Mobile Resource Management for a Better User Experience: An Audio Case Study

Roberto Morales, Beatriz Otero, Marisa Gil

Computer Architecture Department

Universitat Politècnica de Catalunya

{rmorales, botero, marisa}@ac.upc.edu

Abstract

Ubiquitous environment's research has evolved considerably over the last years. The wide range of mobile devices, their high diversity and mobility have raised a variety of challenges being resource management a predominant, and therefore attracting special attention in the research community. The Composable-Adaptive Resource Management (CARM) middleware library provides a flexible infrastructure where personal devices create seamlessly on-demand interconnections links to share ubiquitous resources. In this paper we present a CARM interesting use case, consisting of an improved audio listening experience by sharing a high quality audio resource. A proof-of-concept implementation is provided, and a testbed comprised of two CARM enabled mobile phones with Bluetooth connectivity making use of remote device's audio resource is described. Our approach demonstrates the importance and usability of enhancing the dynamic resource sharing experience without altering the bandwidth efficiency.

1 Introduction

As Mark Weiser envisioned [1], embedded and mobile devices have become part of people's daily life. Most people are already carrying mobile phones, portable music and game players or even wearing smart-jackets with small sensors. All these heterogeneous devices are highly diverse from software to hardware [2]; first, devices include different operating systems (e.g. Windows Mobile, Symbian, Android, iPhone OS, etc.) and a great variety of programming interfaces; second, devices may include one or more sort of communication technologies (new or improved ones like: NFC, ZigBee, Bluetooth, or WiFi), different displays sizes and resolutions, and diverse sound quality systems. With this pool of assorted ubiquitous resources, devices have the opportunity of *i)* dynamically use some of them to improve their capabilities or *ii)* dynamically add new ones. For example, devices can aggregate new better resources to improve sound quality (e.g. with higher definition speakers), video (e.g. higher resolution or bigger display), or to augment its capacity (e.g. remote storage), among others. However, the capabilities of maintaining access to the available resources while moving is a major challenge that yet research activities have not fully addressed. There exist numerous projects [3], [4], and [5] that address remote resource's management, but they are limited to specific environments (e.g. home or office). Other projects such as [6] consider additional environments but they rely on a fixed infrastructure or central servers to manage resources. Therefore to maintain the information about availability and allocation of resources in a moving environment a mobility-enable solution is still required, to overcome these issues we propose the use of the Composable-Adaptive Resource Management (CARM) [7] middleware which takes advantage of device's mobility and dynamically manages the set of available resources to exploit it. With CARM, applications are able to use resources by opportunistically annexing friendly devices they encounter and managing its available resources in a seamlessly and transparent manner. CARM contains a Core module that interacts directly with ubiquitous devices to provision shared resources among applications, and provides a high-level management of them. CARM also provides a Resource Abstraction Layer which allows access to a specific functionality of each device, dealing in this way with the high diversity of devices.

In this research work, the authors present a specific use case in order to demonstrate the effectiveness of CARM middleware. The use case is based on the assumption that many devices have high level capabilities and are able to share them. Considering this, we have chosen to share the high level sound quality regardless of the audio source format (mp3, phone call or multimedia file), i.e. the audio is streamed directly to the high definition speakers in real-time in the form of raw audio packets. Hence, the contribution of this paper can be summarized as follows: *i)* we show how to improve the sound quality between two mobile phones by increasing the quality of user's listening experience, where the first mobile phone has the minimal audio set to listen music with poor quality (mono audio), and the second mobile phone has built-in medium quality sound; and *ii)* we implement a prototype which uses the CARM API to develop the audio use case.

The remainder of this paper is organized according to the following structure: The basics of the CARM middleware library are described in Section 2. Section 3 proposes the study of an audio use case scenario towards the implementation of a testbed and a CARM-based audio resource sharing model. Implementation details and testbed considerations are explained in Section 4 and a proof-of-concept test and preliminary results are outlined in Section 5. Section 6 reviews the background work related to our research. Finally the main conclusions and future directions are drawn in Section 7.

2 The CARM middleware library

CARM provides an integrated approach in which operating systems and applications collaborate to manage seamlessly and transparently resources in ubiquitous environments dealing with their hardware and software dependencies properly. The middleware deals with situations in which users usually need additional or better resources than what is currently available, interacting directly with other ubiquitous devices to provision shared hardware resources, thus providing high-level management of such resources. Next, we will summarize important benefits:

- Transparency: in CARM, the complexity of management of ubiquitous resources remains hidden to the applications.
- Portability: CARM was built in modules to separate specific platform dependent parts from the independent ones.
- Quality: the quality of service can be improved in CARM by selecting the best resources from the pool of available resources (e.g. using external high quality speakers to provide a better audio experience) .
- Extensibility: CARM can extend the devices capabilities with other resources like display, microphone, audio, etc.

2.1 CARM Features

The CARM middleware is basically comprised of two main modules:

1. *The CARM Core* module is the manager engine which orchestrates and coordinates all tasks, its main goal is to manage the available resources in the vicinity keeping updated all runtime information regarding availability, an example of this information is outlined in Table 1. To accomplish the resource management, the Core module provides a scheduler with simple allocation policies and available resources monitoring. This information will be updated in various situations: *(i)* when a

Table 1: Related runtime information

Device descriptor	Data related to the remote devices	Address, capabilities, etc.
Resource descriptor	Data related to the remote resources	Device owner, state, availability, capability, etc.
Context descriptor	Other useful context information	Proximity, position, battery level, etc.

new resource is requested, *(ii)* when the managed resources finish their assigned tasks or *(iii)* every certain period of time.

The communications are done through an independent layer which is configurable according to user needs and availability. From the local pool of communication technologies, CARM selects a suitable one, according to the current situation, taking into account user preferences and system restrictions. Once a communication technology is selected, CARM messages are sent to the collaborative group; as the group changes fast (due to the high mobility of devices) the communication protocol demands a light-weighted protocol to avoid communication delays, therefore in CARM, we have designed a light-weighted protocol basically using two communication channels; one to exchange signaling messages and another one to transfer specific data such as audio packets. Since we are working with heterogeneous environments, the CARM middleware also includes a Resource Abstraction Layer on a higher operating system level and will be introduced next.

2. *The Resource Abstraction layer (RAL)* module is responsible of mapping a resource request from partners to the actual underlying platform. This layer exploits the specific access to the restricted low level OS's set of native functions when required, and makes it available to the library. Basically, the RAL creates a full-duplex (two-way) access channel to configure and control the associated resource through the CARM Core module. If required, RAL can also configure additional channels to communicate to the OS and partners.

2.2 CARM Communications Protocol

To support the orchestration and maintenance of local and remote resources, CARM communications protocol considers two different channels to exchange messages:

1. A signaling channel used to exchange control information messages among devices, this information may include configurations, availability, changing states, and other control related details.

2. A data channel used to transmit substantive data information relevant to a shared resource (e.g. audio packets).

When initial communication is established, devices exchange synchronization and coordination messages through the dedicated signaling channel which provides a *request* type of notification to the server device. When the *request* signal is detected, the server device could receive additional requester's device information in order to process and respond the received request. After synchronization and coordination processes are done, for resource information sharing (a device making use of a shared resource) both devices make use of the data channel to send and receive the corresponding data; the Figure 1 shows the sequence diagram for a client requesting a resource. For the initial testbed implementation, CARM's communication protocol considers a basic packet format comprised of a header and a payload.

The header implementation mainly contains the packet type information; when used in the resource exchange process other fields for controlling source and resource destination might be added.

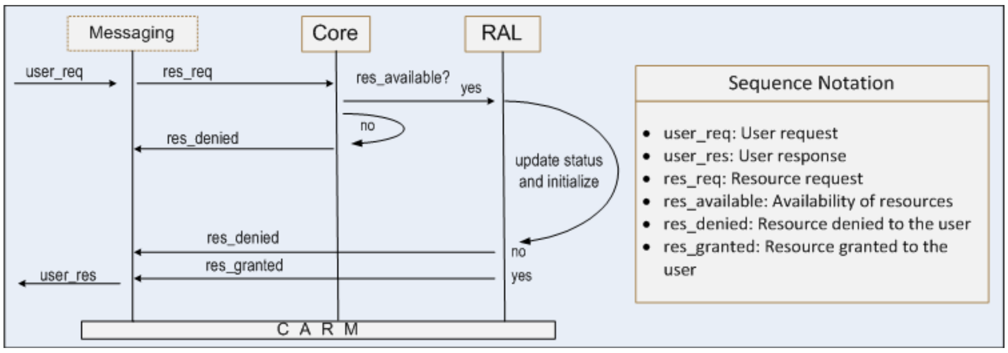
Messaging
Core
RAL
user_req
res_req
res_available?
yes
no
res_denied
update status and initialize
res_denied
no
res_granted
yes
user_res
C A R M
Sequence Notation
• user_req: User request
• user_res: User response
• res_req: Resource request
• res_available: Availability of resources
• res_denied: Resource denied to the user
• res_granted: Resource granted to the user

Figure 1: Resource request communication sequence.

The payload data consists of a sequence of bytes representing the expected data in correspondence to the packet type and the current process. In the resource exchange process, this byte sequence represents specific resource shared information (e.g. audio streams), and in the signaling process this sequence represents other information like security related information or additional requested information (e.g. list of available shared resources). Finally, it is worth to mention that, both security and resource discovery processes are beyond the scope of this paper.

2.3 CARM API

The management of the local resources is done in the CARM Core (for convenience we will call it "Core"). The Core keeps track of local resources and, the information related to each resource is stored in memory using hash tables which are updated every time a change occurs. Then, resources are published to make them available to other devices, when these devices want additional resources, the Core captures all requests and depending on the availability it assigns the corresponding resource to the requester in turn. The Core applies a FIFO policy to assign each resource and it is hold until the assigned device relinquishes it, providing also a standard API for sharing and controlling ubiquitous resources (e.g. audio or video). The API was designed to be flexible with respect to protocols, resources and features supported by various devices. Table 2 list the functions supported by the API implementation, note that not all CARM functions are documented here.

3 An Audio Use Case

To show the effectiveness of CARM, we have developed a prototype which uses the CARM middleware library. Since granted audio resources on mobile phones are at the moment extremely tight due to size and power constraints, we have considered for the initial CARM testbed implementation, sharing the high quality audio resource. Next we present the audio use case which involves communication and collaboration between distinct devices:

> *The group of friends has just sat down at its favorite restaurant and Claire decides she wants to play her last favorite song to the group. As Bob owns a high quality sound device and after they push a few buttons on their phones and engage their built-in micro-speakers they are ready to play the music on Bob's phone.*

Our scenario consists of a Personal Area Network (PAN) with a group of users and

Table 2: CARM API

Function	Description
`void Core();`	Creates all internal instances: repositories and communication module.
`void initialize(type);`	Initializes with a specific configuration. The type parameter specifies which role to apply as server or client.
`void configureLocalResource(type, configuration);`	Configures the local resource with appropriate parameters.
`void finalize();`	Close the current connection.
`void requestResource(type, configuration);`	Realizes the request call to a server when a device needs additional resources.
`void resourceRelinquish(type);`	Relinquishes the assigned resource when the task is already finished.

a set of known components (e.g. mobile phones). With CARM it is possible to extend the audio capabilities and have real time control of multiple sound channels positioned and moving dynamically around the user, providing an immersive 3-dimensional sound environment, allowing amazing effects, and optimizing user's audio listening experience. Of course, other possibilities exist, but for simplicity in this paper we will focus on just one shared resource.

The proposed prototype works in two ways: *(i)* a server device exposing the high definition audio speakers resource to the environment using the CARM API, and *(ii)* a client device willing to improve its limited audio resource, benefits from the audio sharing process by making use of the available resource, as shown in the Figure 2. For the proposed use case implementation different considerations should be taken into account just as presented in the next section.

4 Implementation Details

This section describes the basics of a testbed implementation and the considerations taken towards developing the above mentioned audio use case.

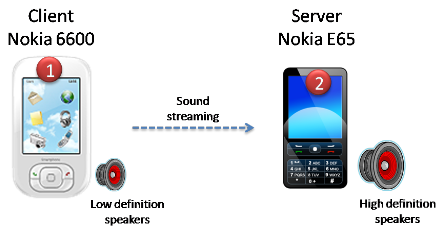

Figure 2: Example of and audio session sequence between two devices.

4.1 Testbed and tools

The CARM middleware library works with well-known devices for the sake of simplicity (resource discovery is not our main topic), even though adding devices and services are not dynamically allowed, these could be resolved through a variety of plug-in technologies. To evaluate our CARM proof-of-concept we use two mobile phones: a Nokia E65 that runs a 220MHz Dual CPU ARM9 processor, and a Nokia 6600 that runs a 104MHz ARM9 processor. Both devices are based on Symbian OS and the interaction is done through Bluetooth connectivity as shown in Table 3.

The initial Core system prototype was designed on Java 2 Micro Edition (JavaME) because of its wide manufacturers sup-

Table 3: Device Specifications

Device	OS	Audio
Nokia 6600	Symbian OS v7.0s Series 60 2nd Edition	Audio speakers (low quality)
Nokia E65	Symbian OS v9.1 Series 60 3rd Edition	Audio speakers (medium quality)

port, and open standards adoption. Even though JavaME provides the facilities necessary to target many platforms simultaneously, currently its standards lack the ability to access devices specific functionality [14]. To overcome this platform dependency, the RAL approach provides device-specific services through its abstraction layer and it is written in the native development environment (Symbian C++) because of the needed libraries that are only accessible from the native domain.
We have considered this configuration because of the great variety of devices that include these capabilities, however we are aware that this solution is not applicable to those with specific requirements such as iPhone or Android based mobile devices.

4.2 CARM API usage

The CARM API is able to create new virtual resources and connect devices together through the importation of the library. The core of the middleware is the Core class, where an application must create the Core instance, and it is advisable to store this instance as a singleton or some other handy location. The listing 1 shows a common pseudocode sequence to execute CARM in devices.

Once a Core is created (line 1), we need to initialize the middleware as a client (resource consumer). Then resources within it can be created and initialized also. Local resource initialization allows configuring resources behavior once the sharing has been started. For example, if we are going to use better speakers from other device, it would be nice to put the sound in silent mode of local resource to enjoy

Listing 1: Calls to put CARM in action

```
1  CARMCore carm = new CARMCore();
2  carm.initialize(asClient);
3  carm.configureLocalResource
4             (CRESOURCE_AUDIO, conf);
5  carm.start();
6  ...
7  carm.requestResource
8             (CRESOURCE_AUDIO);
9  carm.relinquishResource
10            (CRESOURCE_AUDIO);
11 ...
12 carm.finalize();
```

the new feature (lines 2-4). In order to allow coordination between the several nodes of the system the application must start the middleware instance. Internally, the middleware create the links between the available devices to allow the resource sharing (line 5). In this state, applications are ready to request additional resources to other devices if needed. In case no devices were found, an exception is thrown and no additional actions are taken. To request and relinquish a remote resource methods in lines 7 and 9 should be called. The function parameter denotes the resource in order to request or relinquish. Also, if not available resources are found an exception is thrown. Finally, once remote resources are used, the middleware needs to release all instances created (line 12).

5 Proof-of-concept Testing

5.1 Execution flow overview

In our audio use case, CARM can have two roles: as a server or client. These roles are hidden in the middleware but it is possible to configure the role with the initialization parameter described in the API above. In the prototype, the device environment configuration is to put the first device (Nokia 6600) as client, and the second device (Nokia E65) as the server. With this configuration, the server device is the central console in which the action of playing music takes place, taking ad-

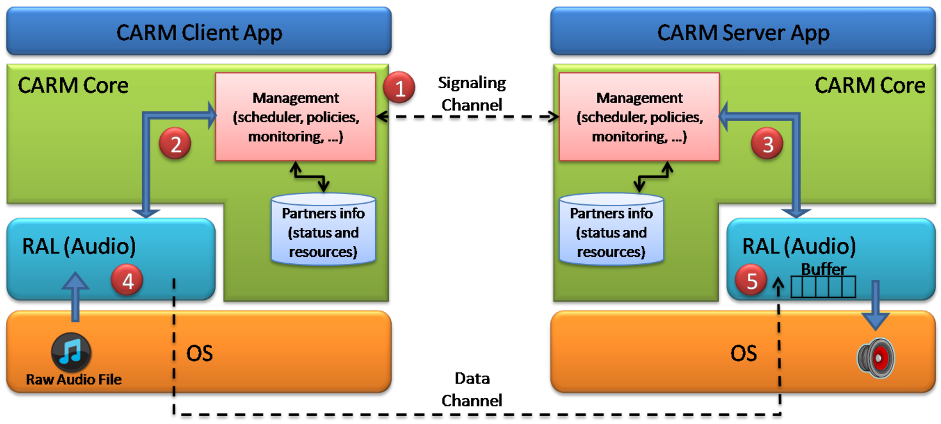

Figure 3: Composable-Adaptive Resource Management internals overview. Example of and audio session sequence between two devices.

vantage of the better sound quality of speakers. The client should be aware of this situation and will read decoded audio streams to redirect the stream flow to the server device. In turn, the server receives these streams and put them to the local audio queue. The server must prepare the local audio system to play the available queue buffer. Note that for the sake of simplicity and despite its importance, our example considers an audio file with raw audio content (PCM16 format) which acts as a real hardware audio stream. However considering a more advanced configuration, audio packets will be taken directly from the audio device hardware. Following the Figure 3, the audio sharing in the client side consists on the next steps:

1. Device recognition: at the beginning, devices exchange credentials in order to gather information about each other. Device information would consist of battery level, computational power, and the list of shared resources. Once devices are bounded[1], the client device requests for an audio resource. During this process, devices use the signaling channel to exchange communication information.

[1]At this point we will assume that the two devices were previously identified and connected.

2. The client management sends an internal ready signal to prepare the local resource.

3. In parallel to the client management, the server management sets ready its corresponding resource. In this state, the RAL listens for incoming audio stream.

4. When the RAL client is ready, it begins the execution of its assigned task. In this case, starts to read decoded (PCM16 format) stream data from file. Each data chunk is 4Kb size. However, the data size will depend on the complexity of the audio file. When one chunk has been already read, the RAL module sends the data to the server using the data channel. This process repeats until the end of the file.

5. The RAL server starts the reception of the audio stream. Every chunk of data is copied into a buffer and once the buffer is filled, a "playback" event is launched to start playing the received audio.

6. When processes are already done, CARM releases the corresponding resources.

5.2 Preliminary Results

The Core middleware library is approximately 3500 lines of Java code for a 71Kb of library size, and 848 lines of Symbian C++ code for the resource abstraction layer component with a size of 28Kb. Initial results demonstrate the viability of the CARM middleware and how CARM is able to effectively stream audio data between two mobile devices.

The experimental setup environment consisted of a small piconet[2] comprised of two mobile devices and a L2CAP connection between them with a data transmission rate of 721 kbit/s (typical on mobile devices). As showed in Figure 4, we have measure the time consumption considering the following processes: *i)* connecting the two devices, *ii)* requesting a resource, *iii)* relinquishing a resource, and *iv)* disconnecting. For a more accurate measurement we have taken into account statistical variations by performing the tests 100 times.

Preliminary measurements indicates that the connection process is the most time consuming process but acceptable for a session. Transmission of audio packets present a minor packet loss ratio due to environmental conditions affecting the sound quality (noisy sound). It seems that due to the small library size, power consumption cannot be considered as a major drawback, however we are looking for effective methods in order to have more realistic and accurate measurements.

6 Related Work

The interest in ubiquitous computing environments has given rise to a proliferation of systems that allow resources to be dynamically discovered and utilized.

So far, research directions in this area can be classified into two distinct types of works, one based on standards and specifications, and another based on projects with similar goals.

To dynamically manage resources, standards and specifications research groups such

[2]A piconet is a computer network consisting of devices using the Bluetooth technology protocols to allow one master device to interconnect with up to seven active slave devices

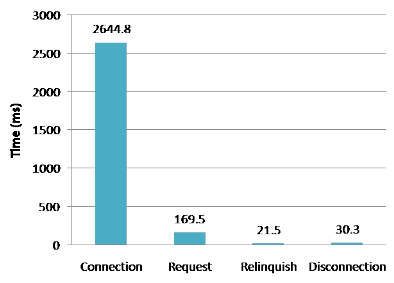

Figure 4: Processes time consumption.

as DLNA [8], OSGi [9] and UPnP [10] allow devices to control multimedia contents and other devices (smart-home) using certified devices. The main difference with CARM is that we can manage resources on the go, i.e. in ubiquitous environments, and we use commercial off-the-shelf (COTS) devices without any special library or server infrastructure. The Dynamic Composable Computing from Intel Research (DCC) [11] share similar directions as CARM in the sense that allows users to easily and seamlessly extend the capabilities of their mobile device with the nearby resources in their environment, and furthermore allows its resources to augment other devices in the locality. However the main drawback is that DCC was designed to run on more powerful devices like UMPCS and MIDs, while CARM has the capability to work with mobile devices with scarce or constraint resources. Another important aspect is that CARM communicates with typical device communication technologies like Bluetooth and DCC uses a proprietary UWB radio prototype thus limiting its usability.

There exist other projects that deal with resource hungry operations (discovery, memory, or computational) like MobiGo [12], which consists of a middleware system that migrates service states to achieve seamless mobility. Such services are saved and resumed in other environments with adaptation of available resources; and Kimberly [13] which enables fixed infrastructure rapid software provisioning for transient mobile device use. Moreover to augment the capabilities of a mobile device Kimberly uses virtual machine technology. These projects among others share a common characteristic, they rely on fixed infrastructure.

7 Conclusions and Future Work

Due to the increasing complexity of software and the high diversity of embedded devices, middleware technologies have become paramount in current mobile systems, thus playing a fundamental role in pervasive environments. In this paper we have demonstrated the benefits of using the CARM middleware library to share mobile resources in ubiquitous environments. An audio use case involving sound quality improvement between two mobile phones was presented in order to show the effectiveness and usability of CARM. CARM provides an integrated approach in which operating systems and applications collaborate to manage seamlessly and transparently resources by properly dealing with their hardware and software dependencies.

Although there are still a number of open issues like security, scalability, reliability and power consumption, we believe that this paper gives an important contribution to the area by presenting CARM architecture usage for dynamic resource management in ubiquitous environments. Currently we are working on the integration of a rigorous authentication and authorization mechanism based on the use of Attribute Certificates (AC) to guarantee resource protection and to prevent unauthorized access.

Naturally, future research directions point out to the inherent privacy challenges of ubiquitous environments, also to new communication strategies between CARM's server and client (to improve bandwidth efficiency) and clearly an extension of the middleware API considering a new set of technologies and mechanisms able to support other systems such as iPhone or Adroid-based mobile devices.

Acknowledgments

This work has been supported by the European Commission in the context of the SARC Integrated Project (EU contract 27648-FP6), the HiPEAC European Network of Excellence and the Ministry of Science and Technology of Spain, the European Union (FEDER) under contract TIN2007-60625, the Generalitat de Catalunya (2009-SGR-980), and the Mexican National Council for Science and Technology (CONACyt). All products or company names mentioned herein are trademarks or registered trademarks of their respective owners.

References

[1] Mark, W. (1999). *The computer for the 21st century*, SIGMOBILE Mob. Comput. Commun. Rev. 3(3): 3-11.

[2] George, H. F. and Z. John (1994). *The Challenges of Mobile Computing*, Computer 27(4): 38-47.

[3] Bellavista, P., A. Corradi, et al. (2001). *Mobile Agent Middleware for Mobile Computing*, Computer. 34: 73-81.

[4] Kon, F., T. Yamane, et al. (2001). *Dynamic Resource Management and Automatic Configuration of Distributed Component Systems*, Proceedings of the 6th USENIX Conference on Object-Oriented Technologies and Systems (COOTS'2001).

[5] Schubert, L., A. Kipp, et al. (2009). *Service-oriented operating systems: future workspaces*, Wireless Communications, IEEE 16(3): 42.

[6] Stephen, S. Y. and K. Fariaz (2004). *A context-sensitive middleware for dynamic integration of mobile devices with network infrastructures*, J. Parallel Distrib. Comput. 64(2): 301-317.

[7] Morales, R. and M. Gil (2008). *CARM: Composable, Adaptive Resource Management System in Ubiquitous Computing Environments*, Advances in Soft Computing. J. M. Corchado, D. I. Tapia and J.

Bravo, Springer Berlin / Heidelberg. Volume 51/2009: 335-342.

[8] Digital Living Network Alliance, http://www.dlna.org/.

[9] OSGi Alliance, http://www.osgi.org.

[10] UPnP, http://www.upnp.org.

[11] Roy, W., P. Trevor, et al. (2008). *Dynamic Composable Computing*, Proceedings of the 9th workshop on Mobile computing systems and applications. Napa Valley, California, ACM.

[12] Xiang, S. and R. Umakishore (2007). *MobiGo: A Middleware for Seamless Mobility*, Proceedings of the 13th IEEE International Conference on Embedded and Real-Time Computing Systems and Applications, IEEE Computer Society.

[13] Wolbach, A., Harkes, J., et al. (2008). *Transient Customization of Mobile Computing Infrastructure*, MobiVirt'08: The First Workshop on Virtualization in Mobile Computing, Breckenridge, CO.

[14] Chan, E., J. Bresler, et al. (2005). *Gaia Microserver: An Extendable Mobile Middleware Platform*, IEEE International Conference on Pervasive Computing and Communications.

[15] Larry, R. (2009). *A Virtualization Infrastructure that Supports Pervasive Computing*, IEEE Pervasive Computing 8(4): 8-13.