bachelor's thesis

Accessible UIP client for Windows Phone 8

Vojtech Novak

May 10, 2014

Macik Miroslav Ing.

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

Acknowledgement

Acknowledgements

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.

Abstract

UIProtocol je jazyk pro specifikaci uživatelských rozhraní vyvíjen na FEL ČVUT pro účely výzkumu. Je navržen jako klient-server systém, kde na serveru běží aplikace a popis jejího uživatelského rozhraní je poskytnut klientu. Klient 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. 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 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
WP8	Windows Phone 8
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

This thesis is intended to build upon the Naviterier UIP project and bring an UIProtocol client application to Wp8 platform. So far, UIP clients are built for Windows, iOS and Android, smart TV emulation for intelligent household.

2. Analysis

Before moving onto the design and implementation chapters of the work, allow us to introduce you to the technology used for development, analysis of UIProtocol, overview of other navigation systems and accessibility analysis of the current mobile platforms.

2.1. About Windows Phone 8

The Windows Phone 8 is the first of Microsoft's mobile platform to use the Windows NT Kernel, which is the same kernel as the one in Windows 8 [1]. 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 [2]. 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 more potential for code reuse.
Also, Windows Phone 8 and windows 8 share the same .NET engine [2]. This is to deliver more stability and performance to the apps, so they can take advantage of multicore processing and improve battery life. Most new devices are now dual or quad-core, and the operating system and apps are expected to be faster because of this technology [2]. 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 [3]. It was developed by Microsoft within its .NET initiative and its first version was released in 2002 [3]. 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 is 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 [4]. The specification called the Common Language Infrastructure (CLI) is an international standard by ISO and ECMA todo which specifies execution and development environment that allows multiple high-level languages to be used on different computer platforms without being rewritten for specific architectures [4]. The CLR is Microsoft's implementation of the CLI standard. Other CLI implementations are Mono [5], DotGNU Portable.NET [6] 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 [4]. 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 therefore can interact with code that was generated from the .NET versions of Visual Basic, Visual C++, or any other CTS-compliant languages [4].

2.2. UIProtocol

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

2.2.1. About UIProtocol

Universal Interface Protocol (UIProtocol) is a user interface specification language [7] being developed at FEE CTU for research purposes. At the time of writing this thesis, 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 [7].
UIProtocol is an application protocol that allows for describing the hierarchical structure of the GUI along with the placement and visual appearance of the containers and components. Implemented are its XML, JSON and binary versions. 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 documents which contain all components and values needed for rendering the UI. The client first initiates the communication and receives respective XML description from the server. The description can be of four different types: interfaces, i.e. the UI components and containers, models which contain the data displayed and actions. The communication from client to server only consists of event descriptions. The architecture of UIProtocol, is shown in figure 1 and it indicates the information flow between client and server.

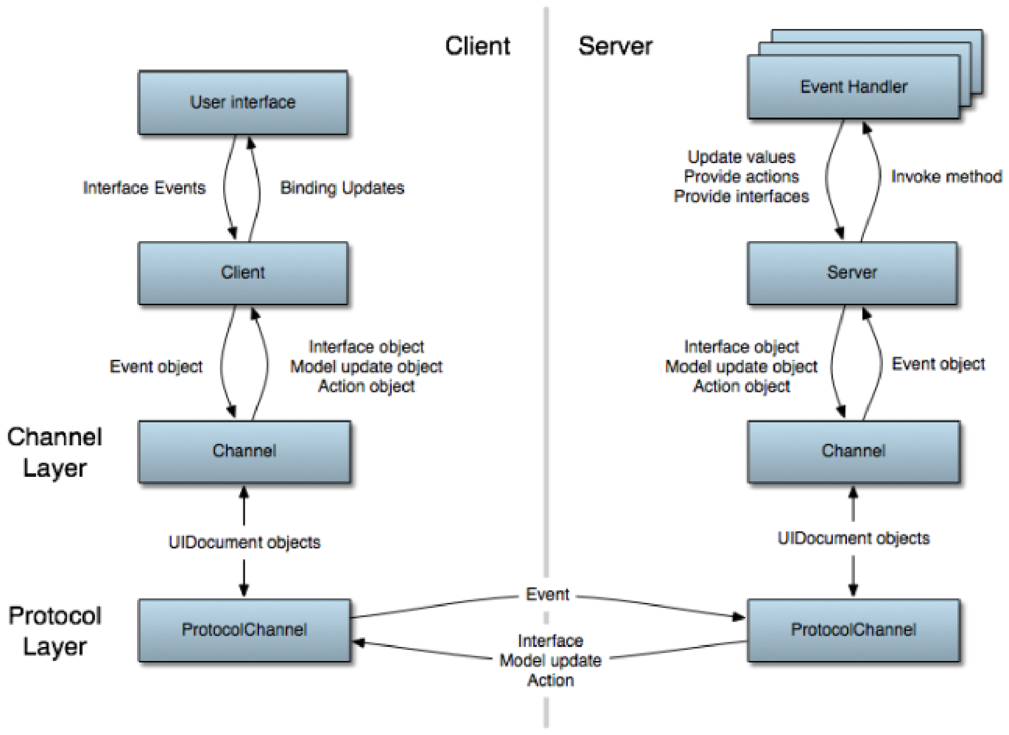

Figure 1. the client-server architecture of UIP

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 handheld 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 is sent to the server. The part done directly at the client are easy tasks, such as visual effects when pressing the button. Even these effects can be, ahead of time, specified by the server. The request for weather is then sent to the server (in a form of event), which processes the information and responds by sending the interface structure, components and the weather information. This is then 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, e.g. server can send updates to the client as soon as the displayed information needs to be updated, without having to wait for an update request. Should an application not communicate with a remote server, 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 [7]. The device running the client is thought to be the one user directly uses, 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 [7]. What has to be implemented is the minimal functionality, e.g. a client that is able to render a user interface, sending event information to the server and update the application it by data coming from it.

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 [7]. It must be able to service multiple client simultaneously and is intended to run on a machine which is considered safe [7].

2.2.2. Syntax of UIProtocol

The UIP document syntax is shown below in Listing 2.1 which shows the four possible tags with 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 actions tag.

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>
```

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

2.2.3. Elements of UIProtocol Communication

As mentioned previously, the information exchange between client and server concerns Interfaces, Models, Actions (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 UIP 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 element into which a user can enter text. It also shows how interfaces can be embedded. This is done by including 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). Note that element tag can have an id attribute, as shown.

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" key="gui:fstName
           "/>
      </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 basic mechanism of client to server communication and therefore has to be supported by client. The event element contains a unique id which specifies the event source. An event can contain any number of properties which describe it more closely. 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>
```

Models

Models serve as the storage for data that is displayed to the user. A model contains data in properties which are uniquely identified by key attribute. The key is separated by colon into two parts - first referencing a model and second referencing a property within the model. When changing the contents of a model, all of the UI elements bound to the model by the key attribute are updated. UIP provides support for this binding. 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, changing the respective property within that model will be immediately reflected.
An important concept in models is the one of data binding. If a UI control text (or any other property) is represented by a UIP property that contains a nonempty key, the appropriate model is requested from server. Upon its arrival, a data binding between the UIP property and the UI element's property must be created. Then, when the client receives an update of the model, the UI element's bound property (text, in this example) gets immediately updated. Models in UIP are application-wide so they can be referred to from any point of the application.

Properties

Properties are the most nested elements of an UIP document and define the visual appearance, positioning of GUI objects and 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 is described by this property. 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 the key attribute which, if set, binds the displayed value to a model. The part before the colon references a model and the part after colon references a property within the model. That is, if there is a model gui with property fstName, the value of the property is used. Moreover, whenever the value of the referenced property is updated by the server, the update is automatically reflected in all bound properties.

2.3. Accessibility of Current Mobile Platforms

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

2.3.1. Windows Phone 8 Accessibility

This section covers all of the features for ease of access that are included in the Windows Phone 8 operating system. For the purposes of this project, we are particularly interested in features that may help disabled users. 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 wide 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 them: 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 content of an SMS [8]. This is very similar to the Voice Command feature, but the key difference is that speech recognition occurs when user is in the app, and Voice Commands occur from outside of the app [8]. 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 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 [9]. 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 7.
A developer can also define her own set of voice commands and allow users not only to open the app using voice but also to carry out more advanced tasks within the app [9]. 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 section.

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 or strings formatted according to the industry-standard Speech Synthesis Markup Language (SSML) Version 1.0 [8]. TTS is also used in some of the other features for ease of access which are covered in the next section.

Other Speech Features A feature named Speech for phone accessibility allows the following [9]:

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 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 build-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. [10]), switch the display theme to high-contrast colors and use the screen magnifier [11]. 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 [11]. 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 [11] or later. For an unknown reason, an update to this version is not available for our phone.

Conclusions

Windows Phone 8 platform offers some features to make its accessibility to disabled users easier. However, there are still gaps to be filled such as the non-existence of a built-in screen reader. Its absence puts both Windows Phone 8 and 8.1 usage out of the question for visually impaired users. There is an limited screen reader option available from version 8.0.10501.127 [11] but this only works with some apps and update to this version is not available to all devices at the time of writing. The platform has recently been experiencing slow growth in some world markets but is stagnating in others [12] and it cannot be estimated how much effort will be put into the development of more accessibility features. 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, with the emphasis on visually impaired users. We will analyze the features of the latest Android OS released, 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

Android also offers the option to interact with the device using speech and has some interesting accessibility features and compared to Windows Phone 8 offers a wider language support. Similarly to Windows Phone, 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 TalkBack. At the time of writing, the speech recognition supports more than 40 languages including several accents of English, and even minor languages such as Czech. Other functions do not have such a wide support.

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 before, this feature supports many languages but on the other hand required internet connection. There is not an option to use the recognition offline. We do not consider this a major drawback, as nowadays a mobile internet connection is more available than ever.

Voice Action Commands In Android, Voice Action Commands are closely related to the Google Now feature. Google Now has a wide range of uses not specifically designed for visually impaired. It can, however, serve them well by allowing them 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 much more. The full list of Voice action Commands is in Table 8.

Note that for some commands, the system gives you a spoken answer. The current drawback of the system is that it only supports English, French, German, Spanish, and Italian. 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 supported only in a limited scope. TTS is also used in TalkBack which is described in the next section.

Other Speech Features TalkBack is an important functionality that strives for more accessible phone control for visually impaired. 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. The blog post of a blind accessibility engineer from Mozilla Foundation [13] 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 font size and the screen magnifier.

Conclusions

Compared to Windows Phone, Android has better accessibility options. It includes 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

In this chapter, we will cover the accessibility of Apple's iOS. Again we consider the latest iOS released, which is version 7.0.4. Overall, the accessibility features of iOS are very similar to those of Android and therefore I will describe the features more briefly.

Speech Features

As with the previous two platforms, iOS also offers users o interact with a device using speech. Apple was the first one to introduce the features for people with disabilities, such as VoiceOver. 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 but does not allow developers to expose their own voice commands.

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 before, 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. 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 It could be said that Google's TalkBack is Apple's VoiceOver. Both offer very similar functions and the reason for existence is reading the content of the screen based on touch input and control of the device by gestures. The mentioned blog post of the blind accessibility engineer from Mozilla Foundation favors VoiceOver over TalkBack [13].

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). iOS devices also support a number of Bluetooth wireless braille displays out of the box. 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.

Conclusions

Apple's iOS was the first to offer advanced accessibility features and the first to become usable for visually impaired users. The accessibility options are comparable with the feature set of Android. iOS allows developers to create apps accessible for a wide range of users.

2.3.4. Comparison of Analyzed Platforms

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

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

Platform	Built-in screen reader	Text to speech	Speech recognition
Windows Phone	no	yes	yes
Android	yes	yes	yes
iPhone	yes	yes	yes

2.4. Navigation Systems Analysis

There is a number of interesting papers in the field of navigation systems for disabled. A large number of them is 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 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 [14] 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 only on mobile phones with voice output, which is a non-problematic 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 current mobile phone voice output. This system does not keep track of the user location. Instead, it breaks the directions into smaller pieces and then sequentially gives the pieces to the user who follows them and asks for next portion when ready. This system was tested with visually impaired users.
Recently this system was combined 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 navigational system called NaviTerier UIP (NUIP) [15] which combines the navigational part of NaviTerier and the ability of UIP to generate and deliver customized user interfaces that can fit better people with disabilities.

2.4.2. PERCEPT

A promising approach is shown in the PERCEPT [16] project. Its architecture consists of the three system components: the Environment, the PERCEPT glove and 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 next part of the environment are the 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 the instructions for the part of the route, repeat previous instructions and get instructions back to the kiosk. Also, after scanning the R-tag the gloves sends its information to the app running on user's Android phone. The 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.

2.4.3. System by Lopez et al.

Another example of RFID use is presented in Lopez et al. [17] where 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.

2.4.4. System by Luis et al.

Luis et al.[18] propose a system which uses an infrared transmitter attached to the white cane combined with Wiimote units (the device of the Wii game console) 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. TTS engine running on the phone converts the directions to speech. The system has undergone preliminary testing with five blindfolded users.

2.4.5. System by Riehle et al.

An indoor navigation system to support the visually impaired is presented in [19]. 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 people.

2.4.6. System by Treuillet at al.

Treuillet and Royer [20] proposed a 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 want to navigate. It follows that the system is not suitable in environments that are visited for the first time. For the case when it has learned the way, the system performs well.

2.4.7. System by Ozdenizci et al.

Authors of [21] 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 or triangulation. 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 able to navigate the user using simple directions.

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 scenes recognition [22], [23], [24]. Some of these solutions allow for creating the 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 [25] is interesting because they integrated these devices (apart from the computing) into a white cane. However, the solution has the limitations in being too heavy and large, and the laser scanner being directional.

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 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.

2.5. 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 disabilities, which is called Guidelines for designing accessible apps and is accessible online at [26]. If a developer follows the principles of accessible design, the application will be accessible to the widest possible audience.

Table 2. Quick comparison of the analyzed navigation systems

System	active components	tested with
NaviTerier	mobile device, kiosk	visually impaired
PERCEPT	RFID, mobile device, kiosk	visually impaired
Lopez et al.	RFID, mobile device, kiosk	n/a
Luis et al.	infrared, Wiimote	blindfolded users
Riehle et al.	UWB tracking system	blindfolded users
Treuillet at al.	camera	n/a
Ozdenizci et al.	NFC, mobile device, kiosk	n/a

Reasons for Developing Accessible Applications

Users of an application may have different kinds of disabilities. By keeping in mind the rules of accessible development, a developer can substantially improve the user experience. Also, let us not forget it 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 a visually impaired users depends heavily on it. Therefore it is important to present it sufficiently and correctly. The provided UI element information describes its name, role, description, state and value [26].

Name

Name is a short descriptive string that the screen reader uses to announce an UI element to the user [26]. 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 how to set or get a name for different XAML UI elements.

The container elements, such as Panels or Grids do not provide their accessible name because it would, in most cases be meaningless [26]. 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 [27]. 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 [27]. Assistive technologies can obtain these values through methods exposed by the control’s AutomationPeer definitions.

Table 3. Accessible Name for Various UI Elements

Element type	Description
Static text	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. See Name from inner text.
Images	The XAML Image element does not have a direct analog to the HTML alt attribute of img and similar elements. To provide a name, AutomationProperties.Name can be used.
Form elements	The accessible name for a form element should be the same as the label that is displayed for that element.
Buttons and links	By default, 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, AutomationProperties.Name can be used to provide a text-only equivalent of the button's intended action.

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 [26]. 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 [26].

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 [26]. An accessible app UI therefore has to scale and change 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 [26].

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 the UI – and must recreate and present the description to the user again [26]. 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 [26].

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 will conclude requirements for the application which will be developed. Further on, the sections will describe 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.
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 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, a screen reader, is not present by default. This holds true even for the Windows Phone 8.1 which was released in February 2014. 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. Apart from following the guidelines for developing accessible apps, as discussed in 2.5, 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 fully functional UIP client capable of displaying valid UIP documents.

Summary of Requirement

From the requirements section, we have developed the following lists of non-functional and functional requirements, respectively.
Non-functional requirements:

- UI components have platform native look
- App will be will be written in C#
- The client should not use much phone resources when idle
- App should be stable and able to process valid UIP documents

- Compatibility with UIP standard, draft 8
- Ability to run on any WP8 device
- UI loading times below 0.5 s 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 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-IP 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 [7], the server sends the XML files describing the UI. The UML diagram of the classes responsible for the communication is shown in figure 2.

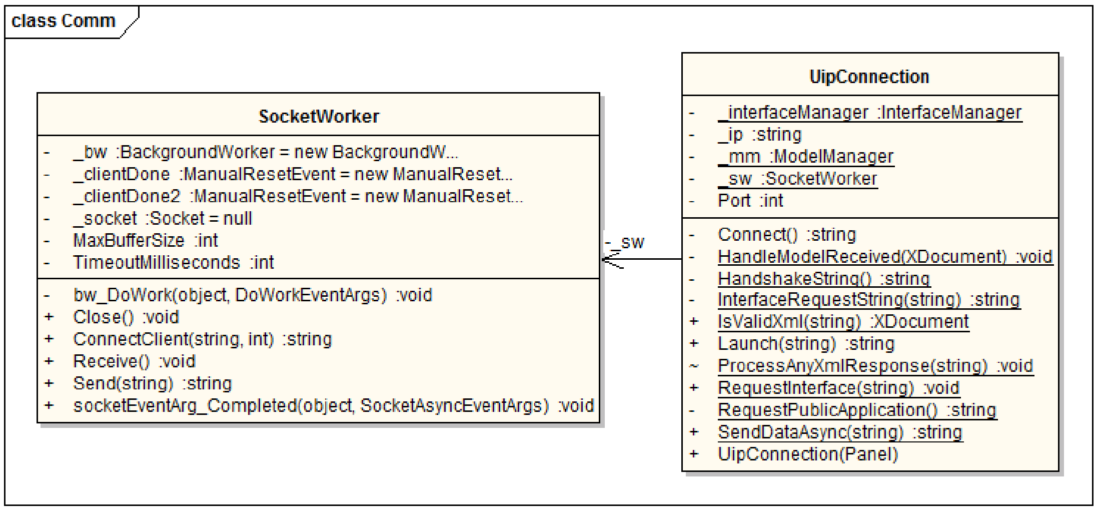
class Comm
SocketWorker
- _bw :BackgroundWorker = new BackgroundW...
- _clientDone :ManualResetEvent = new ManualReset...
- _clientDone2 :ManualResetEvent = new ManualReset...
- _socket :Socket = null
- MaxBufferSize :int
- TimeoutMilliseconds :int
- bw_DoWork(object, DoWorkEventArgs) :void
+ Close() :void
+ ConnectClient(string, int) :string
+ Receive() :void
+ Send(string) :string
+ socketEventArg_Completed(object, SocketAsyncEventArgs) :void
-_sw
UipConnection
- _interfaceManager :InterfaceManager
- _ip :string
- _mm :ModelManager
- _sw :SocketWorker
- Port :int
- Connect() :string
- HandleModelReceived(XDocument) :void
- HandshakeString() :string
- InterfaceRequestString(string) :string
+ IsValidXml(string) :XDocument
+ Launch(string) :string
~ ProcessAnyXmlResponse(string) :void
+ RequestInterface(string) :void
- RequestPublicApplication() :string
+ SendDataAsync(string) :string
+ UipConnection(Panel)

Figure 2. Inheritance tree of several sample UI classes

3.0.3. Parsing XML into Inner Object Representation

When the UIP documents are received, they will be delegated to instances of ModelManager and InterfaceManager classes. ModelManager will be responsible for processing possible new models or model updates and will be discussed later.
The 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.

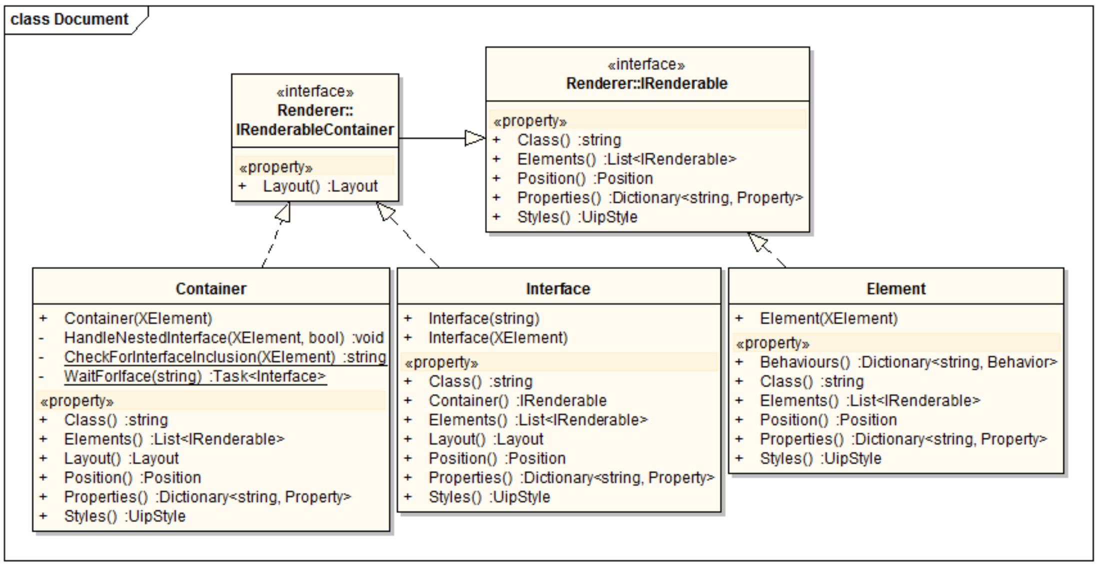
class Document
«interface» Renderer:: IRenderableContainer
«property»
+ Layout() :Layout
«interface» Renderer::IRenderable
«property»
+ Class() :string
+ Elements() :List<IRenderable>
+ Position() :Position
+ Properties() :Dictionary<string, Property>
+ Styles() :UipStyle
Container
+ Container(XElement)
- HandleNestedInterface(XElement, bool) :void
- CheckForInterfaceInclusion(XElement) :string
- WaitForIface(string) :Task<Interface>
«property»
+ Class() :string
+ Elements() :List<IRenderable>
+ Layout() :Layout
+ Position() :Position
+ Properties() :Dictionary<string, Property>
+ Styles() :UipStyle
Interface
+ Interface(string)
+ Interface(XElement)
«property»
+ Class() :string
+ Container() :IRenderable
+ Elements() :List<IRenderable>
+ Layout() :Layout
+ Position() :Position
+ Properties() :Dictionary<string, Property>
+ Styles() :UipStyle
Element
+ Element(XElement)
«property»
+ Behaviours() :Dictionary<string, Behavior>
+ Class() :string
+ Elements() :List<IRenderable>
+ Position() :Position
+ Properties() :Dictionary<string, Property>
+ Styles() :UipStyle

Figure 3. document class diagram

3.0.4. Managing Models

If an UI element has a property that refers to a model, ModelManager will be responsible for requesting the model containing the property. Once it is received, it will be processed by the ModelManager instance which will keep their value stored and will manage future model updates.

3.0.5. Managing Interfaces

3.0.6. Rendering the UI

After the XML description of UI elements is processed, rendering takes place. The Renderer class traverses the tree of the newly created classes from figure todo and calls the
behavior, leyouts, positions, tcp and uipconnection, settings

3.0.7. Representing the Platform-native UI Components

In order to support easy addition of the supported UI elements, we created classes shown in figure todo. There is an abstract base class, UipBase which contains methods for model updates, styling and element dimensions.
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

Events are triggered by user actions - usually clicking on an interactive UI element such as button. The event firing - e.g. notifying server of an action taking place, is a relatively simple process which will be handled by two classes: EventManager: the class's only task is to provide a method for sending an event. The class won't contain any state and will therefore be static, and will only forward the events to SocketWorker class instance. Event class will represent a particular event that will be sent to the server. It will contain all the necessary information for the server to be able to identify the event triggered.

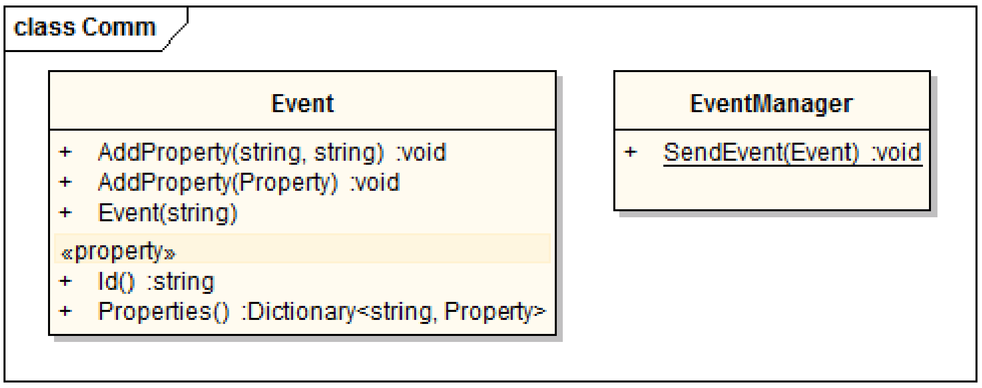
class Comm
Event
+ AddProperty(string, string) :void
+ AddProperty(Property) :void
+ Event(string)
«property»
+ Id() :string
+ Properties() :Dictionary<string, Property>
EventManager
+ SendEvent(Event) :void

Figure 4. event class diagram

3.0.9. Properties

Properties are the most nested objects in UIP documents. They are used extensively within many classes, including Model, Event or Element. They are directly attached to the class instance.

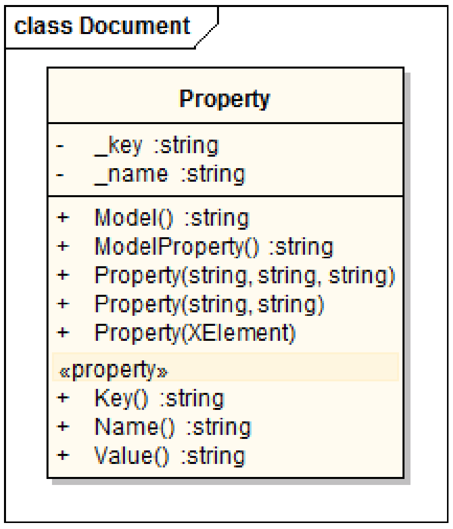

Figure 5. properties class diagram

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 being developed, what technologies were used, what problems were encountered and how they were tackled.

Development Environment

As previously said, the application was written in the Visual Studio 2013 IDE and using C#, a modern 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. ReSharper

Overview of 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, illustrating how inner UIP Document representation is stored (Table 4), how rendering functions (Table 5) and how the communication with the server is handled 6. Apart from the classes mentioned in the tables, there is a number of other helper classes.

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 and can be embedded into another interface, as specified in Listing 2.2.
Container	This is the class that stores the information about the particular UI elements. A Container can contain other Containers and instances of Element class.
Element	Class representing particular UI elements such as button, textfield and more.

Communication With UIP Server

As mentioned in Table TODO, the communication with server in 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 server that is sent to it from a SocketWorker instance.

Table 5. 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.
IRenderable-Container	Extension of IRenderable interface. It also provides support for layouts and is implemented by instances of Interface and Container.

Table 6. 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.

SocketWorker is the low-level socket communication class which ultimately sends events to the server, such as interface requests or events informing about user actions. 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.
The communication was observed from another, already implemented client.

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 code which acquires the models and creates binding between model properties and properties of UI elements makes heavy use of asynchronous methods. The async/await operations were introduced with C# 5.0 and provide a developer with a comfortable way to deal with operations that are potentially blocking. Because requesting and receiving models happens over the internet, it can be considered such. If model request and receive 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 doesn't depend on the web resource until

the potentially blocking task finishes.

Both ModelManager and interface manager classes only need to be instantiated once, so that the application state is stored in one place. Therefore ModelManager and InterfaceManager are singleton classes.

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 grid and etc. Since the model updates are always received as string, the binding has to be provided with a converter (IValueconverter instance) which, considering the given examples, converts the received string to SolidColorBrush, double and integer types, respectively.

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 the UIP document. All supported native components are therefore wrapped into other classes whose names indicate the enclosed UI element (ie UipButton is a wrapper class of standard Button class). All of these wrapper classes inherit from abstract class UipBase which provides a 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. UipContainer, on the other hand, does not implement it because a container itself does not have anything to style - what about background color?? TODO
The figure 6 shows a class diagram of a few wrapper classes and also ITextStylable being implemented.

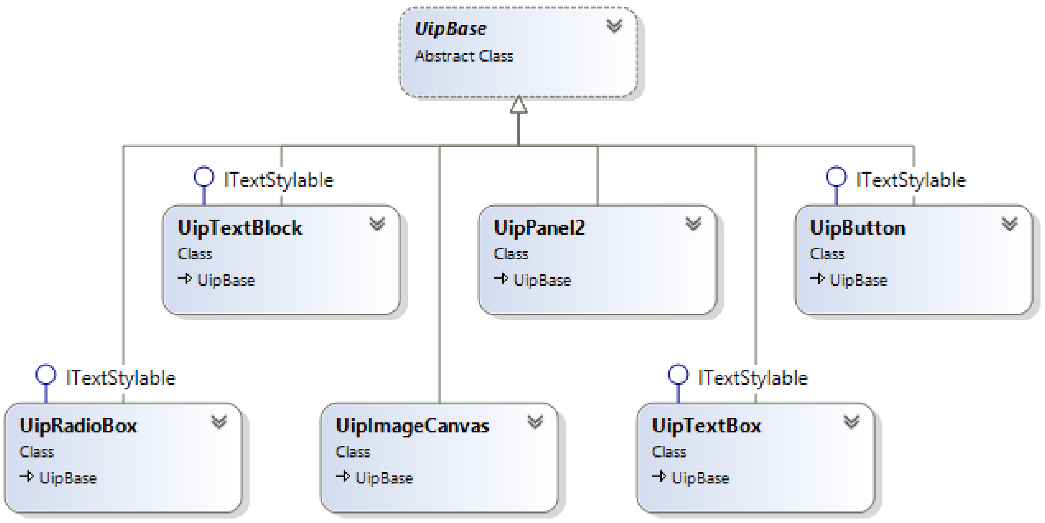

Figure 6. Inheritance tree of several sample UI classes

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 TODO neco jako dan. To illustrate this, let us consider the following example: The server asks the client to render an UIP element of class public.input.choice.single – an UI element known under the WP8 platform as ListPicker (or more generally, a dropdown). 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).

Event Communication

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

5. Testing

The last task we need to fulfill is to show that the client is able to process UIP documents. In order to do so, we created a testing application.

splneni pozadavku, testing application unit testy novy chapter conclusions / conclusion and future work slovnicek pojmu a pak *ewew* prefetching

Testing was primarily done by directly running the developed solution on the provided windows phone device (HTC 8S). Visual Studio offers means for running and debugging the application which allow for fast development. We developed a testing app written in UIP XML language which was then run by the developed client. This application covered the testing of switching interfaces, including them, models and their updates, animations and events.

To widen the coverage of testing we also added unit tests which should detect bugs that were not discovered while running the testing application. The unit tests cover the most important classes of the project.

conclusions

Appendix A.
Speech Accessibility Features

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

Table 7. Windows Phone 8 Speech Commands

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.

Table 8. Android Speech Commands

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"
"Listen to"	Play music in the Google Play Music app by speaking the name of a song, artist, or album	"Listen to: Smells Like Teen Spirit"
"Call"	The name of one of your contacts	"Call George Smith"

Bibliography

[1] *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).

[2] *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).

[3] *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).

[4] *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).

[5] *Mono Framework website. Speech for Windows Phone 8 - development center.* 2014. URL: http://www.mono-project.com/Main_Page (visited on 05/01/2014).

[6] *DotGNU Portable.NET.* 2014. URL: http://www.gnu.org/software/dotgnu/pnet.html (visited on 05/07/2014).

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

[8] *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).

[9] *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).

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

[11] *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).

[12] *iPhone market share shrinks as Android, Windows Phone grow.* 2014. URL: http://news.cnet.com/8301-13579_3-57616679-37/iphone-market-share-shrinks-as-android-windows-phone-grow/ (visited on 01/06/2014).

[13] *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).

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

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

[16] 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`.

[17] 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`.

[18] 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`.

[19] 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.

[20] 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`.

[21] 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`.

[22] 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.

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

[24] 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`.

[25] 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.

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

[27] *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).