# Contents

[Contents 1](#_Toc375927199)

[List of pictures 2](#_Toc375927200)

[List of tables 3](#_Toc375927201)

[1 Introduction 1](#_Toc375927202)

[1.1 Android platform briefly 1](#_Toc375927203)

[1.2 Widgets 1](#_Toc375927204)

[1.3 History 1](#_Toc375927205)

[1.4 Extensibility 2](#_Toc375927206)

[2 Aims of the work 3](#_Toc375927207)

[3 Existing solutions 4](#_Toc375927208)

[4 Design and architecture 5](#_Toc375927209)

[5 Implementation 6](#_Toc375927210)

[6 Testing 7](#_Toc375927211)

[7 Conclusion 8](#_Toc375927212)

[8 Bibliography 9](#_Toc375927213)

[A. User screeners 10](#_Toc375927214)

[B. CD/DVD contents 11](#_Toc375927215)

# List of pictures

# List of tables

# Naming conventions

Widgets, apps, applications, components, GUI widget, layout, ppi zkratka

# Introduction

## Android platform briefly

Nowadays one of the most popular operating systems targeted on mobile devices is undoubtedly Android. As of second quarter 2013, almost 80 percent of newly sold devices use Android as its platform. The greatest share of all manufacturers belongs to Samsung.

This operating system cannot be compared to previous versions of desktop platforms, because the one of the key advantages is missing. The touch input. For majority of users, especially the ones who browse the internet a lot, the real breakthrough came with all the touches. In history not so distant, the only possibility to comfortably search the web was to use standard desktop computing inputs – mouse and a keyboard. But what about mobile users? They tend to operate in much more compact environment with limited access to all the gadgets connected with desktop computing. As a result, all users, hungry for browsing the web were sentenced to use arrows or in better cases, some kind of trackball. The real power of fast browsing basically came with three things – wifi/3G internet connection, high definition screens and the ability to input via touches. Its important to stress, that the first two mentioned would not significantly improve the browsing experience itself. But with the connection of the touch screen everything took the whole new direction. Let’s have an example. Filling the form on e-shop – a classic task on desktop computer. What a nightmare on pre-historical mobile device. User was forced to skim through all these visible elements to finally acquire the targeted one. The whole operation was matter of seconds. Compared to touch input, this was very poor performance. Average user, when given low-end Android device, is able to target the form field in a matter of tens or hundreds of milliseconds. And when it comes to filling some input, the screen of the mobile gadget is so large, that a full version of qwerty keyboard fits in.

One of the key values, that caused the major breakthrough of Android is its Linux kernel ,which is hidden under the shiny hood of all the functions, that are presented to the final user. This “glue” is used to offer an interface to all the internal hardware, such as gyroscope, accelerometer and proximity sensors, that are brought to the end user to enrich the interaction with either default or third party applications.

Open source nature of Android, device manufacturers, professional and enthusiast developers are free to update the code under Apache licence.

## Widgets

Synonym for application, or app. Widget is not connected only with Android platform, it can be easily found on desktop PC, as well as on Windows Phone, or iOS. When compared to common software packages, or even large software suites, widget is a lightweight application, that does not necessarily consumes all users attention. Easily imagined example would be widget, that shows the current CPU/memory load, weather forecast app, or calendar.

Among the coding society a wide variety of terms related to widgets is used. Content of this work will be further discussed, but it is targeted on GUI widgets. These elements are previously known from desktop computing under the name components. All the checkboxes, radio buttons, faders and even grouped elements like dialog boxes - these are all GUI widgets.

Android comes with rich set of widgets, that are professionally looking, ready to be used in an out-of-the-box manner. Adequate event-driven mechanism is included, so that these widgets respond accordingly to user actions, like touches, clicks, drags, etc.

Android developers have basically two possibilities to include existing widgets into their apps.

* Specify the layout with all the widgets in XML document (preffered way)
  + Pros and cons,
  + Easily changed without the explicit need to recompile source
* Set the layout programmatically
  + Pros and cons

But what about the situation, when none of the supported widgets fits the developer’s needs and he wants to design a custom one? Of course, there is the possibility to create own components, that might extend the basic functionality of existing ones, or to offer a whole new approach. The question of setting up the initial layout stays almost the same. While there is of course need to specify custom functionality programmatically, once the widget is set up, it can be added to existing layout via XML, or by code change.

Example of programmatic layout setup and XML-based

## History

The original developer of today most popular platform - Android, Inc. was founded in 2003. Their primary intention was to develop a smart operating system targeted on digital cameras. When studying the situation on the market, the original targeted device changed and all the effort was now directed towards newly originated smartphones with clear vision to rival Symbian and Windows Mobile. The idea was clear, keep the newly designed operating system as a secret as long as possible. In 2005, the whole company was acquired by Google, with all key employees and the founders staying.

With this acquisition, Google secretly planned to enter the growing market of smart phones. In spite of this move, certain hardware components were lined up and several software partners were asked to cooperate.

While the background development went on, Apple presented their first version of touchscreen-based phone – the iPhone. At that time, Google had a prototype codenamed “Sooner”. This prototype was similar to BlackBerry phones, because it has no touchscreen and only the physical QWERTY keyboard was present. Immediate actions were taken and the whole operation system was reengineered. Combination of traits, that came from previous designs with overall experience marked the clear goal – prepare competitive rival for iPhone.

On November 5, 2007, the Open Handset Alliance (OHA) was publicly presented. OHA incorporates technology companies, wireless carriers, and device and chip manufacturers in common intention to develop open standards for mobile devices. As a result of all previous steps, first Android based smart phone was released – HTC Dream on October 22, 2008. Critical reviews stated that the phone itself looks both futuristic and retro, but it is not an “iPhone killer.” On the other hand Android as an operating system was received quite well and a major commercial success was yet to come.

Android corresponds to real world actions, like touching, sliding, pinching, reverse pinching, etc. Direct manipulation in general.

Other visual stimuli are presented through the ability to receive events through internal hardware devices, like gyroscope, accelerometer, proximity sensors, GPS module and many others.

## Extensibility

Android is a very popular platform for developers. As of April-May 2013, 71% developers targeted their applications on Android.

Android is open source and build on Linux kernel. Source codes are released by Google by Apache License.

October 2012 almost 700000 apps were available to be downloaded via Google Play and the total number of downloads from Google Play reaches 25 billion.

As of May 2013 the total number of app download reaches 48 billion.

September 3, 2013: 1 billion Android devices has been activated so far.

Despite primary targeted on smartphones and tablets, Android quickly attacked the market with smart televisions, game consoles digital cameras and other electronics.

Easy to start bundle, free to use even for commercial development, mention more than ADT bundle, but also the IDE based on IntelliJ IDEA (compare this one to Eclipse, user satisfactory, usage)

What is necessary to perform to write custom apps

Bridge between hardware devices(gyro, sensors, …) easy used through API

Developers today are given the opportunity to keep using Eclipse IDE even for Android development. The only thing needed is Android Developer Tools (ADT) plugin that gives them useful tools when building, debugging and testing their application. There is also possibility to install the whole ADT bundle package, which contains the Eclipse with ADT plugin pre-configured.

Some developers argue that Eclipse is heavyweight and slow. Quick and easy solution is to use Android Studio based on IntelliJ IDEA. Developers can benefit from IDEA’s valued features, like code navigation, refactoring, code analysis etc. Despite the fact, that final version of Android Studio has not yet been released; it is obvious, that it will be the preferred IDE in the future.

Novice developers may use the integrated UI builders that help them create custom layout in a drag and drop manner, while experienced ones will directly write into layout XML files and value the desired control over the UI elements.

Because the Android platform is “developer friendly,” Google has incorporated several functions that enable the developer to quickly run the application on target device via USB. Common problems, like memory leaks or performance critical areas can be revealed by using profiler directly on target device. In a situation, when no physical device is available, an embedded emulator can be used. When in developer mode, useful statistics, like CPU and memory usage, device’s network bandwidth usage or debugging information, can be displayed.

Every level of a newly created application needs its tools for testing. Android testing framework comes as an integral part of ADT plugin. Test suites are based on JUnit, which gives you the opportunity to use classic JUnit to test classes that does not use Android API. In case of more sophisticated classes that use Android API, JUnit extension should be used when testing. These extensions make it possible to create mock objects and methods that help the developer affect the lifecycle of the component.

## History of Android releases

### Android 1.5 Cupcake

First major Android release was introduced April 30, 2009. Noticeable features, which were presented in this release, were: newly integrated home screen widgets, folders visible on home screen, stereo Bluetooth support, ability to copy/paste within web browser, and video recorder with playback. Devices worth mentioning were: HTC Hero, Samsung Moment or Motorola Backflip.

### Android 1.6 Donut

Donut release presented the quick search box, interface for camera was updated, as well as Android Market, and gallery. Battery usage indicator and text-to-speech engine were added. Devices, which benefited from these new features were LG Optimus, Samsung Galaxy, or Sony Xperia X10.

### Android 2.0 – 2.1 Eclair

Google Maps Navigator in beta version was firstly presented with the Eclair release. Other features were: update browser based on Chrome engine, support for multiple accounts, or improved keyboard. High-end devices, which used Eclair were HTC Desire, Samsung Galaxy S and Motorola Droid.

### Android 2.2 – 2.3 Froyo

This release finally added support for Adobe flash, as well as multiple keyboard languages. Certain actions were taken and the whole system was optimized towards higher performance with slightly decreased computational demands. Noticable devices of this period are Nexus One or HTC Evo.

### Android 2.3 – 2.3.7 Gingerbread

As a minor release, Gingerbread offerred noticable UI refinements, support for NFC and faster text input. From all the devices from this period, Nexus S, HTC Sensation or Samsung Galaxy S II stand out the most.

### Android 3.0 – 3.2 Honeycomb

Honeycomb release meant a breakthrough mainly for table users, because holographic UI design was added, as well as action bar, improved multi-tasking or updated Android apps, which came in the bundle. Among others, Motorola Xoom, Eee Pad Transformer or Samsung Galaxy Tab were presented.

### Android 4.0 Ice Cream Sandwich

Release 4.0 meant improved multi-tasking, ability to unlock the screen by face detection, resizeable widgets and the option, that soft buttons can replace hard keys. This era presented Galaxy Nexus, HTC One or Sony Xperia T.

### Android 4.1 – 4.3 Jelly Bean

Jelly Bean came with streamlined UI, multi-user profiles for tablets, lock screen widgets and OpenGL ES 3.0. Devices worth mentioning are Samsung Galaxy S3 and S4, HTC One or Sony Xperia Z.

### 

# Aims of the work (motivation)

## Motivation

The initial motivation for this work was closely related to VR devices, specifically CAVE. When using CAVE environment, users can be immersed in both static and dynamic content. While static visualization allows diving into 3D projection enhanced with the feeling that observer is placed inside the scene, real power of immersive environment is revealed with dynamic content. The ability to walk or manipulate elements in virtual environment is the main motivation nowadays. But the real question is: how to navigate and manipulate within this virtual environment. One option that comes to a mind of gamer is to use some kind of game controller. Most of them are wireless, have plenty of buttons, switches, sticks or triggers. These controllers suit well the purpose of navigation. On the other hand, manipulating the elements of virtual scene is a bit tricky.

Let’s step aside and think of a controller that can detect the change of movement as the user turns around in the scene. Or another controller, that would contain various number of buttons, or other UI elements. What about putting all these features into one device? The answer is simple – Android. This operating system bridges all the hardware devices hidden under the hood of a smart phone, tablet, etc.

Devices running Android platform are already available on the market. By default, they do not support all the functionality, required by the immersive environment. But the extensibility of the Android platform allows the developer to design particular application that suits desired needs and offers required functionality.

But the situation may get more complicated. There may be two or more CAVE users at the same time – will they share the same controller? Or would it be possible for each user to have its own controller and all the controllers will be synchronized?

Let’s step even one step further and imagine the situation, when one cluster controls the CAVE. Cluster has multiple interactive visualizations installed. Each visualization comes with different UI elements that trigger different actions on cluster. There are several Android devices that are to be used as controllers. In this situation, different visualization would need different controller application to be run on Android device and in each case, synchronization would be needed. It is also important to bear in mind, that targeted devices probably won’t share the same screen dimensions and resolution. For users convenience will matching widgets have the same physical dimension across multiple devices with different resolution and screen dimensions.

## Aims of the work

From the early motivation, aims of this work slightly changed. Resulting system will no longer be primarily targeted on VR devices like CAVE but will server more general purposes. We still need to have a tool that allows us to remotely configure the UI on target Android device. This means not only to remotely setup the UI, but also to receive events, generated by the UI widgets and respond accordingly.

Finished tool should allow multiple devices to operate simultaneously. That means user actions on one device will not only cause an adequate action on server machine, but also propagates the change of UI to other connected devices.

Because one of the standard accessories of each Android device is Wifi adapter, wireless network will be used to transmit the communication necessary to send and receive user events and appropriate state change information.

Remotely configured user interface will be consisting of components that are well known from desktop applications, web, etc. Due to the required modularity it will be possible to create custom components.

## Study different ways of user interaction with VR devices at various levels of immersion.

## Study possibilities of interaction using Android-based mobile devices

## Propose and implement methods for general description and dynamic generation of graphical user interfaces on Android, based on the needs of VR applications. Make the system easily extensible.

## The result of the work will be a protocol description, an application for Android platform, client library and a simple application using the library demonstrating usability of the protocol.

## Conduct user testing and evaluate usability.

Document how to extend the system with new controls.

## Existing solutions

### UIProtocol

UIProtocol uses the concept of client and server. Its nature is derived from well-known technologies used on web, like HTML, CSS, JavaScript and AJAX. Unlike previously mentioned, this protocol is targeted on exchanging user interface related information between client and server. The idea is that UI is defined once and then send across wide variety of environments without the need to be even slightly modified.

The difference between similar approaches and UIProtocol is, that UIProtocol does not allow any application logic to be contained on client side. This forces potential developer to include the logic on server side only, which also means, that to logic has to be written only once.

As written before, UIProtocol is designed to be used by two entities - server and client. Server contains all the application logic and holds the necessary documents. These documents are requested by client on demand, thus no need to store any of them directly on client. Task of client is to display the user interface set by server and notify server about all the events that originated from user actions. It’s up to server to handle these user actions.

### Simple UI Protocol

QNX, well known for developing UNIX-based embedded systems, presented simple UI protocol, targeted primarily on automotive industry. This technology allows transferring informative widgets from smartphone to head-up display. It works both ways. Passengers on back seats can view the current temperature in the cabin and with functional widget they even have the ability to control the A/C settings. This protocol is built on bluetooth and unfortunately available only via OEM licence or Tier 1 suppliers.

### Apple accessory protocol

It is a simple protocol, which is used by a wide variety of Apple products to communicate with each other. On one hand, this system is capable of propagating metadata on more platforms, but on the other hand displayed user interface is not configurable, and the widgets have always the same functionality.

### Verse

Verse is a network protocol that allows real-time communication not only between graphic applications. Due to its capabilities, 3D artist, who uses Blender, can work on the interior of the building that is still under construction by architect using AutoCAD. All the changes made by architect are in real time propagated into the Blender and vice versa. The usage of Verse is not limited to a strict communication of graphics packages, but allows to be integrated in custom projects, like gaming industry, visualization etc.

# Design and architecture

## Design

Application design is based on layered architecture. De

### Client side

### Server side

Server application design is based on three main layers – network, xml and business. Each layer is allowed to communicate only with its direct neighbors.

#### Network layer

This layer establishes connection between server and all connected clients. Most crucial element is ServerNetworkAgent, which is running on particular IP address and port. Whenever new client tries to connect, new handler thread is created. This handler serves both to receive and accept client messages, while the connection lasts.

#### XML layer

XML layer serves as a converter between XML messages and message objects.

#### Business layer

Business layer takes the message object, identifies the class of concrete instance and based on that information

## Protocol

## Modularity

todo

## Layouts

Container, where all respective components are stored is called layout. There are two kinds of layouts in this work – state layout and metric layout. Both layouts serve the same idea – synchronization of client devices. As said before, connected clients may not share the same resolution or screen dimensions. But one of the requirements is that all widgets on target device would have comparable dimensions. Therefore state layout only encapsulates the information about the state of each component. There are no attributes regarding position or dimensions of individual widgets in state layout. Obviously, state layout is kept as singleton and guarantees that state of every single widget on each connected device will be the same.

Apart of that, metric layout exists in separate instances for every connected device. During the process of adding a new device, initial configuration layout (config.xml) is loaded with all metric information in millimeters. Knowing the ppi value for concrete device, value in millimeters is converted into pixels and saved into metric layout. During the lifetime of the application these values never change.

As a result, final layout that serves as the image for graphic context, which will be later rendered on client device, is composed of singleton state layout and specific metric layout.

## Message objects

Due to various common properties, message objects are composed into hierarchy. Each message is uniquely identified by messageId. Also each message is connected with exactly one so-called session via sessionId. Session represents the connection between particular client machine and server. When the client initiates connection, server checks for duplicity and if the client has not yet been connected, new sessionId is generated and returned to client. From now on, every newly created message object created on client side is transmitted under this sessionId. It works both directions, because every response object, created on server side is transmitted to a particular client under certain sessionId.

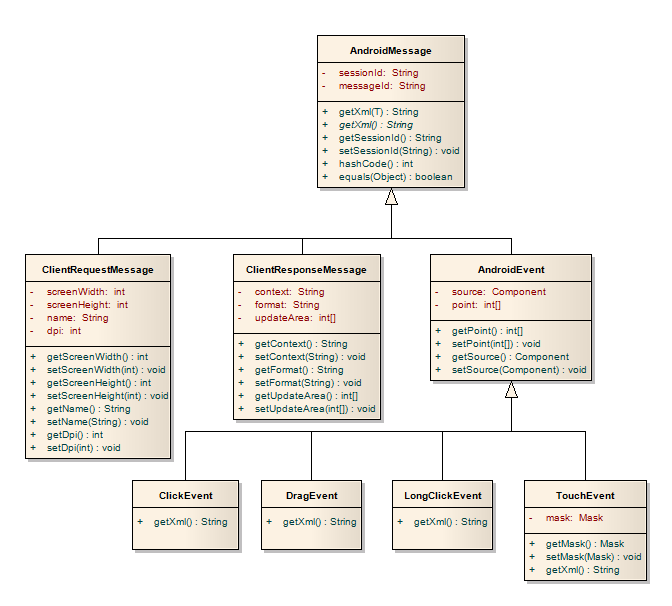


Figure : Class diagram, that represents the hierarchy of message objects.

## Initial client connection

When the construction process of client is over, another logical step is to establish connection with server. Because the part of response is initial layout, server has to know the dimensions of the screen of client device. Client has to therefore ask its hardware about the resolution, ppi information. Other device information like manufacturer, model name and serial number must also be known in order the server could generate session id (composed of serial number and additional random number).

When all the information about the underlying hardware are gathered, initial request message object is created (ClientInitRequestMessage). On XML layer, message object is marshalled into XML string and given to network layer to be further processed. Network layer makes an attempt to send the message to server. There is a possibility that this initial message does not reach server machine. This may be caused by momentary malfunction of wireless network. Another attempt to re-send the initial message is made every 5 seconds, so in case, that the network service becomes available again, message is successfully sent to server.

Network layer of server accepts the message and sends the XML string to be processed within XML layer. Message object is unmarshalled and sent further to business layer. Based on concrete message instance further steps are decided. In this particular case client asks for new connection. Therefore special class called DeviceManager is queried, whether newly connected device is already contained in a list of already connected devices. If no entry is found, new device is added and all relevant information from the message object, like device’s manufacturer, exact model, serial number, screen dimensions and ppi are set on newly created device object on server side. Newly added device is also registered to receive events regarding new graphic context. Due to this feature any synchronization will just be matter of calling the update method on current graphic context and all registered observers will be given the most current version of state layout.

Apart from state layout, metric layout for concrete client still needs to be set, based on given screen dimensions and ppi information. Once all the metric dimensions are converted into pixels and with the knowledge of current state layout, graphic context for concrete client device may be created.

DefaultPainter class serves perfectly our needs. When the new graphic context is needed, the only thing needed is to call getContext method with parameters (ClientDevice, MetricLayout). Each widget is then decomposed into its individual building blocks that are rendered into final canvas in designated image format. Returned BufferedImage needs to further be converted into Base64 string in order to be transferred via network in text format.

Because everything needed for new client to be initialized is ready, response message object may now be created. Not only encoded graphic context, but also image format needs to be included in order the image could be correctly decoded on client side. Final message object is then passed to XML layer, where the underlying serializer converts the input into its corresponding XML representation. Another step is to pass the marshalled message to network layer and send it to client.

Client follows the process in reversed steps. Message is received and sent to the XML layer to be processed. Unmarshalling process reveals the response message object and is passed to business layer. Knowing the resulting graphic format, image is restored from Base64 string and set as a background for ImageView, where appropriate event listeners have already been added. Client device is now ready to receive UI updates as well as generate specified events.

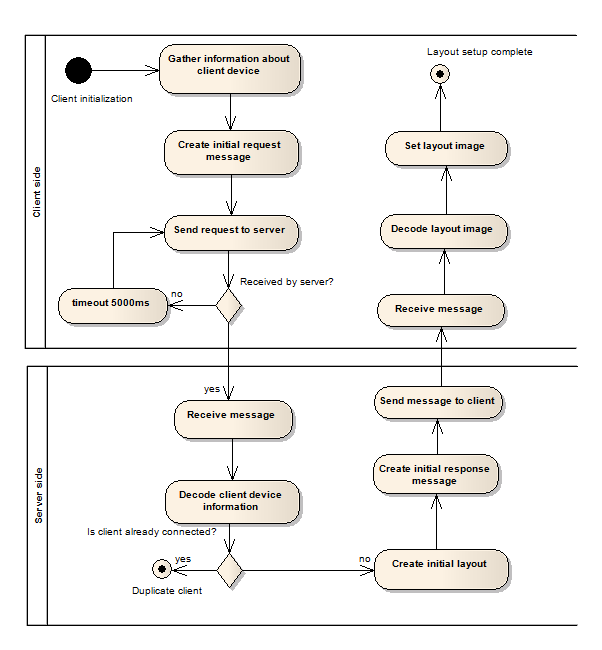


Figure : Activity diagram showing the workflow of client initialization.

## Client update

In a situation, when client device is successfully connected and initial layout has been set, certain UI updates may occur. These updates are caused by particular events, which originated on one of the client devices and it is up to server to decide what to do next. Basically, there are two things to perform.

Firstly, an event has to be propagated further. Server side is intended to be started from an external application. To receive notifications about particular events, external application must implement particular observer interface. The exact procedure about starting server will be discussed further.

Secondly, user interfaces of all connected client devices need to be updated. This is completely managed by the server application itself. The whole procedure begins on business layer of the server, where new event has been recognized.

First thing, that needs to be performed is to reveal the source device, which activated the event. This is quite simple, because the event object has the id of session in its header. With sessionId in mind, DeviceManager instance is used to get particular client device instance.

Another step is to identify source component. There are two layouts present, but only metric one will help us get the desired widget. For this reason, every component has defined its so-called action area. Action area is defined as rectangular shape that identifies the particular widget in space. In order for the whole procedure to work, there must be no overlaps between each component. Now the task of identifying the source component is a matter of simple computational geometry. Iterating through all existing widgets will give us either source component, or null when only blank layout was hit.

Before DefaultPainter is ready to do its part, current component state has to be updated. For this reason, an update call is made on the state layout. Based on current state and identified event, new state of widget is adequately set. In this case, DefaultPainter will use the second overloaded method getContext with parameters (ClientDevice, MetricLayout, Component). Thing worth mentioning is, that even though we’ve been talking about state layout, metric is used as a parameter. As a singleton, state layout can be accessed across the entire application and therefore it is not necessary to specify it as a parameter. Apart of that, metric layout is necessary in order to render graphic context correctly for particular client device.

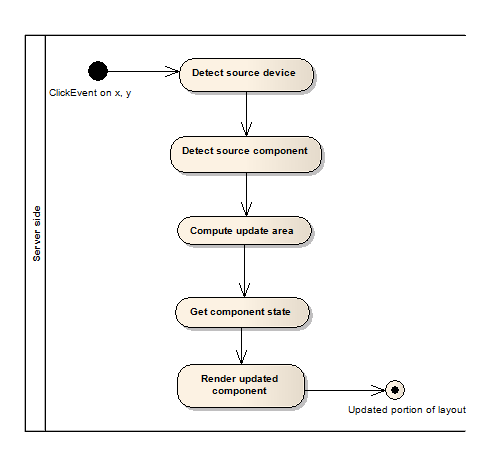


Figure : Activity diagram showing the process of getting only the updated portion of layout.

## , client update, event generation, ...

# Implementation

## Server development

NetBeans IDE 7.3.1

## Client Development

ADT bundle v22.3.0

## Used libraries

### Simple XML (simple-xml-2.7.1)

Simple is used to deserialize and serialize objects from and to XML. Its primary aim is to provide high performance framework that enables rapid development of applications with XML-based communication protocols and configuration files. Using Simple is quick and easy, only several annotations are needed to know in order to begin marshalling and unmarshalling objects. No additional mapping is needed, the whole XML schema is represented by field annotations. Reason, why Simple was preffered to standard JAXB is its easy integration into Android project. Due to this feature, client and server side can both share same message objects that are sent during the communication and there is therefore no need to write platform specific converters.

### JUnit (junit-4.10)

JUnit is well known unit testing framework and an integral part of test-driven development. In this work, JUnit is used to test repetitive tasks, where potential errors may originate. Across the whole platform, there was no need to create mock objects, so frameworks like Mockito or EasyMock are not used.

## Increasing performance

Diagram of the process of generating images

generating only the portion of the image

# Usage

## Configuration files

## General rules

* All colors that are further used across XML files are defined using hexadecimal notation with preceding hash sign (#).
* Every component is given with required element name. This name uniquely identifies the component and is recommended to be used later, when identifying the source of particular event.
* Layout components use absolute positioning. One unit nearly equals one millimeter. This inaccuracy is caused by only the approximate ppi value given by the device’s manufacturer.

### Server side

There are several configuration files available on server. Each pre-defined component has its own file named after the component (DefaultButton.xml, DefaultFader.xml, etc.). These configuration files define the default look and behavior of built-in components. Developers are free to alter any parameters, but it is important to state, that all changes will be projected onto all components of particular type.

Pre-defined components can be used in user-defined layout. Layout file (layout.xml) is designed to configure the remote user interface. Table 1 shows the list of required elements that needs to be specified for particular default component. Not all components are directly visible. For example DefaultRadioGroup is used as a smart container for DefaultRadioButtons. It holds the information about the selected element and ensures that at most one DefaultRadioButton is selected. DefaultRadioButtons placed outside the DefaultRadioGroup behave the same as regular checkboxes and developer cannot assure that only one DefaultRadioButton is selected.



Figure : Example of server-side configuration file, used to specify the remote UI.

|  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Component | name | posX / posY | | | label | pressed | | selected | |
| DefaultButton | x | | x | x | | | x | - |
| DefaultComboBox | x | | x | - | | | - | - | |
| DefaultFader | x | | x | - | | | - | - | |
| DefaultLabel | x | | x | x | | | - | - | |
| DefaultRadioButton | x | | x | x | | | - | x | |
| DefaultRadioGroup | x | | x | - | | | - | - | |
| DefaultToggleButton | x | | x | x | | | x | - | |

Table : List of required elements that need to be specified in order for the built-in components to behave properly

### Client side

Configuration on client side contains single file named config.xml and resides in assets folder. Its main purpose is to specify IP address and port of server machine. At the same time client can only be connected to one server, therefore only one configuration file is needed.



Figure : Configuration file config.xml on client side.

# Testing

# Conclusion

## Causes of higher latency + suggestions on how to decrease it

## Future development

## Difference between my approach and UIProtocol

action triggered by user is directly specified in protocol description, while in my case multiple actions might be triggered, because it is handled by particular implementation of event listener.

# Bibliography

1. A brief history of Android releases. *AndroidGuys.* [Online] 2013. [Cited: December 20, 2013.] http://www.androidguys.com/2013/10/10/key-moments-history-android-releases/.

2. Android. *Wikipedia.* [Online] 2013. [Cited: December 20, 2013.] http://en.wikipedia.org/wiki/Android\_(operating\_system).

# User screeners

# CD/DVD contents