ARUIML: A Three-dimensional User Interface Description Language for Augmented Reality

Mikel Salazar Gonzalez [mikel.salazar@deusto.es]
Carlos Laorden Gómez [claorden@deusto.es]
DeustoTech Computing, University of Deusto, Spain

Abstract

In this position paper, we present an initial draft of a new user interface markup language specifically designed for Augmented Reality systems, ARUIML. This new language enables developers to define, in a simple and easy manner, meaningful user experiences seamlessly embedded into the real world. To achieve this goal, we also define a theoretical model that provides the foundation for novel interaction techniques that take into account the context of the user to adapt the user interface to every aspect of the daily life.

1. Introduction

As the Augmented Reality Markup Language [1][1] continues its journey towards standardization, there are growing concerns that the limitations of this language will lead it to obsolescence even before it has the time to grow. And the main problem is not that it is being badly constructed –far from it –. It is just that, as basic HTML, it describes the content at such a low level that it cannot be used directly to accommodate meaningful user experiences.

When Tim Berners-Lee envisioned the World Wide Web, little did he know that his HTTP protocol would grow into the most popular software distribution system (after all, he designed it just to transfer hypertext documents). Nowadays, the number of applications that make use of the browser as the main medium to present their user interfaces (usually known as “webapps”) easily exceeds the catalog of all the app stores/markets available in mobile platforms. Yet, in almost all these applications, the HTML document is just a hollow structure. A mere container ready to be filled with the content generated with a JavaScript engine.

This logic disassociation between the concepts of “document” and “user interface” is nothing new. As standardized document formats (such as PDF, ODT or HTML) become more popular, more content creators and advanced users begin to explore what they can do with them. As the expectations of the final users grow, they start demanding functionality beyond the scope of what it can be considered a document: animations, multimedia content, games, etc. And, if the new versions of the standard do not catch up in time, there will be no shortage of middleware developers that will create plugins to fill that void (Macromedia/Adobe Flash player being the prime example of this tendency).

Against this background, in this document we present a three-dimensional user interface markup language specifically designed for Augmented Reality systems. This language aims to provide developers with an easier and more interoperable way to create user interfaces seamlessly integrated into the real world, making the user interaction much more intuitive and engaging.

This document is divided as follows. After introducing the main design principles in section 2, section 3 presents the interaction model, detailing its characteristics, components and distribution architecture. Section 4 describes the features of the user interface markup language, defines the object model and enumerates the top-level elements. Finally, section 5 offers some final thoughts on the current state of the framework and several venues for future work.
2. Design Principles

The interaction model presented in this document is based on the following principles:

2.1. User-Centered Approach

Often excluded completely from the development process, users have little to no control over how they interact with software applications. We want to revert completely this approach by enabling the users to customize the interfaces to their needs and desires. Ultimately, our goal is to break the barrier between developers and final users, allowing everyone to easily create their own user interfaces.

2.2. Context Awareness

In order to achieve a truly mobile AR experience, the system must be able to analyze—and act accordingly with—the task the user is engaged in at any given time. Using the information acquired by the device sensors and the user preferences, the user interfaces should be adapted to the needs of the user in real time.

2.3. Unified Syntax

As any developer of web-based applications knows, having multiple languages for each specific aspect of the user interface may seem beneficial at first (from the standpoint of specialization) but, at the end of the day, it makes learning and appropriately using those languages a very cumbersome task. To abstract themselves from this complexity, many developers ultimately have to rely on third-party frameworks, something that has a very negative effect on the standardization processes of the underlying technologies. For this reason, we have opted for a simple but extensible syntax to specify all aspects of the user experience.

2.4. Layered Complexity

While the KISS (Keep It Simple, Stupid) principle is always a good starting point, as any system incorporates more functionality, it becomes exponentially more difficult to support and upgrade it. Instead of focusing on simplicity just for the sake of it, for the model described in this document, we propose one single hierarchical structure for each user interface. In this way, we can create complex interaction environments without losing the global view of the whole system.

2.5. Logical Consistency

As a holistic system, all the parts of the model should maintain a strong logical consistency, both in terms of coding and in terms of user experience. This is essential to sustain interactions over long periods of time but, more importantly, it allows third-party developers to build their own logical frameworks over the proposed model with a high enough degree of confidence.

2.6. Open Collaboration

Due to hardware limitations, until recently, Human-Computer Interaction has been centered only on single-user approaches. However, current computer networking models make possible the creation of shared environments, where the users can collaboratively interact with the application data. In the proposed user interface description language, developers can take advantage of these new techniques to easily define collaborative edition environments (where even entire widgets can be shared between multiple users).

2.7. Enhanced Security

Beyond protecting the user from cybersecurity threats, any system based on the proposed model should also help the user to avoid potential risks in the physical world (e.g. entering dangerous zones or colliding with elements—including other users—of the surrounding environment).
3. Interaction Model

Before explaining in detail the User Interface Markup Language, it is necessary to define the interaction model on which it is based. This section presents its main characteristics, enumerates its basic components and describes the distribution architecture that ties it all together.

3.1. Main Characteristics

Since human beings are used to operate with the elements of the physical world, the best way to achieve an intuitive and immersive interaction model is by emulating the behaviors of the real-world objects. Although this idea has been present since the inception of HCI as a research field, thanks to the improved hardware capabilities of current AR-enabled systems, we can finally define **physic-based interaction mechanisms** that enable the users to perceive and operate with the virtual objects in a similar manner than they are used to with common objects (e.g. buttons that require a certain amount of pressure to register the action or knobs that allow the selection of a different action depending on the rotation value). The main problem with this approach continues to be the lack of a non-intrusive way to provide tactile feedback for the virtual widgets but this issue can be partially solved by creating tangible user interaction surfaces or by associating certain actions to proprioception-based hand gestures (pinching, extending an specific finger, touching the palm of one hand with a finger of the other, etc.).

Another important aspect of the proposed interaction model is the **adaptability to different situations**. As stated in the previous section, the model takes into account the task the users are performing at any given moment to provide them with only the user interfaces that are meaningful in that context. To achieve this goal, instead of grouping the widgets into convoluted MDI windows, we propose a model based on a **context-aware layer structure**. These **layers**, as their name implies, are grouping entities that are superimposed over a background image (in the case of AR systems, the video feed obtained from the camera/s). Each user interface contains several layers that can be resized, reordered or even removed depending on the associated contextual information.

The implications of using this layer system go beyond providing a more streamlined experience, however. For instance, it reduces the amount of information displayed at a given time, allowing the users to focus their attention on the task at hand.

Another key feature of the layer system is that, due to the lower number of widgets in each layer, **there is no real need to define style properties** in the User Interface markup language. While developers can propose different properties to customize the general appearance of the user interfaces, in last instance, the users have complete control over how each layer (and the internal widgets) is displayed. This approach offers multiple benefits:

- **Consistency**: The user does not need to waste time trying to identify each widget and its functionality. All components behave in a way that the users can predict and, thus, they feel more comfortable with the system and are able to perform their tasks with the minimum amount of resources.

- **Accessibility**: Since the layer structure and contents are automatically adapted to the user preferences, there is no need to create accessible versions of the user interfaces. If the user requires an additional layer (e.g. a virtual keyboard) to interact with the user interface it will be handled by the system itself.

- **Faster and easier development**: Without having to define the style properties, developers can easily define the communication with the final users and focus their efforts on improving other parts of their applications/services.
3.2. Interaction Components

The interaction model can be divided into several components:

- **Users**: Persons that inhabit the physical world. They interact through the client systems and provide the user preferences and the basic situational information (e.g. feelings, needs, intentions, etc.).

- **Client Systems**: The AR-based computer system that each user interacts with (several users can interact through a single device, but for mobility and customization purposes it is highly recommended to use a single client system per user). Either fixed, hand-held, head-mounted or embedded, these devices show the AR environment from the point of view of the user. They can share user interface elements between their peers.

- **Server Systems**: Dedicated computer systems that provide the user Interfaces description files. If necessary, they may also serve as an arbitrator for the communication in collaboration environments (to ensure that the original intent is always preserved).

- **Non-Spatial Layers**: These layers are displayed relative to the point of view of the user, at a –perceived– close distance. The widgets contained within these layers can interact with the elements of the environment but are not actually embedded in it (i.e. they are rendered using the same lighting conditions but they cannot be occluded by real-world objects).

- **Spatial Layers**: These layers contain three dimensional widgets that are integrated into the physical environment. They can be positioned using natural feature recognition systems, fiducial markers or positioning systems.

Figure 1: A representation of a collaborative modeling application showcasing the different interaction components.

3.3. Distribution Architecture

One of the main problems with current Human-Computer Interaction paradigms is the difficulty to find relevant contents for the user within the myriad of options available. In this model, instead of defining a unique software distribution system (such as the App Store), we propose a distributed architecture where the discovery of user interfaces is not based on character strings (URLs, search terms, etc.) but on the situation of the user (geo-location, current task, necessities, feelings, etc.)

In this model, the client system is constantly looking for nearby server systems that offer services and if they meet the criteria set by the user, downloads the associated user interface and displays it. This location-based discovery allows a more direct and focused communication while, at the same time, makes it more secure by reducing the number of intermediaries.
4. User Interface Markup Language

In order to properly implement an interaction model, a language to describe the user interfaces is required. In this section, we present the first draft of our User Interface Markup Language, the Object Model that supports it and several of its top-level elements.

4.1. Basic Concepts

Augmented Reality-based User Interaction Markup Language (ARUIML) is a descriptive, XML-based data format that facilitates the development of the presentation layer of Augmented Reality applications. With a strong user-centered focus, this UTF-8 encoded dialect of XML provides the communication mechanisms to support meaningful and engaging interactions with final users.

Although ARUIML is heavily inspired by other user interface markup languages such as XUL [2], XAML [3] and UsiXML [4], due to the complex and variable characteristics of the real-world environments in which the interaction takes place, there are notable differences with the aforementioned languages:

- **Three-dimensional**: All components of the user interface are defined as entities of a three-dimensional space. Elements with lower number of dimensions (such as pictures or texts) may be reoriented in real-time for better viewing purposes but, ultimately, they are handled as any other object in the scene graph.
- **Spatial Positioning**: Either by using geospatial location or fiducial markers, widgets may be positioned within the surrounding environment. While not applicable to all elements of the user interface, it is recommended to take advantage of this feature whenever possible to create more natural interactions.
- **Occlusion Handling**: Since spatial positioned widgets can become occluded by elements of the environments (or other components of the user interface), ARUIML provides several ways to ensure that any important interaction element is always accessible.
- **Physical Properties**: To further enhance the immersion effect, all components of the user interface have several physical attributes (such as mass, density, hardness or elasticity). In this way, users can predict the behavior of widgets and, thus, interact intuitively with them.

4.2. User Interface Object Model

The User Interface Object Model (UIOM) is a data structure that not only contains the representation of all the user interfaces (present at any given time in a single client system), but also contains all the information about the current context and the user preferences. Completely described in ARUIML, this data structure manages the logical and spatial representation of each and every user interaction component.

This structure greatly facilitates the management of multiple user interfaces and enables the sharing of widgets between several users. Also, by dividing the user interfaces into smaller entities (called layers), it is possible to reorder or hide different interaction components to adapt them to the context of the users in real time, allowing them to focus their attention on the situation at hand.

Another important advantage of using a single Object Model is the possibility to create interoperable interaction methods that go far beyond sharing content via clipboard. Since all the user interfaces use the same widget structure it is relatively easy to share -and extend- their functionality, by incorporating additional layers. For example, it is possible to redirect the output of a calculator application into a text document or a 3D model can be modified without having to save it into a file and open it with a modeling application.

Finally, it is important to understand that the UIOM is a –really– dynamic data structure, in which the user interfaces are loaded and unloaded continuously (in a similar way as web-based applications) so it is essential to take it into account when designing complex user experiences.
4.3. Top-Level Elements

In order to be a well-formed XML file, each ARUIML file must have only one root element. This `aruiml` element contains the basic metadata associated to the file (the `name`, `author` and `lang` attributes) and the links to the XML namespace files (`xmlns` attribute).

This root element can contain the following child elements: `include`, `context`, `widget` and `layer`.

4.3.1. Include

The `include` element provides a way to import the contents of other ARUIML files into the current file. This feature can be controlled using the `filter` attribute to import only elements that match a given criteria.

**Attributes:**
- `source`: The URL of the ARUIML file to import.
- `filter`: (Optional) The criteria which the elements must meet in order to be imported.
- `params`: (Optional) Additional parameters for dynamic content generation.

4.3.2. Context

The `context` element defines a situation condition that influences the behavior of widgets and layers.

**Attributes:**
- `id`: Identifies univocally the context.
- `parent`: (Optional) Defines a parent context.

**Child elements:**
- `location`: Describes a spatial location associated with the context.
- `time`: Describes a time range associated with the context.
- `marker`: Describes a fiducial marker that needs to be recognized.
- `action`: Describes an action performed by the user.
- `status`: Describes a logic state of the associated service.

4.3.3. Widget

The `widget` element defines a reusable user interaction component. This element can be added both as a top level element (specifying the properties of a class), within another `widget` element definition or inside a `layer` element.

**Attributes:**
- `id`: Identifies univocally the widget instance. Cannot be used in conjunction with `class`.
- `class`: Establishes the class of the widget. Cannot be used in conjunction with `id`.
- `base`: (Optional) Specifies another widget class that serves as the template for this one.
- `context`: (Optional) Defines the context in which the widget is available.
- `position`: (Optional) Specifies the preferred positioning method.
- `share`: (Optional) Specifies whether or not the layer is shared between multiple users.

**Child elements:**
- `widget`: Describes an internal user interaction component.
- `shape`: Defines the three-dimensional geometry of the widget.
- `material`: Specifies the physical properties of the widget.
- `animation`: Specifies the animation set of the widget.
- `behavior`: Contains a set of scripts to handle the different events.
4.3.4. Layer
The `layer` element describes a user interface layer, an entity that groups several widgets and gives them a common context.

Attributes:
- `id`: Identifies univocally the layer.
- `service`: Specifies the name and/or URL of the associated service.
- `type`: Indicates whether the layer needs to be spatially positioned or not.
- `context`: The context in which the layer is available.
- `position`: (Optional) Specifies the preferred positioning method.
- `priority`: (Optional) The priority level associated with the layer.
- `share`: (Optional) Specifies whether or not the layer is shared between multiple users.

Child elements:
- `widget`: Describes an internal user interaction component.
- `shape`: Defines the three-dimensional geometry of the layer.
- `material`: Specifies the physical properties of the layer.
- `animation`: Specifies the animation set of the layer.
- `behavior`: Contains a set of scripts to handle the different events.

5. Conclusions and Next Steps
In this document we have presented the initial draft of a markup language that facilitates the development of Augmented Reality applications. Nevertheless, since this language focuses on interaction it provides different ways to integrate current standardization efforts such as ARML [1] or glTF [5] (e.g. to specify better three-dimensional geometry definitions for the widgets or to improve the interaction between nodes of a scenegraph).

Another important venue for future work is the implementation of the proposed interaction model not just for single applications but for the entire operating system. This will allow an even greater interoperability between applications and, ultimately, create a completely mobile Augmented Reality-based user experience.

References
[1] ARML Specification  
[4] UsiXML (USer Interface eXtensible Mark-up Language)  
http://www.usixml.eu/
https://github.com/KhronosGroup/glTF