基于移动设备的个人云服务及资源 Mashup

王欢欢 彭 鑫 赵文耘

(复旦大学软件学院 上海 201203) (上海市数据科学重点实验室 上海 201203)

摘 要 个人用户家庭或工作环境中的智能手机、个人电脑、智能相册等通过网络形成个人云,为个人的数据处理和共享带来了便利。然而,如何面向具体应用需求为用户提供方便、高效的个人云服务及资源整合仍然是一个亟待解决的问题。为此,提出了一种基于移动设备的个人云服务及资源 Mashup 框架来统一管理设备和描述服务及资源,并阐述了如何通过框架合理构建 Mashup 应用;此外,开发了基于 Agent 的框架实现以及 Android 客户端。基于该框架及客户端,用户可以通过整合个人云服务及资源方便、快捷地构造个人化应用。通过一个用户案例研究对所提出的 Mashup 框架及其实现的有效性和易用性进行了评估。

关键词 个人云,移动,资源,服务,主体,混搭

中图法分类号 TP311 **文献标识码** A **DOI** 10.11896/j.issn.1002-137X.2016.6.033

Towards Personal Cloud Service and Resource Mashup Based on Mobile Devices

WANG Huan-huan PENG Xin ZHAO Wen-yun

(Software School, Fudan University, Shanghai 201203, China)

(Shanghai Key Laboratory of Data Science, Fudan University, Shanghai 201203, China)

Abstract Personal cloud is a collection of smart phones, personal computers, and smart albums which are seamlessly accessible through network in the surrounding environment of ordinary users. It benefits both the personal data processing and sharing. However, how to provide the users with convenient and efficient personal cloud service and resource integration based on different practical requirements is still a problem demanding prompt solution. To address this problem, we proposed a framework towards personal cloud service and resource mashup based on mobile devices, including how to manage devices, and how to define service and resource. We also explained how to build a Mashup application through the framework. We developed an agent-based implementation for the framework and an Android client to help users complete mashup. Finally, we evaluated the effectiveness and usability of our Mashup framework and Android clientby a preliminary experimental study.

Keywords Personal cloud, Mobile, Resource, Service, Agent, Mashup

1 引言

随着信息技术的迅猛发展以及互联网、物联网的普及,各种智能设备(智能手机、智能相册等)层出不穷。与传统个人电脑相比,这类智能设备在计算能力不逊于之的情况下,更具便携性、多元性、个性化等特点。与此同时,在云计算技术不断成熟的背景下,面向个人用户的个人云(Personal Cloud)[1,2]概念也应运而生。目前没有针对个人云的权威定义,最受认可的是美国 IT 研究与顾问咨询公司 Gartner 关于个人云的描述,即个人云是指可以借助智能手机、平板电脑、电视和 PC,通过网络无缝存储、同步、获取并分享数据的一组在线服务[3,4]。与各种大型云计算不同,个人云在便利、私密性等方面具有更大优势。新兴的个人云计算服务将成为连接各种设备的纽带,将用户从传统的计算方式中解放出来,让用户可以在日常生活的不同时刻依需选择不同设备。

普通用户的个人云中,可以存在智能手机、个人电脑、智能相册、家庭打印机等各种类型的设备。个人云为用户的数据处理和共享带来了便利,设备所具有的各种类型的计算资源和处理能力也能够为用户提供多样化的服务。但是,某些场景下,用户需要同时利用多个设备的服务及资源来满足某些需求。例如,用户想要将新拍的一张照片传到电脑上处理,然后将美化后的照片显示在智能相册上。用户在日常生活的不同场景中会有类似的各种不同类型的应用需求。而用户个人云中设备各不相同、服务及资源存在各种异构问题,如何面向具体应用需求为用户提供方便、高效的个人云服务及资源整合仍然是一个亟待解决的问题。而 Mashup 正是一种新型的基于 Web 的数据集成应用程序,它利用已有的各种 Web 应用所提供的 API 接口将实现不同功能的应用整合混搭成

到稿日期:2016-01-13 返修日期:2016-02-29 本文受国家高技术研究发展计划(863)(2015AA01A203)资助。

王欢欢(1990-),女,硕士生,主要研究方向为软件自适应、移动计算,E-mail:huanhuanwang13@fudan.edu.cn;**彭 鑫**(1979-),男,副教授,博士生导师,主要研究方向为软件维护与演化、软件产品线、自适应软件、移动计算与云计算等;**赵文耘**(1964-),男,教授,博士生导师,主要研究方向为软件工程、软件开发工具及其环境、企业应用集成(EAI)等。

一种新的综合应用提供给用户更加丰富的服务体验[5]。我们可以使用 Mashup 技术来进行服务及资源整合。

针对上述问题，本文提出了一种基于移动设备的个人云服务及资源 Mashup 框架，解决了设备统一管理、消除服务及资源之间的异构等难点问题，并阐述了如何通过框架合理构建 Mashup 应用。在此基础上，开发了基于 Agent 的框架实现以及 Android 客户端。用户完成框架部署后，通过 Android 客户端可以方便、快捷地管理各种设备，整合个人云服务及资源，构造满足不同需求的个人化应用。同时，通过一个简单的真实用户案例研究，验证了所提 Mashup 框架及其实现的有效性和易用性。

本文第 2 节介绍了个人云和 Mashup 概念，并对相关工作进行了分析；第 3 节详细描述了服务及资源 Mashup 框架，包括设备统一管理、资源统一描述、服务及资源 Mashup 方法；第 4 节概述了基于 Agent 的 Mashup 框架的实现，着重阐述了 Android 客户端的开发与使用；第 5 节描述了案例研究的设计以及针对实验结果的分析；最后总结了全文，讨论了本文研究的实际应用前景，并对实际应用中可能出现的问题进行了预想与分析。

2 背景和相关工作

本节首先简要介绍个人云的概念以及目前关于个人云的研究工作，然后介绍 Mashup 技术的相关工作。

2.1 个人云

与传统的公有云、私有云、混合云面向大型企业不同，个人云主要面向个人用户提供服务。目前在此领域的研究中，已有针对利用个人云为用户提供服务的相关工作。文献[1]主要关注的是解决个人云(以用户为中心)中的基础设施挑战问题(数据存储、同步、传递、一致性管理)，提供了一个以消费者为中心的云门户入口，使用户设备可以无缝访问云服务。文献[2]则是将服务云转变为以用户为中心的个人云，提供了一个个人云平台，用来帮助管理云所提供给用户的服务。这两项工作主要关注个人云中服务的访问和管理，与文中的研究不尽相同。文中个人云是由用户周围环境中的设备构建而成，也可访问其它云提供的一些服务来丰富自身功能；同时，也会对个人云中的服务进行管理，以便更好地针对用户的不同应用需求提供服务及资源整合。

个人云是指各种设备通过网络连接提供存储、同步、获取并分享数据的服务。iCloud、SkyDrive、Google Drive 均是企业提供给用户的个人云存储产品，提供存储服务以及简单的文件同步服务；文献[6,7]的研究者在他们的研究中利用 Dropbox、OneDrive 等的数据深入探索了个人云存储服务，包括使用、性能和终端的影响。这些个人云大部分均是提供单一的数据存储、共享等功能，而且是对所有用户一视同仁，不能为用户提供个性化需求服务。文中所提出的服务及资源 Mashup 框架，是利用个人云中的服务及资源，让用户自己构建个人化应用，以满足不同场景下的功能需求，从而更好地使用个人云。

2.2 Mashup

Mashup 是将不同应用混搭到一起组成新应用的技术。到目前，关于 Mashup 的研究已比较成熟，Mashup 也不再单纯地指针对 Web 应用的组合。文献[8]讨论的是针对服务之间的 Mashup，主要研究了服务之间的异构性问题并提出了解决方案。文献[9]用 Mashup 扩展了现有的 SOA 模型来支持服务组合，并提出了一个 Mashup 模块模型和一个 Mashup 运行时设计。文献[10]中开发者开发的 iMashup 也是一款针对服务组合的 Mashup 工具。与这些工作相比，本文更关注在 Mashup 时，个人云中不同设备上服务或者资源之间的异构问题；另外，我们进行的 Mashup 不是单纯服务或者数据的 Mashup，而是在服务和资源(包括数据)之间进行混合性质的 Mashup。

此外，我们的 Mashup 研究是基于移动设备，故更关注针对移动端 Mashup 的研究。文献[11]研究了一款针对不同设备以及不同用户群组的利用 Mobile Mashup 的 WebApp 开发工具，通过使用他们的工具，普通用户和具有一定专业知识的专家用户可以开发出运行在不同设备(个人电脑、平板电脑、智能手机)的 WebApp。文献[12]则是研究跨平台的 Mashup 应用开发，使用他们的平台开发的 Mashup 应用可以适配不同的终端设备(桌面浏览器与各种手机终端)。与这些相比，我们不是在关注使 Mashup 适配移动设备，而是在移动设备上进行 Mashup，所构建的应用也不是只运行在一个设备上，而是涉及到多个设备；此外，移动设备上的服务及资源也是我们 Mashup 的数据来源之一。

3 服务及资源 Mashup 框架

本节首先介绍个人云中基于 Agent 的设备统一管理以及设备资源的统一描述，然后介绍服务及资源 Mashup 方法。

3.1 设备统一管理

针对个人云中包含的各种设备，用户可以选择启用或者不启用，也可以随时根据需要增加、移除设备。为了让用户方便、简洁地管理各种设备，提出了一种基于 Agent 的设备统一管理方法，如图 1 所示。

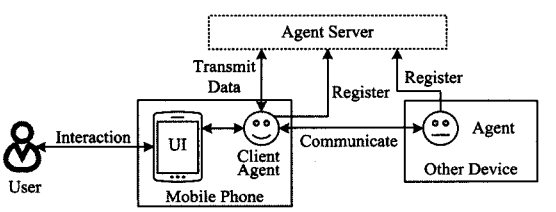


图 1 基于 Agent 的设备统一管理

在统一管理方法中，所有设备上均驻留一个 Agent 程序，Agent 的在线与否即代表宿主设备是否被启用。Agent Server 用来管理所有 Agent 的注册信息及在线状态，用户可以选择在个人云中计算能力较强的节点(如个人电脑)上搭建 Agent Server。首次启用设备时，Agent 将设备信息(包括型号、名称、附加描述)注册到 Agent Server 上，并将该设备状态更新为在线；当 Agent 停止运行后，宿主设备状态更新为离线，不可被访问。用户手机(Mobile Phone)上的 Agent 为 Client Agent，Client Agent 除向 Agent Server 注册设备信息外，还可以从 Agent Server 获取所有个人云中设备的信息，用户通过手机上的一个 UI 界面可以查看、操作所有设备。当用户通

过 UI 界面将一个新设备信息通过 Client Agent 发送给 Agent Server 后，表示新增一个设备，但只有当此新设备上 Agent 连接 Agent Server 成功后，才表示添加设备成功。当然，一个新设备 Agent 自动注册到 Agent Server 上，也表示自动添加设备成功，之后 Agent Server 会将新设备信息推送给 Client Agent。

所有 Agent 根据宿主设备操作系统的不同而相互异构，但为了保证 Agent 之间的通信，框架中的所有 Agent 必须符合 FIPA(The Foundation for Intelligent Physical Agents)组织制定的 Agent 软件标准。

3.2 服务、资源统一描述

在 Mashup 中，构成新应用的源应用种类繁多，实现也各不相同，但其大部分均提供一个开放 API 供 Mashup 开发者调用，其余部分也是可以利用屏幕截取、Web Feed 等方式获取数据[13]。在 Mashup 框架中，服务及资源来源是各种不同型号种类的设备，其操作系统、计算能力等各不相同，为了能够进行 Mashup，屏蔽不同设备间的服务及资源异构，针对服务及资源做了统一描述。

把设备的某种行为能力称为资源，服务则是可以满足某种特定需求的已有实现。比如，智能手机的“拍照”是一种资源，个人电脑上“处理照片(压缩、美化等)”是一种资源；而一些应用如天气预报、运动记录等则可看成是服务。把资源和服务统一描述成 Behavior。Behavior 是一种结构化定义，具体如下：

```
〈BehaviorType hasNeedData="" hasPostData=""〉
    〈name〉behavior name〈/name〉
    〈targetURL〉the url of Behavior API〈/targetURL〉
〈/BehaviorType〉
```

BehaviorType 描述了这个 Behavior 的种类，由提供 Behavior 的设备具体决定，比如，安卓手机上的 Behavior 可以描述为 AndroidBehavior，个人电脑上的 Behavior 可以描述为 PCBehavior。BehaviorType 决定了设备在具体调用 Behavior 时采用的方法。hasNeedData 和 hasPostData 可以取值为 true 或者 false，分别表示在调用这个 Behavior 时是否需要传入数据，以及在调用结束后是否会传出数据。name 为 Behavior的名称，简单描述该 Behavior 可以满足什么样的需求。targetURL 具体确定了该 Behavior 的实现位置，可以是具体的 API 接口，也可以是访问 Web Service 的 WSDL 地址等，总之，可以通过 targetURL 直接调用到想要访问的具体服务或者资源。

每个设备上有多个服务或者资源，文中使用一个 Behavior Base 存储所有的 Behavior。用户可以通过一个 UI 界面查看到每种设备上的 Behavior Base，继而在进行服务及资源 Mashup 时挑选出满足需求的 Behavior。

3.3 服务及资源 Mashup

在一般的 Mashup 中，产物是混合各种功能的新应用。在所提 Mashup 框架中，Mashup 的产物也是一种满足用户个性化需求的个人应用，用 Program 表示这种应用。本节具体介绍了用户对 Program 的构建、管理以及执行。

3.3.1 Program 的构建及管理

用户对服务及资源进行 Mashup 的过程就是构建 Program 的过程。Program 是一个可被用来运行的文件，其中包含了用户选择的用来满足需求的一系列 Behavior。目前框架只支持 Behavior 之间的顺序调用，不支持并行调用。图 2 具体展示了用户构建 Program 的过程以及针对 Program 的管理。

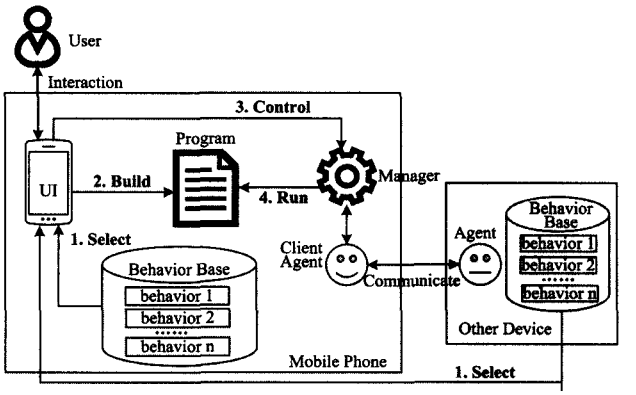


图 2 Program 的构建及管理

如图 2 所示，在所提框架中，各设备中均有 Agent 及 Behavior Base。除此之外，Mobile Phone 上具有 UI 操作界面、Manager 模块以及服务及资源 Mashup 后的产物 Program。用户在确定自己的需求后，可以通过 UI 界面查看到所有设备上的 Behavior Base，然后根据需求在界面上选择合适的 Behavior(图中的步骤 1. Select)；之后，用户在 UI 界面上确定各个 Behavior 的调用顺序后，构建成一个 Program(图中的步骤 2. Build)，构建生成的 Program 以文件形式保存在 Mobile Phone 上。用户可以对构建的 Program 进行各种修改、删除操作，这个操作需要通过调用 Manager 模块(图中的步骤 3. Control)。Manager 模块负责加载、解析 Program，将在 3.3.2 节中具体介绍。最后，用户通过相关 UI 界面操作 Manager，运行 Program(图中的步骤 4. Run)。

在用户构建 Program 的过程中，需要决定各个 Behavior 的调用顺序，这之间会存在数据依赖的问题。比如，“拍照”的 Behavior 必须在“处理照片”的 Behavior 之前。当两个 Behavior之间没有数据依赖时，用户可以根据喜好随意决定两个 Behavior 之间的调用顺序。

3.3.2 Program 的执行

用户根据需求完成 Program 的构建后，可以通过 UI 界面操控 Manager 来运行 Program。从用户角度看，图 2 中步骤 4 只需要一个简单的 UI 操作，但从框架内部分析，这其中涉及到 Manager 与 Client Agent 之间的交互，以及 Client Agent 与其它设备上 Agent 的通信。图 3 是 Program 执行过程的活动图。

从图 3 的活动图中可以分析出，用户通过 UI 界面选择 Start Program 后，Manager 会收到启动指令，之后根据指令加载并解析 Program 中所包含的各个 Behavior，按照顺序直接开始执行调用第一个 Behavior。Manager 会把首个 Behavior的信息(Behavior 结构化定义中包含的信息)及可能需要的相关数据发送给 Client Agent，Client Agent 根据获取到的信息来确定这个 Behavior 是在本机执行还是发送到其他设备上执行。若是在本机执行，直接通过 targetURL 调用执行

具体的服务或资源，获得执行结果（成功或者失败）然后反馈给 Manager；若是在其他设备执行，则 Client Agent 将 Behavior 的相关信息通过 Agent 之间的通信（图 2 中的 Communicate）发送给目标设备上的 Agent，目标设备上的 Agent 之后根据信息调用执行对应的 Behavior，并将执行结果（成功或者失败）返回给 Client Agent，Agent 再将结果反馈给 Manager。如此，首个 Behavior 的执行过程完成。开始执行 Program 中的下一个 Behavior。Manager 只有在收到上一个 Behavior 执行成功的结果后才会开始下一个 Behavior 的执行；否则，会判断 Program 执行失败，不再继续执行后续的 Behavior。在 Program 所包含的全部 Behavior 均被成功执行完毕后，整个 Program 执行成功，用户的需求目标被满足。用户可以随时通过 UI 界面查看 Program 的执行进度，也可以查看最终的执行结果，在完成一个 Program 的执行后，用户可以选择执行新的 Program，也可以选择重新执行之前的 Program，或者选择暂时不再执行 Program。

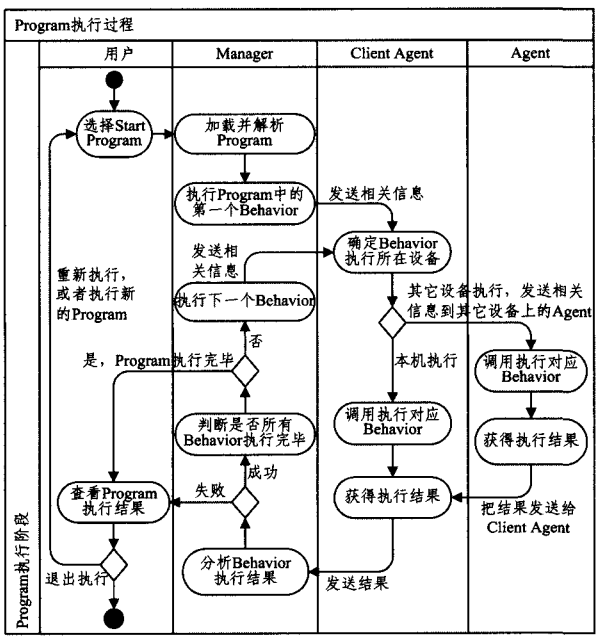


图 3　Program 执行过程的活动图

4　基于 Agent 的 Mashup 框架实现

针对所提出的 Mashup 框架做了基于 Agent 的实现，本节介绍了具体的实现过程，包括 Agent 的开发、不同设备上 Behavior 的具体封装实现，以及用户用来完成 Mashup 过程所依赖的安卓客户端的开发。

4.1　Agent 的开发实现

本文所针对个人云中所有设备上的 Agent 做的实现均是基于 JADE（Java Agent Development framework）平台开发的。JADE 是一个提供了基本中间层功能的软件平台，这些功能独立于具体应用，简化了基于软件 Agent 抽象的分布式应用实现[14]。JADE 符合 FIPA 标准，同时，JADE 除提供 Java环境下的 Agent 开发支持外，还支持在 Android 系统的移动设备上运行 Agent。

图 1 中的 Client Agent 是基于 JadeAndroid 完成的开发。JadeAndroid 是一个为了满足 Android 需求的 JADE 特殊版本，只需要在 Android 应用中导入 JadeAndroid. jar 包即可使应用支持 Agent 的各种功能实现。Client Agent 被实现后，作为一个 Android Service 运行在手机上，它与 Manager 的交流通过 JADE 提供的 O2AInterface 实现，与其它设备上的 Agent通信通过 ACL（Agent Communication Language）实现。其他设备上的普通 Agent 基于正常的 JADE 完成开发，它们作为普通的 Java 线程运行在配置了 Java 环境的设备上，之间的通信同样通过 ACL 实现。

此外，将 JADE 平台运行后的 Main Container 作为 Agent Server 使用，所有设备上的 Agent 在创建自身时会首先创建一个容纳自己的 Container，然后在 Main Container 上注册，此时会将设备的相关信息注册到 Agent Server 上。

4.2　Behavior 的统一包装开发

在目前的实现中，主要针对用户生活中常见的一些设备上的服务、资源做了 Behavior 的统一包装开发，具体包括 Android 手机、PC、智能相册、打印机、网络云盘。这些设备上的 Behavior 均根据 3.2 节中的结构化定义描述，BehavoirType 分别为 AndroidBehavior、PCBehavior、SABehavior、PrinterBehavior、CloudBehavior，而 hasNeedData 和 hasPostData 的取值以及 name 则根据具体不同的 Behavior 而定。targetURL 是可以调用到具体服务或资源实现的接口地址，主要可以分为 3 类。Android 上为 Activity 或者 Service 的完整包路径，调用 Behavior 时可以直接通过 startService（Intent）或者 startActivity（Intent）实现；普通设备上 targetURL 为 Java Class 完整路径，调用 Behavior 时通过 Java 类反射机制确定要执行的 Java Class；云存储之类的 targetURL 为 Web Service 的 WSDL 地址，调用 Behavior 时按照正常的调用 Web Service 的方式调用。

所有 Behavior 的具体实现保存在 Behavior Base 中，整个框架中还存在一个 BehaviorIndex. xml 文件，用来索引所有 Behavior。BehaviorIndex. xml 文件保存在 Mobile Phone 上，安卓客户端可解析此文件供用户查看所有 Behavior 的使用。图 4 是文件的一个样例截图。

```
〈?xml version="1.0" encoding="utf-8"?〉
〈BehaviorIndex〉
  〈AndroidBehavior hasNeedData="false" hasPostData="true"〉
      〈name〉TakePicture〈/name〉
      〈targetService〉service. intentService. takePicture〈/targetService〉
  〈/AndroidBehavior〉
  〈PCBehavior hasNeedData="true" hasPostData="true"〉
      〈name〉BeautyPic〈/name〉
      〈targetService〉cn. edu. fudan. se. computeragent. behaviour.
      BeautyPicture〈/targetService〉
  〈/PCBehavior〉
〈CloudBehavior hasNeedData="true" hasPostData="false"〉
      〈name〉SavePicture〈/name〉
      〈targetService〉cn. edu. fudan. se. cloudagent. behaviourbase.
      SavePicture
      〈/targetService〉
  〈/CloudBehavior〉
〈/BehaviorIndex〉
```

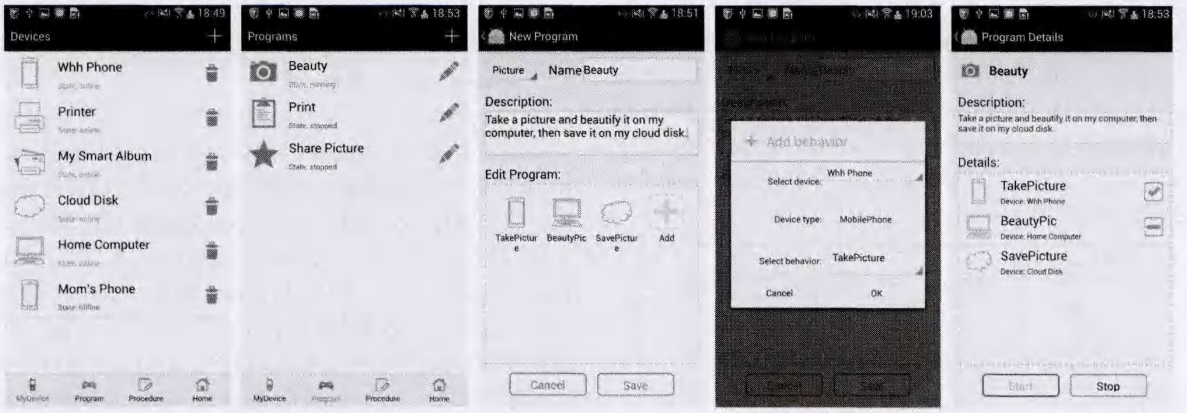
图 4　BehaviorIndex. xml 样例截图

4.3 安卓客户端的实现

为了用户可以方便、简单地进行服务及资源 Mashup 过程，开发了一个 Android 客户端，里面包含了所有 Mobile Phone 上的实现。除了已介绍过的 Agent 和 Behavior Base，本节介绍 Manager 和 UI 界面的开发及使用。

Manager 是一个 Android 线程，它维护着一个消息池，用 BlockingQueue 实现，主要用来接收来自 UI 的用户指令消息和来自 Agent 的 Program 执行消息。除此之外，Manager 中有解析 Program 文件的程序，并根据图 3 的活动图控制 Program 执行的过程。

客户端的 UI 界面主要包括：MyDevice 界面(见图 5(a))，列出了用户周围环境中的个人云、各种端设备；Program 界面(见图 5(b))，列出了用户构建的用来完成某项任务的程序；构建 Program 界面(见图 5(c))，显示了用户将各种资源 Mashup 的具体过程；添加 Behavior 界面(见图 5(d))，显示了在对资源进行 Mashup 时添加某个需要的 Behavior 的操作；Program 执行详情界面(见图 5(e))，列出了用户执行 Program 时的各个 Behavior 执行状态。



(a)MyDevice 界面　(b)Program 界面　(c)构建 Program 界面　(d)添加 Behavior 界面　(e)Program 执行详情界面

图 5　Android 客户端 UI 界面

在 MyDevice 界面中，用户可以查看个人云中所有设备以及它们当前的状态。online 表示在线，服务及资源可用，offline 表示离线；用户点击设备旁边的垃圾箱图标可删除设备，点击右上角的加号按钮可以添加设备，输入设备的相关信息后保存即可。在 Program 界面，用户可查看之前构建的所有 Program 信息以及它们是否正在被执行；点击右上角的加号按钮后会跳转到构建 Program 界面，左上角下拉框用来选择要构建的 Program 与什么类型相关(比如图片、文件、音频等)，Name 输入框中输入名字，Description 文本输入域中输入对这个 Program 的描述；Edit Program 部分是主要界面，用来挑选进行 Mashup 的资源，也就是用户根据自己的需求选择所需的 Behavior；用户在点击 Edit Program 下面虚线框中的加号图片后，会弹出一个悬浮框，可以选择要 Mashup 的 Behavior，选定后点击“OK”按钮会保存这个 Behavior，Edit Program 下面的虚线框中也会立刻添加刚刚的 Behavior；在完成挑选 Mashup 所需的所有 Behavior 后，点击“Save”按钮可保存整个 Program，并跳转到 Program 详情界面。详情界面用户可以开始这个 Program 的执行，并可以查看其中每个 Behavior 的执行状态，也可以点击“Stop”按钮终止这个 Program 的执行。

此外，图 5 中未显示的 Procedure 标签页为 Program 执行过程中的日志输出界面，用户可通过此界面查看 Program 执行的细节；Home 标签页显示的是用户个人的如用户名之类的一些基本信息。

5　案例研究

为了验证所提出的 Mashup 框架的可行性和所做的基于 Agent 的框架实现的有效性，以及所开发的 Android 客户端的易用性，搭建了一套实验系统，并寻找了志愿者参与实验，他们根据自己的需求进行服务及资源的 Mashup。

5.1　实验设计

首先构造了一个普通用户可能所处的设备环境，这些设备组成个人云，在这个环境中，有以下设备可供用户使用：

- 三星手机(MP1)。Android 系统，装有我们所开发的 Android 客户端，是用户自身持有的设备。
- 小米手机(MP2)。Android 系统，装有我们所开发的 Android 客户端，模拟被家人授权可访问的设备。
- ThinkPad 笔记本(PC)。用户在家庭中使用的个人电脑。
- 打印机(P)。可用来黑白打印文件。
- 智能相册(SA)。用另外一台个人电脑来模拟智能相册显示照片。
- 网络云盘(C)。使用一台服务器模拟网络云盘存储。

这些设备均处于开机、网络联通的可被访问的状态，设备上部署了 Mashup 框架的 Agent 实现。寻找了 5 个复旦大学软件学院的研究生作为实验对象，他们被告知所有设备的信息，包括位置分布、所具有的行为能力(即资源)，然后他们通过三星手机来操控所有的设备。实验时，他们自己设定自己的需求目标，然后利用这些设备对其所具有的资源进行 Mashup，构建 Program，来验证是否可达成他们设定的目标。

实验开始前，针对实验对象做了简单的客户端使用培训，告知他们基本的功能操作；实验开始后，每个实验对象分别独立扮演用户，通过手机上客户端来进行资源 Mashup，构建一个 Program，完成自己的需求目标；在实验结束后，通过调查

问卷的形式收集了用户针对整个框架的部署以及 Android 客户端的使用的反馈，用户通过多个维度(功能、易用性、用户界面等)对 Android 客户端进行了打分(1 分表示最差，5 分表示最优)，同时反馈了使用心得及使用过程中的优缺点。

5.2 实验结果分析

收集了每个用户使用 Android 客户端完成资源 Mashup 达成目标的过程信息，具体如表 1 所列。5 个不同用户根据所提供的设备，分别设定了不同的需求目标，然后选择所需设备上的资源进行 Mashup，构建出 Program，并执行了 Program。表 1 记录了用户构建的 Program 的简介、构建以及执行时用到的时间，最后一栏具体显示了这个 Program 所 Mashup 到的 Behavior 以及它们所在的设备。

表 1 用户使用 Android 客户端构建 Program 的结果

用户	Program 简介	构建是否达成目标(是/否)	构建 Program 所用时间(s)	Program 执行是否成功	Program 执行所用时间(s)	用到的设备数(个)	Behavior@所在设备
1	使用手机拍照并将照片保存到网络云盘上	是	43	是	23	2	TakePicture@MP1，SavePicture@C
2	使用手机拍照，然后在笔记本上美化，最后显示在智能相册上	是	36	是	26	3	TakePicture@MP1，BeautyPic@PC，ShowPic@SA
3	把智能相册上的照片分享到家人手机，然后打印出来	是	48	否	—	3	LoadPic@SA，ViewPic@MP2，Print@P
4	把智能相册上的照片显示在自己手机上，并打印出来，然后保存到网络云盘上	是	35	是	13	4	LoadPic@SA，ViewPic@MP1，Print@P，SavePicture@C
5	使用手机拍照，然后在笔记本上美化，分享到家人手机，显示到智能相册并进行打印，最后在网络云盘上备份	是	74	是	38	6	TakePicture@MP1，BeautyPic@PC，ViewPic@MP2，ShowPic@SA，Print@P，SavePicture@C

从表中可以看出所有的用户设置的 Program 均达成了他们的目标。用户 1 构建的 Program 最简单，进行资源 Mashup 时用到的设备数也最少，同时也只用了 43s 就完成了 Program 的构建，另外，这个 Program 只用到两个Behavior，一个是 MP1 上的 TakePicture，另一个是 C 上的 SavePicture。用户 2、3、4 相对于用户 1 来说，所设定的需求目标略复杂，但构建的 Program 也达到了用户目标。用户 3 构建的 Program 在执行时没有成功，通过查询日志发现是由于在调用执行 MP2 上的 ViewPic Behavior 时，MP2 上 Agent 离线导致的(手机网络中断导致)。用户 3 和用户 4 进行资源 Mashup 时分别用到了 3 个和 4 个 Behavior，所构建的 Program 也成功执行。用户 5 的目标最复杂，他首先用 MP1 进行拍照，然后把照片在 PC 上美化，美化完成后发送到家人手机上显示，同时在智能相册上显示照片并在打印机上打印照片，最后把照片保存到个人云上。整个过程中需要用来 Mashup 的资源共有 6 个，分别是 MP1 上的 TakePicture、PC 上的 BeautyPic、MP2 上的 ViewPic、SA 上的 ShowPic、P 上的 Print 和 C 上的 SavePicture，这 6 个 Behavior 均是用户在构建 Program 时按顺序手动选取的。从所有用户构建 Program 所用时间来看，构建一个 Program 并不会花费太多时间，这说明通过我们的 Android 客户端实现资源 Mashup 是可行并且是方便快捷的；同时，相对来说，所要构建的 Program 越复杂，所需要的时间会更长，但这与用户使用 UI 界面的熟练程度有关，因人而异。

在实验结束之后，对参与实验的用户进行了问卷调查，让他们对整个框架的部署以及 Android 客户端的使用做了反馈，其中具体询问了他们对客户端 UI 界面的使用感受，并让他们对比描述使用我们框架及客户端与不使用来完成同一个目标的差别。大部分用户认为若不使用我们的框架及客户端来完成同一个目标会比较繁琐，比如用户 5 反馈说若不使用框架和客户端，他需要通过数据线或者 QQ、微信等应用来不断传输照片，同时，还需要手动打印照片，整个过程会耗费更多的人力与精力；但用户 1 反馈说市场上已有类似的云共享应用可助其完成相关功能，与其进一步沟通后了解到云共享类的应用可以帮助他完成类似实验中简单的任务目标，但是更负责的目标会需要更多的应用，没有我们的框架及客户端方便、节省精力。

此外，调查问卷询问了用户关于 Mashup 的了解程度，5 个用户中只有 1 个在实验前不了解 Mashup 概念，但其还是可以使用我们的框架及客户端进行资源 Mashup 来完成自己的目标。同时，所有用户均认为通过我们的客户端 UI 界面构建 Program 非常方便快捷，可以帮助他们完成自己的需求目标。图 6 是参与实验的 5 个用户针对我们安卓客户端的一个评价概览，这些用户从功能、性能、用户交互 3 个方面做出了评分，1 分表示差，5 分表示优。从结果来看，我们得到了平均分 4.67，其中功能单方面的平均分为 4.6，这说明整体来看，用户对这个 Android 客户端的实现比较满意，它可以有效帮助用户达成资源 Mashup 的目标。另外，在调查问卷的结果中，用户认为最吸引他们的地方在于将很多功能糅合在一起，他们可以通过简单的构造程序来完成复杂的目标，并且 UI 界面美观、操作易懂，但同时用户也提出在网络连接、隐私等方面有待改进。

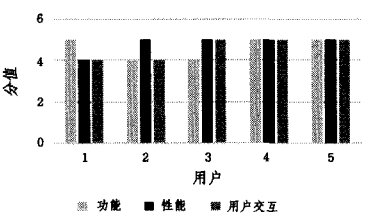


图 6 用户对 Android 客户端的评价

5.3 对比讨论

根据实验后用户反馈的使用我们框架与不使用我们框架的体验差别，收集了几款工业界/科研界与我们的实现功能相关的产品，并从我们目前的框架设计及实现从功能点方面做了对比分析，如表2所列。从表中可以看出，与两款个人云存储产品[15,16]相比，我们的框架及实现能够帮助用户管理个人云中的设备，并且可以完成服务及资源的管理和整合；而与文献[11]中Smartphone上的Mashup工具相比，我们的框架设计可以支持文件的存储、同步和共享，并且文献[11]的工具只支持用户依据模板来构建Mashup应用，而不能让用户根据自己在不同应用场景中的需求构建Mashup。可以看出，我们的Mashup框架及实现在个人云管理和服务及资源Mashup两个方面均有不错的表现。

表2 不同产品功能点对比

产品	管理多设备	支持文件存储	支持文件同步与共享	支持各种设备上服务或资源的调用	支持服务及资源整合	支持根据用户需求构建不同应用
我们的Mashup框架及实现	√	√	√	√	√	√
SEAGATE:Personal Cloud Home Media Storage[15]		√	√			
Western Digital:My Cloud[16]		√	√			
Mobile Mashup Tool (Smartphone)[11]				√	√	

结束语 随着信息技术的发展，各种智能设备层出不穷，智能家居对人们来说也不再陌生。在个人用户的家庭或工作环境中，各种终端设备通过网络连接，形成私密性更好、安全性更高的个人云，为个人的数据处理和共享均带来了很大便利。为了使用户更好地使用个人云中各种设备上的计算资源和处理能力，根据不同需求目标整合不同设备的服务及资源，提出了一个基于移动设备的个人云服务及资源Mashup框架。用户使用此框架，可针对个人云中的设备做统一管理，并可以通过整合已统一描述的服务、资源，方便、快捷地构建个人化应用。此外，为了验证此框架的有效性，基于Agent做了框架实现，并开发了一个Android客户端协助用户进行服务及资源的Mashup。案例研究结果表明，所提出的框架具有可行性，所做的实现以及客户端也具有较好的用户评价。

但是，所提框架实现以及Android客户端还存在着一定的缺陷，比如，用户在进行资源Mashup时，只支持Behavior的串行实现，不支持并行；在实现框架时，还需要对资源进行统一的开发实现，这需要一定的工作量；Android客户端的用户交互方面也存在需要改善的地方。在以后的工作中会针对这些问题制定解决方案，改善目前的设计及实现。

参 考 文 献

[1] Kazi A, Kazi R, Deters R. Supporting the personal cloud[C]// 2012 IEEE Asia Pacific Cloud Computing Congress (APCloudCC). IEEE, 2012: 25-30

[2] Ardissono L, Goy A, Petrone G, et al. From service clouds to user-centric personal clouds[C]// IEEE International Conference on Cloud Computing, 2009(CLOUD'09). IEEE, 2009: 1-8

[3] Gartner IT Glossary. Personal Cloud[OL]. (2014-11-18) [2016-01-11]. http://www.gartner.com/it-glossary/personal-cloud

[4] Announcement G. Gartner Says the Personal Cloud Will Replace the Personal Computer as the Center of Users' Digital Lives by 2014[OL]. (2012-03-12)[2016-01-11]. http://www.gartner.com/newsroom/id/1947315

[5] Liu X, Hui Y, Sun W, et al. Towards Service Composition Based on Mashup[C]// 2007 IEEE Congress on Services. IEEE, 2007: 332-339

[6] Drago I, Mellia M, M Munafo M, et al. Inside dropbox: understanding personal cloud storage services[C]// Proceedings of the 2012 ACM Conference on Internet Measurement Conference. ACM, 2012: 481-494

[7] Bocchi E, Drago I, Mellia M. Personal cloud storage: Usage, performance and impact of terminals[C]// 2015 International Conference on Cloud Networking (CloudNet). IEEE, 2015: 106-111

[8] Benslimane D, Dustdar S, Sheth A. Services mashups: The new generation of web applications[J]. IEEE Internet Computing, 2008(5): 13-15

[9] Liu X, Hui Y, Sun W, et al. Towards Service Composition Based on Mashup[C]// 2007 International Congress on Services. IEEE, 2007: 332-339

[10] Liu X Z, Huang G, Zhao Q, et al. iMashup: a mashup-based framework for service composition[J]. Science China Information Sciences, 2014, 57(1): 1-20

[11] Chang Y S, Lee S H, Kim J C, et al. Study on mobile mashup webapp development tools for different devices and user groups [C]// 2014 International Conference on Information Networking (ICOIN). IEEE, 2014: 433-438

[12] Miao Dong-qing, Liu Jin-du, Shen Li-wei, et al. A Mashup Development and Run Platform Supporting Terminal Devices Integration[J]. Computer Applications and Software, 2012, 29(12): 192-195(in Chinese)
苗冬青，柳金杜，沈立炜，等. 一种支持终端融合的Mashup应用开发与运行平台[J]. 计算机应用与软件，2012，29(12)：192-195

[13] Huang G, Ma Y, Liu X, et al. Model-Based Automated Navigation and Composition of Complex Service Mashups[J]. IEEE Transactions on Services Computing, 2015, 8(1): 494-506

[14] 贝利费米尼，开罗，格林伍德. 基于JADE的多Agent系统开发[M]. 程志峰，张蕾，陈佳俊，等译. 北京：国防工业出版社，2013：26-29

[15] SEAGATE. Personal Cloud Home Media Storage[OL]. [2016-02-29]. http://www.seagate.com/as/en/products/network-attached-storage/home-network/personal-cloud

[16] Western Digital. My Cloud[OL]. [2016-02-29]. http://www.wdc.com/en/products/products.aspx?id=1140