Sip2Share: A middleware for mobile peer-to-peer computing

Gerardo Canfora, Fabio Melillo
Department of Engineering, University of Sannio, viale Traiano 1, 82100 Benevento, Italy
gerardo.canfora@unisannio.it, fabio.melillo@gmail.com

Keywords: Mobile device programming, service oriented computing, peer-to-peer, Android.

Abstract: The growing success of mobile devices is enabling a new class of applications that overcome the traditional models of desktop applications and web browsing, and embrace entirely new ways of computing. Service-oriented computing and the rapidly growing power of mobile devices are the key ingredients of a new generation of low-cost, lightweight applications where mobile devices are no longer intended as a means to access server-side data and functionality, but as a source of services that other devices can discover and invoke.
In this paper we introduce Sip2Share, a middleware that allows for publishing, discovering and invoking services in a peer-to-peer network of Android devices. A characteristics of our middleware is that services are advertised, discovered and called using the same native mechanisms of the Android platform, i.e. intent, manifests and broadcast receivers.

1 Introduction

Mobile devices, such as smartphones and tablets, are experiencing a growing popularity, to the point that in 2010 the number of handled devices sold has exceeded the sales of personal computers (Maier, 2011). Most of these devices are increasingly powerful in terms of computation, data storage and visual displays, are equipped with a rich array of sensors, are almost always connected, and have a good autonomy of batteries. According to market research(ComScore, 2012), the top 5 smartphone sold in the world have, on average, 1GHz CPU, 512MB RAM, 1500mAh battery, WiFi and 3G/4G data connection. Thus, it is possible to use handled devices as part of a network, not only as simple users of back-end services, but also as providers of services.

As a matter of fact, the pervasiveness of mobile devices is enabling a new generation of service-oriented applications where different kinds of actors, including people, sensors, and software agents, exchange services in dynamic, peer-to-peer overly networks. This entails overcoming the traditional approach of sharing content among devices to move towards environments that allows for publishing generic services, including human-based services, on smartphones. In this paper, we introduce a middleware architecture to support the development of service-oriented pervasive applications for mobile devices. The middleware, named Sip2Share, implements a peer-to-peer network of services provided by mobile devices and offers mechanisms to publish, advertise, discover and invoke the services. Sip2Share has been designed for the Android[1] platform and uses extended versions of Android native mechanisms, namely intent, manifests and broadcast receivers, to publish, discover and consume services. Thus, developers who are familiar with application development for Android will need very little training to be productive with our middleware. The choice of using Android derives from several factors, including the growing diffusion according to the market data (Pettey, 2011), Android is the most widespread OS for handled devices, the open-source nature of the platform, and the fact that it is well supported and easily programmable using classic IDEs, like Eclipse[2], and the Java language. A point of strength of our middleware is the transparency. From a user point of view, spatial, temporal and synchronization transparency is supported by using a publish/subscribe pattern. In addition, the middleware supports transparency for software developers, which allows for easily migrating existing applications without a need for reengineering.

The middleware has many areas of application, for example: social and collaborative applications, health and emergency management and war games. Whilst the work is still in progress, a stable version of Sip2Share has been developed and is available for

[1] http://developer.android.com/index.html
[2] http://www.eclipse.org

download at Googlecode [3]. To demonstrate the middleware, we implemented a social application, named HelpMe!, that supports networks of people sharing interests and expertise to pose and answer questions on specific topics.

The paper is organized as follows: section 2 discusses technologies for mobile peer-to-peer communication and section 3 introduces the main ideas our middleware and discusses current implementation, including the architecture, APIs, and the communication among components. Section 4 describes the HelpMe! application and section 5 concludes the article by addressing some technical challenges and providing a future development roadmap.

2 Related work

The expansion of mobile communication into IP based networks, allow for designing complex architectures also for handled devices. Most research work has focussed on peer-to-peer communication, due to the dynamic nature of the domain.

Several researchers use the Mobile Ad-Hoc Network (MANet) paradigm for communication. An example is the middleware Rescue (Juszczyk and Dustdar, 2008), designed as a support for emergency teams in the management of natural disaster, a scenario in which, very often, static internet infrastructures are not available. Haggle (Nordström et al., 2009) is a solution for sharing content in spontaneous ad-hoc networks; it uses pure peer-to-peer communication based on the publish/subscribe pattern. Each node stores a graph with the topology of the network; the search of content is topic based and the range in which the search is applied is the set of neighborhood nodes reachable. While these papers present interesting solutions, they do not apply to our idea of building a peer-to-peer network of services using the Internet for communication.

Among the solutions designed for the infrastructure networking mode, many solutions use some form of proprietary protocols for sharing contents (Kelényi et al., 2007)(Hulbert, 2006) (Kotilainen et al., 2005) (Molnr et al., 2007) (Horozov et al., 2002). Other articles in he literature have explored the possibility of publishing traditional web services on mobile phones (Srirama et al., 2006). Some optimization have been applied to reduce consumption of resources on the devices; as an example, the Wireless SOAP (WSOAP) protocol (Apte et al., 2005) reduces the dimension of the message from 3 to 12 times with respect to SOAP.

[3] http://sip2share.googlecode.com/svn/trunk/Sip2Share

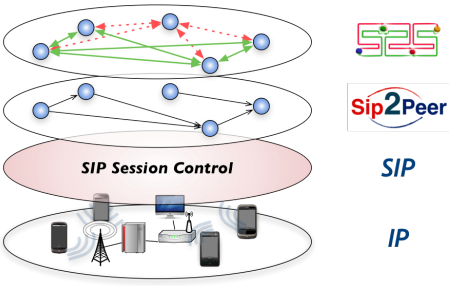

Figure 1: Communication stack

The PeerDroid project (Picone et al., 2010) implements the JXTA (Gong, 2001) stack on the Android platform. PeerDroid has a hybrid peer-to-peer architecture, with a rendez-vous super-peer for the initialization of the communication; the JXTA standard allows for exchanging data among peers independently of the underlying hardware structure. However, the compliance to the standard and the xml-based file exchanged tend to be too heavy for many smartphones.

An alternative is to use use the SIP protocol,(Rosenberg et al., 2002) as in the case of Sip2Peer (Picone, 2011), which exploits a simple SIP datagram to exchange text-based messages among Android peers. Sip2Peer implements mechanisms to exchange messages, including JSON-like messages,(Crockford, 2006) between peers with known addresses in a peer-to-peer network. It also provides a bootstrap peer that stores the addresses of all the peers registered in the network. Our work builds on top of the communication mechanisms of Sip2Peer and adds the additional resources needed for discovery, publishing and using services in a network of Android devices.

3 Sip2Share

The main idea of our work is to share services among peers using the Android platform, and to provide application developers with an API for remote services that resembles the native Android API for inter-application communication on a single device. In particular, we use an extension of the intent mechanism for communication among services on remote peers in a way that is transparent to both the programmer and the users.

Figure 1 shows the layers of communication comprised in our middleware. An IP network is at the bottom level of the stack; over this layer, there is the SIP session control, needed for identifying the nodes over the network. Next layer is Sip2Peer, that uses the SIP protocol to send data. Our middleware is positioned on the top of this stack, and manages the overlay net-

work, with a full duplex communication and the ability to deploy, publish discover and invoke services on mobile devices.

The network is hybrid, with peers and super-peers. The purpose of super-peers is to decouple the peers from each other by implementing a publish/subscribe pattern. Peers are associated to service descriptions that are stored and managed by super-peers; in this way, a peer that needs to connect to the network of services only have to send a request to the super-peer with a description of the services it seeks. The super-peer will return the addresses of the peers in the specific overly network related to the request.

3.1 Communication among services using intents

Sip2Share uses intents as a mechanism for communication among services on remote peers. To help understanding our communication mechanism, we briefly recall how native Android intents work in the local scenario.

Android allows applications on a device to communicate with each other by means of Intents, i.e. abstract descriptions of an operations to be performed on some pieces of data [4]. This mechanism is halfway between message passing and publish-subscribe and provides a facility for performing late runtime binding between different applications. An application that desires to communicate with other applications prepares an intent with the data and the action to be enacted on those data, and sends the Intent to Android, which will search, among the registered applications, those that can satisfy the request. BroadcastReceivers are used to receive intents sent by other applications, and bind the abstract operation to an actual piece of code that implements it. Intents are filtered by Android based on the action they request; only matching actions are sent to the respective receiver. In this way, the applications are loosely coupled because the developer does not need to know which applications are installed on a device; she only sends an Intent to whom can handle it, and the target application is discovered at execution time.

We adopted the same philosophy in our middleware. Sip2Share uses remote intents for discovery and invocation of services. A peer prepares the intent and, through Sip2Share, sends it in broadcast to the peers in the network; if the sending peer already knows the address of other peers that can respond to the particular request, then it can send it directly. Otherwise, it has to send some data (the service descriptor) to the super-peer, and to wait for a reply with the addresses of the peers to contact. All the communications are asynchronous and the peers are alerted by push notifications. Our aim is that, if a developer is already skilled in Android development, she does not have to learn another technology, because Sip2Share uses the same Android's patterns for remote communications among devices as for communication among applications on the same device.

[4] http://developer.android.com/guide/topics/intents/intents-filters.html

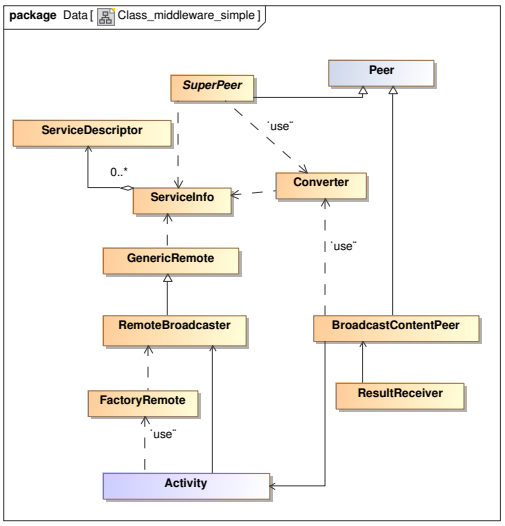

Figure 2: Sip2Share main classes

3.2 Architecture

Figure 2 shows the main classes of the middleware; its boundaries are marked by the `Activity` and the `Peer` classes, the first belongs to the user application, the second one to the Sip2Peer middleware.

The developer is provided with a class, `RemoteBroadcaster`, that has the methods for sending intents across the networks of peers, using the same signature of the local method. The rest of the classes and operations are hidden from the developer, that will be notified asynchronously when a request is completed. The core classes are two: `BroadcastContentPeer` and `Converter`. `BroadcastContentPeer` is the part of the middleware responsible for communication; it uses the `Converter` for marshaling and un-marshaling the messages. When a request is dispatched, `ResultReceiver` is responsible for collecting the results and delivering them to the service that originated the request. Services are modeled by a `ServiceDescriptor`, which is a nested class of `ServiceInfo`. Based on this informations the super-peer can seek the right peer,

comparing the requested and published actions.

3.3 Application Interface

For a developer, it is easy to migrate a local applications to a peer-to-peer architecture using Sip2Share. In fact, there are very few additional actions to be performed with respect to the local scenario. First, the current implementation of our middleware needs a peer for initializing the connections; we provide a super-peer implementing a syntactic matching among actions, and the pattern for push notifications, all for the standard Java environment. Once set up the super-peer, the developer has to create a `RemoteBroadcaster`, using `FactoryRemote`. Then she can send the Intent. An example is in the following fragments of code, that compare the local scenario with the remote one.

```
/*LOCAL*/
1. Intent intent=new Intent("org.unisannio.somethinghappens");
2. intent.putExtra("myLabel", "theValue");
3. this.sendOrderedBroadcast(intent, null, myBroadcastReceiver,
       null, Activity.RESULT_OK, null, null);


/*REMOTE*/
1. Intent intent=new Intent("org.unisannio.somethinghappens");
2. intent.putExtra("myLabel", "theValue");
3. RemoteBroadcast remoteBroadcaster=(new FactoryRemote())
       .getRemoteBroadcaster(this);
4. remoteBroadcaster.sendRemoteOrderedBroadcast(intent, null,
       myBroadcastReceiver, null, Activity.RESULT_OK,
       null, null);
```

It is evident that local and remote broadcasting of events use the same approach, although there are little differences between the two fragments of codes, in line 3 and 4. Also the remote publication of a service is very similar to the local scenario, requiring only the definition of the actions that the device can manage. The following fragment of code shows how a peer publishes its own service, which will be reachable in response to the particular action defined in the descriptor.

```
1. RemoteBroadcast remoteBroadcaster=(new FactoryRemote())
       .getRemoteBroadcaster(this);
2. ServiceInfo sinfo=new ServiceInfo();
3. sinfo.addDescriptor("org.unisannio.somethinghappens", null,
       null, null);
4. remoteBroadcaster.sendToSuperPeer(sinfo, Converter.PUBLISH);
```

3.4 Super-peers and service discovery

In addition to the middleware, we provide a super-peer that can handle several types of messages, needed for discovering and publishing services. Our super-peer is an abstract class that needs to be implemented, either in Android or standard Java, to implement specific matching and discovery algorithms. A peer that is seeking some service sends a request to the super-peer; this returns the data needed by the peer to contact the other devices that offers services eligible for satisfying the peer request.

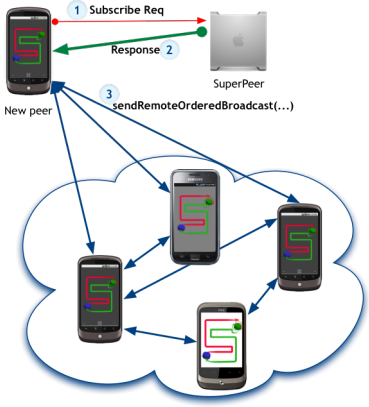

Figure 3: High level operations

Figure 3 shows an an example. A peer is looking for a service in a new network; it sends a subscribe request to the super-peer (1), if there are some peers that can handle the request the super-peer returns their addresses (2), and the peer can contact them directly (3). Otherwise, the peer request will be stored on the super-peer, and the peer will be notified when another peer will have published the requested service.

A peer can publish a service by sending a message to the super-peer with its service descriptor. A service descriptor is a structure in which are stored the action and tags; the action is the same as the Android actions, the tags are possibly used by the super-peer to matching requests against available services. The message sent to the super-peer may differ, according to the purpose of the communication: publishing or retracting a service, subscribe or unsubscribe a service or a category of services.

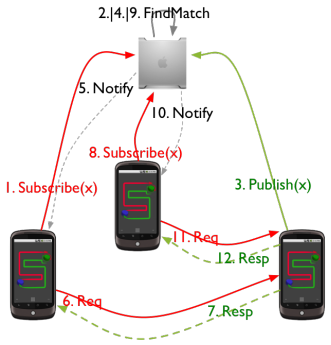

Figure 4: Possible interactions

As an example, Figure 4 depicts an interaction scenario between two requesting peers, a super-peer and a peer that offers a service that march the requests. The main steps are: *(a)* peer A subscribes for a service

x; *(b)* the super-peer looks for a match, there are no results for that request, then no address is returned; *(c)* another peer, B, publishes a service descriptor for x; *(d)* the super-peer finds a match and notifies peer A with the address of B; *(e)* A and B communicate using Sip2Share; *(f)* a new peer, C, subscribes for x; *(g)* the super-peer finds a match and notifies C with the address of B; *(h)* C and B communicate using Sip2Share. If the peers which offer services matching the request of the requesting peer are more than one, steps (e) and (h) use an ordered broadcast to send the message; all responses are addressed only to the original requester.

4 An Example of Application: HelpMe!

In this section we describe a simple application that exploits the Sip2Share middleware to publish human-based services. The basic idea is to help people to share knowledge with other members of a thematic network. Knowledge sharing happens in the form of threads of queries and answers. As an example, a user, globally connected, is in a shopping mall to buy a camera and, in front of several offers, she needs a suggestion from an expert to make a decision. She can use HelpMe! to broadcast a query on the topic *Photography*; all the peers that have registered as experts on the same topic will receive the query, and can answer it. All the answers will be collected by the original requester device and the user will be alerted using notifications.

Figure 5 illustrates an example of use of HelpMe! The first peer submit a query to the peers network on a particular topic, in this case Photography [5(a)]. The scenario assumes that there are no peers registered on the topic. Later, peer B publishes services for two topics, *Computer* and *Photography* [5(b)]; the super-peer checks the matches of the new services with the stored queries, and send a notification to peer A. The middleware component on peers A receives the list of peers of interest – in this case only peer B – and sends the query. Peer B receives a notification with the query and can answer it [5(c), 5(d)]. In case of multiple peers that match a query, each answer produces a notification on peer A, and all the answers are shown in a list below the correspondent query [5(e), 5(f)].

Figure 6 shows the structure of the application. Thanks to `RemoteBroadcaster`, the application needs only the main Activity and the specific receivers: the middleware hides from the developer all the complexity of discovery, notification and communication.

(a) `A_1` (b) `B_1` (c) `B_2`

(d) `B_3` (e) `A_2` (f) `A_3`

Figure 5: Simple question and answers

The application, while simple, illustrates the typical structure of a Sip2Share application, which comprises middleware components on the devices and a super-peer, deployed in a remote machine reachable by an IP connection. On the device, the application provides users with the possibility of: setting manually the bootstrap super-peer address, sending a query, reading the answers, registering as an expert for a given topic, answering queries. We have implemented a simple super-peer, using standard Java, MySql[5] and the Hibernate framework[6], that implements a publish/subscribe model where look up and matching is performed syntactically upon topics.

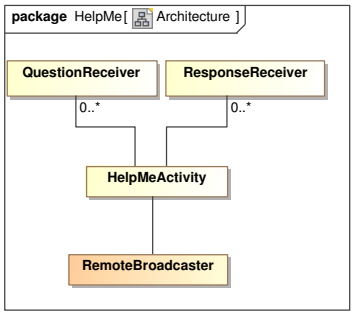

Figure 6: HelpMe structure

5 Conclusions

The paper has introduced a middleware, named Sip2Share, for peer-to-peer computing in a network of mobile devices running Android. The middleware allows discovery, publishing and invoking remote services using the same native mechanisms of the Android platform, i.e. intent, manifests and broadcast receivers. The use of the middleware has been illustrated with a simple application, HelpMe!, that supports knowledge sharing within a thematic network.

[5] http://www.mysql.com/

[6] http://www.hibernate.org/

Whilst HelpMe! uses human-based services, the middleware can be used to publish automated services: for example, in a war game on a closed field, the application can query the devices of its own team, to know the GPS positions.

In the current implementation, data items are exchanged using JSON based messages; for the future, we are working towards providing developers with methods for sending large-size data. In particular, we are exploring two directions: (i) using TCP based communication, which implies bypassing Sip2Peer communication and developing a proprietary low-level communication layer; (ii) fragmenting the data to be sent into several chunks, and implementing a layer to reconstruct the original data onto the receiver device. Another area of investigating is binary object management, such as images or videos. Currently, the exchange of binary files is handled outside the middleware, and diverges slightly from the Android approach. In fact, on a single device, Android does not need to transfer the whole file, but only its reference, which is stored in the device. In the remote scenario, the middleware needs to transfer the file, or to upload it on some server known to the network of peers.

In the current version of the middleware we have neglected the privacy aspect of peer-to-peer for the future, the middleware will need to support privacy and security policies for peer interactions. We intend to include in Sip2Share a trust component that can block some peers, and in general assigns ranks to peers, based on past activities; e.g. in the HelpMe! example a particular peer that sends a lot of spam for many different topics.

REFERENCES

Apte, N., Deutsch, K., and Jain, R. (2005). Wireless SOAP: Optimizations for Mobile Wireless Web Services. In *Proceedings of the 14th international conference on World Wide Web, Chiba, Japan*, pages 1178–1179.

ComScore (2012). 2012 Mobile future in focus. *WhitePaper Available at http://www.comscore.com/Press_Events/Presentations_Whitepapers/2012/2012_Mobile_Future_in_Focus (last checked: 24/04/2012*, (February).

Crockford, D. (2006). RFC 4627 - The application/json Media Type for JavaScript Object Notation (JSON). In *Internet Engineering Task Force IETF Request for Comments*, number 29 May 2009, pages 1–11. Internet Engineering Task Force.

Gong, L. (2001). Industry report: Jxta: A network programming environment. *IEEE Internet Computing*, 5(3):88–95.

Horozov, T., Grama, A., Vasudevan, V., and Landis, S. (2002). Moby - a mobile peer-to-peer service and data network. In *31st International Conference on Parallel Processing (ICPP 2002), Vancouver, BC, Canada*, pages 437–444.

Hulbert, D. (2006). Wizbit: A bittorrent client for s60 symbian smartphones using python. *[Available at https://sites.google.com/site/dave1010/wizbit] (Last checked: 20/04/2012)*.

Juszczyk, L. and Dustdar, S. (2008). A middleware for service-oriented communication in mobile disaster response environments. In *6th International Workshop on Middleware for Pervasive and Ad-hoc Computing, December 1-5, 2008, Leuven, Belgium*, pages 37–42.

Kelényi, I., Csúcs, G., Forstner, B., and Charaf, H. (2007). Peer-to-Peer File Sharing for Mobile Devices. In Fitzek, F. and Reichert, F., editors, *Mobile Phone Programming*, chapter 15, pages 311–324. Springer Netherlands.

Kotilainen, N., Weber, M., Vapa, M., and Vuori, J. (2005). Mobile chedar - a peer-to-peer middleware for mobile devices. In *3rd IEEE Conference on Pervasive Computing and Communications Workshops (PerCom 2005 Workshops), HI, USA*, pages 86–90.

Maier, D. (2011). Sales of Smartphones and Tablets to Exceed PCs. *Available at http://www.practicalecommerce.com/articles/3069-Sales-of-Smartphones-and-Tablets-to-Exceed-PCs- (last checked: 24/04/2012)*.

Molnr, B., Forstner, B., and Kelnyi, I. (2007). Symella 1.40. *Available at http://symella.aut.bme.hu (last checked: 23/04/2012)*.

Nordström, E., Gunningberg, P., and Rohner, C. (2009). A search-based network architecture for mobile devices. *Department of Information Technology, Uppsala University, Tech. Rep [Available at http://www.it.uu.se/research/publications/reports/2009-003/2009-003-nc.pdf] (last checked: 24/04/1012)*.

Pettey, C. (2011). Gartner Says Android to Command Nearly Half of Worldwide Smartphone Operating System Market by Year-End 2012. *[Available at http://www.gartner.com/it/page.jsp?id=1622614] (Last checked: 20/04/2012)*.

Picone, M. (2011). sip2peer Tutorial Android Example Outline. *[Available at http://code.google.com/p/sip2peer/wiki/sip2peerTutorial] (Last checked: 20/04/2012)*.

Picone, M., Farber, B., and Hu, L. (2010). Introduction to Peerdroid. *[Available at http://dsg.ce.unipr.it/userfiles/file/peerdroid/PeerdroidPresentation_03_2010.pdf] (Last checked: 20/04/2012)*.

Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and Schooler, E. (2002). SIP: Session Initiation Protocol. RFC 3261. RFC Editor United States.

Srirama, S. N., Jarke, M., and Prinz, W. (2006). Mobile web service provisioning. In *Advanced International Conference on Telecommunications and International Conference on Internet and Web Applications and Services (AICT/ICIW 2006), 19-25 February 2006, Guadeloupe, French Caribbean*, page 120.