Unified RemoteUI for Mobile Environments
Miguel Almeida Carvalho  João Nuno Silva
INESC-ID Lisboa – Instituto Superior Técnico Portugal – Universidade de Lisboa, Portugal
miguel.d.carvalho@tecnico.ulisboa.pt  joao.n.silva@inesc-id.pt

ABSTRACT
In our daily lives we assist to an exponential growth of mobile and fixed devices that surround us, though many of them having limited resources, and not even providing an interface screen. In this paper, we present remoteUI, a middleware that allows the interaction of those devices with users, resorting to simple but expressive programming mechanisms, and providing efficient implementation and communication.

Categories and Subject Descriptors
C.2.4 [Computer-Communication Networks]: Distributed Systems – Distributed applications
H.5.2 [Information Interfaces and Presentation]: User Interfaces – Graphical user interfaces (GUI)

Keywords
Mobile user interface, Mobile computing, IoT, Middleware

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, expanding their capabilities. This has motivated developers to design applications that can improve the communication between them.

Ambient intelligence is another 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. Also Internet of Things aims to interconnect any kind of objects and requires user interaction so that users can access and control them.

Developing applications that interact with users is demanding. In many cases programmers need to develop a whole server-client architecture application, and it’s from greater importance to provide them with tools that can ease this process. Good examples of these devices are “Chromecast” from “Google” and the smartwatches, devices that interact with or are controlled by mobile phones, but require the development of both client and server applications.

In this work we present a middleware that allows complex interaction between devices (with limited resources and no screen) and users, resorting to local connectivity and mobile devices (such as smartphones or tablets), also relieving the programmer from the communication and interaction programming burden.

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. In order to reduce the programming of explicit client/server interaction between the devices and the screens, several approaches exist.

Several mobile UI systems (such as MoCHA [5]) are web-based systems and allow for rich interactions between pervasive displays and users with mobile devices, in which both interact through a web page available at a certain address. MoCHA assumes that there is an Internet connection, and only a prototype is presented, not providing any kind of API.

PuReWidgets [2] 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 implies the dependency of Internet connection.

A web-based toolkit for remote direct manipulation interaction with public displays via smartphones [3] 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 [4] is a framework for distributed web applications based on widgets. It helps managing of a set of devices providing easy distribution of widgets among them, and guaranteed the preservation of the state. This work is more targeted for designing collaborative and multi-user applications.

The concept of remote screen isn’t new in computation as many systems based on this idea started being developed long time ago, though we only mention the most relevant. One of the most interesting systems with the same client-server approach we present is the X Window System (X11) [1]. X11 uses a client-server model where the users terminal is the server and the applications the clients. It provides a framework for a GUI environment, though it doesn’t implement the user interface, as each application is responsible for that, resulting in many different types.

Virtual Network Computing (VNC) [6] 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 is has generated a vast number of implementations.

2.1 Current limitations
Most of the mobile UI systems are implemented in a browser-based architecture, and in most cases require Internet access so they aren’t able to operate in a closed environment. Moreover
many of them 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 the same UI to the user when dealing with different applications is another negative aspect.

3. ARCHITECTURE
The main requirements of our system are: i) it should be generalizable to avoid the duplication of implementations; ii) provide the presentation of the usual interactions so that the interface matches the appearance of the most popular applications; iii) hide the client/server architecture from the programmer making it possible to develop without worrying with the underlying interaction methods; and iv) be interoperable with the majority of the available devices.

3.1 Solution
Figure 1 depicts the proposed solution architecture. Both the headless and mobile devices are represented and divided by layers.

The first layer (L1) represents the network, which consists in any kind of connection, such as Wi-Fi or Bluetooth, or even a wired connection such as USB. We assume that there is a communication socket established where the middleware application operates. Thus we don’t implement discovery mechanisms, how devices recognize each other and establish the communication process is out of the scope of this work.

The second layer (L2) is the core of the middleware system, and it’s represented both in the headless device, where the main application runs, and in the mobile device by the remoteU API library. The system will be responsible for providing interaction mechanisms between them and this will be made through the generation and management of defined messages.

The programming API represents the third level (L3) of the architecture, as it’s one of the distinctive features of the remoteU 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.

![Figure 1 - System Architecture.](image)

RemoteU is communication agnostic. It operates over any kind of socket connection that is available. In order to exchange the messages between the client and server, instead of the traditional languages such as XML or JSON we use “Protocol Buffers”, which are an efficient mechanism for serializing structured data, and are simpler and faster than the others. Resources and service discovery mechanisms are also independent of our solution. The application logic is programmed with the regular language structures, and the user interaction is done through the remoteU API.

3.2 Widgets
Widgets are one of the most useful resources programmers have nowadays for implementing interaction features in their applications, as they represent interface components with clear functionality dedicated to small tasks. Widgets are reusable for multiple purposes in different applications so their use in the design of mobile applications is very common. It reduces the overhead in the development and also provides users with a common language for interacting with the applications.

RemoteU allows the creation of screens containing instances of the more common classes of widgets: Button; Textbox; Checkbox; Radio Button; Slider; Range Slider; Date Picker. It is also possible to display text or images on the screen, or send files to open in the appropriate application. "jQuery Mobile" is used for the widgets implementation, offering wide compatibility and availability and allowing the presentation using a consistent design and layout.

3.3 Interaction
RemoteU allows atomic interaction implementing request-response transactions, much like an RPC call in “command line interfaces”. The main difference is that the interaction process is programmed on the client and presented in the mobile device. It is also possible to implement an asynchronous behavior, so that the system may send results whenever there are changes on the widgets input. It is also possible to update fields like images or text fields of an already presented screen. The interaction flow is defined by the structure of the source code (not event driven) and is programmed using the developed API that provides methods to generate each type of widget mention in section 3.3.

As use case scenario, we present an example in the context of IoT, in home automation domain. Imagine you want to use the mobile phone to control the lighting system of a room. Instead of the programmer having to implement the interfaces according to each light device, that in many cases differs form each other, the devices could generate their own interactions, and the programmer just needed to aggregate and route the requests and responses. Even in the present reality remoteU can help the developer providing interaction abstractions and allowing generating an interface screen with just a few lines of code. Figure 2 shows the necessary code (in C++) to generate the interface screen example presented in Figure 3. It allows changing the present configuration of two sets of light of a room, the ambient and task lighting.

```c++
//Create RemoteWidgets object:
RemoteWidgets rw;
//Add Text field:
rw.addText("Id-1", "The system automatically adjusted the ambient and task lights by default. You may change it now.");
//Add button to turn off ambient lights:
rw.addButton("Turn off ambient lights");
//Add slider to adjust lamp lights:
vector<string> optionslider;
optionslider.push_back("Choose intensity of lamp lights:");
//title optionslider.push_back("[""); //default value
optionslider.push_back("]"); //min, value
optionslider.push_back("-]"); //max, value
optionslider.push_back("="); //step value
rw.addSlider(optionslider);
//Add button to close screen:
rw.addButton("Close");
//Get object to send to socket:
RemoteScreen rs = rw.GetRemote();
```

![Figure 2 – Example of code to generate an interface screen.](image)
Another good example is in the context of public displays, where it’s very common to have mapping applications available for users to search points of interest and get directions to a certain destination. Most of the times this requires having a display with touch screen technology so users can interact and input some query. Besides being more expensive than normal ones, current solutions have other disadvantages: correct physical positioning of the screen and difficulty in sharing with multiple users. Using the mobile phone as an input mechanism solves these drawbacks: remoteU\textsubscript{i} allows the development of complex user interaction with the screens.

RemoteU\textsubscript{i} can easily be integrated with already developed systems, not requiring significant changes in an existing application to extend features like presenting UI’s in a mobile device recurring to our middleware.

4. EVALUATION

At the present moment, remoteU\textsubscript{i} is being used as the UI framework for a mobile phone accessory (in the context of the PCAS\textsuperscript{1} project) that securely stores user files and controls its access using biometrics. It connects via USB although it could use Bluetooth or Wi-Fi, allowing easily generating necessary interaction screens that are displayed to the users.

Public displays are gaining popularity, and most of them are used only for advertising and information, though users are feeling more interested in the possibility of using them as a resource to their needs. This requires the use of simple interactions, like sending text or selecting between options through their smartphones. In the context of “IoT”, remoteU\textsubscript{i} will allow the interaction with a series of objects on the everyday life, for configuration, management or normal operation, even with power or connection constrained devices.

4.1 Performance metrics

In order to evaluate the efficiency of the system we now present, in table 1, some performance metrics based on the payload of messages to generate interaction screens from the application to the mobile device, and the respective response.

The payloads are minimal, leading to efficient communication independently of the bandwidth and latency of the selected network link.

5. CONCLUSION

The remoteU\textsubscript{i} 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 complex 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 architecture, implementation and API presents satisfies the previously presented requirements: i) allows the reuse of the middleware for multiple applications; ii) implements most available widgets and is usable in most classes of applications; iii) relieves programmers from client-server programming; and iv) is interoperable with the majority of the devices, and communication links.

6. ACKNOWLEDGMENTS

This work was supported by national funds through Fundação para a Ciência e a Tecnologia (FCT) with reference UID/CEC/50021/2013, and by the European Commission through the contract no. 610713 (PCAS).

7. REFERENCES


---

\textsuperscript{1} www.pcas-project.eu