An Interface Prototyper Supporting Free Design Components Specification

Pedro M. Teixeira-Faria[1] and Javier Rodeiro-Iglesias[2]

[1] School of Technology and Management - Polytechnic Institute of Viana do Castelo, Portugal
pfaria@estg.ipvc.pt
[2] School of Informatics Engineering, University of Vigo, Spain
jrodeiro@uvigo.es

Abstract. Complex components allow increasing the abstraction in a visual interface specification process, with independence of any platform or programming language to represent an user interface. In order to support this type of components a XML specification was created which allows specifying components visual appearance, composition and dialog. It provides a user interface abstraction to free design components (without any dependency of libraries of predesigned user interface components – toolkits). All information containing in the specification allows showing what will be the user interface final visual aspect, using a handmade or a computer technique (this is the objective of this paper). Using complex components, being incrementally more complex, simplifies the user interface designing and prototyping processes. In order to demonstrate the possibility to show the visual appearance of an interface and to validate the specification, a prototype to visualize any user interface specified using UIFD was created.

Keywords: User Interface Prototyper, Visual Appearance, User Interface Free Designer.

1 Introduction

In our days, direct manipulation interfaces [5] are virtually universal. Interfaces that use windows, icons and menus have become a standard in traditional computational systems. The process of building that final user interfaces can be represented by models. Usually, a model is supported in some user interface specification and according with [1][7][9] there are three elements that must be considered on a user interface specification:

- Interface graphic elements definition (visual presentation);
- Component composition;
- User interaction over the referred components (dialog).

In previous studies [10] the creation of a simple game for younger children was proposed, using DGAUI (*Abstract Definition of Graphical User Interface*) [9]. The interactive visual user interface prototype was created from an abstract representation

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2013, LNCS 8004, pp. 490–499, 2013.

containing visual elements representing sport balls and sport fields that the user should connect with each other. After the interface has been implemented it was considered that certainly the implementation work could have been reduced if the DGAUI system embraced the *complex component* concept [13]. It was questioned if eventually could be possible to reduce the representation complexity in terms of number of states and visual transitions, through identification and grouping of visual elements which have a common logic. The *complex component* concept evolved from other abstract interaction objects concepts (*AIOs*) [2][3][11]. Basically, is a component composed of other components (*simple* or/and *complex*) which interact with each other through its *self* and *delegate events/actions* working toward a common goal (e.g. a *toolbar* allows a user to select a specific tool to perform some task at a given time).

The *complex components* creation process is based on a XML specification (*UIFD – User Interface Free Design*) developed to allow representing visual components freely designed. The specification has a structure divided in three fundamental parts: *repository*, *library* and *interface*. It allows increasing the abstraction level on the components representation process used to design the final user interface. Following, in order to verify and validate the XML specification, a prototyper was developed. An example of a game interface for younger children was used to verify loading components into a memory structure, to obtain the global visual states and to represent the visual components on the display, in order the user start interact with the interface.

1.1 Study Motivation

The term user interface is most of the times associated to what a user see and interact with. [6] for example, gives us an interesting interface definition: "*interface is where people and bits meet*". Thus, is imperative that an user interface defining process always take in consideration the indications of user needs in order to successfully accomplish the tasks that he needs/wants. Since a long time, a technique to achieve that is used and is called low-fidelity prototyping ("lo-fi" for short): the idea is building prototypes on paper (e.g. sketches) and testing them with real users. The value of prototyping is widely recognized [8] because it effectively educates developers to have a concern for usability and formative evaluation, and because it maximizes the number of times we get to refine the design before commit to code. Following, the next step will be the translation of those sketches to user interface components, usually called widgets (*wi*ndows ga*dgets*) because most of the visual interfaces are created for the user to interact with them, using those kinds of interactive visual components. However, usually those visual components are not sufficient to represent a complete user interface (regarding the visual appearance, the components composition and the dialog) in many cases by the dependency of libraries of predesigned user interface components (toolkits) that not allow user interface customization, as a user interface designer would like. Thus, we would like to focus this study on the importance for an interface designer, on having the possibility to design free visual components, to be used to represent complete interfaces prototypes.

1.2 Defined Problem

We developed a process which allows, from a XML representation, to test and to validate how a free component (*simple* or *complex*) would appear and could be used in a visual user interface. And, thereby, it is possible to refine the components (and consequently the user interface) according with user needs. However, we had the need to create a prototyper, in order to validate the specification previously created. The main question to be solved and presented in this study is the following: it is possible to obtain and to represent the global visual states of an user interface though grouping the visual states of individual components, designed with a XML specification and responding to user interaction through events over those components? The answer to this question must be considered under our concept in which we treat global visual user interface states as groups of visual components states. For example, a typical web form represents a single web state, but a form has several possible states, in result of user interaction with it. In result of this study we developed a prototyper to represent complete visual user interfaces obtained from the UIFD specification.

2 Free Design Components

Lately, to specify user interfaces using XML (XML-UIDL) is considered to be one solution for the standardization and interoperability between applications [4][12]. However, as previously referred, most of current approaches to interface complex components usage, refer using pre-defined widgets, which limits the designer ability to do components customization (at both visual appearance and dialog levels). Thus, a specification to support free design of user interface *complex components* was established (UIFD). It is a structure divided in three parts:

- *Repository*: contains components nominal definition (*simple* and *complex*) to be used on the *library*;
- *Library*: contains *complex components* complete description, ready to be used on the interface design. It is possible to use multiple instances of the same *repository* component to create a *library* component;
- *Interface*: this structure supports the interface (obtained from *library* components).

It is possible to verify below a XML structure (Fig. 1) of a library component. In this case is one of the sport fields' components (*CC_Field_1*).

Each one of the referred three specification parts is supported in a XML [14] file with the correspondent DTD validation. The XML files constructing process is triggered by the designer when he's constructing personalized *complex components* during the user interface design process.

From the referred previous study [10] we verify that knowing in advance the user interface functionality, it is possible to obtain a visual representation of it (in a form of a state diagram) having the states and the transitions between them, in result of user interaction.

```
<Complex_Component Name="CC_Field_1">
  <Composition>
    <SC>SC_Field_1_N</SC>
    <SC>SC_Field_1_C</SC>
  </Composition>
  <Visual_Appearance></Visual_Appearance>
  <CC_States>
    <CC_State ID="0" Visible="true" Active="true">
      <Status>
        <SC Name="SC_Field_1_N" Visible="true" Active="true" />
        <SC Name="SC_Field_1_C" Visible="false" Active="false" />
      </Status>
      <Dialog_State>
        <Self_Evt ID="1" Event="LeftClick" Component="SC_Field_1_N"
                                                    Ini_State="0" End_State="1" >
          <Preconditions>
            <Pre_Cond Component="CC_Ball_3" State="1" />
          </Preconditions>
        </Self_Evt>
      </Dialog_State>
    </CC_State>
    <CC_State ID="1" Visible="true" Active="false">
      <Status>
        <SC Name="SC_Field_1_N" Visible="false" Active="false" />
        <SC Name="SC_Field_1_C" Visible="true" Active="false" />
      </Status>
    </CC_State>
  </CC_States>
  <External_Events>
    <Delegate_Actions>
      <Trigger_DA ID="1" SELF_STATE="1" TO="CC_Ball_3" Trigger_DE_ID="1" />
      <Trigger_DA ID="2" SELF_STATE="0" TO="CC_Ball_1" Trigger_DE_ID="2" />
      <Trigger_DA ID="3" SELF_STATE="0" TO="CC_Ball_2" Trigger_DE_ID="2" />
    </Delegate_Actions>
  </External_Events>
</Complex_Component>
```

Fig. 1. Example of a *complex component* designed using the UIFD specification (simplified)

We verify that the visual interface representation is simplified, although increasing the complexity of the components used to represent it. In order to measure the simplification obtained with our specification we must look at the decreasing number of components available as the abstraction level increases. This happens in virtue of encapsulation property available in *complex components*. We are using fewer components from lower to higher abstraction level, in result of the components being contained inside other components at a higher level. The use of this component type, being incrementally more complex and at the same time being at a more abstract level, simplifies the process of designing and prototyping the user interface.

Taking as an example the game interface for younger children previously mentioned, on (Fig. 2) is shown a simple schema of the process to increase the simplification on a user interface representation, while the abstraction level to represent visual

components increases. At the top of the figure is possible to verify the existence of 15 simple components (on the left side) that will successively be grouped into *complex components* (6 in the first abstraction level, 2 in the second and 1 *complete complex component* that represents the final user interface). At the bottom of the figure two examples of *complex components* (*CC_Ball_1* and *CC_Field_1*) are shown. For each *complex component* it is possible to verify the visual states it contains, the restrictions, the internal visual transitions and the events it supports (user events, and *delegate events*). Thus, we verify the amount of information that is encapsulated in each *complex component*, and we also perceive that as complexity increases (as abstraction increases) at the same time the components are simplified to be used representing an user interface.

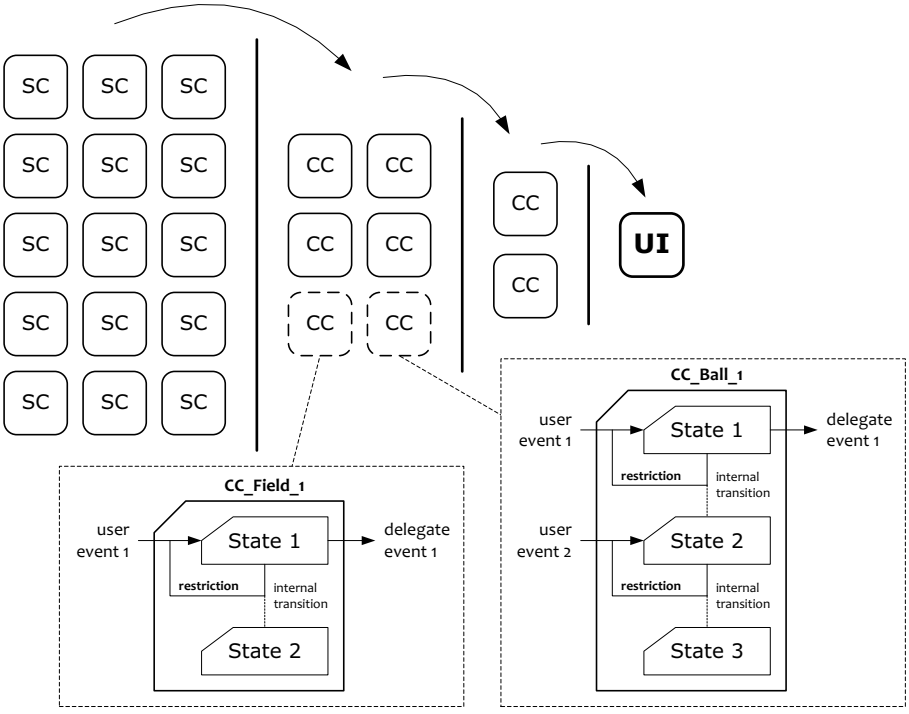

Fig. 2. Decreasing number of components as the abstraction level increases

Another important feature provided by *complex components* usage is their possibility of being reused (at semantic and functional levels). All components to be used are available in the common *repository* and the components personalization is done at the *library* level, promoting components reuse and also contributing to simplify the interface design and behavior, which can be freely established by the interface designer.

3 Interface Prototyper

In order to validate the XML specification previously indicated, an automatic interface prototyper was created. The developed tool allows reading an interface specification in XML and following, to represent the prototype of a concrete interface. The prototype obtained is a functional representation of a component visual appearance and behavior, which is also a complete user interface (*complete complex component*).

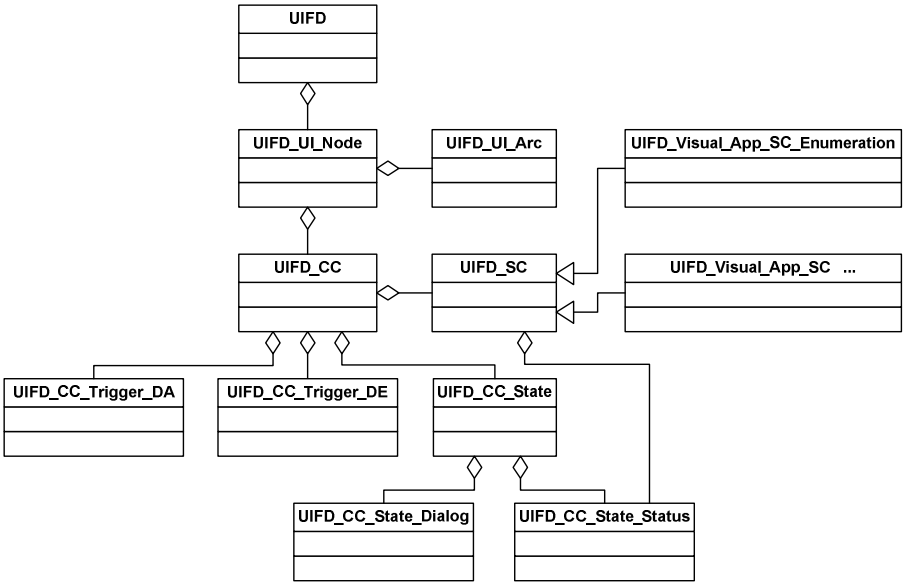

Fig. 3. Complex *component* core class architecture (simplified)

One of the major advantages of using *complex components* is the independence of any platform or programming language that can be used to represent an user interface. From several technologies available, it was decided to implement a first version of a prototype using Adobe Flash. Some advantages of this technology are:

- Currently, to be a technology widely distributed and able to be used by a large number of users;
- Supporting the visual design of graphical primitives and thereby to enable *complex components* visual representation, obtained from data structures (from the preloaded XML files). Despite being vector-based, allows bitmaps incorporation where needed;
- Its flexibility allows to export exactly the same content through web browsers and platforms, without any extra code;
- Supports video, audio, animation, and advanced interactivity, which provides flexibility for future evolution of *complex component* concept.

A simplified diagram of the class main architecture, respecting to the design and implementation of the interface prototyper is depicted in (Fig. 3). The diagram represents an overview of the application main classes that were used to achieve the experimental results here presented. The resulting framework architecture is rather flexible, enabling easy integration of new modules in the *complex components* context. The `UIFD_UI_Node` class supports all user interface global visual states and the connection between those states in result of user interaction is stored by `UIFD_UI_Arc` class. The `UIFD_CC` class is responsible for managing the *simple* and *complex components*. On the case of *simple components*, several classes are available to support any graphical primitives (the `UIFD_Visual_App_SC_Enumeration` class is one of these examples, used to represent the images employed in the game interface for younger children previously created). On the case of *complex components*, the `UIFD_CC` class manages his visual states in `UIFD_CC_State` class, which supports the *self events* triggered by the interface user (`UIFD_CC_State_Dialog`) and the *visible* and *active* properties of each *simple component* available on a *complex component* (`UIFD_CC_State_Status`). Finally, the *delegate events/actions* of a complex component are respectively managed by `UIFD_CC_Trigger_DE` and `UIFD_CC_Trigger_DA` classes.

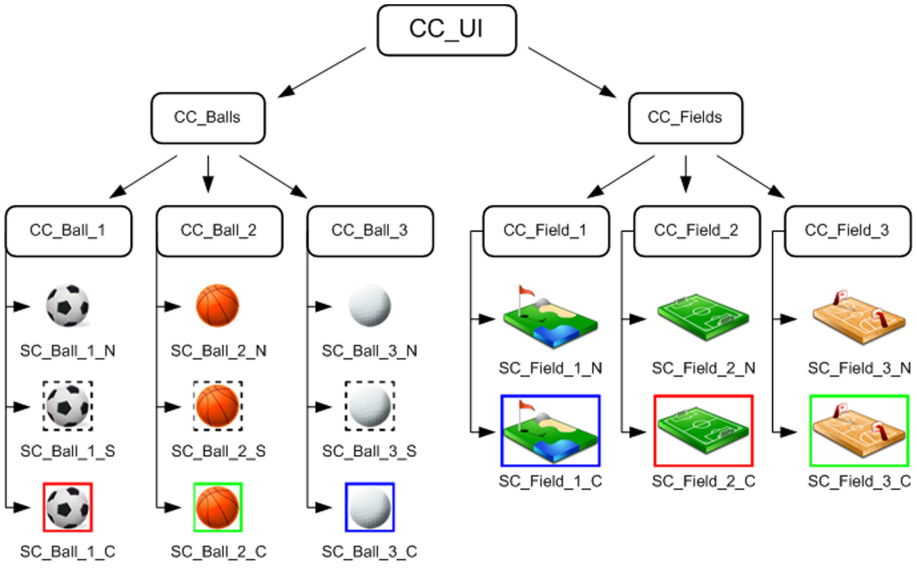

Fig. 4. User interface simple components arranged in a tree

Initially, the prototyper reads the .XML files, structured into a hierarchy of one components tree (Fig. 4). As we can verify on the figure the logical data structure of the game interface, which is composed of two *complex components* (*CC_Balls* and

CC_Fields) each one having three other *complex components*. Each ball (*CC_Ball_*) has three *simple components* and each sport field (*CC_Field_*) has two *simple components*. In order to simplify the global visual states generation, the following step of the algorithm is to convert the components tree into a data structure in memory, which is this case is a linked list (Fig. 5). Each list node (left side) represents all the *complex components* connected in sequence. And, each node has a list of connected arcs, representing the components (*simple* or *complex*) inside each one of the nodes.

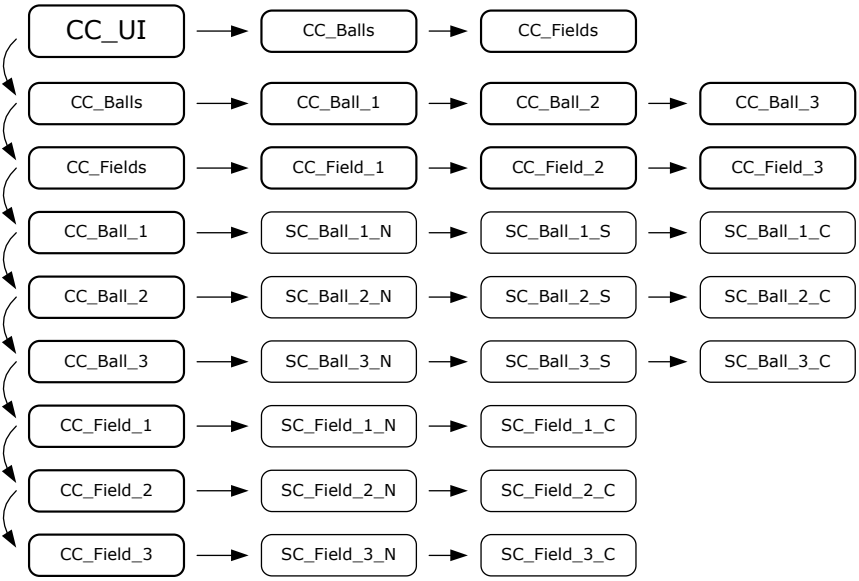

Fig. 5. Linked list obtained from components tree

After concluding the first step of uploading the components hierarchy into the memory structures, is necessary to calculate the user interface global visual states and relate each other according with the dialog events established to change from one visual state to the other. The objective is to obtain a state diagram representing the user interface functionality, with the global visual states (`UIFD_UI_Node`) and the existent transitions between them (`UIFD_UI_Arc`).

As previously indicated, we are at this step trying to obtain user interface states as groups of components states. Thus, an algorithm to traverse the memory structure and to obtain the global visual states was created. Basically, the algorithm follows the steps:

1. For each *self event* (*SE*) available in (`UIFD_CC_State_Dialog`) representing the dialog produced by user interaction over a *complex component* state, the algorithm will verify if any *delegate action* (*DA*) exists in (`UIFD_CC_Trigger_DA`), related to that state;

2. For each one of these (*DA*) found on the state dialog, which executes a *delegate event* (*DE*) the algorithm will look for what is necessary to change on the other affected *complex component*;
3. One global state will be the result of the changes caused by a (*SE*) on the *complex component* which it belongs plus the changes caused by (*DE*) triggered by other components.

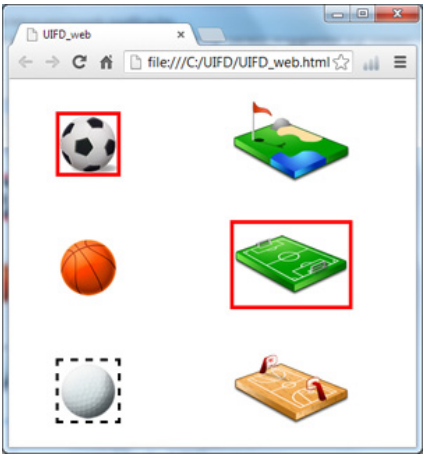

Fig. 6. User interface represented by the UIFD prototyper

After running the algorithm to obtain the global visual states, the prototyper will graphically represent the visual interface (Fig. 6), in order the user may interact with it and test the interface prototype, which in this case is the game interface previously referred.

4 Conclusions

In previous studies a XML structure/specification to design user interfaces was created. That specification allows specifying *simple* and *complex components*. Based on these components, an user interface was created and then loaded by the developed prototyper in order to visually represent the user interface. Having a first version of a prototyper, which allows simulating all the global visual states and all the interaction with his components, the following study line will be to upgrade it, developing techniques and integrating a no-heuristic evaluation module, in order to obtain usability results from multiple interface users. Thus, will contribute to detect errors on interfaces under tests and consequently to improve user interfaces efficiency.

Acknowledgments. This work was supported by:

1. Grant SFRH/PROTEC/49496/2009 of MCTES – Ministério da Ciência, Tecnologia e Ensino Superior (Portugal).
2. Project TIN2009-14103-C03-03 of Ministerio de Ciencia e Innovación (Spain)
3. Project 10DPI305002PR of Xunta de Galicia (Spain).

References

1. Carr, D.: Specification of Interface Interaction Objects. In: CHI 1994 – ACM Conference on Human Factors in Computer Systems, pp. 372–378 (1994)
2. Cowan, D., Lucena, C.: Abstract Data Views: An Interface Specification Concept to Enhance Design for Reuse. IEEE Transactions on Software Engineering (1995)
3. Duke, D., Harrison, M.: Abstract interaction objects. Computer Graphics Forum 12(3), 25–36 (1993)
4. Guerrero-Garcia, J., González-Calleros, J., Vanderdonckt, J., Muñoz-Arteaga, J.: A Theoretical Survey of User Interface Description Languages: Preliminary Results. In: Latin American Web Congress, pp. 36–43 (2009), doi:10.1109/LA-WEB.2009.40
5. Hutchins, E., Hollan, J., Norman, D.: Direct Manipulation Interfaces, vol. 1, pp. 311–338. Lawrence Erlbaum Associates, Inc. (1985)
6. Negroponte, N.: Being Digital. Vintage Books, New York (1994)
7. Paterno', F., Santoro, C., Spano, L.D.: MARIA: A Universal, Declarative, Multiple Abstraction-Level Language for Service-Oriented Applications in Ubiquitous Environments. ACM Transactions on Computer-Human Interaction 16(4) (2009)
8. Rettig, M.: Prototyping for tiny fingers. Communications of the ACM 37(4), 21–27 (1994)
9. Rodeiro, J.: Representácion y Análisis de la Componente Visual de la Interfaz de Usuario. PhD Thesis, Universidad de Vigo (2001)
10. Rodeiro-Iglesias, J., Teixeira-Faria, P.M.: User Interface Representation Using Simple Components. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2011. LNCS, vol. 6761, pp. 278–287. Springer, Heidelberg (2011)
11. Savidis, A.: Supporting Virtual Interaction Objects with Polymorphic Platform Bindings in a User Interface Programming Language. In: Guelfi, N. (ed.) RISE 2004. LNCS, vol. 3475, pp. 11–22. Springer, Heidelberg (2005)
12. Souchon, N., Vanderdonckt, J.: A Review of XML-compliant User Interface Description Languages. In: Jorge, J.A., Jardim Nunes, N., Falcão e Cunha, J. (eds.) DSV-IS 2003. LNCS, vol. 2844, pp. 377–391. Springer, Heidelberg (2003)
13. Teixeira-Faria, P.M., Rodeiro-Iglesias, J.: Complex Components Abstraction in Graphical User Interfaces. In: Jacko, J.A. (ed.) Human-Computer Interaction, Part I, HCII 2011. LNCS, vol. 6761, pp. 309–318. Springer, Heidelberg (2011)
14. W3C Recommendation: XML, XML 1.0 (2008), `http://www.w3.org/TR/REC-xml/`