RemoteU¡: A Middleware for user interaction mobile applications

Miguel Almeida Carvalho

INESC-ID Lisboa – Instituto Superior Técnico – Universidade de Lisboa, Portugal
miguel.d.carvalho@tecnico.ulisboa.pt

Abstract. In our daily lives we assist to an exponential growth of mobile and fixed devices that surround us, many of them having limited resources, and some of them not even providing an interface screen. Nonetheless the connectivity and interoperability between these devices is still low, and the user interaction is still limited. This paper presents RemoteU¡, a middleware that allows the interaction of devices with users, using the ubiquitous smartphone. RemoteU¡ offers simple but expressive programming mechanisms, and provides efficient implementation and communication.

Key words: Middleware, Mobile applications, Ambient Intelligence, Internet of Things, User Interaction

1 Introduction

In the past few years there has been a popularization of mobile devices, especially smartphones, and nowadays many accessories are being developed to interact with them. Studies have also shown that people have great interest in the use of mobile devices as a replacement for the traditional remote controls, mouse and keyboard input mechanisms [1]. This has motivated developers to design applications to improve the integration and communication between mobile devices and numerous others. Ambient intelligence is an area where this type of applications are relevant, since one of the main goals is that the technology disappears and only the user interface is perceived by users. The Internet of Things aims to interconnect any kind of objects but requires user interaction so that users can access and control them.

Developing applications that interact with users is demanding, even when developing centralized applications (display and logic in the same device).

This document presents RemoteU¡, a middleware that allows the implementation of complex interaction between devices and users, resorting to local connectivity and mobile devices (such as smartphones or tablets), relieving the programmer from the communication and interaction programming burden. Since the data encoding is optimized for speed, the devices can be connected by any network link (ZigBee, Bluetooth, WiFi, or USB), independently of its bandwidth.
1.1 Motivation

Researchers are exploring the creation of new networks of pervasive displays that go beyond the traditional broadcast model and support innovative new applications and highly personalized content, tailored to nearby viewers [2, 1, 3]. It is proved that there are many factors in the impact that these displays can have in the interaction with spaces and people [4, 5]. This brings many challenges, privacy issues being one of the most relevant. The use of personal devices such as smartphones, can be an efficient way of overcoming these concerns [6].

Another featured area of research is ambient intelligence, for example, home automation/domotics, that helps people monitor and control the mechanical, electrical and electronic systems. This may be done through the use of proprietary controllers or with personal and mobile devices.

Wearable computing is another featured topic, not only due to the popularity of devices such as smartwatches or wrist bands. The advances on lightweight, small-size, and ultra-low-power monitoring sensors that can be embedded in clothes, shoes, and others, allow the creation of body area networks (BANs) [7]. These requires user interaction, usually resorting to users mobile devices.

1.2 Document Structure

The next section presents systems related to the objectives of RemoteUj. Section 3 presents the system requirements, architecture and implementation details. Section 4 presents the system evaluation, and this document ends with the conclusions.

2 Related Work

In current mobile environments the presentation of information on remote devices resort to remote screen protocols, distributed UI infrastructures or to web based architectures. These systems allow the interaction between the user and the scattered devices, but require the existence of external infrastructures, complex programming or demand high computational resources.

2.1 Mobile UI Systems

Several mobile UI systems (such as MoCHA [8]) are web-based systems and allow rich interactions between pervasive displays and users with mobile devices, in which both interact through a web page available at a certain address. MoCHA is divided in two mainly components, a distributed architecture based on modern web technologies and cloud computing, that provide rich interactions between mobile devices and pervasive screens, and a mechanism for practical and secure binding based on their proximity. The proposed system assumes that there is an Internet connection for the devices to communicate, and only a prototype is presented, not providing an API for third party programmers.
The iCapture [9] project is a web application designed to facilitate spontaneous user-interaction with situated display environments through the use of camera enabled mobile phones. The prototype application consists in a large display that is responsible for exhibiting news headlines with a corresponding code, which users may capture to read it on the phone. Although it is an interesting example scenario the interactions are rather simple, and the system is limited to public displays interactions, not being able to implement with headless devices.

PuReWidgets [10] is a toolkit for developing interactive public display applications, composed by a widget library and web service that handles interaction events, providing programmers with high-level interaction abstractions. It was developed using a cloud platform that requires the existence of an internet connection.

A web-based toolkit for remote direct manipulation interaction with public displays via smartphones [11] is being developed to help programmers in the creation of applications with less effort. The toolkit provides controls such as joystick, text input, and multi-touch cursor events. The work is still in development, and it is a browser-based architecture.

The DireWolf [12] is a framework for distributed web applications based on widgets. It helps managing a set of devices providing easy distribution of widgets among them, and guarantees the preservation of the state. This work is more targeted for designing collaborative and multi-user applications.

2.2 Remote Screen

The concept of remote screen isn’t new in computation as many systems based on this idea started being developed long time ago.

X Window System (X11) [13] is an interesting system also implementing a client-server approach. X uses a client-server model where the users terminal is the server (where UI is rendered) and the applications are the clients, and provides a framework for the development and execution of GUI applications environment. It doesn’t implement the system user interface, so each application is responsible for the UI definition, appearance and behavior. Although X hides the distribution of UI’s into remote displays, the programmer should take into account the events and program all the necessary handlers, requiring high programming skills and resulting in high complexity programs.

Virtual Network Computing (VNC) [14] is an application originated in the late nineties that allows sharing a screen remotely. It is particularly useful for remote technical support and accessing files on remote computers, like accessing a work computer from home. One of the most interesting features of VNC is that is platform-independent and the code is open source, thus has generated a vast number of implementations. Although platform independent, during its operation, the complete screen (encoded as a bitmap) is sent to the client computer, thus incurring in high communication overhead. Furthermore the presented UI is independent of the client UI framework, offering different user experience. Currently other remote screen systems exist, such as TeamViewer or Microsoft Remote Display.
2.3 Discussion

Most of the mobile UI systems are implemented in a browser/cloud based architecture, and in most cases require an Internet connection so they aren’t able to operate in a closed environment, like for example with some phone accessories. Moreover many of the presented works are prototype architectures that don’t provide any API for third-party developers.

Regarding to X11 and VNC, both of these systems present many disadvantages when implementing in small devices, the most relevant being the fact that, as they aren’t designed for this purpose, they will consume too many resources. Also they add too much network overhead when transmitting the necessary information to generate interactions. The fact that they don’t provide a consistent UI to the user when dealing with different applications is another negative aspect.

All existing solutions require high computation resources in terms of network connectivity and devices processing power, thus limiting its wide deployment and usability.

3 RemoteU¡

This section presents the detailed architecture of the system, along with its implementation and programming API.

One important goal of RemoteU¡ is to be a generic middleware and usable in multiple situations, environments and applications, so that the duplication of implementations and effort are avoided.

Another objective is to provide the user with the presentation of the usual interactions. It also needs to hide the client-server architecture from the programmer, providing easy implementation. The middleware needs to be inter-operable, with respect to the device where the application will run, and also with the mobile devices.

3.1 Architecture

Figure 1 depicts the proposed solution architecture. Though the main device is characterized as being headless, it may still have an accessible user interface, such as a screen or speakers. This simple characterization emphasizes the fact that the middleware is designed to a wide variety of equipments and usable in the several domains. Both the headless and mobile devices are represented and divided by layers and a description of each of them is now presented.

The first layer represents the network, which consists in any kind of connection, such as Wi-Fi or Bluetooth, or even a wired connection such as USB. RemoteU¡ assumes that there is a communication socket previously established between devices. Thus RemoteU¡ can be integrated with multiple discovery mechanisms, though how devices recognize each other and establish the communication process is out of the scope of this work.
RemoteUı: A Middleware for user interaction mobile applications

Fig. 1. System Architecture

The second layer is the middleware system, and it’s represented both in the headless device, where the main application runs, and in the mobile device by the UI application. The system is responsible for providing interaction mechanisms between them and this is made through the generation and management of defined messages.

The programming API represents the third level of the architecture, as it’s one of the distinctive features of the RemoteUı system. It provides a set of methods that can be invoked in the application that will automatically generate the widgets that will be displayed in the user mobile device. At present it is available in Java and C++ languages.

RemoteUı is communication agnostic. It operates over any kind of socket connection that is available. Protocol Buffers\(^1\) are used to serialize messages exchanged between the client and server. When compared with other widely available alternatives (such as XML or JSON), Protocol Buffers is more efficient when serializing data (with respect to processing time and serialization results size)\(^2\) and is also widely available.

### 3.2 Interaction models

RemoteUı allows the implementation of various types of interaction. Atomic interaction is the most useful since it implements request-response transactions, much like an fgets call in command line interfaces. The main difference is that the interaction process is programmed on the client and presented in the mobile device and allows the input of complex data. When the user finalizes the interaction the results are sent back to the application.

It is also possible to implement an asynchronous behaviour, which is the case when the programmer wants to receive results whenever the user interacts with a widget. In this mode, a single result is propagated to the client each time a change occurs.

Complementary to the previous modes, it is also possible to update the fields on the mobile device while the user interacts. Images or text fields can be dynamically updated in an active presented screen. This allows a more dynamic interaction with the user.

---

\(^1\) developers.google.com/protocol-buffers/

\(^2\) https://github.com/eishay/jvm-serializers/wiki
RemoteWidgets rw;
    rw.addTextBox("Name:");
vector<string> params;
    params.push_back("Age:");
    params.push_back("0"); //min
    params.push_back("100"); //max
    params.push_back("20"); //initial
    rw.addSlider(params);
    rw.addButton("Submit");
    write(sock, rw.serialize(), rw.len());

Fig. 2. getInput() function pseudo-code and corresponding mobile screen

3.3 Programming API

In order to define and create the remote screens and retrieve the user input, the programmer uses a set of classes (Java or C++). These classes encapsulate the remote screen and allow the definition of the widgets that will be presented.

Figure 2.a) presents the C++ code needed to generate a screen requesting the user to input his name (textbox) and age (slider), and could be used in a welcome screen of an application. Figure 2.b) shows the output result.

The creation widget methods (e.g addTextBox) receive as parameter the name/label of that widget. In widgets that require more complex configuration (min, max and initial value) the creation method receives a vector with those values. Due to space limitation these configurations are not detailed in this document. After the instantiation and configuration of the rw object the programmer should send it through the selected communication channel (the write function). The widgets will be created in the same order as defined in the code.

In order to retrieve the user input, the program should read from the communication channel a representation of the user input. The response objects (res) contain a key-value pair list: the keys are the widgets identifiers, while the value correspond to the user input.

3.4 Implementation

Figure 3 presents how the RemoteUj is structured and implemented. RemoteUj provides some library files to include in the client application so that the programmer can generate the messages that contain the description of the widgets. This is done creating instances of the class RemoteWidgets, each representing a different interface screen, and then serialized with protocol buffers. There is another class that allows receiving the objects regarding the responses.

In order to parse the messages in the mobile devices, the middleware provides the communication library to include in an Android application. Once again there are classes that allow to instantiate objects in order to decode the messages. The rendering of the widgets is made with a WebView that loads a web page containing a Javascript library, which handles the requests and presents the
widgets. Also there is a Javascript interface class that allows the value results to be returned from the widgets to the mobile application.

*jQuery Mobile*[^3] is used for the widgets implementation, offering wide compatibility and availability and allowing the presentation using a consistent design and layout. It is particularly useful to automatically adjust the widgets to different screen sizes. The use of this high-level library provides independence of the system where the UI is presented, thus avoiding problems regarded with proprietary OS widgets, which may have different kinds of configurations.

### 4 Evaluation

At the present moment, RemoteU¡ is being used as the UI framework for a European Funded project (PCAS[^4]) aiming at reducing data exposure when using mobile devices. PCAS will provide a secure mobile data storage and transfer environment, using biometric technologies to ensure the identification and authentication of users. The secure storage and biometric authentication is done inside the SPD (Secure Personal Device), which is a smartphone accessory that mimics a protective sleeve but will store information, authenticate the user and mediate secure communication with service providers.

RemoteU¡ can also be implemented in other context systems, like for example with public displays or IoT (smart refrigerator, wearables, etc.). The work has been presented and published as a poster in Mobicom 2015 conference.[^15]

#### 4.1 Communication Performance

In order to evaluate the performance of the system some tests were performed and the messages payloads were measured with different screen configuration (variable number of widgets and complexity). Table 1 presents the payload of the requests (creation of the screens) and response messages (after user clicks submit).

---

[^3]: jquerymobile.com/
[^4]: https://www.pcas-project.eu
Table 1. Payload on different types of messages

<table>
<thead>
<tr>
<th></th>
<th>Request size (byte)</th>
<th>Response size (byte)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Single widget</td>
<td>27 (average)</td>
<td>58 (average)</td>
</tr>
<tr>
<td>2 widgets</td>
<td>29 (min) / 138 (max)</td>
<td>77 (min) / 437 (max)</td>
</tr>
<tr>
<td>3 widgets</td>
<td>48 (min) / 201 (max)</td>
<td>91 (min) / 646 (max)</td>
</tr>
<tr>
<td>10 widgets</td>
<td>196 (min) / 642 (max)</td>
<td>443 (min) / 2105 (max)</td>
</tr>
</tbody>
</table>

The results show that there is a fixed size message payload (Single Widget test), and that it is minimal, only about 30 bytes.

The Request size varies with the number and type of the widgets created. To simple widgets the request size does not increase much, ranging from 27 bytes to 196 bytes. If more complex widgets are used (requiring more labels, such as checkboxes), the size increases with the number of widgets, number of options and length of the option labels. For example, a checkbox with three options (Red, Green, Blue) represents a larger message payload than a date: 22 bytes against 4 bytes.

The same happens with the response object, in some widgets it is fixed (the date is always 22 bytes), in other it varies with the user input. The response payload for a text box varies with the length of the input text, for example with the input Frideric Handel the cost is 27 bytes. The worst result (2 kByte) corresponds to a screen with 10 widgets each one containing about 200 characters.

The results show that, in all cases, the payloads are minimal, leading to efficient communication independently of the bandwidth of the network link.

Comparison RemoteU¡ can also be compared with two existent remote screen technologies and techniques: VNC and X11. In this test two different screens were implemented. The first is composed by a text box, a slider and a submit button. The second presents the same widgets, plus a check box (with three options) and the data widget (to force scrolling).

To perform the first test scenario a VNC like application (TeamViewer\textsuperscript{5}) was installed on both devices, a smartphone and a desktop PC. A simple android application that shows the same set of widgets as presented in the RemoteU¡ application was developed and presented on the remote device using TeamViewer. One of the examples requires window scrolling, having a significantly impact in the traffic flow between the VNC client and server.

The setup for the X11 case was developed using TkInter\textsuperscript{6}. The widgets used were the same, with the exception of the date widget, that in this case was alternatively represented by a text box.

The interaction in all scenarios was very simple: entering the first and last name, selecting the age on the slider, selecting two options on the checkbox and typing the data on the textbox. The results of the remote interactions are

\textsuperscript{5} www.teamviewer.com
\textsuperscript{6} wiki.python.org/moin/TkInter
presented in table 2. Each test only took a few seconds and the traffic was monitored and measured using Wireshark⁷.

<table>
<thead>
<tr>
<th></th>
<th>RemoteUI (syn)</th>
<th>RemoteUI (asynch)</th>
<th>VNC</th>
<th>X11</th>
</tr>
</thead>
<tbody>
<tr>
<td>2 Widgets</td>
<td>0.5 KByte</td>
<td>5.5 KByte</td>
<td>88 KByte</td>
<td>1000 KByte</td>
</tr>
<tr>
<td>4 Widgets</td>
<td>0.6 KByte</td>
<td>7 KByte</td>
<td>1738 KByte</td>
<td>1800 KByte</td>
</tr>
</tbody>
</table>

Table 2. Network traffic comparison with VNC and X11

As shown, there is a significant difference in the messages payloads between RemoteUI and the other systems. One of the reasons for this difference is the efficient way the screen is encoded. In RemoteUI, a representation of the screen takes few bytes, in X11 the objects creation is less efficient and in VNC a bitmap representation of the screen is transferred.

In RemoteUI, using synchronous mode during the filling of the forms there is no data transfer: only when the application transfers the remote screen and the user submits the form there is network traffic. In asynchronous mode, whenever the user changes the value of a widget communication happens. On all other options (VNC and X11), whenever a relevant event happens, data is transferred: VNC transfers data when the mouse cursor position changes and when screens are updated (user writes on a text box); X11 also transmits the mouse events, along with all keystrokes.

All tests demonstrate that RemoteUI has much better performance in terms of traffic generated on the network, when dealing with this kind of interaction mechanisms. None of the generated traffic depends on the interaction duration, but only on the data produced by the user.

5 Conclusion

The RemoteUI in conjunction with the programming languages control structures allows the efficient definition of complex interactions with the users. Besides the simple declaration of screens to present, the program logic allows the definition of the interactions without resorting to event programming, complex resource files or complex communication patterns.

This allows the use of this interaction on had-hoc, encounter networks and deployment on low resources devices. Its generality and communication channel agnosticism will allow the use of the same middleware in the interactions with several resources (smartphone accessories, embedded device, IoT, public advertising) by a common infrastructure. Furthermore its efficient encoding of screen makes it apt to any network link.

The communication performance is high compared with other existing approaches, due to the reduce message payload (on the order of bytes).

⁷ www.wireshark.org/
References