动态上下文知识的获取与共享

於志勇 周兴社 王海鹏 倪红波 於志文 王 柱
(西北工业大学计算机学院 西安 710072)

摘 要 上下文感知系统需要获取和共享多种上下文知识,不仅包含环境参数的具体取值,也包含对环境状态描述时所用的词汇,也就是概念。上下文知识具有与生俱来的动态性,在设计时难以预知系统可能涉及的上下文信息。而大多数已有的系统缺乏完善的动态上下文知识维护机制,因此不支持上下文感知应用的运行时的扩展。提出一个基于本体的层次化上下文模型,以及基于此模型的动态上下文知识获取与共享框架(DCASI)。框架中按需分散获取机制让各类上下文知识出自最有资格定义它的实体,使得上下文知识充足而又不冗余;双库集中共享机制按共享的不同作用维护两个知识库,有效支持了新上下文知识的定义和发现。原型系统验证了本框架的有效性。

关键词 上下文感知,动态上下文知识,上下文模型,上下文框架
中图法分类号 TP391 **文献标识码** A

Dynamic Context Knowledge Acquisition and Sharing

YU Zhi-yong ZHOU Xing-she WANG Hai-peng NI Hong-bo YU Zhi-wen WANG Zhu
(School of Computer Science, Northwestern Polytechnical University, Xi'an 710072, China)

Abstract Context-aware systems need to acquire and share various kinds of context knowledge, which not only include the concrete values of environmental parameters, but the vocabularies used to describe the environmental states, i. e., concepts. Context knowledge is inherently dynamic. It is difficult to predict the context information involved in the system at the moment of design. Most of existing systems lack of effective mechanisms for dynamic context knowledge maintenance, thus unable to support run-time scalability of context-aware applications. This paper introduced an ontology-based hierarchical context model. On the basis of this model, a dynamic context knowledge acquisition and sharing infrastructure (DCASI) was proposed. The mechanism of distributed acquisition on demand ensures that each class of context knowledge comes from the entity with the most competence to define it, which makes the context knowledge in the system sufficient but not redundant. The mechanism of centralized sharing with double-repository maintains two knowledge repositories according to different functions, thereby efficiently supports the definition and discovery of new context knowledge. A prototype system was implemented and the proposed infrastructure was validated.

Keywords Context-aware, Dynamic context knowledge, Context model, Context infrastructure

普适计算系统采集用户及其所处环境的信息,识别用户意图,调整服务内容,同时避免干扰用户,这种特征被称为上下文感知[1,2]。显而易见,在一个现实的智能空间中,上下文的种类和取值都是无穷无尽的。已有的方案是:事先选定现实世界中与应用相关的部分[3]。这种方案不能维护动态的上下文知识,限制了系统的扩展性。

上下文知识是指用于描述上下文的概念(或称为词汇)、概念之间的关系以及概念的实例等信息。其动态性不仅表现在实例的不断变化(例如,一个房间的温度是不断变化的),还表现在概念的不断变化:无论在设计时考虑多么周全,后来总需要增加新的上下文概念;上下文感知应用和传感器可能在运行时加入或退出;高级上下文还会不断调整其所依赖的基本上下文及构成规则。例如,系统在最初设计时,只监控房间的"温度",后来有了新的应用,需要监控房间的"亮度",这样就需要向原有的系统中加入亮度传感器和这个新应用;原应用还可能重新定义多大的温度值才是"高温"。

现有的大多数系统,其上下文模型在形式化能力和表达能力上都有不足[4],不容易加入新的上下文,很少支持上下文感知应用的扩展[5],更没有完善的动态上下文知识维护机制。当需要更新上下文知识,尤其是增加概念或改变规则时,只能重新部署系统,于是带来了维护困难、容易出错、花费大量时间等问题。

到稿日期:2008-10-27 返修日期:2009-01-07 本文受国家自然科学基金项目(60803044),国家863基金项目(2006AA01Z198),教育部高等学校博士学科点专项科研基金项目(20070699014),西北工业大学博士论文创新基金项目(CX200814)资助。

於志勇(1982-),男,博士生,主要研究领域为普适计算、上下文感知系统等,E-mail:yuzhiyong@mail.nwpu.edu.cn;周兴社(1955-),男,教授,博士生导师,CCF会员,主要研究领域为嵌入式计算、分布式计算、传感器网络等;王海鹏(1975-),男,博士,讲师,CCF会员,主要研究领域为普适计算、资源管理等;倪红波(1975-),男,博士生,讲师,主要研究领域为普适计算、嵌入式网络等;於志文(1977-),男,博士,讲师,CCF会员,主要研究领域为普适计算、智能信息处理、个性化技术等;王 柱(1983-),男,博士生,主要研究领域为普适计算等。

为了解决这个问题，本文提出了二集合三层次上下文模型(2×3cm，2 Sets and 3 Layers Context Model)，以及在此模型上构建的动态上下文知识获取与共享框架(DCASI，Dynamic Context Knowledge Acquisition and Sharing Infrastructure)。框架中包含了按需分散获取机制和双库集中共享机制，从如下几个方面支持上下文感知应用的开发。

· 与应用解耦合。解决了以往系统中上下文的获取和使用与单个应用集成过于紧密的问题[6]，增强了框架的可扩展性。

· 提供开放的智能空间平台。用户通过定义可插入、可共享的上下文概念，从而构建自己的应用[7]。

· 高效率地支持应用透明化。封装并重用传感器，避免不必要的传输和推理，为应用提供无冗余的上下文。

为了验证DCASI的有效性，本文以智能博物馆为应用场景，实现了原型系统iMuseum[8]。系统通过多种传感器感知博物馆内的情景变化，从而帮助游客有效获取文物信息，也帮助管理员实时掌握环境参数(文物对温度、湿度、亮度等都很敏感)和安全信息(如防盗)，以更好地保护文物。其中有一个上下文感知应用iGuide：当游客对身边的文物感兴趣，那么他/她所持的PDA将自动播放该文物的多媒体解说。

第1节介绍本研究的相关工作；第2节给出上下文模型；第3节描述上下文框架；第4节和第5节分别详细说明上下文获取与共享机制；第6节实现原型系统；最后总结全文。

1 相关工作

为了简化上下文感知应用的开发，采用框架是必要的。上下文感知计算的系统框架主要负责对实体和上下文进行管理，屏蔽计算环境的复杂性、多样性和动态性。框架不仅为应用提供获取上下文数据的途径，还允许新的分布异构数据源的简单注册[9,10]。

Context Toolkit[11]中的Widget对上层隐藏了传感器处理的复杂性。Hydrogen[12]对远程上下文和本地上下文进行区分，当设备相互靠近时，能通过P2P的方式交换上下文。Gaia[13]使应用能查询和注册特定上下文信息和高级上下文对象，需要推理的应用可指定上下文知识库中必须包含的事实。Sentient Object Model[14]中同一时刻只有一个上下文是活跃的，从而一次推理仅需执行少数规则。以上框架缺乏负责上下文存储和共享的中心节点(即使有中心节点也仅用于实体注册)，假设普适设备的计算能力足够强大，能收集、存储、合成自己所需的上下文，但这个假设在大多情况下并不成立。

一些具有集中共享节点的框架有效克服了轻便设备的资源限制，一定程度上提高了网络性能。CoBrA[15]的中心节点Context Broker由上下文知识库、上下文推理引擎、上下文获取模块和隐私管理模块组成，为多智能体维护一个共享的上下文模型。SOCAM[16]的中心节点Context Interpreter通过分布的上下文提供者获取上下文数据，并把加工后的形式提供给应用，其中用Upper-Level Ontology表示通用上下文信息，Domain Ontology利用import标记扩展上层本体以表示详细的领域信息。基于CC/PP协议的上下文服务器CS4WMS[17]中包含合并模块，以在属性值更新时选择覆盖、锁定或添加。这些框架有一定的静态可扩展性和动态实例更新能力，但没有运行时概念更新能力。如果要增加新的上下文概念或者改变推理规则，则只能手动修改本体，然后重启系统。

部分研究者意识到这种不足，在他们的集中式框架中允许运行时更新上下文概念。Context Managing Framework[18]使用上下文"认知服务"从上下文"原子"中创建高级上下文对象，而新的认知服务可以容易地加入系统。CASS[19]的规则存储在与解释器分离的数据库中，当规则改变时，不用重启系统。但这些框架只能由中心节点定义或更新上下文概念，使得用户没有能力独立构建自己的应用。它们也没有按需传输和处理上下文知识，这使网络和中心节点的效率低下。

DCASI不仅吸取并综合了以往上下文感知系统框架的优点，如封装传感器、采用集中式共享上下文知识、由应用定义高级上下文概念、仅对知识或规则的活跃部分推理、可在运行时更新上下文概念等，而且通过按需分散获取机制和双库集中共享机制，探索了从根本上解决可扩展问题和效率问题的途径。

2 上下文模型

基于本体的上下文模型可以在不同工具和系统中共享词汇，采用基于领域知识的推理技术，而层次化的数据模型可以高效地引进传统数据库管理技术。结合基于本体的模型[20]和层次模型[21]，提出二集合三层次上下文模型，即2×3cm，如图1所示。

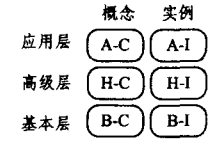

图1 上下文模型(2×3cm)

首先，把上下文知识分为概念和实例两个集合。这种思想源于描述逻辑[22]的TBOX(关于问题领域的一般知识)和ABOX(关于个别问题的知识)。概念用于描述一类个体的词汇、关系或状态，实例用于描述特定个体的状态。将本体区分为概念和实例的目的在于有效支持本体的运行时的更新，从而使得系统不局限于初始定义的本体。

然后，把上下文知识分为3个层次。基本层上下文直接来自传感器；高级层上下文不能直接从传感器那里得到，需要进行一定的融合或推理，即合成；应用层上下文是全部上下文的一个抽取，包含某个应用所需要的特定部分。把上下文知识分层是为了有效支持高级上下文处理和应用透明化。

这样，所有上下文知识被分为6类：基本层上下文概念(B-C)、基本层上下文实例(B-I)、高级层上下文概念(H-C)、高级层上下文实例(H-I)、应用层上下文概念(A-C)、应用层上下文实例(A-I)。

在iMuseum系统中实现了2×3cm的表示。采用OWL-DL[23]表示基本层上下文概念、基本层上下文实例、高级层上下文实例，用OWL-DL和SWRL[24]结合的方式表示高级层上下文概念，用SWRL表示应用层上下文概念。OWL-DL是保证推理完全性条件下表达能力最强的本体语言，SWRL是对OWL-DL扩展的语义网规则语言，能与OWL-DL无缝集成，而且同时具有定义规则和查询语句的能力，这使我们可以将所有上下文知识统一进行表示和处理。

以 iGuide 为例，限于篇幅仅给出部分上下文知识的表示，如表 1 所列。以一切从应用需求出发的原则，应用需要“游客及其感兴趣的文物”，然后从文物内容库中取出该文物的多媒体解说，发送到游客所持的 PDA。其中“感兴趣”就是高级层上下文概念(iGuide 将其定义为：如果游客在某文物前停留超过 5s)。为了获得它，又需要有“游客”、“文物”、“停留地点”、“停留时间”这些基本层上下文概念。

表 1　上下文知识的表示

		概念	实例
应用层	语义	定义概念“我需要游客及其感兴趣的文物”。	游客是 Visitor_1，文物是 Relic_1。
	表达*	im:Visitor(? x) ∧ im:Relic(? y) ∧ im:interest_in(? x,? y)→query:select (? x,? y) ∧ query:columnNames("visitor","relic")	游客　　文物 Visitor_1　Relic_1
高级层	语义	定义概念“感兴趣”。	Visitor_1 对 Relic_1 感兴趣。
	表达	〈owl:ObjectProperty rdf:about="&im;interest_in"〉 〈rdfs:domain rdf:resource="&im;Visitor"/〉 〈rdfs:range rdf:resource="&im;Relic"/〉 〈/owl:ObjectProperty〉 im:Visitor(? x) ∧ im:Relic(? y) ∧ im:localize_near(? x,? y) ∧ im:localize_for(? x,? z) ∧ swrlb:greaterThan(? z,5)→im:interest_in(? x,? y)	〈im:Visitor rdf:about="&im;Visitor_1"〉 〈im:interest_in rdf:resource="&im;Relic_1"/〉 〈/im:Visitor〉
基本层	语义	定义概念“游客”、“文物”、“停留地点”和“停留时间”。	Relic_1 是文物，Visitor_1 是游客，Visitor_1 在 Relic_1 附近，Visitor_1 停留了 7 秒。
	表达	〈owl:Class rdf:ID="Visitor"/〉 〈owl:Class rdf:ID="Relic"/〉 〈owl:ObjectProperty rdf:ID="localize_near"〉 〈rdfs:domain rdf:resource="#Visitor"/〉 〈rdfs:range rdf:resource="#Relic"/〉 〈/owl:ObjectProperty〉 〈owl:DatatypeProperty rdf:ID="localize_for"〉 〈rdfs:domain rdf:resource="#Visitor"/〉 〈rdfs:range rdf:resource="&xsd;int"/〉 〈/owl:DatatypeProperty〉	〈im:Relic rdf:about="&im;Relic_1"/〉 〈im:Visitor rdf:about="&im;Visitor_1"〉 〈im:localize_near rdf:resource="&im;Relic_1"/〉 〈im:localize_for rdf:datatype="&xsd;int"〉7 〈/im:localize_for〉 〈/im:Visitor〉

* “im”是初始本体的名字空间。粗体表达式是人读形式的 SWRL，其机读形式也遵循 OWL。

3　上下文框架

DCASI 以 2×3cm 为基础，定义了 3 类在网络上分布的计算实体，如图 2 所示：上下文提供者(简称“提供者”)、上下文感知应用(简称“应用”)和上下文服务器(简称“服务器”)。

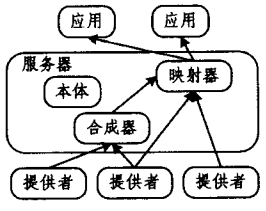

图 2　上下文框架(DCASI)

上下文提供者是有能力以约定的表示和传输方式向服务器提供基本层上下文实例的抽象实体。一般的传感器通过加强能力就是提供者。有些传感器没有能力将上下文封装为特定的表示方式，或者没有能力通过网络接口传输上下文，则它需要借助有这些能力的设备代为封装和传输，这时就把代理看作提供者。新的提供者可能在运行时加入，它通过定义新的基本层上下文概念向服务器注册自己，以表明自己提供上下文的能力(B-C 的来源之一)。

上下文感知应用是使用应用层上下文实例实现某种具体服务的抽象实体。它本身包含了如何响应环境变化的策略引擎，但不直接从提供者收集基本层上下文实例，也不自己合成高级层上下文实例。新的应用可能在运行时加入，它通过定义新的应用层上下文概念向服务器注册自己，以表明自己对上下文的需求(A-C 的来源)。需求中还可能附带定义高级层上下文概念的规则(H-C 的来源)，让服务器代为合成。

上下文服务器是 DCASI 的核心，它最先被初始化，然后不断有新的提供者和新的应用向它注册。它是应用与提供者之间的桥梁，负责建立需求与能力之间的映射关系，并在提供者发布更新后，获取新的应用层上下文实例，这项工作由映射器完成。服务器还要负责合成应用所需要的高级层上下文实例，该功能由合成器实现。由于所有上下文知识都是基于本体表示和推理的，因此服务器还要维护一个动态的本体。

以智能博物馆为应用场景，按 DCASI 框架实现了 iMuseum 原型系统(详见第 6 节)。以 iGuide 为例，上下文服务器和应用(服务端)运行在博物馆的后台计算机上，每位游客手持的 PDA(附带 RFID 阅读器)作为应用客户端，每件文物都以 RFID 标记，PDA 上运行的 RFID 上下文提供者能提供“游客停留在哪个文物附近”和“游客停留多长时间”等基本层上下文。

4　按需分散获取

某类上下文知识如果由不够资格的实体来收集，则很可能会出现收集步骤繁琐或使用起来不贴切的情况。应该让各类知识出自最有资格获取它的实体，可使得上下文知识充足而不冗余。

因此 DCASI 中包含了按需分散获取机制，运行时新的上下文知识通过如下 6 种增长方式动态加入到系统中：

(1) 上下文服务器最先定义基本层上下文概念。以后的增长都是基于此共同语义的，因此这个基础必须较为完善。通常，只有系统设计者了解这个智能空间的上下文的大致组成和分类，从而可定义此基础，形成初始本体。

(2) 上下文感知应用加入或更新应用层或高级层上下文概念(即需求和规则)。应用是高级上下文的使用者，因此即使由别的实体定义，也必然要与应用沟通，以了解它确切的上下文需求。由应用自身定义，免去了沟通以及沟通可能带来的误解。同理，应用层上下文概念亦如此。

(3) 上下文提供者加入新的基本层上下文概念(即能力)。这是因为提供者直接封装了传感器，它的开发者最了解传感器的功能，由其定义基本层上下文概念最准确。

(4) 上下文提供者加入新的基本层上下文实例。这在 DCASI 中已经定义，是其能力的具体表现。

(5) 上下文服务器获取高级层上下文实例。高级层上下文实例不能由单个提供者提供，也不能由应用自己合成，通过

服务器中的合成器获取高级层上下文实例较合适。

(6) 上下文服务器获取应用层上下文实例。如果将服务器收集的所有基本层上下文实例和高级层上下文实例都发给应用,让其从中挑选所需的上下文实例,则传输了很多无用的信息,并给应用造成负担。所以,系统通过服务器中的映射器获取应用层上下文实例。

系统的全局本体是指引用了系统中所有本体文件的本体。方式(2)、(3)、(4)都是分散定义的新知识,它们在初始本体(1)的基础上,不断更新全局本体,如图 3 所示。这 3 种方式获取的知识要从外界输入到服务器,方式(5)的输入输出都是服务器,方式(6)获取的知识要从服务器输出到外界。这些获取都是按需进行的:只订阅与应用层相关的基本层上下文实例、只合成跟随基本层更新且与应用层相关的高级层上下文实例、只查询跟随基本层更新的应用层上下文实例。例如,如果房间内有温度传感器,而没有温度的应用,则温度提供者并不向服务器传输当前温度;如果应用定义了"高温"这个高级层概念(需要"温度"这个基本层概念),但温度提供者未发布更新,则服务器并不进行判断"是否达到高温"的推理;如果应用只需要"高温"警告,则未到高温时,服务器并不向应用传输当前温度。显而易见,这样的机制节省了带宽、存储空间、处理时间以及能量消耗。

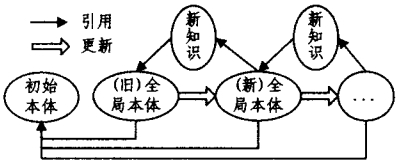

图 3 全局本体的运行时更新过程

在 iMuseum 中实现了按需分散获取机制。采用 SWRL 封装上下文的需求消息,保证了所需的内容是确切定义的,避免了冗余信息的传输。使用 ActiveMQ[25](JMS 的一个实现),服务器与应用之间、服务器与提供者之间可以进行松耦合通信,从而完成各种上下文知识的异步、细粒度、点对点传输,如图 4 所示。

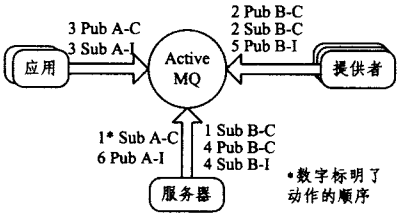

图 4 基于发布/订阅的上下文知识传输

服务器首先订阅(Sub)能力和需求,然后提供者和应用加入系统后分别发布(Pub)自己的能力或需求,同时订阅来自服务器的回应。服务器收到能力和需求后,映射器更新两者之间的映射关系,发现哪些能力被用到,便发布对该能力的需求,同时订阅来自该提供者的回应。提供者收到对自己能力的需求后,每当更新事件发生时发布相应的基本层上下文实例,服务器收到后,经过处理获得应用层上下文实例,发布之。

服务器中的映射器通过映射算法维护需求与能力之间动态的映射关系,如图 5 所示,实现了按需获取。映射算法由相应的更新事件触发执行,如需求或能力的加入和退出、规则的改动等。有些需求不仅包含基本层上下文概念,也包含高级层上下文概念,所以首先需要在合成器的支持下把高级层上下文概念分解为基本上下文概念。因为概念都是语义共享的,所以可以进行同一性判断。算法思路如下:一个需求与一个能力发生关联,当且仅当至少存在一个概念,同时属于该需求和该能力。这样就形成了每个需求的能力列表和每个能力的需求列表。如果某需求至少存在一个概念没有能力包含它,则该需求覆盖失败,等待算法下一次执行重试。本算法并没有充分利用需求和能力的语义、可能关联冗余的能力。改进映射算法是本文的未来工作。

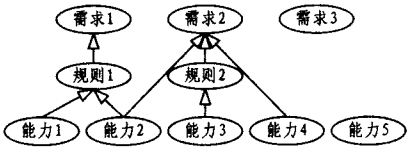

图 5 需求与能力的映射关系

根据映射关系,对于某个基本层上下文实例的更新,只有部分规则和需求是活跃的,也就只需合成部分高级层上下文实例和发布部分应用层上下文实例,这将大大减少处理时间。如图 5 所示,如果"能力 1"发送了新的上下文实例,则只有"规则 1"和"需求 1"需要重新执行。同理"能力 2"更新时,只有"规则 1"、"需求 1"和"需求 2"需要重新执行。"能力 5"不需要发布更新,因为没有需求用到它。"需求 3"总不会执行,因为还没有能力满足它。

5 双库集中共享

上下文知识的共享是指让系统中分布的各类实体都能访问或更新该上下文知识。上下文知识的共享有两个作用:

(1) 部分新知识是在已有知识的基础上定义的,即通过对原全局本体的引用,如基本层上下文概念、基本层上下文实例、高级层上下文概念、应用层上下文概念,如图 3 所示。

(2) 部分新知识是在已有知识的基础上发现的,即通过对全局本体的推理(这里指广义的推理,包含一致性检查、重分类、合成、查询),如高级层上下文实例、应用层上下文实例。

分布式共享方式使得高级层上下文实例的合成只能由应用完成,上下文知识的集中共享有利于提高合成效率,因为:

• 一个高级层上下文实例一般需要多个基本层上下文实例才能得到,这多个基本层上下文实例都要向应用发布,这样就大量消耗了在普适环境中本就不丰富的带宽资源。而中心节点可以合并重复的传输,并只传输合成结果,不传输合成前的数据。

• 应用不一定运行在资源丰富的设备上,可能完全没有能力从事"合成"这种计算量很大的工作。而中心节点可保证丰富的资源和足够的计算能力。

• 合成本身是针对本体的处理,与应用逻辑关系不大,不同应用上的合成实际上是做同样的工作,让每个应用都去实现合成,增加了开发难度和计算负担,应该独立出来作为一个通用的服务。

• 一些高级上下文需要对历史上下文进行合成,集中存储不用担心某个提供者的退出而影响对其历史上下文的使用。

按照上述上下文知识共享的两种不同作用和集中的原

则，DCASI 中包含了双库集中共享机制。服务器维护了共享上下文知识的“文件库”和“内存库”，如图 6 所示。它还可以加入数据库，用于存储和处理历史上下文，这可作为本文的未来工作。

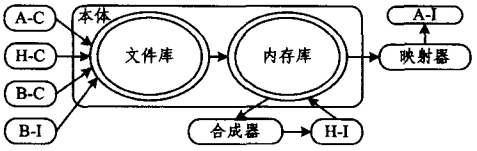

图 6　双库集中共享

文件库对外提供一个唯一的引用地址，指向全局本体，网络上任何地方的实体都能通过这个引用地址导入它。开发人员在定义新的上下文概念时，可以借助工具（如 Protégé[26]）引用并浏览原全局本体中已有的知识，避免重复定义和冲突定义，可见应用是通过服务器来了解智能空间中有哪些可用的感知能力。文件库对外还提供一个唯一的上传地址，让所有以文件形式定义的新的上下文知识都能加入到该文件库，加入时需要向服务器提供指向自身的引用地址，以更新内存库和全局本体（更改一个高级层概念的定义只需要同名文件覆盖即可）。这样文件库就实现了其共享任务。

内存库的上下文知识除了直接来自文件库外，还生成高级层上下文实例，并最终提供应用层上下文实例。内存库的共享任务包括：

（1）一致性检查。一致性检查能发现知识定义冲突，这在新知识分散定义的系统中显得尤其重要，它是所有后续推理运算的基础。冲突解决是本文的未来工作。

（2）重分类。重分类能发现潜在的类层次关系，从而为高级层上下文实例的获取做准备。实际上，重分类也是产生新上下文知识的一种方式。

（3）合成高级层上下文实例。高级层上下文实例不能由本体自身产生，因为本体的大部分能力在于对概念的推理，即（1）和（2）。为了实现对实例的推理，可以采用基于规则的推理，这就是 DCASI 用规则定义高级层上下文概念的原因。

（4）查询应用层上下文实例。整个上下文知识库中，只有应用层上下文实例才是直接有用的，而它只是实例集的子集，DCASI 用查询语句（如 SWRL）定义应用层上下文概念，可按条件提取知识。

文件库和内存库是不能互相替代的，而是互相补充、相互依存的关系，它们分别完成不同的共享任务（定义与发现）。在 iMuseum 系统中实现了双库集中共享机制。采用 HTTP 的超链接作为全局本体的引用地址，采用 FTP 的共享文件夹作为唯一的上传地址。以 Protégé OWL API[26] 管理内存库，其中采用 Racer[27] 对 OWL-DL 推理，包括一致性检查和重分类；采用 Jess[28] 对 SWRL 推理，包括高级层上下文实例的获取和应用层上下文实例的获取。

6　实现

为了验证 DCASI 的有效性，以智能博物馆为应用场景，实现了原型系统 iMuseum，其中包含一个上下文感知应用 iGuide，如图 7 所示。后台计算机上部署的 FTP 服务器、HTTP 服务器、文物内容库和 ActiveMQ[25] 首先启动，接着上下文服务器启动，然后 iGuide 启动，利用 ActiveMQ 向上下文服务器声明自己的上下文需求（本体文件存入 FTP 服务器和 HTTP 服务器）；游客进入展厅，携带事先部署好的 PDA（附带 RFID 阅读器，安装了 RFID 上下文提供者和 iGuide 客户端），PDA 启动后通过无线网络向上下文服务器声明自己的上下文提供能力；服务器发现上下文能力满足了上下文需求，于是向 RFID 上下文提供者订阅上下文的更新；当游客靠近附有 RFID 的文物时，阅读器读到 RFID，便把其标识号和持续时间发给服务器，服务器推理后得知游客是否对文物感兴趣，若是，则把标识号等信息发给 iGuide；iGuide 从文物内容库中取出该文物的多媒体解说，发给游客 PDA 上的 iGuide 客户端；最后游客享受着智能博物馆向他/她主动提供的丰富信息，几乎感觉不到系统的复杂性。

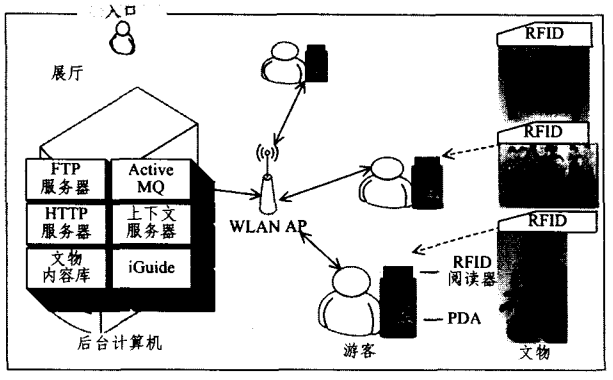

图 7　iMuseum 系统结构

后台计算机是一台配置为 Intel（R）Pentium D CPU 3.0GHz，1.0GB RAM，MS Windows 2000 的 PC。关键技术的实现在以上各节的末尾都已介绍。重点考察服务器维护动态上下文知识的时间与空间性能，试验 20 次取平均值。

上下文服务器的运行可分为 3 个不同阶段，分别计时得：初始化时间为 4969ms；响应注册的时间为 422ms；响应上下文变化的时间为 813ms；响应时间并未随上下文知识的增加而明显延长。初始化时间稍长，但可以接受，因为初始化只在系统启动时执行一次，且对用户来说不可见。响应时间在 1s 以内，完全可以满足实体的动态注册和即时感知环境变化的要求。

设文件库初始大小为 S_{fo}，运行时每加入一个文件，平均增大 S_{fi}；内存库初始大小为 S_{mo}，运行时每加入一个文件，平均增大 S_{mi}；系统中上下文提供者的个数为 N_p，应用的个数为 N_a。由一个提供者提供一个能力文件和一个相应的实例文件，一个应用提供一个需求文件，因此文件库大小为 $S_{fo}+S_{fi}\times(2N_p+N_a)$，内存库大小为 $S_{mo}+S_{mi}\times(2N_p+N_a)$。测得 $S_{fo}=41$kB，$S_{fi}=3.5$kB，$S_{mo}=20$MB，$S_{mi}=1.6$MB。40 个提供者和 10 个应用将占用 356kB 的硬盘空间和 164MB 的内存空间。可见系统规模（N_p 和 N_a）的上限主要取决于后台计算机的可用内存。

结束语　本文提出了动态上下文知识获取与共享框架 DCASI，主要包括一个模型（二集合三层次上下文模型 2×3cm）和两个机制（按需分散获取机制、双库集中共享机制）。原型系统 iMuseum 验证了该框架的可行性。DCASI 的特点包括：1）动态，体现在系统在运行时向全局本体不断加入新的上下文知识；2）专职，体现在不同上下文知识来自不同实体，分散定义，各施其职；3）按需，体现在需求和能力以映射关联，

按需传输和处理上下文知识;4)集中,体现在上下文服务器集中存储、合成、共享、发布上下文知识。

本文框架在其他许多方面仍有待继续研究。未来的工作包括增加存储历史上下文的数据库、上下文知识的冲突解决、改进需求与能力的映射算法等。将个性化推荐[29]引入智能博物馆也是我们的设想之一。

参 考 文 献

[1] Satyanarayanan M. Pervasive computing: Vision and challenges [J]. IEEE Personal Communications, 2001, 8(4): 10-17

[2] 徐光祐,史元春,谢伟凯. 普适计算[J]. 计算机学报, 2003, 26(9): 1042-1050

[3] Rehman K, Stajano F, Coulouris G. An architecture for interactive context-aware applications[J]. IEEE Pervasive Computing, 2007, 6(1): 73-80

[4] Henricksen K, Indulska J. Developing context-aware pervasive computing applications: Models and approach[J]. Pervasive and Mobile Computing, 2006, 2(1): 37-64

[5] Mostefaoui G K, Pasquier-Rocha J, Brezillon P. Context-aware Computing: A Guide for the Pervasive Computing Community [C]// Proc. of IEEE/ACS International Conference on Pervasive Services (ICPS'04). New York, USA: IEEE CS Press, 2004, 39-48

[6] Addlesee M, Curwen R, Hodges S, et al. Implementing a Sentient Computing System[J]. Computer, 2001, 34(8): 50-56

[7] Dey A K, Sohn T. Supporting End User Programming of Context-aware Applications [C]// Proc. of Conference on Human Factors in Computing Systems (CHI'03) Workshop on End User Development. Fort Lauderdale, USA: ACM Press, 2003: 23-26

[8] 王海鹏,周兴社,倪红波,等. 面向智能博物馆的主动式个性化信息服务[C]//第三届和谐人机环境联合学术会议(HHME'07)论文集. 济南:清华大学出版社, 2007

[9] Baldauf M, Dustdar S, Rosenberg F. A survey on context-aware systems[J]. International Journal of Ad Hoc and Ubiquitous Computing, 2007, 2(4): 263-277

[10] 李蕊,李仁发. 上下文感知计算及系统框架综述[J]. 计算机研究与发展, 2007, 44(2): 269-276

[11] Salber D, Dey A K, Abowd G D. The Context Toolkit: Aiding the Development of Context-enabled Applications[C]// Proc. of Conference on Human Factors in Computing Systems (CHI'99). Pittsburgh, USA: ACM Press, 1999: 434-441

[12] Hofer T, Schwinger W, Pichler M, et al. Context-awareness on mobile devices-the hydrogen approach[C]// Proc. of the 36th Annual Hawaii International Conference on System Sciences (HICSS'03). Big Island, USA: IEEE CS Press, 2003: 292-302

[13] Roman M, Hess C, Cerqueira R, et al. Gaia: A Middleware Infrastructure to Enable Active Spaces[J]. IEEE Pervasive Computing, 2002, 4(1): 74-83

[14] Biegel G, Cahill V. A Framework for Developing Mobile, Context-aware Applications[C]// Proc. of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom'04). Orlando, USA: IEEE CS Press, 2004: 361-365

[15] Chen H, Finin T, Joshi A. Using OWL in a Pervasive Computing Broker[C]// Proc. of International Conference on Autonomous Agents and Multi-Agent Systems (AAMAS'03) Workshop on Ontologies in Agent Systems. Melbourne, Australia: ACM Press, 2003: 9-16

[16] Gu T, Wang X, Pung H, et al. A Middleware for Context-Aware Mobile Services[C]// Proc. of IEEE Vehicular Technology Conference (VTC'04). Los Angeles, USA: IEEE Vehicular Technology Society Press, 2004: 2656-2660

[17] 胡鑫喆,王克宏. 觉察上下文计算环境中上下文服务器的设计[J]. 计算机工程, 2005, 31(14): 113-115

[18] Korpipää P, Mäntyjärvi J, Kela J, et al. Managing Context Information in Mobile Devices[J]. IEEE Pervasive Computing, 2003, 2(3): 42-51

[19] Fahy P, Clarke S. CASS-Middleware for Mobile Context-aware Applications[C]// Proc. of the 2nd International Conference on Mobile Systems, Applications, and Services (MobiSys'04) Workshop on Context Awareness. Boston, USA: ACM Press, 2004

[20] Wang X, Zhang D, Gu T, et al. Ontology-based Context Modeling and Reasoning Using OWL[C]// Proc. of the 2nd IEEE Conference on Pervasive Computing and Communications (PerCom'04). Orlando, USA: IEEE CS Press, 2004: 18-22

[21] Schmidt A. A layered model for user context management with controlled aging and imperfection handling[C]// Proc. of the 2nd International Workshop on Modeling and Retrieval of Context (MRC'05). Edinburgh, UK: Springer-Verlag Press, 2005: 86-100

[22] Baader F, Calvanese D, McGuinness D, et al. The description logic handbook: Theory, implementation and applications[M]. Cambridge, UK: Cambridge University Press, 2003

[23] Dean M, Schreiber G, Bechhofer S, et al. OWL Web Ontology Language Reference[EB/OL]. http://www.w3.org/TR/owl-ref, 2004

[24] Horrocks I, Patel-Schneider P F, Boley H, et al. SWRL: A Semantic Web Rule Language Combining OWL and RuleML[EB/OL]. http://www.daml.org/rules/proposal, 2004

[25] ActiveMQ[EB/OL]. http://activemq.apache.org, 2007

[26] Protégé[EB/OL]. http://protege.stanford.edu, 2007

[27] RacerPro[EB/OL]. http://www.racer-systems.com, 2007

[28] Jess[EB/OL]. http://www.jessrules.com/jess, 2007

[29] Yu Z, Zhou X, Zhang D, et al. Supporting Context-aware Media Recommendation for Smart Phones[J]. IEEE Pervasive Computing, 2006, 5(3): 68-75