
User Interface Generator

Ing. Miroslav Macík

PhD program: Computer Science and Engineering
Supervisor: Prof. Ing. Pavel Slavík, CSc.
January 2011
This research has been done within project Automatically generated user interfaces in nomadic applications founded by grant no. SGS10/290/OHK3/3T/13 (FIS 10-802900) (Studentská grantová soutěž) and has been partially supported by the MSMT under the research program MSM 6840770014. This research has been partially supported by the MSMT under the research program LC-06008 (Center for Computer Graphics).
Abstract

Although there were multiple attempts to use automatic user interface generation, none of them was widely practically used. From the developer’s perspective, the main issues are the overall complexity of input to the user interface generators (input models), platform dependency, and corresponding poor scalability. From the user’s perspective, the automatically generated user interfaces are usually problematically accepted. Main problems are the low consistency of these interfaces, especially the poor consistency with user’s knowledge and habits. Automatically generated user interfaces then often look in effect too artificial and over-complicated.

This document describes an extensive survey of the state of art in the field of automatic user interface generation, as well as a design and implementation of a system which has a potential to resolve, at least partially, mentioned problems. We simplified the input models on both the system level and the immediate input to the user interface generator. Furthermore, we propose a solution that accommodates combinatoric user interface optimisation, template application and consistency optimisation in one tool. This, potentially increases consistency and naturalness of automatically generated user interfaces. The proposed design provides a maximal scalability: It is based on UIProtocol, a platform independent language for describing user interfaces and communication.

The solution proposed in this text is demonstrated on an exploratory implementation. The system is able to automatically generate user interfaces for various target user interface platforms and user’s properties and preferences for a given abstract description of the desired functionality of the user interface.

Keywords

Automatic User Interface Generation, User Interface Consistency, Model-Based User Interface Design, User Interface Design Patterns
1. Introduction and Motivation

The information society has been forming for a few tens of years. During this process more and more people were involved in process of using electronic information and communication technologies (ICT). Each device that needs to communicate with its users requires some kind of user interface. Nowadays the majority of people on the Earth use some kind of complex electronic device (e.g. computer, cellular phone). According to [1], the number of mobile phone connections surpassed the threshold of five billions in 2010. Consequently, there are billions of different people with different needs requiring more and more different electronic devices. One of key aspects of the human computer interaction is the fact, that a single user interface cannot fit needs of all potential users (see [2]). This is the main motivation for automatic user interface generation because of it is not possible to develop personalised user interface for each combination of user, interaction device and corresponding application manually. The most sophisticated approaches like SUPPLE [3] take into account the usage context of the application, user and corresponding interaction device.

During the history of user interface generation were made many more or less successful attempts to satisfy various requirements by the automatic user interface generation. So far, user interfaces for most applications need to be generated manually, although usually with help of powerful tools like layout managers. There are some domains where the automatic user interface generation is used massively. For example automatic form generation for databases is used for years (for example in Microsoft Access [4] or tools like phpMyAdmin[5]).

Some modern applications are not static and therefore can dynamically change over time. For example a set of devices connected into an intelligent household [6] (or smart home [7]) can vary over the time. This makes another demand for the automatic user interface generation and motivates multiple projects, for example Personal Universal Controller [8], Uniform [9].

The recent massive grow of complex electronic devices in count, complexity and variability changed the traditional comprehension of applications as device-based. Parts of modern applications, or more precisely, parts of their user interfaces can accommodate the user’s environment in order to maximise usability and overall user experience. This is a key aspect of so-called ubiquitous computing (see [10]). When an application can move with its users across different contexts (different devices, different position, through a dynamically developing environment etc.), it can be tagged as “nomadic” (see [11]). These new aspects of application development set new demands to automatic user interface generation. To design user interfaces manually for all the possible combinations of users, devices and applications is humanly unmaintainable, apart from not all the necessary information is available in the design time.

Currently, there are multiple issues that prevent the automatic user interface generation from massive commercial usage. From the point of view of application developers the most burning problem is the usual overall complexity of abstract models that are used as input for the process of user interface generation and corresponding lack of development tools (see [12]). More importantly, the target users see the problems directly in the generated user interfaces. Automatically generated user interfaces are often inconsistent with the concepts the user is familiar with. This is usually a result of optimisation that overestimates certain technical aspects of user interface structure (for example element size, grouping etc.). Consequently, the key problem is to find an optimisation criteria that estimates the usability of user interface (or more importantly the quality of user experience with the user interface) correctly. User interfaces
automatically generated by a machine also often look too artificial, this is due to lack of an artistic value, which is present in well designed man-made user interfaces.

1.1. Motivation

There are more aspects that motivate the automatic user interface generation, most of them arise from problems described above. Generally this approach has potential to save development costs and even to solve problems that are otherwise humanly unmaintainable. For example, class of problems where part of necessary information is not known in the design time can be generally solved only by automatic user interface generation.

According to literature, automatic user interface generation is crucial for addressing requirements that are set on modern application.

“Pervasive computation cannot succeed if every computational device must be accompanied by its own interactive hardware and software. Diverse populations cannot be served by an architecture that requires a uniquely programmed solution for every combination of service and interactive platform.” (from [2])

Moreover, the conventional tools used for accessibility extract the necessary information from the state-of-the-art user interfaces (however sometimes designed with accessibility in mind). This complicates the process and consequently leads to worse user experience of impaired people. On the other hand, automatic user interface approach can provide user interfaces personalised directly to address various needs of different individuals, even impaired people.

1.1.1. Use-case scenario

Purpose of the following scenario is to illustrate potential use of the proposed system.

Igor’s career grows steeply since he graduated from his university, at 40 he works as head of analysis department of an international auditing company. Today an ophthalmologist visit in the Motol hospital is planned. Igor got the invitation by e-mail. Besides usual things, this time the message contained a link to an application, that can be used for communication with the doctor and navigation inside the hospital. Igor installed the application into his cellular phone and using a simple wizard he set-up his personal profile.

Although Igor is very busy at work, he knows, that health is the first and leaves the office for the doctor visit. Unfortunately, he got stuck in a traffic jam, he uses the application to inform the doctor. In this way is his appointment postponed by 30 minutes.

At the moment he entered the hospital, he notices clearly marked terminal. He identifies himself by QR code on his printed invitation. System shows the shortest way to the doctors office and notifies Igor to hurry. Because of the system knows Igor’s preferences, this user interface is personalised according Igor’s worsening sight. Furthermore Igor is informed that he can use simple navigation terminals on the way by using QR code on the invitation or on display of his mobile phone.

This is not Igor’s first time in the hospital, so he made it quickly to the waiting room. There is a big screen that says that estimated waiting time for Igor is 5 minutes. After this short while Igor is asked to go into the doctors office.

Although Igor didn’t anticipated serious problems, the diagnosis isn’t good. Igor’s intraocular pressure is so high, that he must be immediately operated. Otherwise a durable damage of his
retina and eye-nerve will be unavoidable. Igor did not expect such variant, before his anaesthesia he managed just to send a mail to his office to postpone his agenda.

As soon as Igor woke up, he realised that he is not able to see anymore. Fortunately, a nurse told him, that this is because of bandage on his head and that the operation went well. The expected time of return of normal sight is two weeks. Igor’s user profile was updated in order to enable him to control systems in his room. Since then the system knows that Igor is not able to use his eyesight anymore. In the first few days Igor used his voice to control systems in the room, in front of all the air-conditioning, radio and to adjusting his hospital bed. The available commands are presented to Igor via synthesised voice in order of their importance. First the system notified Igor about the presence of emergency-button and about that he can use also a voice command to call the nurse.

During the recovery Igor’s eyesight gets better and better. After a few days is he able to use an interactive TV in his room to control things that he was able to control solely via voice at the beginning. Considering his so-far not very good sight, the user interface consists of elements with maximal size, of course at the expense of complex structure on multiple screens. At the same moment as Igor’s eyesight gets better and better, grows the amount of services and devices Igor can interact with. Igor sometimes use his cellular phone to control the system. For example to adjust temperature in his room while on a walk.

After two weeks Igor leaves the hospital, although he is supposed to use eye-drops for some time, he is glad that everything went well and his eye-sight is much better than before the operation. After return to his office, user interfaces in his work-environment are updated according this now fact. This makes Igor more effective, now he can has more time for his beloved family.

1.2. Problem definition

Aim of this section is to formally define the problem of automatic user interface generation. Assume that various input models (task model, workflow model, dialog model etc. - see bellow) are already transformed into an abstract user interface (AUI). AUI is a structure that describes what should be presented to the user instead of how (more details are in chapter 3.1.9 and 4.1.2).
The process of automatic user interface generation shown in Figure 1 is unambiguous mapping of concrete user interface elements to their abstract variants (each abstract element can be typically represented by multiple concrete elements). Inseparable part of this process is layout of the mapped concrete elements to fit contextual requirements. The automatic user interface generation can be called “context sensitive” if it considers the context (see bellow). Context contains most importantly information about the target platform, target user and environment. Information about target platform is for example the modality used, screen resolution, available elements, their size etc. Important information about the target user are most importantly his abilities like eyesight or hearing, cognitive abilities, physical properties or preferences like favourite colour or even metadata about previous experience with the application.

1.3. General Aims

This section shows the high-level aims for this work - user interface generator. Type of this work determines the necessity of scientific contribution, therefore the detailed requirements can be determined only upon extensive analysis of the state of the art in this research area (chapter 3). The general requirements are:

- General platform-independent approach.
- Support for ubiquitous computing and nomadic applications.
- Look of automatically generated user interfaces should be as natural as possible.
- User interface generator will be integrated to a complex platform, which contains a workflow engine and module for advanced user modelling.

1.4. Structure of this thesis

This thesis consist of following parts:

In chapter 1 - Introduction the background, motivation and general aims of this thesis are shown.

Chapter 2 - Basic Definitions summaries the terminology and shortcuts used in this thesis.

Chapter 3 - Previous Work is a detailed analysis of state of the art in the field of automatic user interface generation. The most important projects in this area are discussed. Namely, Uniform, iCrafter and Supple have brought important ideas of consistency, combinatoric optimisation and template application that are further developed by this thesis. Additionally, chapter 3 contains a brief summary of another tools that are relevant for our approach of automatic user interface generation.

Chapter 4 - Proposed solution consist of two different parts. The first part describes tools and environment that has been developed to accommodate automatic user interface generator – UiGE. Firstly, UIProtocol – an innovative protocol that can handle both user interface description and communication is introduced. Secondly, UIProtocol clients – platform dependent program solutions that enables using UIProtocol on various platforms are shown. Lastly, the complex UiWP platform overview is proposed, this platform integrates advanced user modelling, workflow based interaction design and automatic user interface generation.

The second part of chapter 4 described the UiGE user interface generator itself. UiGE is main focus of this thesis, therefore the conceptual requirements are clearly summarised on the beginning of this section (4.3.1). Follows a conceptual design that endeavours to resolve the requirements:
• Section 4.3.2 describes how the complexity of input models is lowered and how the user and device model is clearly separated.
• Section 4.3.3 propose an idea of generalised mapping, that solves the problem on how to accommodate combinatoric optimisation, templates and consistency in a single design.
• Section 4.3.4 contains specification of requirements on the optimisation cost function in section. Detailed specification of this function is subject of future work.
• Section 4.3.5 shows how user interface generator and its environment deals with requirements of generality and platform independency.
• Section 4.3.6 describes how is the internationalisation and localisation supported by the design.

Remaining sections of chapter 4 describe the architecture of UiGE (4.3.7), the user interface generation pipeline (4.3.8) and how the user interface design patterns are applied to the automatically generated user interfaces (4.3.9).

Chapter 5 shows the intermediate results. Concretely the status of implementation of particular components and screenshots of user interfaces on three different platforms that were automatically generated by the UiGE.

Chapter 6 - Evaluation of Results justifies the reason why an extensive evaluation was not performed yet. Furthermore it provides list of recommendations for the future evaluation.

Chapter 7 - Conclusion and Future Work - summarises the state of the art achievements of the proposed solution and formulates what is the subject of future work.

Remaining chapters contains the Bibliography, list of author’s publications and an abstract and proposed topics of his dissertation thesis.
2. Basic Definitions

This chapter contains basic definitions and shortcuts used in this work. As specific places (in front of all in the state of the art analysis - chapter 3) are some proprietary terms explained at the same place.

2.1. Definitions

<table>
<thead>
<tr>
<th>item</th>
<th>definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>User Model</td>
<td>Is a set of user’s characteristics relevant for user interface generation (preferred language, quality of eyesight etc.)</td>
</tr>
<tr>
<td>Task Model</td>
<td>A hierarchical structure describing how to perform activities to reach users' goals.</td>
</tr>
<tr>
<td>Workflow</td>
<td>A sequence of connected steps in order to achieve a goal.</td>
</tr>
<tr>
<td>Layout manager</td>
<td>Program component responsible for automatic or semi-automatic placement of user interface components</td>
</tr>
<tr>
<td>Design pattern</td>
<td>Generic reusable solutions to commonly occurring problems is some domain.</td>
</tr>
<tr>
<td>Computationally hard problem</td>
<td>Problem that contains to NP-hard class of problems. There is no knows solution in polynomial time for problems of this class.</td>
</tr>
<tr>
<td>Accessibility</td>
<td>term describing the level how much a product, device, service, or environment is available to as many people as possible (including people with special needs)</td>
</tr>
<tr>
<td>Usability</td>
<td>Is ease of use and learnability of a user interface.</td>
</tr>
<tr>
<td>User Experience (UX)</td>
<td>A person’s perceptions and responses that result from the use and/or anticipated use of a product, system or service [28]</td>
</tr>
<tr>
<td>Localisation</td>
<td>addition of special feature to the product to be used is specific locale [29]</td>
</tr>
<tr>
<td>Internationalisation</td>
<td>adaption of product for potential use everywhere (support of multiple locales) [29].</td>
</tr>
<tr>
<td>Internal consistency</td>
<td>Measure of internal uniformity of an user interface.</td>
</tr>
<tr>
<td>External consistency</td>
<td>Measure of uniformity with user interfaces an concepts the user is familiar with.</td>
</tr>
</tbody>
</table>

Table 1: Basic definitions
## 2.2. Shortcuts

<table>
<thead>
<tr>
<th>shortcut</th>
<th>meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>AUI</td>
<td>Abstract User Interface</td>
</tr>
<tr>
<td>CUI</td>
<td>Concrete User Interface</td>
</tr>
<tr>
<td>UiGE</td>
<td>User Interface Generator (also name)</td>
</tr>
<tr>
<td>UiWP</td>
<td>User Interface and Workflow Platform</td>
</tr>
<tr>
<td>XML</td>
<td>eXtensible Markup Language</td>
</tr>
<tr>
<td>i18n</td>
<td>internationalisation</td>
</tr>
<tr>
<td>l10n</td>
<td>localisation</td>
</tr>
<tr>
<td>a11y</td>
<td>accessibility</td>
</tr>
</tbody>
</table>

*Table 2: Shortcuts*
3. Previous Work

Aim of this chapter is to summarise the state of the art effort in the field of automatic user interface generation. The focus is not only on the most sophisticated approaches like SUPPLE [3], Uniform [9] or iCrafter [13], but also on pioneer projects that introduced the basic ideas and terminology like Maser Mind [14], Mobi-D [15], XIML [16] or XWeb [2]. Furthermore other tools, standards and approaches like UIML [17] or XAML [18], which are relevant to this work, are discussed.

3.1. User interface generation based on abstract models

The most successful approaches are based on some kind of formal abstract model. Often some kind of task or workflow model is used as a suitable model behind the automatic user interface generation. As proposed, a general solution should address specific properties of different applications, users, user interface platforms and also varying context of usage. The main input for process of user interface generation is usually an abstract user interface (AUI). Basically, AUI describes what should be presented to the user (e.g. five pieces of text information and two action triggers) instead of how (text information can be mapped to label or pronounced by a text reader, trigger can be represented by a button or menu item). Systems based on other models usually use an AUI as an explicit or implicit internal representation.

Furthermore, in general approaches is always used some kind of abstract user model. This model is used for describing properties, habits and needs of a user. Another model describes properties of user interface device used for interaction - device model.

3.1.1. Master Mind

As described in [14], MASTERMIND is one of first systems that integrate multiple models together. As shown in Figure 2, an application model, task model and presentation model are used in the design-time environment. Proprietary notations for all models were used in MASTERMIND. This makes this system rather good example of early model-based toll than reusable approach.

*Figure 2: Architecture of MASTERMIND environment (from [14])*
3.1.2. Mobi-D

This system described in [15] provides assistance in development process rather than automatic design. Interface and application developers still involved in the development process. Mobi-D is a set of tools to support development cycle. There are several clearly defined models: user-task model, domain model, dialog model and presentation model. Relations among these models are also explicitly defined.

Development process starts by deriving user tasks, starting by informal description converted to an outline. The next step is definition of user-tasks and the domain. Skeleton domain model is derived from task outline. Both domain and user model are refined by a developer. The framework provide explicit methods for generalising pieces of models to be reused in other designs. Final step is design of presentation (user interface) and dialog. Decision support tools provide recommendations in order to help developer to build the final interface.

This system provides recommendations that do not limit flexibility, but make the development process more organised. Mobi-D is supposed to be used by usability engineers rather than by standard developers. The models for more complex interactions require significant effort to be developed. For purposes of this work is important an idea of explicit separation of models. Furthermore reusability of pieces of models can lower required development effort of any system and provide more consistent results.

3.1.3. XIML

XIML stands for eXtensible Interface Markup Language [16] and is based on Mobi-D approach. This system use well described notation of abstract user interfaces (AUI) on the input and produces concrete user interfaces for various platforms (CUI). Unlike in the more sophisticated approaches, the mapping between AUI and CUI elements is explicitly determined by the designer. This is an interesting idea that was reused in our work to build so called initial concrete user interface that is input for the process of optimisation.

3.1.4. XWeb

This system described in [2] tries to apply the web metaphor to services in general in order to support higher levels of interactivity. The main motivation is to enable creators of services to abstract from interactive technologies and devices. Although neither XWeb was massively practically used, it brings an important idea of platform-based thin client. This is actually generalised web approach that becomes more and more adopted by Web 2.0 technologies like Google docs. The idea of moving as much logic to the server and supply the user with as thin platform-specific client as possible was used as main motivation for developing UIProtocol (see chapter 4.1, [19]).

3.1.5. Personal Universal Controller

The main motivation for Personal Universal Controller (PUC) [8] project was to simplify controlling of appliances by using single device with richer interface that is able to control more appliances at once. Mobile devices like PDA’s or Smart phones were used as the main controllers.
In Figure 3 is depicted the architecture of PUC showing one connection between PUC device and an appliance. This complex system allows controlling various appliances by various controlling devices (PUC devices). PUC use its own proprietary appliance-oriented langue for describing abstract user interfaces. For generation of concrete user interfaces PUC uses simple rule-based approach (e.g. command is always represented by button). Furthermore PUC is able to generate speech user interfaces. It is subject of feature work to include similar feature into our solution, especially because the better accessibility of resulting user interfaces.

### 3.1.6. Uniform

The Uniform [9] is another promising user interface generator that has been brought into public in 2006. The main contribution is that the consistency of final user interface is taken into account. It means that the system is trying to find similarities between currently generated user interfaces and interfaces that have been presented to the user in past. The final look of the user interface is therefore adapted to be as consistent with current user interfaces as possible.
Figure 4 shows an example of user interfaces generated by UNIFORM. The first two images show the user interfaces of two independent copy machines. On the second two images there are depicted user interfaces rendered to be consistent with copy machine A or B respectively.

3.1.7. ICrafter

ICrafter [9] is a framework for services and their user interfaces in a class of ubiquitous computing environments. Authors refers the ubiquitous computing environment as an interactive workspace. The aim is to enable users to interact with services in this environment using various modalities and input devices. ICrafter provides user interface generation and so called on-the-fly aggregation of services. According to the authors, a service is either a device (e.g. light, projector or scanner) or an application (e.g. web browser or Power point running somewhere in the environment).

**Figure 5: ICrafter Architecture (from [13])**

In Figure 5 architecture of the ICrafter framework is depicted. Appliances request user interfaces from interface manager, the request contains appliance description. Interface manager first selects appropriate UI generator (software entities that can generate UI for one or more services for a particular appliance). The appropriate generator is selected in the following order: generator for the service instance, generator for the service interface and finally service independent generator as a fallback. In the next step the selected generator is executed with access to service descriptions, appliance description and the context. Using this information, the appropriate generator constructs the final user interface.

Unlike other systems ICrafter use specific UIGenerators for particular services and user interface languages (target platforms). Most of them are implemented using a template system.

An interesting idea in ICrafter is usage of so-called service patterns. Patterns are recognised in the available services. ICrafter then generates user interfaces for these patterns. On one hand, this leads to better consistency and easier service aggregation. On the other hand, unique functionality is not available in the aggregated service. Another contribution is involvement of template system in user interface generators. For most cases the user interface designed by
human designer is better than automatically generated equivalent (exceptions are for example user interfaces generated for users with specific needs).

3.1.8. User interface generation based on source code annotation

This system approach has been developed at the Czech Technical University and described in [20, 21]. This approach uses annotated source code as an input for the user interface generation. The motivation was to remove the necessity to manipulate with abstract models explicitly, but keep the flexibility they provide.

In [21] the abstract user interface (AUI) is described as a set of following elements: text input, number input, single item selection, multiple item selection, monitoring, responding to alerts and other (specific) elements. This definition is sufficient for most applications, for the purposes of this work is used slightly more generic approach based on UIProtocol [19].

A simple approach for generation of concrete user interfaces was used - an explicit platform dependent mapping of concrete widgets to abstract user interface elements. Furthermore simple vertical layout was used.

This system is designed to support ubiquitous computing (in this case an interactive workspace). There is a key idea of services (devices or applications) that are used as an input for the process of user interface generation. Generation of the concrete user interface depends on the selected generator. Usually the generator is platform and service specific but a simple automatic approach is possible.

3.1.9. Supple

Supple [3] is user interface generator introduced in 2004. It is based on functional description of a user interface and takes into account both device capabilities and usage patterns. It specifies the user interface generation process as an optimisation problem. The basic idea on how to generate a pseudo-optimal user interfaces, is to minimise estimated user's effort while interaction with the generated UI. The SUPPLE user interfaces are based on a functional specification of an interface, device model and user model.

3.1.9.1. Functional Interface Specification

According to [3], a functional representation of the user interface says which functionality should be exposed to the user, instead of how. As proposed this specification can by called as an abstract user interface (AUI). It is a hierarchical representation of functionality that a user interface should expose to the user. In SUPPLE authors define an AUI as a composite of a set of interface elements and a set of interface constraints specified by the designer. The AUI elements can be one of the following types:

- primitive types (int, float, string and bool),
- container types (used to create groups of simpler elements),
- constrained types (primitive or container types extended by a set of constraints),
- vectors (ordered sequence of zero or more primitive types - are used to support multiple selection),
- actions (elements used for invoking application methods).
The mapping (red arrows) depicted in Figure 6 shows relation between functional representation and a sample concrete user interface. Notice primitive types (Light Level - int, Power - bool, Screen - bool, Vent - int), container types (Classroom, Light Bank, A/V, and Projector), constrained types (Light Level $\langle\text{int, [0,10]}\rangle$, Vent $\langle\text{int, [0,3]}\rangle$), and vector type ($\text{Input } \langle\text{string \{data1, data2, video\}}\rangle$). The action type is not present in this example.

### 3.1.9.2. Device Capabilities

In SUPPLE a display-based device is defined as a composite of following sets:

- set of available widgets,
- set of device-specific constraints.

It is straightforward that the UIGenerator must deal with the problem how to describe the controlling device. The set of supported widgets and parameters of the device as screen resolution have to be taken into account as well. Additionally, the UIGenerator will need to evaluate suitability of particular widgets in a generated user interface.

SUPPLE says that for a good user interface is it crucial to minimise the estimated user effort needed to manage work with the interface. Consequently, the metric of suitability of particular widgets must correspond to the estimated effort the user will have to deal with them.

### 3.1.9.3. User model

Authors of SUPPLE declare that the best rendering depends on how a user will use the final user interface. Users with different needs (in particular disabled people) may disagree on the best graphical rendering. SUPPLE designers believe that a good estimate of the effort requires a usage model. They chose so-called user traces about the source of information of usage of an interface. A user trace is a set of trails, where the term trail refers to sequences of elements manipulated by the user.

For purposes of this work, this way of user modelling could provide an inspiration. On the other hand, we are sceptical to benefits of frequent dynamic updates of user interfaces. This can lead
rather to confusion and inconsistency that to better usability. In Supple this problem is solved by special user interface that tests the user capabilities before the first real customised user interface is rendered. This idea can be used also in this work for generating the initial user models automatically.

3.1.9.4. Rendering and optimisation algorithm

Because of the choosing of appropriate mapping between abstract and concrete elements is a computationally hard problem (see [3], section 7.4) (very common to the knapsack problem) the optimisation could be a time demanding task. In SUPPLE they adapt the branch and bounds method and get satisfactory results. It has been proven that the final algorithm is very effective, following design choices are used to prune the space of possible states during the optimisation:

- the admissible heuristic which helps prune provably suboptimal solutions,
- the constraint propagation strategy which helps eliminate provably illegal solutions,
- the variable and value ordering heuristics which speeds up the optimisation by processing the most constrained elements first.

![Figure 7: SUPPLE optimisation algorithm combining optimisation and constraint satisfaction mechanisms (from [3])](image)

In Figure 7 is shown the optimisation algorithm used in SUPPLE. The variables correspond to elements of abstract user interface (functional specification). Values correspond to the set of available widgets (concrete user interface elements) that can be mapped to a particular variable. Constraints include both interface and device constraints. The solution is in each step stored in bestRendition. Authors of SUPPLE conclude that this algorithm is very effective and can provide final concrete user interface instantly (i.e. in ones of seconds) for most cases.

3.1.9.5. Cost function

Currently in SUPPLE are used two different cost functions. First function enables fast computation of admissible heuristic (see above) and enables to generate different styles of final user interfaces based on different parameters. Using this function it is possible to generate user interface addressing user’s subjective preferences. The second function reflects the time that a particular user needs to perform a typical set of tasks with a particular user interface. This second function leads to optimising the speed of use based on the objective motor abilities of the user.
3.1.9.6. Summary

SUPPLE is yet probably the most sophisticated approach for automatic user interface generation. It considers both user and device model and use well defined metrics as the optimisation criteria.

3.1.10. Experimental user interface generator for URC

Master’s thesis described is [22] aimed to adopt automatic user interface generation for purposes of intelligent household ([23]). The implicit abstract user interfaces was based on explicit specifications of available appliances. The system was based on the Universal Remote Console (URC) approach (see [23]). There was only target device rendering the user interfaces - an interactive TV.

![User interface generated by experimental user interface generator](image)

**Figure 8:** User interface generated by experimental user interface generator (from [22])

In Figure 8 is an example of user interface generated by user interface generator described in [22]. Concrete user interfaces were optimised to the available screen size using simulated evolution.

3.1.11. Summary

We summarised several projects that aimed to solve various problem by using automatic user interface generation.

- **Master Mind** is one of first systems that used model based approach. Although this idea is important, the system is yet obsolete and use proprietary notation that were not further adopted.
- **Mobi-D** is rather a tool to support development cycle. Important is a good definition of several models (task model, user model, domain model).
- **XIML** is based on Mobi-D and bring a good example of specification of abstract user interfaces (AUI’s). User interface generation is limited to explicit mapping of concrete user interface elements to abstract user interface elements.
- **Personal Universal Controller** is intended for controlling appliances. It brings another proprietary language for describing abstract use interfaces. For generation of concrete
user interfaces is uses a simple rule-based explicit mapping (similarly to XIML). Important is another output modality - speech user interfaces. This makes this system very strong in terms of accessibility.

- **iCrafter** is designed to support ubiquitous computing (in this case an interactive workspace). There is a key idea of services (devices or applications) that are used as an input for the process of user interface generation. Generation of the concrete user interface depends on the selected generator. Usually the generator is platform and service specific but a simple automatic approach is possible.

- **Uniform** is based on Personal Universal Controller approach but furthermore brings an important idea of consistency. User interfaces for new appliances are generated in order to be consistent to user interfaces the user already uses and is familiar with.

- **Supple** is from discussed approaches the most sophisticated in terms of automatic user interface generation. The generation of concrete user interfaces is formally described as combinatoric optimisation problem and the optimisation criterion is the estimated user effort to deal with the resulting user interface. In Supple authors proved that the proposed optimisation is computationally hard problem, variant of branch and bounds method is used to solve the problem in satisfactory times. User model is considered during the optimisation process. In Supple, a graphical user interface (GUI) is the only output yet supported, which limits the system in terms of multi-modality and accessibility.

Supple is yet the most promising approach, therefore this work follows the idea of describing user interface generation as optimisation problem. Although Supple and the other discussed projects provided great ideas, none of them is widely used for commercial purposes yet. We see following problems that currently limit the usability of the discussed approaches:

- Overall complexity of the abstract models used as input for the process of user interface generation. Lack of tools for these models.
- Some properties specified in user model are often bound to a particular user interface device (e.g. minimum font size the user is able to recognise in pixels).
- Many platforms are limited to a specific environment (e.g. Personal Universal Controller or UNIFORM).
- Some systems are limited to a specific output platform.
- Discussed approaches do not provide generic support for internationalization and localisation.
- Discussed approaches do not provide alternative content to be used as label for the abstract elements (for some users, typically impaired ones, is good to use some kind of long description (e.g. visually impaired) or icon (illiterate)).
- Mapping of concrete elements to their abstract variants is done explicitly or use some kind of optimisation metrics. Neither approach considers at the same time suitability of particular concrete elements, internal consistency (within generated user interfaces), external consistency (consistency to user interfaces to user already knows) and usage of user interface design patterns. Supple almost fulfils this point but does not support user interface design patterns and support for consistency is not yet well defined.
- Neither approach considers importance of particular elements that can be defined in the structure of abstract user interface. This criteria can be crucial for increasing usability of complex user interfaces, especially while used by people with special needs. The most important elements can be rendered on the top level in convenient form and with appropriate size, whereas less important elements can be rendered deeper in the structure using space-saving concrete forms or to be skipped at all.
Although Supple propose two methods how to estimate user effort to deal with generated user interface, we see further research potential in this field. In front of all, findings of modern psychology should be involved into this process in order to maximise not only performance and usability of the resulting user interface but more importantly also the overall user experience.

In order to fulfil requirements to scientific contribution, our approach should address as many of proposed problems on the conceptual level.

### 3.2. Other related approaches and tools

This section is to summarises tools and approaches relevant to user interface generation. Aim of tools and approaches described bellow is not to generate concrete user interfaces, but they can be used in various stages of this process. They can be either used to describe user interfaces (UIML [17], XAML [18]). Furthermore, approach how to use design patterns in scope os user interface design is shown.

#### 3.2.1. UIML

UIML [17] is a platform independent XML based user interface description language. The main goal of UIML is to save development costs by using a single language instead of many (WML, HTML, XUL). Similarly to HTML the layout of elements is defined in styles. This enables platform independent rendering but in comparison to full automatic user interface generation there are limitations (for example screen size - the user interface is not automatically divided to parts to fit the screen it the most convenient way).

UIML partially inspirited UIProtocol (see chapter 4.1), which has similar capabilities but the overall structure is simpler. Unlike UIML, UIProtocol can describe both AUI and pixel perfect CUI in the same way.

#### 3.2.2. XAML

XAML [18] stands for The Extensible Application Markup Language. It is a declarative XML-based language created by Microsoft. XAML is widely used in .net framework since version 3.0. The Windows Presentation Foundation is tightly connected to XAML notation.

From perspective of this work, XAML is not a sufficient language for platform independent user interface description. Firstly, it is tightly connected to Microsoft® technologies (.net, Silverlight) and their internal object structure (see bellow). Secondly the language is quite complex and can not be easily extended without adding new elements or using namespaces.

This language has an explicit XML variant that is used for initialising the runtime object structure. This object structure in maintainable from the code behind. This object representation is final product of UIProtocol (see chapter 4.1) rendering on various .net based clients (see chapter 4.1.1).

#### 3.2.3. Patterns in model based user interface generation

This chapter is based on work of Sinning at al. in [24]. Authors took inspiration from the famous design patterns for object oriented programming (published in [25]). Generally design patterns are generic reusable solutions to commonly occurring problems is some domain. In the HCI domain, the usage of patterns has potential to increase overall consistency, to save development
costs and when applied correctly, to lead to better usability of the final user interfaces. In this work authors proposed three basic types of patterns: dialog patterns, presentation patterns and layout patterns. From perspective of this work, the second two types are relevant. Presentation patterns can be used in order to map complex tasks (such as advances search, or login) to a predefined set of interaction elements. The layout of concrete elements is defined by layout patterns. According authors, pattern application consists generally form the following steps:

- Identification - reducing the domain size to a subset relevant in particular step.
- Selection - the most appropriate pattern is selected.
- Adaption - the abstract pattern is instantiated and adjusted to the context of use.
- Integration - the pattern instance is integrated back to the overall solution.

Contribution of user interface patterns to automatic user interface generation is in front of all in usage of well-proved man-made designs when possible. Consequently it potentially leads to better consistency and usability of resulting user interfaces.
4. Proposed Solution

This chapter summarises the current state of development of UIProtocol user interface generator (UiGE). Firstly, is the current version of UIProtocol introduced. Furthermore, this chapter focusses on improvements of UIProtocol to support automatic user interface generation. Secondly, a complex development platform (for purposes of this work called User Interface and Workflow Platform - UiWP) is introduced. This platform incorporates, among others, the UiGE, Torsion Workflow Server and User Modelling Server (see bellow). Architecture of this platform and integration of UiGE is also scope of this chapter.

The main focus is on the UiGE user interface generator. In context of the UiWP, the purpose of the UiGE is to generate final concrete user interfaces (CUI’s) from a workflow model transformed by the Torsion Workflow Server into an abstract user interface. This chapter summarises key conceptual ideas and approaches behind the design of UiGE. Firstly, the main goals for UiGE are summarised in detail. Secondly, concepts, ideas and algorithms to fulfil these goals are introduced. Last but not least, the architecture of UiGE and design of its main components is introduced.

4.1. UIProtocol

UIProtocol\(^1\) [19] has been designed to provide effective solution for Client-Server applications including heterogeneous client platforms and to support so called ubiquitous computing (see [10]). The main features of UIProtocol are listed bellow:

- XML syntax,
- MVC approach – clear separation of presentation, model and application logic,
- direct support for data binding (see [26]),
- language independent,
- implementation of the complete protocol specification is not necessary for simple client. This feature ensures scalability,
- extensible without modifying the protocol specification,
- precise visual definition of elements if necessary,
- animation support.

\[\text{Figure 9: Traditional (a) and UIProtocol (b) Client-Server Architecture}\]

---

\(^1\) Author of this work co-operated on UIProtocol specification with estimated concern about 15% within framework of study period of his Ph.D. study
In a classical Client-Server architecture (see Figure 9 (a)) is the application logic divided between various client platforms and a common server part. There are also various user interface description languages. This requires additional effort from developers to deal with multiple heterogeneous platforms. In case of UIProtocol (Figure 9 (b)), there is common application logic on the server, the user interface can be delivered to any client platform. Advanced approach is necessary to adapt the user interface to specific features of this platform. This saves development costs, makes possible to transfer application across different platforms (ubiquitous computing [10], nomadic applications [11]) and also makes the application easily maintainable.

**Figure 10: UIProtocol communication**

UIProtocol is not only user interface description language, but also defines the communication between client(s) and the server. As shown in Figure 10, only events that are later handled by server side application logic are propagated in the direction from client to server. In the opposite direction the server can push to client interfaces (description of user interface structure) and models (usually data that are represented to the user through the user interface). As soon as a model is updated on the server, this update is propagated to all (or selected) clients and instantly affects the user interface using data binding [26].

### 4.1.1. UIProtocol clients

UIProtocol supports various client platforms\(^2\), namely we already have client implementations in Java, .net (TV, desktop, multitouch-screen, iOS - using cross compilation), HTML (PHP proxy, ASP proxy), Adobe Flash and Microsoft Silverlight. The UIProtocol specification (see [19]) supports wide range of clients of different capabilities. The user interface can be delivered on complex clients that support animations or complex user interface elements but in the same way it can be rendered on simple clients like HTML web interface. This is made possible by implementing so called graceful degradation (for more details see [19]). The other important feature is data binding (see [26]), that ensures instant propagation of updates on all clients. Development of client application for a new technology requires significant effort, but the total costs are lower because of there is a common server side application logic and also user interfaces can be used for virally all clients.

---

\(^2\) Author of this work implemented various .net based UIProtocol clients. Within framework of his Ph.D. study period, there were designed and implemented three automatic user interface generation ready clients.
For purposes of UiGE there have been developed three specific .net based clients. As depicted in Figure 11, all these clients use common client core that contains as much common logic as possible (it is responsible for communication, storing user interfaces and models, transform interface structure into internal object model and to provide support like logging). Platform specific logic is implemented in particular platform clients (currently .net WPF client and iOS client). Moreover, these technologies supports various specific user interface platforms (e.g. desktop, touchscreen for .net or iPhone or iPad for iOS). To address this issue the clients are extensible by using different so-called user interface element sets. Each client provides API that supports adding or uprating element sets without need of recompilation.

4.1.2. Abstract user interface

UIProtocol was originally designed to describe concrete user interfaces and communication. It has been developed in order to be as simple as possible. Consequently its structure is universal enough to describe abstract user interfaces (AUI's). Abstract user interface is a functional specification of an application and declares what should be represented to the user instead of specifying how. Therefore the AUI is usually the main input for the automatic user interface generation. Specification of UIProtocol for abstract user interfaces\(^3\) can be found in early draft specification of Torsion Workflow System (see [27]). An abstract user interface can contain one of the following basic element types:

- **container** – an element containing child elements used for grouping elements into a hierarchy,
- **display** – an element that is used to display an information and is not intended for interaction,
- **trigger** – an element that causes a triggered interaction (or action) to be executed,
- **input** – an input that asks a user for data.

Types listed above are base types. Each of them can be specified more concretely. For example input.email or input.date specify that the awaited value is email or date respectively. Torsion Workflow System contains explicit importance of each interaction (see [27]) as an essential part.

---

\(^3\) Specification of UIProtocol for abstract user interfaces is result of the team work on the design of UiWP. Estimated authors contribution is 35%.
The importance is reflected in abstract user interface elements as their importance property. One of key aspects of this work is to reflect importance in the resulting generated concrete user interface. Notice this property specified for elements on structure depicted in Figure 12. Importance of elements higher in the structure (here container and interface *.user.root) is maximum from particular importances of child elements.

Figure 12: Example of structure abstract user interface

Figure 12 shows an example abstract user interface. Notice that it is not specified how the element should be represented in the final user interface. Furthermore each element contains title, description and icon properties. These properties can be used by the user interface generator for representing the element in a best way for a particular user (it makes no sense to show text title to people who can not read, description can be used for visually impaired people in speech user interfaces etc.). Notice also, that the importance property that can have value in a range from 0.0 to 1.0 (0.0 means redundant element, 1.0 means critically important element). Importance value can be finally reflected in the concrete user interfaces (less important elements can be rendered with less user-friendly but space-saving widgets (e.g. combo-box) or to be skipped at all). Concrete form of abstract user interface represented in UIProtocol can be found in Appendix A.

4.1.3. Concrete user interface

In contrast to abstract user interface described above, a concrete user interface is a structure that can be directly rendered on a particular UIProtocol client. Notice that all elements in example interface depicted in Appendix B have a concrete class (public.text - “label” or public.button - “button”) and a fixed position. Additionally, there are platform and user specific properties like font.color or font.height.

Concrete user interfaces are the output of UiGE. Concrete user interface depicted in Figure 13 was generated automatically and corresponds to abstract user interface depicted in Figure 12 and in Appendix A. Additionally, Appendix B shows UIProtocol representation of concrete user interface shown in Figure 13 generated for the .net touchscreen client.
Figure 13: Screenshot of rendered concrete user interface. The interface was rendered upon structure depicted in Appendix B

4.2. The UiW Platform

UiGE is a component of a UiWP\textsuperscript{4} that aims to address complex goals for model based user interface generation. This section shows the context of UiGE in the overall platform overview. As shown in Figure 14, UiGE is tightly connected to UIProtocol Server Core. This core is used for communication with Torsion Workflow Server and throughout UIProtocol Server consequently with particular UIProtocol clients.

As proposed, UIProtocol Server communicates with UIProtocol clients using UIProtocol (basics of UIProtocol communication are described in chapter 4.1 above). For communication between UIProtocol server and Torsion Workflow Server a specific variant of UIProtocol is used, the main properties and differences are:

- Events from UIProtocol clients are propagated into Torsion Workflow Server.
- UiGE can request an abstract user interface from Torsion Workflow Server similarly like UIProtocol client requests a concrete user interface from UIProtocol server.
- Torsion Workflow Server can push models and abstract user interfaces into UIProtocol server core (and UiGE consequently).
- Model updates from Torsion Workflow Server are instantly propagated to all connected clients.
- Pushed abstract user interfaces are stored for purposes of concrete user interface generation (in future version will be pre-processed in order to minimise time necessary for on-the-fly generation of corresponding concrete user interface).
- Models in Torsion Workflow Server are synchronised with models in UIProtocol server using special events (Torsion can refuse to update its internal model).

\textsuperscript{4} Author of this work contributed on the design of UiWP platform. His estimated concern is 35%.
4.3. UiGE - User interface generator

Input of user interface generator are abstract user interfaces and its output concrete user interfaces. The AUI’s are usually product of the Torsion Workflow Server. The concrete user interfaces are optimised according to context. The context specifies needs and preferences of particular user and the corresponding user interface platform.

4.3.1. Requirements

Requirements to the user interface generator result from problems proposed in introduction (chapter 1) and more importantly from issues emerging from analysis of state of the art technologies (chapter 3). Follows a list of requirements that should be covered our user interface generator:

1) Decrease complexity of abstract models used as input for automatic user interface generation.
2) Provide good separation of user and device model.
3) Combine, support for UI consistency, templates and combinatoric optimisation in the same design.

Torsion Workflow Server further communicates with User Model Server - a system module responsible for user analysis and modelling. It stores definitions of user models and updates them upon analysis of their behaviour while interacting with the system. Updates of the user model are propagated back to Torsion Workflow Server and consequently into UiGE. The communication between all components is handled by UIProtocol.
4) Reflect importance of particular abstract user interface elements in the resulting concrete user interfaces.

5) Better reflect usability factor in the optimisation criterion (this is a research topic with high potential).

6) Support wide range of output platforms.

7) Provide generic design that is not limited to a specific environment.

8) Seamless localisation and internationalization by design.

Requirements listed above stand for high-level conceptual design requirements. It is obvious that there are additional requirements determined by the platform and its purpose. Follows conceptual design of the UiGE. The main challenge is to address requirements listed above while keeping the complexity of the design as low as possible.

4.3.2. Input models

One of biggest drawbacks of model based user interface generation is the overall complexity of input models. Because of the design of the Torsion Workflow Server is beyond focus of this work, the main focus is on the immediate input for the user interface generator - abstract user interfaces, user model and device model.

Structure of an abstract user interface has been already described above (see chapter 4.1.2). It adopts the UIProtocol notation and is maximally consistent to the notation of concrete user interfaces. This fact maximises simplicity of design and consequently lowers the barrier for designers that are already familiar with the convectional (concrete user interface) approach. As soon as the system is provided with the abstract user interface it can provide concrete user interfaces for the whole variety of corresponding concrete user interface platforms automatically.

The crucial requirement to user model is to provide information for adjusting the user interface to needs and preferences of a particular user. This model should reflect in front of all user's physical and psychical abilities. In order to keep the design as generic as possible, values in this model should not be bound to any particular platform. Therefore we decided to use relative values.

The device model should reflect properties of a particular user interface device. For the process of user interface generation the most important properties are for example the screen size, list of supported elements or possible modalities. Additionally this model comes with standard values of user-bound properties (values for a typical user interacting with the particular device). In this case, user-bound device properties have absolute values.

![Figure 15: Computation of user-bound property](image)
Figure 15 shows computation of final value to be used. The motivation is to separate properties of a particular user in user model and user-bound device properties. This enables to separate model of individual user and device.

<table>
<thead>
<tr>
<th>combination</th>
<th>resulting value</th>
<th>user model</th>
<th>device model</th>
</tr>
</thead>
<tbody>
<tr>
<td>user 1, device 1</td>
<td>min. font size = 22px</td>
<td>font size = 1.0</td>
<td>font size = 22px</td>
</tr>
<tr>
<td>user 2, device 1</td>
<td>min. font size = 33px</td>
<td>font size = 1.5</td>
<td>font size = 22px</td>
</tr>
<tr>
<td>user 3, device 1</td>
<td>min. font size = 11px</td>
<td>font size = 0.5</td>
<td>font size = 22px</td>
</tr>
<tr>
<td>user 1, device 2</td>
<td>min. font size = 10px</td>
<td>font size = 1.0</td>
<td>font size = 10px</td>
</tr>
<tr>
<td>user 2, device 2</td>
<td>min. font size = 15px</td>
<td>font size = 1.5</td>
<td>font size = 10px</td>
</tr>
<tr>
<td>user 3, device 2</td>
<td>min. font size = 5px</td>
<td>font size = 0.5</td>
<td>font size = 10px</td>
</tr>
</tbody>
</table>

*Table 3: Example computation of a user-bound property*

Table 3 illustrates situation on example of minimum font size property, that reflects the minimum size of font the user able to recognise comfortably. Instead of using absolute values in pixels or millimetres, relative values are used in the user model. In the example in Table 3, User 1 has normal-sight and therefore the value is set to 1.0, User 2 has slightly worse sight - relative value is set to 1.5, the vision of User 3 is outstanding, therefore the relative value is 0.5. Device 1 is an example of touch screen panel, the standard minimal font size estimated for population with normal sight is in this case 22px. Device 2 is a smart phone with estimated standard minimum font size of 10px. Additionally, Table 3 shows particular values to be used for all possible combinations of user and a device. The resulting values are computed as a product of user model relative value and device model absolute value.

We assume that these properties of the design covers requirement no. 2 (decrease complexity of input models).

### 4.3.3. Generalised Mapping and Optimisation

This section describes process of mapping. Mapping is an assignment of one or more concrete user interface elements to one abstract user interface element. Particular abstract element can be represented by different concrete elements (e.g. input.date can be represented by a text-box, set of combo-boxes or a complex dialog etc.). The mapping is a subject to a number of restrictive rules. The rules can be expressed as follows:

1) A set of possible representations of abstract user interface element on a given user interface platform is given by a fixed list (list of explicit mappings)
2) The list is iterated and each item is evaluated whether or not it is legal in the given context.
3) Each mapping computes its cost value (optimisation criterion).
Process of application of mapping rules is depicted in Figure 16. Available mappings are assigned to corresponding abstract user interface element and pruned according current context. Purpose of following optimisation is to find globally (for a whole user interface) optimal variant.

In order to keep the design of system maximally transparent and to fulfil requirement no. 3 (support combinatoric optimisation, templates and consistency) at the same time, three different types of mapping have been defined - mapping to concrete elements, mapping to templates and virtual mapping. Implementation of each mapping is responsible for computation of corresponding value of cost function (optimisation criterion). Furthermore, each mapping decides about its legality in the particular context. Follows detailed description of each type.

The first type is depicted in Figure 17. It represents the basic mapping between abstract element and its concrete representation. Mapping is specified explicitly and the implementation is responsible for determining legality of this mapping. This is important because only a subset of possible mappings is applicable in a particular context (e.g. corresponding UIProtocol client does not support mapped concrete element or button representation makes no sense for user, who is able to control the system solely via voice).

The second type of mapping is shown in Figure 18. It is quite similar to concrete mapping but in this case the mapped abstract element is more complex (typically container, exceptionally whole interface). This determines that the mapped structure is also complex, in this context it is called a template. Instead of one or two concrete elements it could for example represent a whole login dialog. Consequently, well-applied templates increase the consistency of the user interface and potentially improve the user experience by including well-designed man-made structures in an automatic design. Templates, that are well-defined, well-documented and adopted by the community can be called user interface design patterns [24].
The last type of mapping is virtual mapping shown in Figure 19. Notice that there is no mapped concrete structure. The aim of this mapping is to affect the cost value of particular abstract element. Unlike other types of mapping, in this case can to a single abstract element apply multiple virtual mappings at the same time. The cost value depends on some particular criterion that is usually recognised in the already assigned concrete structure (element(s) or temperate). In general, it depends on the whole structure of user interface and also on context (device, user or environment). For example virtual mapping can determine internal consistency of an user interface or whether particular structure is consistent with state of the art experience of the user (external consistency).

Proposed solution is called generalised mapping because of it is not limited to simple concrete structure, but it additionally supports templates and virtual mapping. Combinatoric optimisation takes place as soon as all possible mappings are determined and the optimal one should be chosen. Because of the limitations of target user interface platforms (most importantly screen size) it is typically not possible to render user interface using the minimal cost mapping for each abstract element (locally optimal). The aim of the optimisation is to find legal solution with globally minimal value of total cost function.

As proposed each mapping determines a cost value that is used for finding globally optimal solution. Suppose, that the importance of particular abstract user interface elements is not equal. This is the reason why in our design the cost function is further weighted by this importance. This ensures that critical elements are rendered with the best available concrete structure, while non critical (or redundant) elements are rendered with less space demanding representation or skipped at all.

Design ideas introduced in this section address important requirements no. 3 (combinatoric approach, templates and consistency) and 4 (considering importance of AUI elements). Follows an introduction of ideas behind computation of the optimisation criterion - cost function.

### 4.3.4. Optimisation criterion - cost function

This section aims to describe basic properties of the cost function used for optimisation. It is beyond scope of this work to provide exact formulation of cost function, therefore only overall requirements are specified. Similarly to Supple [3], the optimisation criterion should correspond to overall user’s effort of dealing with the particular concrete user interface. The criterion should also reflect importance of particular abstract user interface elements. Value of the criterion must
be explicitly computable for each abstract element with fixed mapping and consequently for the whole user interface. Generally this function depends on

- the particular concrete mapping,
- user context,
- device context,
- environment context,
- abstract element importance
- using virtual mapping,
  - internal consistency of user interface,
  - external consistency of user interface.

Because only the basic requirements on the const function has been defined yet and in the implementation are used ad-hoc values instead, we can not mark requirement no. 5 as resolved. It is probably technically unfeasible to estimate the user effort or even his experience with a particular user interface exactly, therefore we assume that a separate user-cantered research is necessary to define this criterion to be as confident as possible.

4.3.5. Generic environment

This section highlights properties of the UiGE design (or UIProtocol itself) that ensures versatility of the approach. First of all there is a wide range of different output platforms. The UiGE should enable, upon an abstract user interface, generation of concrete user interfaces for this whole variety. The idea is to follow design recommendation: “Provide a reasonable default but allow for fine tuning” (see [13]). This is ensured for output platforms that implements the minimal set of concrete elements as defined in UIProtocol specification [19]. There is specified a “fallback” mapping for each abstract element ensuring that the user interface can be rendered on a output platform that has no other mapping specified.

The UiWP platform components are only loosely-coupled, furthermore, UIProtocol is used for communication between all components. This ensures the platform-independence by the design (actually UiGE, UIProtocol server is implemented in .net, Torsion Workflow Server and User Interface Server in Java and there is whole variety platforms of UIProtocol clients). Consequently the design-level reliability is higher because of each component can be seamlessly backed-up. We assume that features of the design mentioned above address requirement no. 7 (support for wide range of platforms) and 8 (provide environment-independent design).

4.3.6. Localisation and internationalisation

Support for localisation and internationalisation is another aspect that must be addressed by an universal approach. Terms internationalisation (i18n) and localisation (l10n) are explained in Table 1. From the point of view of system design, there is a clear difference between i18n and l10n. To support i18n, the system must provide means for implementation of applications that will operate in an international environment (especially support for multiple languages). Additionally, l10n requires means to adapt an application to a particular locale (support for different units, at best support for cultural differences).

Internationalisation is in UIProtocol supported by design using model variants (see specification [19]). As shown in Figure 20, element properties to be internationalised are bound to i18n model properties. This ensures separation of language-specific constants from the user interface specification. A change of the language of user interface is done by setting particular model
variant. Using the binding, the user interface is updated instantly (for more details see chapter 4.1 above).

**Figure 20: i18n model binding and model variants**

Figure 21 shows an example of using i18n model for internationalisation. Property “text” of a label is bound to the property “name” in the i18n model. At the same time, the selected variant of the i18n model is czech (“cs”). The data binding ensures that in the user interface will be used the right value “Jméno”.

**Figure 21: Example of i18n model binding**

The UIProtocol implementation of i18n was described above. In complex environment of UiWP the model variant can be selected automatically by using information from user model. Table 4 bellow shows a fragment of user model with language-specific properties. These properties specify the user’s knowledge of particular languages. In this case czech language (“cs”) can be chosen automatically.

<table>
<thead>
<tr>
<th>property</th>
<th>example value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>language.cs</td>
<td>0.9</td>
<td>relative language knowledge:</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 1.0 - university professor of</td>
</tr>
<tr>
<td></td>
<td></td>
<td>literature in particular</td>
</tr>
<tr>
<td></td>
<td></td>
<td>language</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• 0.0 - foreigner who does not</td>
</tr>
<tr>
<td></td>
<td></td>
<td>know neither script nor</td>
</tr>
<tr>
<td></td>
<td></td>
<td>words of the language</td>
</tr>
<tr>
<td>language.en</td>
<td>0.6</td>
<td></td>
</tr>
<tr>
<td>language.de</td>
<td>0.3</td>
<td></td>
</tr>
</tbody>
</table>

**Table 4: Fragment of user model**

There are two possible strategies how to layout elements in an internationalised automatically generated user interface. If there is expected a frequent change of the language, the elements of
user interface are layouted to fit for each language variant. Otherwise, the layout can be performed for the selected language variant, which requires less space.

The localisation can be consequently more complex. For example we want different layout for different cultures. The user interface generator deals with this problem by implementing location-specific mappings. Application of such mapping depends on the particular location, therefore dramatic differences in the final look of generated concrete user interface are possible.

This chapter summarised platform support for internationalization and localisation, even in complex cases. We assume that requirement no. 8 (localisation and internationalisation) as resolved.

4.3.7. UiGE Architecture

This section shows the architecture of UiGE on the high-level. It is beyond scope of this work to show implementation details, therefore only most-important components are described. Figure 22 shows important components of UiGE and also corresponding components of UIProtocol server.

![Figure 22: Components of UiGE](image)

In UiGE, the abstract interface builder and provider is responsible for maintaining abstract user interfaces. It transforms the UIProtocol representation of an abstract user interface into an internal object form. This form is on request passed to the process of optimisation.

Before process of optimisation is resolved the possible mapping. For each abstract element is determined set of legal mappings. This mappings are in the UiGE specified explicitly. Component for providing available mappings is mapping provider. Applicability of mapping in the particular context depends among others on available concrete elements. Similarly to mapping, concrete elements are specified explicitly and maintained by the concrete element provider.

Abstract user interface with resolved mapping is optimised according the total cost function by optimiser. Each abstract user interface with resoled mapping ca be rendered to a concrete representation using methods implemented in assigned explicit mappings and concrete elements.
To support the process of user interface generation, the UIProtocol server exposes necessary components to the UiGE. Most importantly, *model manager* maintains all available UIProtocol models including representation of context models like user, device or environment. Availability of these models is crucial for filtering of mappings, cost function computation and optimisation. *Interface manager* maintains available concrete user interfaces, that can be requested by UIProtocol clients. If a concrete user interface is not available in UIProtocol server, it asks UiGE to try to generate the interface automatically. In case of success, the UiGE provides the UIProtocol server with the automatically generated user interface which can be send to the corresponding UIProtocol client.

This section described functionality of main components of UiGE and UIProtocol server that are important for automatic user interface generation. Follows a brief explanation of the automatic user interface generation process.

### 4.3.8. User Interface Generation Pipeline

Generation pipeline depicted in Figure 23 describes steps of process of final *concrete user interface* (5) generation. The input is an *abstract user interface* (1) - structure described in chapter 2.2 above. Usually each element of abstract user interface can be mapped to multiple concrete elements (public.trigger can be for example represented as a button, menu item or eventually keyboard shortcut). All these *available mappings* (3) are explicitly specified within the UiGE. Explicit mapping is in detail described in chapter 4.3.3 above.

*Available mappings* (3) are filtered (a) in order to select *legal mappings* (4) corresponding to the *current context* (2) (target, client, environment). *Cost value* (4) of each mapping is also determined. In the following step (b), mappings for particular abstract user interface elements are ordered according the value of cost function. Upon this mapping, an initial concrete user interface can be generated. If it is legal (fulfils user interface platform limitations), it is also optimal solution. Otherwise it must be *optimised* (c) in order to find *globally optimal solution* (5). At each step of optimisation chat be checked the *legality* (d) of actual user interface.

**Figure 23:** Concrete user interface generation pipeline

This section summarised process of user interface generation. In Appendix E is this algorithm depicted using pseudo-code representation. The process of combinatoric user interface *optimisation* (c) is not implemented yet. Following chapter shows basic concepts behind it.

### 4.3.9. Concept of optimisation

The aim of optimisation is to minimise estimated user effort necessary to deal with the resulting concrete user interface. The target platform limits the possible combinations (for example a
complex user interface does not fit into the screen). Finding a globally optimal solution is a computationally hard problem (see [3]). There are multiple methods how to solve problems of this class. For a constrained problem like this, it is usually sufficient branch and bound method (see [3]), that finds the optimal solution. For complex cases it is necessary use a faster method, that finds pseudo-optimal solution in a sufficient time. Example of such method is an artificial evolution (evolutionary algorithm - EA) or simulated annealing (see [22]).

The optimisation has not been implemented yet, we plan to use branch and bound method in front of other methods that finds a pseudo-optimal solution. It is technically possible to combine more methods, for example evolutionary algorithm can be used on low-performance hardware.

### 4.3.10. Using design patterns for concrete user interface generation

Although automatically generated user interfaces can be optimal in terms of a mathematical function estimating its quality, user interfaces generated by a skilled designer are usually better accepted by the target user audience. The reason is that some its aspects of estimating user interface quality cannot be technically solved by a machine. To bring this gap we decided to involve so called user interface design patterns (see [7]) into the concrete user interface generation process.

Design patterns are in terms of computer science general reusable solutions to commonly occurring problems. The most known and commonly used are design patterns for object oriented programming introduced by Gamma at al. in their famous book Design patterns: elements of reusable object-oriented software [8]. In the HCI domain are user interface design patterns well-adopted solutions to occurring HCI issues.

The UiGE can accommodate patterns as special case of mapping - template mapping (for more details see section 4.4.2 above). A good example of pattern usage is login dialog for individual platforms. Also template-mappings provide its cost function value (estimated user effort) and are subject of the optimisation process. It is technically possible to skip an existing pattern if the estimated user effort will be lower (for example decomposition of login process into multiple steps using big elements for moderately visually impaired users).

### 4.4. Differences from Supple

Because some key aspects of our approach, in front of all the combinatoric optimisation were inspired by Supple [3], it is necessary to specify the differences. The main differences between our approach and SUPPLE are follows:

- optimisation cost function considers element importance,
- user interface design patterns are involved into the generation process by defining templates,
- explicit mapping definition,
- definition of virtual mapping (affects the total cost function, for example consideration of interface-level user interface consistency)
- UIProtocol is used for both AUI and CUI, this makes the approach more uniform,
- possible support for multimodal user interfaces,
- sophisticated layout managers are subject of the future work,
- advanced accessibility is subject of the future work.
SECTION 5 - Intermediate Results

5. Intermediate Results

This chapter shows the current state of system implementation. Although not all components are currently implemented, the system is functional and can automatically generate user interfaces for three different platforms. Table 5 contains list of system components with current status of implementation.

<table>
<thead>
<tr>
<th>component</th>
<th>status</th>
<th>note</th>
</tr>
</thead>
<tbody>
<tr>
<td>touchscreen client</td>
<td>implemented</td>
<td>implementation of clients is currently for purposes of UiGE. Support for full UIProtocol specification (see [19]) is not ensured.</td>
</tr>
<tr>
<td>desktop client</td>
<td>implemented</td>
<td></td>
</tr>
<tr>
<td>iPhone client</td>
<td>implemented</td>
<td></td>
</tr>
<tr>
<td>UIProtocol server</td>
<td>implemented</td>
<td></td>
</tr>
<tr>
<td>UiGE - abstract interface builder and provider</td>
<td>implemented</td>
<td></td>
</tr>
<tr>
<td>UiGE - mappings provider</td>
<td>partially implemented</td>
<td>supports concrete mappings</td>
</tr>
<tr>
<td>UiGE - concrete element provider</td>
<td>implemented</td>
<td></td>
</tr>
<tr>
<td>UiGE - optimiser</td>
<td>not implemented</td>
<td></td>
</tr>
</tbody>
</table>

Table 5: Implementation status

Following figures show examples of user interfaces generated for particular client platforms. These user interfaces were generated upon abstract user interface shown in section 4.1.2, respectively in Appendix A.

Figure 24: Automatically generated user interface for desktop client

In Figures 24-26 notice different size of elements and layout on different platforms. These properties are affected by both device and user model.
Figure 25: Automatically generated user interface for touchscreen client.

Figure 26: Automatically generated user interface for mobile platform - Apple® iPhone.

Values for properties like screen size are resolved according to concept shown in section 4.3.2, that means that both user and device models are respected.
6. Evaluation of Results

The UiGE is intended to generate user interfaces, therefore it is obvious that the evaluation requires user-centered approach. Such evaluation was not performed yet, in front of all because of the complexity of system implementation. Already implemented components was tested on various abstract user interfaces. This testing proved that the system is able to generate simple concrete user interfaces for three different platforms.

Because of the facts listed above, complex evaluation is a subject of the future work. The evaluation should follow these recommendations:

- Qualitative usability testing of generated user interfaces to uncover basic problems.
- Quantitative testing of a typical generated user interface. Control group will use similar, but manually designed user interface. Expected result is to specify the effectivity of automatic solution against traditional approach.
- Quantitative testing of a generated user interface personalised for purposes of users with special needs. The control group will use traditional non-personalised manually designed user interface.
7. Conclusions and Future Work

This document summarised conceptual design and intermediate results of an system for automatic user interface generation. The conceptual design builds upon requirements derived from an extensive state of the art analysis (chapter 2). The most important aims were:

- lower complexity of input models,
- accommodate combinatoric optimisation, usage of templates and optimisation of consistency in a single tool,
- reflect importance of particular AUI elements in generated CUI,
- provide generic approach not limited a particular platform.

The complexity of input models is lowered by a uniform specification of the immediate input - abstract user interfaces. They use the similar UIProtocol notation as the output - automatically generated concrete user interfaces. User and device models were also separated in order to simplify the process of input specification. These futures make the development simpler. Consequently are the total development cost for complex applications (especially nomadic and ubiquitous applications) lower.

Templates and consistency optimisation is supported by the generalised mapping. Each mapping provides its value of cost function. This cost-value of particular AUI elements is weighted by their importance. Feasible mappings with resolved value of cost function are subject of later combinatoric optimisation. Its aim is to find globally optimal solution (with minimal total value of the cost function). Involving importance into the optimisation consequently ensures, that critical elements are represented by the best available concrete structure, while less-important elements are rendered using a space-saving interface components.

Proposed design is demonstrated on an exploratory implementation. It accepts abstract user interfaces, user models and device models. There is an API for specification of mapping. The current system automatically generates user interfaces for various target user interface platforms.

An important subject of the feature work is to resolve the optimisation. It is necessary to develop an optimisation algorithm, that finds an optimal solution in satisfactory times. Another requirement with high research potential is a specification of the cost function. In effect this function must take into account the target platform, target user, current context, user interface consistency, and application of templates. The total value of optimisation function should correspond to estimated usability of user interface. Furthermore, it is necessary to split the process of optimisation from the rest of the platform so that low-end computation devices (such as PDA’s and smart phones) are not required to perform complex computation. This computation will be outsourced to powerful data centres and cloud technologies. This is necessary to fully support nomadic application and ubiquitous computing.
8. Bibliography


9. Relevant refereed publications of the author


10. Remaining refereed publications of the author

11. Unrefereed publications of the author

12. Dissertation thesis

Title: User interface generator

Abstract:

Although there were multiple attempts to use automatic user interface generation, none of them was widely practically used. From the developer’s perspective, the main issues are the overall complexity of input to the user interface generators (input models), platform dependency, and corresponding poor scalability. From the user’s perspective, the automatically generated user interfaces are usually problematically accepted. Main problems are the low consistency of these interfaces, especially the poor consistency with user’s knowledge and habits. Automatically generated user interfaces then often look in effect too artificial and over-complicated. Current methods of the automatic design usually do not allow the developers to explicitly specify the importance of individual elements of the input model and the desired “look and feel” of the resulting user interface. The former problem does not allow the user interface generators to construct the user interface where the elements are placed (or even omitted) based on their actual importance. The latter problem makes all user interfaces look alike and thus diminishing the potential to articulate a user experience.

We simplified the input models on both the system level and the immediate input to the user interface generator. Furthermore, we propose a solution that accommodates combinatoric user interface optimisation, template application and consistency optimisation in one tool. This, potentially increases consistency and naturalness of automatically generated user interfaces. Moreover, the optimisation considers the importance of individual elements of the input models. The application of templates has a potential to explicitly specify the “look and feel” of the generated user interfaces and thus increase the user experience. The proposed design provides a maximal scalability: It is based on UIProtocol, a platform independent language for describing user interfaces and communication.

The solution was demonstrated on an exploratory implementation. The system is able to automatically generate user interfaces for various target user interface platforms and user’s properties and preferences for a given abstract description of the desired functionality of the user interface.

The key aspect of the future work is development of an optimisation algorithm, that provides acceptable solutions in satisfactory computation times. A point with high research potential is the specification of the cost function that corresponds to the usability of produced user interface. The scalability of the system must be further improved. It is necessary to split the process of optimisation from the rest of the user interface platform so that low-end computation devices (such as PDA and mobile phones) are not required to perform complex computation which can be outsourced to powerful data centres and cloud technologies. This is necessary to fully support nomadic application and ubiquitous computing.

Keywords:

Automatic User Interface Generation, User Interface Consistency, Model-Based User Interface Design, User Interface Design Patterns
Appendix A - UIProtocol representation of AUI

```xml
<interface class="cz.ctu.igs.user.root">
    <container>
        <property name="title" value="Title 1"/>
        <property name="description" value="Description 1"/>
        <element class="public.display">
            <property name="title" value="Pokracujte dvermi A5 (modra chotba)"/>
            <property name="description" value="Tady bude delsi popis pro nevidome"/>
            <property name="icon" value="map1.png"/>
            <property name="torsion.importance" value="1.0"/>
        </element>
        <element class="public.display">
            <property name="title" value="Tady bude nejakej delsi popis..."/>
            <property name="description" value="Tady bude delsi popis pro nevidome"/>
            <property name="icon" value="terminaly.mov, terminaly.jpg"/>
            <property name="torsion.importance" value="0.7"/>
        </element>
        <element class="public.trigger">
            <property name="title" value="Rozumím"/>
            <property name="description" value="Tady bude delsi popis pro nevidome"/>
            <property name="icon" value="ok.png"/>
            <property name="torsion.importance" value="1.0"/>
            <behavior trigger="action" action="root.confirm"/>
        </element>
        <element class="public.trigger">
            <property name="title" value="Potrebuji poradit"/>
            <property name="description" value="Tady bude delsi popis pro nevidome"/>
            <property name="icon" value="question.png"/>
            <property name="torsion.importance" value="0.8"/>
            <behavior trigger="action" action="root.needAdvise"/>
        </element>
        <element class="public.trigger">
            <property name="title" value="Pomoc"/>
            <property name="description" value="Tady bude delsi popis pro nevidome"/>
            <property name="icon" value="help.png"/>
            <property name="torsion.importance" value="1.0"/>
            <behavior trigger="action" action="root.help"/>
        </element>
    </container>
</interface>
```
Appendix B - UIProtocol representation of CUI

```xml
<interface class="cz.ctu.igs.user.root">
  <position>
    <properties names="x,y,width,height" values="0,0,1024,768" />
  </position>
  <container>
    <element class="public.text">
      <position>
        <properties names="x,y,width,height" values="50,100,700,45" />
      </position>
      <property name="text" value="Pokracujte dvermi A5 (modra chotba)" />
      <property name="font.height" value="32" />
      <property name="font.color" value="FFFFFF" />
    </element>
    <element class="public.button">
      <position>
        <properties names="x,y,width,height" values="50,145,1024,45" />
      </position>
      <property name="text" value="Tady bude nejakej delsi popis, ze clovek cestou ..." />
      <property name="font.height" value="32" />
      <property name="font.color" value="FFFFFF" />
      <behavior trigger="mouse.up" action="root.confirm" />
    </element>
    <element class="public.button">
      <position>
        <properties names="x,y,width,height" values="50,240,300,100" />
      </position>
      <property name="text" value="Rozumim" />
      <property name="font.height" value="32" />
      <property name="font.color" value="FFFFFF" />
      <behavior trigger="mouse.up" action="root.help" />
    </element>
    <element class="public.button">
      <position>
        <properties names="x,y,width,height" values="50,540,300,100" />
      </position>
      <property name="text" value="Pomoc" />
      <property name="font.height" value="32" />
      <property name="font.color" value="FFFFFF" />
      <behavior trigger="mouse.up" action="root.needAdvise" />
    </element>
  </container>
</interface>
```
Appendix C - User model implementation

<table>
<thead>
<tr>
<th>property</th>
<th>example value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>id</td>
<td>q001</td>
<td>user’s unique identification</td>
</tr>
<tr>
<td>contrast</td>
<td>0.8</td>
<td>minimum relative contrast value</td>
</tr>
<tr>
<td>brightness</td>
<td>0.7</td>
<td>minimum relative brightness value</td>
</tr>
<tr>
<td>line width</td>
<td>1.2</td>
<td>minimum relative line width the user can recognise</td>
</tr>
<tr>
<td>font size</td>
<td>1.2</td>
<td>minimum relative font size the user can recognise</td>
</tr>
<tr>
<td>sound volume</td>
<td>1.0</td>
<td>minimum relative sound volume the user can recognise</td>
</tr>
<tr>
<td>speech volume</td>
<td>1.1</td>
<td>minimum relative speech volume the user can recognise</td>
</tr>
<tr>
<td>key size</td>
<td>1.2</td>
<td>minimum relative button size, minimum spacing</td>
</tr>
<tr>
<td>key press time</td>
<td>1.1</td>
<td>minimum relative repeat time between button presses</td>
</tr>
<tr>
<td>hand tremble</td>
<td>0.8</td>
<td>relative hand tremble 0-bad, 1-good</td>
</tr>
<tr>
<td>one hand</td>
<td>FALSE</td>
<td>user can use only one hand</td>
</tr>
<tr>
<td>no hand</td>
<td>FALSE</td>
<td>user can not use hands</td>
</tr>
<tr>
<td>body height</td>
<td>178</td>
<td>users height in centimetres</td>
</tr>
<tr>
<td>in wheelchair</td>
<td>FALSE</td>
<td>user is in wheelchair</td>
</tr>
<tr>
<td>info text</td>
<td>0.9</td>
<td>text representation quotient. 0-impossible, 1-no problem</td>
</tr>
<tr>
<td>info picture</td>
<td>0.8</td>
<td>graphics representation quotient. 0-impossible, 1-no problem</td>
</tr>
<tr>
<td>info colours</td>
<td>0.3</td>
<td>colour representation quotient. 0-impossible, 1-no problem</td>
</tr>
<tr>
<td>info sound</td>
<td>0.7</td>
<td>sound representation quotient. 0-impossible, 1-no problem</td>
</tr>
<tr>
<td>info speech</td>
<td>0.8</td>
<td>speech representation quotient. 0-impossible, 1-no problem</td>
</tr>
<tr>
<td>info simple haptic</td>
<td>0.6</td>
<td>simple haptic repre. quotient. 0-impossible, 1-no problem</td>
</tr>
<tr>
<td>info braille code</td>
<td>0.05</td>
<td>braille code repre. quotient. 0-impossible, 1-no problem</td>
</tr>
<tr>
<td>information complexity quotient</td>
<td>0.8</td>
<td>how complex information can use accept. 0-bad, 1-good</td>
</tr>
<tr>
<td>language.x (x - language code)</td>
<td>0.9</td>
<td>language knowledge quotient. 0-bad, 1-good</td>
</tr>
</tbody>
</table>

Appendix D - Device model implementation

<table>
<thead>
<tr>
<th>property</th>
<th>example value</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>contrast</td>
<td>50</td>
<td>min. abs. contrast a typical user can use (in dev. units)</td>
</tr>
<tr>
<td>brightness</td>
<td>40</td>
<td>min. abs. brightness a typical user can use (in dev. units)</td>
</tr>
<tr>
<td>line width</td>
<td>2px</td>
<td>minimum absolute line width a typical user can recognise</td>
</tr>
<tr>
<td>font size</td>
<td>10px</td>
<td>min. absolute font size a typical user can recognise</td>
</tr>
<tr>
<td>sound volume</td>
<td>70</td>
<td>min. absolute sound volume a typical user can recognise</td>
</tr>
<tr>
<td>speech volume</td>
<td>80</td>
<td>min. absolute speech volume a typical user can recognise</td>
</tr>
<tr>
<td>key size</td>
<td>50px</td>
<td>min. absolute key size, min. absolute spacing</td>
</tr>
<tr>
<td>minimal user height</td>
<td>0cm</td>
<td>minimal required user height</td>
</tr>
<tr>
<td>screen width</td>
<td>1024px</td>
<td>current screen width</td>
</tr>
<tr>
<td>screen height</td>
<td>768px</td>
<td>current screen height</td>
</tr>
<tr>
<td>screen dpi</td>
<td>150</td>
<td>current screen dpi</td>
</tr>
<tr>
<td>supported elements</td>
<td>&lt;list&gt;</td>
<td>list of supported UIProtocol elements</td>
</tr>
<tr>
<td>is touchscreen</td>
<td>TRUE</td>
<td>specifies whether the device is touchscreen</td>
</tr>
<tr>
<td>is multitouch</td>
<td>TRUE</td>
<td>specifies whether the device support multitouch</td>
</tr>
<tr>
<td>has keyboard</td>
<td>FALSE</td>
<td>specifies whether the device physical keyboard</td>
</tr>
<tr>
<td>has mouse</td>
<td>FALSE</td>
<td>specifies whether the device physical mouse or similar dev.</td>
</tr>
</tbody>
</table>
Appendix E - User interface generation algorithm

1. optimalMappings ← null
2. foreach element in abstractUserInterface
3.    legalMappings ← call getLegalMappings
4.     call sortAccordingCost(legalMappings)
5.     foreach mapping in legalMappings
6.        currentMappings ← mapping[0]
7.     initialConcreteInterface ← call constructConcreteInterface (abstractUserInterface, legalMappings)
8.     if call isLegal(initialConcreteInterface)
9.        optimalMappings ← currentMappings
10.     return initialConcreteInterface
11. else
12.     optimalMappings ← call optimumSearch(initialConcreteInterface, legalMappings)
13.     return constructConcreteInterface (abstractUserInterface, optimalMappings)