面向普适环境的上下文感知中间件研究*

张青川，曾广平
（北京科技大学 计算机与通信工程学院，北京 100083）

摘　要：针对普适环境下上下文感知计算需求，引入广义模型化理论，建立了一种面向通用环境资源的上下文信息数据模型；在此基础上，提出了上下文感知中间件体系框架，并详细阐述了其构件化的实施方案。该中间件平台的上下文获取层能够封装各类感知器捕获的资源信息，中间处理层负责信息的管理、推理和聚合，基于门面模式的上下文访问层提供同步和异步相结合的上下文信息统一访问入口。通过实验测试了平台的时间损耗，表明该中间件可提供通用的上下文感知服务且具有较好的系统性能。

关键词：上下文感知；普适计算；上下文建模；中间件
中图分类号：TP311　　**文献标志码**：A　　**文章编号**：1001-3695(2012)05-1747-04
doi:10.3969/j.issn.1001-3695.2012.05.039

Research on context-aware middleware for ubiquitous environments

ZHANG Qing-chuan, ZENG Guang-ping
(*School of Computer & Communication Engineering*, *University of Science & Technology Beijing*, *Beijing* 100083, *China*)

Abstract: According to the requirement of context-aware computing in ubiquitous environments, this paper proposed a context information data model based on generalized modeling theory with an extensible and multi-leveled hierarchical structure. Moreover, this paper established the context-aware middleware based on the data model which contained context acquisition layer, context handler layer and context access layer. Acquisition layer was used for capturing context information; Handler layer was responsible for context fusion; Access layer implemented the convenient context information access mechanism. Finally, experimental results show that the context-aware middleware has better performance on providing context-aware service.
Key words: context-aware; ubiquitous computing; context modeling; middleware

随着计算的移动化、分布化和网络化，移动设备得到了飞速的发展和普及，软件系统的运行环境也因此变得越来越复杂，同时用户需求不断提升，人们对软件智能化的要求越来越高，这都给软件开发带来了新的挑战。

前 Xerox PARC 首席科学家 Mark Weiser 提出了“普适计算”的思想，这一概念是以用户为中心，强调应用计算本身与其所处环境信息的融合，从而实现了在任何时间、任何地点以任何方式均可便捷地获取应用。然而，目前这种应用的提供还很缺乏，其发展的一个主要瓶颈是环境信息感知的开销及效率问题。当前几乎所有的应用都是自身去完成其所处环境上下文信息的感知工作，随着运行环境的多样变化，感知程序也必须进行相应的调整，这就大大增加了软件开发的成本和难度。

目前，上下文信息的建模方法多是针对不同的应用需求或不同应用领域而设计的，具有一定局限性。常见的建模方法[1]有键值对模型、标记模型、图形模型、面向对象模型、基于逻辑模型和基于本体模型等。上下文信息的访问和共享是上下文感知的目的。上下文感知器直接获取的上下文信息与上层应用程序所需要的上下文信息之间存在一定的差距，因此，如何将已获取的这些底层上下文信息加工成为上层应用程序所需的高层上下文信息[2]，并提供一套高效、便捷的上下文信息访问机制是上下文感知的一个核心问题。Georgia Institute of Technology 的 Anind K Dey 提出的 context toolkit 采用 context interpreter 实现了上下文信息在类型上的简单转换[3]；Trinity College 的 Patrick Fahy 开发的 context-awareness sub-structure 将 rule engine 与 knowledge base 结合在一起，通过知识库中的规则来支持高层的上下文信息抽象[4]；马里兰巴尔的摩郡大学设计的 context broker architecture 包括一种反射式的推理引擎，支持两种类型的推理，使用 ontology 来推理出上下文知识，以及使用启发式知识来推理并解决上下文信息的冲突问题[5]。

在上述研究工作的基础上，本文提出一种面向普适环境的上下文感知中间件，将应用本身的业务逻辑与上下文信息的获取逻辑进行分离，使得应用程序的开发从繁琐的上下文信息感知工作中解脱出来，进而提高了普适计算应用的开发效率。

1　上下文信息数据模型

上下文建模是实现上下文感知计算的前提和出发点。上下文模型是指上下文信息的数据表示形式和组织方式。随着应用需求的不断变化，上下文信息的形式也是千差万别，因此，对上下文模型的通用性和扩展性提出了更高的要求。a）模型要灵活且易于扩展，以适应上下文信息多样、多变的特性；b）模型结构要简单而高效，易于被表达、推理和聚合，以降低系统复杂度，提高系统整体性能。

收稿日期：2011-10-25；**修回日期**：2011-11-27　　**基金项目**：国家自然科学基金资助项目(60973065)；国家“863”计划资助项目(2009AA01Z119)

作者简介：张青川(1982-)，男，河北石家庄人，博士研究生，主要研究方向为分布式计算、人工智能(zqc1982@126.com)；曾广平(1962-)，男，教授，博导，博士，主要研究方向为软计算与智能计算、分布/迁移/网络计算、Linux 操作系统及应用、智能网络与智能通信.

针对上述要求,依据大系统理论[6]中"广义模型化"思路,采用广义知识表达法,为上下文信息定义一个通用的、易扩展的、多级递阶结构的数据模型。上下文信息 context 由若干资源(resource, R)组成,每个资源 R 又是由若干个子资源(sub resource, SubR)和若干个属性(property, P)组成的,如图 1 所示。

下面给出上下文数据模型(context data model, CDM)的形式化描述。

定义 1 CDM 定义为这样的一个三元组:CDM = 〈Cxt, R, P〉。各元组的定义如下:

a) Cxt 为 CDM 的统一访问接口 context,是一组由上下文资源 R 组成的集合,即 Cxt = $\{R_1, R_2, \cdots, R_n\}$ ($n \geqslant 1$)。

b) R 是一个由若干个子资源和若干个属性组成的二元组,即 $R = \langle \sum \mathrm{Sub}R_i, \sum P_j \rangle$ ($0 \leqslant i \leqslant n, 0 \leqslant j \leqslant n$)。其中,$\mathrm{Sub}R_i$ 表示隶属于资源 R 的某一子资源,亦满足上述关于资源 R 的描述;P_j 表示隶属于资源 R 的某一属性。

c) P 是一个由属性名和属性值组成的二元组,即 $P = \langle K, V \rangle$。其中,K 表示属性名,V 表示属性值。

2 上下文感知中间件的设计

上下文感知中间件由上下文获取层、上下文处理层和上下文访问层三部分构成,其体系框架如图 2 所示。

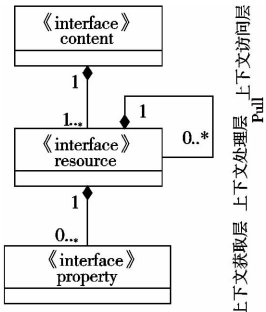

图1 上下文数据模型

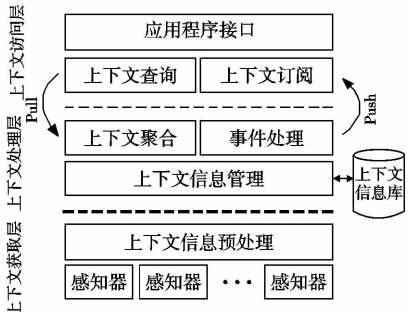

图2 上下文感知中间件体系结构

上下文获取层通过统一编程接口的方式,对各类感知器进行软件抽象,实现其封装,提高了软件的复用性;上下文处理层采用基于规则推理和事件驱动的信息处理机制,实现上下文信息的聚合和上下文事件的管理;上下文访问层采用同步请求和异步通知相结合的上下文信息互补访问模式,为上层应用程序提供便捷的、高效的上下文信息访问机制。

2.1 上下文获取层

上下文信息高效、稳定的获取是上下文感知的前提。上下文获取层通过统一编程接口的方式实现对各类感知器的封装和管理,负责对软件运行上下文状态信息和上下文事件信息进行采集和捕获。由上下文信息预处理模块和感知器组成(图 2)。

感知器按照其感知方式的不同可分为主动型感知器和被动型感知器,利用 OMG IDL 对感知器接口 contextSensor 定义如下:

```
module ustb_smcavs_context _ client {
  interface activeSensor{
    string getName( );
    int start( );
    int stop( );
    boolean isStarted( );
    int getPeriodicTimeUnit( );
    void setPeriodicTimes( in int duration);
    void setLisener( in ustb_smcavs_context:: ContextLisener
    contextLisener);
  };
    interface passiveSensor{
      object getUnderlyingValue( );
    };
};
```

a) 主动型感知器(activeSensor)。主动型感知器负责收集原始上下文信息,并把这些信息按照设置的调度策略主动地发送给上下文信息预处理模块。在主动型感知器 activeSensor 中,getName( )方法用于获取感知器的名称标志;start( )、stop( )和 isStarted( )方法提供对感知器生命周期的控制和对当前运行状态的查询;getPeriodicTimeUnit( )方法用于获取感知器获取上下文信息的最小周期时间单元;setPeriodicTimes( )方法用于设置感知器的感知周期;setLisener( )方法用于设置上下文信息预处理模块中接收原始上下文信息的监听器 context Lisener。

b) 被动型感知器(passiveSensor)。被动型感知器受上下文信息预处理模块管理,当被动型感知器被调用时,通过 getUnderlyingValue( )方法返回其监视的原始上下文信息。

c) 上下文信息预处理。该模块负责管理系统中的上下文感知器,并依据上下文模型对感知器获取的粗糙的原始上下文信息进行格式上的统一化处理,将其加工成为底层上下文信息,以支持上下文信息的自由传输、便捷互操作和高效存储。

2.2 上下文处理层

上下文处理是上下文感知中间件的核心部分,采用基于规则推理和事件驱动的信息处理机制,实现上下文信息的聚合和上下文事件的管理,由上下文信息库、上下文信息管理模块、上下文聚合模块和事件处理模块组成(图 2)。

2.2.1 上下文管理

上下文管理模块负责底层上下文信息的管理和维护,上下文信息分为实时上下文信息和历史上下文信息两种。

a) 实时上下文信息存储于内存中的动态上下文信息模型中,上下文管理模块依据上下文获取层捕获的上下文信息,动态创建上下文信息模型,并对其进行结构上的调整和状态上的更新。

b) 历史上下文信息存储于上下文信息库中,上下文管理模块根据上下文信息自身特点和应用需求的不同,按照内存中已创建的上下文信息模型的结构,将需要持久化的上下文信息存储到上下文信息库中,为上下文信息的聚合推理提供数据支持。

2.2.2 上下文事件处理

上下文事件处理模块负责管理上下文中的各类事件信息,为上下文访问层提供事件查询及订阅支持。

软件系统中的事件通常蕴涵两种语义[7]:a) 系统中某些具有特定意义的瞬时发生,表示为事件被触发的时刻;b) 系统中某些实体状态的变化,表示为事件作用的时间段。

本文提到的上下文事件属于第二种语义,指由系统所处环境变化或系统自身行为产生的系统状态信息的变化,这些变化经由上下文感知器的捕获,进而引发了上下文信息的实时变更,具体来说,体现在上下文管理模块中动态上下文信息模型在结构上的调整或在状态上的更新。

上下文信息的每一次变更均可定义为一个上下文事件,即上下文感知器的每一次上下文信息捕获都可能产生一个潜在的上下文事件。从应用的角度来说,单个上下文事件的发生意义并不大,更值得关注的是多个事件的组合发生,即高复杂性上下文事件的描述。因此,本文引入事件代数[8]以丰富和扩充上下文事件的表示能力,实现了对复杂事件的表述。下面给出上下文事件及其操作的形式化描述。

定义 2　上下文事件(context event, CE)用来描述上下文信息的变更过程,依据其是否可再拆分定义为原子上下文事件(atomic context event, ACE)和复合上下文事件(composite context event, CCE),即

$$CE = \langle ACE \mid CCE \rangle$$

定义 3　原子上下文事件 ACE 可定义为这样的一个四元组:

$$ACE = \langle id, type, source, sourceType, timeStamp, duration \rangle$$

其中:id 表示上下文事件的标志;type 表示上下文事件类型,对应上下文信息模型在结构上或者状态上的两种变更方式,可划分为结构型事件(architecture event)和状态型事件(status event)两种;source 表示上下文事件源,采用 URI 的表示方式,指明上下文信息模型中所对应的具体节点;sourceType 表示上下文事件源的类型,即资源事件源或属性事件源;timeStamp 表示上下文事件被检测到的起始时刻;duration 表示上下文事件的发生期长度。

定义 4　上下文事件运算集(context even operations, CEO)是一组定义在原子事件上的运算的集合,即

$$CEO = \{\neg, \triangle, \nabla, \rightarrow, \lhd, \rhd\}$$

其中,$\neg$ 为事件否定运算,表示某一事件不发生;$\triangle$ 为事件析取运算,表示两个事件同时发生;∇ 为事件合取运算,表示两个事件至少有一个发生;$\rightarrow$ 为事件相继运算,表示两个事件相继发生;$\lhd$ 为事件前序运算,表示在某一事件之前发生;$\rhd$ 为事件后序运算,表示在某一事件之后发生。

定义 5　复合上下文事件 CCE 是由一个或多个原子事件集合 $S = \{ACE_1, ACE_2, \cdots, ACE_n\}$ $(n \geqslant 1)$,通过上下文事件运算集中一次或多次事件运算的叠加来描述原子事件之间的逻辑复合关系,形成对复杂上下文事件的描述。

2.2.3　上下文聚合

上下文聚合是指将由上下文获取层捕获到的底层上下文信息进行聚合推理,加工成为满足上下文信息查询需求的高层上下文信息,旨在拉近上下文信息与上层应用需求之间的距离[9]。上下文聚合模块由上下文过滤单元和上下文推理单元两部分组成,完成底层上下文信息经过滤、推理到融合的信息聚合过程,如图 3 所示。

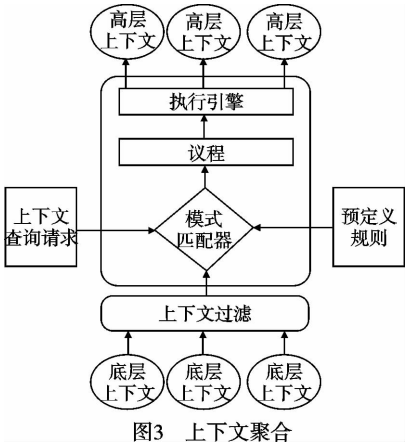

图3　上下文聚合

上下文过滤单元是上下文聚合的第一步,用于对底层上下文信息进行过滤,剔除底层上下文中的不精确、不稳定的信息,以提高上下文信息的准确度。底层上下文信息是对软件运行环境中某一资源或者资源的某一属性在某一个时间点上的描述,这些信息在形式上是离散的、分散的。因此,底层上下文信息在一定程度上存在不确定性、不稳定性和局限性,甚至信息间存在一些冲突和错误等,需要对其进行过滤,为上下文推理提供支持。

上下文推理单元用于对底层上下文信息进行推理以消除上下文信息间存在的冲突和错误。其主要功能是将底层上下文信息,结合上下文查询请求及预定义的推理规则,采用基于 rete 算法的前向规则推理,得到与上层应用需求适配度高的高层上下文信息。这部分功能主要是由上下文规则引擎实现的。

上下文规则引擎由模式匹配器、议程和执行引擎构成(图 3)。推理引擎的输入由底层上下文信息和上层访问请求组成;模式匹配器负责将这些输入信息与预定义的规则进行模式匹配,同时进行优先级排队生成最优的执行顺序,并将其放入议程中;执行引擎会按照顺序依次执行议程中的推理规则,从而完成底层上下文的数据融合,得到更有价值的高层上下文。

2.3　上下文访问层

上下文门面访问层以定制统一访问接口的方式,向上层应用程序提供同步请求模式和异步通知模式相结合的上下文信息互补访问机制,旨在实现系统环境资源信息的即时共享和便捷访问。上下文同步请求访问模式,即 Pull 模式,它依据上层应用程序的上下文查询请求,通过上下文处理层中的上下文聚合模块将上下文管理模块获得的底层上下文信息进行聚合推理,加工成为满足上下文查询请求的高层上下文信息,为上层应用程序提供即时的上下文信息查询服务。上下文异步通知访问模式,即 Push 模式,以发布/订阅方式实现上层应用程序对上下文信息的异步访问。上层应用程序通过 context 门面类在上下文感知中间件中注册监听器,指定要订阅的上下文信息及其触发事件类型,当所指定的监听事件发生时,订阅的上下文信息将被主动推送给其订阅者。

上下文访问层由上下文查询模块、上下文订阅模块和上下文门面接口组成(图 2)。在上层上下文聚合模块和事件处理模块的支持下,上下文查询模块和上下文订阅模块分别实现了上下文信息的同步请求和异步通知两种访问模式;基于门面模式设计的上下文接口类 Context 作为提供上下文信息服务的唯一高层类,为上层应用程序提供上下文信息高效、便捷的访问入口。利用 OMG IDL 对其定义如下:

```
module ustb_smcavs_context_ api {
    typedef sequence <string> ResourceArray;
    typedef sequence <string> PropertyArray;
    interface Context{
      string getName();
      int destroy();
      int creatResource(in string resource);
      int removeResource (in string resource);
      int creatProperty(in string resource, int string value);
      int removeProperty(in string resource, int string property);
      ResourceArray getSubResource(in string resource);
      PropertyArray getProperties(in string resource);
      object getValue(in string property);
      int setValue(in string attribute, in Object value);
```

```
        int register( in ustb_smcavs_context::ContextLisener contextLisner,in Object eventType, in string uriStr);
        int registerByEQL( in ustb_smcavs_context::ContextLisener contextLisner,in string queryStr);
        void unRegister( in int regID);
    };
};
```

其中,createResource( ) 和 removeResource( ) 方法用于创建和移除某一资源信息;createProperty( ) 和 removeProperty( ) 方法用于创建和移除某一资源的属性信息;getSubResource( ) 和 getProperties( ) 方法用于对资源节点和属性节点动态信息的即时查询;getValue( ) 方法用于查询属性节点的即时属性值;setValue( ) 方法用于对某一资源的属性信息进行更新;register( ) 方法指定一个上下文监听器 contextListener,用以接收上下文管理模块提供的事件通知;参数 eventType 用于指定上层应用程序感兴趣的上下文事件类型;参数 uriStr 用于指定当然监听事件发生时需要查询的上下文信息;registerByEQL( ) 方法采用 EQL 来描述所要查询的上下文信息;unRegister( ) 方法用于注销上下文监听器 contextListener。

3 上下文感知中间件总体框架

依据上下文感知中间件的设计,基于笔者团队提出的 SMC 构件模型[10]对中间件平台进行编程实现,并通过系统全局结构可视化展台,给出其体系结构描述,如图 4 所示。

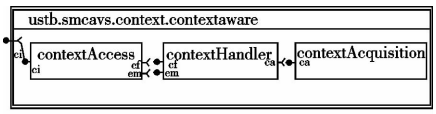

图4 上下文感知构件

构件 contextAware 作为上下文感知中间件的顶层构件,由三个子构件 contextAccess、contextHandler 以及 contextAcquisition 组成,分别对应图 3 中的上下文获取层、上下文处理层及上下文访问层。其对外提供的服务接口 ci 即为第 2.3 节所论述的上下文门面类 context。下面分别给出构件 contextInfo 所辖子构件的结构描述及其功能说明。

a)上下文获取构件 contextAcquisition。ContextAcquisition 作为上下文处理层的实现构件,由子构件 roughContextHandler 以及多个感知器子构件 sensors 组成,其结构如图 5 所示。其中,构件 roughContextHandler 用来管理上述隶属于它的所有感知器构件 sensors,并将其获取的原始上下文信息,依据上下文信息元素的 URI 表示方式进行统一的数据格式预处理。

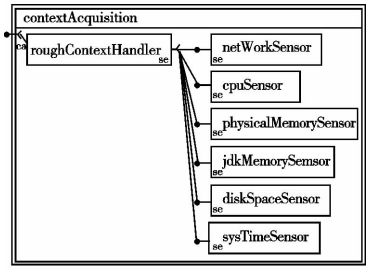

图5 上下文获取构件

b)上下文处理构件 contextHandler。ContextHandler 作为上下文处理层的实现构件,由四个子构件 contextFusion、eventManagement、contextDB 以及 contextManagement 组成,其结构如图 6 所示。其中,构件 contextManagement 提供了一个上下文信息模型的基本实现,并给出上下文信息模型的根节点,在此基础上,依据上下文获取构件 contextAcquisition 收集的上下文信息对上下文信息模型进行结构上的动态调整和数值上的实时更新;构件 contextFusion 负责对上下文信息进行过滤推理,消除其间存在的冲突和错误,使其聚合成为更有价值的高层上下文信息;构件 eventManagement 负责对系统上下文信息模型的结构变化和数值更新进行实时监听,并将其封装成 contextEvent 对象,为上下文异步通知访问模式提供事件订阅支持;构件 contextDB 负责存储所有需要持久化的历史上下文信息。

c)上下文访问构件 contextAccess。ContextAccess 作为上下文访问层的实现构件,由三个子构件 contextInterface、contextQuery 以及 contextSubScription 组成,其结构如图 7 所示。其中,构件 contextQuery 在构件 contextFusion 支持下,实现上下文同步请求访问模式;构件 contextSubScription 在构件 eventManagement 支持下,实现上下文异步通知访问模式单元;构件 contextInterface 负责将上述两种上下文访问模式的对外服务接口进行统一的封装。

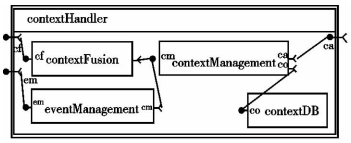

图6 上下文处理构件

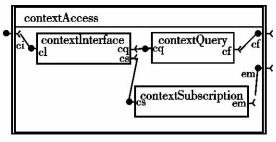

图7 上下文访问构件

4 实验

本文以获取本地主机 CPU 利用率、获取本地网络速度及获取本地主机可用内存值三个应用为例,分别采用直接编程获取方式、基于 context toolkit 编程获取方式和感知中间件获取方式对时间开销进行测试,并利用 MATLAB 对测试结果进行分析,如图 8 ~ 10 所示。

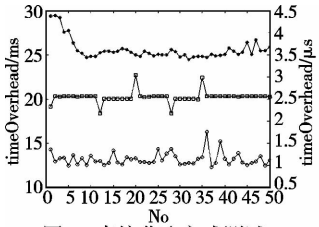

图8 直接获取方式测试

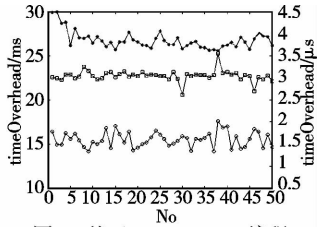

图9 基于context toolkit编程获取方式测试

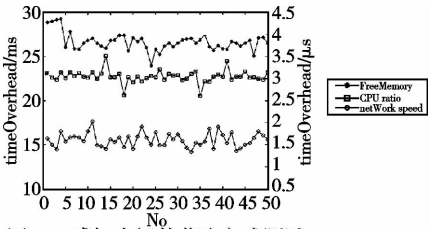

图10 感知中间件获取方式测试

实验分为 50 组,每组进行 100 次测试并求其时间开销的平均值。图中方形标记线对应左侧 y 轴给出获取本地主机 CPU 利用率的平均时间开销,三种方式测试所得的平均值分别为 20.22 ms、22.77 ms、22.75 ms,方差为 0.31、0.55、0.46;圆形标记线对应左侧 y 轴给出获取本地网络速度的平均时间开销,三种方式测试所得的平均值分别为 14.07 ms、16.13 ms、16.18 ms,方差为 6.78、7.67、7.62;星型标记线对应右侧 y 轴给出获取本地主机可用内存值的平均时间开销,三种方式测试所得的平均值分别为 3.81 μs、3.90 μs、3.89 μs,方差为 0.21、0.15、0.14。

(下转第 1760 页)

(上接第 1750 页)

直接获取方式的平均时间开销表示为 t_1,其他获取方式的平均时间开销表示为 t_2,则本文定义此种获取方式的系统损耗率 $\theta=(t_2-t_1)/t_1$。依据上述实验数据,可分别计算出基于 context toolkit 编程获取和感知中间件获取两种方式的系统损耗率:获取 CPU 利用率的损耗率分别为 12.65%、12.51%,获取网络速度的损耗率分别为 14.64%、14.99%,获取可用内存值的损耗率分别为 2.36%、2.10%。可见,中间件获取方式与基于 context toolkit 编程获取方式相比,系统损耗率基本相同,但数值方差较小,因此,感知中间件具有更优的系统稳定性和性能。

5 结束语

本文针对普适环境下上下文感知计算的应用需求,基于广义知识表达法,为上下文信息构建了一种通用、易扩展且结构简洁高效的数据模型;并在此模型基础上构造上下文感知中间件,包括上下文获取层、上下文处理层和上下文访问层,实现了应用本身的业务逻辑与上下文信息感知的逻辑相分离,使得应用程序的开发从繁琐的上下文信息获取和管理工作中解脱出来。实验表明该中间件可提供通用的上下文感知服务且具有较好的系统性能。

参考文献:

[1] ARDISSONO L, GOY A, PETRONE G. A framework for the development of distributed, context-aware adaptive hypermedia applications [C]//Proc of the 5th International Conference on Adaptive Hypermedia and Adaptive Web-based Systems. Berlin: Springer-Verlag, 2008:259-262.

[2] 李蕊,李仕伦. 面向上下文感知计算的贝叶斯网络结构自学习算法的研究[J]. 计算机应用研究,2010,27(1):107-110,113.

[3] HRISTOVA A, BERNARDOS A M, CASAR J R. Developing an ambient home care system: context toolkit-based design and implementation[C]//Proc of the 9th International Conference on Parallel and Distributed Computing, Applications and Technologies. Washington DC:IEEE Computer society,2008:455-461.

[4] Van KRANENBURG H, BARGH M S, IACOB S, *et al*. A context management framework for supporting context-aware distributed applications[J]. IEEE Communications Magazine,2006,44(8):67-74.

[5] SEKKAS O, ANAGNOSTOPOULOS C D, HADJIEFTHYMIADES S. Context fusion through imprecise reasoning[C]//Proc of IEEE International Conference on Pervasive Services. Washington DC: IEEE Computer Society, 2007:88-91.

[6] 徐序彦, 王枞, 郭燕慧. 大系统控制论[M]. 北京:北京邮电大学出版社, 2005:20-40.

[7] ETZION O, NIBLETT P. Event processing in action[M]. Stamford: Manning Publications,2010:61-85.

[8] 刘家红, 吴泉源. 一个基于事件驱动的面向服务计算平台[J]. 计算机学报, 2008,31(04):587-599.

[9] ROY N, GU Tao, DAS S K. Supporting pervasive computing applications with active context fusion and semantic context delivery[J]. Pervasive and Mobile Computing,2010,6(1):21-42.

[10] YUE Yang, AI Dong-mei, ZENG Guang-ping. A dynamic evolution framework for SoftMan system[C]//Proc of International Conference on Computer Application and System Modeling. Piscataway: IEEE Press,2010:262-266.