Creation and Visualization of Context Aware Augmented Reality Interfaces

Allan Oliveira and Regina B. Araujo
Computer Department Federal
University of São Carlos
Washington Luis Highway, km 235,
São Carlos, São Paulo, Brazil
55 16 3351-8232
{allan_oliveira, regina} dc.ufscar.br

ABSTRACT

Augmented Reality User Interface (AR UI) is a growing research area, which faces many design and implementation challenges. The design of a proper application ARUI is a process that demands time and expertise in the AR field. These interfaces are projected on the real world, complementing it, while providing useful information for the user’s goal. The implementation of an AR UI is a time consuming task and current solutions, using interface components, lack the variety (and customization) of interface elements to create different types of applications. This paper presents a pattern-driven AR interface visualization framework (VISAR), which allows the development of context-aware AR interfaces using interface components (patterns). These patterns accelerate implementation and design time by offering an interface element for every user task. They also enable the design of adaptive interfaces, and provide share and reuse of interfaces’ components. The paper also presents an AR Interface Editor (VISAR IE) that supports interface design by defining adaptation rules and offering interface patterns choices.

Categories and Subject Descriptors

H.5.1 [Multimedia Information Systems]: Artificial, augmented and virtual realities—User Interfaces; H.5.2 [User Interfaces]: Graphical user interfaces (GUI)—Standardization—User interface management systems (UIMS);

General Terms

Design, Standardization.

Keywords

Augmented Reality, Adaptive User Interface, Context aware AR Interfaces.

1. INTRODUCTION

Augmented Reality (AR) is a research area in which synthetic computer-generated objects supplement or compose real environments, providing users with a more resourceful tool to visualize their surroundings and related information without space or mobility constraints.

Augmented Reality is increasingly used as part of applications in a number of sectors spanning medicine, industry, advertisement, robotic and entertainment. Nevertheless, there are still challenges for bringing AR to users’ daily reality.

One of the challenges is the design and implementation of AR User Interfaces (AR UI), which is our focus in this paper. The design of proper applications UI is a process that demands time and expertise in the AR field. These interfaces project on the real world, complementing it, while providing users with useful information for their goals.

The implementation of AR UI is a time consuming task when done in a low level of programming. Current solutions for high-level development, using interface components, lack the variety (and customization) of interface elements to create different types of applications.

Therefore, standard tools and design methods from Human-Computer Interaction (HCI) area have to be either complemented or adapted to support AR UI design. Few works have been done in this particular area [1], compared to the basic areas of AR (tracking, displays and devices [2]).

In this paper, we introduce VISAR, an AR interface visualization framework that assists AR developers in the creation of user interfaces. Interface design patterns developed for AR applications are used targeting intelligent buildings (environments with the necessary hardware for tracking the user and sensing the ambient). VISAR also builds a connection between UI and contexts, allowing adaptation of the UI according to the user’s current situation at run time what helps filtering and so reducing the information to the interface.

In addition, we introduce the VISAR Interface Editor (VISAR IE), which assists developers to model AR interfaces based on interface patterns and context-based adaptation rules. Experts of different application areas (firefighters commanders, industry safety managers, commercial and industrial managers, etc), can use this interface editor, which requires only an understanding of modeling tools (flowchart modeling, mind map modeling) and AR interfaces (no programming language expertise is necessary).

Thus, the editor supports designing the UI and its behavior, and the framework implements and runs (in real-time) this interface, while being controlled by an application.

This paper is organized as follows: Section 2 discusses related works. VISAR is introduced in section 3. VISAR IE (Interface Editor) is described in Section 4. An application example is shown in Section 5, followed by Conclusions (Section 6) and Bibliographical References.

2. RELATED WORK

Regarding development, there are two mainstream approaches in the AR user interface area to assist developers in creating their UI
and applications, which are: (1) frameworks; and (2) authoring tools and description languages.

Frameworks try to cover the whole implementation process of AR applications. However, they focus more on the application logic than UI, requiring developers to understand and implement in a low level programming, demanding more time, but allowing the UI to be controlled and designed in detail. The best-known examples are ARToolkit [3] and Studierstube [4].

There are also several frameworks from commercial companies, such as Juniao, an augmented reality browser by Metaio; Umifeye, an AR SDK also by Metaio; Qualcomm AR SDK; Layar, an AR browser with creates different layers of visualization. On the one hand, these frameworks offer several tracking solutions (tracking images, objects), especially for mobiles and handholds, with an easy but powerful API to program, and targeting web content (especially social networks).

On the other hand, authoring tools offer a certain amount of implemented behaviors and interface components, which the developers can, visually program applications. Some tools use description languages, which are high-level description languages to describe the UI in text files. AR authoring goal is the creation and edition of AR content. The most significant difficulty in the authoring area is to create editing tools to develop complex applications fast, as well as tools to be used by non-experts in computer science to develop AR content.

MacIntyre et al. [5] introduce DART (Designer AR Toolkit). Built upon Macromedia Director, this tool permits creating AR applications by using visual design and scripting. It also provides several predefined interface behaviors, low-level support for trackers, sensors and cameras, and accepts the creation of virtual reality applications.

An example of description language is APRIL [6], for authoring AR presentations, which is part of the Studierstube system [4]. It allows developers to create presentations, from interface to behavior (including interactions), while hiding the implementation complexity.

DART and APRIL offer very basic AR UI elements, lacking a bigger range of interface components. Moreover, extending the created behaviors and components is not a simple task.

None of the works above uses the interface pattern concept. Only a few authoring tools use components and these do not include 2D and 3D components. In addition, developers cannot build applications beyond presentation (like APRIL). Our work VISAR offers a solution for UI design and implementation providing interface patterns and a modeling tool, while still keeping developers with full control of their applications.

3. GENERAL VIEW OF VISAR

VISAR framework main goal is to make AR UI easier to develop/implement independent of the application. To accomplish this, interface patterns are used. AR systems tend to use UI elements with common characteristics, independent of the application area, just like WEB and desktop systems do. These elements are interface patterns, i.e., interface components built from a known and embraced solution for a common problem; they express an accepted approach to display information [7]. The most common example of interface pattern in AR is the displaying of a 3D object on top of a marker, when the marker is visible by the camera.

Applications using VISAR control the patterns in real time, filling them with the necessary information displayed to the user. Each pattern has its application-programming interface (API), which are functions the application can use to manage them, by controlling the information displayed or the pattern position in the scene.

As a complement to VISAR, the interface editor (VISAR IE) allows developers to model the UI and its rules by choosing which patterns to show depending on the users’ context (any information used to characterize the user’s situation [8]).

VISAR IE also saves the design result in an XML file that describes the interface. VISAR then reads this file to load the interface structure.

3.1 VISAR Framework Operation

The application execution starts by loading to VISAR the XML created by VISAR IE - it contains the description of the UI (its contexts and patterns). Then the application sends the name of the device the UI displays and VISAR begins by rendering the patterns from the initial context, and proceeds supporting the application with functions to control the user interface (API).

There are two types of messages for the application to send: (1) context activation and deactivation; and (2) alteration in a pattern content (updating information in the UI).

For (1), all the active context have their patterns rendered, therefore activating contexts make VISAR render their patterns, while deactivating make the opposite. Therefore, it is possible to change completely the interface in real time, promptly and easily, if the interface design is well done in terms of projecting what has to be shown to the user during each context.

For (2), the modification in the content of a pattern happens when the application requires updating information value in the UI. Figure 1 shows an example, in which the application detects the temperature increasing from 60° to 85°. It then notifies the framework that updates the UI (right screen in Figure 2).

An example of a simple application built upon VISAR is a system for firefighters in an emergency environment. The system begins at the “NORMAL” context, with the displaying of information such as temperature, remaining oxygen and the room floor plan. Then the user (firefighter) enters in a completely dark place and the application notifies VISAR to activate the “DARK” context. Now, besides the patterns of “NORMAL”, a night vision camera image appears on the screen. When the temperature rises from 60° to 85°, the application informs VISAR of this new value, and requires painting one more bar in the pattern. Figure 1 shows this example, in which the interface patterns are enlarged for a better view, and the interface structure can be seen in figure 2.

VISAR has a data structure for modeling the interface, which is a list of contexts, with each context having its own list of patterns. The context list contains every adaptation situation modeled for the system.

3.2 VISAR Interface Patterns

By using patterns for AR interfaces, it is easy to create or reuse interfaces for an application, and control them during execution, making the interface design and development processes shorter.
Interface structure model for a firefighter simple example.

Figure 1: Example of an interface created by VISAR, in real time. In the left screen, “NORMAL” context is active; in the right screen, “DARK” context is also active.

Figure 2: Interface structure model for a firefighter simple example.

Figure 3: Examples of Interface Patterns.
In addition, new patterns can be created and added to VISAR, increasing the range of applications it can cover.

Interface patterns can be of four different types. These types make it easy to understand the requirements and logic of each pattern - they facilitate the building of new patterns through inheritance and help to specify the tracking conditions of patterns. The description of these patterns and corresponding properties are as follows:

- **Static Normal**: represents patterns that have fixed position in relation to the user, normally attached to the HUD. Their properties are screen position (display), and size and distance (depth) between the pattern and the HUD;

- **Static Tracked**: also represents patterns with fixed position in relation to the user. However, it is necessary a tracking system, because they use both information about the user and position of objects of interest. Their properties are screen position (display), and size and distance (depth) between the pattern and the HUD;

- **Dynamic Tracked**: these patterns require a localization system to track the user and every object of interest. This type of patterns represents virtual objects inserted in the real environment. Their properties are: the position in the ambient (X, Y, Z) and size;

- **Dynamic with Vision**: represents dynamic patterns based on computer vision for tracking (image recognition). Properties are the image (and their properties) and the 3D objects size.

Figure 3 shows some examples of patterns:

a) **Virtual Label**: a dynamic tracked pattern, it shows a label with information on some target (near or on top of the target). Its API functions are set the pose (6DOF) of the label, set the text in the label;

b) **Arrow**: a static tracked pattern, it shows a 3D arrow to guide the user in relation to the direction of a destiny point. The pattern gets the user position and the destination position, and then calculates where the arrow has to point to (automatically and in real time). Its API has just one function, to set up the destination position;

c) **Pathfinder**: a dynamic tracked pattern, it renders a virtual path throughout the user route. This path is constructed and calculated based upon the 3D model (CAD) of the building - the virtual path has to continuously go to the user position until the next corner or door (so that the path is not rendered in places the user cannot see to avoid displaying unnecessary information). Its API has just one function, to set up the destination point;

d) **Virtual marker**: the most common pattern for AR, it renders a 3D object on the top of a marker. Its API functions are set the marker image and set the position of the virtual object in relation to the marker.

Using these concepts, any new graphical component necessary to a common problem can become an interface pattern. If developed in a standard structure (like our interface pattern model), this component can be reused, saving development time.

4. **VISAR INTERFACE EDITOR**
The VISAR Interface Editor is an application for AR UI modeling, built to work in conjunction with VISAR.

An expert in the application area can do the design, because it does not require programming skills. Rather, it requires a basic understanding of modeling tools (like flowchart or mind map), knowledge about the AR interface patterns (not how they work, but what they show), and planning well the interface (user requirements).

Figure 4 shows the editor general view (the image is edited for better visualization), which can be divided in four sections:

- **Section (1)** presents the pattern palette, with are all interface patterns.

- **Section (2)** shows the current interface model with contexts and patterns.

- **Section (3)** represents the user display. It is possible to enter the display size (and resolution) in the properties (a cellular in the example given). Here the designer can position the 2D interface patterns indicating where the patterns will be rendered in the user display.

- **Section (4)** displays the properties of the selected pattern (each interface pattern has its own properties, which can be
altered here). The properties here refer to the display, not the pattern.

VISAR and VISAR IE exchange information via XML documents. After designing and configuring the UI in the editor, this generates an XML, which describes the UI and all the information configured in the editor that is important to VISAR. VISAR then reads this XML document and saves this data in its internal structure (fig. 2). It then renders the user interface in the display and waits for both: the call of functions and updates from the application.

5. APPLICATION EXAMPLE

5.1 AR System for firefighters

This is an application intended for firefighters in an emergency environment, such as firefighting, rescuing, etc. Because several contexts can be involved in such applications, the interface is modeled to help in each of these contexts.

![Figure 4: VISAR Interface Editor.](image)

- Enter the building
- Go to the second floor
1. Go to the room (Open the door)
2. Rescue victims

![Figure 5: AR System for firefighter in an emergency environment.](image)

For example, if a fire is occurring in a closed room, the door cannot be suddenly opened, because in this case too much oxygen would go into the room and an explosion would occur (back draft).

Figure 5 shows a part of the interface for this application. Several interface patterns are in this figure, but not all of them have to appear at the same time, and others are not present. The interface patterns found in Figure 5 include: a list with completed tasks, a list with tasks to do, a night vision camera (turned on only if needed, a 2D map of the ambient, showing the fire location and also others firefighters location, the oxygen and temperature values, the current time, the connectivity of the equipment with a wireless network (that feeds the system with all sorts of data), a virtual path showing the destination route, an “X” in a door that must not be opened, and a 3D representation of persons (in this case victims) in the ambient and their localization.

6. CONCLUSION

A myriad of applications is increasingly using Augmented Reality (AR). As visualization devices get smaller, faster and more mobile, current challenges of software and hardware are being overcome.

An important issue with AR is the difficulty to construct AR user interfaces, a time consuming task. AR UI must be ready to adapt in response to different contexts, in real time, and must use a mix of 3D and 2D to unleash its full potential.

This paper presents an AR UI framework (VISAR) and joint editor (VISAR IE) for fast development of AR adaptive interfaces. These tools have the following novel features: to accelerate the UI design and implementation process by using interface patterns (allows to create interfaces for a variety of applications); to facilitate reuse of interfaces; to allow the modeling of AR interface adaptations that occur in real time, in response to different ambient context modifications; to make possible for experts of an application domain to model different AR user interfaces easily, without the need of computer programming skills (makes easier the cooperation between designer and user).

The presented tools (framework and editor) are a presentation/visualization layer for AR applications that can assist developers in the process of AR design and implementation.

7. ACKNOWLEDGMENTS

The authors wish to thank CNPq and FAPESP for the support to the INCT-SEC Project, processes 573963/2008-8 and 08/57870-9.

8. REFERENCES


