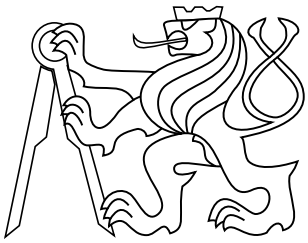
bachelor's thesis

Accessible UIP client for Windows Phone 8

Vojtěch Novák



May 23, 2014

Macík Miroslav Ing.

Czech Technical University in Prague
Faculty of Electrical Engineering,
Dept. of Computer Graphics and Interaction

Acknowledgement

I would like to thank my advisor, Ing. Miroslav Macík for his help and explanations of UIProtocol. Also I would like to thank my family for continuous support in my studies for which I am very grateful. Lastly, I want to thank all my friends from the International Student Club CTU in Prague who made this past year very special and supported me a great deal.

Declaration

I declare that I worked out the presented thesis independently and I quoted all used sources of information in accord with Methodical instructions about ethical principles for writing an academic thesis.

Prague, May 23rd, 2014 .....

Abstract

UIProtocol je jazyk pro specifikaci uživatelských rozhraní, jež je vyvíjen na FEL ČVUT pro účely výzkumu. Je navržen jako klient-server systém, kde server má k dispozici soubory popisující aplikaci a její uživatelské rozhraní. Tento popis je poskytnut klientu, který vykresluje ono uživatelské rozhraní, informuje server o akcích uživatele a zpracovává odpovědi serveru na tyto události. Tato práce popisuje vývoj přístupného UIProtocol klienta pro platformu Windows Phone 8. Tento klient je vyvíjen jako součást koplexního navigačního systémy jehož cílem je usnadnit navigaci postižených uživatelů uvnitř budov. Tato práce se proto zabývá i způsoby, jak zvýšit přístupnost aplikace pro hendikepované uživatele, například zrakově postižené. Klient musí být schopen provádět standardní úkony, jako komunikace se serverem, vykreslování uživatelských rozhraní, obsluha akcí uživatele a další.

Klíčová slova

Navigace, Generování UI, Přístupnost, Windows Phone 8, UIProtocol, C#

Abstract

UIProtocol is a user interface specification language being developed at FEE CTU for research purposes. It is designed as a client-server system where the server runs an app and provides its user interface description to the client. The client renders the user interface, informs the server of user-triggered events and processes the server response. This work describes the development of an accessible UIProtocol client for Windows Phone 8 platform. The client is developed as a part of a complex system for navigation of visually impaired inside buildings. Therefore, we also explore the possibilities of making the application more accessible to people with disabilities, such as visually impaired. The client application has to be able to perform standard operations such as communicating with the server, rendering the user interfaces, handling user actions and more.

Keywords

Navigation, UI Generation, Accessibility, Windows Phone 8, UIProtocol, C#

Contents

Abbreviations

The list of abbreviations used in this document

FEE CTU	Faculty of Electrical Engineering of the Czech Technical University in Prague
UIP	UI Protocol developed for research purposes at the FEE CTU
UI	User Interface
WP	Windows Phone
WP8	Windows Phone 8
WP8.1	Windows Phone 8.1
TTS	Text-to-Speech
OS	Operating System
API	Application Programming Interface
NFC	Near Field Communication
IDE	Integrated Development Environment
XAML	Extensible Application Markup Language

1. Introduction

Spatial navigation and mobility is an integral part of everyday living. Whether a person wants to go on a field trip, go shopping or visit a physician, they need to be able to navigate themselves. For certain groups of people, however, independent exercise of these actions is harder than for others. These include especially visually and motor impaired, elderly, or people with other disabilities.

With today's development of technology, there are new means of helping such people to navigate in various environments without being dependent on another person's help. Development of navigational solutions has been of interest of both commercial and academic research groups and with the availability of GPS, there were great advancements in various fields ranging from navigation of individuals on roads and in cities to agriculture and marine or aviation applications.

Still, indoor navigation, where GPS or similar systems typically are not available, remains a not-so-developed part of the field. Various tools are used to navigate people indoors, such as maps and visual navigation systems (including banners, signs on the floor, flashing light and etc.). These systems, however, assume that their user does not suffer from visual impairment and, generally, is able-bodied.

Indeed, research shows that for a healthy person, sight accounts for 70-90% of information that the brain receives [1]. Visually impaired therefore lack the most important information source and have to use their other senses and assistive tools to navigate and orient themselves. It should be noted that there are situations when not only visually impaired have problems to orientate themselves. In large and complex buildings, such as hospitals, airports, university or government facilities and other, navigating can be a tough task even for a healthy and fit person.

1.1. Motivation

The World Health Organization states [2] there are 285 million people estimated to be visually impaired worldwide, of whom 39 million are blind and 246 have low vision. Also, the number of elderly people - who often suffer some form of motor or other impairment - is growing. This is true especially in the developed economies of western Europe, USA or Japan and according to the US Census Bureau report [3], the number of elderly people (citizens 65-years-old and older) in the United States is expected to double within the next four decades, making up 21% of the US population. The data show that the target group for an indoor navigation system is growing and technologies that assist with indoor navigation will become of greater importance.

To help tackling the problem, there has been a number of research projects proposing solutions to indoor navigation of visually or motor impaired or elderly (see section 2.4). One such project focused on visually impaired, called NaviTerier [4] is being developed at FEE CTU. The project, described in greater detail in section 2.4 is intended to run on a handheld device (a smartphone) and works on the principle of sequential presentation of carefully prepared descriptions of the building segments to the user

by means of mobile phone voice output. NaviTerier has been brought together with UIProtocol (UIP) - another research project of FEE CTU whose goal is to create an platform-indpendent UI description language and which is described in greater detail in the next chapter.

NaviTerier and UIP are being integrated to form NUIP which combines the navigational part of NaviTerier and UI generator running under UIP Server [5]. NUIP consist of several sub-systems: firstly, there is a route planner which supports customizing of the route according to the user abilities and preferences. Secondly, a navigation description generator which creates description of the route with respect to the users limitations was developed. Finally, a context-sensitive UI generator [6] adapts the user interface according to navigation terminals and personal devices used during the navigation.

For this system to be available to as wide a set of users we need to have UIP clients for the most common smartphone platforms. So far, UIP clients are built for Windows, iOS and Android. Since there is no client for Windows Phone 8 (WP8), within this thesis we intend to build a UIProtocol client application for this platform.

1.2. Assignment

The goal of this thesis is to develop an accessible UIProtocol client for Windows Phone 8. We will analyze the UIProtocol and introduce the reader to its features and architecture. Since the developed client is intended to work in conjunction with the NaviTerier project we will also cover the state of the art of indoor navigation systems.

The platform of development was chosen to be Windows Phone 8. Still, we will perform an accessibility analysis of the most common smartphone platforms to be able to compare the platform's friendliness toward users with special needs and put it into the context of its competitors.

The main part of the thesis is the actual development of the UIProtocol client which will be covered extensively. The last task is to verify the functionality of the implemented client, for which we developed a testing application and unit tests.

1.3. Document Outline

The document has five more chapters:

Chapter 2 - Analysis
Contains an analysis which was carried out before the development. The analysis covers the UIProtocol itself, discusses the accessibility of today's smartphone platforms, shows other projects in the field of indoor navigation and shortly explains the accessibility guidelines for WP8 development.

Chapter 3 - Design
The chapter goes through the design of the UIProtocol client. We cover the outcomes of the analysis, requirements for the app and how we designed the application architecture. Several class diagrams are included.

Chapter 4 - Implementation
Establishing upon the design, the fourth chapter covers the implementation of the UIProtocol client, problems encountered during the development and how they were solved.

Chapter 5 - Testing
To finish the development cycle, testing of the developed application is needed. For

that purpose, a testing application and unit tests were developed. This chapter covers the results of testing.

Chapter 6 - Conclusions and Future Work

The last chapter discusses the conclusions and future work.

2. Analysis

Mobile devices and technologies play an important role in today's everyday life. The number of mobile phones, tablets and other handheld devices has been increasing and according to a report by Cisco [7], the number of mobile-connected devices will exceed the number of people on earth by the end of 2014. The world smartphone market is dominated by three main platforms: Android with market share of about 78%, iPhone with 17% and Windows Phone with 3% [8].

Since the thesis is developed with the NaviTerier in mind, it is clear we will develop a client for one of the mobile platforms. UIProtocol clients for iPhone and Android are already implemented and we have therefore chosen Windows Phone as the platform we will develop for.

Reports show that Windows Phone is experiencing an overall growth in the world market [8]. Recently, Microsoft released a new version of the OS, Windows Phone 8.1 and we assume that Windows Phone will keep or strengthen its position, which is also indicated by a report [9] from idc.

Before moving onto the design and implementation, we will introduce the technology used for development, an analysis of UIProtocol, an overview of other navigation systems and an accessibility analysis of the current mobile platforms.

2.1. About Windows Phone 8

The Windows Phone 8 is the first of Microsoft's mobile platforms to use the Windows NT Kernel, which is the same kernel as the one in Windows 8 [10]. Therefore, some parts of the API are the same for both systems. A significant subset of Windows Runtime is built into Windows Phone 8, with the functionality exposed to all supported languages [11]. This gives a developer the ability to use the same API for common tasks such as networking, working with sensors, processing location data and more. Therefore, there is also more potential for code reuse.

Furthermore, Windows Phone 8 and Windows 8 share the same .NET engine [11]. This is to deliver more stability and performance to the apps and improve battery life. Most new devices are now dual or quad-core, and the operating system and apps are expected to run faster thanks to this [11]. The development for Windows Phone 8 is supported by Visual Studio 2013 IDE.

2.1.1. About C# and .NET Framework

C# is a multi-paradigm programming language encompassing strong typing, imperative, declarative, functional, generic, object-oriented, and component-oriented programming disciplines [12]. It was developed by Microsoft within its .NET initiative and its first version was released in 2002 [12]. The latest release at the time of writing is C# 5.0. C# was developed at Microsoft by a team led by Anders Hejlsberg and was inspired by the C++ programming language.

.NET Framework is a part of Windows OS which provides an virtual execution system called common language runtime (CLR) and also includes an extensive set of classes

and libraries that offer a wide range of functionality [13]. The specification called the Common Language Infrastructure (CLI) is an international standard by ISO/IEC 23270:2006[1] and ECMA-334[2] which specifies execution and development environment that allows multiple high-level languages to be used on different computer platforms without being rewritten [13]. The CLR is Microsoft's implementation of the CLI standard. Other CLI implementations include Mono[3], DotGNU Portable.NET[4] and other.

In .NET, C# source code is compiled into Common Intermediate Language and stored in an executable file, typically with exe or dll extensions [13]. When executing the program, the CLR performs just-in-time compilation, producing executable machine-readable code and also handles garbage collection and other tasks. The key point is that the CIL code compiled from C# conforms to the Common Type Specification (CTS) and can interact with code that was generated from the .NET versions of Visual Basic, Visual C++, or any other CTS-compliant language [13].

2.2. UIProtocol

This chapter introduces the reader to the UIProtocol, its architecture and communication between UIP client and server.

2.2.1. About UIProtocol

Universal Interface Protocol (UIProtocol, UIP) is a user interface specification language [14] being developed at FEE CTU for research purposes. At the time of writing, the specification is not publicly available. UIProtocol provides means for describing user interfaces and transferring data related to interaction between user and an UIProtocol based application. It is designed to be cross-platform, programming language independent and easily localized [14].

The goal of UIProtocol is to simplify development for various target platforms: with UIProtocol, a developer only has to describe the application once, and the application then runs on all platforms with UIP client implementation. This should simplify the development process and make one application easily available to multiple platforms. UIProtocol client uses platform-native UI components, which makes it different from HTML-based applications.

UIProtocol is an application protocol that allows for describing the hierarchical structure of GUIs along with the placement and visual appearance of the containers and components. It is designed for a client-server system and for facilitating client-server applications it defines the communication rules between the two. The communication is based on exchange of XML (or other supported) documents. The client first initiates the communication and receives a UI description document from the server.

The description can be of two different types: interfaces, i.e. the UI components and containers, and models which contain the data displayed in the UI components. The communication from client to server only consists of event descriptions, that is, actions that the user has done (e.g. a button click). The architecture of UIProtocol is shown in figure 1 which also indicates the information flow between the client and server. The figure also shows Actions, which are an advanced, optional feature and we will not cover it.

[1] http://www.iso.org/iso/home/store/catalogue_tc/catalogue_detail.htm?csnumber=42926
[2] http://www.ecma-international.org/publications/standards/Ecma-334.htm
[3] Mono Framework http://www.mono-project.com/Main_Page
[4] DotGNU Portable.NET http://www.gnu.org/software/dotgnu/pnet.html

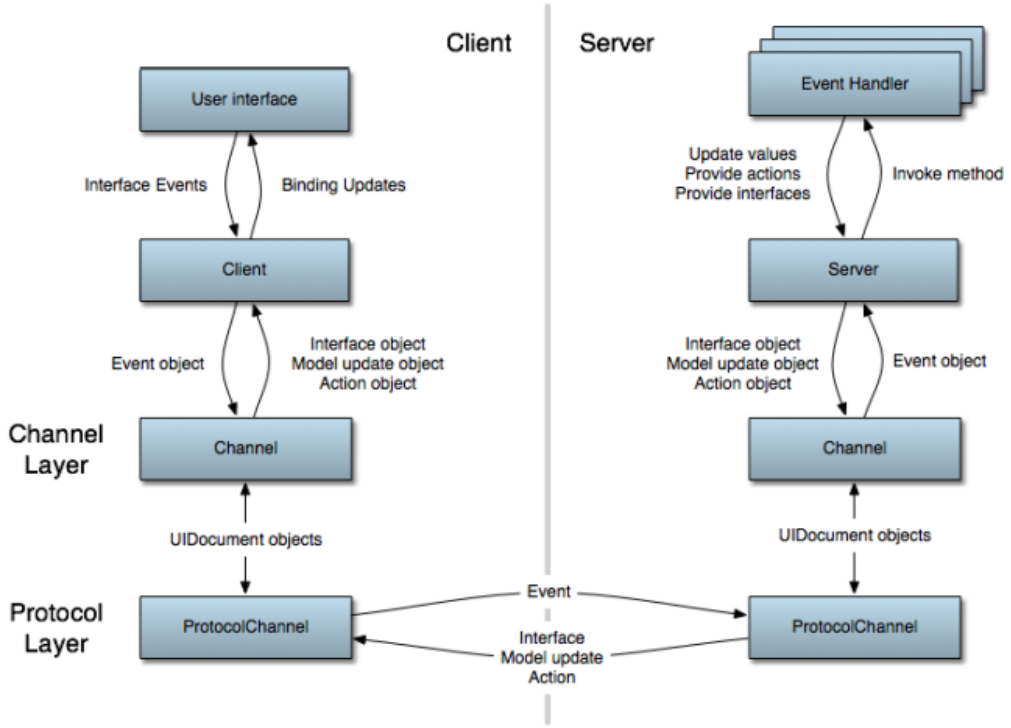


Figure 1. the client-server architecture of UIP, taken from [14]

Let us give an example of when an event is sent to the server: Consider a situation when a user requests a weather app to his device. As he enters his location and presses a button to request the weather information, part of the job is done directly by the client and the other part is sent to the server. The part done directly at the client are easy tasks, such as visual effects when pressing the button. The request for weather information is then sent to the server. Server processes the request and responds by sending the interface structure information, UI components' description and the weather data. This is displayed to the user, who then has other options to interact with the app.

The documents of UIProtocol can be sent in either direction usually through a single channel without waiting for a request, i.e. the server can send updates to the client as soon as the displayed information needs to be updated, without waiting for an update request.

Should there be such need, there is the possibility of both client and server running on the same machine although this is not a typical usage.

UIProtocol Client

UIProtocol client is thin, i.e. no application code is executed on the client side [14]. The device running the client is thought to be the one user directly interacts with, that is, it renders the content to the user and receives input from her. From the UIProtocol point of view, the client device is also considered insecure, i.e. the device may be misused to send invalid data to the server and may be used to attack it.

The UIP client may not implement the whole feature set defined by UIProtocol [14]. What has to be implemented is the minimal functionality, i.e. a client that is able to render user interfaces, send event information to the server and update the application by data coming from it. A use case diagram of actions performed by user in the client

app is shown in figure 2.

UIProtocol Server

UIProtocol server is the part of the architecture which is responsible for evaluating the client events and sending a correct response - this is where the application logic is executed [14]. Server must be able to service multiple clients simultaneously and is intended to run on a machine which is considered safe [14]. A use case diagram of server actions is shown in figure 2.

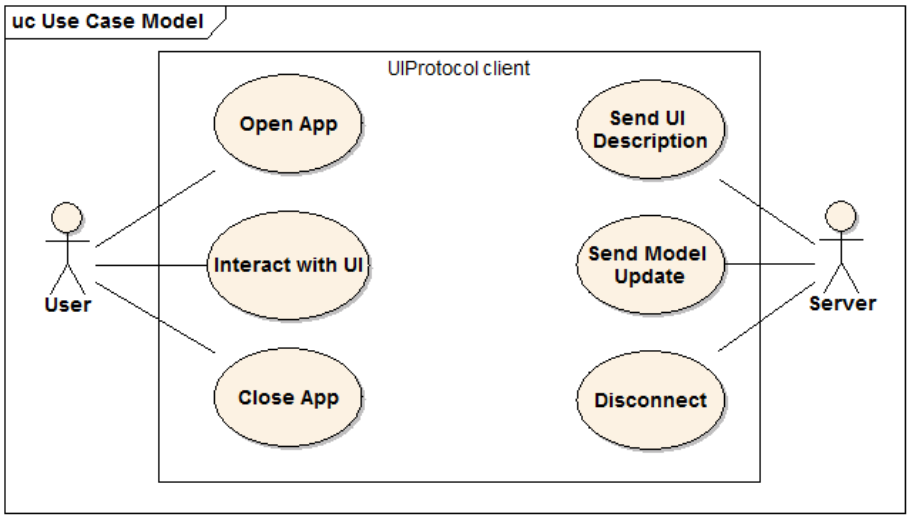


Figure 2. Use case diagram for user and server

2.2.2. Syntax of UIProtocol

The UIP document syntax in listing 2.1 shows the four possible tags and the root element, with the actions tag being optional. The tags define the behavior of the application and are covered later in the chapter, with the exception of the actions tag.

Every UIProtocol document must contain an XML header with the version and encoding (UTF-8 recommended).

Listing 2.1 UIP document Syntax

```
<?xml version="1.0" encoding="UTF-8"?>
<UIProtocol version="1.0">
  <interfaces>
    <!-- interface deinitions -->
  </interfaces>
  <models>
    <!-- model definitions -->
  </models>
  <events>
    <!-- event definitions -->
  </events>
  <actions>
    <!-- action definitions - optional -->
  </actions>
</UIProtocol>
```

2.2.3. Elements of UIProtocol Communication

As mentioned previously, the information exchange between client and server concerns interfaces and models (which are sent from server to client) and events (sent from client to server). In the following subsections we will describe these in greater detail and also include more information on UIProtocol syntax.

Interfaces

Interface describes the structure and components of the user interface. Every interface can nest containers and elements that form a part of user interface. An example can be seen in listing 2.2. The listing includes containers and elements of different types, for example `public.input.text` is a standard component which will be rendered as an UI control into which the user can enter text. It also shows how one interface can be embedded into other. This is done by an element or container with class name corresponding to different interface's class.

The interfaces are uniquely identified by the class attribute (unlike most other objects in UIProtocol and other markup languages identified by id attribute).

Listing 2.2 Interface Description Example

```
<interfaces>
  <interface class="ui.Interface1">
    <!-- interface description -->
  </interface>
  <interface class="ui.Interface2">
    <container class="public.container.panel">
      <element class="public.input.text" id="nameTextBox">
        <property name="text" value="Enter first name"/>
      </element>
      <element class="ui.Interface1"/>
    </container>
  </interface>
</interfaces>
```

Events

Events inform the server that some action was triggered at the client side (e.g. a button press) or that there was some other update (e.g. change in sensor readings). This is the mechanism of client-to-server communication and therefore has to be supported by the client. The event element contains a unique id which specifies the event source. An event can contain any number of properties which describe it in greater detail. An example is shown in listing 2.3.

Listing 2.3 Events Example

```
<events>
  <event id="login">
    <property name="username" value="user"/>
    <property name="password" value="pass"/>
  </event>
</events>
```

Properties Properties are the most nested elements of UIP documents and define the visual appearance of UI controls, their positioning, content and much more. They are heavily used in many UIP's structures - in Models, in styling and more. Every property has to have a name, which defines what feature of the connected object the property describes. For example, the property in listing 2.2 with name `text` defines the text displayed in the text field. Value is a constant that will define the text.

The mentioned property can also contain a key attribute which, if set, binds the displayed value to a model property (see the next paragraph).

Listing 2.4 Property key example

```
...
<element class="public.input.text">
  <property name="text" key="modelName:modelPropertyName"/>
</element>
...
```

Models and Model Binding

Models serve as a storage for data. Within models, data is stored in properties which are uniquely identified by model name and name of the property within that model. As explained in the previous paragraph, properties can be used to define content and appearance of UI elements and more. The property value can be provided as a constant, or it can refer to a model, using the key. The key is separated by colon into two parts - the first referencing a model and the second referencing a property within the model (see listing 2.4 for an example).

For example, consider the element in listing 2.4. The property representing the UI control's text contains a nonempty key. It follows that the `modelName` model is requested from the server. Upon its arrival, the `modelPropertyName` property is looked up in the model and the text of the UI control is set to this UIP property's value. Also, a data binding between the `modelPropertyName` property and the UI element's text property is created so that when the client receives an update of the `modelPropertyName` property, the UI element's text gets immediately updated.

When updating the value of a model property, all UI elements bound to the model property are updated. For example, there may be two representations of humidity level in a given environment (text and a graphical representation). If they are both bound to the same model property, update of the given property will be immediately reflected by both.

Note that Models in UIP are application-wide so they can be referred to from any point of the application.

2.3. Accessibility of Current Mobile Platforms

In this section, we will analyze the accessibility features of today's most common mobile platforms - Windows Phone, Android and iPhone. Since this thesis is about development of a UIP client for Windows Phone 8, we will put an emphasis on this OS.

2.3.1. Windows Phone 8 Accessibility

For the purposes of this project, we are particularly interested in features that may help users with special needs. From this point of view, one of the most important are the

voice commands and speech recognition features which Windows phone 8 has built-in and which support a range of languages.

Speech Features

Users can interact with the phone using speech. There are three speech components that a developer can integrate in her app and the user can take advantage of: voice commands, speech recognition, and text-to-speech (TTS). We will explore these features in the following paragraphs. At the time of writing, the speech features support 15 major languages ranging from English to Russian or even English with the Indian accent. Czech, however, is not supported. To use the speech features, the user has to download a language pack.

Speech Recognition Users can give input to an app or accomplish tasks with it using speech recognition. An example usage can be dictating the content of a short text message (SMS) [15]. This is very similar to the Voice Command feature, but the difference is that speech recognition occurs when user is in the app, and Voice Commands occur from outside of the app [15]. The second key difference is that the Voice Commands are defined on a finite and usually small set of words (commands), whereas the Speech Recognition should recognize words from a much larger dictionary – in the ideal case a whole human language.

Voice Commands When a user installs an app, they can automatically use voice to access it by speaking "open" or "start", followed by the app name [16]. The range of actions that can be triggered by Voice Commands is much wider, the full list of available speech commands that are provided by the operating system is listed in table 10 (Appendix A).
A developer can also define her own set of voice commands and allow users to carry out more advanced tasks within the app [16]. This is important for our work since it allows for exposing a wider range of commands to potential visually impaired users. Note that technically, this still happens from the outside of the app, as described in the previous paragraph.

Text to Speech (TTS) TTS can be used to speak the application content (text only) to the user via the phone's speaker or headset. The spoken text can be simple strings or strings formatted according to the industry-standard Speech Synthesis Markup Language (SSML) Version 1.0 [15]. TTS is also used in some of the other features for ease of access which are covered in the next paragraph.

Other Speech Features A feature named Speech for phone accessibility allows the following [16]:

1. Talking caller ID
   When getting a call or receiving a text, the phone can announce the name of the caller or the number.
2. Speech-controlled speed dial
   User can assign a number to a person from the contact list and then say "Call speed dial *number*" (where *number* is the assigned number) to call the person. Assigning the speed dial number is also speech-enabled.
3. Read aloud incoming text messages

Other Tools for Ease of Access

Windows Phone 8 comes with more features for ease of access which can help lightly visually impaired users: User can change font size in selected built-in apps (The API for determining if the font size was changed by user is available only from WP 8.1 and the app developer can decide whether she will respect the user font size settings. [17]), switch the display theme to high-contrast colors and use the screen magnifier [18].

Mobile Accessibility is a set of accessible apps with a screen reader, which helps use the phone by reading the application content aloud. These applications include phone, text, email, and web browsing [18]. When Mobile Accessibility is turned on, notifications like alarms, calendar events, and low battery warnings will be read aloud. This feature, however, is only available in version 8.0.10501.127 [18] or later. For an unknown reason, an update to this version is not available for our device.

Windows Phone 8.1 contains a new accessibility feature, called Narrator, which allows to read aloud the content of the screen [19]. The feature is intended for visually impaired but since the WP8.1 SDK was released in April [20] and the Windows Phone 8.1 devices are yet to come, we did not have the chance to evaluate the feature.

Conclusions

Windows Phone 8 platform offers some features to make its usage to users with special needs more pleasant. However, there are still gaps to be filled such as the non-existence of a built-in screen reader. Its absence puts Windows Phone 8 out of the question for visually impaired users. The platform has recently been experiencing growth of about 6% in some countries of Europe but only slow growth in others [21].

With the Windows Phone 8.1 update, there is a new screen reader feature included, but it has not been out only for a short time and thus we cannot make any conclusions about its usefulness. It should be noted that the other two major platforms, iOS and Android both include a screen reader.

2.3.2. Android Accessibility

Similarly to the previous section, here we will analyze the accessibility options for devices running the Android operating system. The analysis covers the features of the latest Android OS released at the time of writing, which is version 4.4, code name KitKat. It should be noted that there were no major updates to the accessibility options since Android 4.2.2 Jelly Bean.

Speech Features

Similarly to WP8, Android also offers the option to interact with the device using speech and has some interesting accessibility features. Compared to Windows Phone 8, Android offers a wider language support. Similarly to WP8, an Android developer can take advantage of speech recognition and text-to-speech (TTS). Android comes with a number of built-in voice commands but unlike the Windows Phone, Android does not allow developers to expose their own voice commands. The last important feature on Android is the TalkBack screen reader. At the time of writing, the speech recognition supports more than 40 languages including even minor languages such as Czech. The text to speech only supports the world's most common languages.

Speech Recognition Users can give input to an app or accomplish tasks with it using speech recognition. An example usage can be dictating content of an SMS. As mentioned above, this feature supports many languages but on the other hand, internet connection is required [22] because the recognition is done at Google servers. We do not consider this a drawback, as the UIP client assumes internet availability anyway.

Voice Action Commands In Android, Voice Action Commands are closely related to the Google Now feature. Google Now has a wide range of uses. It can also serve well to users with special needs because it allows to get information using voice. In general, Google Now should provide the user with relevant information when they need it. Google describes it by the phrase "The right information at just the right time". This includes telling the user the weather forecast, showing the best route to work, calling someone, creating a reminder and more [23]. The full list of Voice action Commands is in table 11 (Appendix A).

Note that for some commands, the system gives the user a spoken answer. The current drawback of the system is that it only supports English, French, German, Spanish, and Italian [24]. With other languages, user can only make a voice-induced Google search with no voice response.

Text to Speech (TTS) TTS can be used to speak text to the user via the phone's speaker or headset. The spoken text can be simple strings. The industry-standard Speech Synthesis Markup Language (SSML) is not mentioned in the API documentation. Supported are only major world languages, enlisted in the previous paragraph. TTS is also used in TalkBack which is described in the next paragraph.

Other Speech Features TalkBack is an important functionality that strives for more accessible phone control for visually impaired [25]. Basically, it is a touch-controlled screen reader. When enabled, user can drag finger across the screen selecting the components and getting their acoustic description. By double tapping anywhere in the screen, user can open/use the last selected item. TalkBack also supports gestures. This way, a user can get a complete description of the user interface [25]. The blog post of a blind accessibility engineer from Mozilla Foundation [26] claims that visually impaired users of this system still have to overcome some obstacles.

Other Tools for Ease of Access

Android too comes with more features for ease of access which can help lightly visually impaired users which include change of font size and a screen magnifier.

Conclusions

To date, Android has better accessibility options when compared to Windows Phone, especially for visually impaired. Android comes with the usual functions, such as text to speech, speech recognition or font size settings. It also offers a built-in screen reader, called TalkBack. Android aims to be usable even for visually impaired.

2.3.3. iOS Accessibility

This chapter, covers the accessibility of Apple's iOS. Again, we consider the latest iOS released at the time of writing, which is version 7.0.4. Overall, the accessibility features

of iOS are very similar to those of Android and therefore we will describe them more briefly.

Speech Features

As with the previous two platforms, iOS also offers users to interact with a device using speech. iOS supports speech recognition and text-to-speech in 15 major languages (the same number as Windows Phone 8). iOS also comes with a number of built-in voice commands [27] but does not allow developers to expose their own.

Speech Recognition - Dictation Users can give input to an app or accomplish tasks with it using speech recognition. An example usage can be dictating content of a text. As mentioned above, this feature supports 15 languages and requires an internet connection.

Voice Control - Siri Siri in iOS can be thought of as an equivalent to Android's Google Now. Siri can send emails, set reminders and more [27]. If asked a question, it can read aloud the answer.

Text to Speech (TTS) TTS can be used to speak text to the user via the phone's speaker or headset and this feature was added only recently, in iOS 7.0. The spoken text can be simple strings. The industry-standard Speech Synthesis Markup Language (SSML) is not mentioned in the API documentation.

Other Speech Features iOS comes with a screen reader called VoiceOver. Google's TalkBack is very similar to it. VoiceOver offers very similar functions and allows reading the content of the screen based on touch input and controlling the device by gestures [28]. The mentioned blog post of the blind accessibility engineer from Mozilla Foundation favors VoiceOver over TalkBack [26].

Other Tools for Ease of Access

iOS too comes with more features for ease of access which can help lightly visually impaired users. The user can change font size, invert colors and use the screen magnifier (Zoom) [27]. iOS devices also support a number of Bluetooth wireless braille displays out of the box [27]. User can pair their braille display with the device and start using it to navigate it with VoiceOver. iPad, iPhone, and iPod touch include braille tables for more than 25 languages. Moreover, iOS offers compatibility with hearing aid devices.

Conclusions

Apple's iOS was the first to offer advanced accessibility features and the first to become usable for visually impaired users. Of all analyzed platforms, iOS offers the widest accessibility feature set. iOS allows developers to create apps accessible for a range of users.

2.3.4. Comparison of Analyzed Platforms

A quick overview of the accessibility features of the three most common mobile platforms that we analyzed is given in table 1. The table compares the three analyzed platforms

based on features that are important for visually impaired users. It can be easily seen that to date, iOS has the widest support for accessibility.

Table 1. Quick comparison of the accessibility features of today's mobile platforms

Platform	Built-in screen reader	Text to speech	Speech recognition	Built-in braille display support
Windows Phone 8	no[1]	yes	yes	no
Android	yes	yes	yes	no
iPhone	yes	yes	yes	yes

2.4. Navigation Systems Analysis

There is a number of research projects in the field of navigation systems for disabled. A large number of them are oriented toward visually impaired or people with movement disabilities. Generally speaking, there are ongoing efforts to create maps for indoor environments, with the Google Indoor Maps[2] being the head of this movement. Currently, the Google Indoor Maps are in beta and are not a priori intended for navigation but merely to provide the user with an approximate idea of where they are. In this chapter we will analyze some of the existing works which specifically address the problem of indoor navigation.

2.4.1. NaviTerier

NaviTerier [4] is a research project at FEE CTU which aims at the problem of navigating visually impaired inside buildings. This system does not require any specialized technical equipment. It relies on a mobile phone with voice output which, for a visually impaired, is a natural way of communicating information. The navigation system works on a principle of sequential presentation of carefully prepared description of the building to the user by the mobile phone voice output. This system does not keep track of the user location. Instead, it breaks the directions into small pieces and then sequentially gives the pieces to the user who follows them and asks for next portion when ready.

This system is being integrated with UI Protocol platform, which is another research project of FEE CTU developed for the purpose of creating user interfaces customized to abilities and preferences of individual users. The result is navigation system called NaviTerier UIP (NUIP) [5] which combines the navigational part of NaviTerier and UIP - the context-sensitive UI generator that can adapt the user interface based on several criteria: device (navigation terminal, smartphone, etc.), user (visually impaired, elderly, ...) and environment (inside, outside, etc.). Over the course of development, the system has been tested several times with a total of about 100 visually impaired.

2.4.2. PERCEPT

A promising approach toward an indoor navigation system is shown in the PERCEPT [29] project. Its architecture consists of three system components: Environment, the

[1] Windows Phone 8.1 contains a screen reader feature called Narrator
[2] Google Indoor Maps https://www.google.com/maps/about/partners/indoormaps/

PERCEPT glove and an Android client, and the PERCEPT server. In the environment there are passive (i.e. no power supply needed) RFID tags (R-tags) deployed at strategic locations in a defined height and accompanied with signage of high contrast letters and embossed Braille. The users' part of the environment are kiosks. Kiosks are where the user tells the system her destination. They are located at key locations of the building, such as elevators, entrances and exits and more. The R-tags are present here and the user has to find the one she needs and scan it using the glove.

The glove is used to scan the R-tags and also has buttons on it that the user can press to get instructions for the next part of the route, repeat previous instructions and get instructions back to the kiosk. Also, after scanning the R-tag the glove sends its information to the app running on user's Android phone.

The Android app connects to the internet and downloads the directions from the PERCEPT server. These are then presented to the user through a text-to-speech engine and the user follows them. The system was tested with 24 visually impaired users of whom 85% said that it provides independence and that they would use it.

2.4.3. Blindshopping

Another example of where RFID is used for indoor navigation is presented by Lopez et al. [30]. The system is devised to allow visually impaired people to do shopping autonomously within a supermarket. The user is navigated by following paths marked by RFID labels on the floor. The white cane acts as an RFID reader and communicates with a smartphone which, as in other projects, uses TTS to give directions. The Android application is also used for product recognition using embossed QR codes placed on product shelves. The authors conducted a small usability study from which no conclusions can be drawn.

2.4.4. System by Riehle et al.

An indoor navigation system to support the visually impaired is presented in [31]. The paper describes creation of a system that utilizes a commercial Ultra-Wideband (UWB) asset tracking system to support real-time location and navigation information. The paper claims that the advantage of using UWB is its resistance to narrowband interference and its robustness in complex indoor multipath environments. The system finds user position using triangulation and consists of four parts: tracking tag to be worn by the user, sensors that sense the position of the tracking tag, handheld navigator and a server which calculates the location of the tracking tag and communicates it to the navigator. The handheld device runs software which can produce audio directions to the user. In tests, the system proved useful; it was, however, tested only on blindfolded subjects. Testing showed that they found their way faster with the navigation system.

2.4.5. System by Treuillet at al.

Treuillet and Royer [32] proposed a computer vision-based localization for blind pedestrian navigation assistance in both outdoors and indoors. The solution uses a real-time algorithm to match particular references extracted from pictures taken by a body-mounted camera which periodically takes pictures of the surroundings. The extraction uses 3D landmarks which the system first has to learn by going through a path along which the user later wants to navigate. It follows that the system is not suitable in environments that are visited for the first time. Accordingly to the authors, for the case when it has learned the way, the system performs well.

2.4.6. System by Ozdenizci et al.

Authors of [33] developed a system for general navigation and propose to use NFC tags. They claim the NFC navigation system is low cost and doesn't have the disadvantages present with the systems which use dead reckoning[1] or triangulation[2]. The proposed system consists of NFC tags spread in key locations of the building and a mobile device capable of reading the tags. The device runs an application which is connected to a server containing floor plans. The application is able to combine the information from the sensor with the floor plan and navigate the user using simple directions. The paper only proposes a system and does not contain any testing.

2.4.7. System by Luis et al.

Luis et al. [34] propose a system which uses an infrared transmitter attached to the white cane combined with Wiimote units (the device of the Wii game console). The units are placed so that they can determine the user's cane position using triangulation. The information from Wiimote units is communicated via Bluetooth to a computer which computes the position and then sends the directions to the user's smartphone via wifi. The distance limit of 10 m imposed by Bluetooth can be extended by using wifi but the authors do not consider this relevant because the proposal is in a preliminary evaluation phase. TTS engine running on the phone converts the directions to speech. The system has undergone preliminary testing with two blind and seven blindfolded users and the authors claim that "Results show that blind and blindfolded people improved their walking speed when the navigation system was used, which indicates the system was useful".

2.4.8. Other

There are also research works in the fields of robotics and artificial intelligence that study the problem of navigation. More specifically, they tackle the problem of real time indoor location recognition [35], [36], [37]. Some of these solutions allow for creating a reference map dynamically. Even though they proved to be useful in the domain of robotics and automotive industry, their applications to navigating people are limited, as they require expensive sensors and powerful computing resources. Wearing these devices would make the traveling of the users more difficult and limited. For these reasons, the solution proposed by Hesch and Roumeliotis [38] is interesting because they integrated these devices (apart from the computing) into a white cane. However, the solution has the limitations of being too heavy and large.

2.4.9. Conclusions and Comparison

There are two main approaches to the problem of navigation. In the first, the navigation system consists of active parts which, using triangulation or other methods, are able to determine the user's position at all times and then give her directions based on knowing where she is.

In the second approach, the system does not possess the information about user's position at all times. Instead it synchronizes the position at the beginning of the

[1] Dead reckoning: The process of calculating one's position by estimating the direction and distance travelled rather than by using landmarks or astronomical observations.

[2] Triangulation is the process of determining the location of a point by measuring angles to it from known points at either end of a fixed baseline, rather than measuring distances to the point directly

navigation task and then gives the user directions broken into small chunks. When the user believes she reached the destination described by the first chunk, she asks for the next one and etc. The disadvantage of this approach is that the user can get lost and not end up at the expected location. This problem can be solved by adding more "synchronization points" to strategic locations of the building. These "synchronization points" are often done through NFC tags.

A quick review of the analyzed navigation systems is given in table 2.

Table 2. Quick comparison of the analyzed navigation systems

System	Components	Tested with
NaviTerier	mobile device, kiosk	visually impaired
PERCEPT	RFID, mobile device, kiosk	visually impaired
Blindshopping	RFID, mobile device	visually impaired
Luis et al.	infrared, Wiimote, mobile device	blindfolded users
Riehle et al.	UWB tracking system, mobile device	blindfolded users
Treuillet at al.	camera, laptop	not found
Ozdenizci et al.	NFC, mobile device	not relevant

2.5. Windows Phone Accessibility Guidelines

Microsoft specifies a set of rules that ought to be followed by a developer in order to create an application which is friendly toward users with special needs, which is called Guidelines for designing accessible apps and is accessible online [39]. If a developer follows the principles of accessible design, the application will be accessible to the widest possible audience.

Reasons for Developing Accessible Applications

Users of an application may have different kinds of needs. By keeping in mind the rules of accessible development, a developer can improve the user experience. Also, it is one of the goals of this work to develop an accessible UIProtocol client.
In the rest of the section, we will cover several accessibility scenarios:

Screen Reading

Users who have some visual impairment or are blind use screen readers to help them create a mental model of the presented UI. Information conveyed by the screen readers includes details about the UI elements and visually impaired users depend heavily on it. Therefore it is important to present it sufficiently and correctly. A correctly provided UI element information describes its name, role, description, state and value [39].

Name Name is a short descriptive string that the screen reader uses to announce an UI element to the user [39]. It should be something that shortly describes what the UI element represents. For different elements this information is provided differently. The Table 3 gives more details on accessible names for different XAML UI elements.

Table 3. Accessible name for various UI elements

Element type	Description
Static text UI elements	For `TextBlock` and `RichTextBlock` elements, an accessible name is automatically determined from the visible (inner) text. All of the text in that element is used as the name.
Images	The XAML `Image` element does not have a direct analog to the HTML `alt` attribute of `img` and similar elements. WP8 does not provide an alternative text. WP8.1 provides `AutomationProperties.Name`
Buttons and links	The accessible name of a button or link is based on the visible text, using the same rules as described in the first row of the table. In cases where a button contains only an image, WP8 does not provide an alternative text. WP8.1 provides `AutomationProperties.Name`

The container elements, such as Panels or Grids do not provide their accessible name because it would, in most cases be meaningless [39]. Therefore containers are not covered in the table. It is the container elements that carry the accessible name and other information, not the container itself.

Role and Value Role is the 'type' of the UI elements, e.g. Button, Image, Calendar, Menu, etc [40]. Every UI element therefore has a role. Value, on the other hand, is only present at the UI elements that display some content to user – e.g. TextBox. The UI elements and controls that are the standard part of the Windows Runtime XAML set already implement support for role and value reporting [40].

Keyboard Accessibility For screen reader users, a hardware keyboard is an important part of application control as they use it to browse through the controls to gain understanding of the app and interact with it. An accessible app must let users access all interactive UI elements by keyboard [39]. This enables the users to navigate through the app by tab and arrow keys, trigger an action (e.g. a button click) by space or enter keys and use keyboard shortcuts [39].

Visual experience accessibility

Some lightly visually impaired people (elderly, for example) prefer to consume the apps content with increased font size and/or contrast ration [39]. Since WP8.1, an accessible app UI can scale and change its font size according to the settings in Ease of Access control panel. If color is used to express some information, developer has to keep in mind there might be color-blind users who need an alternative like text, or icons [39].

Additional Guidelines

There is a number of other guidelines for developing accessible applications. For example, it is recommended to not automatically refresh an entire app canvas unless it is really necessary for app functionality. This is because the screen reader assumes that

a refreshed canvas contains an entirely new UI – even if the update considered only a small part of it – and must recreate and present the description to the user again [39].

Since Windows Phone 8.1 there is `IsTextScaleFactorEnabled` property available for every text element which, if set to true, will override the app's font-size setting and set the font size to whatever value it was set by user in the Ease of Access control panel [39].

3. Design

After analyzing the problem, this chapter will go through the design of the application architecture of our UIProtocol client. The design phase is of crucial importance as it is the time when important design decisions are made. In this phase, the application's architecture needs to be thought through so that its future extensions are relatively easy to implement and cost of maintenance is low.

From the analysis, we conclude requirements for the application which will be developed. Further on, the section describes the design of several sub-systems which are responsible for handling the communication, models, events, inner representation of the UI elements, their rendering and more. The chapter contains several UML diagrams. Note that most of them are simplified.

Even though there are existing implementations of UIProtocol client, the design of this one was not influenced by any of them.

3.0.1. Requirements

The client application will be developed and run on a Windows Phone 8 device. Since the entire user interfaces and information about events is intended to be transferred over wireless internet connection, there will be a delay present in the application's reaction time, which is a inescapable consequence of the client-server architecture. The delay should be reasonably small to allow for a comfortable usage of the application. Even with this delay, the application should perform well in terms of UI rendering times, reaction time and overall feel.

There is a number of requirements an app should meet in order to be truly accessible. In our analysis, we found that the support of Windows Phone 8 for accessibility is lower than at the competing platforms. Namely, support for a key accessibility feature, the screen reader, is not present by default. This can be a major flaw to the application accessibility – especially for visually impaired who would have to use a third-party screen reader in order to be able to navigate through the app[1].

We may propose some new features that could be implemented by UIProtocol to increase its own support for accessibility. At any rate, even with the Windows Phone 8 platform's low accessibility support, the developed app will remain a functional UIP client capable of handling valid UIP documents.

Summary of Requirements

We have developed the following lists of non-functional and functional requirements, respectively.

Non-functional requirements:

- UI components have platform-native look
- Client app will be will be written in C#
- The client should not use much phone resources when idle

[1] Windows Phone 8.1 users may use the Narrator feature

- App should be stable and able to process valid UIP documents
- Compatibility with UIP specification, draft 8
- Ability to run on any WP8 device
- UI loading times below 0.5 s for UIs of usual size, with stable internet connections

Functional requirements:

- Support for basic user interface elements
- Graceful degradation for unsupported elements
- Support for binding and model-wide binding
- Support for interpolated model updates (animations)
- Support for UI generator API
- Support for Events
- Support for absolute and grid layouts
- Support for styling (font size, colors, etc.)

3.0.2. Client-server Communication

The client will communicate with the server over TCP connection which will be handled by a standard socket. Upon this communication channel, UIProtocol XML files will be transferred.

Once the client connects to the server and goes through the connection procedure described in [14], the server sends the XMLs describing the UI. The UML diagram of the classes responsible for the communication is shown in figure 3. Since some of the methods will only work with the passed parameters and not modify the object's state, they will be made static.

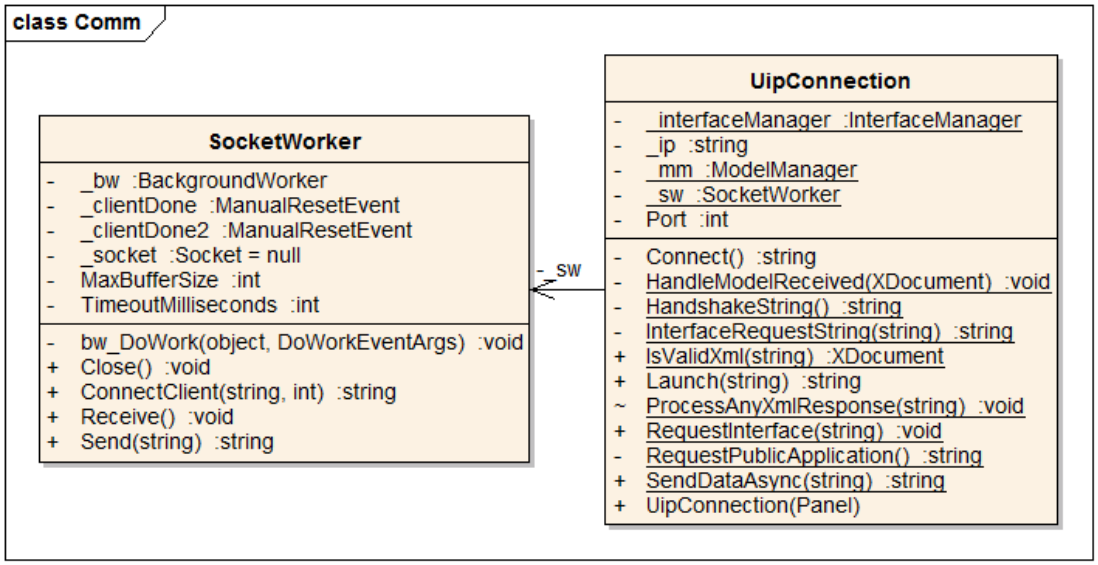


Figure 3. Class diagram of communication classes

Apart from the mentioned classes that will serve for communicating the XML messages, there there will be another class responsible for acquiring resources (such as images) from the server. For this purpose, the server is awaiting a HTTP connection on another port and the client can open a connection and make a standard HTTP request for the resource. This functionality will be implemented in the `HttpConnection` class.

3.0.3. Parsing XML Into Inner Object Representation

After the UIP documents will be received by the `UipConnection` class, they will be passed to instances of `ModelManager` and `InterfaceManager` classes. `ModelManager` will be responsible for processing possible new models or model updates and will be discussed later in more detail.

All of the UIP elements need to be represented by objects, so that they are easily manipulated. To create the object representation, `InterfaceManager` class will process the XML data that describes the UI by recursively traversing the XML tree and creating instances of `Interface`, `Container` and `Element` classes, based on the type of the considered XML node. These instances will represent the UIP elements of the same name - UIP interface, UIP container and UIP element, respectively. This way, every UIP element will be parsed into an inner object representation that will be easy to handle in further work with the objects.

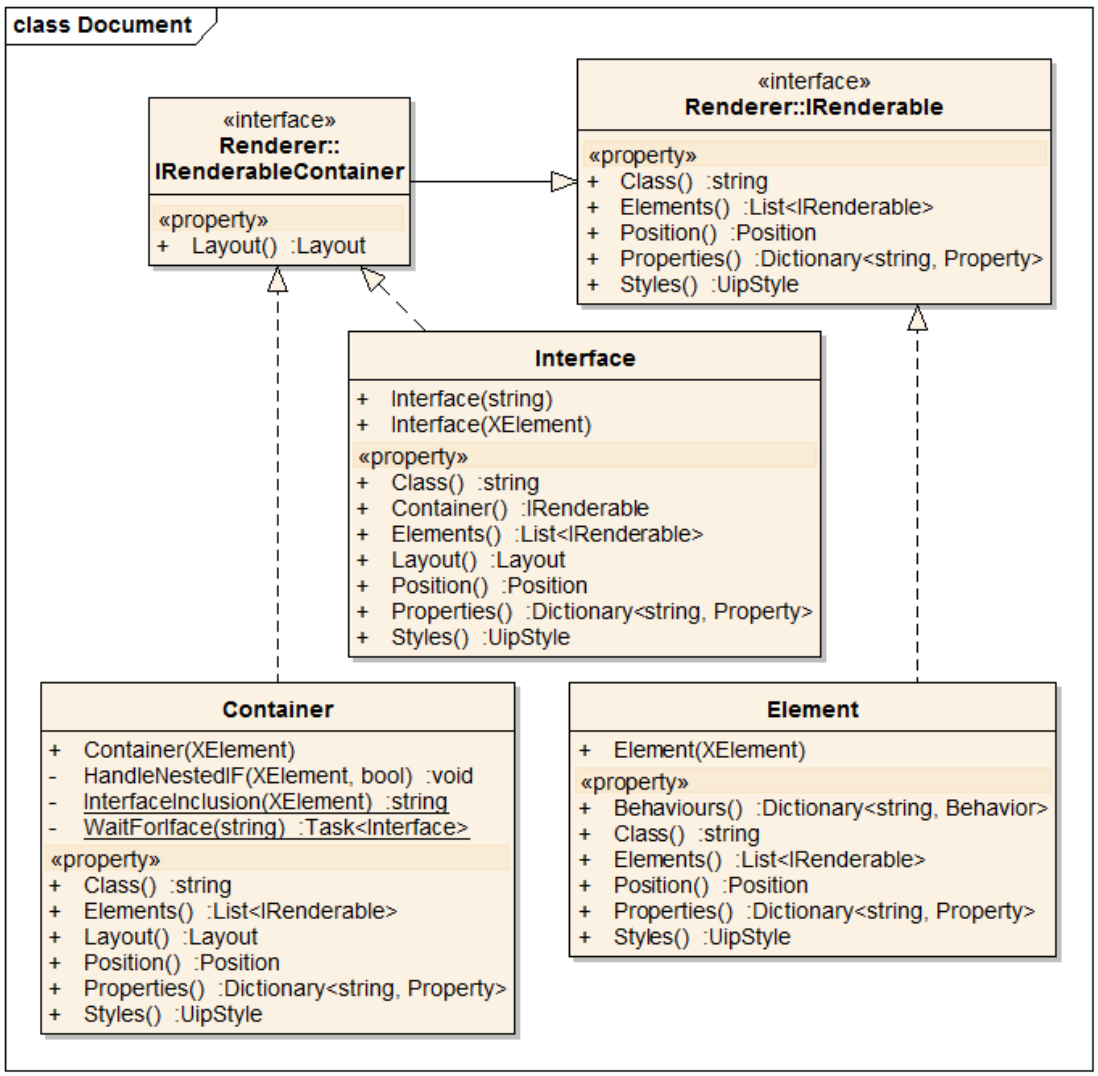


Figure 4. Class diagram of the classes representing UIP interface, container and element

An important component is the `IRenderable` interface which is implemented by the `Interface`, `Container` and `Element` classes. This interface represents the functionality all of the classes have in common - most importantly the UIP class (i.e. type of the

UI control - see 2.4 for example of `public.input.text` which represents the C#'s `TextBox`), UIP properties and contained elements. `IRenderableContainer` only extends the `IRenderable` interface by adding a method for obtaining layout. Since layout is a container feature, only `Interface` and `Container` classes will implement it.

3.0.4. Managing Models and Binding

To conform the UIP specification, the client must be thin, i.e. it will only store as much data as is needed to render the required interfaces and not execute any code on that data. The data shown to the user can be either constant or come from models, which are designed to be a data storage, as discussed in 2.2.3. Models will be managed by one instance of `ModelManager` class.

If an UIP element has a property that refers to a model, `ModelManager` will be responsible for requesting the model containing this property. Once the model is received, `ModelManager` will store all its properties and will manage future model updates. Note that the updates can come from the server at any time. `ModelManager` will also need a reference to `InterfaceManager` because server's models can contain a request to render an interface. The relationship between classes that are involved in model management is shown in figure 5.

When a UIP property refers to a model, a binding will be created so that when the UIP property is updated, the update is shown in the UI. For that reason there has to be a data binding between the two. We will make use of the data binding API which is built-in to the Windows Phone platform.

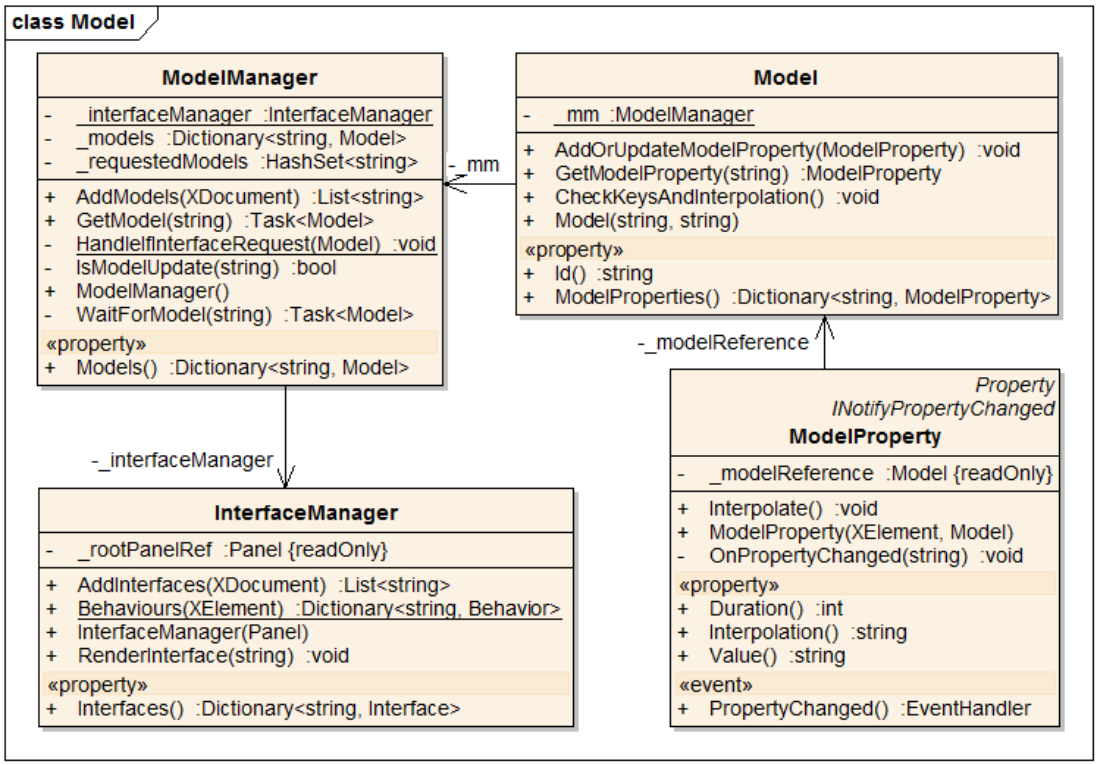


Figure 5. Model and interface management class diagram

3.0.5. Managing Interfaces

Interface is the root element for all UI controls in UIP documents. An application can contain a large number of interfaces. We therefore need a class to keep the interface information. `InterfaceManager` will serve as a place for storing information about received and requested interfaces. The most important methods implemented in it will be for adding received interfaces, obtaining an interface and rendering. The process of rendering is described more closely in the next paragraph.

3.0.6. Rendering the UI

The information about UI elements comes from the server in form of XML description. This description is parsed into inner object representation - classes shown in figure 4.

After this is done, `InterfaceManager` will call the `Render()` method of the `Renderer` class. This method will traverse the tree of the newly created instances of classes from figure 4 and for each one creates a new class instance which will wrap the platform-native UI controls, as described in the next section.

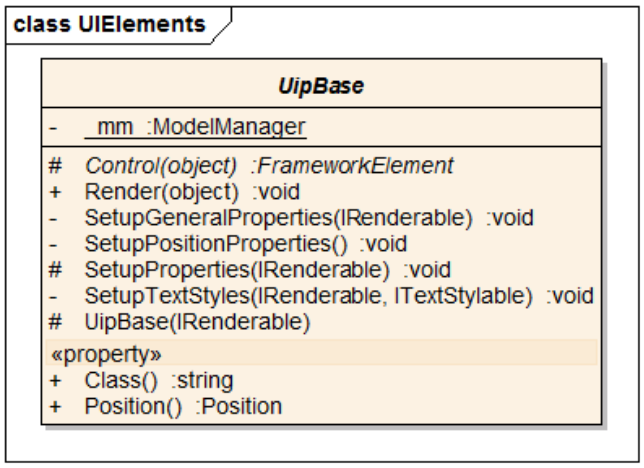


Figure 6. UipBase class diagram

3.0.7. Representing the Platform-native UI Components

There is one more step toward the controls that are rendered to the user which involves the transition from the inner object representation to the platform-native components.

The platform-native components will be wrapped into wrapper classes whose names will indicate the component which is wrapped inside (i.e. `UipPasswordBox` will be the wrapper class of WP8's `PasswordBox`). There will be an abstract base class, `UipBase` which will contain everything the wrapper classes have in common: methods for binding to models, and support for styling, element dimensions and positioning. Any particular UI element needs to inherit from the base class in order to support rendering, model updates and other functionality provided by the base class.

3.0.8. Events

The application is designed to support the client-to-server communication in form of events. Events are the only data sent by the client and their intent is to inform server of an user action or request missing data - models.

Events could be divided into two categories.

Static
Events that are static and are hard-coded within the application. These events are used rarely, typically while going through the procedure of connecting to the server. Currently these are the `public.connection.connect` and `public.request.model`, as well as `public.application`.

Dynamic
Other events are the ones triggered by the user or by the client itself, when it requests a model. These events are dynamic, created at runtime. The event firing - i.e. notifying server of an action taking place, is a relatively simple process which will be handled by two classes. Its working is described in the Implementation chapter.

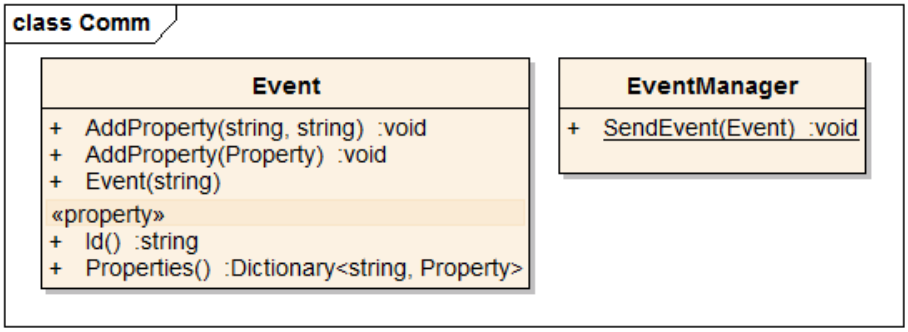


Figure 7. event class diagram

3.0.9. Properties

Properties are the most nested objects in UIP documents. They are used extensively within many classes, including `Layout`, `Event` or `Element`. In all classes they will be stored in dictionaries, identified by their name. The `ModelProperty` class used in Model will inherit from `Property` class.

3.0.10. Layouts

Windows Phone 8 supports two main types of layouts: absolute and dynamic. In an absolute layout, child elements are arranged in a layout panel by specifying their exact locations relative to their parent element. Absolute positioning doesn't consider the size of the screen.

In a dynamic layout, the child elements are arranged by specifying how they should be arranged and how they should wrap relative to their parent. With dynamic layout, the user interface appears correctly on various screen resolutions.

Both layout types will be supported in our client. Even though absolute layout is not recommended by the accessibility guidelines [39] it is a basic form of layout and decision has been made to support it.

Layouts are a feature of containers and interfaces which support it through the `IRenderableContainer` interface. Every container, therefore can place its content into one of the two layouts. As with any property, the positioning will be binding-enabled.

The client will support a container with scrollable content (through `ScrollViewer`) so that content which does not fit in the screen can be reached by scrolling.

3.0.11. Configuration

Client will have support for basic configuration - i.e. the default IP address and ports on which the socket is tying to connect to the UIP and the HTTP server. Also the constants which are used in the portions of XML throughout the application will be stored in one static class so that they are easy to maintain.

3.0.12. Behaviors

Behaviors are used to attach event listeners to UI controls. C# has a built-in support for events through Event and Delegate classes and we will take advantage of it.

Behaviors will be attached to the classes inheriting from `UipBase` as event handlers. Each handler is for one type of behavior and when the event is fired, the event handler catches it.

3.0.13. Interpolation

The client will support interpolation of UI controls. In the context of UIProtocol, interpolation means animation of UI elements. For example, some action may trigger interpolation which will cause an UI element to move on the screen of the phone. A model update will specify the direction, duration and position where the UI element should move. UIProtocol specifies multiple types of interpolation, our client will only support linear (`public.number.linear`) and immediate (`public.immediate`) interpolations.

4. Implementation

After providing an analysis of UIProtocol, settling down on the requirements and working out the design of the app, we can now present how the application was developed, what technologies were used, what problems were encountered and how they were tackled. It has been said that the app architecture was not influenced by any of the existing implementations. The client-server communication, however, was observed from another, already implemented client. Inspecting the communication logs helped to develop this client.

4.0.14. Development Environment

As previously said, the application was written in the Visual Studio 2013 IDE and using C#, a programming language developed by Microsoft. The reasons for choosing Visual Studio (VS) are clear: VS is the main development tool for the whole .NET platform, fully supports C# and Windows Phone development and debugging. VS is therefore the main tool to be used for most .NET development.

The programming was backed up by running the code directly on a Windows Phone 8 device, namely HTC 8S. We also used ReSharper, a useful plugin for code inspection, maintenance, refactoring and coding assistance.

4.0.15. Overview of the Core Classes

In this section, we will cover the most important classes of the application, to give a brief idea of how the UIP documents are handled, stored, processed and how the UI is rendered. There are several tables in the following pages, documenting classes for inner UIP Document representation (table 4), rendering support (table 7), the communication with the server (table 5) and the classes for management of interfaces and models (table 6).

Table 4. UIP Document representation classes

Class name	Class description
Interface	This class represents the UIP interface as a container for more UI elements. This class has its own position, a `Container` instance and can be embedded into another interface, as specified in listing 2.2.
Container	This class stores the information about particular UI elements. A `Container` can contain instances of other `Containers` and instances of `Element` class.
Element	Class representing particular UI elements such as button, textbox and more.

Table 5. UIP Server connection classes

Class name	Class description
UipConnection	Initiates the connection and is responsible for sending events to the server and processing its responses. Does basic XML validation.
SocketWorker	Handles the socket communication with UIP server. Sends events and runs a separate thread for receiving server's responses.
HttpConncetion	Class responsible for acquiring resources via HTTP.

Table 6. ModelManager and InterfaceManager classes

Class name	Class description
ModelManager	Keeps and updates all requested and received models. Is implemented as a singleton class.
InterfaceManager	Stores all received interfaces. Provides getter method and method for rendering an `Interface`. Is also implemented as a singleton.

4.0.16. Communication With UIP Server

As shown in table 5, the communication with server is implemented in `UipConnection` class which exposes its functionality for sending events to the rest of the application - namely the `EventManager` class. It also is responsible for processing any XML data received from the server. This data is forwarded to it from the `SocketWorker` instance.

`SocketWorker` is the low-level socket communication class which sends events to the server. It also runs an instance of `BackgroundWorker` class which, in an extra thread, awaits data from the server. The reason there is a separate thread for receiving data is that we cannot make any assumptions about when the server will send data to the client. Generally speaking, server can decide to send model updates at any time - not only as a response to a certain user action.

4.0.17. Model Updates and Binding

Any property of UIP document can, instead of direct value, contain a reference to a model and its property, as described in 2.2.3. As an example, let us consider a button. The text displayed in the button (its `Content`) can be either hard-coded into the UIP document or there can be a reference to a model property. If the reference is present, the `ModelManager` looks into an internally stored dictionary of models and if the model is present, the value of its corresponding property is immediately used. If that is not the case, `ModelManager` makes a request for the model and once it arrives, its corresponding property's value is used. In both cases, a binding is created so that the future updates of the model property are correctly propagated throughout the application. The `ModelManager` class only needs to be instantiated once, so that the application state is stored in one place. Therefore `ModelManager` is a singleton class.

The code which acquires the models and creates bindings between model proper-

ties and properties of UI elements makes heavy use of asynchronous methods. The async/await operations were introduced with C# 5.0 and provide the developer with a comfortable way to deal with operations that are potentially blocking.

Because requesting and receiving models typically happens over a wireless connection, it can be considered potentially blocking. If a model request and its receiving was blocked within a synchronous process, the entire application would be forced to wait. However, by taking advantage of the asynchronous programming, the application continues with other work that does not depend on the web resource.

This is also the reason why sometimes the UI is drawn gradually. This happens when properties refer to models which have to be requested and received from the server - for example if a button text color is specified in a model. In such case, the button is rendered with the standard text color, and only once the model arrives, the font color is changed to whatever value is specified in the model. The delay before the update is noticeable but very short, and therefore does not limit the app usage in any way. The same happens to resources (images) from the HTTP server which are also fetched asynchronously.

4.0.18. Interpolations (animations)

Interpolation allows to move UI controls on the canvas. Because interpolation uses model updates, the prerequisite for it is that the UI control's coordinate which we want to change is bound to a model (e.g. if we want to move a button horizontally, we need to bind the `x` coordinate of its `Position` to a model property). It is triggered by an event which is fired by activating some UI control. The server responds by a model update whose body contains "interpolation" and "duration" attributes. An example of such model update is shown in listing 4.1.

Listing 4.1 UIP model update specifying interpolation

```
<UIProtocol version="1.0">
  <models>
    <model id="cz.ctu.bp.sampleModel" update="partial" duration="500"
        interpolation="public.number.linear">
      <property name="x" value="100" />
    </model>
  </models>
</UIProtocol>
```

Interpolation works through model-wide binding. The `ModelManager`, when it receives an interpolation model, starts a new `Task` which periodically updates the given property value (property `x`, considering our example). Because of the data binding, this update also triggers update of the UI control's position on the canvas, which causes the UI control to move (animate).

4.0.19. Binding Converters

It has been said that any property can be bound to a model. However, properties in UIP document can convey a wide range of information - including color, font size, row and column position in a grid and etc. Since the model updates are always received as string, the binding has to be provided with a converter which, considering the given examples, converts the received string to `SolidColorBrush`, double and integer types, respectively. All of our converters implement `IValueConverter` interface, which is the standard way to apply custom logic to a binding.

4.0.20. Implementing the UI Element Classes

When deciding how to represent the platform native components which are displayed to the user, we chose to use wrapper classes which will expose the wrapped object's methods and at the same time be able to set up its properties from an instance of `Element` class (i.e. from the inner object representation).

All supported native components are therefore wrapped into other classes whose names indicate the enclosed UI element (e.g. `UipButton` is a wrapper class of the standard `Button` class). All of these wrapper classes inherit from abstract class `UipBase` which provides common support for model binding for all inherited components. This way, adding new UI components with binding support is made easy.

The `ITextStylable` interface is implemented by classes which contain text which can be styled. For example, a `UipTextBlock` implements this interface in order to be able to set font size, color and more. `UipPanel`, on the other hand, does not implement it because a container itself does not have any text to be styled.

The figure 8 shows a class diagram of a few wrapper classes and also `ITextStylable` being implemented.

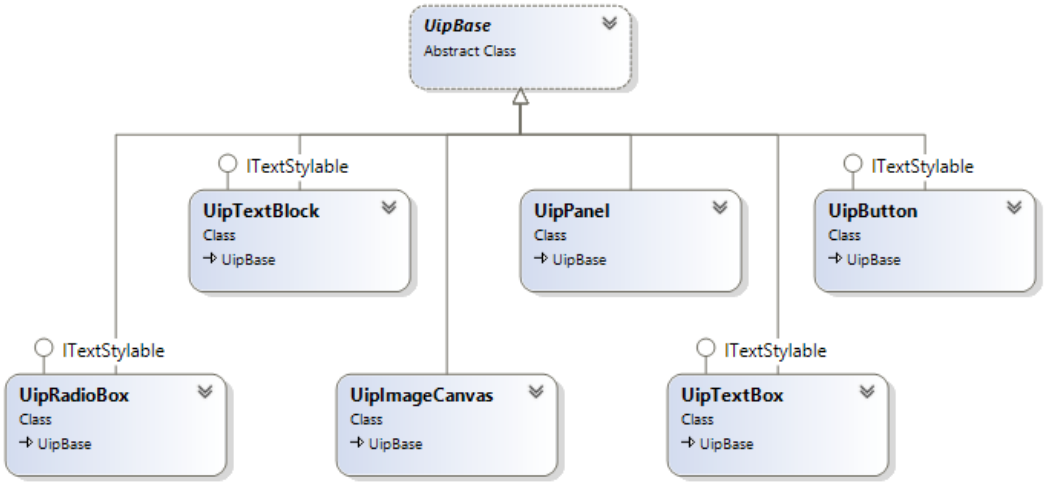


Figure 8. Inheritance tree of several sample UI classes

4.0.21. Rendering

Rendering is always triggered by `InterfaceManager` class which calls the `Render()` method of `Renderer` class. Part of method body is shown in listing 4.2. The method receives an `IRenderable item` (i.e. instance of `Interface`, `Container` or `Element`) and a parent container. The method traverses the tree of UI elements, calling itself recursively if it encounters a container.

The foreach loop iterates over all elements in the container and creates a wrapper class for each of the `IRenderables`. Note the graceful degradation taking place. This wrapper class contains a platform-native UI control and sets it up according to the `IRenderable`. `UipBase`'s `Render()` method is called at the end of the foreach loop.

From then on, rendering continues in the `UipBase` which adds all of the UI controls to the parent Panel - this makes them visible to the user.

Listing 4.2 The Render() method

```
public void Render(IRenderable item, object parent)
{
  ...
  foreach (var element in item.Elements)
  {
      if (element is IRenderableContainer)
      {
          Render(element, pass);
          return;
      }
      UipBase uib = null;
      switch (GracefullyDegrade(element.Class))
      {
          case Consts.UIelements.TextBox:
              uib = new UipTextBox(element);
              break;
          ...
          case Consts.UIelements.InputTime:
              uib = new UipTimePicker(element);
              break;
      }
      if (uib != null) uib.Render(pass);
  }
}
```

UI elements can be rendered into containers with absolute or grid layouts. If the interface is larger than the screen dimensions, it should be put into the `public.scroll` container. That ensures it is rendered into a scrollable container, making all its elements reachable for the user.

When using `public.scroll`, the `position` tag specifying interface dimensions should be placed as an direct ancestor. If the dimensions are not provided, the client will assume the interface has screen dimensions.

The `InterfaceManager` is implemented as a singleton because we only need one place to store the `Interface` instances. Table 7 contains an overview of the classes taking care of the rendering mechanism.

Table 7. Classes ensuring the rendering of UI elements

Class name	Class description
Renderer	The main class responsible for rendering the elements stored in the classes of table 4. Its rendering method walks through the tree structure of UIP Document and invokes rendering of each element. It also does the graceful degradation of unsupported elements.
IRenderable	An interface which defines methods for acquiring class, style, position and other properties of UIP elements. It is implemented by all classes in table 4.
IRenderableContainer	Extension of `IRenderable` interface. It provides support for layouts and is implemented by instances of Interface and Container.

4.0.22. Graceful Degradation

Graceful degradation is a mechanism which replaces unsupported UI elements by supported ones while rendering is being done. This replacement, of course, does not happen without loss. To illustrate this, let us consider the following example:

Server asks the client to render an element of class `public.input.choice.single` – an UI element known under the WP8 platform as `ListPicker`. This element, however, may not be supported by the client. If this is the case, the graceful degradation takes place and degrades this to `public.input.choice` which will be rendered as a group of radiobuttons (assuming this basic UI element is supported).

If the graceful degradation does not find a suitable ancestor, an empty `TextBox` is rendered instead.

4.0.23. Event Communication

Event management is relatively simple: two classes take care of it.

`Event` class represents a particular event that will be sent to the server. It contains all the necessary information for the server to be able to identify the event that has been triggered, as specified in [14]. This identification information is stored in properties that are attached to the object.

`EventManager` class provides a method for sending an event. The class does not contain any state and is therefore static, which makes it easily accessible from any point of the application. Its purpose is to wrap the event into an UIP Document header and forward the event to `SocketWorker` class instance which does the actual job of sending it to the server.

4.0.24. Constants

The code contains a large number of string constants that are used to acquire XML elements from the XML documents or to create events that are sent to the server (typically in static methods of the `UipConnection` or `Event` classes). To make the maintenance of these constants easy, they are all stored centrally in the `Consts` class and split into the following categories:

UIelements
: All constants used while parsing the UIP documets into the inner object representation. Examples are `public.scroll` or `public.input.text`.

Events
: Constants used for constructing events such as `public.request.interface`.

Styling
: Constants that have to do with appearance of UI controls or layouts. Examples include `width`, `font.color` or `public.grid`.

General
: Constants that do not fit into any of the previous categories.

4.0.25. Configuration

The client connects to the UIP server and HTTP server on ports that are settled on by the specification ahead of time. The port, default IP address and socket buffer size are all stored in the `Settings` class where they can be easily modified.

4.0.26. Problems in Implementation

We encountered several problems in the development and some of them are described in this section.

First issue was related to receiving communication from the server. The difficulty was that the model updates can arrive at any time, not only as a response to a certain user action. The problem was solved by running the receive operation in an extra thread. This way, the client socket is always ready to receive data.

Another issue was related to chained UIP properties. The UIP properties are transitive - consider property A whose key refers to property B. Property B also has a key which points to property C. Property C contains a constant value. The constant has to be propagated back to properties A and B. Also, when property C is updated, the update has to be reflected in properties A B. A simple solution would be to create a custom `DependencyProperty` and then create a binding. However, we were unable to create the chained binding and chose an alternative way instead. The solution involves creating event listeners. In the example given above, the property B would create an event handler that would hook up on changes in C's value. Similarly, property A would be listening for updates of B.

Next, rendering containers of the class `public.scroll` into grid layout created unnecessary top and left margins of the container. These margins were removed by setting the `VerticalAlignmentProperty` and `HorizontalAlignmentProperty` so that UI controls in grid align in the direction of the top left corner.

Unfortunately, some of the native UI control classes inherit from different base classes and provide different interfaces for setting their text. For example, the text of the `Button` class is indicated by `Content` property whereas the text of `TextBlock` is indicated by `Text` property. Also, some components such as combo boxes have multiple items whose text can be set. Therefore we introduced the `ITextStylable` interface which contains methods for setting the text and for binding.

5. Testing

Testing is an important part of any application's development cycle. By testing we want to show that the client is able to process valid UIP documents. We will perform two types of tests: test using an verification UIP application on a Windows Phone 8 device and unit tests. Visual Studio offers means for running and debugging the application.

5.1. Functional Testing

Testing was primarily done by directly running the developed solution on a HTC device whose details are listed in table 8. For the purpose of testing, we developed a testing app written in UIProtocol XML which was then run on the client.

The testing application covers the following scenarios:

Interface switching
Switching of the interfaces is one of the basic tasks a UIProtocol client has to implement, since this is the mechanism of navigating through the application. Our testing app contains several interfaces with different UI components and layouts.

Models and their updates (including interpolated)
The testing application contains several situations when a model update is sent from the server, including a linearly interpolated model update which is used to animate the position of a button.

Interface embedding
This test evaluates whether embedding (nesting) one interface into other works as expected. Interfaces can be included using both UIP element and container.

Model-wide binding
In our testing application we bound several properties to models. The binding was tested with: text color, text size, position of UI control (interpolation), integer value.

Events
Testing application contained several events that were triggered by different user actions. The events were sent by different button presses (button, radiobutton) and using several UIP `Behaviors`.

Table 8. Testing device details

Information	Value
Phone model	HTC 8S
OS version	8.0.10327.77
Screen resolution	480x800px
RAM	512MB
Processor	Qualcomm S4 1 GHz, Dual-core

5.2. Unit Testing

To widen the coverage of testing we also added unit tests of the core functionality. They are added as a separate Visual Studio project. The tests all pass.

5.3. Monitoring and Profiling

Visual Studio contains several features that help to understand the quality of the application, ranging from code analysis to profiling.

The app monitoring feature aims to capture key metrics that are relevant from a quality perspective, and then to rate the application based on these metrics [41]. The goal of profiling is to help understand the performance of an application [42].

Visual Studio offers app performance testing using Execution and Memory profilers. Execution profiler analyzes the performance of drawing visual items and method calls in the code [42]. Memory profiler analyzes object allocation and the use of textures in the app [42].

We performed all of the tests and summarized the results into table 9. The results revealed that sometimes the application probably has a high CPU usage. Significant bugs or errors were otherwise not found. However, it must be noted that the testing application was not large scale and so the testing might not have been thorough.

Table 9. Results of monitoring and profiling

Test	Result
Monitoring	App startup time meets requirements. App is responsive.
Execution profiler	In some parts throughout the profiling, execution profiler reports high CPU usage by system threads.[1]
Memory profiler	No issues found.

[1]Full content of one of the reports: System and other apps are using 32,71% of the CPU. This CPU usage may be caused by other tasks that are running on the system or they may be caused by system operations that are triggered by our app.

6. Conclusions and Future Work

Navigation and mobility are key abilities to a comfortable living. Navigating outdoors is, thanks to global systems such as GPS significantly simplified. Indoor navigation remains a less developed field. There are research projects that develop solutions for indoor navigation based on various technologies. One of such projects is NUIP from FEE CTU which combines NaviTerier (path planner and directions generator) with UIProtocol.

We analyzed the state of the art in indoor navigation and compared the accessibility features of the most common smartphone platforms. The reader was also introduced to UIProtocol, its architecture and how it functions.

The goal of the thesis was to implement an UIProtocol client supporting the basic UIP features. The platform of development was chosen to be Windows Phone 8. The client was expected to be able to be used within the NUIP project and consequently, we researched ways of making it accessible to people with special needs. We found, however, that the Windows Phone 8 platform is not very friendly toward visually impaired. Yet, the developed client remains a functional UIP client for the Windows Phone 8 platform.

6.1. Current Features

Within the bachelor thesis we successfully implemented a basic UIProtocol client which implements:

- Basic UI controls
- Binding and model-wide binding
- Graceful degradation of unsupported elements
- Support for interpolated model updates (animations)
- Support for absolute and grid layouts and scrollable containers
- Support for styling (background color, font size and font color)

The features we implemented completely cover the functional requirements we have set in the Design chapter, with the exception of UI generator API support.

The non-functional requirements were also met. The UI loading times are below 0.5 s in all interfaces of the testing application. The execution profiling has shown increased CPU usage in some situations, but when idle, the consumption of phone's resources is normal.

6.2. Accessibility

On Windows Phone 8, the app accessibility could be improved by not redrawing the whole UI when one interface is embedded into other. Also, support for voice commands could be added to UIProtocol. Windows Phone 8.1 brings new accessibility features and ways to provide content to users with special needs - particularly interesting is the Narrator screen reader or the option to increase font size in all apps. UIProtocol can easily facilitate additional information that might be needed by these features, such as alternative text for images.

WP8.1 was released at a late stage of this work and therefore could not be incorporated in it much. We tried to run the developed app on a WP8.1 emulator and explore the new features but the Hyper-V feature required to run the emulator was not supported by our OS version. We therefore could not directly evaluate the new platform. This is a subject of future work, as discussed in the next section.

6.3. Future Work

Implementing the full feature set of an UIProtocol client was not possible in the scope of this work. Therefore there is a number of features by which the client could be extended. In particular the extensions may include:

StackPanel layout panel
: The StackPanel is a simple layout panel that arranges its child elements into a single line that can be oriented horizontally or vertically. Currently, the app supports Grid and Canvas panels to which UI controls can be added. StackPanel should be implemented to widen the choice of layout panels.

More UI controls
: The client comes with a small set of implemented UI controls. New UI controls can be implemented.

Wider support for styling
: The app could support more font styles, such as font family, font decoration (e.g. italic, bold). These features can be easily added.

Performance optimization
: The performance of the current client is satisfactory, it could, however be improved by techniques such as interface pre-fetching.

Error handling
: The client ignores errors received from the server, for example if a requested interface is not found at the server. It also assumes all data it receives is correct.

Support for different screen orientations
: When a user rotates the phone, the UI remains the same. This is not so important for the NUIP project but an UIProtocol client should have this feature implemented.

Support for UI generator API

Screen reader friendliness
: Currently, the implementation redraws the whole UI when interface is embedded into another one. This approach was chosen for its simplicity. It is, however, discouraged by the accessibility guidelines for WP.

Additional accessibility features based on WP8.1
: An accessibility analysis of WP8.1 should follow, assessing the changes that should be made to UIProtocol and the client to take the full advantage of WP8.1.

Appendix A.

Speech Accessibility Features

The following two tables describe the speech commands which can be used in WP8 and iPhone, respectively.

A.1. Windows Phone Speech Commands

Table 10. Windows phone 8 speech commands (from [16])

Operation	Say this
Call someone from your contact list	"Call contact name" (where contact name is the name of someone in your contact list) If the person has only one phone number in your contact card, the call will start. If he or she has multiple phone numbers, you'll see an option to choose one.
Call any phone number	"Call phone number" (where phone number is any phone number, whether it belongs to a contact or not)
Redial the last number	"Redial"
Send a text message	"Text contact name" (where contact name is the name of someone in your contact list). This will start a text message to that person. Then you can dictate and send the message—hands-free.
Call your voicemail	"Call voicemail"
Open an application	"Open application" or "Start application" (where application is the name of any application on your phone, such as "Calendar," "Maps," or "Music")
Search the web	"Find search term" or "Search for search term" (where search term is what you're looking for). If you say "Find local pizza," for example, Bing will bring up a map of nearby pizza places.

A.2. Android Speech Commands

Table 11. Android speech commands (from [23])

Say	Followed by	Examples
"Open"	App name	"Open Gmail"
"Show me my schedule for the weekend."		Say "What does my day look like tomorrow?" to see tomorrow's agenda.
"Create a calendar event"	"Event description" & "day/-date" & "time"	"Create a calendar event: Dinner in San Francisco, Saturday at 7:00PM"
"Listen to TV"	Displays TV cards relevant to the TV show that's currently being broadcast	While a TV show is being broadcast, say "Listen to TV"
"Map of"	Address, name, business name, type of business, or other location	"Map of Golden Gate Park, San Francisco."
"Directions to" or	Address, name, business name, type of business, or other destination	"Directions to 1299 Colusa Avenue, Berkeley, California" or
"Navigate to"		"Navigate to Union Square, San Francisco."
"Post to Google+"	What you want posted to Google+	"Post to Google+ I'm going out of town."
"What's this song?"		When you hear a song, ask "What's this song?"
"Remind me to"	What you want to be reminded about, and when or where	"Remind me to call John at 6PM."
"Go to"	Search string or URL	"Go to Google.com"
"Send email"	"To" & contact name, "Subject" & subject text, "Message" & message text (speak punctuation)	"Send email to Hugh Briss, subject, new shoes, message, I can't wait to show you my new shoes, period."
"Note to self"	Message text	"Note to self: remember the milk"
"Set alarm"	"Time" or "for" & time, such as "10:45 a.m." or "20 minutes from now," "Label" & name of alarm	"Set alarm for 7:45 p.m., label, switch the laundry"
"Call"	The name of one of your contacts	"Call George Smith"

Appendix B.

User Manual

Running the developed solution is simple. Open the WP8uip.sln file in Visual Studio and install in on your Windows Phone 8 device. Alternatively, you can also run it in an emulator.

Run the UIP server and connect to it by providing the client with server's IP address. After connecting, you can browse through the application freely.

B.1. Contents of the Attached CD

```
/
├── src ..... C# sources
└── text ..... the thesis text directory
    └── latex ..... the LaTeX sources
```

Bibliography

[1] David N. Hyerle. *Visual Tools for Constructing Knowledge*. 1996.

[2] *Visual impairment and blindness*. 2013. URL: http://www.who.int/mediacentre/factsheets/fs282/en/ (visited on 05/11/2014).

[3] *An Aging Nation: The Older Population in the United States*. 2014. URL: http://www.census.gov/prod/2014pubs/p25-1140.pdf (visited on 05/11/2014).

[4] *NaviTerier. navigation system in buildings for the visually impaired*. 2013. URL: http://usability.felk.cvut.cz/naviterier (visited on 01/01/2014).

[5] Zdenek Mikovec Jan Balata Miroslav Macik. "Context Sensitive Navigation in Hospitals". In: *30th Annual International IEEE EMBS Conference* (2013).

[6] Miroslav Macik et al. "Platform-Aware Rich-Form Generation for Adaptive Systems through Code-Inspection". In: *Human Factors in Computing and Informatics*. Ed. by Andreas Holzinger et al. Vol. 7946. Lecture Notes in Computer Science. Springer Berlin Heidelberg, 2013, pp. 768–784. ISBN: 978-3-642-39061-6. DOI: 10.1007/978-3-642-39062-3_55. URL: http://dx.doi.org/10.1007/978-3-642-39062-3_55.

[7] *Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2013–2018*. 2014. URL: http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/white_paper_c11-520862.html (visited on 05/14/2014).

[8] *Android and iOS Continue to Dominate the Worldwide Smartphone Market with Android Shipments Just Shy of 800 Million in 2013*. 2014. URL: http://www.idc.com/getdoc.jsp?containerId=prUS24676414 (visited on 05/06/2014).

[9] *Despite a Strong 2013, Worldwide Smartphone Growth Expected to Slow to Single Digits by 2017*. 2014. URL: http://www.idc.com/getdoc.jsp?containerId=prUS24701614 (visited on 05/20/2014).

[10] *Microsoft's Windows Phone 8 finally gets a 'real' Windows core*. 2012. URL: http://www.zdnet.com/blog/microsoft/microsofts-windows-phone-8-finally-gets-a-real-windows-core/12975 (visited on 04/04/2014).

[11] *Windows Phone 8 and Windows 8 platform comparison*. 2012. URL: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj681690(v=vs.105).aspx (visited on 04/04/2014).

[12] *C Sharp Language Specification*. 2002. URL: http://download.microsoft.com/download/a/9/e/a9e229b9-fee5-4c3e-8476-917dee385062/CSharp%20Language%20Specification%20v1.0.doc (visited on 04/01/2014).

[13] *Introduction to the C# Language and the .NET Framework*. 2014. URL: http://msdn.microsoft.com/en-us/library/z1zx9t92.aspx (visited on 05/07/2014).

[14] V. et al. Slovacek. "UIProtocol specification, draft 8". In: *CTU in Prague* (2010).

[15] *Speech for Windows Phone 8. Speech for Windows Phone 8 - development center.* 2014. URL: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206958(v=vs.105).aspx (visited on 01/01/2014).

[16] *Voice commands for Windows Phone 8.* 2014. URL: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj206959(v=vs.105).aspx (visited on 01/04/2014).

[17] *Meeting requirements for accessible text (XAML).* 2014. URL: http://msdn.microsoft.com/en-us/library/windows/apps/hh868163.aspx (visited on 05/06/2014).

[18] *Accessibility on your phone.* 2014. URL: http://www.windowsphone.com/en-ZA/how-to/wp8/settings-and-personalization/accessibility-on-my-phone (visited on 01/04/2014).

[19] *Narrator on Windows Phone 8.1.* 2014. URL: http://www.windowsphone.com/en-us/how-to/wp8/settings-and-personalization/use-narrator-on-my-phone (visited on 05/14/2014).

[20] *Microsoft releases Windows Phone 8.1 SDK Development Tools RC.* 2014. URL: http://www.wpcentral.com/microsoft-releases-windows-phone-81-sdk-development-tools (visited on 05/20/2014).

[21] *Android ends the year on top but Apple scores in key markets.* 2014. URL: http://www.kantarworldpanel.com/Global/News/Android-ends-the-year-on-top-but-Apple-scores-in-key-markets/ (visited on 01/06/2014).

[22] *Developing Android Applications with Voice Recognition Features.* 2014. URL: https://software.intel.com/en-us/articles/developing-android-applications-with-voice-recognition-features (visited on 12/14/2013).

[23] *Google Now.* 2013. URL: http://www.google.com/landing/now/ (visited on 12/14/2013).

[24] *Google Text to Speech.* 2013. URL: https://play.google.com/store/apps/details?id=com.google.android.tts (visited on 12/14/2013).

[25] *Android Accessibility.* 2014. URL: http://developer.android.com/design/patterns/accessibility.html (visited on 01/04/2014).

[26] *Android and iPhone accessibility comparison. Switching to Android full-time – an experiment.* 2014. URL: http://www.marcozehe.de/2013/04/05/switching-to-android-full-time-an-experiment/ (visited on 01/01/2014).

[27] *iOS Accessibility.* 2013. URL: https://www.apple.com/accessibility/ios/ (visited on 12/14/2013).

[28] *iOS VoiceOver.* 2013. URL: https://www.apple.com/accessibility/ios/voiceover/ (visited on 12/14/2013).

[29] Aura Ganz et al. "PERCEPT Indoor Navigation System for the Blind and Visually Impaired: Architecture and Experimentation". In: *Int. J. Telemedicine Appl.* 2012 (Jan. 2012), 19:19–19:19. ISSN: 1687-6415. DOI: 10.1155/2012/894869. URL: http://dx.doi.org/10.1155/2012/894869.

[30] Diego López-de-Ipiña, Tania Lorido, and Unai López. "Blindshopping: Enabling Accessible Shopping for Visually Impaired People Through Mobile Technologies". In: *Proceedings of the 9th International Conference on Toward Useful Services for Elderly and People with Disabilities: Smart Homes and Health Telematics*. ICOST'11. Montreal, Canada: Springer-Verlag, 2011, pp. 266–270. ISBN: 978-3-642-21534-6. URL: http://dl.acm.org/citation.cfm?id=2026187.2026232.

[31] N. A. Giudice T. H. Riehle P. Lichter. "An Indoor Navigation System to Support the Visually Impaired". In: *30th Annual International IEEE EMBS Conference* 30 (2009), pp. 4435–4438.

[32] SYLVIE TREUILLET and ERIC ROYER. "OUTDOOR/INDOOR VISION-BASED LOCALIZATION FOR BLIND PEDESTRIAN NAVIGATION ASSISTANCE". In: *International Journal of Image and Graphics* 10.04 (2010), pp. 481–496. DOI: 10.1142/S0219467810003937. eprint: http://www.worldscientific.com/doi/pdf/10.1142/S0219467810003937. URL: http://www.worldscientific.com/doi/abs/10.1142/S0219467810003937.

[33] B. Ozdenizci et al. "Development of an Indoor Navigation System Using NFC Technology". In: *Information and Computing (ICIC), 2011 Fourth International Conference on*. Apr. 2011, pp. 11–14. DOI: 10.1109/ICIC.2011.53.

[34] Luis A. Guerrero, Francisco Vasquez, and Sergio F. Ochoa. "An Indoor Navigation System for the Visually Impaired". In: *Proceedings of Sensors 2012*. 2012, pp. 8237–8258. URL: http://www.ncbi.nlm.nih.gov/pubmed/22969398.

[35] P. Espinace et al. "Indoor Scene Recognition through Object Detection." In: *Proceedings of the 2010 IEEE International Conference on Robotics and Automation*. 2010, pp. 1406–1413.

[36] A. Quattoni A.; Torralba. "Recognizing Indoor Scenes." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2009, pp. 413–420.

[37] Anna Bosch, Xavier Muñoz, and Robert Martí. "Review: Which is the Best Way to Organize/Classify Images by Content?" In: *Image Vision Comput.* 25.6 (June 2007), pp. 778–791. ISSN: 0262-8856. DOI: 10.1016/j.imavis.2006.07.015. URL: http://dx.doi.org/10.1016/j.imavis.2006.07.015.

[38] J.A. Hesch and S.I. Roumeliotis. "Design and analysis of a portable indoor localization aid for the visually impaired." In: *Int. J. Robot. Res.* (2010), pp. 1400–1415.

[39] *Guidelines for designing accessible apps*. 2014. URL: http://msdn.microsoft.com/en-us/library/windows/apps/hh700407.aspx (visited on 05/04/2014).

[40] *Exposing basic information about UI elements (XAML)*. 2014. URL: http://msdn.microsoft.com/en-us/library/windows/apps/hh868160.aspx (visited on 05/06/2014).

[41] *App monitoring for Windows Phone 8*. 2013. URL: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj215907(v=vs.105).aspx (visited on 05/15/2014).

[42] *App profiling for Windows Phone 8*. 2013. URL: http://msdn.microsoft.com/en-us/library/windowsphone/develop/jj215908(v=vs.105).aspx (visited on 05/15/2014).