单元. 图 20 展示了地图应用在 Dell Axim X51v 上的运行情形. 在时刻 1, IBM_T42 标签在房间 510, txp 标签在走廊上. 在时刻 2, txp 标签在房间 510 而 IBM_T42 标签在走廊上. 此时 Context 服务器根据 IBM_T42 标签属于 txp 标签, IBM_T42 标签的安全区域是房间 510 这两条规则得出在时刻 2 IBM_T42 标签产生了非正常移动. 工作流引擎得到 IBM_T42 标签非正常移动上下文后, 立即触发应用进行报警.

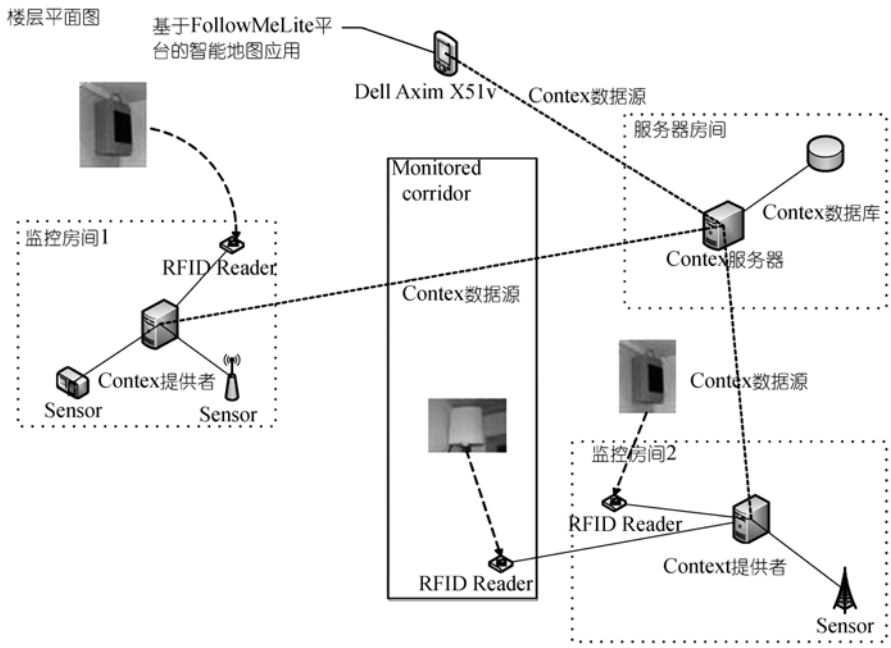
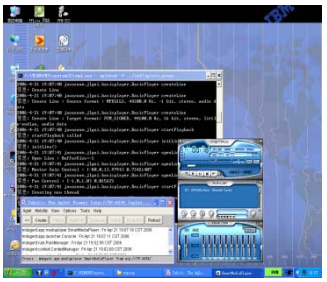


图 19　智能地图应用的物理部署

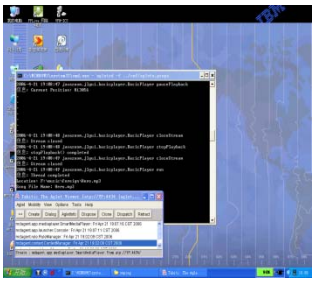
图 20　运行示例

此外, 我们还在 Artemis-FollowMeAgent 平台上开发了可以“Follow-Me 迁移”的应用程序

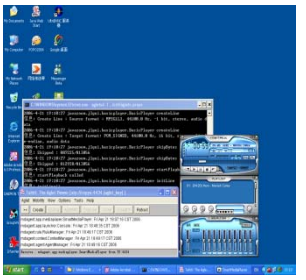
——文本编辑器和音乐播放器, 文本编辑器和音乐播放器可以根据一类特殊的上下文——用户位置的变化从 PC 迁移到 Laptop 和 PDA 或手机上. 见图 21 和 22.



运行在 laptop 上的播放器

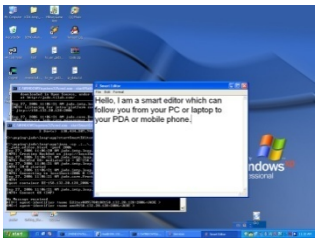


当用户离开时, 暂停播放器, 并且隐藏应用



当用户到达新环境时, 播放器迁移到用户新使用的 PC 上, 从之前暂停的点继续播放

图 21 音乐播放器迁移



运行在 PC 上的编辑器

用户要离开, 需要切换设备

编辑器迁移到手机上

图 22 文本编辑器迁移

通过这些原型系统的分别实践, 我们相信一个具有多重交互的环境驱动的软件模型是可行且有效的. 基于本体的上下文建模与处理技术可用以实现系统化、多层次、环境显式化处理, 基于内置运行时体系结构的动态协同技术可用以支持系统的动态调整, 而基于本体和规则推理技术的适应行为驱动技术可在一个较高的抽象层面上将前二者联系起来, 形成完整的具有环境适应能力的软件系统. 我们将在进一步的工作中将上述原型系统进行整合, 实现一个相对完整的面向网构软件的环境驱动软件支撑平台.

5 相关工作

如何适应开放、动态、难控的网络环境是越来越多的软件系统需要面对的挑战. 这方面有着大量的相关研究. 本节仅从面向网构软件的环境驱动模型所涉及的几个关键技术侧面讨论其中若干有代表性的工作.

在环境上下文的显式化描述与处理方面, 缺乏形式化的通用上下文模型与适用于动态环境的上下文处理技术成为一个日趋突出的问题. 这方面已有工作的主要目的是建立通用的上下文模型, 为开放环境驱动模型提供上下文处理支撑技术.

就环境上下文的显式化描述而言, 已有上下文建模及其处理技术总体来说呈现一种由特

定处理到系统建模, 由非形式化到形式化, 由静态信息到动态上下文的发展趋势. 在我们所见工作中, Context Toolkit[9]采用非形式化的上下文建模方式, 它将原始的上下文信息建模成使用XML描述的Key-Value对. Context Fabric[53]定义了一个专用的上下文描述语言来完成上下文信息的抽象. Ubi-UCAM[54]与Browsing the World[55]等将应用涉及的上下文信息用少量的词汇来简单地概括.另一方面, Gaia[56]采用一阶谓词逻辑进行上下文建模, 并用DAML+OIL来描述上下文. CoBrA[57]为智能空间环境建立了一个基于本体的上下文模型. 上述工作相对非形式化模型在可复用性方面有了很大的提升, 但是在动态上下文建模和处理方面主要依赖于应用逻辑. 总体而言, 上述工作在动态上下文建模和处理方面尚存在可改进之处. 而考察已有的上下文信息的融合、检索、隐私保护等关键技术, 可发现它们相对于环境驱动应用所需的系统化、高质量的环境信息服务需求仍有较明显不足. 现有上下文信息分布查询技术主要基于本体为上下文信息建立索引, 但是这一索引建立过程具有较大的随意性[29]. 而现有的隐私保护策略主要简单地将上下文信息设定为可访问, 或是不可访问两种模式[58]. 这一简单的隐私保护策略在大规模上下文信息应用中效率较为低下.

与上述已有工作相比较, 我们以系统建模为研究目标, 以本体作为形式化基础, 以动态上下文建模为切入点, 引入时间本体, 自行设计相应的动态本体, 构建了一个能同时有效进行静、动态本体工程的上下文模型, 并在动态上下文推理方面进行了尝试. 进而, 在相关关键技术方面, 我们从上下文的使用与查询两个方面度量上下文信息间的距离, 为上下文信息建立聚类, 提升了上下文分布式查询的效率; 基于上下文信息暴露度的动态模糊计算, 自适应、高效地实现了上下文信息的隐私保护. 这些工作一方面弥补了上下文研究领域中关于动态上下文研究的不足, 另一方面也通过建模、管理和应用多个环节, 支持了多层面的环境显式化处理, 为环境驱动系统的适应性行为提供了依据.

在软件系统与环境的交互方面, 缺乏适合直接刻画环境驱动的软件系统特点的软件模型和支撑技术成为一个日趋突出的问题. 这方面已有工作的主要思路是贯彻关注分离的原则, 使得环境上下文的处理与应用逻辑之间尽量解耦. 其中, 一部分已有工作试图在中间件层次上提供对上下文感知应用的支持[9,59~62]. 这样的中间件负责环境上下文的收集与处理, 尽量使得对上下文的处理对上层应用透明, 从而使软件开发者可以集中于应用逻辑自身的开发. 在中间件的层次上处理上下文感知使得我们可以开发用于上下文处理的通用模块(例如context widget[9]), 并且通过中间件支撑平台的重新配置即可实现这些模块的复用.

虽然在中间件的层次上处理上下文使得上层应用不需要关心上下文的收集、管理等工作, 但是如何利用上下文信息自适应地调整自身的行为仍然包含于应用逻辑自身. 基于这一紧耦合的上下文感知应用开发方法, 对上下文的考虑显著地增加了系统开发的难度. 因而Martinez和Lopes等学者研究在应用开发中更早的阶段和更高的层次上处理环境上下文. Martinez等[63]提出了一个概念模型, 将上下文感知处理为软件体系结构中一个新增的方面. 软件体系结构中的部件与连接子均可以显式地收集, 自动处理上下文信息. Lopes[64,65]将上下文感知显式地处理为软件体系结构中的第 1 类实体, 使得软件体系结构能够表示和组织系统涉及的上下文, 在设计系统的构件和连接子时利用上下文信息. 这使得对上下文的处理能够与体系结构中的

其他模块逐步组合、演化, 对上下文的处理与应用逻辑在软件体系结构的层次上实现了分离. 上述上下文处理方式可以与软件体系结构已有技术很好地结合.

另一类研究工作从编程模型的角度来研究如何支持应用与上下文的互动. 当开发者使用传统编程语言开发上下文感知应用时, 对上下文的处理往往横跨程序的多个功能模块, 紧耦合于应用自身的代码之中. Munnelly等[66]借鉴了面向方面编程的思想, 将上下文信息进行更为细致的划分, 并针对精细划分后的上下文信息编写粒度更细和行为更加可控的程序模块. 这一上下文处理方式在一定程度上将处理上下文的代码从应用自身代码中分离, 增强软件的可理解性、可维护性、可管理性. Keays[67]的工作则更进一步, 他基于代码填充的原理, 提出了面向上下文的编程. 面向上下文的编程将应用中与上下文无关的部分实现为一个程序骨架(skeleton), 而与上下文相关的部分则根据上下文信息的不同而分别实现为不同的程序段(stub). 随着上下文的变化, 面向上下文的编程机制会将相应的程序段插入到程序骨架中, 由此获得上下文感知的应用. 在面向上下文编程中, 上下文与面向对象程序中的“类”一样, 被处理为编程语言中的第 1 类构件.

与这些工作相比, 本文所提出的环境驱动模型将环境因素进一步显式化和独立化, 解除处理环境因素的软件部分和实现应用功能的软件部分的直接耦合, 再在高层应用目标和领域知识定义的框架下将它们连接起来, 从而便于二者的独立演化和复用, 以期更符合开放网络环境的需要.

6 小结

本文面向网构软件的需要, 在前期工作[6]的基础上, 讨论了环境驱动的软件系统的行为模式和技术需求, 提出了一个面向网构软件的环境驱动结构模型; 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, 并构建了若干实验原型系统. 当然, 相对于环境驱动的网构软件的总体目标, 这一工作仍较为初步, 需要进一步的研究. 在环境显式化方面, 我们期望首先将时态逻辑和环境上下文建模有机结合, 为动态上下文模型奠定坚实的语义基础. 其次, 针对环境交互的涌现模式和混合模式, 开展涌现上下文的识别和管理工作, 进而面向开放环境下的异构多环境, 研究其融合技术, 最终建立一个面向网构软件的环境模型. 在环境驱动软件结构模型方面, 需要进一步研究环境适应行为的各种具体行为模式, 以及这些模式抽象表达和具体化实现; 进而探讨适合开放网络环境需要的环境驱动软件语言、方法学及其支撑技术体系. 在环境驱动的应用方面, 我们正着手结合某些具体应用领域, 应用本文所述之模型与技术, 开展具有环境适应性的开放软件应用系统的开发. 最后, 依据我们在文献[1,6]中所讨论的网构软件模型之“开放协同模型 ⇒ 环境驱动模型 ⇒ 智能可信模型”的技术发展路线, 下一步重点的研究内容还将包括开放环境下软件实体和系统的可信性评估与保障, 以维持“满意度”或“足够正确性”为目标的智能决策等.

致谢 感谢南京大学计算机软件研究所徐峰、胡昊、宋巍和周宇对本文部分工作的贡献.

参考文献

1 吕建, 马晓星, 陶先平, 等. 网构软件的研究与进展. 中国科学, E 辑, 2006, 36(10): 1037—1080
2 Dahl O J, Nygaad K. SIMULA-an algol-based simulation language. Commun ACM, 1966, 9(9): 671—678
3 Goldberg A, Robson D. Smalltalk-80: The Language and Implementation. Reading: Addison Wesley, 1983
4 Meyer B. Object-Oriented Software Construction. New York: Prentice Hall, 1997
5 Booch G. Object-Oriented Analysis and Design with Applications. Reading: Addison-Wesley, 1994
6 吕建, 陶先平, 马晓星, 等. 基于 Agent 的网构软件模型研究. 中国科学, E 辑, 2005, 35(12): 1233—1253
7 Wooldridge M J. An Introduction to Multiagent Systems. Chichester: John Wiley & Sons, 2002
8 Abowd G D, Dey A K, Brown P J, et al. Towards a better understanding of context and context-awareness. In: Goos G, Hartmanis J, Leeuwen J, eds. Proceedings of the 1st International Symposium on Handheld and Ubiquitous Computing. Lecture Notes in Computer Science, vol 1707. Karlsruhe: Springer-Verlag, 1999. 304—307
9 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, 2001, 16: 97—166 [DOI]
10 Dourish P. What we talk about when we talk about context. Pers Ubiquit Comput, 2004, 8: 19—30 [DOI]
11 Gruber T R. A translation approach to portable ontology specifications. Knowl Acquis, 1993, 5: 199—220 [DOI]
12 Ma X, Zhou Y, Pan J, Yu P, 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
13 Medvidovic N, Taylor R N. A classification and comparison framework for software architecture description languages. IEEE Trans Softw Eng, 2000, 26(1): 70—93 [DOI]
14 Perry D E. An overview of the state of the art in software architecture. In: Proceedings of the 19th International Conference on Software engineering. Boston: ACM, 1997. 590—591
15 Lamsweerde A V. Goal-oriented requirements engineering: a guided tour. In: Proceedings of the 5th IEEE International Symposium on Requirements Engineering. Toronto: IEEE Computer Society, 2001. 249—262
16 Kalfoglou Y, Schorlemmer M. Ontology mapping: the state of the art. Knowl Eng Rev, 2003, 18(1): 1—31 [DOI]
17 Antoniou G, Harmelen F V. Web ontology language: OWL. In: Staab S, Studer R, eds. Handbook on Ontologies. Germany: Springer-Verlag, 2004. 67—92
18 Li J, Bu Y, Chen S, et al. FollowMe: on research of pluggable infrastructure for context-awareness. In: Proceedings of the 20th International Conference on Advanced Information Networking and Applications. Washington: IEEE Computer Society, 2006. 199—204
19 Bu Y, Chen S, Li J, et al. Context consistency management using ontology based model. In: Hopfner H, turker C, Konig-Ries B, eds. Current Trends in Database Technology. Lecture Notes in Computer Science, vol 4254. Berlin: Springer-Verlag, 2006. 741—755
20 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
21 Chen S, Gu T, Tao X P, et al. Application based distance measurement for context retrieval in ubiquitous computing. In: Proceedings of MobiQuitous 2007. Philadelphia: IEEE, 2007. 1—7
22 Lu W. Shadow: towards privacy protection in pervasive computing environment. Master Thesis. Nanjing: Institute of Computer Software, Nanjing University, 2007
23 Shi B, Tao X P, Lu J. Rewards-based negotiation for providing context information. In: Proceedings of MPAC 2006. Melbourne: ACM, 2006. 8
24 Strang T, Popien C. A context modeling survey. In: Proceedings of the 1st International Workshop on Advanced Context Modelling, Reasoning and Management. Nottingham, 2004. http://www.mobile.ifi.lmu.de/common/Literatur/MNMPub/Publikationen/stli04a/PDF-Version/stli04a.pdf
25 Borst W N. Construction of engineering ontologies for knowledge sharing and reuse. Ph D. Enschede: University of Twente, 1997

26 Gu T, Pung H K, Zhang D Q. Towards an OSGi-based infrastructure for context-aware applications in smart homes. IEEE Perv Comput, 2004, 3(4): 66—74 [DOI]

27 Rumbaugh J, Jacobson I, Booch G. The Unified Modeling Language Reference Manual. Redwood: Addison-Wesley Professional, 1998

28 Nejdl W, Wolpers M, Siberski W, et al. Super-peer-based routing and clustering strategies for RDF-based peer-to-peer networks. In: Proceedings of the 12th International Conference on World Wide Web. Budapest: ACM, 2003. 536—543

29 Gu T, Pung H K, Zhang D Q. A peer-to-peer overlay for context information search. In: Proceedings of the 14th IEEE International Conference on Computer Communications and Networks. San Diego: IEEE, 2005. 395—400

30 Heylighen F. Mining associative meanings from the web: from word disambiguation to the global brain. In: Poceedings of the International Colloquium: Trends in Special Language & Language Technology. Antwerpen: ACM, 2001. 15—44

31 Davida G I. Security and privacy. In: Proceedings of the 4th International Conference on Very Large Data Bases. West Berlin: VLDB Endowment, 1978. 54

32 Duckham M, Mason K, Stell J, et al. A formal approach to imperfection in geographic information. Comput Envir Urb Syst, 2001, 25(1): 89—103 [DOI]

33 Worboys M F, Clementini E. Integration of imperfect spatial information. J Vis Lang Comput, 2001, 12(1): 61—80 [DOI]

34 Worboys M F, Duckham M. GIS: A Computing Perspective. 2nd ed. CRC Press, 2004

35 Sweeney L. K-anonymity: a model for protecting privacy. Int J Uncertain Fuzz, 2002, 10(5): 557—570 [DOI]

36 Machanavajjhala A, Kifer D, Gehrke J, et al. L-diversity: privacy beyond k-anonymity. ACM Trans Knowl Discov Data, 2007, 1(1): 1—3 [DOI]

37 Li N, Li T, Venkatasubramanian S. T-Closeness: privacy beyond k-anonymity and l-diversity. In: Proceedings of IEEE 23rd International Conference on Data Engineering. Istanbul: IEEE, 2007. 106—115

38 Joelle C, James L C, Simon D, et al. Context is key. Commun ACM, 2005, 48(3): 49—53

39 Raiffa H. The Art and Science of Negotiation. Harvard University Press, 2006

40 Dardenne A, Lamsweerde A, Fickas S. Goal-directed requirements acquisition. Sci Comput Program, 1993, 20(1-2): 3—50 [DOI]

41 Castro J, Kramer J. From software requirements to architectures (STRAW01). SIGSOFT Softw Eng Notes, 2001, 26(6): 49—51 [DOI]

42 潘健. 一种基于本体的软件自适应机制的设计与实现. 硕士学位论文. 南京: 南京大学, 2007

43 Monroe R T, Kompanek A, Melton R, et al. Architectural styles, design patterns, and objects. IEEE Softw, 1997, 14: 43—52 [DOI]

44 Garlan D, Monroe R, Wile D. Acme: an architecture description interchange language. In: Johnson J H, ed. Proceedings of the 1997 Conference of the Centre for Advanced Studies on Collaborative Research. Toronto: IBM Press, 1997. 7

45 McKenzie C, Preece A, Gray P. Semantic web reasoning using a blackboard system. In: Alferes J J, Bailey J, May W, et al, eds. Principles and Practice of Semantic Web Reasoning. Berlin: Springer, 2006. 204—218

46 Zhou Y, Pan J, Ma X, et al. Applying ontology in architecture-based self-management applications. In: Proceedings of the 2007 ACM symposium on Applied computing. New York: ACM, 2007. 97—103

47 Want R, Hopper A, Falc V, et al. The active badge location system. ACM Trans Inf Syst, 1992, 10: 91—102 [DOI]

48 Kindberg T, Barton J. A web-based nomadic computing system. Comput Netw, 2001, 35: 443—456 [DOI]

49 Chen G. Solar: building a context fusion network for pervasive computing. Ph D Thesis. Dartmouth: Dartmouth College, 2004

50 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 [DOI]

51 Chen S, Bu Y, Li Y, et al. Toward context-awareness: a workflow embedded middleware. In: Ma J, Jin H, Yang L T, et al, eds. Ubiquitous Intelligence and Computing. Berlin: Springer, 2006. 766—775

52 Yu P, Cao J, Wen W, et al. Mobile agent enabled application mobility for pervasive computing. In: Ma J, Jin H, Yang L T, et al, eds. Ubiquitous Intelligence and Computing. Berlin: Springer, 2006. 648—657

53 Hong J I, Landay J A. An infrastructure approach to context-aware computing. Hum-Comput Interact, 2001, 16(2-4): 287—303 [DOI]

54 Jang S, Woo W. Ubi-UCAM: a unified context-aware application model. In: Blackburn P, Ghidini C, Turner R M, et al, eds. Modeling and Using Context. Berlin: Springer, 2003. 178—189

55 Castelli G, Rosi A, Mamei M, et al. A simple model and infrastructure for context-aware browsing of the world. In: Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications. New York: IEEE Computer Society Press, 2007. 229—238

56 Henricksen K, Indulska J, Rakotonirainy A. Modeling context information in pervasive computing systems. In: Mattern F, Naghshineh M. eds. Pervasive Computing. Berlin: Springer, 2002. 79—117

57 Chen H, Finin T, Joshi A, et al. Intelligent agents meet the semantic web in smart spaces. IEEE Internet Comput, 2004, 8: 69—79 [DOI]

58 Hong J I, Landay J A. An architecture for privacy-sensitive ubiquitous computing. In: Proceedings of the 2nd International Conference on Mobile Systems, Applications, and Services. New York: ACM Press, 2004. 177—189

59 Roman M, Hess C K, Cerqueira R, et al. Gaia: a middleware infrastructure for active spaces. IEEE Pervasive Comput. 2002, 1(4): 74—83 [DOI]

60 Bellavista P, Corradi A, Montanari R, et al. Context-aware middleware for resource management in the wireless internet. IEEE Trans Softw Eng, 2003, 29(12): 1086—1099 [DOI]

61 Gu T, Pung H K, Zhang D Q. A service-oriented middleware for building context-aware services. J Netw Comput Appl, 2005, 28(1): 1—18 [DOI]

62 Capra L, Emmerich W, Mascolo C. CARISMA: Context-aware reflective mIddleware system for mobile applications. IEEE Trans Softw Eng, 2003, 29(10): 929—945 [DOI]

63 Martinez J J, Salavert IR. A conceptual model for context-aware dynamic architectures. In: Proceedings of the 23rd International Conference on Distributed Computing Systems. Washington: IEEE Computer Society, 2003. 138

64 Lopes A, Fiadeiro J L. Algebraic semantics of design abstractions for context-awareness. In: Fiadeiro J L, Mosses P, Orejas F. eds. Recent Trends in Algebraic Development Techniques. Berlin: Springer, 2005. 79—93

65 Lopes A, Fiadeiro J L. Context-awareness in software architectures. In: Morrison R, Oquendo F, eds. Software Architecture. Berlin: Springer, 2005. 146—161

66 Munnelly J, Fritsch S, Clarke S. An aspect-oriented approach to the modularisation of context. In: Proceedings of the 5th IEEE International Conference on Pervasive Computing and Communications. Washington: IEEE Computer Society, 2007. 114—124

67 Keays R, Rakotonirainy A. Context-oriented programming. In: Proceedings of the 3rd ACM International Workshop on Data Engineering for Wireless and Mobile Access. New York: ACM Press, 2003. 9—16