ShAir: Extensible Middleware for Mobile Peer-to-Peer Resource Sharing *

Daniel J. Dubois[1], Yosuke Bando[1,2], Konosuke Watanabe[1,2], Henry Holtzman[1]
[1]MIT Media Lab, Cambridge, MA, USA
[2]Toshiba Corporation, Tokyo, Japan
ddubois@mit.edu, yosuke1.bando@toshiba.co.jp,
konosuke.watanabe@toshiba.co.jp, holtzman@media.mit.edu

ABSTRACT

ShAir is a middleware infrastructure that allows mobile applications to share resources of their devices (e.g., data, storage, connectivity, computation) in a transparent way. The goals of ShAir are: (*i*) abstracting the creation and maintenance of opportunistic delay-tolerant peer-to-peer networks; (*ii*) being decoupled from the actual hardware and network platform; (*iii*) extensibility in terms of supported hardware, protocols, and on the type of resources that can be shared; (*iv*) being capable of self-adapting at run-time; (*v*) enabling the development of applications that are easier to design, test, and simulate. In this paper we discuss the design, extensibility, and maintainability of the ShAir middleware, and how to use it as a platform for collaborative resource-sharing applications. Finally we show our experience in designing and testing a file-sharing application.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures

General Terms

Design

Keywords

Resource sharing, peer-to-peer, mobile devices, middleware

1. INTRODUCTION AND MOTIVATION

Increasingly large numbers of always-on smartphones and tablet computers roam our cities with powerful resources such as high bandwidth WiFi radios, high computational capabilities, and large quantities of flash storage. Some of these resources may be available in a device but not used, while some of them may be unavailable, but needed. This may happen for example when some popular video, photo, information (e.g., weather forecast, news, etc.) are needed, but the cellular/WiFi network is not available to download them. Another important emerging trend is the appearance of very small pervasive devices such as the FlashAir SD Card [22], which is an SD card equipped with a tiny WiFi adapter and a tiny web server for sharing its files via WiFi. We envision that tiny devices that are approaching the general market are likely to become more common in the future and therefore there is an increasing industrial effort to find innovative ways for exploiting them.

The purpose of this work is to provide an abstraction layer for mobile and pervasive devices that application developers can exploit to share the resources of their devices and to use the resources available from other devices in a crowd-sourced manner [7]. These resources will be provided to the application layer as a service in a similar way as it happens in the cloud computing paradigm [2]: applications can fulfill their resource needs using resources provided by third parties on demand. The difference between this idea and cloud computing is that there is no clear distinction between who uses the resources and who offers them since each device can play both roles. In the case of computational resources only, this computing paradigm has been defined as *crowd computing* [14]. However, in this paper we extend such definition to any kind of resource that can be shared, such as data, network connectivity, and storage.

The abstraction layer we are proposing consists of middleware that exposes to mobile applications some Application Programming Interfaces (APIs) to share and receive shared resources, as well as an extension mechanism to define new resources and additional ways for importing/exporting/managing them. The middleware is independent from the mobile platform and is built on top of some additional low level APIs that abstract device-specific and network-specific low-level details. The modular nature of the middleware results in an increased separation of concerns, code reusability, and it also allows applications to avoid loading parts of the middleware that are not relevant for them. One of the key features of ShAir is the possibility to add/remove/replace modules at run-time depending on the context and application needs, thus allowing the middleware not to be static, but capable of *dynamic evolution*.

Last, but not least, our modularization has been designed in such a way to ease testing applications and middleware extension modules in existing peer-to-peer simulators. These simulators are able to scale to a significant number of simultaneous devices, and to simulate device movements with real mobility patterns [11, 13].

The proposed middleware has been implemented as a Java library. Its usage has been exemplified in a case-study scenario of a file-sharing application for the Android platform [9].

2. RELATED WORK

The following is a non-comprehensive list of previously proposed middleware for mobile peer-to-peer resource sharing.

Sip2Share [5] is a middleware infrastructure for sharing services and activities specific for the Android platform. It extends the Android programming model with the addition of remote computation. μMAIS [17] has a similar purpose, but resources are exported in the form of Web Services [1] and it also supports orchestrated

*This work has been partially funded by the Fondazione Fratelli Rocca (Progetto Roberto Rocca fellowship – MIT-Italy Program).

ESEC/FSE '13, August 18–26, 2013, Saint Petersburg, Russia

services compositions. Other service-oriented paradigms have been recently studied in [6, 15]. All these works focus on the concept of exporting resources as services. MobiClique [16] and a work from Rodriguez *et al* [18] propose modular middleware infrastructures that contain a full network/routing stack for exporting resources in mobile networks. CoCam [21] is a middleware infrastructure specialized in sharing persistent resources and real-time streams. This approach adopts a central coordination for managing group relationships of its peers. SSN [23] and a work from Helgason *et al* [10] propose two different Android middleware infrastructures for sharing data in an opportunistic way. Both of them contain full network stack and the middleware is static, meaning that it cannot change overtime. Mist [19] is a publish-subscribe middleware infrastructure that contains a full network stack and that works in a delay-tolerant way. It is designed to be used as a lower level component of higher-level middleware since it does not offer any higher level of abstraction to the applications. LaCOLLA [12] is a middleware framework for general purpose resource sharing without routing and delay-tolerant communication support. Haggle [20] is another middleware infrastructure that is network-agnostic, supports multiple networks, allows complete separation between application logic and transport logic and also abstracts the concept of resource, which can be any kind of data object or service. Haggle's authors state that in future it can be extended to support proactive behavior, and opportunistic/delay-tolerant communication.

All the middleware infrastructures proposed above have at least one of the following limitations. Middleware equipped with full network stack cannot be easily extended to support evolution and multiple networks, since the network is not abstract. Moreover, middleware infrastructures that abstract network are static and have fixed APIs, therefore it is difficult to adapt their capabilities at run-time. Other works require centralized coordination, just support a limited types of resources, or do not support delay-tolerant communication.

In ShAir we have the goal of overcoming all those issues. To do this we have used Haggle [20] as a source of inspiration, but with the additional capability to support dynamic evolution of the middleware services and APIs at run-time, and the possibility to simulate the applications in real, emulated, and simulated contexts, without altering the middleware code or configuration.

3. ShAir

In ShAir we propose an architecture that is independent from the type of network, the number of networks, the nature of the resources and so on, but that focuses on giving to the middleware the capability to be extended and evolve over time. This need has been recently pointed out in works such as [3, 4], which say that modern highly complex distributed systems should not be based on fixed middleware and fixed APIs, but middleware and application logic should be able to *emerge* and *evolve*. Moreover, applications with different evolutions of the same middleware should be able to coexist [3]. In our solution we are taking into account these guidelines of the Software Engineering research community, and, at the same time, we want to avoid over-engineering and keep the middleware simple enough to be used in an agile way. This simplicity would be fundamental if we want to use very simple mobile devices such as FlashAir [22], as discussed in Section 1.

3.1 Architecture

The ShAir architecture (depicted in Figure 1) is composed of three parts: **Application Logic** contains the actual logic of the application, which interacts with the Middleware Logic through a collection of specialized APIs (see Figure 2 for details); **Middleware Logic** contains all the decision-making logic for managing (adding, querying, notifying, updating, deleting, transferring) resources and data persistency; **Device Specific Logic** contains the implementation of low level APIs (detailed in Figure 2) for accessing networks (e.g., WiFi, 4G, etc.) and storage devices (e.g., flash memory, cloud storage, etc.), which may be device/platform-specific. All the exposed APIs abstract observable components and are accompanied with additional observer APIs used to manage callbacks. This structure enables separation of concerns between the application, the specific platform, and actual middleware.

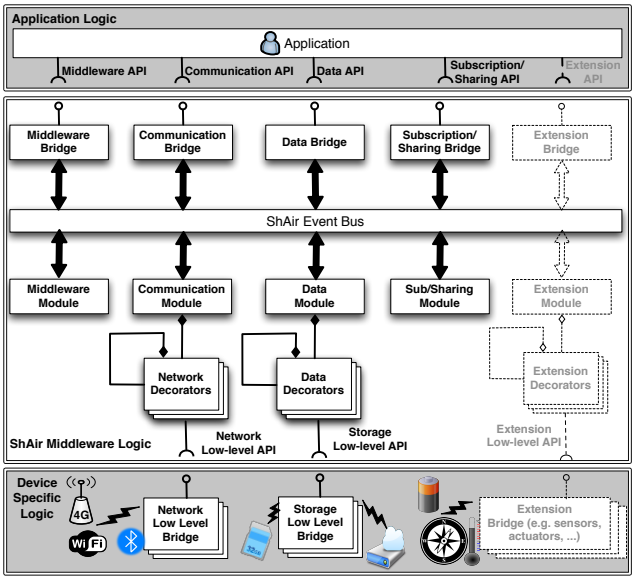

Figure 1: ShAir architecture.

The middleware is not layered and contains three parts: **Bridges** are instances of the Bridge design pattern used to decouple the APIs from the internal components; **Modules** contain the actual business logic of middleware, such as its decision-making logic. Each module may have a reference to the low-level devices. Device accessors functionalities can also be arbitrarily augmented by using **Decorators**, which can provide additional functionalities at run-time (e.g., for adding routing capabilities to the network or to support multiple networks). The communication among the modules is managed by an **Event Bus** that is responsible for decoupling the interaction among the modules using events in a publish-subscribe way. The basic modules are: **Middleware Module**, to manage the middleware lifecycle and add/remove Modules at run-time to make the middleware adaptable and evolvable; **Communication Module**, to manage all network-related decisions; **Data Module**, to manage resources information and generic data; **Subscription/Sharing Module**, to share resources and subscribe to resources in order for the application to be notified when they are available.

3.2 Extensibility and Evolvability

In Figure 1 we can see that some blocks are grayed out and depicted with dotted lines. Such blocks represent optional components for the middleware that can be added at runtime by the running application. This breaks the traditional view that the evolution of the middleware and that of the application logic should be completely independent. In fact, in lightweight middleware, adding additional functionalities may require to implement them in the Application Logic, thus breaking the separation of concerns. To overcome this, ShAir gives application developers the possibility to extend the middleware with additional modules, which may be bridged with additional APIs, equipped with new device decorators, and that may use additional device-specific components, such as GPS, sensors, and power management. The possibility to implement additional functionalities in the form of pluggable modules makes it possible to reuse them in different applications and

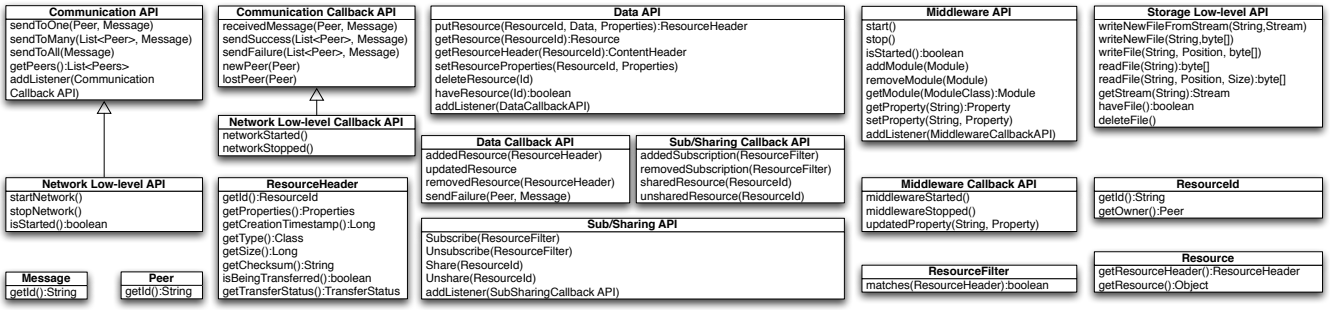

Figure 2: ShAir Application Programming Interface.

to publish/distribute them as middleware plugins in public repositories or as shared resources.

3.3 Security

The logic needed to validate pluggable modules and to ensure secure communication is not part of the standard middleware we are proposing, but is left to custom modules implementations. We plan in the future to study a systematic way for standardizing additional optional modules offering these functionalities. However, in the currently proposed middleware additional pluggable modules are permitted to include their custom solutions for labeling themselves with a version tag, for including a digital signature signed by any trusted authority, and for validating versions and signatures of other modules during interactions.

3.4 Maintainability and Testability

ShAir is designed to have basic simple components, and to have additional features (such as advanced peer-to-peer protocols) added as additional plugins in the form of modules. This simplicity results in less components and code to maintain, and therefore makes the middleware simpler to be used and maintained. The independence of the Middleware logic with respect to the Device-specific Logic also simplifies the testing and simulation phases of middleware components. If the low-level bridges of the Device-specific Logic are implemented as virtual devices (devices that work in a simulated way or in conjunction with existing peer-to-peer simulation platforms such as [11, 13]), then the whole Middleware Logic and the whole Application Logic can be run in simulation mode in a seamless way. Not having to deal with real devices during simulations makes it possible to scale integration tests to a large number of devices (depending on the computational resources of the simulation environment). Another characteristic of the middleware that results in a more effective testing is the possibility to exclude modules that are not needed, to speed-up testing and to help isolate problems.

4. IMPLEMENTATION

We have implemented a first basic prototype of ShAir middleware using Java 1.6, then we have extended the middleware infrastructure to support delay-tolerant peer-to-peer file sharing. We have finally validated middleware by creating a file-sharing App for the Android [9] 2.3.3 platform[1].

4.1 Middleware

All the basic modules (see Figures 1 and 2) have been implemented as single Java classes. The EventBus used to exchange events among the modules is the Guava EventBus[2]. Module instantiations are managed by the Guice dependency injection library[3]. With the use of these external libraries we have been able to keep the code base of all the basic middleware under 5000 lines of code. As will be detailed in Section 4.3 the network abstraction of the middleware does not take care of peer-to-peer communication since that task is delegated to the Android-specific low-level network device we have implemented.

4.2 Testing and Simulation

Middleware has been tested using customized VirtualNetwork and VirtualStorage implementations. Each integration test made use of these implementations in addition to standard object mocking with Mockito [4]. Tests have been run using TestNG [5], while code quality and test coverage have been assessed with Sonar [6]. We have also integrated VirtualNetwork and VirtualStorage into PeerLets for the ProtoPeer simulator [13] to permit a future study of non-functional properties and massively distributed simulations.

4.3 File-sharing App

Requirements. The file-sharing App we have implemented has the following requirements: Android [9] platform support; the application should let the user take pictures with the built-in camera and share them as a file resource into ShAir; the user should also receive the pictures shared by others, put comments in the pictures, delete them, chat with other nearby users, receive rewards in terms of scores every time they use their bandwidth to share pictures.

Middleware Extensions. The additional modules we have added to the system are: *ResourceAdvertisingModule*, to advertise nearby devices of the availability of the resources of a device; *ResourceSeekingModule*, to ask a nearby device for a resource the application is interested in; *ResourceRequestModule*, to respond to resource requests; *ResourceResponseModule*, to actually handle responses to resource requests; *NickNameAssociationModule*, to associate a nickname to a peer. Each new module may be as small as under 100 lines of code, therefore easy to maintain.

Device-specific Logic for Android. We had first to implement the bridges for the Device-specific Logic. As storage device we have used the internal built-in flash storage. For peer-to-peer networking we have used the built-in WiFi only: the device alternates between WiFi AP mode and WiFi Client mode in a random fashion to create dynamic WLANs and discover nearby peers [8]. This choice is due to the fact that alternative peer-to-peer approaches such as WiFi-direct and Bluetooth either require mandatory user-assisted device pairing or device rooting, while we want to share files in a transparent opportunistic way that also works in unmodified devices. In addition to this we have also experimented, without modifying the middleware logic, a configuration in which the built-in WiFi is a permanent WiFi Client that connects to FlashAir WiFi SD cards [22] in permanent WiFi AP mode disseminated in the environment.

[1] http://shair.media.mit.edu
[2] http://code.google.com/p/guava-libraries
[3] http://code.google.com/p/google-guice/
[4] http://code.google.com/p/mockito
[5] http://www.testng.org
[6] http://www.sonarsource.com

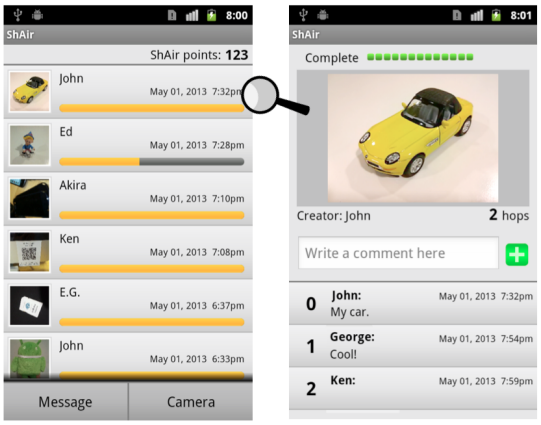

Figure 3: ShAir FileSharing Android App.

Figure 4: Experiment with 12 devices.

Application Logic. Since all the communication and data management are abstracted by middleware, the Application Logic consists mostly of user-interaction logic, with some additional basic logic to instantiate and manage the state of the application, of the network, and of middleware. A screenshot of the running application is shown in Figure 3: on the left side we can see several file transfers in progress, while file details after the transfer can be seen on the right side. We have experimented our App by sharing pictures simultaneously among 12 devices (tablets and phones) from several vendors (see Figure 4).

5. CONCLUSIONS

In this paper we have seen how ShAir can be used as a building element of emergent mobile resource-sharing applications in which mobile devices can opportunistically cooperate and share resources to each other without any fixed existing infrastructure. The middleware infrastructure we have proposed has been designed with the idea of run-time extensibility and run-time evolution in mind, thus going beyond the concept of static APIs of most preexistent works in the area. In addition to this, we have also shown that the non-device-specific nature of ShAir makes its codebase reusable in different real and simulated platforms, and also simplifies unit and integration testing since the emergent-behavior nature of a mobile peer-to-peer system is difficult to mock. We have finally shown our development experience with a middleware implementation in Java and a case study based on a file-sharing application for Android.

The next developments for this work will focus on the optimization of non-functional properties of middleware, thus increasing security, the number of devices and resources that can be shared, while minimizing energy consumption.

6. REFERENCES

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web services*. Springer, 2004.

[2] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, and M. Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, Apr. 2010.

[3] M. Autili, P. Inverardi, P. Pelliccione, and M. Tivoli. Developing highly complex distributed systems: a software engineering perspective. *J. of Internet Services and Applications*, 3(1):15–22, 2012.

[4] G. Blair and P. Grace. Emergent middleware: Tackling the interoperability problem. *Internet Comp.*, 16(1):78–82, 2012.

[5] G. Canfora and F. Melillo. Sip2share - a middleware for mobile peer-to-peer computing. In S. Hammoudi, M. van Sinderen, and J. Cordeiro, editors, *ICSOFT '12*, pages 445–450. SciTePress, 2012.

[6] M. Caporuscio, P.-G. Raverdy, and V. Issarny. ubiSOAP: A Service-Oriented Middleware for Ubiquitous Networking. *IEEE Trans. on Services Computing*, 5(1):86–98, 2012.

[7] A. Doan, R. Ramakrishnan, and A. Y. Halevy. Crowdsourcing systems on the world-wide web. *Commun. ACM*, 54(4):86–96, Apr. 2011.

[8] D. J. Dubois, Y. Bando, K. Watanabe, and H. Holtzman. Lightweight Self-organizing Reconfiguration of Opportunistic Infrastructure-mode WiFi Networks. In *IEEE SASO '13*. IEEE, 2013.

[9] Google Inc. Android. `http://www.android.com`.

[10] O. R. Helgason, E. A. Yavuz, S. T. Kouyoumdjieva, L. Pajevic, and G. Karlsson. A mobile peer-to-peer system for opportunistic content-centric networking. In *MobiHeld '10*, pages 21–26, 2010.

[11] A. Keränen, J. Ott, and T. Kärkkäinen. The ONE simulator for DTN protocol evaluation. In *Simutools '09*, pages 55:1–55:10, 2009.

[12] J. M. Marques, Z. Vilajosana, T. Daradoumis, and L. Navarro. LaColla: Middleware for self-sufficient online collaboration. *IEEE Internet Computing*, 11(2):56–64, 2007.

[13] A. Montresor and M. Jelasity. Peersim: A scalable p2p simulator. In *IEEE P2P '09*, pages 99–100, 2009.

[14] D. G. Murray, E. Yoneki, J. Crowcroft, and S. Hand. The case for crowd computing. In *MobiHeld '10*, pages 39–44, 2010.

[15] K. Nakao and Y. Nakamoto. Toward remote service invocation in android. In *UIC/ATC '12*, pages 612–617, 2012.

[16] A.-K. Pietiläinen, E. Oliver, J. LeBrun, G. Varghese, and C. Diot. Mobiclique: middleware for mobile social networking. In *ACM WOSN '09*, pages 49–54, 2009.

[17] P. Plebani, C. Cappiello, M. Comuzzi, B. Pernici, and S. Yadav. MicroMAIS: executing and orchestrating Web services on constrained mobile devices. *Softw. Pract. Exper.*, 42(9):1075–1094, Sept. 2012.

[18] J. RodríGuez-Covili, S. F. Ochoa, J. A. Pino, R. Messeguer, E. Medina, and D. Royo. A communication infrastructure to ease the development of mobile collaborative applications. *J. of Network and Comp. Applications*, 34(6):1883–1893, 2011.

[19] M. Skjegstad, F. Johnsen, T. Bloebaum, and T. Maseng. Mist: A reliable and delay-tolerant publish/subscribe solution for dynamic networks. In *NTMS '12*, pages 1–8, 2012.

[20] J. Su, J. Scott, P. Hui, J. Crowcroft, E. Lara, C. Diot, A. Goel, M. Lim, and E. Upton. Haggle: Seamless Networking for Mobile Applications. In J. Krumm, G. Abowd, A. Seneviratne, and T. Strang, editors, *UbiComp '07*, volume 4717, pages 391–408. Springer, 2007.

[21] E. Toledano, D. Sawada, A. Lippman, H. Holtzman, and F. Casalegno. Cocam: A collaborative content sharing framework based on opportunistic p2p networking. In *IEEE CCNC '13*, pages 158–163, 2013.

[22] Toshiba Corp. FlashAir: SD Card with Embedded WLAN. `http://www.toshiba-components.com/FlashAir`.

[23] J. Yin and M. Chen. SSN: a seamless spontaneous network design around opportunistic contacts. *J. Mob. Multimed.*, 7(4):239–255, Dec. 2011.