软件学报 ISSN 1000-9825, CODEN RUXUEW
Journal of Software,2015,26(9):2191−2211 [doi: 10.13328/j.cnki.jos.004607]

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

一种面向移动应用的探索式服务组合方法*

白 琳[1,2,3], 魏 峻[1,2], 黄 翔[4], 叶 丹[1,2], 黄 涛[1,2]

[1](中国科学院 软件研究所 软件工程技术中心,北京 100190)

[2](计算机科学国家重点实验室(中国科学院 软件研究所),北京 100190)

[3](中国科学院大学,北京 100190)

[4](中国能源建设集团 广东省电力设计研究院,广东 广州 510663)

通讯作者: 白琳, E-mail: bailin@otcaix.iscas.ac.cn

摘 要: 开放移动平台的涌现,加速了服务组合技术在移动应用开发过程中的应用和发展.当前的移动应用开发大多采用静态的服务分类聚集的组合方式,很容易引起功能过载和服务访问链过长的问题,严重影响了移动应用的易用性.针对这一问题,结合移动应用领域的特点,提出一种探索式服务组合方法.该方法通过感知上下文变化为用户构造当前环境下可用的服务集合,并通过交互将用户选择的服务即时地组合到应用中.基于上下文构造可组合的候选服务集合是其中一个核心技术,采用历史挖掘的算法,利用用户在不同上下文环境下选择服务的历史记录,挖掘出上下文与服务间的关联关系,以此作为匹配候选服务的依据.在关联规则挖掘方面,对传统的 FP-tree 算法进行了扩展,使其支持移动应用领域中二维数据项的挖掘.实验结果表明,扩展后的算法比传统算法在服务匹配方面具有更高的准确率和命中率.

关键词: 探索式服务组合;上下文感知;移动应用;关联规则;FP-tree

中图法分类号: TP311

中文引用格式: 白琳,魏峻,黄翔,叶丹,黄涛.一种面向移动应用的探索式服务组合方法.软件学报,2015,26(9):2191−2211. http://www.jos.org.cn/1000-9825/4607.htm

英文引用格式: Bai L, Wei J, Huang X, Ye D, Huang T. An exploratory service composition approach for mobile application. Ruan Jian Xue Bao/Journal of Software, 2015,26(9):2191−2211 (in Chinese). http://www.jos.org.cn/1000-9825/4607.htm

An Exploratory Service Composition Approach for Mobile Application

BAI Lin[1,2,3], WEI Jun[1,2], HUANG Xiang[4], YE Dan[1,2], HUANG Tao[1,2]

[1](Technology Center of Software Engineering, Institute of Software, The Chinese Academy of Sciences, Beijing 100190, China)

[2](State Key Laboratory of Computer Science (Institute of Software, The Chinese Academy of Sciences), Beijing 100190, China)

[3](University of Chinese Academy of Sciences, Beijing 100190, China)

[4](Guang Dong Electric Power Design Institute, China Energy Engineering Group, Guangzhou 510663, China)

Abstract: The emergence of open mobile platform accelerates the development of service composition in mobile application. However, most current mobile applications fulfill service through static and clustering-based composition. It is prone to “software function overloading” and overlong service accessing path, hindering the efficient use for the users. In this paper, an exploratory service composition method for mobile application. With this method, candidate composable services are provided to the user when the context changes and the selected service(s) is(are) composed into the application just in time. To acquire the candidate services, association rules between the contexts and services are mined based on the history of service selection by different users in different contexts, and used as the matching basis. Further, the traditional FP-tree algorithm is extended to enable the mining of two-dimensional items in mobile

* 基金项目: 国家自然科学基金(61170074, 61173005); 国家科技支撑计划(2012BAH14B02); 国家高技术研究发展计划(863)(2012AA011204)

收稿时间: 2013-12-13; 修改时间: 2014-03-27; 定稿时间: 2014-05-05

application. Experiments show that the extended algorithm has higher precision and recall rate than the traditional method in matching for candidate services.

Key words: exploratory service composition; context-aware; mobile application; association rules; FP-tree

移动应用(mobile application,或 mobile APP,http://en.wikipedia.org/wiki/Mobile_app)是专门针对智能手机、平板电脑、甚至是新型穿戴式电子设备等智能移动终端而设计的应用软件.云计算时代,在云端不断增强服务能力的同时,端的精彩表现逐渐成为互联网信息消费模式变革的又一强力的助推器.传统依靠PC机的信息消费模式逐渐被更加灵活、便携的移动消费模式所取代.

随着互联网技术和移动技术的发展,移动开放平台的出现为移动互联网的变革带来新的契机,微信公众平台(https://mp.weixin.qq.com/)、百度轻应用开放平台(http://open.baidu.com/)、UC+开放平台(http://www.uc.cn/business/ucplus.shtml)以及新浪开放平台(http://wiki.platform.sina.com.cn/)等接踵而至.移动开放平台为移动应用屏蔽了底层操作系统的差异性,使应用不必为不同的操作系统开发和维护不同的应用版本,免除了应用下载和升级的烦恼.最重要的是,它使得基于平台的服务组合成为可能.例如,“航班管家”这个微信公众账号在应用中集成了新浪微博的服务,除了为旅客提供航班查询类服务,还能够为候机旅客提供新闻阅读服务.可见,通过服务组合,一个应用就能为用户提供更多服务,用户不必在多个应用中来回切换,从而获得更佳用户体验.

近年来,服务组合技术在电子商务、金融、电信等多个领域得到了应用和验证,已经逐渐发展成为一项成熟的网络化软件开发技术.根据服务组合机理的不同,服务组合技术可分为两大类型:第 1 类是以业务为中心的服务组合[1−3],通过构建业务流程模型实现服务间的交互与协作,流程模型刻画了服务间的控制关系和数据依赖关系,服务的执行轨迹是预先定义的、可控的;第 2 类是以用户为中心的服务组合[4−7],服务与服务之间没有严格的业务逻辑关系,而是松散耦合在一起,通过用户访问产生时序上的关联,这种关联大多是临时性的,且不可预知的,不同用户可能产生不同的服务调用序列.

在移动应用领域,大量应用涉及的是上述第 2 类以用户为中心的服务组合.为了满足用户在不同场景下的需求,移动应用在其自身的版本升级过程中总是倾向于集成更多的服务,因而很容易变得臃肿而庞杂.对于实际用户而言,往往只需要用到其中很少一部分服务,应用中大量冗余存在的服务常常给用户在选择和使用服务时造成干扰,降低了用户的使用效率[8].尤其在屏幕和操作都大受限制的移动设备上,这种干扰就愈加明显和突出.因此,需要有一种更加灵活和智能的服务组合方式,能够根据包括用户特征在内的上下文信息,为当前用户量身定做一个专属于该用户的可用服务集合,免除冗余服务对用户造成的干扰.针对这一问题,本文提出一种面向移动应用的探索式服务组合方法,该方法对移动应用领域中的上下文进行建模,利用数据挖掘算法挖掘出用户所处的上下文与服务间的关联规则,并利用这些规则向当前用户推荐当前应用场景下可用的服务集合.在应用的使用过程中,用户所访问的服务序列并不是预先定义的,而是根据应用在每个阶段的执行状态,以一种探索的方式动态生成的.

本文的主要工作与贡献体现在以下几个方面:

1) 提出了面向移动应用的即时服务组合.当前的移动应用广泛采用静态服务组合机制,不区分用户差异而构造统一的服务集合,不能根据用户的实际需要增删服务,灵活性较差.本文的服务组合是在充分考虑用户即时性需求的前提下,在用户使用应用时动态产生,对用户而言,是一种即时的个性化的装配过程,服务组合的针对性更强,也更加灵活;
2) 提出了面向移动应用的探索式服务组合.当前的移动应用广泛采用集合式的服务组合方法,将服务按照业务功能分类聚集,然后完全依靠用户的主观经验,由用户主动选择并触发使用服务,产生服务调用序列.本文所提出的探索式服务组合方法能够及时捕捉到用户当前的个性化需求,并自动构造出适用于当前应用场景的可选服务集合,为用户选择和使用服务提供一种更加智能的帮助和支持,避免了无关服务对用户所造成的干扰;
3) 提出一种面向移动应用的探索式服务组合的实现方法,采用基于FP-tree的关联规则挖掘算法对上下文与服务间的关联规则进行挖掘,并以此作为构造用户在某个上下文环境下的可选服务集合的基

础.本文对 FP-tree 算法进行了扩展,对不同的上下文属性分别定义其最小支持度,而同一上下文属性对应的不同属性值则具有相同的最小支持度.这样,既避免了单一最小支持度所带来的稀少数据项问题[9],同时也将多重最小支持度的复杂性控制到最小,使其能够很好地支持移动应用领域中二维数据项⟨上下文属性,属性值⟩的挖掘.

本文第 1 节分析移动应用中服务组合的特点及存在问题.第 2 节提出上下文感知的探索式服务组合方法,重点介绍其在移动领域中的应用.第 3 节通过一组具体案例和一组实验说明本文方法的有效性.第 4 节与相关工作进行比较分析.第 5 节总结全文并指出下一步工作计划.

1 问题分析

1.1 移动应用中服务组合的特点

(1) 采用分类聚集的集合式组合逻辑

当前,移动应用中普遍存在的是一种分类聚集的集合式服务组合,服务之间为无序的集合元素关系,不存在控制依赖、数据依赖.服务间是彼此独立的个体,其组合逻辑体现为服务间的分类聚集,即按照业务功能的相关性对服务进行类别划分与归类,在服务间构建具有层次结构的索引目录,如图 1 所示.服务组合即是按照业务相关性向服务集合的索引目录中添加服务元素的过程.

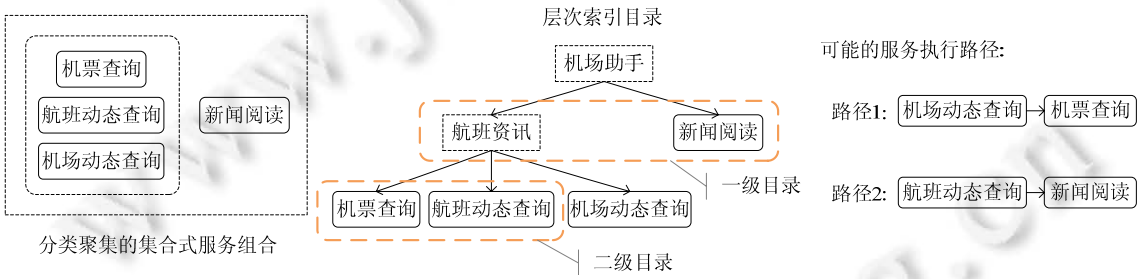

Fig.1 Service composition in APP

图 1 移动应用中的服务组合

(2) 服务执行路径由用户随机产生

分类聚集的服务组合使移动应用中任何服务都可作为服务调用的起点或终点,服务执行没有固定流程,而是由用户主观意愿决定,不同用户可产生不同的服务调用序列.例如,“航班管家”APP 提供了“机票查询”、“机场动态查询”、“航班动态查询”以及“新闻阅读”等服务.这些服务之间不存在固定的业务逻辑关系,而是在用户访问时,通过用户的访问行为产生一种即时的时序上的关联,从而形成特定的服务执行路径.例如:有的用户使用“机场动态查询”服务查询未来一周机场天气后,使用“机票查询”服务确定返程机票(如图 1 中路径 1 所示);有的用户使用“航班动态查询”服务查询所乘航班无延误或取消通告后,使用“新闻阅读”服务在候机时阅读感兴趣的新闻(如图 1 中路径 2 所示).

(3) 对上下文信息更加敏感

移动应用由于其移动\便携的特点,对位置\场景等上下文信息更加敏感.GPS 接收器、MEMS(micro-electro-mechanical system)等传感器为移动应用获取并综合运用上下文信息、开创更先进的应用领域提供了有力支持.如,通过感知用户的位置、运动速度及方向,利用增强现实(augmented reality)技术,为用户提供周围餐饮、购物、交通等服务.可见,上下文已成为移动应用提供个性化服务不可或缺的重要依据.

1.2 存在的问题

(1) 应用功能过载

当前的移动应用采用的是一种静态的服务组合.应用中选择哪些服务组合在一起、以怎样的方式组合,都是预先定义好的,不能根据用户的实际需要进行动态的调整(如增加或删除某些服务).任何用户访问应用,所看到和所能使用的都是相同的服务集合.为了满足用户在多种应用场景(上下文)下的多元个性化需求,应用提供商往往采取扩充软件功能的方式,不断地集成更多的服务,以应对更多的功能需求,因此,很容易造成功能过载问题.

(2) 服务索引层次过深

移动应用中的服务大多采用分类聚集的组合方式,服务按照功能相关性划分为不同类别,形成具有层次索引结构的菜单目录.当应用中集成了过多的服务功能(即功能过载)时,很容易使菜单设计过于复杂,服务索引层次过深,如“航班管家”APP 中就包含了长达五级的菜单目录(版本号 3.7.2,下载自“豌豆荚”移动 APP 商店).

(3) 用户使用效率低,体验差

分类聚集的组合方式使服务间的关联过于薄弱,需要用户主动去发现和选择所需的服务.因此,当服务索引层次过深而用户又缺乏相关应用的使用经验时,这种发现和选择将变得异常困难.尤其对于屏幕尺寸十分有限的手持智能终端而言,过深的服务索引意味着用户需要更多次交互才能找到需要的服务.例如,在“航班管家”APP 中,用户想要到最近的 ATM 机取款,需要依次点击“机场→服务与设施→机场设施→T3 航站楼→ATM”5 级菜单.此过程中,每一步选择都有可能偏离目标服务,且应用的功能越多,菜单设计越复杂,用户偏离目标的概率越大,用户查找和使用服务的效率也就越低.特别是随着移动应用资源的日益丰富,涌现出大量功能相同或相似的应用,但不同应用对同一服务的实现程度可能有所不同.如:同样是 ATM 机查找服务,“航班管家”APP 以文字描述 ATM 机的位置,“趣机场”APP 以地图图片显示 ATM 机位置,而“机场达人”APP 则提供动态室内导航的功能,将用户引导至所需 ATM 机的位置(如图 2 所示).在现有的服务组合方式下,用户只能通过依次下载使用才能了解每个服务的差异,最终找到满意的服务.这种服务查找和使用方式无疑是十分低效的,用户体验较差.

(a) (b) (c)

Fig.2 Three kind of navigation services for ATM

图 2 3 种 ATM 机导航服务

2 上下文感知的探索式服务组合方法

由上述分析可知:当前,移动应用中所采用的静态服务组合机制面对多元用户需求很容易导致应用功能过载,而松散的分类聚集的服务组合方式使用户不得不在功能繁多的服务集合中仅仅依靠效率低下的导航机制来完成服务的查找和使用;并且,静态的服务组合机制使得每个应用都是一个固定的服务集合,应用间彼此独立,缺乏统一的服务查找入口,更加剧了用户选择和使用服务的负担.针对这一问题,本文提出上下文感知的探索式服务组合方法,采用边执行边构造[10]的组合方式,根据当前所处的上下文环境确定下一步可组合的服务.由

于探索式服务组合是在服务的执行过程中,根据用户的即时需求,逐步添加新的服务到当前应用中,应用无需事先集成过多的服务,用户也只需面对当前状态下可能参与组合的服务,因此能够很好地解决移动应用中可能出现的功能过载问题;同时为用户选择服务提供支持,提高用户的使用效率.

上下文感知的探索式服务组合以上下文作为服务组合的主要依据,其关键是在上下文与服务间建立关联,然后依据该关联关系找出适合于当前上下文的服务集合,由用户从中选择一个或一组并发服务(如边阅读新闻边收听音乐)动态加载到当前应用中,实现由上下文驱动的探索式的服务组合过程,如图 3 所示.

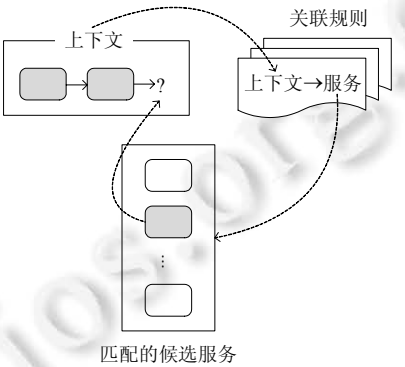

Fig.3 Context-Aware exploratory service composition

图 3 上下文感知的探索式服务组合

本文采用基于历史挖掘的方法对用户选择服务的历史进行统计和挖掘,获取上下文与服务间的关联规则.FP-tree 算法以其高效的执行效率,逐渐成为关联规则挖掘的主流算法.本文将在传统的 FP-tree 算法的基础上,结合移动应用领域的实际应用情况对传统算法进行扩充和改造,使其能够更好地应用于移动应用场景.本文方法包含 3 个主要步骤:首先,对上下文进行定义和建模;然后,在此基础上对传统的 FP-tree 算法进行改造,挖掘出上下文与服务间的关联规则;最后,利用上述规则形成与当前上下文相匹配的候选服务集合,为用户选择服务提供支持和帮助.

2.1 上下文及其模型定义

Dey 和 Abowd[11]定义上下文为:“能够描述一个实体状态的任何信息,这个实体可以是人、地点,或与用户-应用交互相关的对象,包括用户和应用本身”.时间和空间是两类最常见的上下文,此外还包括用户的年龄、性别、职业、兴趣爱好等用户特征以及业务申请、审核、审批等应用状态信息.

为了规范上下文的描述和定义,本文基于 MOF 元模型框架[12]定义上下文元模型及上下文模型.上下文元模型(如图 4 所示)定义了上下文的概念、约束及其之间的关系.其中,概念是对上下文对象的描述,概念之间可存在泛化关系或包含关系.约束是为了规范概念及其取值而定义的契约,由概念来实现.概念与约束之间为实现关系.上下文模型是对上下文元模型的实例化.由于上下文涵盖的范围非常广泛,定义统一集中的上下文模型显然是不现实的.因此,我们需要根据不同领域的应用特征定义不同的上下文模型.例如:旅行类应用可通过关注旅行期间的天气情况、用户的身体状况等上下文,为用户推荐合适的游玩项目;娱乐类应用则可关注用户当前的情绪状态来推荐不同风格的音乐或影片.本文中,我们以机场导航类应用为例定义上下文模型如图 5 所示,其中,ISO 8601 和 UTC 为当前时间所需实现的两个约束条件,分别表示当前时间的格式为国际标准格式 yyyy-mm-dd,hr:mi:se(24 小时制),且为世界标准时间.

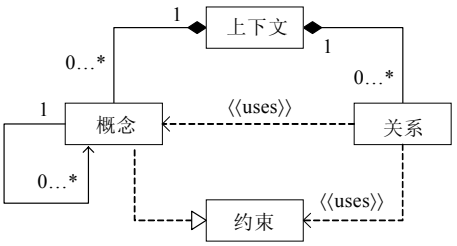

Fig.4 Context metamodel

图 4 上下文元模型

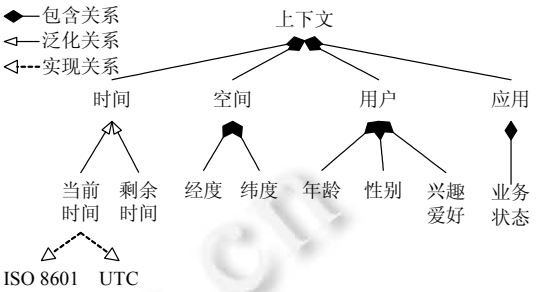

Fig.5 Context model

图 5 上下文模型

基于图 5 所示的上下文模型,对每个上下文概念赋予符合约束约定的值**,即可得上下文实例,其描述如下:

Context={*Time.current*=2013-10-01,11:00:00;

Time.last=60minutes;

Location.longitude=31°21′33″;

Location.latitude=121°47′67″;

User.age=22;

User.gender=male;

User.interests={sports};

Business.state=security_check_done

}

其中,*T.c* 和 *T.l* 分别表示当前时间和剩余时间,可通过访问设备操作系统获得;*L.long* 和 *L.lat* 分别表示当前位置的经纬度,可通过设备 GPS 定位获得;*U.a*,*U.g* 和 *U.i* 分别表示用户的年龄、性别及兴趣爱好,可通过访问用户注册信息获得.其中:用户的兴趣爱好是一个由若干兴趣点所组成的集合,兴趣点之间以“,”分隔;业务状态 *B.s* 可通过与用户的交互获得,当用户完成某项任务或活动时,由相应的结束指令触发业务状态的改变.

2.2 关联规则挖掘

2.2.1 数据预处理

我们将用户在上下文 *C* 下选择服务 *S* 的历史行为记为一条事务,记做 *C*→*S*.其中,*C* 为第 2.1 节定义的上下文,*S* 为用户在上下文 *C* 下所选的服务的集合,即 $S=\{s_1,s_2,\ldots,s_m\}$.之所以将 *S* 表示为服务集合的形式,是因为在移动应用中,用户常常需要将若干服务同时执行.例如用户在机场候机时,想要边听音乐边阅读新闻.此时,用户所选择的服务 *S* 即为{音乐播放,新闻阅读}.

表 1 列举了事务的若干示例.

Table 1 Examples of transactions

表 1 事务示例

tID	事务
1	{*T.c*=11:03:19, *T.l*=57m, *L.long*=31°21′33″, *L.lat*=121°47′67″, *U.a*=22, *U.g*=m, *U.i*={sports}, *B.s*=security_check_done}→{就餐}
2	{*T.c*=14:25:52, *T.l*=120m, *L.long*=31°11′53″, *L.lat*=121°48′19″, *U.a*=30, *U.g*=f, *U.i*={shopping}, *B.s*=check_in_done}→{购物}
3	{*T.c*=11:10:37, *T.l*=50m, *L.long*=31°8′5″, *L.lat*=121°29′11″, *U.a*=65, *U.g*=m, *U.i*={news}, *B.s*=security_check_done}→{候机}
4	{*T.c*=11:32:28, *T.l*=58m, *L.long*=31°11′9″, *L.lat*=121°53′1″, *U.a*=62, *U.g*=m, *U.i*={news}, *B.s*=security_check_done}→{候机}

可以看出,由设备直接获取的上下文数据,其值的分布非常分散,很难从中发现数据的规律,并挖掘出数据

**由于文章篇幅有限,本文只给出了当前时间的约束条件.事实上,每个上下文概念都可定义类似的约束.例如,可约束剩余时间的计量单位为 minutes.

间的关联规则.因此,需对事务中的数据进行预处理操作,将分散的数值归纳为有意义的数据区间.例如,“当前时间”这个上下文是通过访问设备操作系统而获得的,其数值精确到“秒”.在“秒”的量级上进行挖掘,很难得到有意义的关联规则.因此,我们将具体的时间划分到以下 6 个时间段:dawn,morning,noon,afternoon,nightfall,evening.同理,按照表 2 的数据操作符,对“剩余时间”、“经纬度”、“用户年龄”等上下文进行类似的操作,得到的事务列表见表 3.

Table 2 Data operators

表 2 数据操作符

操作符	说明
T.c={dawn,morning,noon,afternoon,nightfall,evening}	将“当前时间”按照以下区间划分到相应的时间段: dawn——0:00 至 5.59 morning——6:00 至 10:59 noon——11:00 至 12:59 afternoon——13:00 至 16:59 nightfall——17:00 至 19:59 evening——20:00 至 23:59
T.l=*CEILING*(*x*)hours	将“剩余时间”换算为以“小时”为单位,并向上取整.
L.long+*L.lat*⇒*L.d*=*CEILING*(*x*)harf kilometers	将“经纬度”换算为当前位置与目标位置的距离,以“五百米”为单位,并向上取整.
U.a=*CEILING*(*x*/10)*10	将“用户年龄”以 10 为单位向上取整

Table 3 Examples of transactions after pre-process

表 3 经过数据预处理的事务示例

tID	事务
1	{*T.c*=noon, *T.l*=1h, *L.d*=1hk, *U.a*=30, *U.g*=m, *U.i*={sports}, *B.s*=security_check_done}→{就餐}
2	{*T.c*=afternoon, *T.l*=2h, *L.d*=1hk, *U.a*=30, *U.g*=f, *U.i*={shopping}, *B.s*=check_in_done}→{购物}
3	{*T.c*=noon, *T.l*=1h, *L.d*=2hk, *U.a*=70, *U.g*=m, *U.i*={news}, *B.s*=security_check_done}→{候机}
4	{*T.c*=noon, *T.l*=1h, *L.d*=1hk, *U.a*=70, *U.g*=m, *U.i*={news}, *B.s*=security_check_done}→{候机}

2.2.2 基于 FP-tree 的关联规则挖掘

(1) 相关概念说明

为了叙述方便,结合本文的应用场景,我们对 FP-tree 算法中的相关概念进行以下说明:

定义 1(事务数据库). 由若干条形如 $C\rightarrow S$ 的事务所组成的集合称为事务数据库,记做 D.

定义 2(项). 事务 $C\rightarrow S$ 中的每一个上下文 $c_k(c_k\in C)$和每一个服务 $s_j(s_j\in S)$统称为项,记做 i.

为了便于区分,将 c_k 称为上下文项,s_j 称为服务项.在本文的应用场景中,挖掘上下文 C 与服务 S 间的关联规则,主要是识别 C 中上下文项的频繁模式,并将相同模式下的服务进行合并,最终形成频繁的上下文项与服务间的关联规则.

定义 3(项的支持数与项的支持度). 项 i 在事务数据库 D 中出现的次数称为项 i 的支持数,记做 $n(i)$.

项 i 的支持数与数据库中事务数总和 N 的比值称为项 i 的支持度,记做 $sup(i)$,即 $sup(i)=n(i)/N$.在本文的应用场景中,以支持度表示上下文项出现的频繁程度,便于与用户指定的最小支持度进行比较,识别出频繁项集;以支持数表示服务项出现的频繁程度,便于计算和用户选择.这里,要求用户指定项的最小支持度,而非项的最小支持数.原因在于:当事务数据库中的事务发生变化时(如添加了新的事务,使项的数目发生了变化),最小支持度以百分比的形式表达用户对项的频繁度的要求,不受项的数目变化的影响,而最小支持数则需根据数据库中项的数量多少随时调整大小.

(2) 算法描述

应用传统的 FP-tree 算法挖掘上下文 C 与服务 S 间的关联规则,包含以下两个主要步骤[13]:

1) 对事务数据库 D 中所有的上下文项,计算其支持度,删除支持度小于最小支持度的上下文项,然后按支持度进行降序排列;

2) 扫描事务数据库 D 中的所有事务,构造 FP-tree 树,使每条事务都对应树中的一条路径,即:对于每一条事务,或者在树中增加一条新的路径,或者在已有路径上增加相应节点的计数.

图 6[***]为对表 3 所示的事务数据库采用传统的 FP-tree 算法所构建的 FP-tree 树(最小支持度设置为 0.5).树中的每一条路径都蕴含了上下文 C 与服务 S 间的关联关系,即,每条路径都代表了一个关联规则,记做 R:$C\rightarrow S$.其中,C 为路径中除根节点外的分支节点所构成的上下文,S 为路径的叶子节点.

项	支持度/支持数	节点头
T.c=noon	0.75	
T.l=1	0.75	
L.d=1	0.75	
U.g=m	0.75	
B.s=security_check_done	0.75	
U.a=30	0.5	
U.a=70	0.5	
U.i=news	0.5	
候机	2	
购物	1	
就餐	1	

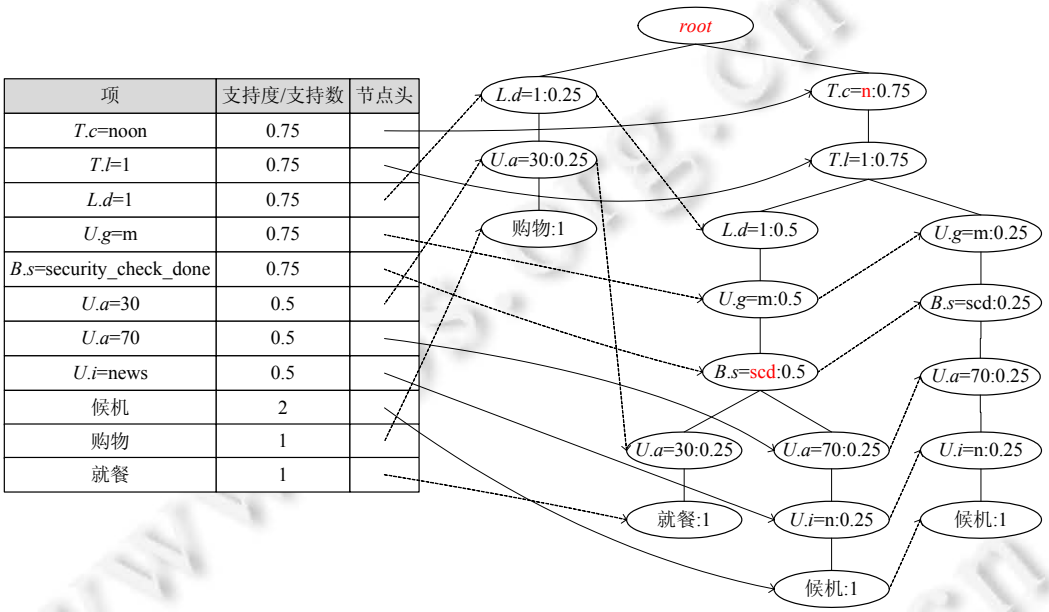

Fig.6 Fp-Tree built by traditional algorithm

图 6 采用传统算法构建的 FP-tree

(3) 存在的问题

然而在实际应用中,采用传统的 FP-tree 算法可能造成稀少数据项[9]的问题,即:规则中缺少了出现频次少的项,使所挖掘的关联规则不够准确.例如,图 6 中的规则{*L.d*=1,*U.a*=30}→{购物}表示年龄在 30 岁的用户,当购物地点在周围 500 米范围内时,通常会去购物.但是根据实际情况可知,用户是否购物的关键因素取决于当前时间距离航班起飞的时间间隔,也就是说,上述规则中丢失了一个很重要的上下文项:*T.l*=2.

造成这一问题的原因在于:传统的 FP-tree 算法对所有的上下文项采用统一的最小支持度,使得支持度小的上下文项被丢弃了.在传统的 FP-tree 算法应用场景中,事务中的项都是一维的,如牛奶、面包、薯片等.这些项的本身就是项的取值,因此只需设置一个最小支持度,就能够识别哪些项是频繁的.而在本文的应用场景中,事务中的上下文项是二维的,每个上下文项都是一个由上下文属性(即上下文模型中的概念)及其取值所形成的键值对,且每个上下文属性的取值范围皆不相同.取值范围较广的上下文属性与取值范围较窄的上下文属性相比,在数据分布均匀的情况下,其对应的上下文项的支持度会偏低.这是因为上下文属性的取值范围越广,其所对应的每个可能的上下文项的出现概率越小,即,出现的次数越少,因而支持度也越小.例如,用户年龄可在 0~100 之间浮动(即使按照表 2 所示的数据操作符进行数据预处理后,仍有 10 个可选的取值),而用户性别却只有 f 和 m 两个选择.显而易见,在同一数据库中,*U.a*=*x*(0<*x*<100)出现的概率远比 *U.g*=m 或 *U.g*=f 出现的概率小.如果对所有的上下文项都设置统一的最小支持度,那么出现概率小的上下文项则有可能被丢弃.因此对于二维项,应当根据

***为了便于显示,树中的节点对部分上下文项的取值进行了缩写,如 *T.c*=noon 缩写为 *T.c*=n,*B.s*=*security_check_done* 缩写为 *B.s*=scd,*U.i*=news 缩写为 *U.i*=n.

项的取值范围,对属于不同上下文属性的项设置不同的最小支持度.

2.2.3 基于 FP-tree 扩展算法的关联规则挖掘

本节对传统的 FP-tree 算法进行扩展,使之支持对二维上下文项的挖掘.核心思想是:对每个上下文属性都设置一个最小支持度;然后,按照该最小支持度对数据库中的项进行过滤和排序,形成二维的频繁项集;最后,依据二维频繁项集对事务数据库中事务的项进行过滤和排序,生成 FP-tree 树.算法过程与描述如图 7 所示.

tID	事务
1	{*T.c*=noon, *T.l*=1h, *L.d*=1hk, *U.a*=30, *U.g*=m, *U.i*={sports}, *B.s*=security_check_done}→{就餐}
2	{*T.c*=afternoon, *T.l*=2h, *L.d*=1hk, *U.a*=30, *U.g*=f, *U.i*={shopping}, *B.s*=check_in_done}→{购物}
3	{*T.c*=noon, *T.l*=1h, *L.d*=2hk, *U.a*=70, *U.g*=m, *U.i*={news}, *B.s*=security_check_done}→{候机}
4	{*T.c*=noon, *T.l*=1h, *L.d*=1hk, *U.a*=70, *U.g*=m, *U.i*={news}, *B.s*=security_check_done}→{候机}

计数、排序、过滤

首先按照项的最小支持度降序排列

然后按照项的支持数降序排列

删除出现频率低的项

项	最小支持度↓1	支持度	支持数↓2
U.g=m	0.4	0.75	3
~~*U.g*=f~~	~~0.4~~	~~0.25~~	~~1~~
L.d=1	0.35	0.75	3
~~*L.d*=2~~	~~0.35~~	~~0.25~~	~~1~~
B.s=security_check_done	0.35	0.75	3
~~*B.s*=check_in_done~~	~~0.35~~	~~0.25~~	~~1~~
T.c=noon	0.3	0.75	3
~~*T.c*=afternoon~~	~~0.3~~	~~0.25~~	~~1~~
U.a=30	0.25	0.5	2
U.a=70	0.25	0.5	2
U.i=news	0.2	0.5	2
U.i=shopping	0.2	0.25	1
U.i=sports	0.2	0.25	1
T.l=1	0.1	0.75	3
T.l=2	0.1	0.25	1
候机	0	-	2
购物	0	-	1
就餐	0	-	1

默认服务项的最小支持度为0,排在全部上下文项之后

过滤、排序

tID	事务
1	*U.g*=m, *L.d*=1hk, *B.s*=security_check_done, *T.c*=noon, *U.a*=30, *U.i*={sports}, *T.l*=1h, {就餐}
2	~~*T.c*=afternoon,~~ *L.d*=1hk, *U.a*=30, ~~*U.g*=f,~~ *U.i*={shopping}, *T.l*=2h, ~~*B.s*=check_in_done,~~ {购物}
3	*U.g*=m, *B.s*=security_check_done, *T.c*=noon, ~~*L.d*=2hk,~~ *U.a*=70, *U.i*={news}, *T.l*=1h, {候机}
4	*U.g*=m, *L.d*=1hk, *B.s*=security_check_done, *T.c*=noon, *U.a*=70, *U.i*={news}, *T.l*=1h, {候机}

Fig.7 Sketch map of the algorithm

图 7 算法过程示意图

算法. FPTREEBUILDING.

输入:事务数据库 *D*,

上下文属性最小支持度 Min*Sup*={Min$Sup_{attr\text{-}1}$,Min$Sup_{attr\text{-}2}$,…,Min$Sup_{attr\text{-}t}$};

输出:FP-tree.

```
    /*初始化*/
1.  FIS=∅;                                  //频繁项集
2.  root=new Node(·);                       //根节点
3.  FP-tree.root=root;
    /*生成频繁项集*/
4.  for each i in D
5.     if FIS.contains(i)
6.        i.count++;
7.     else
8.        FIS.add(i);
9.        i.count=1;
10.    end if
11. end for                                 //统计项的支持数
12. for each i in FIS
13.    if i is kind of C
14.       i.sup=i.count/N;                  //i 为上下文项,计算支持度
15.       if i.sup<MinSup_i.attribute
16.          FIS.delete(i);                 //删除支持度小于最小支持度的项
17.       end if
18.    end if
19. end for
20. sort FIS in descending order by MinSup; //先按照项的最小支持度降序排列
21. sort FIS in descending order by i.count; //再按照项的支持数降序排列
    /*构建 FP-tree 树*/
22. for each transaction t in D
23.    for each i in t
24.       If !FIS.contains(i)
25.          delete i;
26.       end if
27.    end for
28.    sort i in t with FIS order;          //按照 FIS 中项的顺序对事务中的项进行排序
29.    insert(t,root);                      //将当前事务插入树中
30. end for
31. return FP-tree;
```

算法. INSERT.

输入:事务 *t*,

树节点 *node*;

输出:*NULL*.

```
32. i=t.getFirst(·);                        //获取第 1 个项
    /*i 为上下文项*/
33. if i is kind of C
34.    I=t.removeFirst(·);                  //获取剩余项
```

```
35.    child=node.findChild(i);
36.    if child!=null
37.       child.count++;
38.       node=child;
39.    else
40.       new_node=new Node(·);
41.       node.addChild(new_node);
42.       new_node.count=1;
43.       new_node.parent=node;
44.       node=new_node;
45.    end if
46.    insert(I,node);                          //递归调用
       /*i 为服务项*/
47. else
48.    if node.hasChild
49.       leaf=node.getChild(·);
50.       for each i in t
51.          if leaf.contains(i)
52.             leaf.get(i).count++;
53.          else
54.             leaf.add(i);
55.             leaf.get(i).count=1;
56.          end if
57.       end for
58.    else
59.       leaf=new Node(·);
60.       for each i in t
61.          leaf.add(i);
62.          leaf.get(i).count=1;
63.       end for
64.    end if
65. end if
```

整个算法包含两个主要步骤:生成二维项的频繁项集(第 4 行~第 21 行),依据频繁项集构建 FP-tree(第 22 行~第 31 行).根据第 2.2.2 节对“项”的概念的说明,本文中的项包括两个类型:上下文项为二维项,每个上下文项都是一个由上下文属性和属性值组成的“键值对”;服务项为一维项,由服务名构成.算法依次扫描数据库中每条事务的每个项,统计项的支持数(第 4 行~第 11 行),对其中的上下文项计算支持度,并与相应的上下文属性最小支持度进行比较,删除出现频率较低的项,得到频繁项集(第 12 行~第 19 行).上下文属性的最小支持度是由用户设定的,并且在设定时考虑了上下文属性的取值范围,即,考虑了其中每个属性值可能出现的概率.对于取值范围小的上下文属性设定较高的最小支持度,反之设定较低的最小支持度,用以保证支持度较小的上下文项能够得以保留.如图 7 所示,上下文项 $U.g$=f 的支持度为 0.25,其对应的上下文属性 $U.g$ 的最小支持度为 0.4,表明上下文项 $U.g$=f 在应用中的出现频率较低,对寻找上下文间的频繁模式帮助不大,可忽略.而上下文项 $T.l$=2 的支持度也是 0.25,但由于上下文属性 $T.l$(剩余时间)的取值范围相对较广,用户对其设定了较小的最小支持度 0.1,使得上下文

项 *T.l*=2 保留在频繁项集中.而对于服务项,只需计算其支持数,用作服务排序的依据.在某个特定的上下文环境下,服务项的支持数越高,表明该服务出现的概率越大,可优先提供给用户.为了方便服务匹配,将出现频率高的项放置在树的顶端(靠近根节点的位置),算法第 20 行、第 21 行对频繁项进行降序排列.由于上下文项是二维的,单独依靠支持度不能反映项的出现频度,因此,本文对二维项采用双重排序的方法:先按照每个项所属的上下文属性的最小支持度排序,对同一上下文属性中的项,再按照项的支持度排序.为了便于对上下文项和服务项统一操作,可默认服务项的最小支持度为 0,使得服务项排在上下文项之后,这样便于在构造 FP-tree 树时,将服务项构造为叶子节点.

构建 FP-tree 时,需要依次扫描数据库中的每条事务,将其中的非频繁项(频繁项集之外的项)删除(第 23 行~第 27 行),然后将剩余的项按照频繁项集中项的顺序进行排列(第 28 行),并依次插入到 FP-tree 树中(第 32 行~第 65 行).插入过程与传统的 FP-tree 算法类似,不同之处在于,本文中事务的项包含两个类型:上下文项和服务项.插入时,每个上下文项对应路径中的一个分支节点(根节点除外),要么增加已有分支节点的计数(第 35 行~第 38 行),要么创建新的分支节点(第 39 行~第 45 行),形成新的路径.服务项对应路径中的叶子节点.由于一条事务中可能包含多个服务项,因此,每个叶子节点都对应一个服务项的集合.当插入服务项时,首先判断当前事务在树中是否已存在吻合的路径:

- 若存在,则表明插入本事务之前,树中已经存在这样一条完整的路径,使得路径上分支节点的序列和事务中上下文项的序列一致.此时,只需将事务中的服务项合并到该路径的叶子节点中(第 48 行~第 57 行):要么增加已有服务项的计数,要么添加新的服务项;
- 否则,表明插入本事务时,在树中产生了新的路径.因此,需要创建新的叶子节点,并将事务中的服务项全部添加到新创建的叶子节点中(第 58 行~第 64 行).

算法执行的结果如图 8 所示.

项	最小支持度↓1	支持度	支持数↓2	节点头
U.g=m	0.4	0.75	3	
L.d=1	0.35	0.75	3	
B.s=security_check_done	0.35	0.75	3	
T.c=noon	0.3	0.75	3	
U.a=30	0.25	0.5	2	
U.a=70	0.25	0.5	2	
U.i=news	0.2	0.5	2	
U.i=shopping	0.2	0.25	1	
U.i=sports	0.2	0.25	1	
T.l=1	0.1	0.75	3	
T.l=2	0.1	0.25	1	
候机	0	-	2	
购物	0	-	1	
就餐	0	-	1	

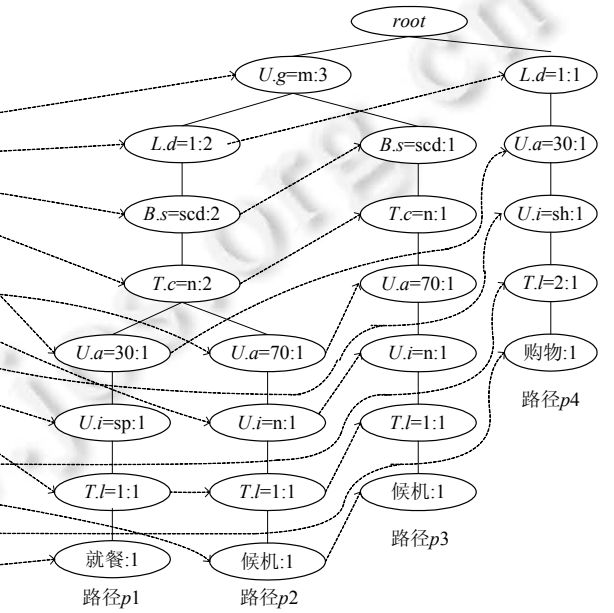

Fig.8 FP-Tree built by extended algorithm

图 8 采用扩展算法构建的 FP-tree

与图 6 所示结果相比,采用扩展后的算法所构建的 FP-tree 包含更多上下文项,使那些出现频率较低但却对规则而言具有重要意义的上下文项得以保留.如:规则{*L.d*=1,*U.a*=30,*U.i*=sh,*T.l*=2}→{购物}保留了支持度较低的上下文项 *U.i*=sh 和 *T.l*=2,与原来的规则{*L.d*=1,*U.a*=30}→{购物}相比更具实际意义,确保依据规则筛选得到

的候选服务也能更加准确.

2.3 基于关联规则的服务匹配

FP-tree 中的每一条路径都蕴含了一条形如 $C\rightarrow S$ 的规则,其中,C 为路径中分支节点(根节点除外)按照自顶向下的顺序所形成的有序集合,S 为路径中叶子节点所包含的服务集合.当用户处于上下文环境 C'时,构造符合 C'的可组合候选服务集,即是在 FP-tree 中查找与 C'相似的路径,路径底端的叶子节点所包含的服务,即为用户在 C'环境下可选的服务.

首先,将用户当前所处上下文 C'中的项按照频繁项集进行过滤和排序,删除频繁项集中不存在的项;同时,将剩余的项按照频繁项集中项的顺序进行降序排列,形成新的上下文项的集合 C'';然后,对 C''与 FP-tree 中的路径 p 求交集,可能存在以下 4 种情况:

1) 路径 p 所包含的上下文项全部属于 C'',称 p 为 C''的子集路径;
2) C''中的每一个项都在路径 p 中,而 p 中至少有一个上下文项不属于 C'',称 p 为 C''的真超集路径;
3) 至少存在一个项 i_x 既属于 C''又属于路径 p;同时,C''中至少有一个项 i_y 不在路径 p 中;并且,路径 p 中也至少有一个上下文项 i_z 不属于 C''.称 p 为 C''的交叉路径;
4) 不存在项 i 既属于 C''又属于路径 p,称 p 为 C''的无关路径.

如图 8 所示的 FP-tree 树,当用户所处上下文 C''={$U.g$=m,$B.s$=scd,$T.c$=n,$U.a$=70,$U.i$=n,$T.l$=1}时,路径 $p1$ 为 C''的交叉路径,$p2$ 为真超集路径,$p3$ 为子集路径,$p4$ 为无关路径.

接着,根据交集的结果对每条路径赋予不同的权重系数.无关路径由于和上下文 C''没有任何交集,对 C''环境下的服务匹配没有意义,因此忽略不计.这里,只需对其他类型的路径计算权重.

本文定义路径权重为 $p.weight=A+B$,其中,A 表示路径 p 的等级.根据路径对当前环境下服务匹配贡献的大小,定义子集路径、真超集路径及交叉路径的等级依次为 3,2,1.B 表示同等级路径中,路径的优劣程度.根据路径类型的不同,B 的计算方式也有所不同:

1) 当 p 为 C''的子集路径时,$B=1-1/|p|$.表示 p 的路径长度越长,p 所描述的上下文与 C''越接近,p 的权重越大;
2) 当 p 为 C''的真超集路径时,$B=1/|p|$.表示 p 的路径长度越短,p 所描述的上下文与 C''越接近,p 的权重越大;
3) 当 p 为 C''的交叉路径时,$B=1-1/|intersection(C'',p)|$.$|intersection(C'',p)|$为路径 p 与上下文 C''的交集中元素的个数.B 表示 p 和 C''重叠的部分越多,p 所描述的上下文与 C''越接近,p 的权重越大.

最后,根据路径的权重系数合并路径中的服务,计算每个服务的综合支持数,即综合多个路径后的总支持数,将总支持数最大的前 k 个服务(Top-k)提供给用户.在计算服务总支持数时,需要区分不同路径中的服务对服务总支持数贡献的大小.例如,路径 p 的权重为 $p.weight$,s 为 p 中包含的一个服务,s 的支持数为 $s.count$.当合并 s 时,路径 p 中的服务 s 所贡献的支持数大小为 $s.count*p.weight$.

具体的算法过程描述如下:

算法. RECOMMENDATION.

输入:FP-tree,

用户当前所处上下文环境 C,

返回结果个数 k;

输出:服务集合 S.

```
1. filter and sort C by FP-tree.FIS;                    //依据频繁项集过滤和排序
2. P=FP-tree.getAllPath(·);
3. for each p in P
     /*计算路径权重系数*/
4.   if C is a subset of p
```

```
5.      p.weight=3+1/|p|;
6.    else if C is a true superset of p
7.      p.weight=2+(1−1/|p|);
8.    else if C and p crossed
9.      p.weight=1+(1−1/|intersection(C,p)|);
10.   else
11.     break;                                  //忽略无关路径
12.   end if
      /*计算服务支持数*/
13.   for each s in p.leaf
14.     if S.contains(s)
15.       s′=S.get(s);
16.       s′.count=s′.count+s.count*p.weight;
17.       S.set(s′);
18.     else
19.       s.count=s.count*p.weight;
20.       S.add(s);
21.   end for
22. end for
23. sort S by s.count;                          //按照服务支持数降序排列
24. filter S to get top k services;             //选取前 k 个结果
25. return S;
```

3 实验分析

本节通过一组具体的案例和一组对比实验来验证本文方法的有效性.其中,案例分析部分对两个典型的应用场景进行分析,目的在于:

1) 说明本文所提出的探索式服务组合方法与传统的移动应用中的服务组合不同,服务间的组合关系不是在应用开发时就定义好的,而是应用使用过程中,随着应用场景的演化,以动态的方式逐步添加用户需要的服务并组合到应用中来;
2) 说明本文所提出的探索式服务组合方法能够在用户探索的过程中给予提示和帮助,即:根据用户所处的上下文环境构造出用户当前可能需要的服务集合,从而减轻用户查找服务的负担.

实验部分对本文所提出的 FP-tree 扩展算法与传统的 FP-tree 算法进行对比分析,目的在于验证本文算法在匹配可组合的候选服务时能够获得更高的准确率和命中率.随后,通过实验对影响服务匹配准确率和命中率的参数进行分析并得出结论.

3.1 案例分析

本节仍然以机场导航类应用为例,通过阐述用户与应用的整个交互过程,说明该类应用是如何在不同的应用场景中为用户选择服务提供支持,并以不同的服务组合结果完成机场导航任务.

场景 1:旅客甲对机场设施非常熟悉.甲到达机场时,距离航班起飞时间还有 70 分钟.甲使用机场导航 APP 的过程如图 9 所示****.

场景 2:旅客乙对机场设施不熟悉.乙到达机场时,距离航班起飞时间还有 3 小时.乙使用机场导航 APP 的过

****因篇幅有限,图中只列举了部分上下文信息.*T.l* 表示当前时间距离航班起飞的时间间隔,*B.s* 表示用户当前所处的应用状态.

程如图 10 所示.

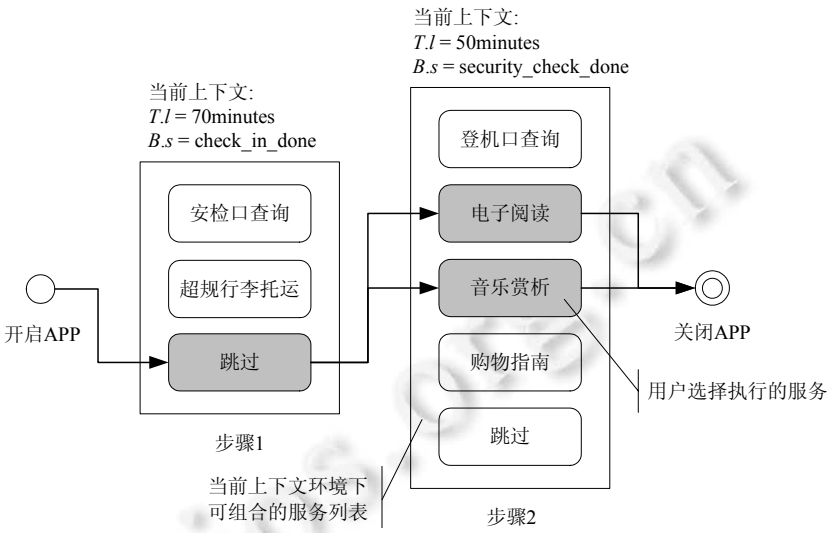

Fig.9 Usage procedure of AirportAssistant APP in scenario one

图 9 场景 1 中机场导航 APP 的使用过程

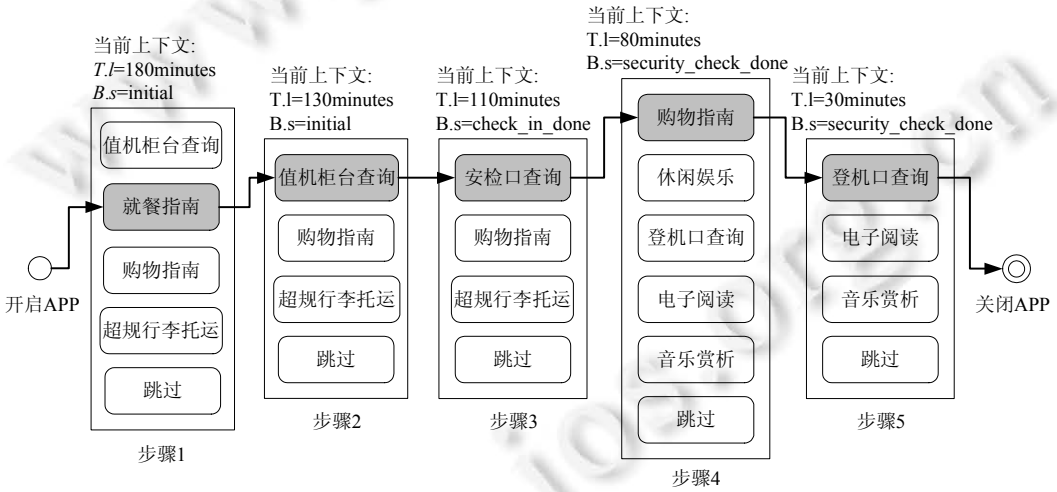

Fig.10 Usage procedure of AirportAssistant APP in scenario two

图 10 场景 2 中机场导航 APP 的使用过程

通过上述两个应用场景的案例可以看出:同样是对用户完成机场导航的任务,由于用户不同,应用场景不同,应用最终组合的服务也不相同.场景 1 中,旅客甲对机场设施非常熟悉,同时受时间的约束,需要的服务较少,应用为旅客甲实际组合的服务为{电子阅读,音乐赏析}.场景 2 中,旅客乙对机场设施并不熟悉,且有充裕的等候时间,因此需要应用提供较多的服务,例如值机柜台\安检口\登机口的查询服务以及就餐和购物时的指南服务等.应用为旅客乙实际组合的服务为{就餐指南,值机柜台查询,安检口查询,购物指南,登机口查询}.此外,在用户使用应用的每个阶段,应用都会以服务列表的形式为用户提供在当前上下文环境下最可能选择的服务.用户无需面对大量冗余无关的服务,使服务的选择和使用变得更加简捷和高效.

3.2 实验方案设计

为了验证本文所提出的 FP-tree 扩展算法比传统的 FP-tree 算法在关联规则挖掘方面具有更强的实用性,即,根据关联规则产生的候选服务集合对用户选择服务具有更高的准确率和命中率,本文设计了以下实验方案:

(1) 实验数据准备

本文采用人工干预与随机生成相结合的方式自动生成 100 条用户在不同场景下选择不同服务的历史记录,作为关联规则挖掘的数据基础.每一条记录都由一组上下文(*C*)及一组服务(*S*)组成,表示用户在 *C* 环境下选择了 *S* 中的服务.上下文 *C* 按照第 2.1 节给出的上下文模型来定义,其值在用户设定的取值范围内随机生成.为不失数据真实性,用户年龄 *U.a* 在[20,50]区间赋予较大权重,当前时间 *T.c* 在[5:00,22:00]区间赋予较大权重,使这两个区间的数据产生的概率较大.服务 *S* 以人工的方式,根据每条记录中随机生成的上下文 *C*,逐条标注.

(2) 实验方法

将 100 条实验数据按照 9:1 的比例随机分为两组:一组用作训练集,用于挖掘上下文与服务间的关联规则,构建 FP-tree 树;另一组用作测试集,用于验证所挖掘的关联规则的有效性,即,运用关联规则匹配服务构造可组合候选服务集合的准确率和命中率.其中:准确率是指依据关联规则产生的候选服务集合中,真正符合用户预期的服务的个数与全部候选服务个数的比值;命中率是指候选集中,符合用户预期的服务个数与用户预期的全部服务的个数的比值.计算公式如下:

- 准确率:

$$Precision = (X \cap Y) \big/ X \tag{1}$$

- 命中率:

$$Recall = (X \cap Y) \big/ Y \tag{2}$$

其中,*X* 为依据关联规则产生的候选服务集合,*Y* 为用户实际期望的服务集合.

(3) 实验环境

本文实验硬件环境配置为 Intel(R) Core(TM)2 Duo CPU T9400@2.53GHz 处理器,2GB 内存,Windows 7 旗舰版 32 位操作系统;软件环境配置为 eclipse3.4-jee-ganymede 开发运行平台,JDK7.0 版本.

3.3 实验结果与分析

图 11(a)为采用传统的 FP-tree 算法在最小支持度取不同值时,算法产生的关联规则数.可以看出:最小支持度设置的越高,产生的规则越少.这是因为最小支持度取较大值,意味着有更多的项被当做非频繁项而被过滤掉,使得频繁项的数量较少,因而产生的规则也相对较少.

图 11(b)展示了采用不同最小支持度时,依据关联规则进行服务匹配所获得的准确率和命中率(重复 10 次,取平均值).随着最小支持度的不断增加,算法产生的规则逐渐减少,许多应用场景因为缺乏可用的规则,使得服务匹配的准确率和命中率双双下降.然而,为了追求较高的准确率和命中率而一味降低最小支持度的做法也是不可取的,因为过低的最小支持度会导致产生数目过于庞大的规则,这将为规则的更新和查询都带来沉重的负担,不仅增加了服务匹配时的计算量,还占据了更多的内存空间,对于计算资源和存储资源都非常敏感的移动智能终端而言是十分不可取的.

为了平衡产生的规则数与服务匹配的准确率和命中率,本文对传统的 FP-tree 算法进行了扩展,对多个上下文属性分别设置最小支持度:对出现频次高的上下文属性设置较大的最小支持度,以此控制频繁项的个数,进而达到缩减规则数的目的;相反,对出现频次低的上下文属性设置较小的最小支持度,用以确保那些出现次数少但对服务匹配起到关键作用的项得以保留,增加规则的有效性,进而提高服务匹配的准确率和命中率.

图 11(c)、图 11(d)为算法扩展前后,随机选取 10 个用户的 10 个应用场景进行服务匹配的准确率和命中率.其中:算法扩展前的最小支持度取 0.2,得到 54 条规则;算法扩展后,最小支持度设置为 Min*Sup*={*U.g*:0.4,*L.d*:0.35,*B.s*:0.35,*T.c*:0.3,*U.a*:0.25,*U.i*:0.2,*T.l*:0.1},得到的规则数为 45,少于原来的 54,但却取得较好的准确率和命中率.在

实验的 10 个用户中,有 4 个用户的匹配准确率和命中率得到显著提高,另外 6 个与原来的比率持平.平均准确率由扩展前的 26.4%提高到扩展后的 39.8%,平均命中率由扩展前的 60%提高到扩展后的 90%.

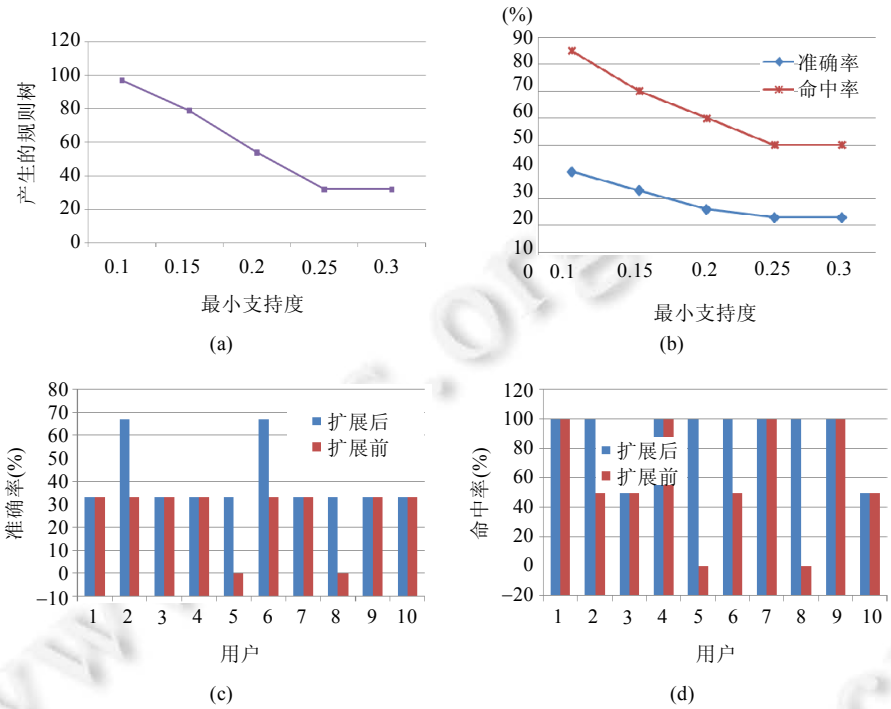

Fig.11 Results of experiments

图 11 实验结果

影响算法有效性的参数除了最小支持度,还包括事务数据库中事务的多少以及算法 Top-k 中 k 的取值,即,提供多少个候选服务给用户(如图 12 所示,其中,准确率和命中率均为同等条件下重复 10 次取平均值的结果).

图 12(a)展示了当 Top-k 取 3 时,服务匹配的准确率和命中率随历史事务数增长的变化情况.

显而易见,事务数据库中事务的个数越多,表明用于挖掘的数据资源越丰富,因而得到的关联规则也越多、越准确,对服务匹配的帮助也就越大.

图 12(b)展示了 Top-k 对服务匹配准确率和命中率的影响:

- 当 k=3 时,在本文实验的数据集上能够取得较为理想的准确率和命中率;
- 随着 k 值的不断增大,服务匹配的准确率不断降低.这是因为在准确率的计算公式(1)中,k 作为分母,其值越大,得到的比值越小.虽然 k 值增大使得用户需要的服务被命中的概率也增大了(图中表现为命中率随 k 值增大而增大),使得公式中分子部分也有增加的可能,但这种增加是不确定的,并且在命中率已经很高的情况下,分子部分的提升空间有限,因此整体表现为准确率随 k 值的增大而降低;
- 当 $k\geqslant 6$ 时,准确率和命中率都维持在一个特定值不变.这是因为:此时,算法产生的全部候选服务的个数小于或等于 k,对候选服务集合取 Top-k 个服务的结果就是候选服务集合本身.也就是说,当 $k\geqslant 6$ 时,不论 k 取何值,公式(1)中的分子和分母都不会改变.其中,分母为算法所能产生的全部候选服务的个数,分子为上述候选服务中用户实际需要的服务的个数;
- 当 $k<3$ 时,虽然准确率有所提升,但命中率却显著下降.这是由于 k 值过小,意味着候选服务列表中的服务个数过少,使得许多用户需要的服务未能被提供给用户.由公式(2)可知,用户实际需要的服务是确定的,即:分母不变,分子减小将导致整个分数值,即命中率的减小.

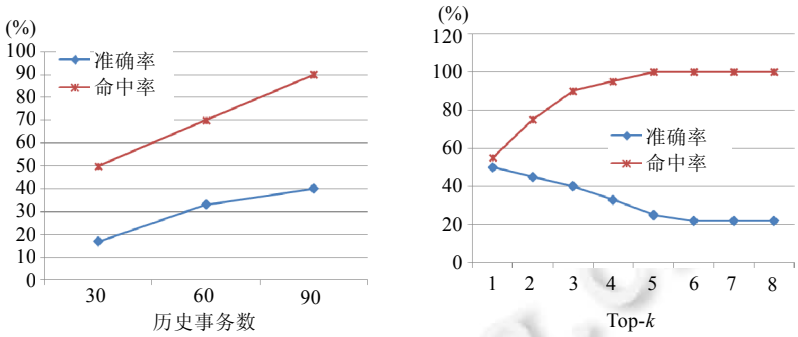

Fig.12 Influence on the results of experimental parameters

图 12 实验参数对结果的影响

4 相关工作比较

探索式服务组合采用边试探边组合的方式,在应用的执行过程中组合服务,适用于过程或结果不可预测或用户需求不确定的应用场景.

Bouillet 等人[14−17]面对模糊且不确定的用户需求,提出一种适用于信息处理应用的交互式的半自动化服务组合方法,利用语义标签 tag 对服务的输入、输出进行标记,并基于标签匹配规划服务流程.整个规划过程以一种迭代的人机互动的方式进行:首先,用户以 tag 标签的方式选择自己的初始需求,服务组合引擎根据该需求规划可能的服务流程,并将全部可能的运行结果仍然以 tag 标签的方式反馈给用户,用户从中选择自己感兴趣的结果标签,明确服务组合的方向,即丰富自己的需求描述,然后进入下一轮迭代过程.

韩燕波等人[10,18]面向科研协作、远程医疗、城市应急等应用领域,针对其过程不确定、结果难预测的特点,提出一种支持最终用户探索式组合服务的方法,将服务抽象为用户可理解的业务积木,业务积木之间通过一种类似于文本超链接的方式进行组合.为了在组合过程中向用户即时推荐可用的业务积木,提供了两种方法:

- 一种是对业务积木的输入输出参数进行语义标注,然后通过本体匹配建立业务积木之间的关联,以此向用户推荐当前可用的业务积木;
- 另一种是利用已构造应用中服务间的跳转关系建立业务积木间的转移概率矩阵,然后基于跳转概率推荐下一步可用的业务积木.用户通过选择业务积木,并配置业务积木间的链接关系,完成应用的探索式构造过程.

刘譞哲等人[19−22]将用户与服务间的按需交互建模为服务社区,利用社区的概念帮助用户完成服务发现及探索式组合的过程.在服务社区中,功能相近的服务被抽象为服务池,对用户提供统一、一致的资源视图,而服务的组合历史则被抽象为任务模版,用于指导用户的组合过程.当不存在满足需求的任务模版时,则利用服务的 tag 信息,通过计算服务间输入输出参数的相似度,构造 tag 的有向无环图,然后基于图进行规划找出所有可选的路径供用户选择.

可以看出:上述工作大多采用基于流程的探索式服务组合方法,即,通过对服务的输入输出参数进行基于语法和语义的匹配,构造出用户下一步可选的服务集合,帮助用户完成探索式的服务组合过程.虽然文献[10]同时提出了一种基于服务历史跳转的组合方法,但这种方法没有考虑上下文对跳转的影响.而在移动应用领域中,上下文对于服务的选择和组合却起着非常关键的作用.因此,这种方法不适合直接应用在移动应用场景中.

与上述基于流程的探索式服务组合不同,本文研究的是一种上下文感知的探索式服务组合方法,根据用户当前所处的上下文决定候选服务集合.

上下文感知的服务发现是上下文感知服务组合中的关键环节.在相关研究中,以文献[13]与本文工作最为接近,都是应用 FP-tree 算法挖掘上下文与服务间的关联关系,并以此作为服务选择与组合的依据.不同的是:虽

然文献[13]将上下文定义为一个二维的概念,即,由情景信息及其取值共同构成一个情景事件,但在关联规则挖掘过程中并没有考虑二维项所带来的问题,而是将二维项等同于一维项对待,对所有的项定义了统一的最小支持度;本文方法则考虑了二维项所带来的项的出现频次不均衡的问题,对不同属性的项定义不同的最小支持度,使出现频次小的项也能够保留在规则中,提高了规则的有效性,使基于规则的服务匹配更加准确.此外,本文在匹配过程中的服务查找和发现方面较文献[13]也有所加强,不仅能够查找到完全符合当前上下文的服务,而且能够发现那些与当前上下文近似的路径中所包含的服务,并且通过对不同路径进行等级划分和权重计算区分隶属于不同路径的服务,使服务选择的范围更加广泛,同时又不失准确性.

类似的工作还包括文献[23,24],作者首先对服务进行分类,对同一类型的服务采用相同维度的上下文进行描述,然后依据具体的上下文信息对服务建立树状索引,并基于该索引树实现服务查找.该工作关注的是当用户需要某一类服务时,如何根据用户所处的上下文,选择一个具体的服务;而本文侧重的是当用户不确定需要哪一类服务时,如何根据当前的上下文,帮助用户找出一个可操作的服务类型,通过服务组合实现用户对应用的某个期望目标.因此,本文不是对服务类型进行上下文建模,而是对某一类应用的上下文进行建模,并且在树结构中不仅表达出上下文与服务间的关联关系,而且体现出不同的上下文对于服务选择所起到的不同作用,对不同的应用场景识别出关键上下文,同时过滤掉无关上下文.换言之,本文的树结构中,路径的每一个分支节点(根节点除外)都是刻画其所属应用场景的典型上下文,这就为基于上下文的服务发现提供了更加准确、可靠的依据.

文献[25]从另外一个视角出发,构造用户的待办事项列表.与本文主动服务查找不同,该文作者首先构造一个理论上可行的活动集合,然后结合上下文,按照一定的规则从初始活动集合中逐渐删减不满足条件的活动,例如已经完成的活动、不可能完成的活动、冲突的活动等,并将剩余的活动按照相关性进行排序.该方法通过对活动集合做减法,实现对用户活动的管理和规划,更适用于时间管理类应用.

5 结 论

本文分析了当前移动应用中普遍采用的静态的服务分类聚集的组合方式,针对其可能引起的功能过载,继而影响用户使用效率的问题,提出一种探索式的服务组合方法.该方法在用户使用应用的过程中,通过感知用户所处上下文的变化捕捉用户的即时性需求,然后以动态的方式逐步添加用户需要的服务组合到应用中.为了方便用户选择和使用服务,本文提出一种基于关联规则的服务匹配算法,通过对用户使用服务的历史记录进行统计和挖掘,得出上下文与服务间的关联关系,并基于该关系匹配可组合的候选服务.在关联规则挖掘方面,本文对传统的 FP-tree 算法进行了扩展,避免了传统算法中单一最小支持度在二维数据项的挖掘过程中所造成的稀少数据项问题.实验表明,扩展后的算法在服务匹配的准确率和命中率方面比扩展前都有显著提高.

目前,本文所提出的探索式服务组合完全依靠用户所处的上下文环境推测用户下一步可能需要的服务,下一步工作考虑将目标规划引入探索式的服务组合过程,利用用户目标明确探索的方向,同时约束探索的范围,提高探索式服务组合的效率.此外,在关联规则挖掘方面,将考虑不同上下文在不同场景下的重要性,对不同属性的上下文项赋予不同的权重,进一步提高关联规则的有效性,进而提高服务匹配的准确率和命中率,提高探索式服务组合的可用性和有效性.

References:

[1] Ardagna D, Pernici B. Adaptive service composition in flexible processes. IEEE Trans. on Software Engineering, 2007,33(6): 369−384. [doi: 10.1109/TSE.2007.1011]

[2] Alrifai M, Risse T, Nejdl W. A hybrid approach for efficient Web service composition with end-to-end QoS constraints. ACM Trans. on Web, 2012,2(7):31. [doi: 10.1145/2180861.2180864]

[3] Nanda MG, Chandra S, Sarkar V. Decentralizing execution of composite web services. In: Proc. of the 19th Annual ACM SIGPLAN Conf. on Object-Oriented Programming, Systems, Languages, and Applications (OOPSLA 2004). New York: ACM Press, 2004. 170−187. [doi: 10.1145/1028976.1028991]

[4] Ngu AHH, Carlson MP, Sheng QZ, Paik HY. Semantic-Based mashup of composite applications. IEEE Trans. on Services Computing, 2010,3(1):2−15. [doi: 10.1109/TSC.2010.8]

[5] Yu J, Benatallah B, Saint-Paul R, Casati F, Daniel F, Matera M. A framework for rapid integration of presentation components. In: Proc. of the 16th Int'l Conf. on World Wide Web (WWW 2007). New York: ACM Press, 2007. 923−932. [doi: 10.1145/1242572.1242697]

[6] Greenshpan O, Milo T, Polyzotis N. Autocompletion for mashups. Proc. of the VLDB Endow., 2009,2(1):538−549. [doi: 10.14778/1687627.1687689]

[7] Janner T, Siebeck R, Schroth C, Hoyer V. Patterns for enterprise mashups in B2B collaborations to foster lightweight composition and end user development. In: Proc. of the IEEE Int'l Conf. on Web Services. 2009. [doi: 10.1109/ICWS.2009.46]

[8] Zhao DY, Guan DS. Taste Mobile Design—iOS, Android, Windows Phone Best Practices for User Experience Design. Beijing: Posts & Telecom Press, 2013 (in Chinese).

[9] Agrawal R, Imieliński T, Swami A. Mining association rules between sets of items in large databases. SIGMOD Record, 1993, 22(2):207−216. [doi: 10.1145/170036.170072]

[10] Han YB, Wang HC, Wang JW, Yan SY, Zhang C. An end-user-oriented approach to exploratory service composition. Journal of Computer Research and Development, 2006,43(11):1895−1903 (in Chinese with English abstract).

[11] Abowd GD, Dey AK, Brown PJ, Davies N, Smith M, Steggles P. Towards a better understanding of context and context-awareness. In: Proc. of the 1st Int'l Symp. on Handheld and Ubiquitous Computing (HUC'99). London: Springer-Verlag, 1999. 304−307. [doi: 10.1007/3-540-48157-5_29]

[12] Meta Object Facility (MOF) Core Specification. OMG Available Specification. Version 2.0., 2006.

[13] Mo T, Chu WJ, Li WP, Wu ZH, Lin HP. Active service discovery method based on context aware event. Ruan Jian Xue Bao/Journal of Software, 2011,22:41−51 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/11025.htm

[14] Riabov AV, Bouillet E, Feblowitz MD, Liu Z, Ranganathan A. Wishful search: Interactive composition of data mashups. In: Proc. of the 17th Int'l Conf. on World Wide Web. Beijing: ACM Press, 2008. [doi: 10.1145/1367497.1367602]

[15] Bouillet E, Feblowitz M, Liu Z, Ranganathan A, Riabov A. A tag-based approach for the design and composition of information processing applications. In: Proc. of the 23rd ACM SIGPLAN Conf. on Object-Oriented Programming Systems Languages and Applications. Nashville: ACM Press, 2008. [doi: 10.1145/1449764.1449810]

[16] Bouillet E, Feblowitz M, Liu Z, Ranganathan A, Riabov A. A faceted requirements-driven approach to service design and composition. In: Proc. of the IEEE Int'l Conf. on Web Services. 2008. [doi: 10.3969/j.issn.1672-822X.2008.06.011]

[17] Bouillet E, Feblowitz M, Feng HH, Liu Z, Ranganathan A, Riabov A. A folksonomy-based model of Web services for discovery and automatic composition. In: Proc. of the IEEE Int'l Conf. on Services Computing. 2008. [doi: 10.3969/j.issn.1672-822X.2008.06.011]

[18] Zhang C. Personalized service recommendation for assembling service-oriented applications [Ph.D. Thesis]. Beijing: Institute of Computing Technology, the Chinese Academy of Science, 2006 (in Chinese with English abstract).

[19] Liu XZ, Huang G, Mei H. A community-centric approach to automated service composition. Science in China Series F: Information Sciences, 2010,53(1):50−63. [doi: 10.1007/s11432-010-0013-0]

[20] Liu XZ, Zhao Q, Huang G, Jin Z, Mei H. iMashup: Assisting end-user programming for the service-oriented Web. In: Proc. of the IEEE/ACM Int'l Conf. on Automated Software Engineering. 2010. [doi: 10.1145/1858996.1859052]

[21] Liu XZ, Huang G, Mei H. A user-oriented approach to automated service composition. In: Proc. of the 2008 IEEE Int'l Conf. on Web Services. 2008. [doi: 10.1109/ICWS.2008.139]

[22] Liu XZ, Huang G, Mei H. Consumer-Centric service aggregation: method and its supporting framework. Ruan Jian Xue Bao/Journal of Software, 2007,18(8):1883−1895 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/18/1883.htm [doi: 10.1360/jos181883]

[23] Doulkeridis C, Vazirgiannis M. CASD: Management of a context-aware service directory. Pervasive Mobile Computing, 2008,4(5):737−754. [doi: 10.1016/j.pmcj.2008.05.001]

[24] Doulkeridis C, Vazirgiannis M. Querying and updating a context-aware service directory in mobile environments. In: Proc. of the 2004 IEEE/WIC/ACM Int'l Conf. on Web Intelligence (WI 2004). Washington: IEEE Computer Society, 2004. 562−565. [doi: 10.1109/WI.2004.10023]

[25] Driver C, Clarke S. An application framework for mobile, context-aware trails. Pervasive Mobile Computing, 2008,4(5):719−736. [doi: 10.1016/j.pmcj.2008.04.009]

附中文参考文献:

[8] 赵大羽,关东升.品味移动设计:IOS,Android,Windows Phone 用户体验设计最佳实践.北京:人民邮电出版社,2013.

[10] 韩燕波,王洪翠,王建武,闫淑英,张程.一种支持最终用户探索式组合服务的方法.计算机研究与发展,2006,43(11):1895−1903.

[13] 莫同,褚伟杰,李伟平,吴中海,林慧苹.基于情境感知事件的主动服务发现方法.软件学报,2011,22:41−51. http://www.jos.org.cn/1000-9825/11025.htm

[18] 张程.面向服务环境中服务的个性化推荐[博士学位论文].北京:中国科学院计算技术研究所,2006.

[22] 刘譞哲,黄罡,梅宏.用户驱动的服务聚合方法及其支撑框架.软件学报,2007,18(8):1883−1895. http://www.jos.org.cn/1000-9825/18/1883.htm [doi: 10.1360/jos181883]

白琳(1980－),女,河北石家庄人,博士生,主要研究领域为网络分布式计算,软件工程.

魏峻(1970－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.

黄翔(1982－),男,博士后,主要研究领域为网络分布式计算,软件工程,大数据.

叶丹(1971－),女,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.

黄涛(1965－),男,博士,研究员,博士生导师,CCF 高级会员,主要研究领域为网络分布式计算,软件工程.