基于动态代理的上下文感知编程模型 EIPM 研究

任蓓丽,齐 勇,李 明,张俊斌,牛玉洁,赵万贺
(西安交通大学电信学院软件研究所,陕西西安 710049)

摘 要: 上下文感知应用和普适计算环境逐渐渗入人们生活环境的今天,上下文感知应用的设计开发时的支撑环境方面仍主要采用传统的面向对象等技术及编程模型,这样在设计时与上下文相关的行为会分布在程序中,即环境上下文和行为在程序编码时就进行了绑定,这使得环境依赖因素和程序控制主体结合的过分紧密,不能适应普适计算环境多样、复杂、动态和多变等特点,更难于进行系统维护和扩展.本文以实现应用程序动态适应环境信息变化为目的,设计并实现了采用动态代理为底层实现机制,通过映射规则的建立将上下文信息和编程逻辑进行分离的上下文感知编程模型 EIPM,提供了相应的开发编译平台和执行容器在内的应用框架系统原型,实现一种适应普适计算环境上下文动态复杂多变特点的编程模型.以普适环境文件访问系统中上下文感知部分模块的开发为例,进行了 EIPM 应用框架原型系统功能的可行性测试,结果表明 EIPM 编程模型具备了对环境上下文变化的动态适应性.

关键词: 普适计算环境; 上下文感知; 编程模型; 动态代理机制
中图分类号: TP311 **文献标识码:** A **文章编号:** 0372-2112 (2010) 2A-010-08

Research of Dynamic Proxy Based Context-Aware Programming Model EIPM

REN Bei-li, QI Yong, LI Ming, ZHANG Jun-bin, NIU Yu-jie, ZHAO Wan-he
(*Dept. of Computer Science and Technology, Xi'an Jiaotong University, Xi'an, Shaanxi* 710049 *China*)

Abstract: As ubiquitous computing pattern has been infiltrating people's daily life nowadays, the context-aware application's development process is still adopting the traditional object-oriented programming model and technique, which makes the context-related behavior design distributed in the application programs, and the binding of environment context and behavior occurred at coding phase. This will lead to the excessive tightly combination of the environment dependence factor and programming control logic, also hard for the system expanding and maintenance. According to the situation, this thesis aiming at the dynamic adaptation of environment context changing, designed and implemented a context-aware programming model called EIPM which implemented dynamic proxy as the base mechanism, separated thecontext information and programming logic by establishing the mapping rule. Also, it provided the corresponding development platform and execution container as the implementation framework prototype system to support work under the programming model. Use Pervasive Document Access System as application background, test and illustrate the feasibility of EIPM application framework, and the results indicate that EIPM programming model has the dynamic adaptability to the changing environment context.

Key words: ubiquitous computing environment; context-aware; programming model; dynamic proxy mechanism

1 引言

普适计算[1](Ubiquitous/Pervasive Computing)思想最早由 Mark Weiser 在上世纪 90 年代初提出.近年来随着互联网已经进入普适网络阶段,基于普适网络的系统平台及其关键技术已经成为重要的研究方向.普适计算要求设备和计算机能够感知用户所处的上下文及其变化,并做出相应的动作,即上下文感知(Context-Awareness)技术.而上下文感知应用程序则指的是那些由用户当前上下文来控制执行动作的软件系统,通过硬件和软件的传感器来察觉到用户的当前环境状况,然后根据一些提前定义好的规则或规范来自动的决定它的行为.其特点在于它们使得计算设备更智能,对用户更贴心,因此将成为未来的关键技术.

收稿日期:2009-05-22;修回日期:2009-07-10
基金项目:国家 863 高技术研究发展计划(No.2006AA012101);国家自然科学基金重点项目(No.60933003);陕西省国际合作项目(No.2008kw-02);IBM 联合项目(No.JLP200906008-1);高效能服务器和存储技术国家重点实验室开放基金项目

目前对于普适计算环境下的上下文感知应用设计开发时支撑环境方面主要采用传统的面向对象等技术及编程方法.由于传统编程模型需要在设计时确定参数个数及类型,环境依赖因素和程序控制主体结合过分紧密,通常将上下文和行为在程序代码中就进行了绑定.但这种开发时绑定往往会在上下文感知应用程序中为下面的一些情况带来困难:(*a*)当新的上下文被引入;(*b*)当上下文和行为的映射规则需要改变;(*c*)增加新的行为.也就是说,一旦环境发生变化,就需要修改程序并重新进行编译、部署等工作,不能适应普适计算环境多样、复杂、动态和多变等特点,更难于进行系统维护和扩展.因此研究针对普适计算环境特点的上下文感知编程模型及其支撑环境十分必要并有重要实际意义.

近年来在面向上下文的编程模型的研究有了新的进展,其中有代表性的包括下面几类.Andry Rakotonirainy 提出的面向上下文编程模型 COP(Context Oriented Programming)[2],通过将程序分为主体框架和与上下文相关的 open terms,根据上下文或 open terms 的描述来选择合适的 stubs 进行填充,实现了上下文和应用逻辑的解耦,但其使用脚本语言作为基础实现,而脚本语言本身能否担负大型工程项目的开发始终被业界怀疑,因此较难适用于复杂的应用;而 Anca Rarau 等人提出的 Aware C# [3]在语言层面上采用多面体概念扩展 C# 语言模型,来将上下文感知功能插入到 C# 中去,但仅仅在语言层面来进行扩展使得程序员在编程时将上下文及各种变化也定义在程序中,对环境变化的适应能力较差;Anand Ranganathan 等人提出的 Olympus[4]模型利用本体来从抽象层定义规则与实体,并通过框架 Gaia 完成虚拟实体到真实空间的映射,程序员需要使用脚本语言和本体来定义规则与实体,因此对环境的适应能力较差.

本文以实现运行时动态适应环境信息变化为目的,设计了采用动态代理为底层实现机制,通过建立上下文空间与应用逻辑行为的映射规则,将上下文信息和编程逻辑进行分离的上下文感知编程模型 EIPM(Expanded Isotope Programming Model).其设计思想是在 Isotope 编程模型(IPM)核心概念的基础上,通过进一步的完善以及实现机制的设计来扩展 IPM 模型.同时为 EIPM 编程模型建立了包括开发、编译以及执行容器在内的应用框架原型系统,来提供一种适应普适计算环境多样、复杂、动态和多变特点的编程模型,减少上下文感知应用程序开发人员开发和维护的工作量和复杂程度.

2　Isotope 编程模型简介

Isotope 编程模型(IPM)[5]的核心思想在于将对象的代码从一个类文件扩展到多个类文件,每个文件表示一种环境对应的执行过程,多个类文件共存于系统中,共同提供服务.这种对象模型类似于化学元素中的同位素:占据相同的周期表位置,具有类似的功能,但具有不同的形式,因此我们称这种模型为 Isotope(同位素)编程模型,简称 IPM.在 IPM 模型对象中除了传统对象的属性声明,还包含一组同位素元素(isotope elements).每个同位素元素代表着对象在不同环境下的执行动作.对象在表示其处于不同环境中的行为时,不需要将所有行为混合在一个对象的声明中,取而代之的是若干的独立的同位素元素,因此对象的声明过程也简化为只包含属性的声明.图 1 是 IPM 模型对于面向对象模型中对象的扩展结构.

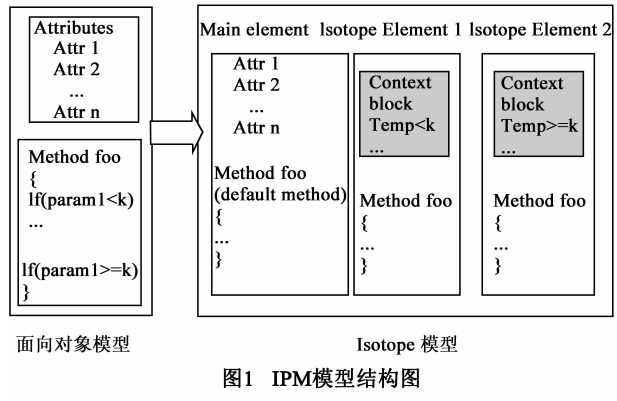

图1　IPM模型结构图

在 IPM 模型结构中,除了对象主体(Main Elements)和与之分离的行为描述部分(isotope element)外,每个 isotope 还各自具有一个上下文声明块(context block),每个 context block 由一组上下文信息条件表达式组成,作为 isotope-element 生效的条件.

IPM 模型的支撑系统框架为 IPM 模型中的对象提供运行时容器来支持基于 IPM 模型的程序运行.运行时容器负责加载和管理对象所有的 Isotope Element,并管理 Context Block 及其行为的对应关系,接受系统对指定对象的调用,并选择合适的 Isotope Element 来完成方法调用.

3　EIPM 编程模型设计关键问题及解决方案

EIPM(Extended Isotope Programming Model)是在 Isotope 编程模型理念的基础上,以基于动态代理机制进行实现为目的的进一步扩充.IPM 作为对传统编程模型的思想扩展,提出了同位素对象的概念,在一定程度上解决了程序对于环境的适应性问题.但是面对较为复杂的应用环境,IPM 中采用环境声明 Context Block 的上下文感知实现机制将很难灵活适应,并且只关注对象方法的分离是不够的.因此要将 IPM 的编程思想体现到应用上,还需要从核心概念和外围支持系统两方面来分别对 IPM 进行扩展.EIPM 编程模型建立在 IPM 的基

础上,对于上下文从空间的角度进行维度划分,设计了多面体和多维上下文空间的映射机制;同时扩展了对象属性环境依赖的分离,将和上下文相关的对象属性值获取通过上下文的本体建模来进行分离;最后,对于IPM 支撑系统进行更详细的设计和改进.

3.1 映射机制和选择策略

上下文感知应用对于传统的面向对象编程模型带来了挑战,即在程序中将上下文和行为进行绑定通常会在下面的情况中造成麻烦:(a)新的上下文被引入到应用中;(b)上下文和行为的绑定需要改变;(c)加入新的行为.因此 EIPM 采用把上下文-行为绑定从程序中分离出来,通过映射规则文件的编写来实现.

在普适计算中的应用环境,每个上下文感知的应用都有自己的上下文空间,其包含了程序依赖的所有上下文.上下文空间往往是多维的,Weichang Du 等人[6]定义上下文空间为各个上下文维度的笛卡尔乘积:

$$ContextSpace = \prod_{i=0}^{n} \overrightarrow{\dim_i} \quad (1)$$

其中,ContextSpace 代表上下文空间,而 $\dim_i$ 则代表空间中的第 i 个维度.每个上下文描述对应于多维上下文空间中的一个点,在 N 维上下文空间中用 N 元组($d1$, $d2$,… ,dn)的形式来表述.在维度 d 的坐标用 $d[i]$的形式表示,例如($time[2]$, $sound[0]$)就表示了某个两维上下文空间中的一个上下文点的值.

在 EIPM 中,将同位素 Isotope Element 和上下文进行相互映射.因为模型的上下文空间定义域是程序所可能遇到的所有上下文的值,因此从 Isotope Element 到上下文空间的映射是一种满射,即从每一种在程序定义域内的上下文都有对应的行为.本文通过"上下文—行为"绑定规则文件来定义上下文和行为的关联.只有在当前上下文在一个特定的上下文集合(整个上下文空间的一个子集)中时,应用程序的某个行为才能被激活.下面用二维上下文空间来进行说明,如图 2 所示.

上图 2 中的二维上下文空间具有两组上下文集合:包括区域 A 和区域 B 的集合 1;包括区域 B, C, D 的集合 2.这里,上下文集合 1 对应的是类 X 的应用空间,集合 2 和类 Y 的应用空间绑定.在每个类对应的集合中,不同的上下文区域对也应着不同的 Isotope 方法:类 X 在上下文区域 A 中会采用 X.Isotope Element 1,而在区域 B 就会换成 X.Isotope Element 2.类似的类 Y 在自己所对应的上下文集合中,也会根据上下文区域的变化而采用不同的 Isotope 方法.在上下文空间中,集合 1 和集合 2 允许拥有重叠,这个交集部分在应用类不同的时候会激活不同的 Isotope Element 行为.在对上下文和行为绑定的时候,分为两个阶段的映射:类 Class 到上下文集合 Context Set 的映射和上下文元组到行为方法的绑定.不同的上下文区域允许绑定到同一个 Isotope 方法,但反过来则不允许,即禁止一个上下文元组对应同一个 Class 中的多个 Isotope 方法.

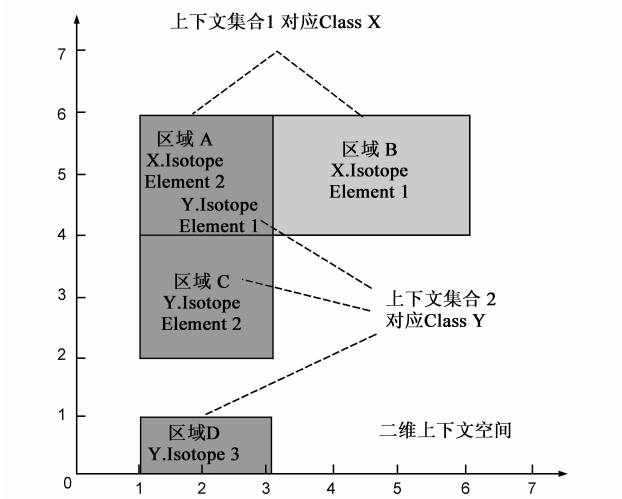

图2 二维上下文空间中的上下文-行为映射示例

EIPM 模型在进行同位素方法选择时,摒弃了 Isotope 模型中采用 Context Block 的固定且难以实现的方式,而采用基于动态代理的框架组件和定义匹配规则文件来实现.应用框架系统将应用程序、执行容器和映射规则整合到一起成为一个完整的上下文感知应用.这样,程序员的注意力将被放在应用逻辑的实现上,而不是程序和环境上下文的具体绑定.而在运行时刻,应用框架中的执行容器读取映射规则配置文件,在程序初始化时创建多维上下文空间.当程序运行过程中,需要调用属于上下文相关对象实例的方法时,执行容器通过从当前应用环境获取上下文数据,若上下文和上次调用时有所变化,则通过查询映射规则,获得匹配的上下文信息,进行同位素方法的选择.在进行匹配选择过程中,需要定义的映射规则包括:(1)应用上下文规则:定义一个应用所涉及到的上下文空间,包括组成空间的各维度信息,并通过定义每个维度的属性信息来详细设计上下文空间的划分;(2)上下文-行为映射规则:将环境上下文和具体的 Isotope 行为进行绑定的配置信息,由专门角色(领域专家或用户)来指定和更改,具有很大的灵活性.在对映射规则的定义中,包括对上下文集合(由 Context_Set 标签表示)所对应的类的信息的指定,上下文区域的信息(Regions 标签)与上下文-Isotope Element 对象的映射信息(Region_ Isotope_ Binding 标签)表示.通过相关角色对规则的定义和设置,来实现在不改变程序代码的基础上,根据环境的变化对应用程序进行动态配置.

3.2 对象属性的上下文依赖分离

我们需要注意的另一个事实是,对象不但会根据不同的上下文信息来有不同的行为,同时也会呈现出

不同的状态(不同的属性值的集合).以智能办公室的多功能打印机为例,当打印机通过用户偏好来转换打印模式时,该用户信息作为上下文就影响了打印机的状态,此时的对象属性中的一部分就发生了变化,通过用户及其相关信息进行了更新或增减.类似的场景在实际应用中比比皆是,因此我们不能忽略对象属性和上下文变化的关系.而对程序员来说,关注这些上下文敏感的属性是一项复杂而耗费时间的任务.因此在EIPM编程模型中,上下文相关的对象属性值的获取被从程序逻辑中分离出来.这首先需要进行上下文的建模.T. Strang 等人在文献[7]中总结了几种上下文模型,并基于六条标准(分布式组成、部分验证、丰富性和高质量、任意性、分级形式化以及可用性)对各种上下文建模方法进行了评估和总结,最后评估结果表明本体(Ontology)建模是最适合于表示上下文信息的模型.本体是一种共享概念模型的明确的形式化规范说明,用于描述概念以及概念间的关系.目前应用广泛的本体描述语言是 W3C 的 Web Ontology Language(OWL).因此,EIPM 模型的属性值分离机制通过采用将实体和它们的上下文抽象为本体来实现,即模型化应用环境中的上下文相关实体,然后通过映射模块来为程序主体动态获取当前环境下上下文相关对象的属性值.这样通过映射关系,对象获得了属性的上下文信息,并为程序员隐藏了底层上下文获取的细节.本体的实体属性更新通过上下文感知系统的上下文获取机制模块来实现,一种方式是将静态数据和动态数据用 OWL 文件形式存储,并随传感器及解释器的结果而更新;另一种方式将实体数据在数据库中存储,通过本体和数据库的映射

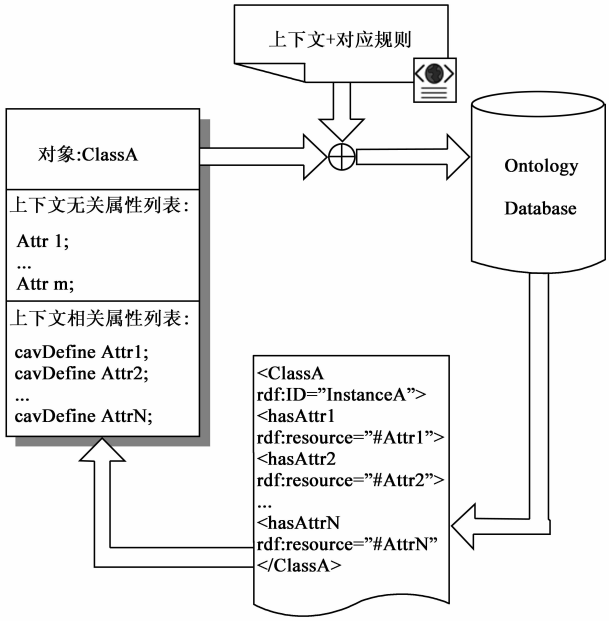

图3　上下文相关属性值的获取

来完成查找.属性值获取原理如图 3 所示.

类的成员变量按照与上下文的关系分为两类:上下文无关属性变量和上下文相关属性变量.其中上下文相关属性变量在运行时其值将由 OWL 本体库中所对应对象的属性来提供,通过上下文信息匹配,获取本体库中相应对象的属性,并提供给程序.这样通过将对象与应用环境空间实体进行映射,来获取并为对象指定与上下文相符合的上下文相关属性的值.

EIPM 模型的执行容器将从本体层次和实体描述来查询发现对应的实体.在为 Isotope 对象选择对应的实体时,需要满足下面几种约束:(1)开发人员在程序中指定的变量约束条件;(2)本体中列出的约束条件;(3)系统管理员为应用场景空间指定的约束条件;(4)系统使用者的用户策略.属性匹配的实体发现过程的执行步骤如下:

(*a*)发现与实体相匹配的类:应用框架查询本体库,取得和主体对象类相同或者最接近的类的列表,若返回是类列表,则按语义相似度排列.

(*b*)类级约束条件过滤:应用框架将本体库返回的列表中的类进行过滤,过滤掉那些不符合约束条件的类.这些类级别限制条件通过策略文件或者本体中的指定来提供.

(*c*)在应用领域本体中查询对象实例:对列表中的类,通过有过滤条件的查询来发现类的对象集合中对应的实例个体.

(*d*)实例级约束检查:对从本体库中返回的所有实例,系统检查它是否满足在程序逻辑中和策略文件中指定的实例级约束条件.得到最终匹配列表,其中实例的都满足匹配要求.

(*e*)在最终列表里选择最佳实例:系统基于多维标准匹配函数来决定最优的实例个体,从而将其属性值映射到主体对象中.

3.3　语言层扩充

为了和 EIPM 的实现机制无缝的衔接,编程模型还根据需要对语言层扩展,引入新的语言成分,使应用程序开发人员可以将 EIPM 模型应用在他所熟悉的面向对象语言编程上.本文以 Java 面向对象语言为例来进行扩展.扩展后的 Isotope 语言是一种中间语言,可以根据不同转换策略将其转化为现有流行的高级面向对象语言.

IContext(上下文描述):IContext 作为一个复合数据类型来定义上下文,可表示为单个的三元组即(Subject, Predicate, Object),或者是若干三元组的列表.程序将 IContext 变量来作为程序员指定的约束条件来过滤获取上下文感知元素(属性值和 Isotope Element).

IClass(Isotope 类定义):IClass 声明用来定义上下文相关类.声明格式和 class 类似:IClass NameOfClass[ex-

tends Superclass].在类体的定义中,包括了普通属性,上下文相关属性变量,以及方法的声明.其中方法体可以作为缺省方法实现也可以为空.在创建类的对象的声明部分可根据需要还指定该对象关联的上下文内容.例如 PDA 的一个对象 PDA_1 是一个 PDA,他属于用户 ID 为 takayama 的用户,且当前时刻正在访问文件服务器 Serv1.则定义如下:

```
lclass PDA extends Device
{
//Classbody……
}
PDAPDA_1 = new PDA(para_1,para_2…):(UseriD is takayama),
(AccessTargetisServ1);
```

或者使用关键词 IContext 来定义:

```
Icontext pdaContext = {(UseriD is takayama),(AccessTarget is Serv1)};
PDAPDA_1 = new PDA (para_1,para_2);pdacontext
```

cavDefine(变量声明):用在对类进行属性定义时,声明其中的上下文相关属性变量(Context-Aware Attributes).其形式为:cavDefine variableType variableName.上下文感知变量定义时只有声明,而不能进行初始化,赋值需要在运行时动态查找本体库得到.

element(同位素声明):使用关键字 element 声明 IClass 类对象的同位素对象的时候,除了主体对象的名称外,还要指定 element 的名称来区别各个 element,以便在编译后和上下文进行绑定.其定义形式为:element Class_Name Element_Name;element 也可以在所属的类定义文件中直接声明,定义形式为 element Element_Name;

这些扩展的语言成分提供了与 EIPM 模型系统的无缝衔接,使系统更好的领会程序员的意图,从而支持了 EIPM 执行容器的工作.采用新语言成分的程序编写的代码都被保存为.ist 文件,需要经过开发环境提供的编译器转换成目标语言文件才能运行.

3.4 执行容器框架设计

显然,对于对象的分离机制(包括上下文相关属性和方法)改变了 EIPM 编程模型的运行模式,传统编程的运行机制要被改造.本文为 EIPM 设计了基于动态代理实现机制的执行容器,作为整个 EIPM 系统框架的运行核心部分来支持新编程模型下的程序运行.执行容器框架的设计除了要考虑 EIPM 的执行流程外,还需要为管理员提供管理接口,以便按照需求调整应用程序,适应上下文环境的变化.

在 EIPM 模型的执行系统中,参与者被分为下面几种角色:(1)程序开发人员:编写程序的应用逻辑,对程序中上下文相关约束进行指定;(2)领域专家:为 Isotope Element 的定义提供支持,设计上下文感知规则文件,并负责应用环境的上下文本体建模;(3)系统管理员:负责管理整个执行容器系统框架,包括维护和更新本体库、Isotope Element 库、以及规则配置文件和约束;(4)用户:用户偏好的来源.事实上,这些角色只是从系统的角度来进行分类的,而现实世界中他们是可以重叠的.

EIPM 编程模型的执行过程从逻辑顺序上可分为下面几个过程,这里我们假定上下文本体库,Isotope 库以及映射配置文件都已经被各个角色定义好.

(1)程序员使用 EIPM 扩展语言进行应用 Isotope 程序的编写;(2)预编译模块对 Isotope 程序进行处理,将其转化为 Java 语言程序.Isotope Element 部分被预编译为 Element 对象,同时动态代理相关代码也将插入 Java 文件中;(3)对于当环境改变时新加入的 Isotope 对象,由系统管理员对映射规则进行更新修改,来将 Element 对象加入到运行系统中;(4)当上下文相关类需要被实例化,通过按实体匹配策略查询上下文本体库来确定目标实体,从中得到上下文相关属性的值,发送给对象;(5)在调用 IClass 的对象方法的时候,系统根据从上下文组件中获取的当前上下文,和缓存上下文进行比较,若不同则根据配置文件中的多维上下文空间匹配规则选择对应的 Isotope 对象,并调用其对应的方法.若相同,则按照上次调用记录使用对应的 Element 对象.若无返回对象则执行主体对象中的缺省方法.

根据前面对 EIPM 模型的运行机制和流程,设计了执行容器体系结构,如图 4 所示.

执行容器由管理员控制台,Isotope Element 管理组

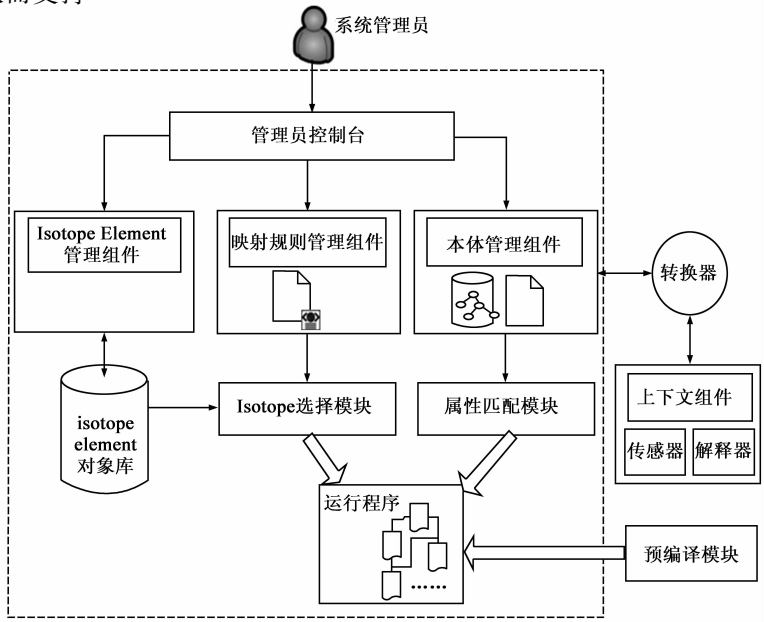

图4 执行容器结构图

件,映射规则管理组件,本体管理组件,Isotope 选择模块,属性匹配模块以及对象访问控制模块 7 个部分组成.系统管理员通过管理控制台来发出指令控制各组件完成任务,管理数据库和规则文件,监视当前的对象信息.管理员有权限查看目前加载的 Isotope 对象信息,卸载或替换已有的 Isotope Element,更改映射规则配置文件,更新当前上下文本体库.而 Isotope Element 管理组件,映射规则管理组件,和本体管理组件用来辅助系统管理员对于上下文相关成分的动态管理.所有的上下文相关成分都由它们直接管理.其中本体管理组件通过读取来自外围模块上下文组件中得到上下文内容,生成本体信息,写入到本体库中.Isotope 选择模块和属性匹配模块在运行过程中通过配置信息和匹配规则,和运行程序交互,应用在为运行程序实体选择 Element 对象和匹配上下文相关对象属性值,完成程序对运行中上下文相关成分的定义和更改.运行程序的执行部分采用动态代理的实现机制,对应用程序中 Isotope 对象方法的调用进行拦截,通过匹配映射策略得到对应的 Isotope Element 方法,返回给执行中的应用程序,从而实现运行时动态感知并适应环境上下文的控制.

4　应用框架

EIPM 模型的应用框架原型系统是用来支撑编程模型在上下文感知应用程序设计和运行过程的.应用框架系统设计目的是,为了给开发人员提供上下文感知编程环境,以及运行时动态感知环境上下文的对象访问控制功能.应用框架共包括三个层次,分别是开发环境,编译环境和运行时环境.系统概要设计如图 5 所示.

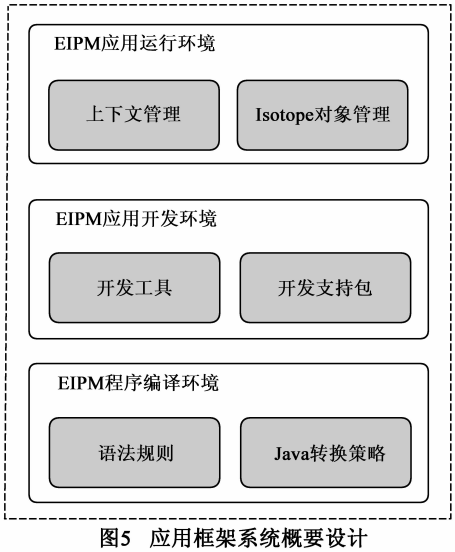

图5　应用框架系统概要设计

EIPM 编程模型以面向对象编程模型为基础,因此目标语言可以是各种面向对象编程语言,本文以 Java 语言为目标语言,因此在下层的编译环境中包含了语法规则的定义和 Java 转换策略来对 EIPM 中的语言扩展成分进行分析转换,其步骤为:(1)Isotope 类和上下文提供类的源程序编写在后缀名为.ist 的文本文件中;(2)编译器将 ist 源程序转化为 Isotope 模型;(3)根据目标语言转换策略将 Isotope 模型转化为后缀名为.java 的源文件;(4)最后与其他普通 Java 类一起编译链接为.class 字节码文件.

中间层的应用开发环境的目标是为编程人员提供上下文感知应用程序开发平台及开发支持包.根据 EIPM 的设计,开发环境包括 Isotope 工程文件生成向导,Isotope 代码编辑器,以及对 ist 文件进行预编译的编译模块.

最上层的应用运行环境根据环境上下文变化来动态选择 Isotope 对象运行,为对象提供匹配的属性值集,以及为管理员提供对应用环境上下文的管理接口.其中主要的运行时支持是通过匹配策略组件和对象访问控制模块的配合实现.在对象访问控制模块中,需要实现的最主要的功能是利用匹配策略组件获得的 Element 信息,动态调用匹配的 Element 对象方法.本文的设计中采用代理的方式,利用拦截器和 Java 反射机制来进行实现.提供一个代理机构,通过在调用主体对象方法的时候,对方法进行拦截,并转而匹配得到的 Element 对象进行方法调用,完成所需要的功能.实现机制的模型包括三部分:执行逻辑,拦截器和代理机构.因此,为了将上面的实现机制和程序紧密的结合,编译器在分析 Isotope 文件代码并转换为 Java 代码的时候做的转换工作就包括了,在 IClass 关键字标注的上下文相关类的 Java 代码中插入对于代理的调用机制.Isotope 程序到 Java 代码中相关的对应转换如下图 6 所示:

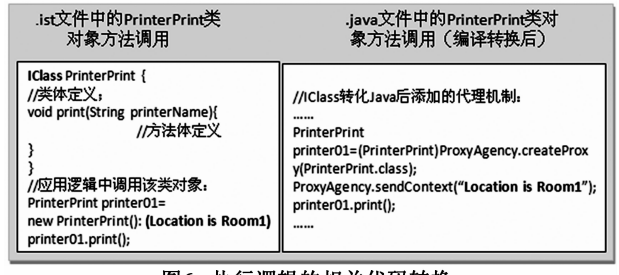

图6　执行逻辑的相关代码转换

先通过定义拦截器并生成提供获取拦截器的接口;然后通过参数得到代理对象,并获取类信息,将其传递给上下文空间-Element 匹配机制组件,获得匹配的 element 对象名;最后代理机制将相应的 element 对象的相应方法传递给执行逻辑,从而实现了对象访问控制.

5　模型应用及分析

5.1　应用背景

为各种电子办公设备都接入了网络的现代化办公

环境设计一种普适文件的访问系统,其中的关键问题之一是在越来越复杂的办公环境下,不同的用户身份和文件访问场合对于操作的要求都不相同.不仅在程序员设计系统时考虑和规定范围内的上下文变化时需要有不同的执行动作,还会随着公司的运行而引进新的上下文和执行动作.在一个工作办公环境范围内,我们只考虑实现通过(1)用户角色(User Role)的不同和(2)用户当前位置(Location)这两个上下文维度来创建上下文空间,根据不同的上下文来为用户提供不同的文件访问功能.

文件的访问方式为,通过智能手机登陆文件服务器,通过认证之后为用户显示符合当前上下文(由用户角色和位置决定)的内容,包括用户可访问的文件列表和文件信息视图.接下来用户根据列表选择需要的文件,此时通过"获取文件"操作得到文件的 Document_ID.之后使用支持 Document_ID 访问文件的系统来进行文件读取.每种访问过程都根据上下文的不同而执行不同的动作.在多处上下文感知应用模块中,本文选择"AccessPoint 服务器根据当前上下文信息为用户手机提供对应的访问文件列表"的功能模块来进行基于 EIPM 的实现.

5.2 应用模型抽象

在例子中的应用环境上下文空间的设计中,上下文分为 2 个维度:(1)用户角色部分 UserRole:小组经理(inGroupManager),小组成员(inGrouper),和非本小组的人员(outGrouper).(2)用户所在地点 Location:会议室(meetingRoom),办公室(officeRoom)和外部区域(outside).

而需要实现的"访问文件列表"功能模块是由 AccessPoint 服务器根据当前上下文信息提供访问文件列表,而这个操作对应于不同上下文的不同操作.操作 1:为小组成员和经理提供会议相关文件列表(对应上下文:小组内成员在会议室访问文件系统);操作 2:提供对应于各自权限的全部文件列表(对应上下文:小组内成员和经理在办公室访问文件系统);操作 3:提供非机密文件的全部列表;并将访问信息记录在服务器的访问日志中(对应上下文:小组经理在公司外访问文件系统);操作 4:提供只包含会议相关文件列表(对应上下文:非本小组成员在公司内部访问文件系统);操作 5:拒绝访问(对应上下文:小组普通成员和非本小组成员在公司外部访问文件系统).

首先要为应用设计各种规则文件,包括应用上下文规则,对象行为定义文件,以及最重要的上下文行为映射规则文件.例子中的规则文件 XML 表示如下图 7 所示:

```
<Context_Set setID="1" classID="1">
 <Regions>
  <Region id="r.1" name="memberInMeetingroom" description="group member in meetingroom">
   <AxisVal dimID="0" startIndex="0" endIndex="1"/>
   <AxisVal dimID="1" startIndex="1" endIndex="1"/>
  </Region>
  <Region id="r.2" name="groupInOffice" description="group member or manager in the office">
   <AxisVal dimID="0" startIndex="0" endIndex="1"/>
   <AxisVal dimID="1" startIndex="0" endIndex="0"/>
  </Region>
  <Region id="r.3" name="managerOutside" description="group manager is not in company">
   <AxisVal dimID="0" startIndex="0" endIndex="0"/>
   <AxisVal dimID="1" startIndex="2" endIndex="2"/>
  </Region>
  <Region id="r.4" name="othersInCompany" description="non-group members in the company">
   <AxisVal dimID="0" startIndex="2" endIndex="2"/>
   <AxisVal dimID="1" startIndex="0" endIndex="1"/>
  </Region>
  <Region id="r.5" name="non-managerOutside" description="non-manager not in the company">
   <AxisVal dimID="0" startIndex="1" endIndex="2"/>
   <AxisVal dimID="1" startIndex="2" endIndex="2"/>
  </Region>
 </Regions>
 <Region_Isotope_Binding regionID="r.1" IsotopeID="AccessPoint. DocListView_01">
 </Region_Isotope_Binding>
 <Region_Isotope_Binding regionID="r.2" IsotopeID=" AccessPoint. DocListView_02">
 </Region_Isotope_Binding>
 <Region_Isotope_Binding regionID="r.3" IsotopeID=" AccessPoint. DocListView_03">
 </Region_Isotope_Binding>
 <Region_Isotope_Binding regionID="r.4" IsotopeID=" AccessPoint. DocListView_04">
 </Region_Isotope_Binding>
 <Region_Isotope_Binding regionID="r.5" IsotopeID=" AccessPoint. DocListView_05">
 </Region_Isotope_Binding>
</Context_Set>
```

图7 上下文-行为映射规则定义

5.3 程序编译与运行结果分析

Isotope 编译器将 ist 文件首先转化成相应的 java 代码,对于在引用 Isotope 类的 java 代码中进行必要的翻译;然后编译生成对应的 class 文件.

根据编译得到的结果运行程序时会发生两种情况:一种情况是行为随上下文的不同而得到不同运行结果.例子中当模拟的上下文模块得到对应于上下文空间中"小组内成员包括经理在会议室访问文件系统"的当前上下文值时,对应了"会议内容"列表如图 8 所示.

```
Problems | @ Javadoc | Declaration | Console
<terminated> MainPart (1) [Java Application] C:\Prog
This is Meeting List content
-------------------------------------
一号文件01.pdf    2009-04-05
RFID讲座.ppt      2009-04-26
项目申请会议资料.ppt      2008-09-30
上下文感知技术讲座.ppt    2009-04-05
联系方式.txt      2009-04-29
下一季度战略方针.doc      2009-05-01
```

图8 会议内容列表

而在运行时分别为其他四种上下文时的程序运行结果如图 9 所示.

另一种情况是在应用背景需求中提到的包括新上下文的引入、映射规则改变以及增加新的行为.例如下面的情况:类 DocListView 类需要为小组普通成员提供在公司外部对文件列表的浏览(原本不包含这个上下文及其对应的操作),则需要做两步:(1)添加上下文 region:r.6 = memberOutside 代表小组普通成员在公司外部的上下文,并填写对应的 Isotope Element 的信息;(2)修改上下文 region r.5 = non-managerOutside,将上下文范围中包括了小组普通成员的部分去除.通过上面的操作,

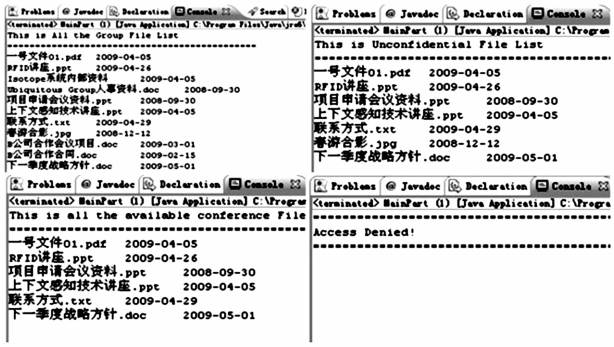

图9　当上下文变化时程序的不同运行结果

系统管理员就将原本对小组普通成员和非本组成员对文件列表的外部访问功能区别开来,分成为两个上下文域,并各自对应了不同动作的 Isotope 列表操作.上下文获取与解释模块通知感应到上下文 r.6 时,则会运行对应的操作.由此可见,根据模拟外界环境上下文的变化,程序运行得到不同的结果.而与此同时,在应用程序中并没有提及上下文的内容,并且上下文和应用程序之间映射关系的变化也不需要程序员在源程序中进行处理,也就是说将功能与环境的绑定在运行时决定.

上面的两项功能分析分别通过模拟外界环境上下文变化,测试程序运行得到不同的结果;以及根据应用环境上下文映射规则的变化,通过管理员控制工具进行处理,使得程序对其动态的适应.由运行结果分析可知,基于动态代理的 EIPM 编程模型通过对上下文信息和编程逻辑进行分离,采用动态拦截和代理的机制来在运行时绑定上下文和编程逻辑,实现了(1)根据当前上下文的变化而动态调整运行不同的行为方法;(2)对应用环境的各种变化,包括上下文、行为及两者的映射规则的变化,不需要修改程序本身就可以处理,并且不需重启系统的情况下完成对应用环境变化的动态适应.

6　结论

本文提出了一种基于动态代理机制的上下文感知编程模型 EIPM.该模型以 Isotope 模型的概念为基础,通过对其扩展和改进,进行了详细的设计,实现了将环境因素和程序控制主体进行解耦,使得上下文感知应用程序开发更加灵活,更好的适应多变的普适计算环境.EIPM 编程模型在编程过程层面将同一对象在不同环境中的不同对象行为和属性值分离出来,在程序管理层面通过映射规则文件和本体文件引入环境上下文因素,并且在运行阶段提供动态的上下文对应行为的选择和调用.在设计理论的指导下,实现了 EIPM 编程模型支撑系统应用框架,为 EIPM 编程模型下的程序开发和运行提供了系统支持.

参考文献:

[1] Weiser M. The computer for the 21st-century[J]. Scientific American, 1991, 265 (3): 94 – 104.

[2] Rarau A, Benta KI, Cremene M. Multifaceted based language for pervasive services with deterministic and fully defined behavior[A]. 2007 Ieee International Conference on Pervasive Services[C]. Los Alamitos, CA: IEEE Computer Society, 2007. 76 – 79.

[3] Yang HI, Jansen E, Helal S. A comparison of two programming models for pervasive computing[A]. International Symposium on Applications and the Internet Workshops[C]. Los Alamitos, CA: IEEE Computer Society, 2006. 134 – 137.

[4] Ranganathan A, Chetan S. Olympus-A high-level programming model for pervasive computing environments[A]. Third IEEE International Conference on 8 – 12 March (PerCom'05)[C]. Los Alamitos, CA: IEEE Computer Society, 2005. 7 – 16.

[5] Xi M, Zhao JZ, Qi Y, et al. Isotope programming model: A kind of program model for context-aware application[A]. MUE: 2007 International Conference on Multimedia and Ubiquitous Engineering[C]. Los Alamitos, CA: IEEE Computer Society, 2007. 597 – 602.

[6] Weichang Du, Lei Wang. Context-aware application programming for mobile devices[A]. Proceeding of the 2008 C^3S^2E Conference[C]. New York: 2008. 215 – 227.

[7] Strang T, Linnhoff-Popien C. A context modeling survey[A]. In Workshop on Advanced Context Modelling. Reasoning and Management associated with the Sixth International Conference on Ubiquitous Computing (UbiComp2004)[C]. Germany: DLR Electronic Library, 2004.

作者简介:

任蓓丽　女,1983 年 11 月出生于陕西榆林.2006 年毕业于西安交通大学计算机科学与技术系,随后在该系继续学习,并于 2009 年 7 月硕士毕业.现在日本富士施乐公司工作,从事普适计算方面的相关研究.
E-mail: rbaylee@gmail.com

齐　勇　男,1957 年 12 月生于西安,工学博士,西安交通大学计算机系教授,博士生导师,从事操作系统、分布式系统和普适计算等方面的研究.　E-mail: qiy@mail.xjtu.edu.cn