2012

Adaptive interfaces for application defragmentation in diverse operating contexts

Arun Kalyanasamy
Iowa State University

Adaptive interfaces for application defragmentation in diverse operating contexts

by

Arun C. Kalyanasamy

A thesis submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Major: Computer Science

Program of Study Committee:

Leslie Miller, Major Professor

Simanta Mitra

Sree Nilakanta

Iowa State University

Ames, Iowa

2012

DEDICATION

I would like to dedicate this thesis to my parents Devikarani and Kalyanasamy without whose support I would not have been able to complete this work. I would also like to thank my friends for their loving guidance during the writing of this work.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my thanks to everyone who helped me with various aspects of conducting research and the writing of this thesis. First and foremost, Dr. Miller for his guidance, patience and support throughout this research and the writing of this thesis. His insights and words of encouragement have often inspired me and renewed my hopes for completing my graduate education. I would also like to thank my committee members for their efforts and contributions to this work: Dr. Simanta Mitra and Dr. Sree Nilakanta.

ABSTRACT

To write a code once and run the application anywhere has been the holy grail of application developers. With the diversities in the operating contexts introduced by various hardware, software, users and carriers, user interfaces must cater to all the present and future operating contexts. The expense of developing a software product to tailor to new market fragmentations is soaring. We propose a model that will aid user-interface designers working in the field of mobile computing to build applications across operating contexts, without the hassle of redesigning it to accommodate the unique constraints introduced by each operating context. This research will make the following contributions: 1) propose a novel user interface model for isolating the common features of an operating context for automatic rendering of the interface according to the constraints of each context, 2) prove feasibility of the model by handling one such operating context constraint namely screen size and present an efficient implementation of the proposed model for it using Microsoft Silverlight and 3) use the above implementation to measure the cost-benefit trade-off with a user study involving tasks of varying complexity.

CHAPTER 1. INTRODUCTION

With the sudden surge of smartphones in the market, small interface devices are everywhere. But these smaller screens make it more difficult to perform some of the most common tasks such as browsing and reading [11, 3]. But mobile computing has some inherent challenges for User Interface (UI) design and development [23]. For instance, UIs of applications must run on multiple Operating Contexts(OC) [19] varying across desktops, gaming consoles, tablets and smartphones, each of which have their own degrees of fragmentation [26]. Moreover each OC has its own unique set of constraints set by different capabilities in terms of hardware, software and user variety. With the reduced screen size on a device such as a smartphone, application UI designers must choose to show only a limited set of features. Also, users tend to use smartphones in environments where they have a limited attention span making manipulation of a complex interface even more complex [27]. With this in mind, the constraints inherent in OCs can be categorized into two main categories: device and user constraints.

To address such limitations posed by mobile OCs, several researchers have proposed adaptive interfaces, where the system adapts to the device and user needs [11, 2]. An important aspect of this is that an application should be able to automatically render an interface on any device for any user. Given the wide range of device types, form factors, interaction styles and users, it is not feasible for each application developer to create an interface for each device and user [14].

1.1 Device-based adaptation

Although customization based on devices would theoretically yield huge benefits, research on adaptation has been mostly done on adaptive web content (eg., [11, 25]) instead of on adaptive

graphical user interface control structures with the exception of SUPPLE [12] which we shall talk about in Chapter 2. Adaptation of content poses different challenges than adaptation of control structures. For instance the interfaces from the PEBBLES [29] project make rough assumptions about the screen size and iCrafter [22] relies heavily on hand-crafted templates for its metadata and XIML(eXtensible Interface Markup Language), a language for user interfaces [21] relies on the designer specifying which widgets have to be used. Tools like Damask [21] greatly simplify the process of designing user interfaces for cross platforms and can be used as a good starting point. The bulk of adaptive graphical user interface (GUI) control structures have been performed on desktop-sized displays, where evaluations have been inconclusive [12]. The primary focus of this research will be on GUI control based adaptation. Also, in this research since there will be variance in the GUI itself, stability and predictability are considered important measures in the user study.

1.2 User-based adaptation

Research as early as Greenberg and Witten [7] showed that an adaptive interface with fewer steps in its navigation path was easier and faster for a user than a static structure. On the other hand, Mitchell and Shneiderman [18] showed that static menus were preferred since adaptive menus reordered based on frequency and confused the user. But since introduced, split menus [24] have reversed this opinion. Gajos et al. showed a strong preference for split menu based adaptive interfaces which are featured in Microsoft Office [13]. Some research has shown that adaptive menus or toolbars are preferred over their static counterparts [13, 7], whereas other research has shown the opposite [4, 16, 18] due to factors such as accuracy, awareness, predictability and stability. There are multiple options when it comes to interfaces, such as: the split interface that copies important function to another toolbar, the moving interface that moves the important functions to another toolbar and the visual pop-out interface that highlights the frequent functions as explored in [13]. The choice of interface type should be decided based on factors such as consistency and awareness [5]. Other than that, adaptation algorithms fall into recency-based algorithm, where the 'n' most recently used commands are promoted by the adaptive interface or the frequency-based algorithm where the most frequently

used commands are captured over a short window of interactions [13]. Another novel approach that has been previously researched is error-based adaptation by Miller [17, 1]. This approach [17] uses vision and motor errors as feedback for generating the adaptive interface. It is highly useful in field applications due to the vulnerability of poor external conditions, such as bad weather. In the other research [1] done by Miller et al. they have explored design issues associated with web applications for the elderly and have proposed an error detection based approach. The study revealed that error-based profiling of the users performed statistically as well as observation-based profiling. In our research the model takes user input using user traces that keeps track of the user traces to provide the information to the model as metadata which is later used for generating the interface (explained in Chapter 3).

1.3 Terminology Review

An OPERATING CONTEXT for an application is the external environment that influences its operation. For a mobile application, the OC is defined as the hardware and software environment in the device, the target user, and other constraints imposed by the carriers.

USER INTERFACE is the space where interaction between humans and machines occurs, the goal of which is effective operation and control of the machine and feedback from the machine to the operator to aid in operational-decisions.

APPLICATION is the computer software designed to help the user perform specific tasks. Mobile applications are those that are built for smaller, low-powered devices such as smartphones.

AWARENESS quantifies the degree to which the user is aware of the full feature set of an application [5].

PREDICTABILITY is when the most recently selected item always appears in the adaptive top.

Elements of the MVVM pattern include: MODEL: model refers to the object model that represents the real state content. VIEW: the view refers to all elements displayed in the GUI such as windows, graphics and controls. VIEWMODEL: serves to databind the view and the model. It changes the Model information into View information and passes commands from

the View into the Model. CONTROLLER: Some references of MVVM include the Controller layer as well, but is irrelevant to our current focus here.

XAML is a declarative XML-based language for initializing structured values and objects in the .NET Framework technologies, particularly, Windows Presentation Foundation (WPF) and Silverlight.

DEPENDENCY INJECTION (DI) is a design pattern that is responsible for decoupling the software components. With DI the object does not need to know how other parts of the system functions. Instead the developer injects the relevant system component in advance with a contract that describes behavior. With DI there is no need to hardcode the dependencies. Instead a component just lists the necessary services and the DI framework supplies these to offer a standard interface.

FUNCTIONALITY is the intended task exposed by the application to the user, the combination of which will be a service. Content-based changes to the UI are not part of the problem being addressed.

CHAPTER 2. REVIEW OF LITERATURE

Adaptation techniques have been usually classified into spatial, graphical or temporal [5]. Spatial approaches rearrange items in the interface to fit the target interface; graphical approaches add to the visualization by highlighting the colors or methods and temporal is the hybrid of the above two with the effects lasting only for a short time. Work on adaptive interfaces has been largely performed on spatial techniques, with very little work done in temporal or graphical techniques.

ICrafter deals with the application as a set of services and proposes a framework for interaction between the services and interfaces using a variety of modalities and input devices with on-the-fly aggregation of the services [22]. This pattern has associated with it the fully-blown cost of a Service Oriented Architecture (SOA) and it relies heavily on hand-crafted templates. On the other hand, SUPPLE did a good job of defining the interface rendition problem [12]. It defines the Interface generation as an optimization problem that takes user and device constraints together. Upon solving the optimization problem we get the optimum interface to be rendered. Although theoretically this approach should prove solid, it is still untested on the new devices and the experiments that were performed on the small devices proved to be slow [12]. Fogarty and Hudson [10] have also worked on optimization based approaches to generating interfaces. They have presented GADGET, an experimental toolkit that supports optimization for interfaces to enable programmers to create multiple views for the interface and later get the optimization using a lazy evaluation framework [10]. Although interesting, this approach has high costs when dealing with a large number of constraints, especially when the mutual exclusiveness of the constraints is unknown resulting in a problem which is NP hard to evaluate.

The Model-View-ViewModel (MVVM) is an architectural pattern which originated in Mi-

crosoft as a specialization of the presentation model design pattern introduced by Martin Fowler [6]. The view-model, a value converter on steroids as it is popularly known [6], exposes the data objects from the model in a way that is easily manageable and consumable. One of the key things to note about MVVM is how it leverages the advantages of XAML (eXtensible Application Markup Language) using data binding and its ability to completely isolate any coding whatsoever from the view layer. MVVM is used as the starting point to build our model on. The metadata from device and user constraints will be injected into the model for generating a view at runtime which will be further explained in Chapters 3 and 4.

Adaptive interfaces have appeared in mainstream commercial applications as well for example the Windows Start Menu and Microsoft Office that features Smart Menus. Although there have been a lot of models to support content-based adaptation for user constraints, very less research has been performed on a model that adapts to both device and user constraints based on interface control adaptation.

CHAPTER 3. USER INTERFACE MODEL

The design of a highly adaptive, multi-platform user-interface cannot be developed using a traditional architectural pattern, which is the reason why we propose that this novel architecture model be based on the Model-View-ViewModel (MVVM) pattern. MVVM was designed to utilize specific functionalities exposed by Windows Presentation Foundation (WPF) and Silverlight. It can be used to separate the development of the view from the rest of the layers so that the interface designers do not have to write view code for each OC that comes into the market. They can just use the markup language XAML to bind the UI controls to the viewmodel objects, created and managed by the application developers.

In the remaining subsections, we shall describe model-based techniques that will facilitate the development of UIs that can adapt to multiple OCs. The user interface model that is proposed here will be a formal and declarative description of the UI. The markup language used here has to be declarative to hold information about the OC constraints in such a way that they can be interpreted by a software system easily. XAML exposed by Silverlight meets these criteria and hence shall be used for UI modeling.

The following are the specific layers we shall consider for UI modeling. They are by no means the only relevant layers to mobile computing. However the techniques we develop shall depend only on these layers and hence described in the following subsections.

3.1 Interfacer dictionary

An interfacer is the element that transforms abstract functional elements into structured values or objects to form the UI. Every platform has at its disposal a extensive set of interface elements which are refered to as interfacers, in_i.

A interfacer dictionary, D_I is defined as

$$D_I = \{in_i | i = 1, ..., n\} \tag{3.1}$$

where i maps to the interfacers available on any design language; n is the total number of interfacers in the language.

3.1.1 Example

Here is an example of an interfacer dictionary populated with some XAML controls.

$$D_I = \{Button, Textbox, Textblock, Slider, Togglebutton, Dropdown\} \tag{3.2}$$

3.2 Type dictionary

Each *model* layer element is defined by its type. The type dictionary is defined here as

$$D_T =< T, S > \tag{3.3}$$

where S is the set of interfacer sets and T is the set of types that have to be modeled, where any type t_i is defined as

$$t_i \equiv b_t \mid a_t \tag{3.4}$$

where b_t is the set of base types that can be modeled and the i^{th} basetype

$$b_i \equiv int \mid float \mid double \mid string \tag{3.5}$$

and a_t is a derived type and is defined as the set of all k basetypes from which they are derived from. Any derived type a_i is given by

$$a_i = \{b_j | j = 1, ..., k\} \tag{3.6}$$

and for each type t_i, there is a set s_i of interfacers from the interfacer dictionary(D_I) that can be used to represent that particular type. Say we have

$$s_i = \{in_i | i = 1, ..., l\} \tag{3.7}$$

then the type dictionary is a one to many mapping from types, T $\rightarrow$ S , the set of sets of interfacers.

3.2.1 Example

Here is an example of a type dictionary with mappings from some types to a set of interfacers. In the case of XAML, the XAML controls form the set of interfacers.

$$D_T = \{int \rightarrow \{TextBox, Slider, DropDown\}, password \rightarrow \{Textbox, Password\}\} \tag{3.8}$$

3.3 Rule definition

A rule in this context is defined as the constraint imparted by any Operating Context such as a device or user that directly controls the UI and is machine readable.

A rule r, when applied on a Interfacer(Interface element) dictionary, D_I shall either limit or retain the dictionary to give the constrained dictionary, D_R (explained in section 3.4), such that

$$|D_I| \geq |D_R| \tag{3.9}$$

Each rule should be machine-readable and will be formed using a markup language which we later call the Rule Descriptive Platform(RDP) (explained further in section 3.6).

3.3.1 Example

The constraint considered here is the display resolution. The constraint represented here as a rule is the $small - screen$ constraint. Here is an example that shows how a rule can be represented using XML. QVGA also known as the Quarter Video Graphics Array represents a display resolution of $320 \star 240$ which is considered fairly small compared to resolutions such as

SVGA - 800 $\star$ 600, XGA -1024 $\star$ 768 and more. Therefore the corresponding rule is represented as follows,

$< rule >< display >< size > small < /size >< /display >< /rule >$

3.4 Restricting the dictionary

The interfacer dictionary is limited based on the rules from the platform and the user traces layer. This in turn directly affects the type dictionary used by the model layer as well. In general, if X_r is the rule set and $Rejected_i$ is the set of interfacers that are to be avoided because of each rule r_i, then the restricted dictionary, D_R is obtained as follows:

$D_R = D_I$;
i = 0
while $X_r \neq NULL$ **do**
i=i+1
$D_R = D_R - Rejected_i$
$X_r = X_r$ - r_i
end while

3.4.1 Example

Here is an example of a rule based dictionary restriction. The development platform is assumed to be XAML coupled with Silverlight. Hence, the interfacer dictionary is already populated with interfacer elements which are XAML controls.

Consider a interfacer dictionary with some example elements given by,

$$D_I = \{Button, Textbox, Radiobutton, Togglebutton, Dropdown\} \tag{3.10}$$

Consider a rule set with one of the rules described as shown in section 3.3. The constraint considered here is display resolution and constraint is that it is *small*. The rule for such a constraint is represented as follows,

$r = < rule >< display >< size > small < /size >< /display >< /rule >$

Such a constraint is represented as a rule in order to capture the effect of it on the dictionary. For a rule defining that it has a $small - screen$ constraint, the rule action in the system will be a dictionary of elements that are to be avoided, which shall be named the avoidable dictionary, D_A.

$$small \rightarrow D_A = \{Radiobutton, Dropdown\} \tag{3.11}$$

This can be done so by using these rules to restrict the dictionary accordingly. Upon restricting the dictionary to avoid elements that have the most manipulation cost in terms of the display rule, we get the restricted dictionary as follows:

$$D_R = \{Button, Textbox, Togglebutton\} \tag{3.12}$$

3.5 Platform

A platform is the piece of software linked to a specific technological device. More specifically it describes the software systems that may run a UI. Our model will include the specific device constraints placed on user interfaces by each device on the UI. The integral use of observing a platform constraint in our case is in capturing the effect it has on a particular device that is being observed, which is done so by representing these constraints as rules. The platform layer contains an element for each individual device and the annotations carrying the features and constraints of each device that are presented in the device model. The device layer can be exploited at design time to be used as a static entity. This way the information can be used to develop a user interface for each device. But with the huge number of devices in the market and with the possible inclusions of more devices to the market, it would be more sensible to exploit this dynamically. For example, if there is a change in the keyboard from Qwerty to touch, platform layer registers a change in the input mode and the UI should only include controls that can be manipulated easily with a finger.

A platform, P is defined here as

$$P =< D_I, R_P, C_{MP} > \tag{3.13}$$

where D_I is the interfacer dictionary; R_P is the rule set for that platform as shown in the above section; C_{MP} is the cost set of manipulating each interfacer from the model layer on that platform. This defines the experimental cost that is associated with the use of each interfacer. This can be hardcoded into the system for initial setup and later updated based on the user traces.

3.6 User trace

Some of the best UI renderings that take device constraints into account may still be non-usable if they do not address the user constraints. Given that the primary research objective is to model an adaptive interface that can adapt to both users and devices, we include user traces holding annotations about the user as another layer. One of the most important constituents of a user trace are paths, where the term path refers to the logical sequence of elements influenced by the user. The logical sequence of elements are captured in the model during transitions as a three-tuple sequence explained below. A transition in a path occurs if the user navigates from one interface group to another. A closure of a path is one where the navigation is closed or reset.

Each user trace, ut_i is defined as

$$ut_i = \{path_i | i = 1, ..., l\} \tag{3.14}$$

where $path_i$ refers to a single path in the trace; l is the number of paths accumulated for each user and it increases with use.

The key thing about this approach is that if the navigation leads away it can only mean that the transition has been performed by the user. And hence the sequence of paths can be directly used to capture the user behavior. The path information can be captured as a set of tuples tu_i, with each tu_i carrying the exiting interfacer group (ex_i), the entering interfacer group (en_i) and if they have a common parent interface group (cp_i). The tuples of the navigation path will

contain (ex_i, en_i, cp_i) which can be grouped together to form the overall path or to calculate the navigation cost (explained further in section 3.8).

Each path $path_i$ is defined as

$$path_i =< ex_i, en_i, cp_i > \tag{3.15}$$

where ex_i is the exiting interfacer group; en_i is the entering interfacer group; cp_i is 0 if ex_i and en_i are the same and 1 otherwise.

Also,

$$path_0 =< root, root, 0 > \tag{3.16}$$

where $root$ is the root interfacer group for the application.

The rendering based on the trace is done as a simple prediction mechanism where the user trace is used to judge the flow of rendering of the interface groups. For example if the trace contains path p_j after path p_i we use the same format of the trace to make changes to the UI. At the same time is important to note that frequent changes to the UI will cause disorientation to the user. The format of the trace is not related to the device and can be used to create a custom rendering when the user accesses the same application from a different OC. For example with the new OC, if the screen size is increased we choose to alter the trace to include a more sophisticated interface group that plays out better on that OC with the help of the platform metadata. To implement this exhaustively, we can build a common Rule Descriptive Platform (RDP) that can support the description of all forms of rules obtained from OCs so that the SVM can read these rules dynamically and adjust the view layer accordingly. However the goal is not to build such a system, but to concentrate on the validation of the model to solve the OC constraint problem provided we use such a rule descriptive platform. This is done by specifying the constraints to alter the traces in a straightforward manner by forming a limited rule repository. The UI has to be rendered even before there is any navigation information available. For this purpose, we ignore user traces completely for the initial rendering and use the same once a decent set of transition tuples have been gathered. Also, it is possible to provide typical user navigation as the default rendering scheme. As the trace information accrues, it

can be used for rendering the UI according to the user choice.

Hence the user trace, U is modeled as the set of all path information obtained, named as PATH.

$$U =< PATH > \tag{3.17}$$

where the cost weight is taken from the common parent attribute directly.

3.6.1 Example

Here is an example of a type dictionary with mappings from some types to a set of XAML controls. Initially, the user trace information, U = DEFAULT which means that there are no user traces accrued and the hardcoded set of user trace information are used.

Consider the root element to be

$$path_0 =< root, root, 0 > \tag{3.18}$$

where *root* is the interfacer group $g_0 =$

$< StackPanel >$

<Button Content = "Play" Click="PlayClick"/>

<Button Content = "Options" Click="OptionsClick"/>

<Button Content = "Exit" Click="ExitClick"/>

$< /StackPanel >$

Each of the button click actions in the group g_0 take us to different grids each with it's own functionality, g_1 and g_2 and g_3.

The following are the two user scenarios performed in this environment.

1. The user clicks the Play button on g_0 to go to g_1. There the user clicks on the Additional instructions button.

2. The user clicks the Play button on g_0 to go to g_1. There the user clicks the phone's *back* to go back to g_1.

When the user hits play the control will transfer to the following grid.

$< Grid >$

$< TextBlockText = "Volume" / >$

$< SliderWidth = "100" Height = "30" Value = "0" Orientation = "Vertical" / >$

$< TextBlockText = "Instructions$: Jump to take doodle to the top."$/ >$

$< ButtonText =$ "Additional Instructions"$/ >$

$< /Grid >$

3.6.1.1 Scenario 1

In this scenario, say the user navigates from the interfacer group, g_0 to g_1 and manipulates the interfacers inside the group, g_1 itself.

So the path information accrued will be

$$path_1 =< g_0, g_1, 1 > \tag{3.19}$$

$$path_2 =< g_1, g_1, 0 > \tag{3.20}$$

Therefore the user trace information accrued now shows

$$U = \{< root, root, 0 >, < g_0, g_1, 1 >, < g_1, g_1, 0 >\} \tag{3.21}$$

3.6.1.2 Scenario 2

In this scenario, the user navigates from the interfacer group, g_0 to g_1 and chooses to come back to g_0.

So the path information accrued will be

$$path_1 =< g_0, g_1, 1 > \tag{3.22}$$

$$path_2 =< g_1, g_0, 1 > \tag{3.23}$$

Therefore the user trace information accrued now shows

$$U = \{< root, root, 0 >, < g_0, g_1, 1 >, < g_1, g_0, 1 >\} \tag{3.24}$$

3.7 Model

The model layer shall describe the object model or the data access layer depending on whether it is the object-oriented approach or the data-centric approach. In addition to the functionality and hierarchy information that will be described in the object model, we shall model information to check if the corresponding task/subtask is optional, repetitive and if it has other sub tasks inside it or linked to it. The dependency information between the AUCs in the SVM (explained further in 3.8) will be driven by the model behavior through the viewmodel layer. Other information passed on from the model includes the state information, type information, label information and grouping information. The model layer provides the type dictionary, D_T, the metadata information to the SVM, their corresponding costs to the platform.

$$M =< E, D_T, C_{MP} > \tag{3.25}$$

where E includes the variables and objects of the model; D_T is the type dictionary; C_{MP} is the manipulation cost of using each possible interfacer from the type dictionary for each type on a particular platform(This is directly used for calculating the final navigational cost).

3.8 SmartViewModel(SVM)

The view model layer performs the same function as in the MVVM pattern where it serves as an abstraction layer between the view and model layers. It acts as a data binder/converter that changes model information to view information and passes commands from the view to the model. The XAML user controls shall be of two types in this model namely Abstract User Controls (AUCs) and Concrete User Controls (CUCs). AUCs are elements on the XAML page that are not tailored to fit on the UI of a specific OC. They form the starting point of the modeling on the SVM. Using the various AUCs, we can generate a view for each possible

object in the Platform Layer. Multiple AUCs will be mapped surjectively to one CUC and the AUC that is most sensible for that OC shall be considered by the SVM as the CUC as shown in Figure 3.1 (This includes the changes in adaptation due to the User Trace layer). More practically, a list of AUCs will be picked from the available user controls in XAML and later optimized from the information obtained from the platform layer.

The SVM has an inbuilt interface generator that makes use of the information from the model layer, OC constraint information from the platform, user information from the user trace layer to form a set of AUCs. Note: The information obtained from the model layer will vary according to the OC constraint type being addressed. For example, for screen size, information such as the state variables, label information, grouping information and type information will be obtained. The dependency is only on the type of OC constraint (screen size, memory, CPU) and not on the constraint itself making this model generic and adaptable for all OC constraints. The interface generator that consumes this information to produce the CUC will form the major part of our research.

3.8.1 User trace analysis

User trace analysis shall form the first and foremost step in the interface generation. This involved checking for usage patterns to realign the cost dictionaries that will be used by the interface generator. For the very first time the UI has to be rendered, the user trace, U that is analyzed will be the default user trace that is hardcoded into the system.

3.8.1.1 Pattern checker

The primary use of the user traces are to analyze the traces to obtain usage patterns and render the UI in such a way that is conducive for the most frequently occuring usage pattern. This is done using a automatic pattern checker that sits at the application level and works as follows,

```
uTrace = U ; PatternDictionary < Pattern, int > = null;
  while uTrace ≠ NULL do
    path = uTrace.First;
    uTrace= uTrace - path;
    tempPattern = tempPattern + path;
    if tempPattern.complete() then
      if PatternDictionary[tempPattern].Exists() then
        PatternDictionary[tempPattern] + +;
      else
        PatternDictionary[tempPattern] = 1;
      end if
      tempPattern.clear();
    end if
  end while
```

3.8.1.2 Cost realignment

Once the pattern checker checks to see if the navigational cost (explained in Section 3.10) is big enough to affect the cost dictionaries, it realigns the cost dictionaries accordingly. The realignment is taken into account in the very next rendering by the SVM, which is the reason why this is considered a pre-processing step after which the SVM continues with the interface generator algorithm.

3.8.2 Interface generator algorithm

The SVM uses the preprocessed cost dictionaries from the user trace analysis in the following algorithm to give an optimized rendering for the application.

- Perform user trace analysis to realign the cost dictionaries that is used by the following steps
- Pre-form a interfacer dictionary D_I and a type dictionary D_T that maps all the types of model variables to a list of possible interfacers (Note there will be custom variable types which will be serializable and hence can be broken down to be treated as inbuilt types)
- Gather the possible types of grouping information to form XAML panels
- Define the type dictionary, D_T as the Initial AUC Set (IAS)

- Add the model layer metadata to the IAS to output a dictionary called the Controlled AUC set (CAS)
- Use the OC metadata (platform and user metadata) to form selection rules
- Use the selection rules to restrict the CAS further to form the Selected AUC set(SAS)
- Select the most optimized mapping in the SAS using the dependency information. (Explained further in 3.10)
- Define this optimum SAS mapping with the dependency information as the Concrete User controls (CUC) for that particular device and application and pass it to the view layer

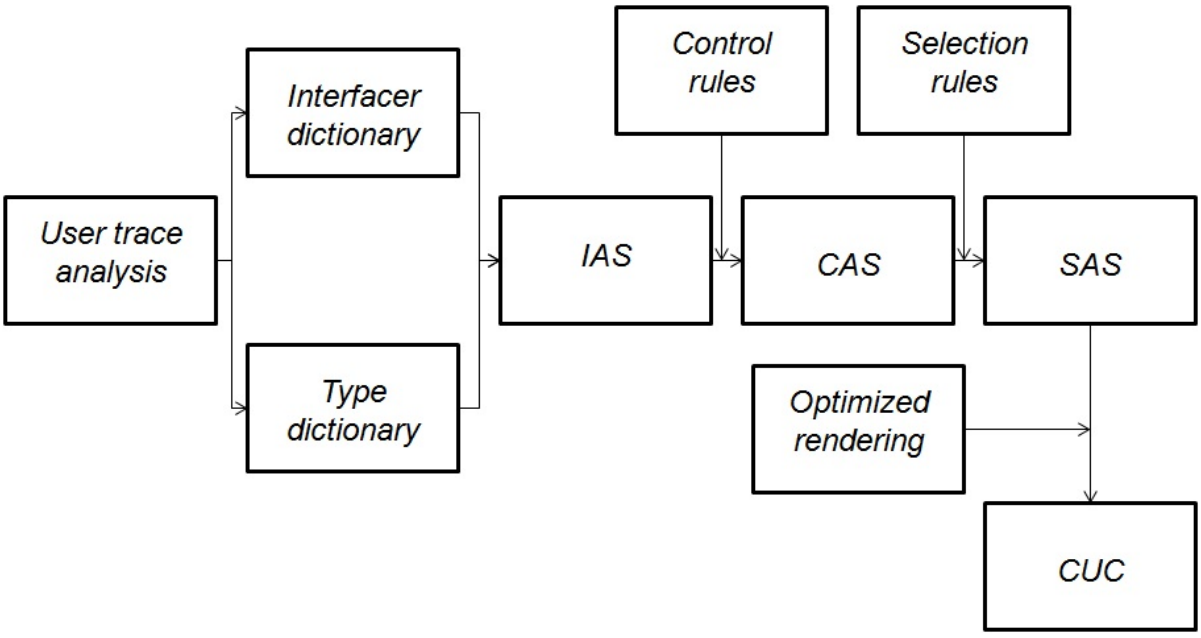

Figure 3.1 Interface Generation

3.9 View

The view layer describes the visual appearance of the user interface and includes information that shows the windows and user controls as a hierarchical model. The entire XAML code is dynamically created using code behind based on the CUC information passed on to the View

layer from the SVM. The chosen set of XAML elements and panels from the dictionary for each interfacer are used to render the UI with groupings performed from panel information passed onto the View layer. The internals of the layers modeled above play a significant role in the design of the UI. Further emphasis is on how each layer interacts with each other which is controlled by the viewmodel. These mappings play a significant role in the UI behavior and this is outlined in Figure 3.2.

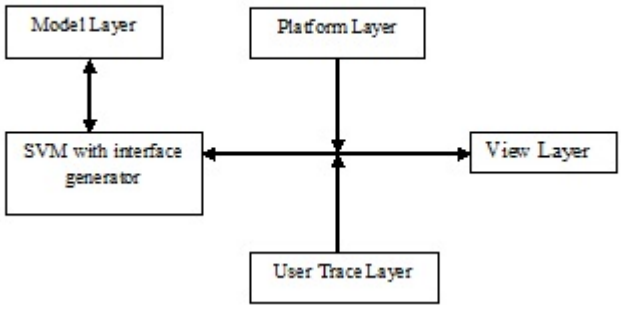

Figure 3.2 User Interface Model

3.10 Optimized rendering

The objective is to render each interface using the most optimized XAML element for that particular device. As used above the one to many mapping, M: Interfacer $\rightarrow_n$ XAML Elements is the mapping that has been filtered across all the rules. Even after the generation of the SAS, there may be multiple possible XAML elements for an interface and we have to obtain the most optimum in terms of the user effort. Hence the user trace layer and SVM are closely bound together in evaluating the SVM rendering. The cost function used to evaluate the model should be simple and fast to use. Using the tuple information in the traces, namely (ex_i, en_i, p_i), we have the entering interfacer group, exiting interfacer group and the parent group, which if different will denote a transition. Associated with each tuple is a weight for each transition, with a higher weight allocated whenever there is a change in the parent group. Given the SAS, the total cost of navigation is the sum of the weighted times of the tuples found in the user trace and the cost of manipulating those interfacers maintained in the platform. (Upon initial

start of the application when there is no user trace we use a default user trace hardcoded into the application for calculation of the navigation cost). Using this navigational cost we can decide the final mapping in M to form the CUC tree.

Given the SAS, we have a dictionary with a many to one mapping, M : Model object types $\rightarrow$ Set of List of interfacers for each object. We also have the dependancy information based on which similar objects can be combined to form panels. Say, PI is the total number of one to one mappings that can be obtained from M, then the interfacer sets is given by

$$SAS = \{sas_i | i = 1, ..., PI\} \tag{3.26}$$

Also from the grouping information, we can group the interfacers into panels as various possible permutations, then the groups, G is given by,

$$G = \{g_i | i = 1, ..., PG\} \tag{3.27}$$

Considering the view information as well, we get a further refined rendering with the view set, V given by,

$$V = \{v_i | i = 1, ..., PV\} \tag{3.28}$$

Using SAS, G and V we can form a matrix with each element of the matrix represented by the navigation cost, δ_{ijk} that corresponds to using interfacer set, sas_i in group g_j in view v_k thereby cross-referencing it with the type of view for each chosen set and grouping.

	g_1	.	.	g_{PG}
sas_1	δ_{111}			δ_{1PG1}
.	.	.	.	.
.	.	.	.	.
sas_{PI}	δ_{PI11}	.	..	δ_{PIPG1}

	g_1	.	.	g_{PG}
sas_1	δ_{112}			δ_{1PG2}
.	.	.	.	.
.	.	.	.	.
sas_{PI}	δ_{PI12}	.	..	δ_{PIPG2}

.

.

.

	g_1	.	.	g_{PG}
sas_1	δ_{11PV}			δ_{1PGPV}
.	.	.	.	.
.	.	.	.	.
sas_{PI}	δ_{PI1PV}	.	..	δ_{PIPGPV}

The navigational cost, δ is dependant on both the grouping and interfacer information and is used to determine the final Concrete User Control(CUC) that will be rendered on the UI, given by

$$\delta = \sum_{views} \sum_{user-traces} \left(\sum_{interfacers} (C_{MP}) + \sum_{paths} w_k * cp_i \right) \tag{3.29}$$

where w_k is constant weight determined from experimental results to be multiplied with the common parent factor obtained from the path information.

3.10.1 Example

Consider a To-Do application with a list of all tasks and another screen for adding new tasks. The following is an example that shows the type of renderings the model will arrive at for the application for screen constraint that can take up two abstract values, namely small and big. The example assumes the development environment to be XAML coupled with Silverlight showing the interfacers to be XAML controls.

After restricting the dictionary with the rules as shown in Section 3.4, the interfacers available for rendering namely the SAS are as follows

$$I = \{\{TextBox, TextBlock, CheckBox, RadioButton, ToggleButton\} \tag{3.30}$$

The groupings available are based on the dependency information and the following is a simple set of groupings that directly map to XAML controls

$$G = \{StackPanel, ListPicker, ListBox, Grid, ItemsControl\} \tag{3.31}$$

The views available for rendering are again listed in V and used to the calculation of δ

$$V = \{Pivot, Panorama, ScrollViewer\} \tag{3.32}$$

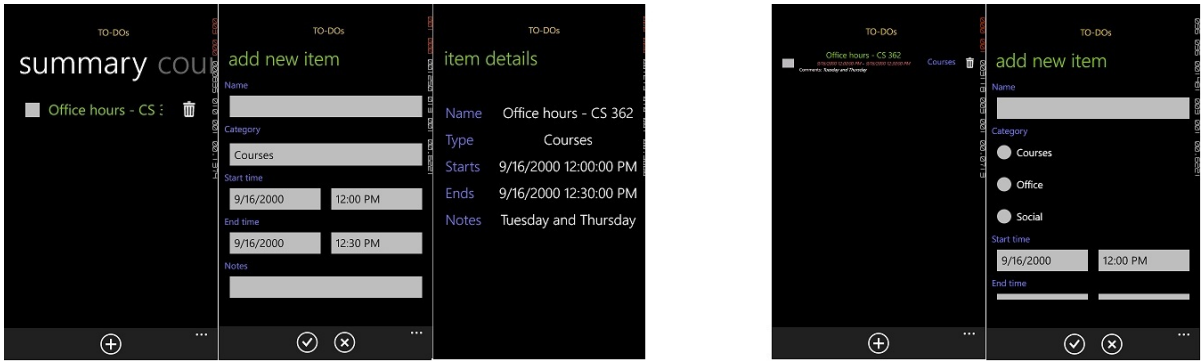

Figure 3.3 a) Screen for smartphone (small screen) b) Screens for tablet (large screen)

Based on these above data the model can produce any of the renderings shown in Figure 3.3. The figure shows the renderings the model displays for two different abstract screen sizes, small and big. The rendering will be checked for consistency everytime the application is initiated to make sure that it is the most efficient UI for the current cost dictionaries. The cost dictionaries will get updated based on the user traces if the navigational cost is higher than the preset cut-off, leading to a different rendering as shown in Figure 5.2. The figure shows a new rendering when the details page of the small screen is accessed frequently.

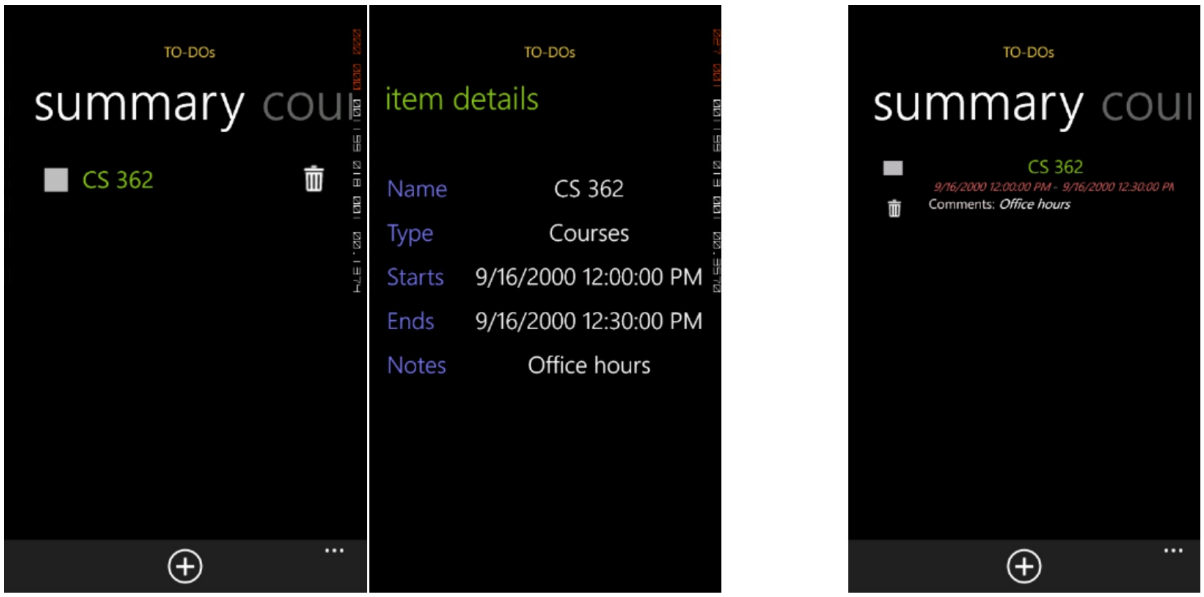

Figure 3.4 a)Initial rendering for small screen b)New rendering for small screen for frequent details page access

3.11 View layer integration

Based on the gathered data in the SVM, the cost matrix is constructed and the cost is calculated by plugging it into Equation 3.29 Assuming that the constraint taken into picture is small screen size. In which case the most optimum view will be Figure 3.3 c) the tabbed view. The optimum rendering is verified with manually optimized renderings. Upon selecting the optimized rendering the chosen output is given as the CUC tree by the SVM to the view layer, where the UI is rendered from code behind using the information from the SVM. Since the model is straightforward, the time taken for rendering is fairly less and the user experience is smooth. The view layer can be easily constructed using a development environment such as Silverlight or WPF, which exposes easy two-way binding between the view and the model layer. Considering that the binding happens through the SVM in our model, any input data from the user can be controlled to re-configure the UI in a way that best fits the particular OC.

3.12 Feasibility check - Addressing the screen size constraint

In Chapter 4 we shall show the feasibility of the model by implementing the same for one particular OC constraint namely, screen size. The variations in screen size among devices has been one of the main focus of research in the area of defragmentation in recent times.

3.12.1 Terms and concepts

The following are certain terms specific to the screen size constraint to be looked at before we proceed

3.12.1.1 Screen size

Screen size is the actual physical size of the screen, which is measured as the screen's diagonal and can be categorized as small, normal, large and extra large based for simplicity

3.12.1.2 Screen density

Screen density is the amount of pixels in an observed area of the screen. This is typically captured as dpi (dots per inch) and can again be categorized as low, medium, high and extra high for simplicity.

3.12.2 Density independence and size dependence

As long as the application preserves the physical size of user interface elements, density independence is achieved automatically. This is important to maintain before addressing the actual constraint, the screen size. Without density independence, an interfacer may appear large on a low density screen or the vice-versa causing usability issues. That is the reason why most of the styling options in XAML support percentages, where each interfacer size can be defined as a percentage of the total available size.

The actual constraint to be addressed is the screen size, where hard-coding different layouts for each UI screen for each device is tedious and time-consuming and worst of all error-prone. Using this model we can automatically create various renderings based on the different screen

sizes and measure the cost associated with each rendering before displaying them. In the following chapter, we shall look at how the screen size constraint is handled in detail and thereby demonstrate the feasibility of the model.

CHAPTER 4. IMPLEMENTATION

One of the most obvious constraints imposed by the diverse number of devices is the variable screen sizes, which vary from a flat screen desktop to a smartphone. We have chosen to apply the proposed model to the screen size constraint and analyze the results with a user study. The reason for choosing to deal with the problem of variable screen sizes is because of its closeness to various aspects of the UI. For example it directly influences the choice of having to split the page into various sub-pages to fit the screen size or to choose the precise set of interface components that fit the screen. Other device constraints such as limited input capabilities or memory size can be handled internally using simple selection of the settings, whereas the display problem is a much more sensitive issue that requires a more comprehensive solution.

We have built a To-Do application using the proposed model in the context of screen size using Silverlight for the user study of the model. The application has been deployed on two target devices, namely a smartphone and a tablet which represents two different abstract screen sizes of small and big respectively. The application has been built to include actions of varying complexities for the user study, namely to view the list of tasks and their details and to add a specific task with custom details. The application includes the following features:

- The application on startup displays the list of all To-Do tasks, if any, which was added by the user and includes the option to mark it as complete or to delete the item altogether
- The list of To-Do tasks displayed may or may not show the details of the task on the same page depending on the size of the target device. The rendering depends on the interface generator algorithm and the user trace accumulation
- Add task page for the user to add tasks with details such as name, type of task (course, office and social), start time of the task, end time of the task and additional notes.

Information gathering here, as it is in any domain, is the most time consuming yet most important step before the actual implementation because it gets directly added to the IAS to form the Controlled AUC Sets(CAS) in the SVM. Apart from the screen information from the platform layer which forms the selection rules, we need more information from the model layer as well as explained below in Section 4.2. The set of information will vary with the type of OC constraint and this is specific to screen size constraint alone. The description should have enough information to generate a good user interface and no information about the look or feel which in turn should be decided by the interface generator.

4.1 Rule implementation

The following code snippet the generic rule type used to store selection rules from the platform layer. The rule datatype will be used in Section 4.3.

```
namespace Notepad
{
    public class Rule
    {
        //context for which the rule is defined for. Eg: Display
        public String Context { get; set; }
        //Type of the context. Eg: Size
        public String ContextType { get; set; }
        //Actual rule information Eg: Small
        public string Information { get; set; }

        public Rule(String context, String contextType, string information)
        {
            Context = context;
            ContextType = contextType;
            Information = information;
        }
    }
}
```

4.2 Model layer implementation

4.2.1 State variables and commands

Interface designers must know what can be manipulated on an appliance in order to build an interface for it. Associated with each state variable is the type that informs the interface generator how the UI can be manipulated. For example, consider a category selector control on the add task page of the To-Do application (refer to implementation section below), which can be represented with a list type with range constraints. The interface generator has information on how to handle a list type with constraints and hence it helps to specify information of the type of the state variables. In situations where an application cannot provide feedback about state changes from the model layer back to the viewmodel layer, commands can be useful.

4.2.2 Type information

Each state variable as mentioned above shall be specified with the type information. Apart from the generic types such as boolean, integer, fixed point, floating point, enumerable datatypes and string we have custom datatypes that are modeled to resemble real world objects. These are represented using serializable objects so that they can be further separated into generic types that are guaranteed to be understandable by the interface generators.

4.2.3 Label information

Apart from the above information, the interface generator must also have information about labeling the individual components that represent the state variables and commands. Label information is highly interface dependant and is sometimes cumbersome to predict for future device additions. Hence generic label information shall be used to avoid limiting the model to existing devices alone.

4.2.4 Grouping information

Grouping of the interface components into similar components is highly common and intuitive in Silverlight or WPF. Also, it makes the interface more readable and stable to the user.

Hence grouping information will go a long way in how similar components can be grouped to make it fit to the screen better.

4.2.5 Dependency information

With the two-way binding we can get the interface to respond when a particular component is disabled. Also, information on the other variables that will be affected has to be specified for better structuring of the graphical interface.

4.2.6 Data context of actual model

This includes typical information that the model layer provides to any ViewModel which can be in the form of a relational database or flatfiles. In the To-Do application here we implement an L2S (Linq to Sql - Refer to [14]) and a data context for adding and removing To-Do tasks from the database. The data context will include the custom datatypes that stores the To-Do tasks and the user trace database table.

4.2.7 Sample model implementation

4.2.7.1 Metadata structure

Sample structure for storing the metadata information that includes all the information mentioned above.

```
public Metadata(bool state, string type, string label, string groupId, bool
    custom)
{
    State = state;
    Type = type;
    Label = label;
    GroupId = groupId;
    Custom = custom;
}
```

4.2.7.2 Dictionary structures

The list of all interfacers available for each type, their corresponding metadata, the groupings available for interfacers and the views available for the groupings are shown in this code snippet

```
//Type dictionary - Mapping from model object to Interfacer dictionary that form
    actual controls
public static Dictionary<object, List<Interfacer>> TypeDictionary;
//Metadata dictionary for each model object
public static List<Metadata> MetadataList;
//Group dictionary - //Type dictionary - Mapping from model object to Interfacer
    dictionary that forms groups
public static Dictionary<object, List<Interfacer>> GroupDictionary;
//View dictionary - //Type dictionary - Mapping from model object to Interfacer
    dictionary that forms views
public static Dictionary<object, List<Interfacer>> ViewDictionary;
```

These datatypes are populated initially during application start by the model layer using the controls available from XAML. The metadata list includes information about both inbuilt and custom datatypes.

4.2.7.3 Tables and data context

This code snippet shows the data context for accessing the SQL tables using the LINQ framework

```
public class ToDoDataContext : DataContext
    {
        // Pass the connection string to the base class.
        public ToDoDataContext(string connectionString)
            : base(connectionString)        { }

        // Specify a table for the to-do items.
        public Table<ToDoItem> Items;

        // Specify a table for the categories.
```

```
        public Table<ToDoCategory> Categories;

        //Specify a table for the user settings.
        public Table<UserPath> Paths;
    }
```

Here is an example table showing how the user trace data context is defined in the database:

```
[Table]
    public class UserPath : INotifyPropertyChanged, INotifyPropertyChanging
    {
        // Define ID: private field, public property, and database column.
        private int _id;
        [Column(DbType = "INT NOT NULL IDENTITY", IsDbGenerated = true,
            IsPrimaryKey = true)]
        public int Id...

        // Define exiting interfacer group: private field, public property, and
            database column.
        private string _exitingGroup;
        [Column]
        public string ExitingGroup...

        // Define entering interfacer group: private field, public property, and
            database column.
        private string _enteringGroup;
        [Column]
        public string EnteringGroup...

        // Define parent interfacer group: private field, public property, and
            database column.
        private string _parentGroup;
        [Column]
        public string ParentGroup...

        INotifyPropertyChanged Members...
```

```
        INotifyPropertyChanging Members...

    }
```

4.3 Platform implementation

The platform layer of the application includes the default cost dictionaries that are used by the interface generator for optimized rendering, display size abstractor that outputs the abstract sizes for which the cost dictionaries have to be formed, the rules populator from the platform layer that directly correspond to selection rules in the SVM.

4.3.1 Sample platform implementation

4.3.1.1 Cost Dictionaries

Default cost dictionaries used for initial rendering which is updated with user trace accumulation

```
//Abstract cost types for measuring the cost of using an interfacers in a
    particular OC
public enum cost { None = 0, Trivial = 1, Considerable = 4 };

//Mapping from the screen size context to the dictionary of interfacers
//Eg: For two different screen sizes small and big there are two default cost
    dictionaries corresponding to both
public static Dictionary<ScreenSizeContext, Dictionary<Interfacer, cost>>
    ScreenCostDictionary;
```

4.3.1.2 Display area abstractor

```
namespace Notepad.Platform
{
    //Get generalized screen sizes to work with from actual screen size
    public static class DisplayArea
```

```
    {
        private static string _screenSize;

        public static string ScreenSize...

        static DisplayArea()
        {
            double pixelSize = System.Windows.Application.Current.Host.Content.
                ActualWidth * System.Windows.Application.Current.Host.Content.
                ActualHeight;

            //The following abstracts only two different screen sizes for the
                purpose of better size delineation
            if (pixelSize <= 800*480)
                ScreenSize = "small";
            if (pixelSize > 800*480)
                ScreenSize = "big";
        }
    }
}
```

4.3.1.3 Selection rules

```
static PlatformRulesPopulator()
{
    SelectionRules = new List<Rule>();

    //New rule that records the display size of the current device
    Rule area = new Rule("Display", "Size", DisplayArea.ScreenSize);

    SelectionRules.Add(area);
}
```

4.4 User trace implementation

The user trace layer saves the set of all user navigational paths as a three tuple sequence on the database for the SVM analysis. The traces are stored as a observable collection of user paths that uses a data context to access the user trace table stored using the L2S.

4.4.1 User trace implementation

```
//User trace table for saving the path data
[Table]
public class UserPath : INotifyPropertyChanged, INotifyPropertyChanging
{
    // Define ID: private field, public property, and database column.
    private int _id;
    [Column(DbType = "INT NOT NULL IDENTITY", IsDbGenerated = true, IsPrimaryKey
        = true)]
    public int Id...

    // Define exiting interfacer group: private field, public property, and
        database column.
    private string _exitingGroup;
    [Column]
    public string ExitingGroup...

    // Define entering interfacer group: private field, public property, and
        database column.
    private string _enteringGroup;
    [Column]
    public string EnteringGroup...

    // Define parent interfacer group: private field, public property, and
        database column.
    private string _parentGroup;
    [Column]
    public string ParentGroup...
```

```
    INotifyPropertyChanged Members...

    INotifyPropertyChanging Members...

}
```

```
public class UserModel : INotifyPropertyChanged
{
    //List of all paths from the data context
    private ObservableCollection<UserPath> _traces;

    //L2S data context
    private ToDoDataContext toDoDB;

    // Class constructor, create the data context object.
    public UserModel(string toDoDBConnectionString)
    {
        toDoDB = new ToDoDataContext(toDoDBConnectionString);
    }

    //A collection of all user paths
    public ObservableCollection<UserPath> Traces...

    // Write changes in the data context to the database.
    public void SaveChangesToDB()
    {
        toDoDB.SubmitChanges();
    }
    //Query database and load the userpath collectionused by the pivot pages
    public void LoadUserSettingsFromDatabase()
    {
        //Speficy the query for getting back the user traces from the database.
        var userSettingsInDB = from UserPath path in toDoDB.Paths select path;

        //Query the database and load the trace information
```

```
        Traces = new ObservableCollection<UserPath>(userSettingsInDB);
    }
    //Add a user path to the database and collection
    public void AddUserPath(UserPath newUserPath)
    {
        //Add a user path item to the data context
        toDoDB.Paths.InsertOnSubmit(newUserPath);

        //Save changes to the database
        toDoDB.SubmitChanges();

        //Add a user path to the user traces
        Traces.Add(newUserPath);
    }

    INotifyPropertyChanged Members...
}
```

4.4.2 User trace accruement

The user traces are accrued whenever the user navigates away from one interface group onto another. The three tuple format is used for capturing the transition in the user focus. The following is the code snippet that is introduced into the navigate away event handler for a particular interfacer group.

```
private void navigateToDetailPage(object sender, System.Windows.Input.
    MouseButtonEventArgs e)
{
    ...

    //Creation of the user path to be sent using the navigation service
    UserPath newUserPath = new UserPath
    {
        ExitingGroup = "main",
        EnteringGroup = "details",
```

```
        ParentGroup = "main"
    };

    //Add the userpath to the usermodel
    App.uModel.AddUserPath(newUserPath);

    //Navigate away from interfacer group to another group
    ...
    NavigationService.Navigate(new Uri(urlWithData, UriKind.Relative));
}
```

4.5 SVM implementation

The SVM is the main focus of our implementation which involves the user trace analysis followed by the interface generation algorithm the output of which is the CUC for the device.

4.5.1 User trace analysis

The primary step in the implementation is the user trace analysis which directly affects the cost dictionaries. The application has a default user trace for the application that is used for initial analysis. As the user trace accrues the analysis is more specific to the user. The analysis is performed in two steps:

- The pattern checker iterates through the user traces using the algorithm explained in 3.8.1.1 and forms the pattern dictionary.
- The cost realignment adjusts the cost dictionaries, C_{MP} if the pattern checker confirms a particular pattern's navigational cost to be higher than the rest of the patterns

4.5.2 Sample user trace analysis implementation

4.5.2.1 Pattern checker

```
//Runs through the user trace to find patterns and their number of occurences
private static Dictionary<Pattern, int> checkPattern()
```

```
{
    Dictionary<Pattern, int> _patterns = new Dictionary<Pattern, int>();

    Pattern tempPattern = new Pattern();
    StringList p1, p2;

    IEnumerator<UserPath> e = uModel.Traces.GetEnumerator();
    e.MoveNext();
    e.MoveNext();

    p1 = new StringList();
    p1.Add(e.Current.ExitingGroup);
    p1.Add(e.Current.EnteringGroup);
    p1.Add(e.Current.ParentGroup);

    do
    {
        p2 = new StringList();
        p2.Add(e.Current.ExitingGroup);
        p2.Add(e.Current.EnteringGroup);
        p2.Add(e.Current.ParentGroup);

        tempPattern.AddLast(p2);
        if (checkForValidPattern(p1, p2))
        {
            if (_patterns.ContainsKey(tempPattern))
                _patterns[tempPattern]++;
            else
                _patterns.Add(tempPattern, 1);
            tempPattern = new Pattern();
            if (e.MoveNext())
            {
                p1 = new StringList();
                p1.Add(e.Current.ExitingGroup);
                p1.Add(e.Current.EnteringGroup);
```

```
                p1.Add(e.Current.ParentGroup);

                tempPattern.AddLast(p1);
            }
        }
    } while (e.MoveNext());
    return _patterns;
}
//Check to see if cost dictionaries are to be affected
public static bool checkFactorChanged()
{
    //Pattern dictionary
    Dictionary<Pattern, int> patterns = checkPattern();

    int count = 0;

    int x = getMaxFactor(patterns);
    foreach (Pattern p in patterns.Keys)
    {
        if (patterns[p] * experimental_weight < x)
        {
            count++;
        }
    }
    if (count == patterns.Count-1)
        return true;

    return false;
}
```

4.5.2.2 Cost realignment

```
//create the user model object to load the user setting if any
user = new UserModel(DBConnectionString);
```

```
user.LoadUserSettingsFromDatabase();

//Sample check to see if navigation cost changed to readjust the costs
if (checkFactorChanged())
{
    foreach (Rule r in Platform.PlatformRules.SelectionRules)
    {
        //Interfacer_x is the currently user interfacer that contributes the
            maximum cost
        (CostDictionaries.ScreenCostDictionary[r.Context + r.ContextType + r.
            Information])[Interfacer_x] = CostDictionaries.cost.Trivial;

        //Interfacer_y is the newly suggested interfacer to replace Interfacer_x
        (CostDictionaries.ScreenCostDictionary[r.Context + r.ContextType + r.
            Information])[Interfacer_y] = CostDictionaries.cost.None;
    }
}
```

4.5.3 Interface generation

The choice of the individual interface components, i.e., the interfacers that form the AUC sets can be done in two ways in the interface generator: i) Based on the rules, shrink/expand the interfacer according to the display size provided by the device. Note that we have to operate within the usability constraints in this case. For instance, we cannot shrink a textblock below the minimum readable pixel size or expand beyond the resolution making it blurry and unreadable. For those interfacers which cannot be shrunk or expanded, we have method 2 which is ii) replace the interfacer with a more optimum interfacer from the XAML library for that particular device dynamically. For example a RadioButtonList would take more space whereas a DropBox or a tabbed view would take less space than a complete view and would be more intuitive than a pop-up based view.The interface generation algorithm for both the small and big screens works on the second approach as explained in Section 3.8.2 where the cost of the current interfacer is increased and the new interfacer with a lower cost for interface

generation at the end of user trace analysis. The algorithm from the model is used to obtain the Selected AUC sets (SAS) (shown in Figure 4.1).

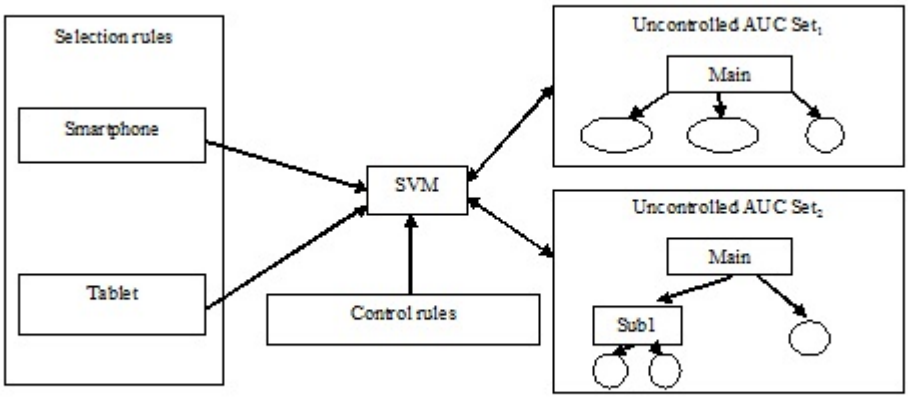

Figure 4.1 SVM interaction with AUC sets using the rules

4.5.4 Sample SVM implementation

4.5.4.1 User trace analysis

Refer to Section 4.5.1

4.5.4.2 Dictionary and rule formation

Refer to Section 4.2 and 4.3

4.5.4.3 CUC Formation

This uses the IAS and the rules to restrict the model information to form the Selected Abstract Sets (SAS). The below implementation shows how the CUC is formed by checking if the SAS components is chosen to be of the abstract value of *cost.None*.

```
public static class RuleBasedInterpreter
{
```

```
//Restict the Dictionaries based on the rules and their corresponding cost
    data

public static Dictionary<string, string> Restricter()
{
    Dictionary<string, string> SAS = new Dictionary<string, string>();

    //Since we handle one constraint display we know the dictionaries that
        are to be accessed. If not we cross reference it with the platform
        rules
    foreach(Rule r in Platform.PlatformRules.SelectionRules)
    {
        Dictionary<string, CostDictionaries.cost> ScreenCostDictionary =
            CostDictionaries.ScreenCostDictionary[r.Context + r.ContextType +
             r.Information];

        //Restrict Views
        foreach(string s in Model.DictionaryMetadataFormer.ViewDictionary.
            Keys)
        {
            List<string> temp = new List<string>();
            temp = Model.DictionaryMetadataFormer.ViewDictionary[s];

            //iterate through the list of possible interfacers to get the one
                 with minimum cost
            foreach (string interfacer in temp)
            {
                if (ScreenCostDictionary[interfacer] == CostDictionaries.cost
                    .None)
                    SAS.Add(s, interfacer);
            }
        }

        //Restrict Groups
        foreach (string s in Model.DictionaryMetadataFormer.GroupDictionary.
```

```
                Keys)
            {
                List<string> temp = new List<string>();
                temp = Model.DictionaryMetadataFormer.GroupDictionary[s];

                //iterate through the list of possible interfacers to get the one
                    with minimum cost
                foreach (string interfacer in temp)
                {
                    if (ScreenCostDictionary[interfacer] == CostDictionaries.cost
                        .None)
                        SAS.Add(s, interfacer);
                }
            }

            //Restrict Views
            foreach (string s in Model.DictionaryMetadataFormer.TypeDictionary.
                Keys)
            {
                List<string> temp = new List<string>();
                temp = Model.DictionaryMetadataFormer.TypeDictionary[s];

                //iterate through the list of possible interfacers to get the one
                    with minimum cost
                foreach (string interfacer in temp)
                {
                    if (ScreenCostDictionary[interfacer] == CostDictionaries.cost
                        .None)
                        SAS.Add(s, interfacer);
                }
            }
        }
        return SAS;
    }
}
```

4.6 Optimized rendering and View layer implementation

The optimum rendering of the CUC tree to the view layer includes the cost calculation in two steps, the user analysis and the CUC formation as explained in Section 4.5.4. The CUC tree is again accessed in the view layer for custom rendering. The view layer can be formed by extracting the CUC tree and the metadata to generate the view in XAML code behind. The view layer generation for small screen devices is explained in the code snippet below:

The CUC tree is parsed to check the View, Group and individual control interfacer values, based on which the following code snippet is executed. The snippet corresponds to the following CUC tree path chosen by the SVM.

- App.ViewModel.SAS["View"] = "Pivot"
- App.ViewModel.SAS["CustomGroup1"] = "ListBox"
- App.ViewModel.SAS["String"] = "TextBlock"
- App.ViewModel.SAS["Boolean"] = "CheckBox"

```
// With the obtained CUC tree generate the corresponding controls in code behind
Pivot pivotCtrl = new Pivot();
pivotCtrl.Margin = new Thickness(0, -36, 0, 0);
var pivotItem1 = new PivotItem();
pivotItem1.Header = "summary";
var pivotItem2 = new PivotItem();
pivotItem2.Header = "course";
var pivotItem3 = new PivotItem();
pivotItem3.Header = "office";
var pivotItem4 = new PivotItem();
pivotItem4.Header = "social";

var listBox1 = new ListBox();
listBox1.Name = "summaryToDoItemsListBox";
listBox1.Margin = new Thickness(12, 0, 12, 0);
```

```
listBox1.Width = 440;
listBox1.ItemsSource = App.ViewModel.SummaryToDoItems;
var listBox2 = new ListBox();
listBox2.Name = "coursesToDoItemsListBox";
listBox2.Margin = new Thickness(12, 0, 12, 0);
listBox2.Width = 440;
listBox2.ItemsSource = App.ViewModel.CoursesToDoItems;
var listBox3 = new ListBox();
listBox3.Name = "officeToDoItemsListBox";
listBox3.Margin = new Thickness(12, 0, 12, 0);
listBox3.Width = 440;
listBox3.ItemsSource = App.ViewModel.OfficeToDoItems;
var listBox4 = new ListBox();
listBox4.Name = "socialToDoItemsListBox";
listBox4.Margin = new Thickness(12, 0, 12, 0);
listBox4.Width = 440;
listBox4.ItemsSource = App.ViewModel.SocialToDoItems;
listBox1.ItemTemplate = (DataTemplate)Resources["ToDoListBoxItemTemplate1"];
listBox2.ItemTemplate = (DataTemplate)Resources["ToDoListBoxItemTemplate1"];
listBox3.ItemTemplate = (DataTemplate)Resources["ToDoListBoxItemTemplate1"];
listBox4.ItemTemplate = (DataTemplate)Resources["ToDoListBoxItemTemplate1"];

pivotItem1.Content = listBox1;
pivotCtrl.Items.Add(pivotItem1);
pivotItem2.Content = listBox2;
pivotCtrl.Items.Add(pivotItem2);
pivotItem3.Content = listBox3;
pivotCtrl.Items.Add(pivotItem3);
pivotItem4.Content = listBox4;
pivotCtrl.Items.Add(pivotItem4);

//Add the view to the content panel
ContentPanel.Children.Add(pivotCtrl);

//Application bar
```

```
ApplicationBar = new ApplicationBar();
ApplicationBar.Mode = ApplicationBarMode.Default;
ApplicationBar.Opacity = 1.0;
ApplicationBar.IsVisible = true;
ApplicationBar.IsMenuEnabled = true;

ApplicationBarIconButton newTaskAppBarButton = new ApplicationBarIconButton();
newTaskAppBarButton.IconUri = new Uri("/Images/appbar.add.rest.png",UriKind.
    Relative);
newTaskAppBarButton.Text = "add";
newTaskAppBarButton.Click += newTaskAppBarButton_Click;
ApplicationBar.Buttons.Add(newTaskAppBarButton);
```

A similar path is followed for large screen devices and the SVM produces a CUC path of

- App.ViewModel.SAS["View"] = "ScrollViewer"
- App.ViewModel.SAS["CustomGroup1"] = "Grid"
- App.ViewModel.SAS["String"] = "TextBlock"
- App.ViewModel.SAS["Boolean"] = "CheckBox"

CHAPTER 5. EVALUATION

The main focus of the user study is to evaluate the model's cost-benefit ratio. It gave us a good understanding of the advantages of using adaptive interfaces implementing this model over the static solution. In order to choose a participant group that will resemble the end-user market for a smartphone device such as Windows Phone, the participants were chosen from both genders and different age-groups. We also selected users who are both skilled and non-skilled in the mobile computing field for this study.

5.1 Conditions

The main goal of the user study is to test the model's adaptability to different screen sizes. This was done using a tablet simulator running Windows Phone and a smartphone running Windows Phone which have two completely different screen sizes. The task list page of the To-Do application is rendered on a tablet sized display and the smartphone. The secondary goal of the user study was to make sure the adaptation is usable and satisfactory to the users, having minimal costs. This was done by providing the users with the To-Do application and asking them to add tasks of varying details using the add new task page. Both these tasks inherently form the high and low complexity tasks respectively. This shows that the model is independent of the application's functionality which is vital in making sure this model is generic enough to be used by all kinds of designers. The third and final goal will be to evaluate the model's ability to adapt to user manipulations. This was done by evaluating the model's ability to re-render the UI based on the user traces accumulated.

5.2 Task

The users were asked to perform three different tasks based on which a survey questionnaire was provided him to evaluate the model

- Start the application and observe the list of all To-Do tasks, if any, which was added by the user on both large and small screens(It includes the option to mark the task as complete or to delete the task altogether)
- Add a new task using the add task page with details such as name, type of task (course, office and social), start time of the task, end time of the task and additional notes on both large and small screens
- Navigate to the details page from the main page repetitively to accrue more usage patterns. Exit the application and reboot application to observe change in rendering based on usage.

5.3 Measures

Finally, these tasks performed by the participants were measured upon the following set of parameters chosen based on previous user research performed in the area

- Awareness - it is the degree to which users are conscious of the full features of the application including those that are not used during the test tasks
- Stability - it is the ability of the application to render the same UI every time it runs on that particular screen. (other constraints such as the user are not taken into account for rendering since only screen size is handled)
- Predictability - guarantee that the application renders the controls that makes the most sense for that particular screen (subjective and hence will be based on the opinion of the participants)
- Timeliness - the time taken to generate and render the interface

Using these parameters, we measure the usability of the model across various operating contexts and also the correlation between the different renderings inspite of the different renderings so that a user transitioning from one operating context to the other does not feel disoriented.

5.4 Results

We have used two different operating contexts, a smartphone and a tablet emulator to test the model's adaptability. The model will be checked for adaptability based on the target OC, followed by the usability based on the measures mentioned in Section 5.3 and finally although the renderings are different, we show that the transition between two different operating contexts is smooth and natural.

5.4.1 Adaptability analysis

The UI generated by the SVM is customized for the target device to make the manipulation easier and more intuitive. The CUC chosen by the SVM varies between the two screens for both the main page where the list of all tasks is displayed and the add task page where a new task can be added.

5.4.1.1 OC based rendering

The UI for large and small screen devices are shown in Figure 5.1 and Figure 5.2 respectively. In contrast to the large screen, on the small screen the SVM uses three pages for the application where the details of the task is displayed on a seperate details page hyperlinked to the main page tasks. On a small screen, the cost of rendering the details page is considered lower than the cost of displaying the details on the same pivot page and making the user zoom in to read the text. Also, in the Add New Item page of the small screen the categories is displayed using a dropdown in contrast to a radio button list control used on the large screen.

5.4.1.2 User trace based rendering

The renderings shown in Figure 5.2 and Figure 5.1 are the default renderings provided by the SVM with the default cost dictionaries. But upon user trace accruement, the cost

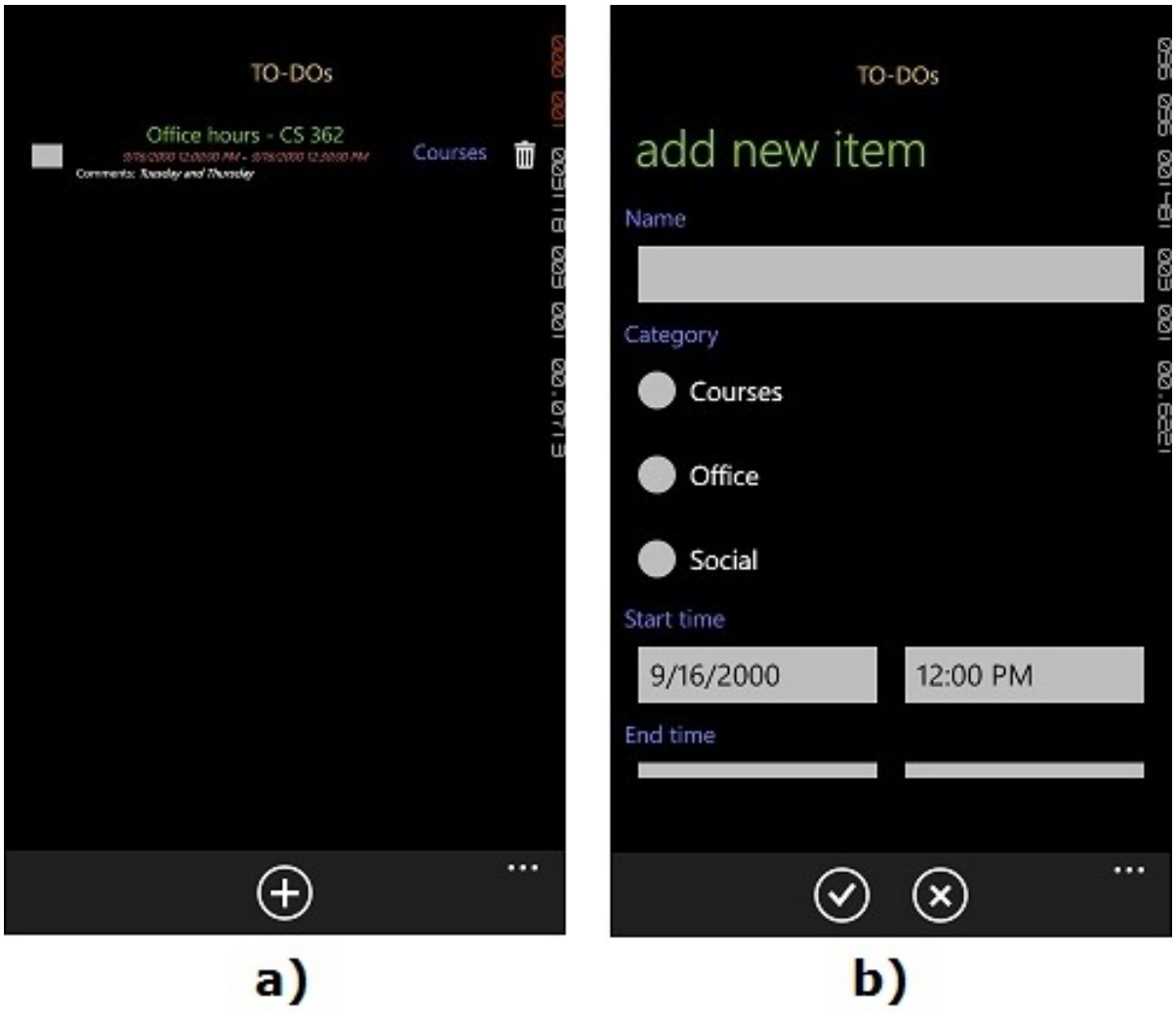

Figure 5.1 To-Do application on a large screen(tablet) a)Main page with task details b)Add task page

dictionaries may get affected thereby paving way for an alternate rendering which is more optimized. Figure 5.1 shows the rendering on the small screen devices for the following scenario. Consider that the user frequents the detail page more than the add task page. Then the user trace accrued spots patterns where the pattern corresponding to the details page is very frequent. In this case the default cost allocated to the choice of displaying the details on a seperate page is affected. Thereby the interfacer cost corresponding to a *ListBox* is reduced as explained in Section 4.5.2.2. This produces a rendering for the small screen as shown in Figure 5.3 where the details are displayed on the main page itself. This rendering will again be

Figure 5.2 To-Do application on a small screen(smartphone) a)Main page b)Add task page c)Task details page

changed based on the user trace accrued thereby making the interface adaptive to user changes as well.

5.4.2 Usability analysis

We have used 12 users * 2 operating contexts * 2 different adaptations (based on OC, based on OC and usage) * 16 questions in the questionnaire to measure awareness, stability, predictability and timeliness using lengthwise and breadthwise analysis of mean and standard deviation on both the target devices. We found that users were slightly more comfortable with UI rendered on the smartphone over the tablet. Figure 5.3 shows the mean and standard deviation analysis for both the the devices. Both the device renderings scored 98 percent on awareness since users were able to recognize all the controls on both the devices although the renderings were completely different. Upon performing post-hoc tests, we found that the users found the smartphone rendering slightly more stable than the tablet giving a stability score of 83 percent (tablet) and 93 percent (on the smartphone). This can also be associated with

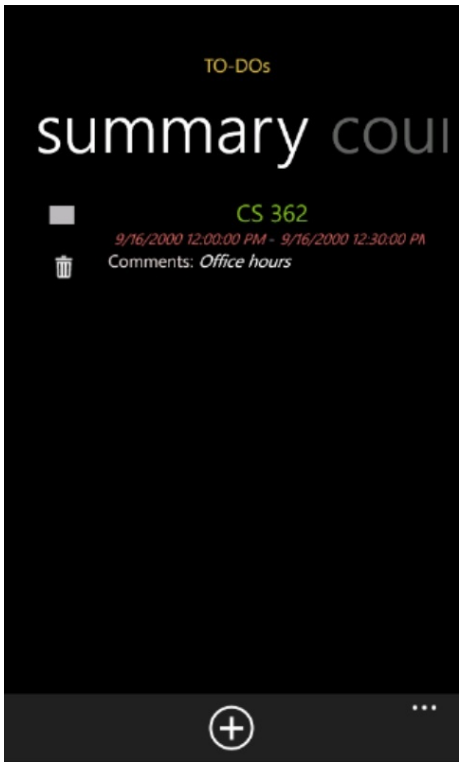

Figure 5.3 To-Do application main page on a smartphone with frequent user visits to the details page

most of the users being smartphone users and not tablet users. This is also due to the change in rendering due to usage, which becomes more stable when reasonably large enough usage characteristics get accrued. Inspite of the changing UI between the devices and usage, there were very high predictability scores of above 96 percent on both the devices, which shows that the model's ability to choose the controls for the device was intuitive. Timeliness scores were high on the smartphone unlike the tablet. The lag in the tablet can also be negated if the emulator can be replaced with an actual device.

5.4.3 Smooth transitioning between renderings

In order to show there are high-benefits at very low costs by using adaptive interfaces that automatically adapt to OC constraints, we also need to show that an user transitioning from one operating context to the other on the same application has a smooth experience. Although the UI may look different between the tablet and the smartphone, in order to create a model

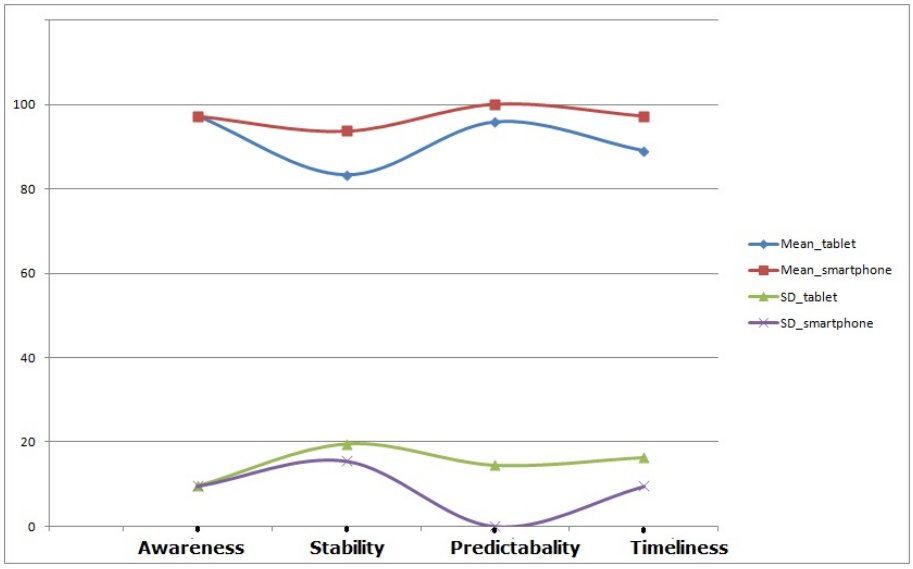

Figure 5.4 Usability measures vs Mean and SD in percentages for both tablet and smartphone

with smooth transitioning capabilities, we show that the correlation between the renderings in terms of awareness, stability, predictability and timeliness is still intact using a t-test.

Table 5.1 t-test between tablet (OC1) and smartphone (OC2)

	Awareness		**Stability**		**Predictability**		**Timeliness**	
	OC1	*OC2*	*OC1*	*OC2*	*OC1*	*OC2*	*OC1*	*OC2*
Mean	0.9725	0.9725	0.8333	0.9375	0.9583	1	0.89	0.9725
Variance	0.0091	0.0091	0.0379	0.0241	0.0208	0	0.0264	0.0091
Observations	12	12	12	12	12	12	12	12
Hypothesized Mean difference	0		0		0		0	
df	22		21		11		18	
P(T≤t) two-tail	1		0.1621		0.3388		0.1465	

Table 5.1 shows the t-test results between the smartphone and the tablet with an alpha value of, $\alpha = 0.05$. From the obtained probabilities, P it is clear that the renderings on the smartphone and tablet are correlated in terms of the usability measures. This ensures that the UI created by the model helps the user to transition between a laptop, tablet or a smartphone easily.

5.4.4 Other user comments

When the participants were debriefed of the experiment, they agreed that the model was adaptable and at the same time useful. They commented on the usefulness of the model to adjust based on usage and liked the fact that the model customized the UI rendering to fit a user more appropriately. Some participants requested for additional custom interfacers in that were more intuitive and those that could make the manipulation easier.

CHAPTER 6. DISCUSSION AND FUTURE WORKS

The results of these experiments was a clear indication of the market need for an adaptive model that can eliminate the need for an application developer to create multiple applications based on operating contexts. It also shows that the adaptation was highly usable and with time a user likes the ability of the model to adapt to usage. Since screen size is one of the biggest and most sensitive operating context issues, we have handled that in our implementation hoping that the research continues and expands to handle other areas of OCs as well.

With increased focus on other forms of changes in operating context apart from the screen size the model can be provided for application developers as a Silverlight or WPF plugin making it very useful. Once the model has been implemented to handle all forms of operating contexts, the model documentation will form a custom architecture pattern that can be used to develop an application once and use forever in all present and future operating contexts that enter the market. We can also test the model by expanding our array to users of android and iphone.

The idea of handling user preferences as a part of the model closely binds the application developer's development environment closely with the target users making this model highly useful for both mobile and immobile devices.

APPENDIX A. USER STUDY

This section will pose questions to the user that will measure the application and in turn the model. This set of questionnaire is provided to the user after the set of tasks performed as mentioned in Chapter 5.

Questionnaire

Personal Information

1. Sex: _____
2. Department: _____
3. Age: _____years
4. PI Level: _____$Graduate/Non-graduate$
5. Current phone model: _____

Generic

1. How often do you download an application to your smartphone?
   (a) Daily
   (b) Weekly
   (c) Monthly
   (d) Rarely
2. What do you use your smartphone the most for?

(a) Play games

(b) Listen to music

(c) Browsing the web

(d) Watch videos

(e) Chat or email

(f) Doing research

3. How much time daily do you spend on your smartphone? _____(Hours)

4. What smartphone do you use the most?

(a) Android

(b) Iphone

(c) Blackberry

(d) Windows Phone

(e) Other _____(Please mention)

5. Which is your favorite smartphone application? _____

6. How many phones do you own?

(a) 1

(b) 2

(c) 3

(d) >3

7. Which is your favorite smartphone platform?

(a) Android

(b) Iphone

(c) Blackberry

(d) Windows Phone

(e) Other _____(Please mention)

8. Other comments: _____

Software Usage

1. On what device was the application easy to use?

   (a) Only on the Smartphone

   (b) Only on the Tablet

   (c) Both

   (d) Neither

2. Were you able to perform the tasks on both devices?

   (a) Yes

   (b) No

3. Did you find there were options to check off or delete a task on both the devices?

   (a) Only on the smartphone

   (b) Only on the tablet

   (c) Both

   (d) Neither

4. Were you able to add a new item on the main page of both the tablet and Smartphone?

   (a) Yes

   (b) No

   (c) Only on the phone

   (d) Only on the tablet

5. List the items you recognized on the Smartphone's 'Add new item' page?

   _____

6. List the items you recognized on the Tablet's 'Add new item' page?

   _____

7. Did you find the transition between the Tablet and the Smartphone smooth?

   (a) Yes

   (b) No

8. Did you find the choice of the elements of the UI natural on both the Smartphone and the Tablet?

   (a) Yes

   (b) No

9. If no, would you have liked to seen exactly the same User Interface (UI) on the Tablet and the Smartphone as well and preferred to zoom in and out?

   (a) No

   (b) Yes (If so why? _____)

10. Did it take you too much time to recognize a button or perform a task on any of the two devices?

    (a) No

    (b) Yes. Which one? _____

11. Did you find the adaptation in the rendering based on the size of the screen useful?

    (a) Yes

    (b) No

12. Did you find the adaptation in the rendering based on the usage useful?

(a) Yes

(b) No

13. Were any of the renderings confusing or unintuitive?

(a) Yes

(b) No

14. Did you find any of the devices slow or inhibited than a usual Smartphone application?

(a) No

(b) Yes. Which one? _____

15. What, if any part of the UI was an issue?

(a) None

(b) _____

16. Other comments:

_____

BIBLIOGRAPHY

[1] Alfred Taylor, Sr., L. M. S. N. J. S. S. M. A. S. and Chama, B. (2009). Using an error detection strategy for improving web accessibility for older adults. *The Second International Conferences on Advances in Computer-Human Interactions.*

[2] Billsus, D., B. C. E. C. G. B. and Pazzani (2002). Adaptive interfaces for ubiquitous web access. *CACM 45.*

[3] Dillon, A., R. J. and McKnight, C. (2005). The effects of display size and text splitting on reading length text from screen, behavior and information technology 9. *ACM.*

[4] Findlater, L. and McGrenere, J. (2004). A comparison of static, adaptive and adaptable menus. *Proc. ACM CHI.*

[5] Findlater, L. and McGrenere, J. (2008). Impact of screen size on performance, awareness, and user satisfaction with adaptive graphical user interfaces. *CHI.*

[6] Fowler, M. (2004). *Presentation Model.*

[7] Greenberg, S. and Witten, I. (1985). Adaptive personalized interfaces: A question of viability. *Behaviour and Information Technology 4.*

[8] J. Nichols, B. A. Myers, M. H. J. H. T. K. R. R. and Pignol, M. (2002). Generating remote control interfaces for complex appliances. *CHI.*

[9] Jacob Eisenstein, J. V. and Puerta, A. (2001). Applying model-based techniques to the development of uis for mobile computers. *ACM.*

[10] James Fogarty, S. E. H. (2003). User interface software and technology. *In Proceedings of the 16th annual ACM symposium.*

[11] Jones M., Buchanan, G. and Thimbleby, H. (2003). Improving web search on small screen devices, interacting with computers 15. *ACM*.

[12] Krzysztof Gajos, D. S. W. (2004). Supple: Automatically generating user interfaces. *ACM*.

[13] Krzysztof Z. Gajos1, 2, M. C. D. S. T. and Weld2, D. S. (2006). Exploring the design space foradaptive graphical user interfaces. *AVI*.

[14] Library, M. M. (2012). Linq to sql. *MSDN Library*.

[15] Lin, J. and Landay., J. A. (2002). Damask: A tool for early-stage design and prototyping of multi-device user interfaces. *In In Proceedings of The 8th International Conference on Distributed Multimedia Systems*.

[16] McGrenere, J., B. R. and Booth, K. (2002). An evaluation of a multiple interface design solution for bloated software. *CHI Letters 4*.

[17] Miller, L. (2010). Modeling error-based adaptive user interfaces. *ISCA 25th International Conference on Computers and Their Applications*.

[18] Mitchell, J. and Shneiderman, B. (1989). Dynamic versus static menus: An exploratory comparison. *SIGCHI Bulletin 20*.

[19] Oracle and its affiliates (2006). Compatiblity and the java platform. *CJP2006*.

[20] Prasad, S. (1996). Models for mobile computing agents. *ACM Comput. Surv. 28*.

[21] Puerta, A. and Eisenstein, J. (2002). Ximl: A universal language for user interfaces. *Unpublished at XIML*.

[22] S. Ponnekanti, B. Lee, A. F. P. H. and Winograd, T. (2001). Icrafter: A service framework for ubiquitous computing environments. *In Proceedings of Ubicomp*.

[23] Satyanarayanan, M. (1996). Fundamental challenges in mobile computing. *Proc of the fifteenth annual ACM symposium on Principles of distributed computing*.

[24] Sears, A. and Shneiderman, B. Split menus:.

[25] Smyth, B., C. P. O. S. (2007). Enabling intelligent content discovery on the mobile internet. *Proc. AAAI*.

[26] Szekely, P. (1996). Retrospective and challenges for model-based interface development. *Proc. of 3rd Int. Workshop on Computer-Aided Design of User Interfaces CADUI'96*.

[27] Thevenin, D. and Coutaz, J. (1999). Plasticity of user interfaces:framework and research agenda. *Proc. of INTERACT'99*.

[28] Wexelblat, A. and Maes, P. (1999). Footprints: History-rich tools for information foraging. *CHI*.

[29] White, J. (2005). Tackle device fragmentation with netbeans and the netbeans mobility pack. *DEVXNB*.