Enabling Resource-Aware Ubiquitous Applications for Personal Cloud with a Pairing Device Framework

Liang (Ben) Chen* and Maria Prokopi†
Orange Labs UK
London, UK
{ben.chen, maria.prokopi}@orange.com

Abstract

Personal devices continue to dominate the consumer market in recent years becoming more affordable than ever. With the increasing capability of these smart devices, we see the growing opportunity of implementing ubiquitous computing in the context of Cloud computing. We propose a common device framework that shall serve the underlying communication requirements, allowing personalised service innovation. We take a device-oriented approach, study the user behaviours within the given context, and look at end-to-end integration feasibility. A prototype framework is developed to evaluate the proposed concepts and demonstrate the relevant use cases. This paper tries to explain the key concepts and proposed solutions, and introduces further discussions along this topic.

Keywords: pairing, android, cloud, context-aware, ubiquitous

1 Introduction

Personal devices, such as smart phones and tablets, are becoming more and more powerful yet affordable than ever for average users. It's not unusual today for people to own more than one of these devices and actually use them for day-to-day tasks. The IT industry is seen accelerate streamlining the "multi-screen" experience by integrating terminal devices with backend service platforms and the mass-volume digital contents to build up mature ecosystems and business models. Notable industrial players range from traditional computer software makers, like Microsoft, Apple, Web service and content providers like Google, Amazon, to manufacturers like Samsung, Asus, etc. In this round of innovation and manufacturing advancement, the so called "smart device" itself is undergoing a change in definition: from phones, to tablets, TVs, set-top-boxes, gaming consoles and many more creative accessories.

From an end user's point of view, existing approaches taken by these service providers towards multi-device integration in fact follow a simple methodology. Take "everything" from the user's terminal devices to the Cloud where the private data is stored and managed centrally. Device updates are synced and pushed across all other registered devices. The approach is rather straightforward. However, taking a real-life example, this means, even if the user's data living on the phone just over the table, they will have to travel through the Internet in order to appear on the user's companion tablet. Furthermore, there are also more ad-hoc scenarios for a user to interact with the others than with the Cloud, or when using Internet becomes less comfortable for the security and privacy concerns. While not rejecting utterly the simplified approach, a ubiquitous context that is composed of better connected local devices, interactions, resources and those with remote presence, may better model real life scenarios. After all, this is still complementing the Cloud with a particular focus on the behaviours on the edge networks. After all the

Journal of Internet Services and Information Security (JISIS), volume: 3, number: 1/2, pp. 83-100

*Corresponding author: Senior Software Engineer, France Telecom R&D UK , Building 10, Chiswick Park. 566 Chiswick High Road, London W4 5XS, UK, Tel: +44-2089891949

†Principal Research Engineer

intelligence of understanding the user behaviours with regards to their physical environments, including their own devices, other user and their devices, helps deliver finer grained services and more intuitive user experience, which is in turn beneficial for both end users and service providers.

Having studied the feasibility and the requirements of such architecture, we propose a device framework stack of common functionality and interfaces as our first step to a tangible solution. The generic device framework addresses the common requirements for easing device-to-device communication, such as establishing network of personal devices, managing unified user identify, maintaining state consistency among paired devices, facilitating context awareness and enforcing security and trust measurements. It builds a systematic template for application development that looks for ubiquitous ability to follow. Prototype implementation and demonstration applications have been built to test the theories and stimulate discussions to follow.

The rest of the paper is organised as follows: we start with the introduction of two user stories. Each focuses on different aspects of in the problem scope, which lead to some key concepts of our work; this is then followed by a chapter ironing out the device framework design and implementation. The paper finishes with the conclusions.

2 Overview

In order to help illustrate the context, two set of use cases are considered. Each has different complexity and emphasis.

2.1 Use Case 1: Personal Devices (UC.PD)

User *Alice* has a smart phone, a tablet and a laptop PC at home. This use journey shows how Alice may seamlessly access resources across all her devices.

(1) Alice is at home and has her new tablet in hand. She launches a system application on the tablet, which generates a QR code on the screen.

(2) Alice launches the same application on her smart phone. She further chooses to bring out the QR code scanner and scans the QR code on the tablet.

(3) A message shows up on both devices saying that the other device has now been recognised.

(4) On the tablet, Alice opens the file browser with which she would like to check out the pictures taken with her phone.

(5) The file browser starts to display thumbnails of pictures from all Alice's digital spaces, including her phone, laptop and albums on the Cloud.

(6) Alice taps on a picture of interest and the picture starts to download from the smartphone and appear on the tablet screen. A cache of the picture is saved in the background as the download completes.

(7) Alice decides to send a SMS message to her friend. Instead of reaching out for the phone, Alice launches the messaging application on the tablet and starts drafting a message. Alice can see all her contacts on the tablet.

(8) Once having finished the message, Alice sends the message through her mobile phone.

We summarise that a few highlights on concept are noteworthy here in the use case:

- Personal devices can be paired up (Step 1-4) to establish trusted relationship among them. They would belong to the same user, under the same identity (provided by the SIM) and would have the same rights towards resources for consumption, and would appear as a whole towards other users or devices in contact.

- Resources, such as data (Step 5-7), services or device capability (Step 8-10), can be shared among all user's devices. Expose, query and access these resources can be potentially addressed by leveraging a middleware agent on device.

2.2 Use Case 2: Smart Meeting Room (UC.MR)

In a more complex use case, user *Bob* is going to a business meeting in his company.

(1) As Bob walking to the meeting room, he is able to see, on this smart phone, list of available services start to appear.

(2) Bob arrives at the meeting room. He touches his NFC[12] phone against the reader at the entrance, confirming his presence to the meeting.

(3) Bob has now the option to browse all the available resources in the room, such as a wireless projector, a shared file space and etc.

(4) Bob also sees all the people attending the meeting including those joining remotely.

(5) It is Bob's time to do a presentation. Bob's presence is recognised by the smart project, therefore authorised to connect to project the slides from Bob's laptop.

(6) *Claire*, although joining remotely, can log in and watch the live presentation on the meeting's web portal.

(7) After the presentation, Bob uploads a document to the meeting's shared folder and selects a list of people who can view it.

(8) All recipients receive a notification to download the file. *David* has been working on his tablet during the meeting, the notification arrives on his tablet as an active device.

(9) The meeting has finished, Claire needs to catch up some contents again. By using the application on her phone, she can select from history the past meeting and resume the same access to, for example, the shared folder, and so on.

This use case traverses end to end a complete flow of activities. Examining from a user's perspective, we find a number of arguments addressed:

- Resources can be broadcasted (Step 1) then dynamically discovered if the user's physical status with respect to the surrounding environment continues to change.

- Context-awareness can be fine tuned by taking into account various device peripheral information (Step 2-3). Hence, the context analysis function that filters and prioritises relevant resource information would impact the middleware performance and efficiency to build personalised applications.

- Resources may be exposed for remote access (Step 7) if the underlying network infrastructure has been designed to support it. Trust and security measures would need to be implemented so individual resources can be relieved from managing this themselves. Nevertheless, either it's local or remote access, the proposed middleware should model in the same way on top of the transport level abstraction.
- User interactions should be mapped intuitively to the technical solution based on device-to-device communication, either it is one or more devices that are involved.
- The notion of "session" becomes relevant not only when dealing with conversational interactions (such as a meeting) (Step 9), but also to facilitate state management for connected devices so that a device can drop out (e.g. phone out of battery) and re-join the "personal Cloud" in the ad-hoc manner without compromising the integrity of the upper level application executions. Logging may also be enabled under the governance of the session.

2.3 Concept Overview

The fundamental elements that help define the problem scope, and their relationships, are shown in the graphical representation of Figure 1.

- `Device` – the physical terminals which users rely on to interact with other resources and users. The ever growing catalogue of heterogenous personal devices should be abstracted away with a middleware layer to an extend whereby a common method can be agreed on to allow further communication with the application specific protocols to be negotiated.
- `Resource` – any consumable capability or data. In our definition, *Resources* can be executable software logic, stored data, or *Device* capability that is exposed appropriately. For example, a mobile phone's accelerometer might be shared with a TV STB so a new service may be created accordingly. Furthermore, device capabilities may be *extendable*, by bridging those of other personal devices. These "add-on" device capabilities may be utilised collectively to determine the contextual status of the user. Finally, a resource in contact may be either local or remote. Being able to support is important for integrating with the Cloud but with a particular focus on the local activities where human interactions take place.
- `People` – the end users of *Devices* and the consumers of *Resources*. From our point of view, a single and consistent recognition of user identity should be maintained so that user interactions can be mapped intuitively and transparently with the underlying technical implementation.
- `Trust & Security` – a transversal function that exists for all other elements and their interactions, such as identity management, access control, and etc. It is also argued that human behaviours could have implications in *Trust & Security* toward technical resolution. For instance, a user's physical presence in the office implies the acceptance of the person's identity being genuine, hence all permitted resources should be exposed to the user and all his devices. Hence, a device middleware needs to facilitate the enforcement of the relevant measures.

3 Framework Architecture and Implementation

Our proposed framework essentially materialises into a device agent that consists of various functional units laid out in a stack architecture.[30] The functional units are abstract and implementation independent. Appropriate solutions may be chosen accordingly to meet specific technical requirements, making

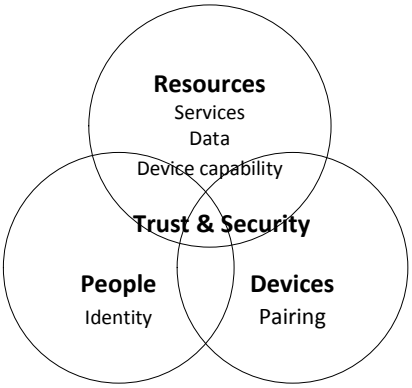


Figure 1: Elements identified in the interaction model

the framework itself flexible in practice to work with various technologies and platforms. Functional specification needs to be defined clearly to ensure the custom adaptation compliance regardlessly, so do the APIs that expose the internal functionality of the framework function units, the interactions between and that with external interfaces. The overall architecture is shown as in Figure 2.

A partial reference implementation of the framework and supporting demonstrator services have been developed to help justify the proposed solution and the concepts behind. Technical comments are given promptly after discussion points, followed by a dedicated section on the prototype itself.

3.1 Connection Stack (CS)

The Connection Stack (CS) is an abstract layer responsible for underlying device connectivity and data communication. It is implementation independent, so meant to leverage existing network protocols. Nevertheless, this is also to accommodate possible more complicated communication models. For example, one could provide a protocol stack that bridges cross-domain communication[14], or enables push notification or messaging-based asynchronous communication[15].

For example, the reference implementation choses the UPnP[13] protocol for the CS. The protocol runs on IP network over WiFi, and has standardised network discovery, control and notification protocols already. With open source UPnP stacks both installed on an Android phone and a Windows PC, it is

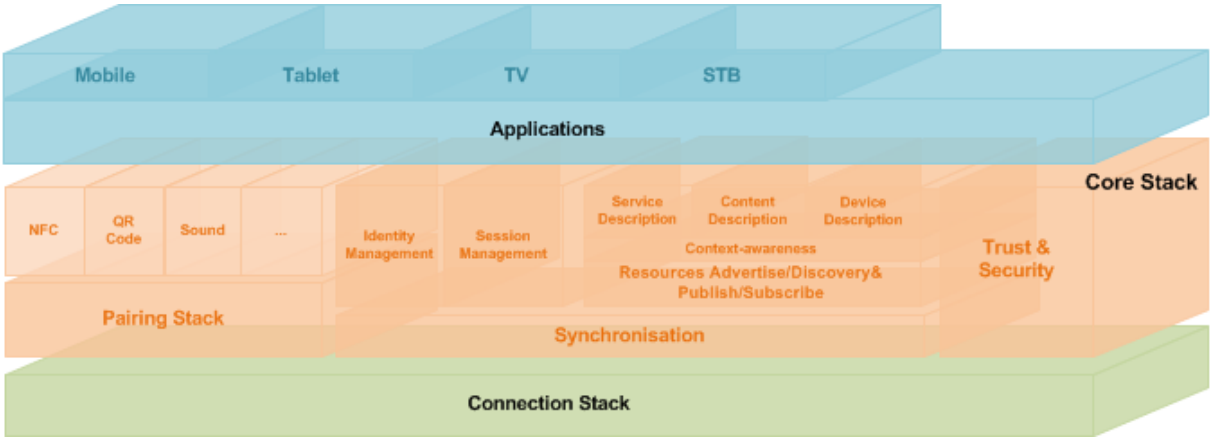


Figure 2: Device framework architecture and components

possible to have the different devices discover and communicate with each other as designed. Since Android 4.0, WiFi Direct[2] is supported and allows compatible devices to connect to each other without the need of a wireless router. Many certified products are available[1] for potential developments.

CS is key to supporting fundamental network connectivity. Network traffic will need to be filtered and routed to the corresponding processors (e.g. synchronisation, resource advertisement, resource publish) in the framework as described in the following sections.

3.2 Application Interface

For a middleware design, it is essential to define the interfaces between the core framework and the residing applications on top. Once having been exposed to, applications can make use of the framework capability through APIs, such as, inter-device communication, resource-awareness, context-awareness and so on.

Since in the peer-to-peer model, any node may be the consumer or the provider of resources, or both, three variations of communication interactions become possible respecitively: communication between applications on the same device framework, between paired personal devices or between two random devices. Each has own considerations towards the framework design.

- Between applications on the same device. The target of the framework is to wrap up, if exists, inter-application communication mechanism and expose it as APIs to support application development. The benefit of using the wrapper APIs instead of direct calls is when having to reach other devices as followed.

- Between applications on different paired devices. This is a more complicated case. First, it is fundamental to distinguish all personal devices of each user from others. The connection set up needs to be secure and trusted. Then, the consistent view of the middleware instances across different devices needs to be maintained, so, to the external nodes, they appear uniformed and the differences in devices are transparent.

- Between applications on different devices. In addition to typical device-to-device communication, the framework must also support various external resources. Being able to identify the differences, establish communication between users' personal networks and enforce the appropriate security checks are essential.

On Android, application data can be passed around with *Intent*s; and named Intents can be matched with the corresponding receivers where on-event logic can be implemented. In the Android prototype, with the help of the CS, Intents are wrapped up within JSON[10] strings in the agreed structure and sent across the network to the paired device. Considering the efficiency, robustness and security aspect of the framework design, the data passed in and out of the middleware is not interrogated, instead the belonging applications have the full flexibility and responsibility towards the data communication. JSON is the preferable method of data formatting for REST[11] services. Similarly for the simplicity in formatting data, the minimal structural overhead and the resulting efficiency of parsing for especially mobile devices, it fits the best our needs. Effectively, this is similar to making remote function calls with parameters. The calls may be synchronous (i.e. blocking) or asynchronous (i.e. non-blocking), even though the later would be much more complicated to cater for things like message correlations, session, persistency and so on. The simpler synchronous model is chosen for the prototype, for example, to fetch files between the tablet and the phone in [UC.PD], to send SMS from a different device than the phone in [UC.PD], or

[1] An incomplete listing of products can be found on the official website, http:///certifications.wi-fi.org/search_products.php

to pass the login token to the room to other device in [UC.MR]. These short-lived, stateless operations, which involve no human interactions, can still be justified with such choice of implementation.

On the other hand, for applications that require or prefer dedicated protocols or channels of their own, for example, for heavy-duty data transfer, transactional commutation and etc., CS may be used to initialise and later tear down the negotiation for the application specific communication, in which case, CS will be acting a pure signalling plane for the upper level applications. Figure 3 labels both the possible signalling and data paths.

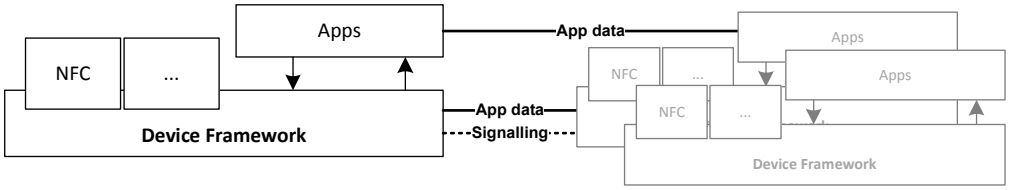


Figure 3: Data and signalling paths between framework instances

3.3 Middleware Stack (MS)

The Middleware Stack, also the core of the device framework, is comprised of several functional blocks. Starting with the *Pairing Stack*, diverse pairing modules may be plugged in to the stack to enable different ways of device pairing. The lower distribution layer propagates and syncs the stack information between the paired devices, enabling other functions such as identity, session and resource management on top. Security and trust are managed transversally across the framework.

3.3.1 Pairing Stack (PS)

The framework allows multiple personal devices to establish atomic personal network by initialising it with the pairing process. A number of methods have been supported by our prototype: NFC, QR code, ultrasound, Bluetooth and SMS. The idea is to pass identification information via one of the appropriate pairing methods so that pairing devices come to realise and trust each other's existence.

In [UC.PD], Alice uses QR code to pair her personal devices. The PS implemented in the Android prototype allows the different pairing methods smoothly integrating to the stack. As a modular component on its own, it has also been wrapped up to create an application that conveniently transfers WiFi connection credentials from one device to another avoiding the manual configuration. (Figure 4)

The UPnP stack used by the CS would give the immediate visibility of devices when they are connected to the network. The pairing process ensures only the paired ones relevant and most importantly, trustworthy. We pass symmetric keys [5] [4] together with the device ID and IP address to ensure only network traffic from indeed a paired device is allowed, and network data can be further encrypted for enhanced confidentiality. Once a peer device is recognised, it is recorded so can be continuously contacted. Implicitly, this means any device only needs to be paired once the minimum. Then device updates will be propagated and synchronised with all fellow devices.

3.3.2 Synchronisation Stack (SS)

Synchronisation is to ensure the state consistency for the middleware instances deployed on multiple devices. The choice is between a centralised approach or a distributed one for the design. With the former, a master device may be made or elected[34][1] to persist and manage the stack information on

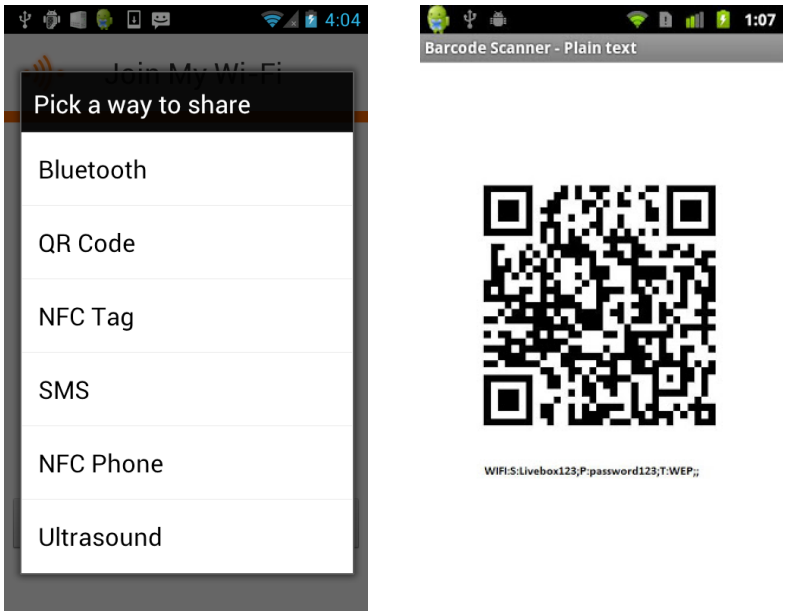


Figure 4: Screenshots of a WiFi sharing application based on PS

behalf of the personal network as a whole. So do all stack queries thereafter have to go through the same device. Hence, the trade-off is whether or not to allow a potential bottleneck device for the simplicity or to distribute the efforts for the robustness but at expense of higher complexity.

The distributed approach suits better the highly dynamic network. A message queue-based solution at the core of the framework implementation would address the typical requirements for networks as such by design[7][9]. In particular, we also see in our context that:

- The personal area network our framework operates in will be very much of ad-doc nature. This involves not only a few number of personal devices, but also a mix of resources, local or remote. Reliability of delivery, durability of data and having non-blocking connections are mandatory.
- The upper level resource management anticipates a publish/subscribe communication model it is discussed in the *Resource Management* section. A messaging based synchronisation layer could be easily mapped to such model, making the overall integration easier.
- In the more complicated case where one of the personal devices is in a remote location, framework synchronisation would still be possible without having to maintain long-lived connections. For example, Google's C2DM[17] push notification may be supported by the middleware to relay uniformed synchronisation messages over the Internet.

Android's *Intent* communication mechanism mentioned earlier in the chapter can as well be used to serve the basic queueing functionality. Separate background service that listens on synchronisation Intents from the CS will update the stack information received from the peer devices. A simple propagation algorithm has been developed[5] in our prototype to guarantee the best-effort delivery of sync messages. Numerous other routing protocols and variations exist for different requirements, especially those designed around MANET[2], for example, DSDV[28], AODV[29], OLSR[6], or HWMP[20].

[2]Mobile Ad-hoc Network

3.3.3 Session Management

Session Management looks after both framework and application sessions. Basic framework information together with those passed on from other functional units, such as identity (e.g. device id and topology), resource (e.g. resource discovery, registry, etc...) and security (e.g. keys or certificates), can be stored and synchronised across devices. On top of SS, framework session is a notion for the lifespan of synchronisation activities. [Handorean et al., 2005] also presents a "follow-me" session implementation with strong context awareness[19].

Applications can explicitly request for creating or destroying a session. Essentially a dedicated space in the framework will be allocated exclusively. It is then up to the application to decide what data should be managed or not in the session. In use cases when the application contacts its remote part or other resources, it is important to record the states of communication and make them transactional in case the remote nodes might come offline unexpectedly. Exception handling or compensation logic would need to be defined accordingly. In any case, SS can still facilitate the replication of session data across the devices.

On Android, the persistent storage comes with several choices, in the order of complexity, which may be *Preference*s, internal or external device file system or SQLite database.

3.3.4 Resource Management

Resource Management plays a major part in the framework architecture that it interacts with the application activities, which involves users, applications and other devices. The architecture design focuses on the following 3 aspects of resource management.

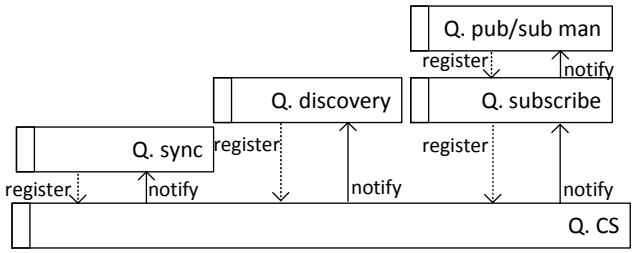


Figure 5: Queueing design for resource management

advertise and discovery The device framework caters for any available resources on a device. As discussed earlier, a *resource* may be a service, data or a device capability. Any identified resource may willingly command the framework to advertise its existence, in a commonly agreed descriptive format. A simple registration would need to be done so that the framework can keep track of the resources it manages. This registry will also be used to look up further information with regards to any specific resource. [8] As for the descriptor, instead of inventing a strictly typed language that intends to map existing description formats, it is generic and, for example, simply labels with MIME[21] types, the referenced resource and lets the matching parser on the consumer side to interpret and decides how to process. For example, a resource that exposes REST[11] APIs would have a completely different payload in its resource descriptor than the one that exposes SOAP[32] APIs. The consumer program would need to programmatically identify the matching type and implement the parsing logic accordingly. [Ha et al., 2007] also presents a solution based on semantic web[18].

Interface	Description
Register resource	Register a resource with the Registry. in: resource name, resource description out: resource ID
Get resource description	Retrieve resource description of a registered resource. in: resource name out: resource description
Register external resource	Register a discovered resource. in: resource name, resource description out: resource ID
Advertise resource	Advertise resource on the network. in: resource name out: response status

Table 1: APIs of Resource Manager

Resource *discovery* is like the reverse process of broadcasting. As a functional counter-part of the *Advertise Processor*, the *Discovery Processor* makes sense of the resource advertisement picked up on the network and passes the parsed resource information up to the local *Resource Manager*, where, similarly, resource information get registered with the registry as resources externally available. Figure 6 shows the various functional blocks and Table 1 lists the related APIs.

Figure 5 gives an overview of possible queueing involved in the framework design. On the Android platform, the same Intent communication mechanism can be used to implement resource advertise and discovery. Registered to the CS queue as an Intent receiver, the *Discovery Processor* will be notified when resource advertisement arrives. Similarly, the *Advertise Processor* can be notified with advertise requests from the *Resource Manager* and builds uniformed messages and contacting the corresponding CS APIs for delivery. Both units run as background services.

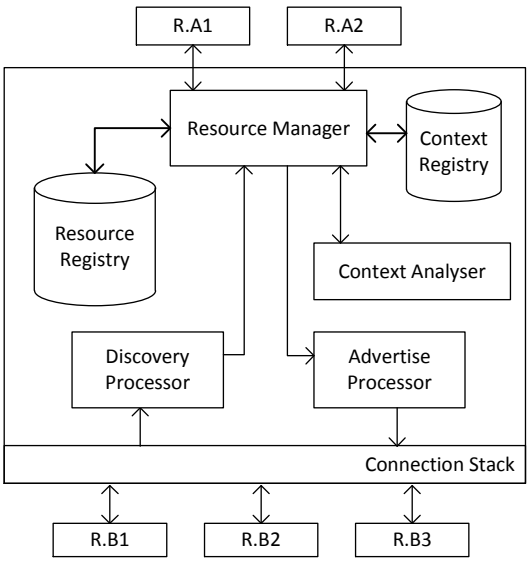


Figure 6: Resource Advertisement and Discovery with context awareness

Interface	Description
Publish	Publish for a resource. in: resource name, message out: status
Subscribe	Subscribe to a resource in: resource name, subscribing resource name out: status

Table 2: APIs of Publish/Subscribe Manager

publish and subscribe Being able to support *publish/subscribe* model is important for enabling loosely coupled communication, which will be generally found in our use cases of interest. The publish/subscribe module to be supported by the framework prioritises the reliability and the efficiency. The basic implementation predominantly maintains a message queue that can buffer and dispatch pub/sub messages, and a companion *Subscription Registry* that keeps subscription records of both local and remote resources, so a local application can query and subscribe to an external resource, or symmetrically a local resource may be subscribed by remote applications. Figure 7 shows the related functional design and Table 2 lists the corresponding APIs.

Following any basic resource information to be included in the resource descriptor, subscription requirements can be attached along the same descriptor when it is advertised in the network. Web service standardisation efforts such as WS-Notification[26], WS-ReliableMessaging[27] and WS-Addressing[33] have tried to define the relevant protocols as part of the Web Services Resource Framework in the past. The intelligence of fully transparent machine-to-machine negotiation is not yet sophisticated as today. Nevertheless, a framework implementation can at least facilitate the common functions like queueing, notification and persistency, while additional features may raise concerns over framework complexity and performance.

Similar to resource *Advertise/Discovery*, *Publish/Subscribe* can reply on the Intent model on Android to handle the relevant events. There is also a similar queueing requirement for the *Pub/Sub Manager*,

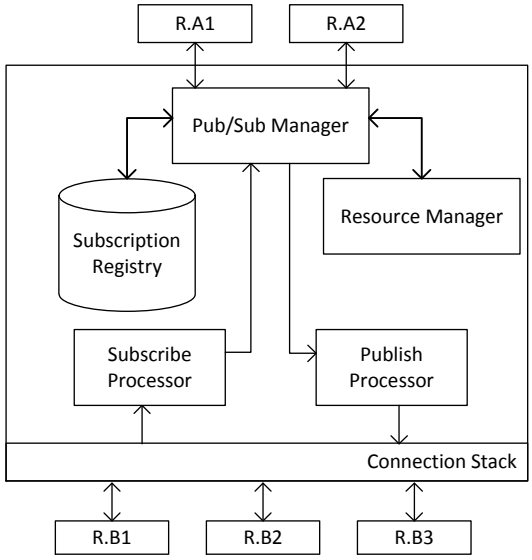


Figure 7: Resource Publish and Subscribe

whereby registered applications can get notifications of update messages from the subscribing resources. Difference is being that the *Pub/Sub Manager* may deal with the UI threads of the applications. In this case, the Android message *Handler* can be defined instead in the UI thread and registers to the message queue that is running as background service. The UI logic can then be called when the resource update is detected.

context awareness Context awareness is included to the framework design to introduce intelligence that evaluates contextual status of a device, opening up space for fine-grained custom services. The approach taken by the framework relies on evaluating the declared contextual requirements attached to the resource description and updates.

For example, a contextual requirement for, user (identified by one's device) being present at a specific location in a room, or preferred device with bigger screen but only when it is in use, and etc. can be translated accordingly and attached to the resource's own description. When the descriptor is picked up by the consumer device, a *Context Analyser* evaluates these requirements against the device's current status, either hardware or software. As a result, filtering and prioritising of resources can be performed, so only relevant discovery is passed on to the upper layer on the stack. For example, in US.MR, Bob won't see any available services such as the wireless projector or the share folder until he touches in using the NFC reader, or user David will receive a word document on his tablet for viewing instead of on the much smaller mobile phone. The *Context Registry*, working as a look up table for resources against various context names, needs to be maintained so that the *Resource Manager* can resolve if any matching resources that have the contextual requirements evaluated *true* exist, hence the discovered resource can be exposed accordingly. Figure 6 shows how the relevant elements fit within resource *Advertise/Discovery*.

Optionally, as this applies both ways, a resource may explicitly decides only under certain conditions shall the framework advertise in the network. However, this requires the middleware to monitor the conditions closely, which further complicates the implementation.[22]

As far as the publish/subscribe is concerned, it is not difficult to establish how resource updates with contextual conditions propagates the exact same way as before. So can the similar evaluation be performed to quantify how relevant the context update is with its subscriber. Also, this applies to publishing as to subscribing symmetrically. This may be simplified if context requirements is not allowed in the more frequent status updates but only in the resource advertise/discovery descriptors. [24]

3.3.5 Trust and Security

As a transversal function, trust and security is also considered carefully across the framework and we argue that: [3] [31]

- Device identities must be managed so it is guaranteed only the recognised devices are allowed communication with those paired up together.
- Inter-device communication within a personal Cloud must be encrypted so the information exchange between devices is not exposed to the shared network. Certificate-based solutions may be employed to distribute keys that can be used to sign or encrypt data. [25]
- Similarly, certificates or certificate-like solutions can be used to maintain a single user identity on top of multiple personal devices. Once one of the paired devices is authenticated and authorised, so should the others. This is feasible if sharing an agreed token among all paired devices. Take SIM cards for example, they provide both physical, software and even infrastructural trust and security. A mobile-centric solution will certainly be tangible.

- In the ad-hoc environment, every node of device with such framework operating may broadcast information as commanded, it is a critical to distinguish what information made available is trustworthy. Allowing user interactions for the confirmation may be a justifiable solution, as what's existed already (e.g. verification of Bluetooth pairing using PIN). [23] [French et al., 2012] also proposes an intelligent model of determining the level of trust in the smart office space. [16]

- As we are suggesting embedding information like publish/subscription and context requirements to resource description, trust and security requirements are equally suitable for taking the same approach. Security handshakes will be performed before any legitimate communication takes place.

4 Prototyping

An enterprise service suite with both client and server side components has been developed. The use cases described in Section 2 gives the outline of the part feature set concerning the device framework. In addition to the technical comments along the architectural discussions in the preceding sections, system configuration and more implementation details of the prototyping are given here.

The software system caters for meetings in a cooperate environment. Physical meeting rooms and meetings are mapped and various resources are made available. Some resources are globally available to all users, like cooperate directory, IM, while the others are only available locally and contextually, like shared file storage for a meeting, conference bridge. Meeting participants come into the meeting with mobile devices like phones and tablets. In order to assist remote users, the network is configured so that authorised local resource would have also public interfaces based on meeting setup. Figure8 shows the overall architecture design.

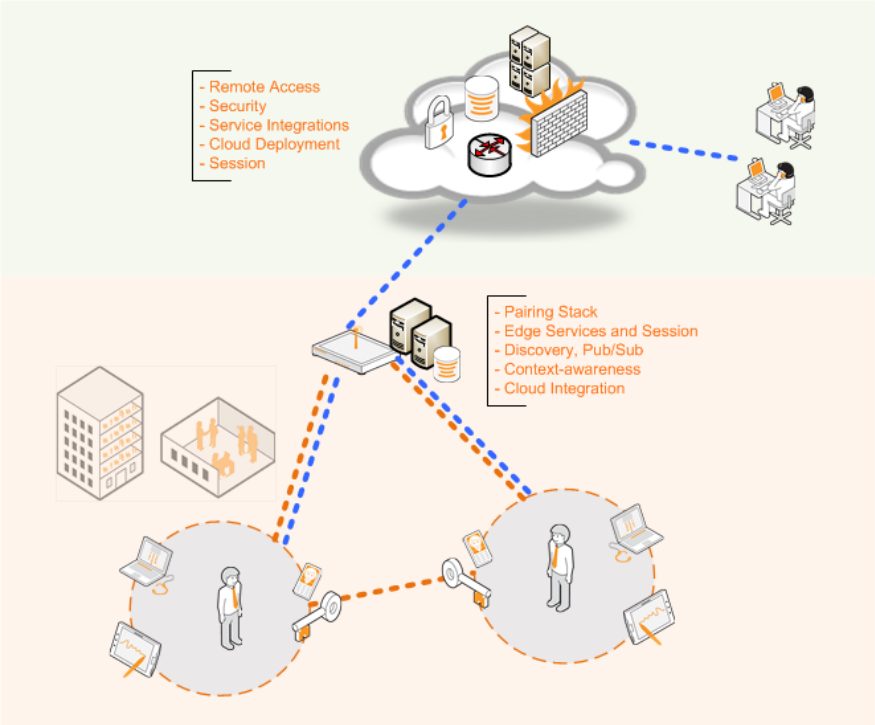


Figure 8: Prototype system design

The user has the Android client installed on the devices, bundling with the device framework. To enter a scheduled meeting, the user would touch the smart phone against the NFC tag at the entrance to the meeting room. On the application level, this is to notify the system of the user's presence and perform the registration accordingly, while on the lower device framework level, this has set the user's mobile to a state to accept meeting rooms resources, since the user's context has changed. Although NFC tag bound to a location is used here, other device peripherals or software information can be used for the context analysis. However, the physical presence of the mobile implies trust upon the user carrying the compatible device. Figure9 are the screenshots before and after the resources are discovered for a demo meeting session.

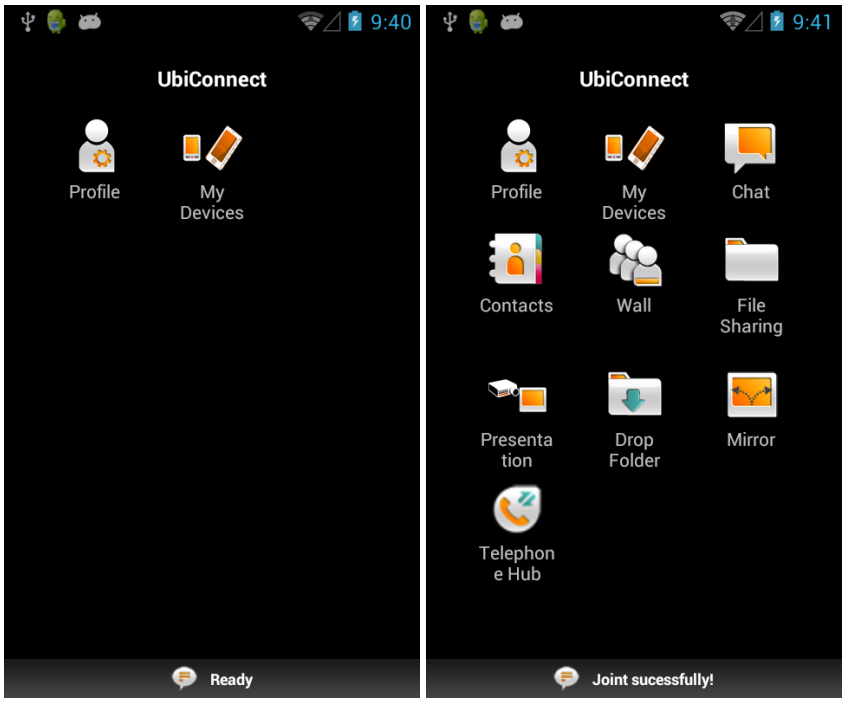


Figure 9: Home screens from mobile client with discovered resources

For users with multiple devices, *My Devices* exposes the pairing capability of the device framework. Figure10 shows how the user's mobile phone and tablet become aware to each other after having the tablet scan the QR code generated by the phone . Furthermore, the demo application has been implemented in the way that uses the device framework API to pass the login intent from the phone to the table, so that even though the user has used the phone to checkin to the meeting, the tablet will automatically set to the same state as the phone and log in to the system without further operations required on the device.

During the meeting, the user may also receive file transfer from other users in the meeting. Using the timestamp of last typing on the device, although both the phone and tablet receive the notifications, only the most recent active device would open the downloaded automatically.

The system also supports remote user access. Logging in from the web portal, the client application will receive public resource IP to connect, unlike the local IPs used by local participants.

The prototype system allows the creation and interlinks between the personal, local and remote networks. The creation of the personal Cloud takes place physically around the user. A community of other users, their devices and other local resources is managed locally where user interactions present. Finally the Cloud is able to integrate with such local community to build a complete information architecture.

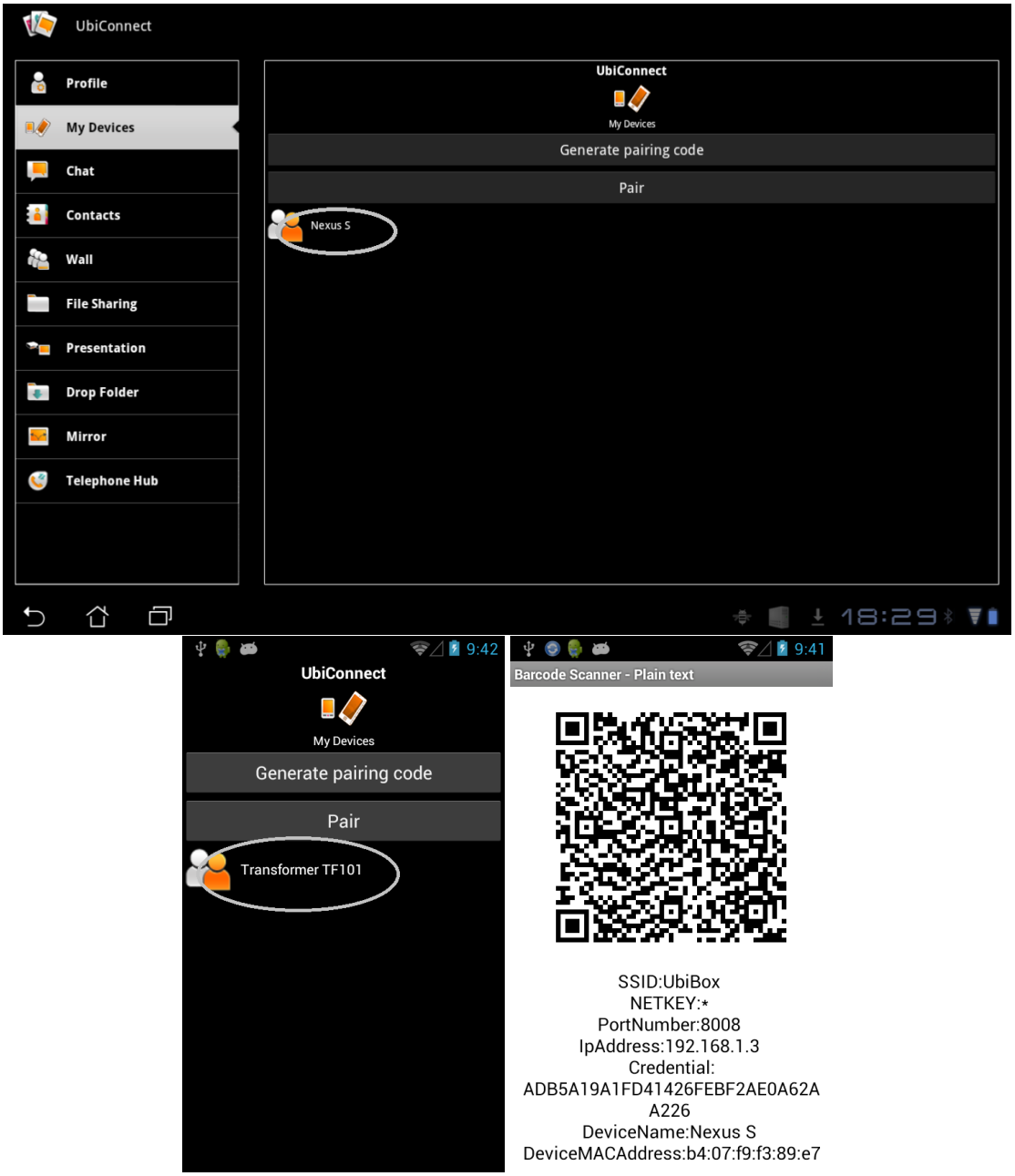


Figure 10: Mobile and tablet are paired up by the use of QR code

5 Discussion and Related Work

Considerable amount of work has been done in the areas where many aspects of the solution proposed in this paper are relevant, such as ubiquitous comping, pervasive computing, Cloud computing, mobile ad-hoc network, smart home, Internet of Things and so on. Each has own influence and brings pros and cons to the attempt of delivering an tangible overall solution. Given the understanding of the today's users' behaviours with their devices, other users, the intelligent environments they are in and the Cloud, with perceived future device and network capability in mind, it has been possible to pinpoint the key factors and technical feasibility for meeting the desired requirements.

Egami, et al.[8] present a Cloud platform that centrally manages ubiquitous services. The proposal clearly details the key components of service discovering, context processing and service interfacing in its layered architecture, which are sensible for building context-aware ubiquitous systems. The platform would be suitable for pre-defined and rather static environments like the living room use case presented in the paper. Even though complete transparent service negotiation and execution remains challenging, the infrastructure-driven architecture is by design less portable and flexible to cope with today's highly dynamic device-driven activities. Therefore extra focus on the devices that play major roles in creating the ubiquitous environment is necessary. Nevertheless, the service repository record design proposed in the paper can be useful for building the generic data structure for communicating service descriptions. Put in a more efficient format, like JSON, with extensions, it is reusable as one variation of resource discovering, registering and context analysis explained in this paper. Finally, structured methodology, like SOA, or UDDI in the matter, can be inefficient and have interoperability problems in the IoT or WeboT domains.

Ha, et al.[18] has also demonstrated another approach leveraging semantic web. It efficiently improves how knowledge of resource capability and interfaces can be obtained and used in highly dynamic way for service composition and execution. However, decentralising or distributing the service discovery and knowledge repository would largely benefit the ad-hoc use cases. Compact and responsive stack would then be critical to be embedded on personal devices. Furthermore, similarly, due to its own limitations, protocols like SOAP, BPEL, could cancel out certain flexibility advantages Semantic Web brings to the implementation.

A lightweight, device-initiated, peer-to-peer agent that can be embedded on various devices, is to fundamentally address the dynamic and robust nature of ad-hoc networks. It is then possible to allow immediate pairing under group user identify, transparent data communication and service execution across paired devices for upper level application design, fully exploiting the advantage of local networks and interactions, and with context awareness beyond the Cloud. The proposed framework solution is designed to serve such a purpose that bridge the personal, local and hosted network areas.

6 Conclusions

The current paper presents our view on enabling ubiquitous applications in the context of personal Cloud that consists of more than one personal devices, which can be paired up by means of physical contacts, and discusses in depth the resulting key requirements for a tangible device middleware framework that focuses on local interactions with surrounding devices, users, as well as those in the Cloud, with the given use case scenarios. A mobile-first approach has been taken towards the technical feasibility followed by the implementation of the prototype middleware. It is demonstrated feasible to implement a compliant middleware following the prosed design of the framework, meeting the objectives of providing a common solution for easing inter-device communication and enabling intuitive user interactions with the surroundings, while challenges in managing end-to-end trust and applying intelligence in resource negotiation remain as work for the future.

Nevertheless, with the growing capability of smart devices, the increased interest in multi-screen user experience, the maturing of Cloud services, and the development of Internet of Thing, Web of Things and M2M, ubiquitous solutions are believed to benefit both consumers and service or infrastructure providers. While there are still challenges in front as we have learned from our exercise, it has been more practical than ever to bring these concepts in to reality.

References

[1] R. Agarwal and M. Motwani. Survey of clustering algorithms for MANET. *International Journal on Computer Science and Engineering*, 1(2):98–104, 2009.

[2] W. Alliance. Wifi direct. `http://upnp.org/sdcps-and-certification/`.

[3] A. Arabo, Q. Shi, and M. Merabti. Towards a Context-Aware Identity Management in Mobile Ad-hoc Networks (IMMANets). In *Proc. of The 23rd IEEE International Conference on Advanced Information Networking and Applications Workshops/Symposia (WAINA'09), Bradford, UK*, pages 588–594. IEEE, May 2009.

[4] A. C.-E. Chan. Distributed symmetric key management for mobile ad hoc networks. In *Proc. of the 23th AnnualJoint Conference of the IEEE Computer and Communications Societies(INFOCOM'04), Hong Kong*, pages 588–594. IEEE, March 2004.

[5] L. Chen and M. Prokopi. A resource-aware pairing device framework for ubiquitous cloud applications. In *Proc. of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'12), Palermo, Italy*, pages 252–258. IEEE, July 2012.

[6] T. Clausen and P. Jacquet. Optimized Link State Routing protocol. IETF RFC 3626, October 2003. http://www.ietf.org/rfc/rfc3626.txt.

[7] E. Curry. *Message-Oriented Middleware*. John Wiley & Sons, Ltd, 2005.

[8] S. M. Egami, Koichi and M. Nakamura. Ubiquitous cloud: Managing service resources for adaptive ubiquitous computing. In *Proc. of the 2011 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops'11), Seattle, USA*, pages 123–128. IEEE, March 2011.

[9] D. et al. Jsr-914, java message service. `http://jcp.org/aboutJava/communityprocess/final/jsr914/index.html`.

[10] D. C. et al. Javascript object notation. `http://json.org/`.

[11] R. Fielding. Representational state transfer. `http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm`.

[12] N. Forum. Near field communication. `http://www.nfc-forum.org/home/`.

[13] U. Forum. Universal plug and play. `http://www.wi-fi.org/discover-and-learn/wi-fi-direct`.

[14] U. Forum. Internet gateway device. `http://upnp.org/specs/gw/igd2/`, 2010.

[15] X. S. Foundation. Extensible messaging and presence protocol. `http://xmpp.org/xmpp-protocols/`.

[16] T. French and N. Bessis. Towards a context-aware and adaptable room system for intelligent trusted office-spaces in smart cities. In *Proc. of the 6th International Conference on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS'12), Palermo, Italy*, pages 148–154. IEEE, July 2012.

[17] Google. Android cloud to device messaging framework. `https://developers.google.com/android/c2dm/`.

[18] J.-C. S. Ha, Young-Guk and Y.-J. Cho. ubiHome: An Infrastructure for Ubiquitous Home Network Services. In *Proc. of IEEE International Symposium on Consumer Electronics (ISCE'07), Dallas, Texas, USA*, pages 1–6. IEEE, June 2007.

[19] R. Handorean, R. Sen, G. Hackmann, and G.-C. Roman. Context aware session management for services in ad hoc networks. In *Proc. of the 2005 IEEE International Conference on Services Computing. Orlando, Florida, USA*, volume 1, pages 113–120. IEEE, July 2005.

[20] IEEE 802.11s. HWMP protocol specification. The Working Group for WLAN Standards of the Institute of Electrical and Electronics Engineers, 2006.

[21] IETF. Multipurpose internet mail extensions. RFC 2045, RFC 2046, RFC 2047, RFC 4288, RFC 4289 and RFC 2049.

[22] M. Ilka, M. Niamanesh, and A. Faraahi. A Context-aware and Group-based Service Discovery in Mobile Ad Hoc Networks. In *Proc. of the 2012 International Conference on Systems and Informatics (ICSAI), Yantai, Shandong, China*, pages 588–594. IEEE, May 2012.

[23] V. Kärpijoki. Security in ad hoc networks. available at]`https://www.cs.tcd.ie/Hitesh.Tewari/papers/netsec00_manet_sec.pdf`, 2000.

[24] J. Liu, D. Sacchetti, F. Sailhan, and V. Issarny. Group management for mobile ad hoc networks: design,

implementation and experiment. In *Proc. of the 6th international conference on Mobile data management (MDM'05), Ayia Napa, Cyprus*, pages 192–199, March 2005.

[25] J. V. D. Merwe, D. Dawoud, and S. McDonald. A Survey on Peer-to-Peer Key Management for Mobile Ad Hoc Networks. *ACM Computing Surveys (CSUR)*, 39, 2007.

[26] OASIS. Web services notification. `https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsn`.

[27] OASIS. Web services reliable messaging. `http://docs.oasis-open.org/ws-rx/wsrm/200702/wsrm-1.2-spec-os.html`.

[28] C. E. Perkins and P. Bhagwat. Highly dynamic destination-sequenced distance-vector routing (DSDV) for mobile computers. *ACM SIGCOMM Computer Communication Review*, 24:234–244, October 1994.

[29] C. E. Perkins and E. M. Royer. Ad-hoc on-demand distance vector routing. In *Proc. of the 2nd IEEE Workshop on Mobile Computing Systems and Applications (WMCSA'99), New Orleans, Louisiana, USA*, pages 90–100. IEEE, February 1999.

[30] A. Ranganathan and R. H. Campbell. A middleware for context-aware agents in ubiquitous computing environments. In *Proc. of the ACM/IFIP/USENIX 2003 International Conference on Middleware (Middleware '03), Rio de Janeiro, Brazil, LNCS*, volume 2672, pages 143–161. Springer-Verlag, June 2003.

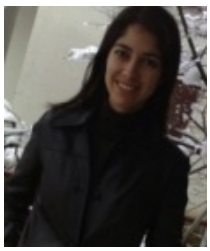
[31] R. Shankaran, V. Varadharajan, M. A. Orgun, and M. Hitchens. Context-Aware Trust Management for Peer-to-Peer Mobile Ad-Hoc Networks. In *Proc. of the 33th Annual IEEE International Computer Software and Applications Conference (COMPSAC'09), Seattle, Washington, USA*, pages 588–594. IEEE, July 2009.

[32] W3C. Simple object access protocol. `http://www.w3.org/TR/soap/`.

[33] W3C. Web services addressing. `http://www.w3.org/Submission/ws-addressing/`.

[34] J. Y. Yu and H. Chong. A survey of clustering schemes for mobile ad hoc networks. *IEEE Communications Surveys and Tutorials*, 7:32–48, 2005.

Liang Chen is a Senior software engineer experienced in Web-Telco convergence, personal Cloud and mobile contactless research and development. He has technically led R&D projects, providing architectural solutions, technical feasibility assessment, conducting research studies and running day to day development actives. As an experienced server side Java developer, he has a MSc degree in Distributed Computing from University College London and was a research fellow at UCL for scientific workflow modelling and orchestration on the Grid under the EPSRC funded Open Middleware Infrastructure Institute. He has publications to IEEE conferences and contributed to several books.



Maria Prokopi is working at Program managing Information Science & Cloud Platforms research streams in Orange Labs UK. For the past five years she oversee studies and prototypes on the overall ecosystem and the different trends contributing to the growth of Personal Cloud with a focus on customer experience enhancement. As Social Media is a strong growing model tied Personal Cloud aspects, she is also managing the team activities on Social Media topics like the dynamically of social networks and the development of social graphs. The results of the work are fed to the Group in order to define a strategic position on the topic and initiating development programs. Over the 10 years career with Orange she has filled patents, authored IEEE international conference proceedings and lead white/positioning papers.