12-2013

uOS : A resource rerouting middleware for ubiquitous games

Fabricio N. Buzeto
Universidde de Brasilia, fabricio@aluno.unb.br

Miriam A M Capretz
Western University, mcapretz@uwo.ca

Carla D. Castanho
Universidade de Brasilia

Ricardo P. Jacobi
Universidade de Brasilia

uOS : A resource rerouting middleware for ubiquitous games

Fabricio N. Buzeto*, Miriam A. M. Capretz† Carla D. Castanho*, Ricardo P. Jacobi*
* Departamento de Ciências da Computação
Universidade de Brasília - UnB
Brasília, Brazil 70.910-900
fabricio@aluno.unb.br, {carlacastanho,rjacobi}@cic.unb.br
† Department of Electrical and Computer Engineering
Western University
London, Canada N6A 5B9
mcapretz@eng.uwo.ca

***Abstract*—Ubiquitous computing (ubicomp) relies on the computation distributed over the environment to simplify the tasks performed by its users. A smart space is an instance of a ubiquitous environment, composed of a dynamic and heterogeneous set of devices that interact to support the execution of distributed smart applications. In this context, mobile devices provide new resources when they join the environment, which disappear when they leave it. This introduces the challenge of self-adaptation, in which smart applications may either include new resources as they become available or replace them when they become unavailable. Ubiquitous games combine *ubicomp* and computer game technologies to enrich user's experience and fun. Such games may benefit from different input and output resources offered by mobile devices. To support the development and deployment of ubiquitous games, this work presents the uOS middleware. Using a DSOA (Device Service Oriented Architecture) based architecture and lightweight service discovery protocols, uOS ensures compatibility among resources, providing resource rerouting between heterogeneous and limited software and hardware platforms. The uMoleHunt game is presented to illustrate the practical application of uOS.**

I. Introduction

In recent decades, computation power has been distributed across a large number of new devices and objects. This reality gave birth to research topics such as "Pervasive Computing", the "Internet of Things" and "Mobile Computing"'[13]. "Ubiquitous Computing" [29], also known as *ubicomp*, places itself in the same context. Its purpose is to use all this computing power to help users perform everyday tasks [30] while requiring as little attention as possible. The environment, then, must be intelligent and pro-active in its actions [31], and for this reason it has been called a "smart space".

Realizing such a scenario involves the development of new technologies, especially those involving human interaction. Touch screens and voice controlled gadgets are good examples that have become very common over the last five years. However, despite research to make user interfaces more natural, traditional devices such as keyboard, mouse and screen continue to be the most common mode of human-machine interaction. Alternatives like HUD (Heads-Up Display) [26], a concept more than a century old, have seen a very slow adoption among users.

One of the tools that can aid in this scenario is the development of so called *ubigames*. This kind of application takes advantage of the technology surrounding the user to build a game. Such games can be developed according to three different propositions [23]. The first is aimed at the game itself and is based on the belief that all technology can be used to enhance fun. The other two approaches use games as a way of shortening the path to adoption of new concepts and technologies. The first approach takes advantage of the greater engagement of users with games than with "serious applications". This is a welcome effect of the playful aspects of such activity. Another characteristic of games is that players want to be challenged, which contributes to the acceptance of new ideas and prevents the rejection of those that initially could be considered odd. The second approach consists of embedding gaming and ludic features in other applications, thus easing their reception by the user. This advantage has been explored by pervasive applications such as FourSquare[1], GetGlue[2], and AccidentBucket [21], in which a game layer is placed over the application to enhance user interaction.

A problem that can benefit from these characteristics lies in selecting the best available resource option in the environment. For example, a game happening in a living room can choose from many different types of inputs. The game console joystick, the TV remote control, the smart phone or even a microphone can be suitable options for the task. Moreover, devices need to exchange knowledge in order to implement smart distributed applications. This raises the challenge of choosing the most adequate services among them [10]. Unlike the way it is achieved today, through direct user intervention, in the smart space applications are expected to take on this responsibility. In this scenario the smart space must be able to identify the possible options in the environment and even choose pro-actively the best ones for the task at hand.

Given that the environment can host multiple users and devices and that these possess a wide variety of usable resources, the smart space is indeed a very complex environment. In addition, both users and devices can enter or leave unpredictably. Leaving to each ubigame the responsibility of

[1] http://foursquare.com
[2] http://getglue.com

handling all the challenges involved in finding, choosing and using these resources is not the best option. Migrating those tasks to a software layer that manages resources and devices may significantly simplify the development of ubigames

This work presents uOS, a middleware that provides dynamic resource rerouting among multiple devices to support the development of ubiquitous games. The middleware follows the DSOA [7] (Device Service Oriented Architecture), which provides a high-level abstraction of how to organize the environment and the use of the uP [6] (Ubiquitous Protocols) set for enabling lightweight communication. This allows the middleware to tackle three important challenges among this type of games: platform heterogeneity, device interaction and discovery, and device limitations. This paper also presents a game developed with the support of uOS to exemplify resource reconfiguration at runtime.

This paper is arranged in the following order. Section II describes ubigames and how they are used to validate new concepts. Section III highlights middleware research focused on resource rerouting. Section IV describes the uOS middleware and its features, while Section V further describes uOS features. Section VI presents how the uOS can be used to build a game application, while Section VII presents the uMoleHunt game that adapts to different resources available in the smart space. Finally, Section VIII covers the conclusions reached and the topics for future research.

II. UBIGAMES

Research into how to make better use of contextual information gave birth to *ubiquitous games* (ubigames) [4]. These games are also known by other names like *pervasive games* and *context-aware games*. The main purpose of these is to blend the concepts of ubiquitous computing and games. Mcgonigal [23] presented three different ways to envision this concept: as motivation, means, or ends.

Ubigames can be analysed according to four dimensions: environment, flexibility regarding players, user interaction and contextual data [5]. Contextual data were the first focus of attention for this category of games. Similarly to other ubiquitous applications, location data received the largest portion of early research focus. This importance is understandable because it makes it possible to target action towards where the user is. In ubigames, this information has been explored in different types of environment and for varying degrees of player flexibility. For example, the "Touch Space" [8] game focuses on indoor environments and team play, while "Treasure Hunt NFC" [14] is designed for outdoor spaces and multi-player interaction.

Exploring the way that users interact with recently developed devices or with a new approach to old technologies is where ubigames stand out. Games like "Uncle Roy All Around You"[3] and "Hitchers" [11] present new views on how location information is gathered from the user, with the aim of using less invasive methods. In the first game, the location is either provided directly by the user or inferred by the patterns perceived while the user interacts with the game map. In Hitchers, location is derived from cellphone towers near the device. Even though the precision is not very high, this approach provides a level of abstraction suitable for the purpose. The "FantasyA" [25] game explores totally different types of interfaces. Using a doll named "SentToy", it can sense the user's emotional state and apply it to the game.

Exploring new interfaces or giving new meaning to well-established ones is a great achievement. To achieve this, resource rerouting plays a very important role, enabling different options to be exploited during runtime without needing to adapt the game. However, these games are not capable of integrating devices dynamically as they become available in the smart space. The Multi-User Application Platform [19] (MUPE) enables the development of ubigames over heterogeneous devices. It relies on a centralized server that contains the contextual data and game logic. The devices in the environment exchange XML data that carry both user inputs and UI (User Interface) descriptions. This approach limits the information that these devices can provide without requiring user interaction. Hybrid Pastry [32] (HP), on the other hand, provides a P2P protocol for integrating devices over an unstable network. It ensures that key-value messages are delivered between nodes. This strategy is efficient but does not ensure compatibility between services during runtime, which limits the evolution of the game. Summing up, resource discovery, access and validation over heterogeneous platform remains unattended to ubiquitous gaming development.

III. RESOURCE REROUTING

The smart space contains a great variety of resources of which many are available for use. Resource rerouting makes it possible to redirect the binding between resources and applications. This binding is usually placed on local options so that, better-suited alternatives can be used. Most applications delegate the decision about which resources to use to the underlying platform or to fixed options during development time. Take as an example the input and output interfaces used for a cellphone game. While engaged with the device, the user relies on the screen for both display and user input. This is the best choice for the application in this context. However, when the game changes and only the screen cannot keep up with this level of engagement, other choices can be used. Suppose that after starting a game session, the user moves towards another room. There a game console joystick and a large LCD screen are available, which are better suited to the task. Redirecting the display and the user input to these, without interrupting the game, enables the player to benefit from these better options.

The ease with which a system can adapt to changes in binding of the resources it requires is defined as adaptability [12]. According to the ubicomp purpose of aiding users in their tasks in the most invisible way possible, adaptability can be classified into the following three levels:

1) *Direct Interaction*: This level of adaptability requires the full attention of the user. He must personally choose which devices must be redirected among all available options. It is also the user's responsibility to decide which characteristics to consider when making this choice. Because of its simplicity, this is the most common type of adaptability. It can be found in situations like plugging a secondary monitor into a laptop, which requires the user to choose which monitor to use and to perform the act of connecting it manually, thus realizing his intent.

2) *Suggested Interaction*: Instead of letting the user decide among all available options, this level presents a selection of options. This approach aims to ease the task of choosing which one to pick. This can be achieved by applying a simple filter on the available set or by ordering the resources by their suitability to the task. For example, when choosing a new input device, those closer to the user can be shown first, because they are the easiest to reach.
3) *Automatic Interaction*: The highest level of adaptability delegates the choice completely to the system. The system is responsible for selecting the best option without requiring interaction from the user. This approach achieves the most invisible interaction possible, but also presents great challenges in providing a suitable result. Understanding the patterns and contexts engaged by the user plays a very important role for applications operating at this level.

Many applications have a fixed set of resource options available for use, which is established at compilation time. This approach cannot be considered for automatic interaction level because adaptability cannot happen during runtime in response to changes in the environment. It is also noteworthy that mixing suggested with automatic interaction can provide good results. In this approach the system can decide which resources to use, but when in doubt, can ask the user to choose among the best identifiable options.

Accessing remote computing capabilities is the main purpose of techniques like VNC [28] (Virtual Network Computing) and RDP [17] (Remote Desktop Protocol). Unfortunately, these options demand high bandwidth while forcing applications to run on the target device without truly sharing its resources. On the other hand, initiatives like Bluetooth[3] and UPnP [18] (Universal Plug and Play) provide capabilities focused on enabling resource redirection for smart houses. The Digital Living Network Alliance[4] (DLNA) developed a communication standard to connect media electronics, and should also be highlighted. This standard classifies each device into a category according to the features it can provide. Built on top of the UPnP, it enables devices to connect with each other, but does not allow applications to redirect such resources.

According to authors' knowledge, literature in resource rerouting in games is sparse. Therefore, general ubiquitous applications will be discussed here. GMote [15] and Cliky[5] have demonstrated that both academia and the market have shown interest in resource rerouting capabilities. GMote is one of many applications that provide easy control of PC functions using a smart phone. It simulates mouse and keyboard inputs using peer-to-peer (P2P) messages that are translated to the underlying operating system. Cliky is an application demo for the capabilities achieved by the GaiaOS middleware [9]. It acts as an interface that makes it possible to move a computer mouse using a PDA as an input interface.

Applications need to access the capabilities present in the smart space seamlessly. While DLNA ensures that interfaces are met when using a service, it is limited to a few media resources defined. On the other hand, Bluetooth allows a broader connectivity but does not ensure that interfaces are compatible. Ensuring both characteristics while allowing for different resources types joined at runtime remains a challenge in ubigames development.

The DSOA [7] presents a set of concepts that aid in modelling the environment, the applications and their interactions considering such issues. Although, as an architecture, it does not provide an implementation to meet such requirements. Figure 1 shows a graphic representation of how these concepts relate to each other. The most basic concept is the *smart space*. It is defined as a set of connected computing devices collaborating to help users. *Devices* are the entities responsible for hosting both applications and resources. These are the basic concepts that interact during the life time of a smart space.

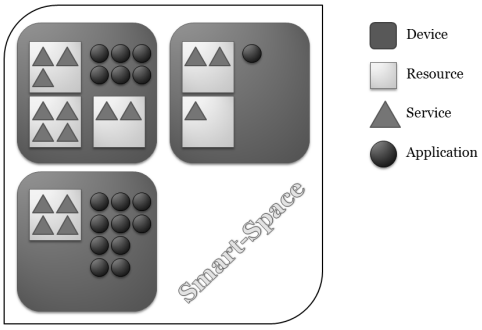

Figure 1. Concepts in a DSOA smart space [7].

A *resource* is a group of functionalities which are logically related. These functionalities are represented by *services* accessible though defined interfaces. A resource interface is defined by an identifier and the set of services that it provides. Service interfaces are composed of the name of the service and its parameters. Interaction with these services can happen either synchronously or asynchronously (through events). This difference is important because the first approach is mostly applied to commands and queries while the second is useful when responding to changes in context. In a DSOA smart space, a cellphone is considered a device that contains various resources. One of them is a camera. It can provide services with the capability of taking snapshots, streaming video, or even alerting when movement is detected.

Applications, on the other hand, implement a set of rules and behaviours relating users and resources in the smart space. They are the entities responsible for providing the intelligence that is required by ubicomp. They rely on the resources available to coordinate their actions while facilitating the tasks performed by users. For example, using the camera resource mentioned earlier, an application can identify which user is present in the smart space. This information can be used to adjust the game settings according to that user's profile.

IV. uOS Middleware

The uOS is a middleware that provides resource rerouting to support the development of ubiquitous games. It is based on DSOA [7] architecture and the protocol set uP (Ubiquitous Protocols) [6]. It is available for multiple software

[3]https://www.bluetooth.org/Building/HowTechnologyWorks/ProfilesAndProtocols/Overview.htm
[4]http://www.dlna.org
[5]http://gaia.cs.uiuc.edu/html/demos.htm

platforms. Its current version is supported by Java VMs like JSE[6] (Java Standard Edition), JME/CLDC[7] (Java Micro Edition/Connected Limited Device Configuration) and Dalvik[8]. Although these versions are based on the Java language, using uP removes the platform constraint. It enables many other software, hardware, and communication platforms to be integrated with uOS applications without needing any kind of modification or configuration. The DSOA provides a set of concepts that makes it possible to model and organize the smart space, with a focus on how resources can be shared among devices and applications.

The following subsections will present the ecosystem around the middleware (Section IV-A), while Section IV-B presents the layered organization of its components.

A. Ecosystem

Figure 2 provides an overview of the ecosystem around the middleware uOS. Its main components are the underlying software and hardware platform, the Network Plugins and the Resource Drivers, which are explained below.

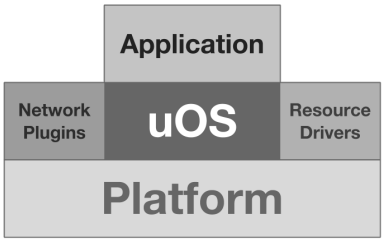

Figure 2. High level view of the uOS middleware.

1) Platform: The underlying platform must consider both the hardware and software that the middleware will abstract. On the software side all layers involved are included, like the operating system and virtual machines such as Java or Python.

2) Network Plugins: Network plugins enable the middleware to provide different types of communication technologies to connect applications and resource drivers. They play an important role because many different technologies are available and many devices can be restricted to one or a few of them. The middleware provides plugins that enable communication for Bluetooth, Sockets (TCP and UDP), and RTP. There is also a Loopback plugin that simulates local communications in memory, enabling consistent data transfer in the middleware. These plugins provide two important features for the middleware. The first is the capability of creating data channels between devices. These data channels are passed transparently to applications that are not aware of the technology that provides them. The second feature is called "Radar" because it provides a way to perceive which devices are available in the environment.

3) Resource Drivers: Drivers are entities which implement the services provided by resources in the devices. Because they are responsible for integrating physical components, they retain access to the underlying platform that is abstracted through their interface. They also possess full visibility of the middleware capabilities. This enables the creation of logical drivers which are not related to physical resources. It also makes it possible to create composite resources, which provide services resulting from the combination of other available resources.

4) Applications: As the main component supported by the middleware, applications have access to an API that enables all the features it provides. This interface makes resource and device discovery and service consumption transparent, regardless of the platform. Relying only on the interface provided by services, any available capability can be put to work. Applications can also exchange messages with each other to coordinate their actions. Complementary information about the device and its neighbours in the smart space are provided as well. This information includes which devices exist, the platform they are running on and which resources they possess.

B. uOS Middleware Components

The inner components are arranged in a three-layer model as shown in Figure 3. The bottom layer is the *Network Layer*, which is responsible for managing the communication streams. The *Connectivity Layer* is responsible for translating the data into messages understandable by the upper layer. Finally, the *Adaptability Layer* manages the ecosystem that revolves around the middleware.

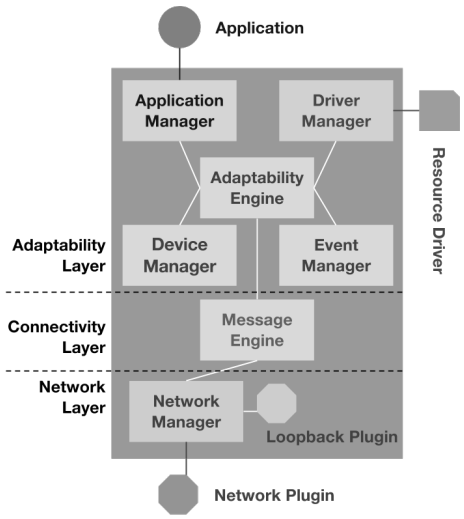

Figure 3. uOS components follows a layer model.

1) Network Layer: The bottom layer is made up of network plugins and the *Network Manager* component. It is responsible for managing both input and output communications. Its main task is to translate requests from the upper layers to the appropriate format transmitted by the network plugin. Using identifiers provided by the uP, it is possible to match the desired data channel with the proper plugin. Input data as well as device discovery are delegated to the upper layers to handle, leaving only the interface management for the network layer.

2) Connectivity Layer: The *Message Engine* understands the uP protocol set and acts as a translation interface between the other layers. Inbound data streams are received as raw bytes; translation reveals the type of each message allowing

[6]http://www.oracle.com/technetwork/java/javase/overview/index.html
[7]http://www.oracle.com/technetwork/java/cldc-141990.html
[8]https://code.google.com/p/dalvik/

to discover which action needs to be performed. The same applies to outbound information which must be encoded into the proper format before transmission.

Specific managed translators can also be applied to data streams. These are responsible for performing transformations on the transported data such as compression or information encryption. One of the applications that this feature provides is described by Ribeiro et al. [27] as a lightweight security protocol that establishes a secure channel between devices with limitations.

3) Adaptability Layer: This layer is in charge of coordinating all interaction through the middleware. Its main job is performed by the *Adaptability Engine*, which forwards each request to the appropriate component according to its responsibility. The purpose of this layer is to enable all service interactions to happen transparently to external entities, both local and remote.

The *Driver Manager* controls the resource driver instances that are available in each device. When a service request arrives, it is responsible for choosing the most adequate instance to perform it. The application life cycle is managed by the *Application Manager*. Messages exchanged by applications also come through this component to find their correct recipient.

The *Device Manager* acts as an endpoint when a device is discovered by the bottom layer. It gathers information about the device and its resources. This information is used by applications and the Adaptability Engine to decide which services to use. The *Event Manager* coordinates all event subscriptions for asynchronous services. Inbound events are redirected to the appropriated listeners while outbound subscriptions are assigned to the corresponding drivers.

V. Features of the uOS

Providing easy access to resources is of major importance in the uOS. Complementary to this, a set of other features are also provided to improve application development.

A. Resource Sub-typing

Certain devices may have similar resources suitable for tasks not predicted by the applications. A game like tic-tac-toe only needs a feature that make it possible to know which spot the user has selected. A resource that provides the coordinates to which the user is pointing and the desire to take action is enough. Although the game was developed to use a mouse for such tasks, if other options were available, they could be used instead. For example a joystick or a touch screen interface, among others, could provide the same information.

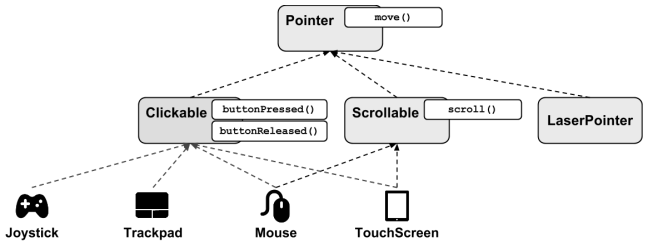

Figure 4. uOS resource subtyping scheme.

If the resource interface were composed of a single unique identifier, all interchangeable options would need to be known beforehand. Otherwise, the application would not be expecting them and could not make use of their capabilities. Delegating this task to the application is not only complex, but also hard to maintain up-to-date during runtime. To address this problem, the DSOA resource interface definition includes the possibility of stating an equivalence with other resources. By taking advantage of this feature, the middleware is capable of broadening the results for a resource query to all equivalent instances available. Figure 4 shows a representation of the equivalence between Pointer resources using the support provided by DSOA. Similarly to what is provided by other projects [22, 20] all information about resources and their relationships is also provided though an ontology to applications. This provides a flexible way to extract new knowledge about which resources can be used in each situation.

B. Proxies

This feature takes advantage of the fact that multiple types of networks are accessible transparently through the middleware, while some devices possess multiple communication interfaces. In this case, devices restricted to only one interface could exploit the fact that others have more flexibility and use them as a bridge, providing access to their resources. Figure 5 shows a cellphone restricted to Bluetooth and a PC restricted to Wi-Fi communication, while the laptop can interact using both types of connections. This makes it possible to create a path between the two disconnected devices. The proxy is created during service consumption and is responsible for relaying the data between the two endpoints. This is implemented by cooperation between the Adaptability Layer and the Connectivity Layer. When a device is able to act as a proxy and recognize unreachable devices, it announces their resources as its own. Later, when these resources are requested, the Message Engine recognizes them and forwards the message to the appropriate device.

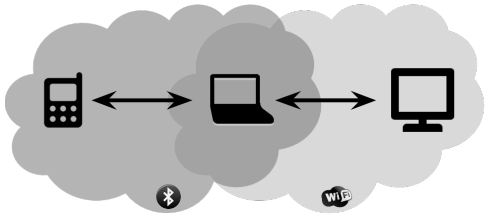

Figure 5. The uOS proxy capability enables devices in different networks to share resources.

C. Resource Discovery

The basis of resource discovery are the network plugins, which are responsible for providing a list of available devices in the smart space. Each plugin uses its own strategy for doing this. The Bluetooth stack has its own mechanism to find nearby peers, while Ethernet has many available options. For example, it could be use ARP (Address Resolution Protocol) broadcast, IP (Internet Protocol) scan or SNMP (Simple Network Management Protocol) queries. Once a device is encountered for the first time, its resources are recorded in a local database, making them visible to applications.

Operating this way, the uOS can work in a peer-to-peer infrastructure without requiring any centralized entity. The radar provides a means to detect devices in range, while the proxy feature makes it possible to reach those that are not within range. However, some very limited devices cannot afford to maintain an active radar or even to store all data about the surrounding environment. In this case, the uOS allows more capable devices to operate as resource registers. They operate like yellow pages, when a limited device is discovered by a register device, the limited device stores only its address. This way, when a resource needs to be queried, the register will be used to access data about what is available in the smart space.

VI. Implementing a Game using uOS

As shown before, applications are responsible for turning the environment a smart space. Games perform this role, taking advantage of the capabilities available to engage with the player. The uOS aims to enable easy access to resources. To achieve this, the middleware provides to both *Drivers* and *Applications* an object named "Gateway". This object is responsible for providing all necessary services regarding the device and the smart space. It is worth noticing that even if the implementation is in Java, the use of *uP* protocols makes it possible to access resources available through other platforms.

Figure 6 shows subset of the class diagram involved in the creation of a uOS application.

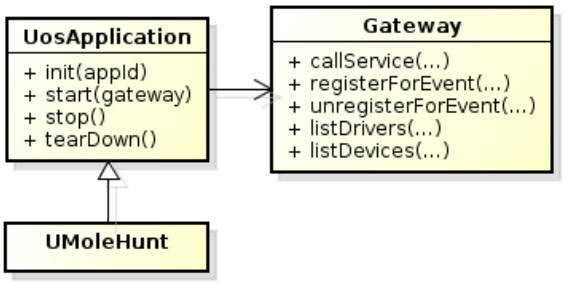

Figure 6. Classes for the creation of a uOS application.

A. Creating an application

The life-cycle followed by an application is composed of four distinct stages:

- *Initialization*, which happens only once; here the application can set up all necessary resources for use during its execution.
- When execution *starts*, the application gains access to middleware capabilities through a "Gateway" instance.
- The application can be *stopped* during its runtime, for example, when it is sent to background on a cellphone or tablet. During this phase, information must be stored for the time when the application is started again.
- When an application is removed (*torn down*) it must clean up any resources attached to it.

During each of these stages, an object provides access to the ontology shared among applications in the device. This object controls changes to ensure consistency throughout execution.

B. Acessing the smart space

The Gateway provides methods that enables easy access to middleware capabilities through the following methods:

- *callService*: Makes a call to a synchronous service returning its response.
- *registerForEvent*: Registers a listener for an asynchronous service on a resource driver.
- *unregisterForEvent*: Stops listening for the desired event.
- *listDrivers*: Provides an inventory of all known resource drivers in the smart space.
- *listDevices*: Lists all devices discovered in the environment.

All these methods are based on the resources discovered in the devices found using the radar on the network layer. Using the resource sub-typing scheme, it is possible to search for and to use services based on a generic type, avoiding multiple calls.

VII. uMoleHunt

The uMoleHunt is a Ubiquitous Game that takes advantage of the resources in the smart space to adapt itself to the number of users available. In this game, each player embodies a member of a different group. The player is either part of a Mafia that controls a smart space (like the laboratory), or a cop trying to put an end to this reign of crimes. Both of these share a similar purpose: they want to discover the name of the informers (moles) inside the organization. For the cops, these informers are a vital part of the investigation, providing information about the criminals. On the other side, the bandits want to find their weak links and terminate them before they are caught and sent to jail.

As shown in Figure 7, the game is played using a public display available to all users in the smart space. A game session is composed of a series of guessing runs. At the beginning of a run, each team is presented with a different mole to be detected. Each team starts a run with 26 points, a vague hint about the name and how many characters it contains. A run involves taking turns between the teams and players and proceeds as follows:

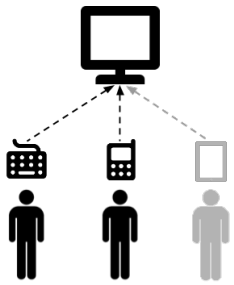

Figure 7. An overview of the uMoleHunt game.

- The player must guess a letter in the name of the mole.
- If the guess was right, the positions corresponding to the letter are uncovered on the screen, displaying part of the name.
- If the guess was wrong the team loses one point.
- If the team makes a sequence of right guesses they create a 50% chance for the other team to miss a right answer. The team does not know whether this was really wrong or part of the bonus.
- If the team reaches 13 points a more helping guess about the name is shown.
- If a team gets the name right, it wins the run and receives the number of points remaining.
- If a team reaches zero points, it loses the run.

The first team to reach 100 points wins the game session. The names are chosen from a database made up of known science personalities with minor changes to resemble mobsters. For example, Newton is displayed as "Isaac Rubberface Newton". Each personality also has a list of hints, about his or her accomplishments in life, which are displayed as described before.

```
List<DriverData> keyboards = gateway
        .listDrivers("uos.keyboard");
for (DriverData keyboard : keyboards){
        gateway.registerForEvent(this,
                keyboard.getDevice(),
                keyboard.getDriver(),
                "keyPressed");
}
```

Listing 1. Finding Resource Drivers and using them.

The current version of this game runs on Android smartphones and PCs accepting any keyboard resource available in the smart space as a player input. The current version of uOS middleware has keyboard drivers available for PCs, JME phones (using the keyboard), and Android devices (using the touch-screen interface). This enables the game to incorporate the most common personal devices that users possess. Listing 1 shows how keyboard devices can be found and accessed in the smart space using the methods provided by the gateway. The code needed to handle a key being pressed is presented in Listing 2.

```
public void handleEvent(Notify event){
        if (event.getEventKey()
                .equals("keyPressed")){
                // handling code
        }
}
```

Listing 2. Handling an asynchronous event.

Because the game involves a single public display that must be shared among all players it is best suited for indoor spaces. This kind of space is best suited for games where player flexibility is based on single or team play [5]. Although in this case players were split into teams, the game can be characterized as a multi-player game, because there is no restriction on how many players each team can include. Few indoor games focus on multi-player or collaborative flexibility as in "Hoodies and Barrels" [2], "Save the Princess!" [24], and "Day of the Figurine" [16]. Unlike these, uMoleHunt does not require any kind of intervention by an administrator to adapt the game session as new players, devices or environment configurations are used.

Using the middleware uOS, the game runs on any device that has a screen available. Dynamically, as new devices become available, if they have a keyboard resource or a subtype compatible with the game they are invited to be part of the game session. New players are incorporated during runtime, requiring no interaction by the user to adapt the game.

VIII. Conclusions and future research

The idea of ubicomp is to help users perform everyday tasks in the most invisible way. In this way, the computation present around the user can be useful while not demanding much attention. One of the tools used to attain this goal is to exploit the pervasiveness of devices available in the environment and the network that connects them together.

Connecting a wide range of devices and heterogeneous platforms is a difficult task for individual applications to handle. This paper has presented the uOS middleware, which helps in creating this kind of applications. Specifically, it focuses on enabling devices to share resources among each other easily. This permits local resources to be redirected to other, more suitable ones.

In this context, ubigames have been used to validate how new technologies respond in real-life situations. This paper has presented the uMoleHunt game developed in this research. It takes advantage of the resource rerouting capabilities provided by the middleware to create a dynamic, multi-player indoor game. uOS can also be used to develop other ubiquitous applications [1] and games.

The implementation of uMoleHunt has been presented in order to illustrate the uOS capabilities for the development of more complex games. Current work also is focused on expanding these capabilities in order to incorporate "Code Mobility" as one of its features. This approach makes it possible to build games that adapt dynamically to the devices in the smart space despite variability in their available resources.

IX. Acknowledgment

The authors gratefully acknowledge the financial support to Fabricio Buzeto from CAPES (*Coordenação de Aperfeiçoamento de Pessoal de Nível Superior*) and CNPq (*Conselho Nacional de Desenvolvimento Científico e Tecnológico*) for the period of this research. We also thank Matheus Pimenta for the assistance provided.

References

[1] L.A. Almeida, F.N. Buzeto, A.H.O.R. Castillo, C.D. Castanho, and R.P. Jacobi. Hydra: An ubiquitous application for service rerouting. In *9th Int. Conf. on Ubiquitous Intelligent Computing and 9th Int. Conf. on Autonomic Trusted Computing (UIC/ATC)*, pages 366–373, 2012.

[2] I. Arroyo, I.A. Zualkernan, and B.P. Woolf. Hoodies and barrels: Using a hide-and-seek ubiquitous game to teach mathematics. In *Proc., 11th IEEE Int. Conf. on Advanced Learning Technologies (ICALT)*, pages 295–299, 2011.

[3] S. Benford, W. Seager, M. Flintham, R. Anastasi, D. Rowland, J. Humble, D. Stanton, J. Bowers, N. Tandavanitj, M. Adams, J. Farr, A. Oldroyd, and J. Sutton. The error of our ways: The experience of self-reported position in a location-based game. In N. Davies, E. D. Mynatt, and I. Siio, editors, *UbiComp 2004: Ubiquitous Computing*, volume 3205 of *Lecture Notes in Computer Science*, pages 70–87. Springer Berlin Heidelberg, 2004.

[4] S. Björk, J. Holopainen, P. Ljungstrand, and K. P. Akesson. Designing ubiquitous computing games - a report from a workshop exploring ubiquitous computing entertainment. *Personal Ubiquitous Comput.*, 6(5-6):443–458, 2002.

[5] F. Buzeto, A. Helena Castillo, C. Castanho, and J. Ricardo. What is going on with ubicomp games. In *XI Brazilian Symposium on Computer Games and Digital Entertainment - SBGAMES*, pages 1–7, 2012.

[6] F. N. Buzeto, C. D. Castanho, and R. P. Jacobi. up: A lightweight protocol for services in smart spaces. In *4th Int. Conf. on Ubi-Media Computing (U-Media)*, pages 25–30, 2011.

[7] F. N. Buzeto, C. B. P. Filho, C. D. Castanho, and R. P. Jacobi. Dsoa: A service oriented architecture for ubiquitous applications. *Int. Journal of Handheld Computing Research*, 2(2):47–64, 2011.

[8] A. D. Cheok, X. Yang, Z. Z. Ying, M. Billinghurst, and H. Kato. Touch-space: Mixed reality game space based on ubiquitous, tangible, and social computing. *Personal Ubiquitous Comput.*, 6(5-6):430–442, 2002.

[9] S. Chetan, J. Al-Muhtadi, R. Campbell, and M.D. Mickunas. Mobile gaia: a middleware for ad-hoc pervasive computing. In *Second IEEE Consumer Communications and Networking Conf. (CCNC)*, pages 223–228, 2005.

[10] C. A. Costa, A. C. Yamin, and C. F. R. Geyer. Toward a general software infrastructure for ubiquitous computing. *IEEE Pervasive Computing*, 7(1):64–73, 2008.

[11] A. Drozd, S. Benford, N. Tandavanitj, M. Wright, and A. Chamberlain. Hitchers: Designing for cellular positioning. In Paul Dourish and Adrian Friday, editors, *UbiComp 2006: Ubiquitous Computing*, volume 4206 of *Lecture Notes in Computer Science*, pages 279–296. Springer Berlin Heidelberg, 2006.

[12] O. Fouial, K. A. Fadel, and I. Demeure. Adaptive service provision in mobile computing environments. In *4th IFIP Int. Conf. on Mobile and Wireless Communications Networks (IEEE MWCN)*, pages 9–11, 2002.

[13] M. Friedewald and O. Raabe. Ubiquitous computing: An overview of technology impacts. *Telematics and Informatics*, 28(2):55 – 65, 2011.

[14] P.C. Garrido, G.M. Miraz, I.L. Ruiz, and M.A. Gomez-Nieto. Near field communication in the development of ubiquitous games. In *Int. Conf. for Internet Technology and Secured Transactions (ICITST)*, pages 1–7, 2010.

[15] F. Gatt. *Turn Your Android Phone Or Tablet Into a Multimedia Hub*. MetaPlume Corporation, 2011. ISBN 9780987165275.

[16] C. Greenhalgh, S. Benford, A. Drozd, M. Flintham, A. Hampshire, L. Oppermann, K. Smith, and Christoph Tycowicz. Addressing mobile phone diversity in ubicomp experience development. In *UbiComp 2007: Ubiquitous Computing*, volume 4717 of *Lecture Notes in Computer Science*, pages 447–464. Springer Berlin Heidelberg, 2007.

[17] ITU-T. Multipoint application sharing. Recommendation T.128, Int. Telecommunication Union, Geneva, 2008.

[18] Michael Jeronimo and Jack Weast. *UPnP Design by Example: A Software Developer's Guide to Universal Plug and Play*. Intel Press, 2003. ISBN 0971786119.

[19] K. Koskinen and R. Suomela. Rapid prototyping of context-aware games. In *2nd IET Int. Conf. on Intelligent Environments (IE 06)*, volume 1, pages 135–142, 2006.

[20] U.P. Kulkarni, J. V. Vadavi, S.M. Joshi, and A. R. Yardi. Ubiquitous object categorization and identity. In *Computational Intelligence for Modeling, Control, and Automation 2006 and Int. Conf. on Intelligent Agents, Web Technologies, and Internet Commerce*, pages 81–86, 2006.

[21] F. L. Law, Z.M. Kasirun, and C. K. Gan. Gamification towards sustainable mobile application. In *5th Malaysian Conf. on Software Engineering (MySEC)*, pages 349 – 353, 2011.

[22] J. Madhusudanan, V. P. Venkatesan, V. Indumathy, A. Kalaiselvi, C. Ramachandran, and K. Preathee. Article: Categorization and grouping of devices in generic pervasive applications. *Int. Journal of Computer Applications*, 45(3):33–37, 2012. Published by Foundation of Computer Science, New York, USA.

[23] J. E. Mcgonigal. *This might be a game: ubiquitous play and performance at the turn of the twenty-first century*. PhD thesis, Berkeley, CA, USA, 2006.

[24] L. Mottola, A. L. Murphy, and G. P. Picco. Pervasive games in a mote-enabled virtual world using tuple space middleware. In *Proc. of 5th ACM SIGCOMM workshop on Network and system support for games (NetGames 06)*, NetGames '06, New York, NY, USA, 2006. ACM.

[25] A. Paiva, G. Andersson, K. Höök, D. Mourão, M. Costa, and C. Martinho. Sentoy in fantasya: Designing an affective sympathetic interface to a computer game. *Personal Ubiquitous Comput.*, 6(5-6):378–389, 2002. ISSN 1617-4909.

[26] S. Pope. The future of head-up display technology. *Aviation Int. News*, 38(1):60–63, 2006. Published by Convention News Company, Incorporated.

[27] B. Ribeiro, J. Gondim, R. Jacobi, and C. Castanho. Private communication. 2009.

[28] T. Richardson, Q. Stafford-Fraser, K.R. Wood, and A. Hopper. Virtual network computing. *Internet Computing, IEEE*, 2(1):33–38, 1998. ISSN 1089-7801. doi: 10.1109/4236.656066.

[29] M. Weiser. The computer for the 21st century. *Scientific American*, (265):94–104, 1991.

[30] M. Weiser. The world is not a desktop. *interactions*, 1 (1):7–8, 1994.

[31] M. Weiser and J. S. Brown. Designing calm technology. Technical Report 1, 1996.

[32] B. Wietrzyk and M. Radenkovic. Enabling rapid and cost-effective creation of massive pervasive games in very unstable environments. In *Fourth Annual Conf. on Wireless-on-Demand Network Systems and Services (WONS ?07)*, pages 146–153, 2007.