软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software, 2011,22(Suppl.(2)):105−119

E-mail: jos@iscas.ac.cn
http://www.jos.org.cn
Tel/Fax: +86-10-62562563

普适计算中基于集合的上下文建模和操作*

马 骏[1,2], 曹建农[3], 马 超[3], 陶先平[1,2+], 吕 建[1,2]

[1](计算机软件新技术国家重点实验室(南京大学),江苏 南京 210093)

[2](南京大学 计算机软件研究所,江苏 南京 210093)

[3](香港理工大学 电子计算学系,香港)

Modeling and Manipulating Context in Pervasive Computing Based on Set Theory

MA Jun[1,2], CAO Jian-Nong[3], MA Chao[3], TAO Xian-Ping[1,2+], LÜ Jian[1,2]

[1](State Key Laboratory for Novel Software Technology (Nanjing University), Nanjing 210093, China)

[2](Institute of Computer Software, Nanjing University, Nanjing 210093, China)

[3](Department of Computing, Hongkong Polytechnic University, Hong Kong, China)

+ Corresponding author: E-mail: txp@nju.edu.cn

Ma J, Cao JN, Ma C, Tao XP, Lü J. Modeling and manipulating context in pervasive computing based on set theory. *Journal of Software*, 2011,22(Suppl.(2)):105−119. http://www.jos.org.cn/1000-9825/11031.htm

Abstract: Context awareness is one of the key characteristics of ubiquitous and pervasive computing. Most current formalism works focus on two aspects: context representing and system modeling. However, how the temporal property of context can be modeled and how context can be manipulated are not well addressed. This paper proposes a formal way based on set theory for modeling and manipulating context, in which a context is defined as a set of context entries and operators are introduced to specify how to manipulate contexts in according to the needs of different applications. To show the usability of the proposed model, a demo implementation is also included in this paper.

Key words: pervasive computing; context; set theory; context aware computing; context modeling; algorithm analysis

摘 要: 上下文感知是普适计算的最主要特征之一.现今关于上下文相关的形式化工作主要集中在两个方面:上下文表示以及系统行为建模;而在关于如何刻画上下文时间特性以及如何对上下文进行操作这方面的形式化工作却很少.提出一套基于集合的用于描述上下文及其操作的形式化模型.在该模型中,上下文被定义为一系列上下文条目(context entry)所组成的集合;在此基础之上,引入一系列上下文操作(operation),通过对这些操作的组合,可以依据应用的不同需求来描述相应的上下文处理流程和逻辑.此外,还给出了一个参考实现,以验证该模型的可用性.

关键词: 普适计算;上下文;集合论;上下文感知计算;上下文建模;算法分析

* 基金项目: 国家自然科学基金(61073031, 61021062); 国家重点基础研究发展计划(973)(2009CB320702); 香港研究资助基金委员会资助项目(PolyU5106/10E)

收稿时间: 2011-07-20; 定稿时间: 2011-12-01

随着技术的不断进步和成熟、网络的不断普及、设备的推陈出新,人们工作和生活的各个场所(家庭、办公室、医院、商场等)都充满各式各样的可见或不可见的计算设备.这些设备通过网络互联,相互沟通、协同为用户提供服务.普适计算是“信息空间与物理空间的融合,在这个融合的空间中人们可以随时随地、透明地获得数字化的服务”[1].上下文感知(context-awareness)[2]是普适计算系统的最为关键的特征之一,它要求系统能够探知用户、环境以及自身的状态(即上下文),并以此为依据调整自身的行为,从而提供更好的服务.形式化系统为描述普适计算中上下文感知特性起着极为重要的基础作用.它将有利于我们掌握上下文的本质、描述上下文感知系统的基本行为.现如今关于上下文感知的形式化工作可以大致划分为两大类:系统行为建模(system modeling)和上下文表示(context representing).其中,系统行为建模关注于“如何对上下文感知系统的各个构件的行为及构件之间的交互进行建模”;上下文表示则主要关注于“如何表示上下文并让计算机有效的理解上下文”.

本文的工作属于上下文表示(context representing)的工作.与现有上下文表示的形式化工作不同,我们尤其关注两个方面的内容:(1) 如何刻画上下文的时间特性.一个普适计算系统所使用的上下文信息往往会随着时间的迁移而变化,不同的时刻具有不同的取值,使得上下文信息之间存在与时间相关的各种联系(例如,先后关系、包含关系、重叠关系等);不仅如此,过去的历史信息可能影响系统当前的决策和行为[3].因此,如何有效地刻画上下文的时间特性以及这些时间相关的关系是我们上下文建模所要解决的问题.(2) 如何对上下文的操作或处理过程进行描述.现有工作往往关注与上下文表示本身,而对上下文的处理则是由应用开发人员以 ad-hoc 的方式来进行的.我们认为对上下文的处理可以通过一系列简单的操作及其组合而完成.

基于上述考虑,我们基于集合论提出了一套描述上下文及其操作的形式化模型.在该模型中,上下文定义为一系列上下文条目(context entry)所构成的集合,每个上下文条目则刻画了某个特定时间段(或时间点),系统中某个实体(entity)的某个状态维(state dimension)的信息.此外,我们对原有集合基本操作进行了修改和扩展,利用这些操作可以描述对上下文的处理逻辑.

本文的主要贡献在于:

- 基于集合给出了一个描述上下文及其操作的形式化模型.
- 引入时间描述机制(即 duration)并提供了相关操作,描述上下文的时间特性以及不同上下文信息之间的与时间相关的各种关系.
- 通过上下文操作的组合,可以依据应用需求定义多种不同的上下文处理逻辑.

本文第 1 节简单介绍一些相关的现有工作.第 2 节详细介绍我们所提出的基于集合的上下文模型,包括duration,context 等基本概念以及定义在其上的各种操作.第 3 节给出该上下文模型的一个参考实现,并基于该实现对关键的方法进行分析.第 4 节对本文进行总结并对今后工作进行展望.

1 相关工作

现有上下文感知的形式化基础的研究工作可以大致划分为两类:系统行为建模(system modeling)和上下文表示(context representing).

以系统行为建模(system modeling)为目标的形式化系统往往是通过刻画系统的主要构件(component)、构件的行为(behavior)以及构件之间的通信(communication)来为整个应用系统建模.例如,CAC[4],CONAWA[5]和CCA[6]都是用于刻画上下文感知系统的进程代数系统(process calculus).它们在 Mobile Ambient[7]基础上进行扩展,引入上下文的概念,从而支持上下文感知系统的建模.文献[8]则通过将上下文引入经典 action system 来描述整个上下文感知系统的行为.

上下文表示(context representing)主要关注于“如何表示上下文,如何让计算机理解上下文”.通常这部分工作都是从其他领域(例如,人工智能、语义网等)借鉴而来的.本体(ontology)[9]是使用的最多的上下文模型[9].例如,SOCAM[10]、CoBrA[11]都利用本体来表示上下文.文献[12]则提出了一个采用 OWL 语言描述的基于本体的形式化的动态上下文模型,以及一个上下文融合和消费机制的实现.GAIA[13]则利用一阶谓词来表示上下文,并利用本体来描述不同谓词之间的结构.

时间是上下文重要特性,文献[2]就将位置(location)、身份(identity)、行为(activity)以及时间(time)视为最为基础的4类上下文.现有上下文表示(context representing)(例如,用RDF描述的基于本体上下文模型)则缺乏对上下文时间特性的刻画.针对本体模型的这一不足,文献[14]将上下文划分为静态(static)上下文和动态(dynamic)上下文.静态上下文直接用 RDF 三元组描述,动态上下文则是通过在 RDF 三元组基础之上添加时间戳(start_time,update_time)和 *ttl* 来刻画上下文信息的时间特性,从而我们可以得知一个具体的 RDF 三元组何时创建、更新以及有效期是多少;文献[12]则是通过对系统的本体定义进行扩展,引入 *TemporalEntity* 以及 *DynamicEntity* 来刻画上下文的时间特性以及不同动作(action)以及事件(event)之间的时间关系.而在 GAIA[13]中,时间因素和其他上下文信息一样是通过谓词进行描述的(例如,“Time(New York,“<”,12:00 01/01/01)”),通过“逻辑与”(∧)与其他上下文谓词进行联系来刻画其时间特性.尽管这些模型能够在一定程度上刻画上下文的时间特性,但是它们都缺乏对这一时间特性统一的操作和处理.

本文的工作属于上下文表示(context representing),但与上述形式化模型以及扩展仅仅关注如何刻画上下文(及其时间特性)不同,本文更关注如何对统一刻画的上下文以及时间特性进行操作和处理.具体而言,我们从集合论的视角来看待上下文,将上下文视为一系列上下文条目(context entry)所构成的集合,每个上下文条目则刻画了某个特定时间段(或时间点),系统中某个实体(entity)的某个状态维(state dimension)的信息.为了刻画上下文的时间特性,我们借鉴了文献[15]中的 time interval 的概念**,利用时间段来刻画系统的时间特性(本文称其为 duration).不过与文献[15]基于 time interval 建立一套逻辑系统不同,我们更关注于如何对一系列 duration 进行操作处理,定义了一系列的基本操作;此外,通过将 duration 引入上下文模型,我们可以描述上下文的时间特性,描述不同上下文信息之间的时间关系;最后,我们还进一步对原有集合基本操作进行了修改和扩展,,通过这些操作及其组合可以刻画对上下文的处理逻辑.

2 基于集合的上下文模型和操作

一个普适计算系统往往包含许多不同的实体,每个实体都具有自己的属性或状态,这些属性或状态的取值均可成为计算系统所使用的上下文信息.然而,这些属性和状态通常不是一成不变的,它们往往会随着时间的迁移而相应变化,不同的时刻具有不同的取值.这种随时间动态变化的特性,使得不同上下文信息之间存在与时间相关的各种联系:两个上下文在时间上可能具有先后(before/after)、包含(within/contains)、重叠(overlap)等一系列的关系.为了刻画上下文这一时间特性,在我们的上下文模型中引入了 duration 的概念来刻画一段连续的时间或一个时间点,进一步刻画上下文与时间相关的各种关系.

本节主要包括两个方面的内容:首先我们介绍 duration 及其相关操作,然后介绍我们的 context 模型及其相关操作.

2.1 Duration

定义 1(duration). 一个 duration 代表一段连续的时间或一个时间点,表示为实数域上的一段区间,即 duration 是由实数轴上的两(端)点间的一切实数(可能包括一个或两个端点所对应实数)所组成的非空集合(记为 $[b,e]$,$[b,e)$,$(b,e]$ 或 (b,e), 当 $b=e$ 时,$[b,e]$ 表示一个具体的时间点).特别地,我们用 $d^U=(-\infty,+\infty)$ 来表示一个特殊的 duration (universal duration),它对应整个实数集合 $\mathbb{R}$.

任意给定两个 duration: d_1,d_2, 我们可以进一步定义它们之间的关系见表 1***.

进一步地,给定一个 duration 的集合 D 及其两个元素: d_1,d_2, 我们说 d_1 和 d_2 在 D 中是互连的(connected,并

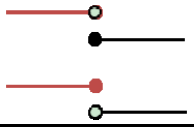
** Allen 在文献[15]中提出了基于 time intervals 的时序逻辑系统.该系统利用 time intervals 来刻画时间信息,定义了 13 种 time intervals 之间的关系,并在此基础上给出了一套逻辑系统来描述和验证一系列事件(event)是否满足一定的时间约束.

*** Directly connected 的示意图中仅给出了 $\big((d_1 \rhd d_2)\wedge\exists d'(d_1 \rhd d' \rhd d_2)\big)\vee\big((d_2 \rhd d_1)\wedge\exists d'(d_2 \rhd d' \rhd d_1)\big)$ 的情况,即 Allen 在文献[15]中所指的 meets 或 met 关系,更多 duration 之间的关系可参见文献[15].

用 $d_1\blacklozenge^* d_2$ 表示)当且仅当(1) $d_1\blacklozenge d_2$ 或者(2) $\exists d_3\in D$ 使得 $d_1\blacklozenge^* d_3$ 且 $d_2\blacklozenge^* d_3$.

Table 1 Some relations between durations

表 1 部分 duration 的关系

关系名称	符号表示	描述	示意图
equals	$d_1=d_2$ $(d_2=d_1)$	$\forall r, r\in d_1\Leftrightarrow r\in d_2$	
before (*after*)	$d_1\rhd d_2$ $(d_2\lhd d_1)$	$\forall r\in d_1,\forall r'\in d_2, r<r'$	
within (*contains*)	$d_1\subset d_2$ $(d_2\supset d_1)$	$(\forall r, r\in d_1\Rightarrow r\in d_2)\wedge$ $(\exists r'\in d_2, r'\notin d_1)$	
pre-overlap (*suc-overlap*)	$d_1\vec{\Diamond}d_2$ $(d_2\vec{\Diamond}d_1)$	$(\exists r, r\in d_1, r\in d_2)\wedge$ $(\exists d'\subset d_1, d'\rhd d_2)\wedge$ $(\exists r', r'\in d_2, r'\notin d_1)$	
directly connected	$d_1\blacklozenge d_2$ $(d_2\blacklozenge d_1)$	$(d_1\not\rhd d_2\wedge d_2\not\rhd d_1)\vee$ $((d_1\rhd d_2)\wedge\nexists d'(d_1\rhd d'\rhd d_2))\vee$ $((d_2\rhd d_1)\wedge\nexists d'(d_2\rhd d'\rhd d_1))$	

给定一个实数集合 $R\subset\mathbb{R}$, 以及 R 中的任意一对实数 r_1, r_2, 我们称 r_1, r_2 在 R 中是互连的(r_1 connects_with r_2),当且仅当 $\forall r', r_1<r'<r_2\Rightarrow r'\in R$. 显然, connects_with 关系是自反的、对称的、传递的,因而它是一个定义在 R 的等价关系;从而我们可以进一步依据 connects_with 这一等价关系获得 R 的一个划分.该划分的任意一个元素都是一段实数区间所包含实数(或一个具体的实数)所构成的集合,即为一个 duration;因此,我们称该划分为"R 所覆盖的 duration 集合",并记为 $dur(R)$. 依据划分的概念,我们可以得知对于 $dur(R)\left(\left|dur\left(R\right)\right|>0\right)$ 任意两个元素 d_1, d_2, 如果 $d_1\blacklozenge d_2$ 则必然有 $d_1=d_2$.

让 $\mathbb{D}$ 表示所有 duration 所构成的集合,我们进一步定义一系列形如 $\mathbb{D}\times\mathbb{D}\to 2^{\mathbb{D}}$ 的函数(其中,−,∩,∪为集合的差集、交集、并集操作):

$$d_1\ominus d_2\doteq dur(R),\ \text{where } R\doteq d_1-d_2=\{r\mid r\in d_1\wedge r\notin d_2\} \tag{1}$$

$$d_1\otimes d_2\doteq dur(R),\ \text{where } R\doteq d_1\cap d_2=\{r\mid r\in d_1\wedge r\in d_2\} \tag{2}$$

$$d_1\oplus d_2\doteq dur(R),\ \text{where } R\doteq d_1\cup d_2=\{r\mid r\in d_1\vee r\in d_2\} \tag{3}$$

依据以上定义, $\forall d\neq d^U$, 有 $d\subset d^U, d\ominus d^U=\varnothing, d\otimes d^U=\{d\}, d\oplus d^U=\{d^U\}$. 如果 $d_1=[3,5], d_2=[0,3]$, 则有 $d_1\vec{\Diamond}d_2, d_1\blacklozenge d_2, d_1\ominus d_2=\{(3,5]\}, d_1\otimes d_2=\{[3,3]\}, d_1\oplus d_2=\{[0,5]\}$.

定义 2(规范 Duration 集合(normal formed duration set,简称 NFDS)). 一个 duration 集合 D 被称为"规范的(normal formed)"当且仅当(1) 其元素个数小于 2(即 $|D|<2$);或(2) 当 $|D|\geqslant 2$ 时,其任意两个不同元素彼此互不相连,即 $\forall d_1, d_2\in D\left(d_1\neq d_2\Rightarrow\neg(d_1\blacklozenge d_2)\right)$. 例如,空集∅、单元素集合 $\{d\}$ 以及 $\{[1,3),[5,7]\}$ 都是 NFDS,而 $\{[1,4],[3,6]\}$ 则不是 NFDS.

任意给定一个 duration 集合 D ,我们用 $\Re(D)$ 表示 D 中所有 duration 元素所包含的实数所构成的集合(即 $\Re(D)\doteq\bigcup_{d\in D}d=\{r\mid r\in d, d\in D\}$),并称其为 D 的覆盖(coverage).进一步地,我们可以通过 $dur\left(\Re(D)\right)$ 构建一个 NFDS,并称其为"D 的范型(normal form)"(记为 $NF\left(D\right)$. 依据 $dur(R)$ 的定义,任意一个 duration 集合 D 的标准形式 $NF(D)$即为 $\Re(D)$ 依据 connects_with 关系的划分),显然, $\Re(D)=\Re\left(NF\left(D\right)\right)$.

类似于公式(1)(2)(3),给定任意两个 duration 集合 $D_1, D_2\in 2^{\mathbb{D}}$, 我们也能定义形如 $2^{\mathbb{D}}\times 2^{\mathbb{D}}\to 2^{\mathbb{D}}$ 的操作如下:

$$D_1\ominus D_2\doteq dur(R),\ \text{where } R\doteq\Re(D_1)-\Re(D_2)=\left\{r\mid r\in\Re(D_1)\wedge r\notin\Re(D_2)\right\} \tag{4}$$

$$D_1\otimes D_2\doteq dur(R),\ \text{where } R\doteq\Re(D_1)\cap\Re(D_2)=\left\{r\mid r\in\Re(D_1)\wedge r\in\Re(D_2)\right\} \tag{5}$$

$$D_1\oplus D_2\doteq dur(R),\ \text{where } R\doteq\Re(D_1)\cup\Re(D_2)=\left\{r\mid r\in\Re(D_1)\vee r\in\Re(D_2)\right\} \tag{6}$$

通过 $dur(R)$的定义可知,上述 3 个操作对于 NDFS 是封闭的.假设 $D_1=\{[0,3],[5,9]\}, D_2=\{(2,6),[7,11]\}$, 则 $D_1\ominus D_2=\{[0,2],[6,7)\}, D_2\ominus D_1=\{(3,5),(9,11]\}, D_1\oplus D_2=D_2\oplus D_1=\{[0,11]\}, D_1\otimes D_2=D_2\otimes D_1=\{(2,3],[5,6),[7,9]\}$.

2.2 上下文(context)

通常情况下,一个普适计算系统往往由多个相互交互的实体(entity)所构成(例如,各种计算设备以及用户).我们用 e_i 来表示一个具体的实体,并用 $\mathbb{E}$ 表示系统中所有实体的集合.

一个实体 e_i 可以具有多个状态维(state dimension,简称 sd),这些状态维的值共同刻画了该实体的当前状态.例如,身高、体重、朋友等都可以视作一个用户的状态维. $SD(e_i)$ 表示实体 e_i 所有状态维的集合,并用 $\mathbb{S}$ 表示一个普适计算系统中所有实体的状态维的集合,即

$$\mathbb{S}\doteq\bigcup_{e_i\in\mathbb{E}}SD(e_i) \tag{7}$$

给定任意 $sd_k^i\in SD(e_i)$, 我们用 $Dom(sd_k^i)$ 来表示 sd_k^i 的取值范围,并进一步用 $\mathbb{V}$ 表示一个系统中所有实体的状态维的取值范围的所构成总集合,即

$$\mathbb{V}\doteq\bigcup_{sd_k^i\in SD(e_i),e_i\in\mathbb{E}}Dom(sd_k^i) \tag{8}$$

定义 3(上下文陈述(context statement)). 一个上下文陈述是一个形如 $\langle e,sd,v\rangle$ 的三元组,其中, $e\in\mathbb{E}$, $sd\in SD(e), v\in Dom(sd)$. 所有上下文陈述的集合用 $\mathbb{Q}$ 表示.

一个上下文陈述 $\langle e,sd,v\rangle$ 表示实体 e 的 sd 维度的取值为 v. 例如, $\langle$Bob,Height,193cm$\rangle$ 表示“Bob 的身高为 193cm”.

有了上下文陈述以及上一节中 Duration 的定义,我们可以最终定义上下文(context)如下:

定义 4(上下文(context)). 一个上下文 C 是关系 $\mathbb{Q}\times\mathbb{D}$ 的子集(即 $C\subseteq\mathbb{Q}\times\mathbb{D}$),而 $\mathbb{C}$ 则表示所有上下文的集合.

上下文中每一个元素形如$\langle cs,d\rangle$的元组,表示 cs 所表示的上下文陈述在 d 所代表的这段时间内是成立(hold)的.我们称元组$\langle cs,d\rangle$为上下文条目(context entry,简称条目,用 ce 表示),并用 $\mathbb{K}$ 表示所有上下文条目所构成的集合.为了方便,在余下行文中我们用 $ce.e$ (以及 $ce.sd, ce.v$)来表示 $ce.sd.e$ (以及 $ce.cs.sd, ce.cs.v$).

此外,给定一个上下文 C, 我们引入两个简单操作: $\kappa(C)$ 和 $\delta(C,cs)$, 使得 $\kappa(C)=\{ce.cs\mid ce\in C\}, \delta(C,cs)=\{ce.d\mid ce\in C\wedge ce.cs=cs\}$.

例 1: 令 $C_1=\{ce_1,ce_2,ce_3\}, ce_1=\langle\langle e_1,sd_1,v_1\rangle,[8,10]\rangle, ce_2=\langle\langle e_2,sd_2,v_2\rangle,[9,10)\rangle, ce_3=\langle\langle e_1,sd_1,v_1\rangle,[9,11]\rangle$ 时,我们有 $\kappa(C_1)=\{\langle e_1,sd_1,v_1\rangle,\langle e_2,sd_2,v_2\rangle\}, \delta(C_1,\langle e_1,sd_1,v_1\rangle)=\{[8,10],[9,11]\}, \delta(C_1,\langle e_2,sd_2,v_2\rangle)=\{[9,10)\}$.

给定两个规范上下文 C_1, C_2 ,我们定义 $C_1=C_2$ 当且仅当 $\forall ce, ce\in C_1\Leftrightarrow ce\in C_2$; $C_1\subset C_2$ 当且仅当 $C_1\neq C_2$ 且 $\forall ce\in C_1,\exists ce'\in C_2$, 使得 $ce.cs=ce'.cs\wedge ce.d\subseteq ce'.d$; 我们进一步用 $C\subseteq C'$ 表示 $C\subset C'$ 或 $C=C'$, 并称“ C 为 C' 的子上下文(sub-context)”,而“ C' 则为 C 的超上下文(sup-context)”.

基于上一节给出的 NFDS 的定义,我们在此也给出一个规范上下文的定义如下:

定义 5(规范上下文(normal-formed context,简称 NFC))一个上下文 C 是规范的(normal-formed)当且仅当对于任意 $cs\in\kappa(C)$, 我们有 $D=\delta(C,cs)$ 是一个 NFDS;否则称其为非规范(non-normal-formed)上下文.

例如,在例 1 中,由于 $\delta(C_1,\langle e_1,sd_1,v_1\rangle)=\{[8,10],[9,11]\}$ 不是 NFDS,因而 C_1 不是一个规范上下文.

任意给定一个上下文 C, 我们定义“ C 的规范型”为满足 $C\subseteq C'$ 的最小规范上下文 C', 并进一步用 $NF(C)$ 表示.实际上, $NF(C)=\bigcup_{cs\in\kappa(C)}(\{cs\}\times NF(\delta(C,cs)))$.

依据此定义,我们可以得到例 1 中 C_1 的规范型 $NF(C_1)=\{\langle\langle e_1,sd_1,v_1\rangle,[8,11]\rangle,\langle\langle e_2,sd_2,v_2\rangle,[9,10)\rangle\}$.

2.3 上下文操作(context operations)

第 2.2 节介绍了我们基于集合的上下文(context)的抽象结构,本小节我们进一步介绍基于该上下文结构的基本操作(operation)及其语义.由于每个 Context 都可以规范化,我们只考虑对规范上下文的操作.

- **并操作——union of contexts($C_1 \hat{\cup} C_2$)**

两个上下文的“并(union)”为它们的最小公共(规范)超上下文,即包括这两个上下文所有条目(entry)的上下文的规范形式:

$$C_1 \hat{\cup} C_2 \doteq NF(C), \text{ where } C = \{ce \mid ce \in C_1 \vee ce \in C_2\} \tag{9}$$

例 2:针对例 1 中的 C_1, 令 $C_2 = NF(C_1)$, $C_3 = \{\langle\langle e_1, sd_1, v_1\rangle, [1,10]\rangle, \langle\langle e_2, sd_2, v_2\rangle, [10,12]\rangle\}$, 则

$$C_2 \hat{\cup} C_3 = \{\langle\langle e_1, sd_1, v_1\rangle, [1,11]\rangle, \langle\langle e_2, sd_2, v_2\rangle, [9,12]\rangle\}.$$

- **交操作——intersection of contexts($C_1 \hat{\cap} C_2$)**

与“并”相对,两个上下文的“交”是指满足 $C \subseteq C_1 \wedge C \subseteq C_2$ 的最大规范上下文 C. 此处,“最大”是指如果存在另一个满足 $C' \subseteq C_1 \wedge C' \subseteq C_2$ 的规范上下文,则一定有 $C' \subseteq C$. 利用交操作我们可以获取两个上下文所共享的公共上下文.

$$C_1 \hat{\cap} C_2 \bigcup_{cs \in \kappa(C_1) \cap \kappa(C_2)} (\{cs\} \times (\delta(C_1, cs) \otimes \delta(C_2, cs))) \tag{10}$$

依据此定义,例 2 中 C_2, C_3 的交 $C_2 \hat{\cap} C_3 = \{\langle\langle e_1, sd_1, v_1\rangle, [8,10]\rangle\}$.

- **差操作——difference of contexts($C_1 \hat{-} C_2$)**

$C_1 \hat{-} C_2$ 为满足 $C \subseteq C_1 \wedge C \hat{\cap} C_2 = \varnothing$ 的最大规范上下文 C, 即

$$\left.\begin{array}{c} C_1 \hat{-} C_2 \doteq C' \cup C'' \\ C' \doteq \{ce \mid ce \in C_1, ce.cs \in (\kappa(C_1) - \kappa(C_2))\} \\ C'' \doteq \bigcup_{cs \in \kappa(C_1) \cap \kappa(C_2)} (\{cs\} \times (\delta(C_1, cs) \ominus \delta(C_2, cs))) \end{array}\right\} \tag{11}$$

依据此定义,针对例 2 中 C_2, C_3, 我们可以得到: $C_2 \hat{-} C_3 = \{\langle\langle e_1, sd_1, v_1\rangle, (10,11]\rangle, \langle\langle e_2, sd_2, v_2\rangle, [9,10)\rangle\}$, $C_3 \hat{-} C_2 = \{\langle\langle e_1, sd_1, v_1\rangle, [1,8)\rangle, \langle\langle e_2, sd_2, v_2\rangle, [10,12]\rangle\}$.

- **更新操作——update of context($C_1 \hat{\Uparrow} C_2$)**

$C_1 \hat{\Uparrow} C_2$ 利用上下文 C_2 中的条目去更新另上下文 C_1 中的条目,具体含义为

$$C_1 \hat{\Uparrow} C_2 \doteq C_2 \hat{\cup} \left(C_1 \hat{-} \bigcup_{cs \in \kappa(C_1)} sub(C_2, cs) \right), \text{ where } sub(C, cs) \doteq \{\langle cs, ce.d\rangle \mid cs.e = ce.e, cs.sd = ce.sd, ce \in C\} \tag{12}$$

例 3:令 $C_4 = \{\langle\langle e_1, sd_1, v_3\rangle, [9,14]\rangle\}$, 针对例 2 中的 C_2, C_3, 有:

$$C_2 \hat{\Uparrow} C_4 = \{\langle\langle e_1, sd_1, v_1\rangle, [8,9)\rangle, \langle\langle e_1, sd_1, v_3\rangle, [9,14]\rangle, \langle\langle e_2, sd_2, v_2\rangle, [9,10)\rangle\},$$

$$C_3 \hat{\Uparrow} C_4 = \{\langle\langle e_1, sd_1, v_1\rangle, [1,9)\rangle, \langle\langle e_1, sd_1, v_3\rangle, [9,14]\rangle, \langle\langle e_2, sd_2, v_2\rangle, [10,12]\rangle\}.$$

- **投影操作——projection of context($C \downarrow p$)**

为了给出投影操作的语义,我们首先需要介绍上下文模式(context pattern,简称模式(pattern)).

定义 6(上下文模式(context pattern)). 一个上下文模式 p, 可以通过一个四元组 $\langle \Pi_{\mathbb{E}}, \Pi_{\mathbb{S}}, \Pi_{\mathbb{V}}, \Pi_{\mathbb{D}} \rangle$ 来定义,其中, $\Pi_{\mathbb{E}} \subseteq \mathbb{E}, \Pi_{\mathbb{S}} \subseteq \mathbb{S}, \Pi_{\mathbb{V}} \subseteq \mathbb{V}, \Pi_{\mathbb{D}} \subseteq \mathbb{D}$, 而且 $\Pi_{\mathbb{D}}$ 是一个 NFDS.

给定一个上下文条目 ce 和一个模式 p, 我们称 ce 匹配(match)模式 p, 当且仅当 $ce.e \in p.\Pi_{\mathbb{E}} \wedge ce.sd \in p.\Pi_{\mathbb{S}} \wedge ce.v \in p.\Pi_{\mathbb{V}} \wedge (\exists d \in p.\Pi_{\mathbb{D}}, ce.d \subseteq d)$.

从而上下文 C 在模式 p 上的投影($C \downarrow p$)则是找到满足条件 $C' \subseteq C \wedge \forall ce' (ce' \in C' \Rightarrow ce' \text{ match } p)$ 的上下文中“最大”的一个标准型的 C'. 此处所谓“最大”是指,如果还有其他标准型上下文 C'' 满足条件: $C'' \subseteq C \wedge \forall ce' (ce' \in C'' \Rightarrow ce' \text{ match } p)$, 则一定有 $C'' \subseteq C'$.

$C \downarrow p$ 的具体含义可以表示为

$$C\downarrow p \doteq \bigcup_{\substack{cs\in\kappa(C),cs.e\in p.\Pi_{\mathbb{E}},\\ cs.sd\in p.\Pi_{\mathbb{S}},cs.v\in p.\Pi_{\mathbb{V}}}} (\{cs\}\times(\delta(C_1,cs)\otimes\Pi_{\mathbb{D}})) \tag{13}$$

例 4:令 $p_1=\langle\Pi_{\mathbb{E}},\Pi_{\mathbb{S}},\Pi_{\mathbb{V}},\Pi_{\mathbb{D}}\rangle,\Pi_{\mathbb{E}}=\{e_1,e_2\},\Pi_{\mathbb{S}}=\{sd_1\},\Pi_{\mathbb{V}}=\{v_1\},\Pi_{\mathbb{D}}=\langle[8,9],[10,11)\rangle$,针对例 2 中的 C_2,C_3,有:

$$C_2\downarrow p_1=\{\langle\langle e_1,sd_1,v_1\rangle,[8,9]\rangle,\langle\langle e_1,sd_1,v_1\rangle,[10,11)\rangle\},\ C_3\downarrow p_1=\{\langle\langle e_1,sd_1,v_1\rangle,[8,9]\rangle,\langle\langle e_1,sd_1,v_1\rangle,[10,10]\rangle\}.$$

- **推导操作——derivation of context($C\circ r$)**

引进推导操作的目的是为了描述从当前可用的上下文推导新的上下文的过程.为了描述推导操作,我们先介绍两个相关的概念:Box 和推导规则(derivation rule).

我们借鉴文献[16]关于 Box 的定义,并进一步扩展,给出我们的定义如下:

定义 7(Box). 给定一个上下文 C,并让 $\beta=[sd_1,sd_2,...,sd_k],sd_i\in\mathbb{S}$ 表示一个 k 维的状态维向量,让 $\varphi(ce_1,ce_2,...,ce_k)$ 表示一个定义在 $\mathbb{K}$ 之上的 k 元谓词,则 C 基于 β/φ 的 Box 为一组定义在 $\mathbb{K}$ 之上的 k 维的向量所构成的集合,其每一个元素(定义在 $\mathbb{K}$ 之上的 k 维的向量)必须满足 φ 所定义的条件.具体表示如下:

$$Box\begin{bmatrix}C\\ \beta/\varphi\end{bmatrix}\doteq\left\{\overrightarrow{C'}\,\middle|\,\begin{array}{l}\overrightarrow{C'}=[ce_1,ce_2,...,ce_k],\ \text{where}\\ ce_i\in C,\ ce_i.sd=\beta[i],\ \varphi(ce_1,ce_2,...,ce_k)=true,\ i=1...k\end{array}\right\} \tag{14}$$

例 5:令 $C_5=\left\{\begin{array}{l}\langle\langle e_1,sd_1,v_1\rangle,[1,2]\rangle,\langle\langle e_1,sd_1,v_1\rangle,[4,5]\rangle,\\ \langle\langle e_2,sd_1,v_1\rangle,(1,3]\rangle,\langle\langle e_2,sd_1,v_2\rangle,(1,2]\rangle,\\ \langle\langle e_2,sd_1,v_2\rangle,[3,4]\rangle,\langle\langle e_1,sd_1,v_2\rangle,[0,1]\rangle\end{array}\right\}$, $\beta=[sd_1,sd_1]$, $\varphi(ce_1,ce_2)\doteq(ce_1.e\neq ce_2.e)\wedge(ce_1.v=ce_2.v)\wedge(ce_1.d\otimes ce_2.d\neq\varnothing)$, 则 $Box\begin{bmatrix}C_5\\ \beta/\varphi\end{bmatrix}=\{[\langle\langle e_1,sd_1,v_1\rangle,[1,2]\rangle,\langle\langle e_2,sd_1,v_1\rangle,(1,3]\rangle],[\langle\langle e_2,sd_1,v_1\rangle,(1,3]\rangle,\langle\langle e_1,sd_1,v_1\rangle,[1,2]\rangle]\}$(不难发现, $Box\begin{bmatrix}C_5\\ \beta/\varphi\end{bmatrix}$ 中的两个向量是对称的,这是由于此处β与φ的定义都是对称的所致).

定义 8(推导规则(derivation rule)). 一条推导规则 r, 可以表示为一个 6 元组 $r\langle\beta,\varphi,sd,f_e,f_v,f_d\rangle$, 此处, β,φ 与定义 7 中的定义一样, $sd\in\mathbb{S},f_e,f_v,f_d$ 则分别是形如 $\mathbb{K}^k\to\mathbb{E},\mathbb{K}^k\to\mathbb{V},\mathbb{K}^k\to 2^{\mathbb{D}}$ 的映射.

于是,在上下文 C 上应用规则 r 的具体含义可表示为

$$C\circ r\doteq NF(C'),\ \text{where}\ C'\doteq\left\{ce\,\middle|\,\begin{array}{l}ce.e=r.f_e\left(\overrightarrow{C^*}[1],\overrightarrow{C^*}[2],...,\overrightarrow{C^*}[k]\right),\\ ce.sd=r.sd,\\ ce.v=r.f_v\left(\overrightarrow{C^*}[1],\overrightarrow{C^*}[2],...,\overrightarrow{C^*}[k]\right),\\ ce.d\in r.f_d\left(\overrightarrow{C^*}[1],\overrightarrow{C^*}[2],...,\overrightarrow{C^*}[k]\right),\\ \overrightarrow{C^*}\in Box\begin{bmatrix}C\\ r.\beta/r.\varphi\end{bmatrix}\end{array}\right\} \tag{15}$$

具体而言, $C\circ r$ 首先依据 $r.\beta$ 以及 $r.\varphi$ 得到 $Box\begin{bmatrix}C\\ r.\beta/r.\varphi\end{bmatrix}$; 然后对这个 Box 中的每个向量 $\overrightarrow{C^*}$, 利用 $r.sd,r.f_e(),r.f_v(),r.f_d()$ 进行运算得到相应的推理结果.

例 6:令 $r\triangleq\langle\beta,\varphi,sd,f_e,f_v,f_d\rangle$ 其中, β,φ 与例 5 中定义相同, $sd=sd_3,f_e(ce_1,ce_2)=ce_1.e,f_v(ce_1,ce_2)=v_4,f_d(ce_1,ce_2)=ce_1.d\otimes ce_2.d$,则在例 5 定义的 C_5 上运用规则 r 可以得到:

$$C_5' \doteq \left\{ ce \;\middle|\; \begin{array}{l} ce.e = r.f_e\left(\overrightarrow{C^*}[1],\overrightarrow{C^*}[2]\right), \\ ce.sd = r.sd, \\ ce.v = r.f_v\left(\overrightarrow{C^*}[1],\overrightarrow{C^*}[2]\right), \\ ce.d \in r.f_d\left(\overrightarrow{C^*}[1],\overrightarrow{C^*}[2]\right), \\ \overrightarrow{C^*} \in Box\begin{bmatrix} C_5 \\ r.\beta / r.\varphi \end{bmatrix} \end{array} \right\} = \left\{ \begin{array}{l} \left\langle \left\langle ce_1.e, r.sd, v_4 \right\rangle, (1,2] \right\rangle, \\ \left\langle \left\langle ce_3.e, r.sd, v_4 \right\rangle, (1,2] \right\rangle \end{array} \right\},$$

$$C_5 \circ r \doteq NF(C_5') = C_5'.$$

2.4 示　例

本节我们通过一个简单的例子来展示如何利用本文所提供的模型来刻画和操作上下文.我们想描述“Bob与 Alice 9:00am–10:00am 在 roomA 开会”这样一个简单的场景.

如图 1 所示,假设存在一个定位服务能够提供 Bob 与 Alice 的位置信息,通过该服务我们获取位置上下文并标记为 C_L;其次,假设 roomA 具有一个探测房间声音强度的服务,通过该服务可以获取 roomA 的声强上下文并标记为 C_S;此外假定还有标明各个房间类型的上下文 C_M,其元素都是稳定的上下文条目(即 $ce.d = d^U$).为了获取“开会”这一上下文信息,我们引入一条推导规则(如图 1 所示),则整个过程可以分为以下 3 步(针对这一问题可以有多种可行的处理流程,图 1 所示的只是其中一种处理过程):(1) 合并 C_L, C_S, C_M,得到系统所有信息的集合,记为 C;(2) 对 C 执行推导规则 r(即 $C \circ r$),获得有关“会议”的两个上下文条目 ce_{r_1}, ce_{r_2} (如图 1 所示),分别表示“Bob 与 Alice 9:00am–10:00am 在开会”和“Alice 与 Bob 9:00am–10:00am 在开会”;(3) 最后用最新推导获得的上下文 C'更新原先的 C(即 $C \Uparrow C'$),得到最终结果.

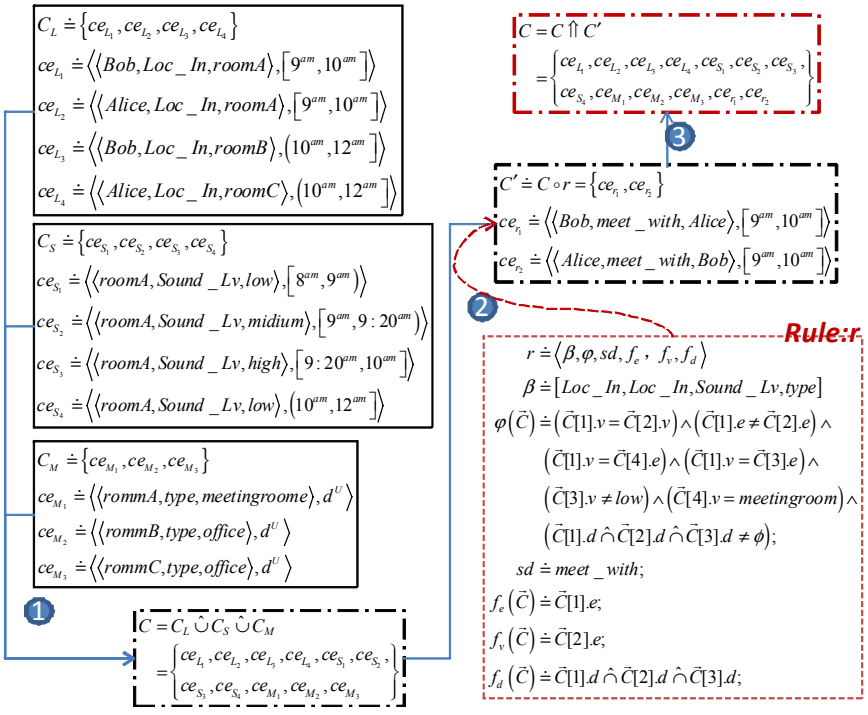


Fig.1　A simple scenario described by the proposed context model

图 1　一个利用本文上下文模型刻画的简单场景

3 实现与分析

上一节我们介绍了基于集合的上下文抽象模型,为了验证该模型的可用性,我们给出一个参考实现.具体而言,对应于前面所介绍的模型中的各个抽象概念,我们分别用不同的类来加以实现.如图 2 所示的类图给出了主要的类以及它们之间的关系.接下来我们介绍详细介绍其中主要的类及其方法.

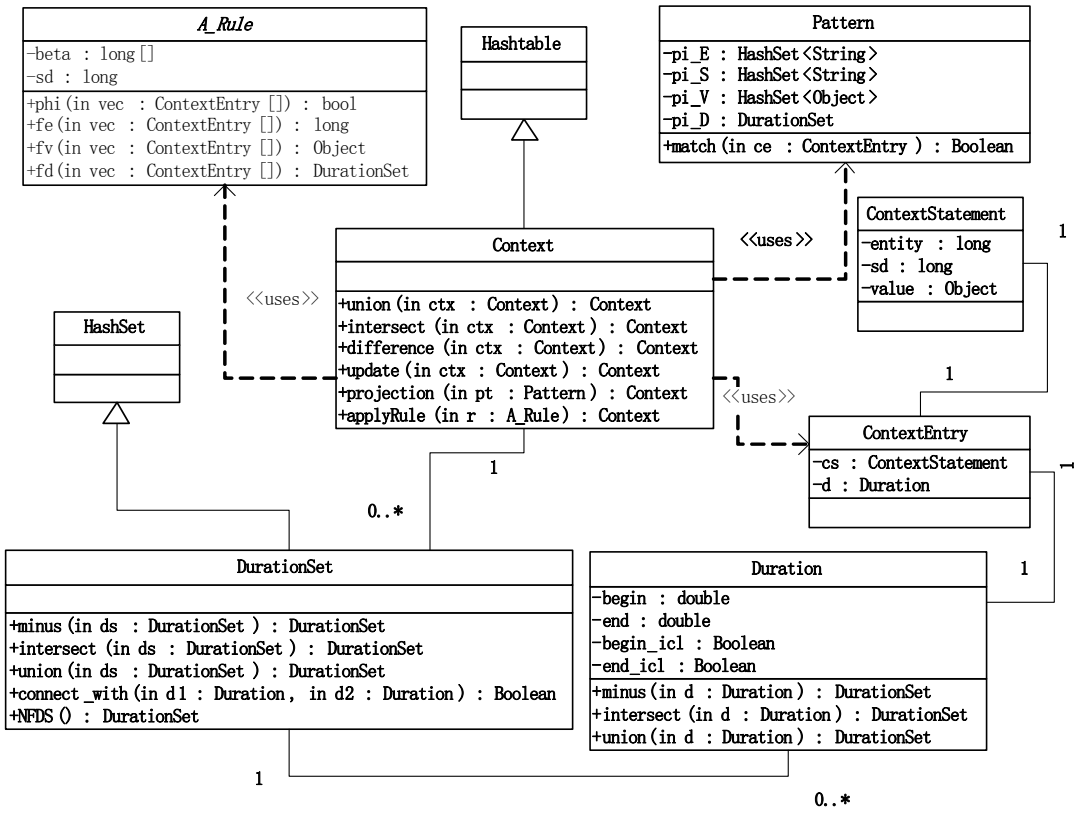


Fig.2 Key classes of the reference implementation

图 2 参考实现的关键类

3.1 Duration与DurationSet

如图 2 所示,Duration 类具有 4 个域(field):*begin*,*end*,*begin_icl*,*end_icl*.其中 *begin*,*end* 用于描述具体的起止时间,而 *begin_icl*,*end_icl* 则用于表述所指 Duration 是否包含两个端点.Duration 类提供了 *equals*(),*before*(),*after*(),*within*(),*contains*(),*pre_overlap*(),*suc_overlap*(),*overlap*(),*dir_connect*()等方法以表述 Duration 之间的各种关系(为了节省空间,这些方法没有在图 2 中显示);*minus*(),*intersect*()以及 *union*()这 3 种方法则分别具体实现 Duration 的⊖,⊗,⊕三个操作.由于上述方法都只涉及到有限步骤(有固定上界)的操作,很容易证明它们的时间复杂度均为 $O(1)$.

DurationSet 类的 *connect_with*()方法用于判断其所包含的两个 Duration 实例是否相连;*NFDS*()方法则是获取指定 DurationSet 实例 *D* 所对应的规范形式(如图 3 所示).该方法首先按照每个 duration 的开始时间对 *ds* 进行升序排序得到 *sort*; 然后按照顺序逐个合并 *sort* 中的相连的 duration(第 3 行~14 行);最终获取 *ds* 的规范形式.假设 *ds* 的大小为 *L*,不难发现,该方法的时间复杂度取决于 *ds* 的排序(第 2 行),为 $O(L\log L)$.

minus(),*intersect*()以及 *union*()这 3 种方法则分别具体实现 DurationSet 的⊖,⊗,⊕三个操作.在此,我们仅介

绍 *union*(),*minus*(),*intersect*()的情况可以类似分析.图 4 给出了 *union*()的伪码实现:给定两个 NDFS, *ds*1 和 *ds*2, 如果两者有一个的大小为 0,则直接复制另一个并返回(见图 4 中第 1 行~第 6 行);否则,首先将两者分别依据每个 duration 的开始时间升序排序得到 sort1/sort2(第 8 行、第 9 行);然后,获取 sort1/sort2 两个序列中开始时间最小的 duration,将其从序列中去除并赋值给 *cur*(第 10 行~第 15 行);接下来则循环完成合并操作直至 sort1/sort2 两个序列均为空(第 16 行~第 29 行):从 sort1/sort2 两个序列获取并删除开始时间最小的 duration,设为 *d*.判断 *d* 是否与 *cur* 相连.如果两者相连,则将两者融合;否则将 *cur* 将不可能与后面的其他 duration 进行融合,则将其加入最终结果,并令 *cur*=*d*.假设 *ds*1 和 *ds*2 的大小分别为 L_1/L_2, 则获取 sort1/sort2 分别需要 $O(L_1\log L_1)$ 和 $O(L_2\log L_2)$ 的时间开销.而第 16 行~第 29 行的循环中,每次都需要从 sort1 或 sort2 中去除一个元素,因而其复杂度为 $O(L_1+L_2)$. 综合以上分析,我们可以得出以下结论:(1) 当 $L_1=0$ 或 $L_2=0$ 时,整个 *union*()的时间复杂度为 $O(L_1+L_2)$;(2) 当 $L_1>0$ 且 $L_2>0$ 时,整个 *union*()的时间复杂度为 $O(L_1\log L_1)+O(L_2\log L_2)+O(L_1+L_2)=O(L_1\log L_1+L_2\log L_2)$.

Algorithm 1: DurationSet $NDFS()$

Input: A DurationSet :ds
Output: The normal form of ds

```
1: DurationSet res←new DurationSet();
2: sort← ds.sort();
3: for (int i = 0;i < sort.length;i++) do
4:    d1 ← sort[i];
5:    for (int j = i + 1;j < sort.length;j++) do
6:       if d1♦sort[j] then
7:          d1 ← (d1 ⊕ sort[j])[0];
8:          i ← j;
9:       else
10:         break;
11:      end if
12:   end for
13:   res.add(d1);
14: end for
15: return res;
```

Fig.3 Pseudo code for *Duration*.*NDFS*()

图 3 *Duration*.*NDFS*()伪码实现

Algorithm 2: DurationSet $union()$

Input: Two DurationSet :$ds1, ds2$
Output: The union of $ds1, ds2$

```
1: if ds1.size() = 0 then
2:    return ds2.clone();
3: end if
4: if ds2.size() = 0 then
5:    return ds1.clone();
6: end if
7: DurationSet res←new DurationSet();
8: sort1← ds1.sort();
9: sort2← ds2.sort();
10: cur← findFirst(sort1.head, sort2.head);
11: if cur = sort1.head then
12:    sort1.remove(cur);
13: else
14:    sort2.remove(cur);
15: end if
16: while sort1.size() + sort2.size() > 0 do
17:    d← findFirst(sort1.head, sort2.head);
18:    if d = sort1.head then
19:       sort1.remove(d);
20:    else
21:       sort2.remove(d);
22:    end if
23:    if cur♦d then
24:       cur← (cur ⊕ d)[0];
25:    else
26:       res.add(cur);
27:       cur← d;
28:    end if
29: end while
30: return res;
```

Fig.4 Pseudo code *for DurationSet*.*union*()

图 4 *DurationSet*.*union*()伪码实现

3.2 Context

正如前文所述,Context 是由一系列 ContextEntry 所构成的集合.在实现中,我们采用了如图 5 所示的层次式索引结构来实现 Context.由于每一层次都是一个哈希表,从而可以提高访问的效率;与此同时,该结构还能避免重复存放⟨*e*,*sd*,*v*⟩信息,节省了存储空间.

基于该结构,我们进一步给出了前文所述的各个上下文操作的实现.具体而言,图 6 给出了实现上下文并操作 ∪ 的方法 *union*()的实现伪码.其基本思路是,针对 $\kappa(c2)$ 中的每一个 *cs*, 分别获取其在 *c*1 和 *c*2 中所对应的 DurationSet *ds*1 和 *ds*2 (第 4 行、第 5 行);如果 *ds*1!= *null*, 即在 *c*1 中包含记录 *cs* 的信息,则将 $ds1\oplus ds2$ 作为最终结果 *res* 中 *cs* 所对应的 DurationSet;否则, *res* 中 *cs* 所对应的 DurationSet 为 *ds*2. 为了分析该方法的时间复杂度,我们假定 *c*1 和 *c*2 中所包含的所有 duration 的数目为 n_1 和 n_2, $\kappa(c1)$ 和 $\kappa(c2)$ 的大小分别为 m_1 和 m_2; 假定 $l_1=\max_{cs\in\kappa(c1)}(\delta(c1,cs).size()), l_2=\max_{cs\in\kappa(c2)}(\delta(c2,cs).size())$. 显然有 $n_i\leqslant m_i\times l_i, m_i+l_i\leqslant n_i+1(i=1,2)$, 于是,克隆 *c*1 到 *res* 需

要将 $c1$ 所有的元素全部克隆一遍,因而时间复杂度为 $O(n_1)$; 通过 $\kappa(c2)$ 获取 s_cs2 的复杂度则为 $O(m_2)$. 由于我们采用哈希表作为存储结构, $\delta()$ 和 $set()$ 操作可以在 $O(1)$ 时间内完成.因此,最终影响复杂度的关键在于当 $ds1!=null$ 时的 $ds1\oplus ds2$ 操作(第 6 行~第 8 行)以及对 $ds2$ 的克隆操作(第 10 行).在最好情况下,即针对 $\kappa(c2)$ 中的每一个 $cs, ds1=null$ 的时刻,此时整个循环体(第 3 行~第 13 行)其实逐一克隆了 $c2$ 中的所有 duration,因而整个算法的复杂度为 $O(n_1)+O(n_2)$; 不难证明(分别假设 $m_1>1$ 或 $m_2>1$, 通过反证法可得证),当 $\kappa(c1)=\kappa(c2), m_1=m_2=1, l_1=n_1, l_2=n_2$ 时,出现最坏情况.此时整个 $union()$ 算法的时间复杂度可以表示为 $O(n_1)+O(m_2)+m_2O(l_1\log l_1+l_2\log l_2)=O(n_1\log n_1+n_2\log n_2)$. 同理,我们不难得到 $difference()$,$intersect()$的在最坏情况下的时间复杂度也是 $O(n_1\log n_1+n_2\log n_2)$.

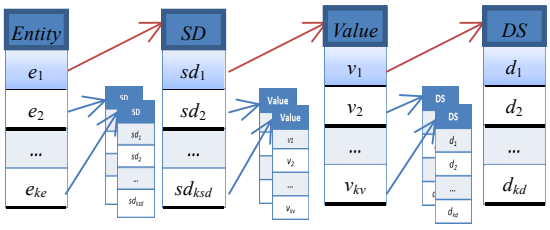


Fig.5 Structure of context class

图 5 Context 类结构

Algorithm 3: Context $union()$

Input: Two Context :$c1, c2$
Output: The union of $c1, c2$

```
1:  Context res← c1.clone();
2:  Set⟨ContextStatement⟩ s_cs2← κ(c2);
3:  for each cs in s_cs2 do
4:      DurationSet ds1 ← δ(c1, cs);
5:      DurationSet ds2 ← δ(c2, cs);
6:      if ds1! = null then
7:          ds1 ← ds1 ⊕ ds2;
8:          res.set(cs, ds1);
9:      else
10:         ds2 ← ds2.clone();
11:         res.set(cs, ds2);
12:     end if
13: end for
14: return res;
```

Fig.6 Pseudo code for *Context.union*()

图 6 *Context.union*()伪码实现

图 7 给出的 $C_1\Uparrow C_2$ 操作的伪码是依据公式(12)来实现的.从公式(12)我们可以看出, $C_1\Uparrow C_2$ 的结果来自于两个部分: C_2 以及 $C_1\doteqdot\bigcup_{cs\in\kappa(C_1)} sub(C_2,cs)$.第 2 行~第 18 行的代码则是用于计算 $C_1\doteqdot\bigcup_{cs\in\kappa(C_1)} sub(C_2,cs)$ 的;最后将获得的结果与 C_2 合并得到最终结果(第 19 行).与前文类似,为了分析该方法的时间复杂度,我们假定 $c1$ 和 $c2$ 中所包含的所有 duration 的数目为 n_1 和 n_2, $\kappa(c1)$ 和 $\kappa(c2)$ 的大小分别为 m_1 和 m_2; 进一步地,我们假定 $l_1=\max_{cs\in\kappa(c1)}(\delta(c1,cs).size()), l_2=\max_{cs\in\kappa(c2)}(\delta(c2,cs).size())$. 从而获取 s_cs1, s_cs2 的时间开销分别为 $O(m_1), O(m_2)$; 在最佳情况下,任意 $cs\in\kappa(c1)$ 以及任意 $cs_2\in\kappa(c2)$, 第 8 行的条件均不满足,而且 $m_1=m_2=1, l_1=n_1, l_2=n_2$.此时需要判断 $m_1\cdot m_2$ 次第 8 行的条件;且整个第 2 行~第 18 行的循环相当于执行了一次 $c1$ 的克隆操作(开销为 $O(n_1)$),在执行完第 18 行代码后 $res=c1$; 因此,最终执行 $union(res,c2)$ 的开销为 $O(n_1+n_2)$. 汇总上述开销,得到 $update()$的总开销为 $O(m_1)+O(m_2)+O(m_1\cdot m_2)+O(n_1)+O(n_1+n_2)=O(n_1+n_2)$. 在最坏情况下(即任意 $cs\in\kappa(c1)$ 以及任意 $cs_2\in\kappa(c2)$, 第 8 行的条件均满足时),针对任意 $cs\in\kappa(c1)$, 都将 $c2$ 中所包含的所有 duration 复制一次并加入 $ds2$ 中(第 7 行~第 14 行),即开销为 $O(n_2)$;且当第 7 行~第 14 行循环体结束后, $ds2$ 的大小为 n_2;之后执行 $ds2\leftarrow ds2.NDFS()$ 的开销则为 $O(n_2\log n_2)$; 最坏情况下,经过规范化后, $ds2$ 的大小仍然为 n_2, 所以执行 $ds\leftarrow ds1\ominus ds2$ 的开销为 $O(l_1\log l_1+n_2\log n_2)$; 从而执行第 4 行~第 18 行循环体的复杂度为 $O(m_1\cdot(n_2+n_2\log n_2+(l_1\log l_1+n_2\log n_2)))=O(m_1\cdot n_2\log n_2+m_1\cdot l_1\log l_1)$; 当执行第 19 行之前, res 的大小为 m_1n_2 且 $\kappa(c1)=\kappa(res)$, 从而执行 $union(res,c2)$ (即第 19 行)的时间开销为 $O(m_1n_2\log(m_1n_2)+n_2\log n_2)=O(m_1n_2\log m_1+(m_1+1)n_2\log n_2)$. 经过汇总,我们可以得到 $update()$ 在最坏情况下的开销为 $O(m_1)+O(m_2)+O(m_1n_2\log m_1+(m_1+1)n_2\log n_2)+O(m_1n_2\log n_2+m_1l_1\log l_1)$; 由于 $m_i\leqslant n_i, l_i\leqslant n_i(i=1,2)$, 最终可以简化得到 $O(n_1n_2\log n_2+n_1n_2\log n_1+n_1^2\log n_1)$.

图 8 给出了实现 $C\circ r$ 的 *applyRule*()方法的具体实现.该方法首先依据 $r.\beta$ 以及 $r.\varphi$ 得到一个 box(第 2 行~第 19 行);然后对 box 中的每个向量 $\vec{c}$,利用 $r.sd, r.f_e(), r.f_v(), r.f_d()$ 进行运算得到相应的推理结果并将其添加到最终结果 *res* 中(第 20 行~第 30 行);最后对获取到的 *res* 进行规范化得到最终结果并返回(第 31 行~第 36 行).为了分析该方法的复杂度,我们假定 c 中所包含的所有 duration 的数目为 $n,\kappa(c)$ 的大小为 $m, l=\max_{cs\in\kappa(c)}\left(\delta(c,cs).size()\right), q=r.\beta.length$,则 *applyRule*()方法需要 $O(m)$ 的时间开销来完成 $\kappa(c)$ 操作,然后执行第 4 行~第 13 行循环体的开销为 $O(q\cdot n)$. 第 14 行对 *box* 进行初始化,实际上是获取各个 $s\_entry[i](i=0:q-1)$ 的笛卡尔积.假定 $p_i=s\_entry[i].length(i=0:q-1)$,则完成 *box* 初始化的开销为 $\Theta\left(p_0\cdot p_1\cdot\ldots\cdot p_{q-1}\right)$,且初始化后 *box* 的大小为 $p=p_0\cdot p_1\cdot\ldots\cdot p_{q-1}$. 在极端情况下(例如 $m=1,\kappa(c)=\{cs\}$,且对任意 $i(i=0:q-1)$,当 $r.\beta[i]=cs$ 时), $p_i=n(i=0:q-1)$,则 $p=p_0\cdot p_1\cdot\ldots\cdot p_{q-1}=n^q$. 不难发现,第 20 行~第 35 行代码的复杂度受到第 21 行所获得的 *ds* 的大小的影响.假定此处得到的 *ds* 平均长度为 *w*,则执行第 20 行~第 30 行代码的开销为 $O(w\cdot p)=O\left(w\cdot n^q\right)$; 在极端情况下,执行完第 30 行, $\kappa(res)$ 只包含 1 个 cs', 且 $\delta(res,cs')$ 的大小恰好为 $w\cdot n^q$, 从而第 33 行进一步执行规范化操作的时间开销为 $O\left(w\cdot n^q\log\left(w\cdot n^q\right)\right)$.

Algorithm 4: Context *update*()

Input: Two Context :$c1, c2$
Output: The Context obtained by updating $c1$ with $c2$

```
1: Context res← new Context();
2: Set⟨ContextStatement⟩ s_cs1← κ(c1);
3: Set⟨ContextStatement⟩ s_cs2← κ(c2);
4: for each cs in s_cs1 do
5:    DurationSet ds1 ← δ(c1, cs);
6:    DurationSet ds2 ←new DurationSet();
7:    for each cs2 in s_cs2 do
8:       if cs.e = cs2.e ∧ cs.sd = cs2.sd then
9:          DurationSet ds_temp ← δ(c2, cs2);
10:         for each d in ds_temp do
11:            ds2.add(d);
12:         end for
13:      end if
14:   end for
15:   ds2 ← ds2.NDFS();
16:   DurationSet ds ← ds1 ⊖ ds2;
17:   res.set(cs, ds);
18: end for
19: res ← union(res, c2);
20: return res;
```

Fig.7 Pseudo code for *Context.update*()
图 7 *Context.update*()伪码实现

Algorithm 5: Context *applyRule*()

Input: A Context :c and a derivation rule: r
Output: The Context obtained by applying r to c

```
1: Context res ← new Context();
2: Set⟨ContextEntry⟩[]s_entry ←new Set[r.β.length];
3: Set⟨ContextStatement⟩s_cs ← κ(c);
4: for (int i ← 0;i < r.β.length;i + +) do
5:    for each cs in s_cs do
6:       if cs.sd = r.β[i] then
7:          DurationSet ds ← δ(c, cs);
8:          for each d in ds do
9:             s_entry[i].add(new ContextEntry(cs, d));
10:         end for
11:      end if
12:   end for
13: end for
14: Set⟨ContextEntry[]⟩box ←
    s_entry[0] × s_entry[1] × ... × s_entry[r.β.length − 1];
15: for each c⃗ in box do
16:   if !r.φ(c⃗) then
17:      box.remove(c⃗);
18:   end if
19: end for; // get the box
20: for each c⃗ in box do
21:   DurationSet ds ← r.f_d(c⃗);
22:   ContextStatement cs ← new ContextStatement();
23:   cs.e ← r.f_e(c⃗);
24:   cs.sd ← r.sd;
25:   cs.v ← r.f_v(c⃗);
26:   for each d in ds do
27:      ContextEntry ce ←new ContextEntry(cs, d);
28:      res.add(ce);
29:   end for
30: end for
31: for each cs′ in κ(res) do
32:   DurationSet ds ← δ(res, cs′);
33:   ds ← ds.NDFS();
34:   res.set(cs′, ds);
35: end for; // get the normal form of res
36: return res;
```

Fig.8 Pseudo code for *Context.applyRule*()
图 8 *Context.applyRule*()伪码实现

由此可见,为了处理 duration,使得 *applyRule*()在最坏情况下的复杂度 $O\left(w\cdot n^q\log\left(w\cdot n^q\right)\right)$ 高于现有的基于规则的推理引擎(例如,JESS(见 http://www.jessrules.com/),针对一条推理规则,JESS 在最坏情况下的复杂度为 $O\left(n^q\right)$).但值得注意的是,当应用不需要考虑时间因素时,我们可以将系统中所有 ContextEntry 的 duration 部分

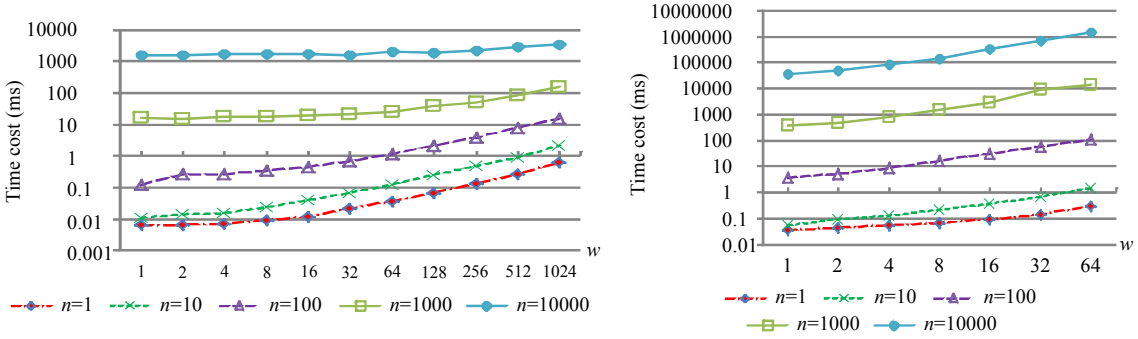
设置为 d^U, 并设置推理规则的 $f_d(\vec{C})=\{d^U\}$; 此时,对于任意 $cs\in\kappa(c)$, 都有 $\delta(c,cs).size()=1$ 且 $\delta(c,cs)=\{d^U\}$. 同理,对于任意 $cs\in\kappa(res)$, 都有 $\delta(res,cs).size()=1$, 从而每次执行第 33 行规范化的开销就转为 $O(1)$, 整个 *applyRule*()的复杂度也就蜕化为 $O(n^q)$.

3.3 实验与评估

为了进一步验证前面的分析结果,我们进行了一系列实验.实验平台配置如下:CPU:Intel Core i5(2.53 GHz), 内存:4GB,操作系统:Window7 专业版.

实验目的在于展示 *applyRule*()的最坏时间复杂度与 w,n,q 三者的联系.在该实验中我们创建一个 Context 实例 C,预先向其添加 n 个 ContextEntry.特别地,所有的 n 元素具有相同的 $cs\langle 1,1,1\rangle$, 但是各自具有互不相连的 duration;此外,我们定制了一个特殊的推理规则 $r\triangleq\langle\beta,\varphi,sd,f_e,f_v,f_d\rangle$,其中, $\beta=[]$ 是一个长度为 q 的且元素全部为 1 的数组,而 $\varphi()$ 不进行任何过滤操作而直接返回 *true*(见图 8 第 15 行~第 18 行), $sd=2, f_e()=2, f_v()=2, f_d$ 则直接返回一个大小为 w 的 NDFS.不难看出,在此配置下,我们能够得到 *applyRule*()最坏时间开销.针对不同的 w,n,q, 我们运行 *applyRule*()方法并记录其运行时间.图 9 和图 10 给出了我们的实验结果.

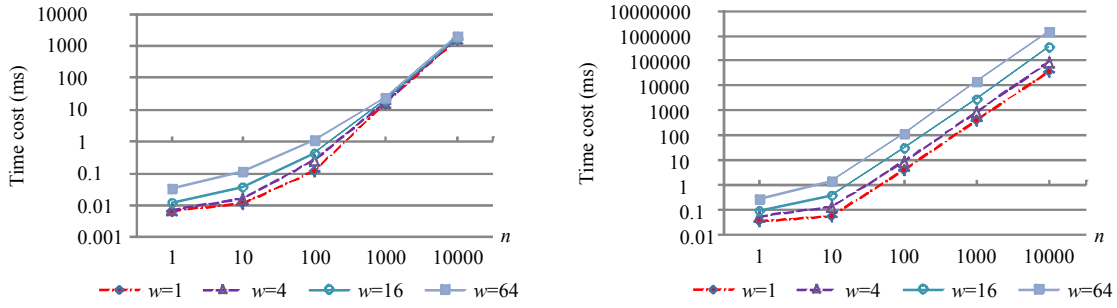
图 9、图 10 分别给出了当 $q=1,2$ 时,在给定 n 的情况下,*applyRule*()运行时间与 w 的关系,以及在给定 w 的情况下 *applyRule*()运行时间与 n 的关系.不难看出, q 是决定时间开销的最关键因素,当 $q=2$ 时(如图 9(b)、图 10(b)所示)的时间开销明显高出当 $q=1$ 时(如图 9(a)、图 10(a)所示)的时间开销.从图 10 不难看出,当 w 的值固定时,时间开销随着 n 的增长而增长,而且 q 越大,这种增长趋势越明显.



(a) q=1 (b) q=2

Fig.9 Running time of *appyRule*() increases with w

图 9 *applyRule*()运行时间随着 w 的增加而增加



(a) q=1 (b) q=2

Fig.10 Running time of *appyRule*() increases with n

图 10 *applyRule*()运行时间随着 n 的增加而增加

4 总结和展望

本文介绍了一种基于集合的,用于普适计算环境中上下文建模并描述其操作的形式化模型.该模型将上下文定义为一系列上下文条目(context entry)所构成的集合,每个上下文条目则刻画了某个特定时间段(或时间点),系统中某个实体(entity)的某个状态维(state dimension)的具体信息.特别地,通过引入 duration 机制,该模型能够统一刻画上下文的时间特性以及不同上下文信息之间与时间相关的各种关系;此外,我们对原有集合基本操作进行了修改和扩展,利用这些操作(及其组合)可以依据应用需求描述对上下文的处理逻辑.在今后的工作中,除了进一步优化我们的参考实现之外,我们的工作主要分为以下两个方面:(1) 结合相关逻辑基础(例如,基于 time interval 的时序逻辑[15]或时段逻辑[17]),尝试利用本文提出的模型以验证一些上下文相关的性质;(2) 结合面向上下文编程(context-oriented programming,简称 COP)[18],期望能够将本文的上下文模型作为语言的扩展而实现,为上下文感知应用编程提供支持.

References:

[1] Xu GY, Shi YC, XIE W. Pervasive/ubiquitous computing. Chinese Journal of Computers, 2010,26(9):1042−1050 (in Chinese with English abstract).

[2] Abowd GD, Dey AK, Brown PJ, Davies N, Smith M, Steggles P. Towards a better understanding of context and context-awareness. In: Gellersen HW, ed. Proc. of the 1st Int'l Symp. on Handheld and Ubiquitous Computing (HUC'99). London: Springer-Verlag, 1999. 304−307.

[3] Chalmers M. A historical view of context. Computer Supported Cooperative Work: The Journal of Collaborative Computing, 2004, 13:223−247.

[4] Zimmer P. A calculus for context-awareness. Technical Report, RS-05-27, BRICS, 2005. 1−21.

[5] Kjærgaard MB, Pedersen JB. A formal model for context-awareness. Technical Report, RS-06-2, BRICS, 2006. 1−24.

[6] Siewe F, Zedan H, Cau A. The calculus of context-aware ambients. Journal of Computer and System Sciences, 2011,77(4): 597−620.

[7] Cardell L, Gordon AD. Mobile ambient, in foundations of software science and computation structures. In: Nivat M, ed. LNCS 1378, Heidelberg: Springer-Verlag, 1998. 253−292.

[8] Yan L, Sere K. A formalism for context-aware mobile computing. In: Proc. of the 3rd Int'l Symp. on Parallel and Distributed Computing/the 3rd Int'l Workshop on Algorithms, Models and Tools for Parallel Computing on Heterogeneous Networks (ISPDC 2004). Washington: IEEE Computer Society, 2004. 14−21.

[9] Du XY, Li M, Wang S. A survey on ontology learning research. Journal of Software, 2006,17(9):1837−1847 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/17/1837.htm [doi: 10.1360/jos171837]

[10] Gu T, Pung HK, Zhang DQ. A service-oriented middleware for building context-aware services. Journal of Network and Computer Applications, 2005,28(1):1−18.

[11] Chen H. An intelligent broker architecture for pervasive context-aware systems [Ph.D. Thesis]. University of Maryland, 2004.

[12] Wu ZQ, Tao XP, Lu J. Journal of Frontiers of Computer Science & Technology, 2008,(4).

[13] Ranganathan A, Campbell RH. A middleware for context-aware agents in ubiquitous computing environments. In: Proc. of the 2003 Int'l Conf. on Middleware, Middleware 2003. New York: Springer-Verlag, 2003. 143−161.

[14] Bu YY, Li J, Chen SX, Tao XP, Lu J. An enhanced ontology based context model and fusion mechanism. In: Yang L, Amamiya M, Liu Z, Guo MY, Rammig, F, eds. Proc. of the EUC 2005. Heidelberg: Springer-Verlag, 2005. 920−929.

[15] Aleng J. Maintaining knowledge about temporal intervals. Communications of the ACM, 1983,26(11):832−843.

[16] Tong X, Paquet J, Mokhov SA. Complete context calculus design and implementation in gipsy. CoRR, 2010,abs/1002.4392.

[17] Zhou CC, Hoare CAR, Ravn AP. A calculus of durations. Information Processing Letters, 1991,40(5):269−276.

[18] Hirschfeld R, Costanza P, Nierstrasz O. Context oriented programming. Journal of Object Technology, 2008,7(3):125−151.

附中文参考文献:

[1] 徐光祐,史元春,谢伟凯.普适计算.计算机学报,2010,26(9):1042−1050.

[9] 杜小勇,李曼,王珊.本体学习研究综述.软件学报,2006,17(9):1837−1847. http://www.jos.org.cn/1000-9825/17/1837.htm [doi: 10.1360/jos171837]

马骏(1980－),男,贵州安顺人,讲师,主要研究领域为上下文感知计算,软件 agent 技术,普适计算中间件.

曹建农(1960－),男,博士,教授,博士生导师,主要研究领域为分布并行计算,计算机网络,移动计算,普适计算.

马超(1982－),男,博士,主要研究领域为分布并行计算,移动社交网络.

陶先平(1970－),男,博士,教授,博士生导师,主要研究领域为对象技术,软件 agent 技术,中间件技术,普适计算技术.

吕建(1960－),男,博士,教授,博士生导师,主要研究领域为软件自动化,面向对象语言,环境和并行程序的形式化方法.