Towards Building Pervasive UIs for the Intelligent Classroom: The PUPIL Approach

Maria Korozi[1] korozi@ics.forth.gr
Stavroula Ntoa[1] stant@ics.forth.gr
Margherita Antona[1] antona@ics.forth.gr
Asterios Leonidis[1] leonidis@ics.forth.gr
Constantine Stephanidis[1,2] cs@ics.forth.gr

[1]Foundation of Research and Technology – Hellas (FORTH)
Institute of Computer Science, Heraklion, GR-70013, Greece
[2]University of Crete, Department of Computer Science

ABSTRACT

Information and Communication technologies have the potential to permeate the classroom and modernize the educational process. However, in the context of a smart classroom, building educational applications poses unique challenges from an HCI perspective, due to the diversity of user and context requirements. This paper introduces a framework that facilitates the design, development and deployment of pervasive educational applications that can automatically transform according to the context of use to ensure their usability. The collection of widgets incorporates both common basic widgets (e.g., buttons, images) and mini interfaces frequently used in educational applications, as ready-to-use modules. The designer can either (i) combine and customize widgets from both categories to build an interface just once, or (ii) build and incorporate it as a custom-made mini interface in the collection for future reuse. Finally, the framework's usability has been evaluated with users obtaining very positive results and potential suggestions for extensions.

Categories and Subject Descriptors

D.2.2 [**Software**]: Design Tools and Techniques - *Modules and interfaces, Object-oriented design methods, software libraries, user interfaces*.

General Terms

Design, Human Factors.

Keywords

Pervasive interfaces, education, smart classroom, widget library.

1. INTRODUCTION

Information and Communication technologies already permeate the classroom environment in many ways. They can play an important role in education by increasing students' access to information, enriching the learning environment, allowing students' active learning and collaboration and enhancing motivation to learn [10]. In this context, the notion of smart classrooms has become prevalent in the past decade [38]. Smart classroom is used as an umbrella term, implicating that classroom activities are enhanced with the use of pervasive and mobile computing, sensor networks, artificial intelligence, robotics, multimedia computing, middleware and agent-based software [11]. Many [12, 33, 38, 39] have envisioned the smart classroom as consisting of technologically enhanced artifacts (e.g., students desk, classroom board, etc.) purposed to offer natural interaction for education, serving both students' and teachers' needs in the processes of disseminating and receiving knowledge.

Building educational applications for the smart classroom poses a series of unique challenges for user interface (UI) design and development. The smart classroom may contain many different artifacts, each one with its own constraints and characteristics: some artifacts are touch-enabled (e.g., the SmartDesk[1]), while others are mouse enabled (e.g., the student's personal computer); some provide large display areas (e.g., the SmartBoard[2]), while others offer a more limited interaction space (e.g., AmIDesk[3], smartphones, e-book readers, or tablets).

UI Migration refers to interfaces that can be transferred among different platforms, allowing the users to continue in real-time their tasks. This is a key dimension of the "intelligent" classroom where traditional educational activities are enhanced with pervasive and mobile technology while applications can be launched in any artifact. Thanks to the migration process, a student interacting with an educational application on his desk can continue the interaction from a shared interactive board from the same point where it was left. As the majority of educational applications should be available through every classroom artifact, application migration should be achieved without compromising usability. However, the diversity of characteristics among the potential artifacts suggests that this process is a particularly complex one. As a consequence, it becomes critical to provide tools which relief designers from the need of building the same application more than once, trying to fit the needs of each classroom artifact, while at the same time ensuring the usability and educational value of the classroom applications.

To address the aforementioned issues this paper presents a framework that facilitates the design, development and deployment of pervasive educational applications. The proposed tools enable designers to build usable UIs that can automatically transform according to each artifact's characteristics. This is achieved through a GUI toolkit targeted to support the development of user interfaces for smart classroom applications. Each of the contained widgets can be appropriately adapted to achieve optimal display on various classroom artifacts. However, the innovation of this work goes beyond the collection of self-adaptive widgets, which has already been investigated elsewhere

[1] Desk artifact equipped with a high-resolution touch-screen.
[2] Touch sensitive interactive whiteboard.
[3] Vision-based touch-enabled desk artifact [4].

[17, 20, 23]. The most significant feature of the proposed framework is that the widget library does not only contain common UI elements (e.g., buttons, textboxes), but also incorporates mini interfaces (e.g., Image Displayer, Multiple Choice Question & Answers, etc.) that can be easily used in educational applications as ready-to-use modules. Furthermore, designers may easily extend the collection of mini interfaces by building and integrating their own custom-made interfaces.

While composing an application UI, designers need to customize numerous basic attributes apart from selecting the desired widgets. This process involves: (i) defining the hierarchy of the application screens, (ii) annotating secondary information and (iii) customizing the application. Therefore, the proposed framework allows the precise definition of the application hierarchy at design time.

The aforementioned framework constitutes the core of the PUPIL [19] system, which additionally introduces a collection of workspaces (namely Classroom Window Managers [18]) tailored to each artifacts' characteristics, while their combination delivers a sophisticated environment for educational applications hosting. The next sections present related work in this domain, introduce the rationale of the framework, describe the migration mechanisms and finally summarize conclusions and future work.

2. RELATED WORK

User interface automatic adaptation has been recognized in recent years as a technical solution for supporting accessibility and enhancing usability of interactive applications and services, in order to support the delivery of personalized user interfaces according to individual users needs, for a variety of devices and in various contexts. This need for adaptive UIs has become more compelling recently, due to the prevalence of mobile devices (smartphones, tablet pcs, notebooks, etc.), as well as due to the advent of ambient intelligence and ubiquitous computing. Notably, a variety of synonym terms has been employed in literature in order to denote the transformation capability of a user interface, including (but not limited to): nomadic [16], migratory [7, 5], plastic [36], adaptive & adaptable [6, 34], multiple [32], multi-device [22], multi-presentation [9], or multi-target UIs [8].

Given the changing nature of an adaptive interface according to the user, the device or the context of interaction, several challenges arise. From the user's point of view, maintaining the usability and consistency of the user interface is fundamental, while from the designer's / developer's point of view, reducing or even eliminating the need for developing a different user interface for each distinct user category, device, context or combination of the above is vital for the feasibility of adaptive user interfaces. To address this need, a number of user interface description languages (UIDLs), tools, models and infrastructures have been proposed in literature. User interface design tools and UIDLs are of particular interest in the context of the presented work. The related most prevalent approaches are reviewed and briefly discussed in this section. User interface prototyping tools such as those proposed by Lin and Landay [22] or Coyette and Vanderdonckt [13] are not being discussed, since they do not produce fully functional UIs and therefore do not introduce real-time UI adaptations.

A category of tools refers to model-based approaches aiming to support UI designers, with the initial approaches focusing on the design and development of desktop applications, such as for example the MOBI-D development environment [28]. More recent tools, aiming to address a larger variety of platforms, can be further classified into automatic generation tools and tools that allow the designer to select from a library of widgets and intervene in the final UI look and feel. SUPPLE [14] is a UI toolkit that automatically generates user interfaces based on device and user models, exhibiting all the advantages of automatic generation, as well as the disadvantages related to the lack of experts' intervention for delivering the most usable and appropriate interface according to a given context of use. TERESA [25] is a tool that supports the design and development of nomadic applications, providing general solutions that can be tailored to specific cases. With TERESA designers can specify the appearance of common UI elements for the supported platforms or modify some general design assumptions. Upon designer's specifications the TERESA tool can generate appropriate XHTML files for each of the supported platforms. The initial version of TERESA has been recently updated [27] in order to support multiple modalities and less traditional devices and modalities (such as digital TV and gesture-based modality).

Graphical user interface tools, on the other hand, allow designers to view and control the layout of the designed interface as they do with traditional GUI builders (e.g., Microsoft Visual Studio [24], NetBeans [26], and Eclipse [37]), and also to define alternative designs for different platforms. Gummy [23] allows designers to generate an initial design for a new platform by adapting and combining features of existing user interfaces created for the same application but for other platforms. PlastiXML [9] allows designers to deal with the plasticity of multi-presentation user interfaces, which however focuses only on the computing platform and its screen size. When the dimensions of a graphical user interface change, the multi-presentation interface automatically switches to the presentation that is the most adapted to this screen. For each different user interface, the designer has to specify its presentation; however, the use of the embedded presentation copy and paste functionality aims to encourage reusability. Various tools [35] have also been developed in order to support user interface adaptation within the Unified user interface development framework [31]. A latest development is the provision of interaction toolkits, which directly embed adaptation capabilities in the available interaction widgets, so as to support the easy development of user interfaces which adapt themselves (e.g., size, colour, fonts, etc.). The JMorph adaptable widget library [20] is a toolkit that inherently supports the adaptation of user interface components. It contains a set of adaptation-aware widgets designed to satisfy the needs of various target devices – Swing-based components for PC, AWT-based components for Windows Mobile devices. Adaptation is completely transparent to developers, who can use the widgets as "conventional" UI building blocks. Another tool focusing on accessibility is MAID [17], which is a multi-platform accessible interface design framework, facilitating the development and customization (through skins) of structured User Interfaces (templates) through the use of predefined UI components (widgets), while promoting reusability and supporting dynamic content population.

Recent approaches rely on the use of XML-based language for the representation of the user interface concepts and the facilitation of the automatic adaptation to multiple devices. For example, TIDE [2] is an environment allowing developers to describe the interface in an abstract language, and specifically UIML, render their work on the desired platform, and then make changes as appropriate. UIML [1] is an XML-based language that can be used to define user interfaces, aiming to reduce the time to develop user interfaces for multiple device families. The TIDE tool was developed mainly as a prototype to exemplify the use of UIML for developing multi-platform user interfaces, supports HTML and

Java desktop applications and does not allow inserting new parts into the interface once its transformation algorithm has been executed. Another UIDL that can support multiple user interfaces at design time and at runtime is XIML [29, 30]. XIML, although being extensible, initially predefines five basic interface components: task, domain, user, dialog and presentation. In addition to the five interface components, XIML also supports relations in order to capture the design knowledge about a user interface and attributes, which are features or properties of elements. In order to solve the problem of creating a presentation component for each target device, XIML uses relations to predefine how an intermediate component (i.e., a single presentation component created by developers) is mapped to a specific widget or control on the target platform. In order to exemplify how XIML could be used for the deployment of a hypothetical application, Puerta and Eisenstein [30] define a framework consisting of three main functions, namely the interactor selection, presentation structure definition and contextual adaptation. However, an actual tool implementing all the above and facilitating developers towards the implementation of a multiple user interface is not reported.

The above overview highlights the key points of the alternative approaches that should be preserved, but also exposes their disadvantages and the areas in which further elaboration is needed. On the one hand, the automated generation of user interfaces based on abstract definitions efficiently addresses multiple target platforms. Nevertheless, such approaches, while scalable, discourage the generation of "rich" interfaces, as their definition becomes extremely complex for a designer to manipulate. Most importantly, these approaches lack fine-tuning and customization facilities to improve the appearance of the generated UI, which as a result weakens its appeal at a visceral level and affects its perceived usability.

On the other hand, approaches based on semi-automated or manual user interface generation techniques deliver "rich" interfaces, but fail to simplify designers work as everything has to be built manually. When such tools are used, it is more common to sacrifice functionality than to re-implement it to support multi-platform delivery. Additionally, the absence of a hierarchy that could guide automatic adaptations to adjust the UI for the new context makes automations unavailable. Finally, similar to the fully automated approaches, customization facilities remain limited and cannot be easily reused.

The approach proposed here endorses the advantages of both automatic and manual UI generation approaches and aims to address their major limitations. In more detail, the explicit definition of the UI hierarchy, the alternative interfaces for each platform, and the ready-to-use mini interfaces significantly simplify the designer's work, as numerous design decisions are made a priori and can be reused. With regards to customization, designers can modify UI's appearance through various presentation parameters that do not compromise usability, whereas by mixing built-in and custom components they can compose new interfaces, which can be reused in an object-oriented manner.

3. OVERVIEW

The work reported in this paper introduces a straightforward yet powerful way to built pervasive UIs for applications targeting an "Intelligent Classroom" [3]. In more details, the envisioned classroom (Figure 1) consists of technologically enhanced artifacts, which incorporate situation-aware functionality offered by the ClassMATE platform [21]. These artifacts aim to replace the traditional student desks, teacher desks and classroom boards, in order to support the developmental processes of disseminating and receiving knowledge.

The proposed framework supports a variety of artifacts that may coexist and cooperate in a classroom environment. Supported artifacts include (i) the student's **laptop**, (ii) the **SmartDesk** (res. 1440x900), which is a desk artifact that combines a personal computer and a touch screen, (iii) the **SmartBoard** (res. 1024x768), which is a touch sensitive interactive whiteboard, (iv) the **AmIDesk** (res. 1600x600) and (v) the **AmIBoard** (res. 1920x1200) artifacts. These last two artifacts are custom-made back-projection devices, which employ computer vision technology. Building applications for the aforementioned artifacts reveals a variety of design challenges. On the one hand, designers need to cope with usual, but hard to resolve, visual constraints like display size variations and rendering performance (clarity, color balance, etc.) of the back-projected devices, while on the other hand they have to support multiple interaction methods (e.g., mouse vs. touch-based) whose properties greatly vary (e.g., sensitivity of vision-based systems, lack of pixel-perfect selection, etc.).

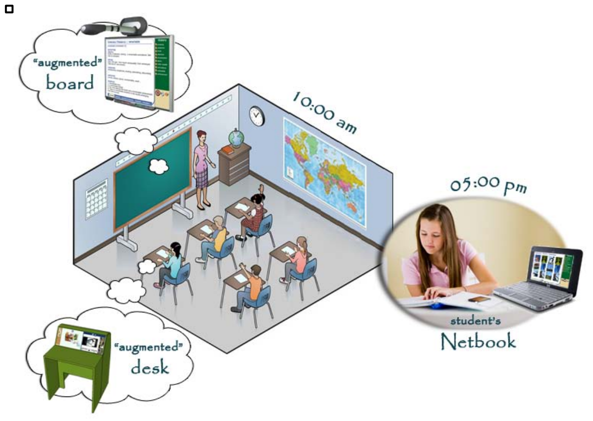


Figure 1. The envisioned classroom.

Within pervasive environments, and particularly inside the "intelligent classroom", user interfaces expand way beyond their static nature and become dynamic components able to react to contextual changes. In such environments, every application can be launched, manipulated and migrated at any intelligent artifact. Generally speaking, designers usually overlook this dimension when building interfaces, resulting in non-scalable interfaces where even the slightest change might require re-design. Many studies [28] raised concerns about such practice, and proposed the concept of abstract model-based descriptions of UIs, where an interface is decoupled from the underlying implementation.

This work aims to solve a practical problem in UI design and development: the level of detail that the design of an application should contain. To address that, it combines abstract interface objects and concrete interface objects to take advantage of their benefits and minimize their pitfalls. Through the introduction of "Application – Logical View – Physical View" triplet (inspired by concepts in [28]), the proposed approach keeps the general design simple and automates time-consuming tasks such as the connection of various screens of an application (AIO), but also permits the designer to freely define the internal structure of each UI and customize it by adjusting various presentation attributes (CIO). Finally, the built-in adaptation mechanism ensures that at runtime the appropriate widget alternative will be selected.

To facilitate development, a mechanism through which designers can model every single interface of an application in a platform-independent manner with minimum effort is provided. The system, during execution, examines the given model and automatically instantiates the appropriate presentation form according to the needs of the targeted classroom artifact. This process ensures applications' portability, as every application has to be designed and built once but can be deployed anywhere.

Application fine-tuning is supported as well. Manual and automatic designs are combined into a hybrid model offering the benefits of both. In the hybrid model, designers' knowledge regarding the context of use is used to drive and customize the generation process by enforcing certain actions to be taken. In addition to dynamic interface generation, the system exploiting the application's structure can automatically generate the necessary navigation paths and link together related screens, thus making deployment easier.

These concepts are encapsulated in a collection of ready-to-use, adaptive, educational-oriented graphical components designed to optimally interact with the various educational applications of the "intelligent classroom" on different devices. Finally, its use adds minimum overhead to application's development process, as the framework can be used directly through Visual Studio and is harmonized with common practices during building UIs for .NET-based applications.

4. THE PUPIL FRAMEWORK

GUI toolkits are the basic blocks through which interfaces are created, while every major platform offers a set of ready-to-use components for the designers to use. The inherent customization facilities allow designers to build countless unique interfaces; nevertheless, from an HCI perspective these interfaces often fail to conform to usability standards and especially satisfy the needs of particular target user groups (e.g., kids, young students). Considering the extended use of GUI toolkits and the available customization facilities, this work offers a collection of widgets that can be automatically adapted to satisfy user and context of use requirements by encapsulating the necessary complexity to achieve that. The collection is built on top of the .NET framework, extends its basic widgets and introduces a set of composite ones as well.

The basic widgets (Table 2) extend native WPF UI Elements (Textblock, Image, etc.) and incorporate the transformation mechanism. Their main objective is to accommodate both passive presentation of information and interaction with the student. They share common characteristics and logic that drives the adaptation process to ensure their legibility (i.e., minimum sizing thresholds, predefined positioning layouts, etc.). Primitive widgets are used as building blocks not only for an application interface, but for the realization of composite widgets as well.

The composite widgets (Table 2), on the other hand, are the realization of common interface patterns recognized among numerous learning applications (e.g., Image Displayer, Multiple Choice Quizzes, etc.) into ready-to-use widgets. The design process for their selection included brainstorming sessions, prototype development and evaluation to better understand their requirements when used in the "intelligent classroom". Designers greatly benefit from composite widgets, since (i) they have at their disposal a set of usable user interfaces based on validated design decisions, (ii) they can use a set fully functional building blocks that also encapsulate their business logic, and (iii) they can expand the collection by introducing their own composite widgets.

Table 1. Basic and Composite Widgets

Basic Widgets	Button, ToggleButton, ImageButton, Image, Icon, ScrollViewer, Slider, TextBlock, TextArea, Table, RadioButton, CheckBox, ComboBox
Composite Widgets	Menu, Tabs, Paging, ZoomableImage, VideoPlayer, ImageViewer, VideoViewer, GraphViewer, MultipleChoice, HintDisplayer, InformationArea, BookReader

4.1 Alternative designs and adaptation facilities

Application migration from one artifact to another fires a number of changes to ensure usability. The major change concerns the application's layout, and in particular (i) the size and position of the main content, (ii) the position of tab and menu items and finally (iii) the activation / deactivation of secondary information areas when free space becomes available. These attributes are encoded into alternative layout schemes designed with respect to each artifact's unique characteristics (e.g., interaction style, screen size, etc.). The system instantiates the appropriate alternative based on the needs of the current artifact and the designer's choices.

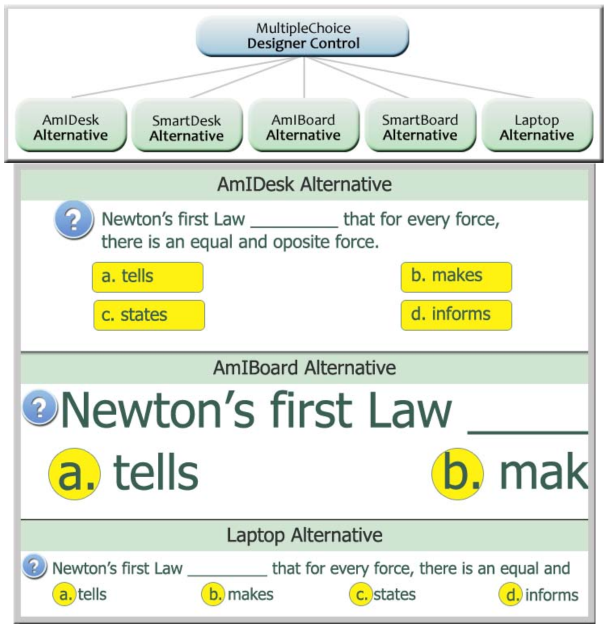


Figure 2. A Designer Control and its alternative views

Nevertheless, transforming the general layout of the application is not sufficient to fully support its migration. As already mentioned, a key feature of this framework is that every contained widget can adapt itself for optimal display according to the artifact's characteristics. For that to be achieved, every widget encapsulates a set of UI alternatives of which the most appropriate is selected at runtime, while for simplicity the designer can use each widget's lightweight instance, the Designer Control. Figure 2 illustrates the benefits of using alternatives designs to deliver controls for solving multiple-choice quizzes in diverse artifacts. Each alternative aims to address artifact-specific problems; for instance, larger font-sizes are used for board artifacts or artifacts with low clarity, larger interactive controls for touch-enabled artifacts where input is not pixel-perfect, and finally foldable layout is used for artifacts with limited display areas.

The first step towards designing each widget's alternatives was to understand the end users' needs using a User-Centered Design (UCD) process. The UCD process was essential to concentrate on the students' needs, wants and limitations. To this purpose, students were actively involved in the design process [19]. TThe feedback coming from student interviews and the results of a formative evaluation conducted on high fidelity interactive prototypes provided great insights on how to better satisfy the users' needs. Likewise, the next step of the design process was to analyze and understand each artifact's unique characteristics. Keeping in mind the variety of screen sizes, interaction techniques and the artifacts' physical position, the alternatives were appropriately designed by incorporating standard usability design practices.

The conducted analysis revealed that a key aspect was widgets sizing. Considering that a designer builds an application interface with an abstract display in mind, appropriate sizing ensures that upon migration all UI elements are clearly visible and legible from an individual student or the entire classroom. The sizing methodology was defined through empirical experiments, where experts evaluated various widgets in various classroom artifacts, in order to determine the minimum sizes of each widget when displayed on every artifact. These minimum values are combined at runtime with designers' size-related choices to proportionally scale every widget for optimal display (e.g., when a designer places a header-like label on the desk, it should look similar on the board as well).

As regards contextual changes that may force widgets to undergo supplementary adaptation to adjust to the new state (e.g., a student activates a sidebar with additional information which eventually reduces the available space), every primitive widget can re-scale until its minimum / maximum limits are reached, whereas every composite widget may have to completely reorganize its layout. In general, each widget is equipped with an internal set of rules that guide scaling and reorganization actions.

4.2 The Designer Control Element

GUI toolkits have gained popularity through the years due to their ease of use. Designers can build their applications using various authoring tools that simplify the placement and customization of such widgets through drag-n-drop facilities. That particular advantage had to be preserved to ensure the library's acceptance. At design time, every widget is accessible through the respective "Designer Control" which is a WPF-compliant UserControl that exposes a number of properties, allowing the designer to (i) use it as a native widget and (ii) customize certain attributes that do not put usability in danger (access is limited to particular attributes only). For instance, the foreground color of a complex radio button (consisting of an interactive control and label) might be altered, but the label must be always placed on the right hand side to facilitate the scanning process.

At runtime, the system instantiates the appropriate widget alternative using the Factory design pattern [15]. Each Designer Control implements a method that given the adaptation rules and the current context of use (AmIDesk, Netbook, SmartBoard, etc.), selects and instantiates the appropriate widget alternative.

4.3 Layout customization

The designer can customize various aspects of an application layout (i.e., menu orientation, tab orientation and style, etc.); however, device limitations may lead to its modification at run time. A relevant example of layout adaptation that is motivated by the artifact's physical position may refer to the placement of interactive navigation items (e.g., tabs and menus). For instance, consider a designer who selects to place the menu bar horizontally above the main content. The system will respect that choice in every artifact that is compatible with it, whereas it will automatically override it when a violation occurs (e.g., in a board artifact the menu will maintain its horizontal orientation but it will be placed below the main content to be reachable by younger students).

5. BUILDING AN APPLICATION

An application usually requires more than one screen to present its data and its interactive controls. For instance, consider a "dictionary companion" application for the "intelligent classroom". Through that application a student can find additional information for a particular word such as: (i) definition(s), synonyms, antonyms, and examples of use, (ii) informative photos and (iii) explanatory videos.

Apparently, if this amount of heterogeneous data were displayed outright, it would be overwhelming for a student to view. For that to be addressed, a UI designer would introduce additional screens, logically divided, to display relevant data. Using the developed tool, the optimal categorization divides the textual from the multimedia information and displays them individually in different interfaces. In more details, the textual information is displayed sequentially in a single UI, whereas the multimedia information is further divided into separate UIs that display either photos or videos.

Traditionally, designers build each UI separately and then developers merge them manually into a concrete application by interpreting the implicit relations. The proposed framework automates this time-consuming task by (i) forcing designers to explicitly define the hierarchy of user interfaces at design time and (ii) automatically generating and linking the separate concrete screens at runtime. For that to be achieved, the application UI is defined following a particular structure, which instructs the system on how to compose the application instance.

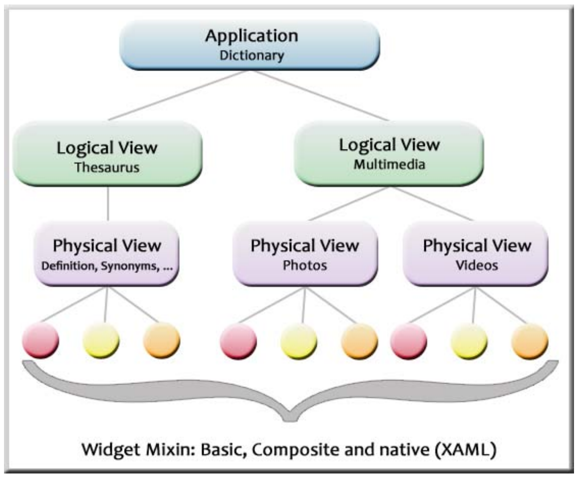


Figure 3. Application tree for a Dictionary application

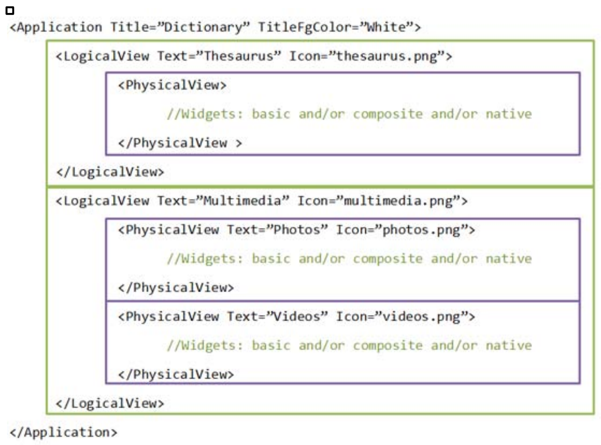
Every application defines its skeleton that includes its Logical and Physical Views (Figure 3). A Physical View is a collection of UI elements that is presented to the final user, while a Logical View refers to the aggregation of various Physical Views of the same Logical category. The following XAML elements are at the designer's disposal to build the application tree:

- **Application:** represents the root of the Application Tree hierarchy and defines the application (e.g., the Dictionary Companion application). It contains one or more LogicalView elements. Through it the designer can customize the basic layout properties (e.g. menu orientation, tab orientation and style, etc.).
- **LogicalView:** denotes a distinct category of an application (e.g., Thesaurus, Multimedia) and contains one or more PhysicalView elements.
- **PhysicalView:** denotes a subcategory of its parent LogicalView (e.g., Multimedia-Photos, Multimedia-Videos). This element defines its layout and holds the UI elements (widgets) that form the user interface; each PhysicalView element is the container of a unique application screen.
- **Widgets:** graphical components that form the user interface. The available widgets are not limited to the self-adaptive primitive and composite ones offered by the presented framework, but include native WPF UI elements as well.
- **Group/BreakLine**: special purpose elements that a designer can use to designate which widgets must always be placed on the same or separate lines respectively.

6. DEPLOYMENT

The application tree (i.e., Application, Logical Views and Physical Views) contributes in the formation of the final UI. The Application element is scanned first and the application layout is formed according to the designer's choices and the current artifact's characteristics. The LogicalView and PhysicalView elements are used to create the navigation routes of the application, while the widgets of each PhysicalView element define the content of each screen.

Navigation routes are automatically generated at runtime as the available structure (Figure 4) facilitates the formation of a logical hierarchy of UIs by linking together the various screens. Through a series of tabs and menu items, students are able to explore the available paths and access assistive material without complex thinking that could disorient and distract them. Designers' explicit annotations define a two-tier structure where the first layer consists of the Logical Views encoded as tab items, while the second layer consists of the Physical Views encoded as menu items that bring their content to the front when activated.



```
<Application Title="Dictionary" TitleFgColor="White">
        <LogicalView Text="Thesaurus" Icon="thesaurus.png">
                <PhysicalView>
                        //Widgets: basic and/or composite and/or native
                </PhysicalView >
        </LogicalView>
        <LogicalView Text="Multimedia" Icon="multimedia.png">
                <PhysicalView Text="Photos" Icon="photos.png">
                        //Widgets: basic and/or composite and/or native
                </PhysicalView>
                <PhysicalView Text="Videos" Icon="videos.png">
                        //Widgets: basic and/or composite and/or native
                </PhysicalView>
        </LogicalView>
</Application>
```

Figure 4. Outline of a sample application tree

Finally, since the navigation options are dynamically generated based on the application's structure, a flexible mechanism was built to update the user interface when a new route is selected. It is based on the Command design pattern where each menu option issues a command to denote that a different screen should come to the foreground and the application controller is responsible to execute it.

An indicative example of that structure, as extracted from the "Dictionary Companion", is the following. The Logical Views "Thesaurus" and "Multimedia" generate two main tabs; the "Thesaurus" tab does not a need a menu for its single Physical View, while on the other the "Multimedia" tab includes two menu items, namely "Photos" and "Videos".

7. DEVELOPED APPLICATIONS

The presented framework was used to build the following educational applications in order to evaluate the UI collection itself and the adaptation mechanisms in general.

- The ClassBook application electronically augments the pages of a physical book. The images and exercises displayed on any page are selectable, and when selected the appropriate applications are launched.
- The Multimedia application displays multimedia content (e.g., images and videos) about preselected topics.
- The Dictionary application displays both textual (e.g., definition, synonyms, etc.) and multimedia information related to a specific word or phrase.
- The Multiple-Choice Exercise application is the electronic representation of a multiple-choice quiz through which a student can solve the exercise by selecting one of the possible answers. A hint button is offered next to each sentence, in case the student needs help to find the correct answer.
- The Hint application is launched when the student explicitly asks for help about a specific exercise. It supports three kinds of hints that are presented gradually to assist the development of critical thinking skills.

Using the aforementioned applications, a heuristic evaluation was conducted in order to eliminate serious usability errors. Five evaluators interacted with the applications on a SmartDesk while an observer was noting down the discovered issues. Their findings were ranked according to their severity rating and were gradually resolved.

8. USER-BASED EVALUATION

In order to examine the efficiency of the proposed methodology for designing pervasive educational applications, a user-based evaluation was conducted. Five designers were chosen to participate in the evaluation process. Only three of them had prior experience in XAML programming. The evaluation process took place individually for each user inside an office room and two evaluators were present, one for assisting the user if needed and one for taking notes.

At the beginning of each session, the user was firstly debriefed about the evaluated framework and then provided with examples on how to build the skeleton of an application and populate it with UI elements. Having a printed document with all the available widgets, the user was offered ten minutes to freely experiment with the framework elements and XAML itself. The next step included the composition of a simple interface in order to observe if the proposed tools (i.e., basic and composite widgets, application tree elements, etc.) were intuitive enough so that designers would easily realize the given application mockup. Throughout the process, the users were asked to think aloud in order to better capture their thoughts and reactions. Finally, a set of 12 questions - using a Likert scale from 5 (Definitely Yes) to 1 (Definitely Not) - were handed to the designers. Once they filled-in the questionnaire, they were debriefed and asked for any additional comments, suggestions, as well as their overall impressions.

Both the questionnaires and the notes taken were analyzed and further studied to define the benefits and drawbacks of the evaluated framework. Generally, through the evaluation process useful feedback was received; results were very positive (Figure 5) and the identified problems and limitations were resolved immediately.

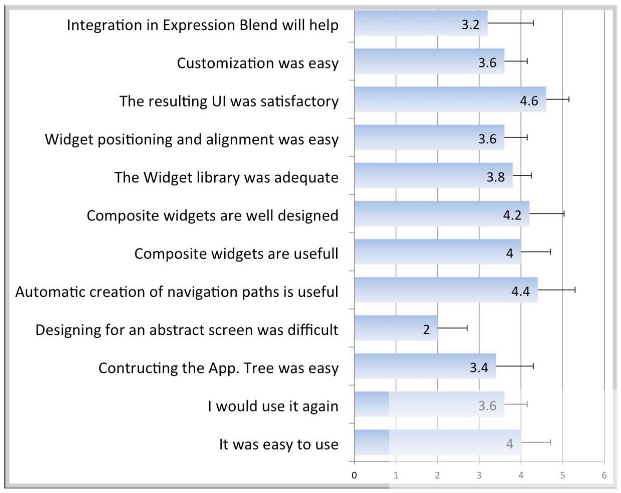


Figure 5. Questionnaire results

The participants appreciated the integration of composite widgets that can be reused as modules to build new applications and the ability to add their own custom-made components. Furthermore, they were excited to see their application's adaptations in action when migrating from one artifact to another and the automatic linkage between the various screens that implicitly generates the application's navigation paths. Finally, the participants appreciated that they could develop a PUPIL-enabled application using the Microsoft Visual Studio IDE, whereas the most experienced ones asked if the framework could be integrated in Microsoft Blend to further simplify the design process.

The findings regarding the functionality and usability of the proposed framework are summarized below:

- The names of the designer customizable properties should become more intuitive to convey their purpose.
- Customizable properties should be accompanied by suggested values that the IDE could autocomplete (e.g., Color, Font Sizes, etc.), while the mandatory properties should be filled with default values.
- Video Viewer and Image Viewer should explicitly require input regarding the maximization policy of their content (i.e., whether the videos / pictures can be maximized or not).
- The Multiple Choice Widget should require input that toggles the visibility status of the hint activator.
- Textblock should include support for typographic features (e.g., bold, italic and underline tags).
- A List (both ordered and unordered) and a Spinner widget should be included in the framework.

9. CONCLUSIONS AND FUTURE WORK

The main objective of this work was to lay the foundation towards the development of user interfaces for the various artifacts, which may comprise the technologically enhanced classroom environment. Using the proposed framework, the designers can easily create applications that can be optimally displayed at the student's desk, student's Netbook or the classroom board. This requires minimum effort on their behalf, and only a single interface implementation is needed. Such interface is able to transform appropriately according to the characteristics of the available artifact. Furthermore, the composite widgets promote reusability of common interface patterns and minimize the designers' work, since numerous design decisions have been made a priori.

Hereafter, additional steps should be taken to fully support the initial concept. The Widget Library needs to be enriched with both basic and composite widgets in order to fully-support the development of educational applications. For that to be achieved it is important: (i) to update the framework based on the evaluation findings and (ii) to identify additional common interface patterns that could be realized into complex widgets.

Finally, considering the increasing number of students that use smart phones / tablets on a daily basis, such devices should be incorporated into the supported artifacts list and in parallel integrate the framework elements in Microsoft Blend to support rapid UI prototyping and development for new artifacts.

ACKNOWLEDGMENTS

This work is supported by the Foundation for Research and Technology Hellas – Institute of Computer Science (FORTH – ICS) internal RTD Programme 'Ambient Intelligence and Smart Environments'.

10. REFERENCES

1. Abrams, M. and Phanouriou, C. UIML: An XML Language for Building Device-Independent User Interfaces. In Proc. XML'99 (1999).
2. Ali, M. F., Perez-Quinones, A. M., Abrams, M. Building Multi-Platform User Interfaces with UIML. In Seffah, A. and Javahery, H (Eds.) Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces John Wiley & Sons, Ltd, Chichester, UK, 2005.
3. Antona, M., Leonidis, A., Margetis, G., Korozi, M., Ntoa, S., and Stephanidis, C. A Student-Centric Intelligent Classroom. In Proc. AMI 2011, (2011, to appear).
4. Antona, M., Margetis, G., Ntoa, S., Leonidis, A., Korozi, M., Paparoulis, G. & Stephanidis, C.: Ambient Intelligence in the classroom: an augmented school desk. In Proc. AHFE 2010, (2010).
5. Bandelloni, R., and Paterno, F. Flexible interface migration. In Proc. IUI '04, ACM Press (2004), 148-155
6. Benyon, D., Murray, D. Adaptive Systems: from intelligent tutoring to autonomous agents, Knowledge-Based Systems, 6, 4(1993), 197-219.

7. Bharat, K., Cardelli, L. Migratory applications. In: Vitek J, Tschudin C (eds.) Mobile object systems: towards the programmable internet. Springer, Berlin Heidelberg New York, 1995, 131–148.
8. Calvary, G., Coutaz, J., Thevenin, D., Limbourg, Q., Bouillon, L., Vanderdonckt, J. A Unifying Reference Framework for multi-target user interfaces. In Interacting with Computers, 15, 3 (2003), 289-308.
9. Collignon, B., Vanderdonckt, J., and Calvary, G. An intelligent editor for multi-presentation user interfaces. In Proc. SAC '08, ACM (2008), 1634-1641.
10. Cook, D. J., Augusto, J. C., Jakkula, V. R. Ambient intelligence: Technologies, applications, and opportunities. Pervasive and Mobile Computing, 5, 4 (2009), 277-298.
11. Cook, D. J., Das, S.K. (2007). How smart are our environments? An updated look at the state of the art, Journal of Pervasive and Mobile Computing 3, 2 (2007), 53-73.
12. Cooperstock, J. Classroom of the Future: Enhancing Education through Augmented reallity. In Proc. HCI Int'l 2001, (2001), 688–692.
13. Coyette, A., and Vanderdonckt, J. A Sketching Tool for Designing Anyuser, Anyplatform, Anywhere User Interfaces. In Proc. Interact'2005, (2005), 550-564.
14. Gajos, K., and Weld, D., S. SUPPLE: automatically generating user interfaces. In Proc. IUI '04, ACM (2004), 93-100.
15. Gamma, E., Helm, R., Johnson, R., & Vlissides, J. Design patterns: elements of reusable object-oriented software, 1994.
16. Kleinrock, L. Nomadicity: anytime, anywhere in a disconnected world. Mobile Network Applications, 1, 4 (1996), 351-357.
17. Korozi, M., Leonidis, A., Margetis, G., & Stephanidis, C. MAID: a Multi-platform Accessible Interface Design Framework. In Proc. HCI Int'l 2009, Berlin Heidelberg: Lecture Notes in Computer Science Series of Springer, 7 (2009), 725-734.
18. Korozi, M., Ntoa, S., Antona, M., & Stephanidis, C. (2011). Intelligent Working Environments for the Ambient Classroom. In Proc. HCI Int'l 2011, (2011).
19. Korozi, M: PUPIL: pervasive UI development for the ambient classroom (Master's thesis), (2010). http://elocus.lib.uoc.gr/dlib/a/e/2/metadata-dlib-81a07682706c2163d8f582245fd9edfd_1288689489.tkl
20. Leonidis, A., Antona, M., & Stephanidis, C. Rapid Prototyping of Adaptable User Interfaces. International Journal of Human-Computer Interaction, (2011, to appear).
21. Leonidis A: ClassMATE: Classroom Multiplatform Augmented Technology Environment (Master's thesis), (2010). http://elocus.lib.uoc.gr/dlib/d/3/6/metadata-dlib-d19ded9b6c0938f4723672b56b78ebe2_12885980 57.tkl
22. Lin, J., and Landay, A. J. Damask: A Tool for Early-Stage Design and Prototyping of Multi-Device User Interfaces. In Proc. DMS' 2002, (2002), 573-580.
23. Meskens, J., Vermeulen, J., Luyten, K., and Coninx, K. (2008). Gummy for multi-platform user interface designs: shape me, multiply me, fix me, use me. In Proc. AVI '08, ACM (2008), 233-240.
24. Microsoft. White Paper: An Overview of Microsoft Visual Studio 2008. http://www.microsoft.com/downloads/en/details.aspx?FamilyID=17319eb4-299c-43b8-a360-a1c2bd6a421b.
25. Mori, G., Paterno, F., and Santoro, C. Tool support for designing nomadic applications. In Proc. IUI '03, ACM (2003), 141-148.
26. NetBeans. NetBeans IDE 6.9.1 Release Information. http://netbeans.org/community/releases/69/.
27. Paterno, F., Santoro, C., Mantyjarvi, J., Mori, G. and Sansone, S. Authoring pervasive multimodal user interfaces. International Journal of Web Engineering and Technology, 4, 2 (2008), 235-261.
28. Puerta, A., Eisenstein, J. Towards a general computational framework for model-based interface development systems, Knowledge-Based Systems, 12, 8 (1999), 433-442.
29. Puerta, A., Eisenstein, J. XIML: a common representation for interaction data. In Proc. IUI '02, ACM (2002), 214-215.
30. Puerta, A., Eisenstein, J. XIML: A Multiple User Interface Representation Framework for Industry. In Seffah, A. and Javahery, H (Eds.) Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces John Wiley & Sons, Ltd, Chichester, UK, 2005.
31. Savidis, A., & Stephanidis, C. Unified User Interface Development: Software Engineering of Universally Accessible Interactions. Universal Access in the Information Society, 3, 3 (2004), 165-193.
32. Seffah, A. and Javahery, H. Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces. In Seffah, A. and Javahery, H (Eds.) Multiple User Interfaces: Cross-Platform Applications and Context-Aware Interfaces John Wiley & Sons, Ltd, Chichester, UK, 2005.
33. Shi, Y., Xie, W., Xu, G., Shi, R., Chen, E., Mao, Y. and Liu, F.: The smart classroom: Merging technologies for seamless tele-education, IEEE Pervasive Computing (2003), 47-55.
34. Stephanidis, C. Towards User Interfaces for All: Some Critical Issues. In Proc. HCI Int'l '95, (1995), 137-143.
35. Stephanidis, C., Antona, M., Savidis, A., Partarakis, N., Doulgeraki, K., Leonidis A. Design for All: Computer Assisted Design of User Interface Adaptation. In G. Salvendy (Ed.), Handbook of Human Factors and Ergonomics (4th Edition). USA: John Wiley and Sons, (2012, to apper).
36. Thevenin, D. and Coutaz, J. Plasticity of User Interfaces: Framework and Research Agenda. In Sasse, A. and Johnson, C. (Eds.) In Proc. INTERACT '99, (1999).
37. Vogel, L. Eclipse IDE Tutorial, Version 1.7, (2011). http://www.vogella.de/articles/Eclipse/article.html
38. Xu, P., Han, G., Li, W., Wu, Z., & Zhou, M. Towards Intelligent Interaction in Classroom. LNCS, 5616 (2009), 150-156.
39. Yau, S., Gupta S., Karim, F., Ahamed, S., Wang, Y., and Wang, B. Smart Classroom: Enhancing Collaborative Learning Using Pervasive Computing Technology. In ASEE 2003, (2003), 1-9.