Reduce Complexity by Increasing Abstraction in Interactive Visual Components

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

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

Abstract. The objective of this study is to introduce a method to abstract complex components in order to create a complete and functional user interface, simplifying the complexity process of user interface design. An example of a simple user interface of a game for younger children is explained and its visual states and transitions represented through a state diagram. However, the level of detail provided by simple components, to represent the user interface is very extensive, making lengthy the interface designing process. Thus, it was decided to increase the abstraction level by introducing a new complex component structure which, due to its encapsulation feature, allows to group components into other more complex components, but with more functionality. An abstraction process through grouping components by levels is detailed, with the intention of proving the validity of the *complex component* concept to simplify the creation of complete, free and functional user interfaces.

Keywords: Abstract interaction objects · Complex components · Visual interface representation

1 Introduction

In previous studies [8] an interactive visual user interface prototype was created, based in the direct manipulation interaction style. After the interface has been implemented it was considered that probably the implementation work could have been reduced if the system embraced a more abstract component concept. Much work in the field of interactive graphics involves describing an interface in terms of a collection of *interaction objects* [3, 6, 9]. An AIO represents a user interface object without any graphical representation and independent of any environment [10]. Usually, is understood as a conceptual representation of an interface object. A *concrete interaction object* (*CIO*) represents any visible and manageable user interface visual component that can be used to input/output information related to user's interactive task and are sometimes called *widgets* (*wi*ndows ga*dgets*). After analyzing the features of 3 AIOs available in the literature, it was considered the possibility to use *complex components* to create the same previous game interface. A detailed process to abstract *complex components* is used to verify the components assembling process, in order to increase the level of

M. Kurosu (Ed.): Human-Computer Interaction, Part I, HCII 2015, LNCS 9169, pp. 426–437, 2015.
DOI: 10.1007/978-3-319-20901-2_40

abstraction. The abstraction term is here introduced with the meaning of simplification, through component's encapsulation. In this way, it was considered to reduce the number of components necessary to use to design and to represent a user interface functionality, in spite of these components being more complex (with more functionality) as the abstraction level grows. Then, a user interface analysis is presented, considering the level of encapsulation (abstraction) introduced by complex components usage, considering the number of events, global transitions and visual states. Following, some conclusions related with the study are presented.

2 Defined Problem

The interactive visual user interface prototype previously created [8] consists of a simple game for younger children, containing visual elements representing sport balls and sport fields. It was possible to use a system to create the visual interface from an abstract representation which uses *simple components* (*SCs*). After the 2D visual game interface had been specified using *SCs*, and a prototype be created, it was possible to verify and to conclude that the level of detail needed to specify visual interface components, using that system, is too deep and the specification becomes too complex for the size of actual direct manipulation interfaces, becoming impracticable the extensive use of it. Thus, it was decided to simplify the design process through seeking for a method to reduce the number of components needed to build a user interface.

3 Complex Components

The notion of *interaction object* do not need to be confined to graphics systems, because it represents a useful structure for thinking and reasoning about the behavior of interactive systems in general [3, 5]. In the interface designing process, it could be necessary the appropriate (AIOs) selection and four of them were analyzed: *interactor*, *abstract data view*, *virtual interaction object* and *complex component*. A concept of *interactor* has been introduced by [4, 6]. It is a component (object) in the description of an interactive system. The *interactors* do not indicate a specification language, but an adequate structure to model an interactive system. It appears as an algebraic conceptualization whose usefulness will be higher in the formal analysis of an interactive system. Some authors advocate the concept of dialog independence, where interactive systems are designed and implemented with the goal of providing a clear separation between the user interface and the application [1]. A user interface design concept which corresponds to that is the *Abstract Data View (ADV)* [2]. However, its specification is not considered in the context of a complete interface, focusing primarily on defining simple components with multiple states and the relationship between them. According with [9], in the context of user interface development, *interaction objects* play a key role for implementing the constructional and behavioral aspects of *interaction*. The author introduced the concept of *virtual interaction objects*, as synonymous of (AIOs). However, they are dependent of different interface toolkits supported by different platforms.

Thus, as none of the 3 previous analyzed AIOs supports components composition, it was decided to verify a new component structure [7] which supports that feature, among others related with supporting visual appearance and interaction: the *complex component*. Basically, a *CC* is a component composed of other components (simple/complex) which interact with each other through its *self events* (*SEs*) and *delegate events/actions* (*DEs*/*DAs*) 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). *DEs* and *DAs* are together a union mechanism concept related with communication between *complex components*. *SEs* are encapsulated in a *complex component* and its internal functionality is responsible for triggering *DAs*, which will be responsible for triggering *DEs* that other *complex components* will receive. The detail of these internal *DAs* functionality is occluded from the interface designer who just needs to establish the connection between complex components through their *DEs*.

4 Detailed Process

In order to build the game interface previously indicated, 15 *simple components* were used. For each sport ball, three possible visual states have been created (*normal*, *selected* and *correct*) while in the case of the sport fields two visual states were created (*normal* and *correct*).

After creating all possible visual states of each component, it has been defined which components respond at which events (the balls and the sport fields in this case). It was decided to use the "*LeftClick*" event detection for the user interaction. The complete functionality of the game is represented by 20 states (nodes) and 48 transitions (arcs) between those states (Fig. 1). The initial state is represented by (*State 0*) and the final state is represented by (*State 19*). The connections (transitions) between the

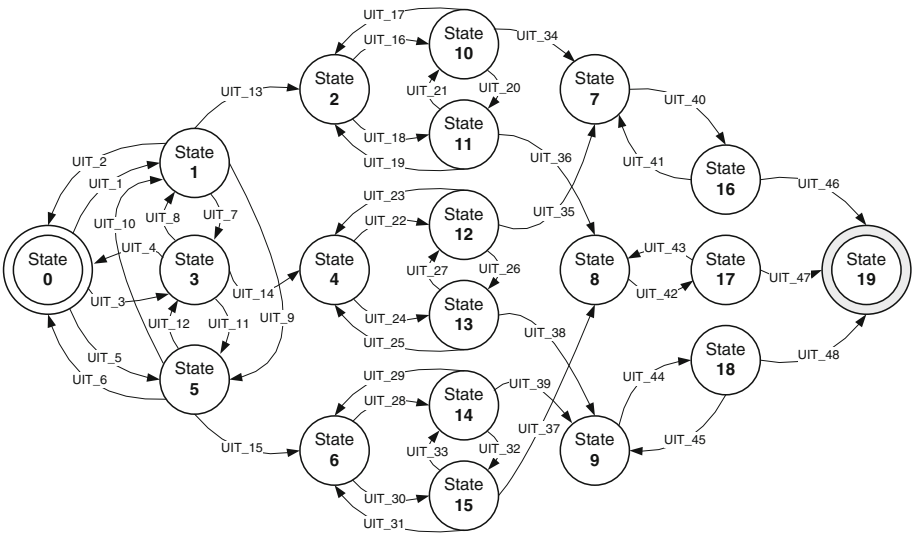

Fig. 1. State diagram representing the 48 game interface visual transitions

nodes (states) always have two parameters associated: the component of the original node and the event which triggers the visual state change. In order to simplify the state diagram understanding of this game interface example was decided to not include the information related with event identification, once the user event is always the same.

4.1 Complex Components: Level 0

The complexity to represent a user interface is given by the number of components and the number of events to consider, in order representing all the global visual states to be obtained, from visual transitions generated by user interaction with the interface components. Following, an analysis is made to the process (distributed over 4 levels) of creating the interface functionality using *complex components*. When considering the *complex component* concept we always need to establish an entrance level, on which will be defined the *simple components* to be used on a user interface and all the possible *visual transitions* between them. The *simple components* will be responsible for the interface visual aspect. In the case of the game previously described, at this level (*Level 0*) we will have 15 visual *simple components* and 12 possible visual transitions (Fig. 2) between them.

4.1.1 Simple Visual Components: Level 0

Each of the three balls can assume:

- (normal) visual state (SC_Ball_i_N, i = {1, 2, 3});
- (selected) visual state (SC_Ball_i_S, i = {1, 2, 3});
- (correct) visual state (SC_Ball_i_C, i = {1, 2, 3}).

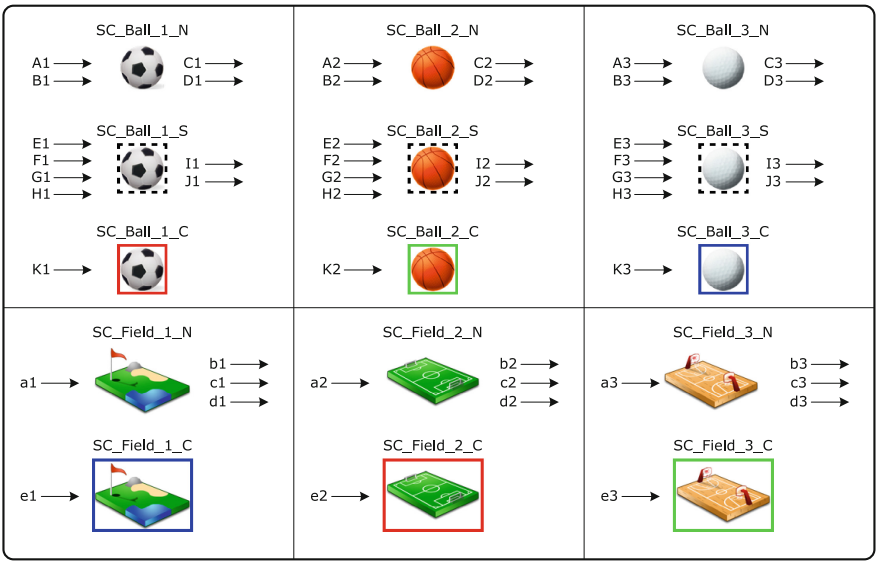

Fig. 2. 15 visual states and 12 visual transitions from *simple components*

Concerning the fields, they can assume:

- (normal) visual state (SC_Field_i_N, i = {1, 2, 3});
- (correct) visual state (SC_Field_i_C, i = {1, 2, 3}).

Throughout this paper the terms (*normal*, *selected* and *correct*) are used to identify the *complex components* visual states.

4.1.2 Visual Transitions: Level 0

The game functionality, previously described, indicates 4 distinct types of visual transitions that may occur between the referred 15 *simple components*:

- TB_1 – Pass the ball from the (normal) state to the (selected) state; (SC_Ball_i_N → SC_Ball_i_S, where i = {1, 2, 3});
- TB_2 – Pass the ball from the (selected) state to the (normal) state; (SC_Ball_i_S → SC_Ball_i_N, where i = {1, 2, 3});
- TB_3 – Pass the ball from the (selected) state to the (correct) state; (SC_Ball_i_S → SC_Ball_i_C, where i = {1, 2, 3});
- TF – Pass the field from the (normal) state to the (correct) state; (SC_Field_i_N → SC_Field_i_C, where i = {1, 2, 3}).

We consider a visual transition as a visual state change, triggered by an event (resulted from user interaction with the interface). According with the number of *simple components* and the game rules established, each of the 4 distinct visual transitions types may occur 3 times (3 visual transitions for each ball and 1 visual transition for each field). These transitions may occur triggered by events from the user or from the components itself.

4.2 Complex Component Abstraction Process

Designing hierarchically organized *complex components*, allows increasing the abstraction level when designing a visual interface. The method to obtain *complex components* by increasing their abstraction levels can be done through a process based on an iterative cycle. Before starting the process is necessary to establish an entrance level, in which *simple components* are identified together with user events and all possible visual transitions between them (those components will be responsible for the interface visual appearance). Then, the iterative cycle to abstract *complex components* follows 3 criteria:

- ***Criterion 1***: a *complex component* must have *simple/complex components* inside of it (internal components) and the (*SEs*) in result of user interaction with the components be identified;
- ***Criterion 2***: the identified *complex components* relate to each other through their input/output *delegate events/actions* which (or other conditions) are not totally satisfied;

– ***Criterion 3***: with respect to new *complex component* creation conditions, if the interface is not fully functional and some of the *complex components* still have input/output (*DE/DA*) or other conditions not totally satisfied (e.g. still waiting for some preconditions to be accomplished), the abstraction level can be increased through grouping components.

On next level (*Level 1*) *simple components* will be grouped into *complex components*. Each ball can have 3 different visual representations and will be grouped in one *complex component*. Each field will be made of 2 different visual representations, which will also be grouped in one *complex component*. Following, 6 different *complex components* and their functionality will be verified.

4.3 Complex Components: Level 1

This second level corresponds to the first abstraction level of *complex components* creation: 6 *complex components* are created. Each ball *CC_Ball_i*, *i = {1, 2, 3}* aggregates 3 *simple components* with their 2 user events and their 3 visual transitions and they can be represented as *CC_Ball_i* = {*SC_Ball_i_N*, *SC_Ball_i_S*, *SC_Ball_i_C*}. The visual states of each ball *complex component* (*CC_Ball_i*) are directly related with the *simple components* that represent them. For this game, each visual state of a ball *complex component* is represented by a single *simple component* (*SC_Ball_i_k, k = {N, S, C}*). The visual state of a ball *complex component* (*CC_Ball_i*) assumes the visual state of the *simple component* (*CC_Ball_i_k*, where *i = {1, 2, 3}* and *k = {N, S, C}*). The other three *complex components* are related with the fields: *CC_Field_i*, *i = {1, 2, 3}*. Each field aggregates 2 *simple components* (*CC_Field_i* = {*SC_Field_i_N*, *SC_Field_i_C*}) with its unique user event and his unique visual transition. Analogously to the balls, each field *complex component* (*CC_Field_i*) has two visual states (*CC_Field_i_k*, where *i = {1, 2, 3}* and *k = {N, C}*). At this first abstraction level, the 6 *complex components* are mutually independent. Each one has his visual states and accepts events which produce their transitions. Furthermore, initially there isn't any communication between those *complex components*.

4.3.1 Components User Interaction: Level 1

On (*Level 0*) 9 user events were considered, with which the user interacts in the game interface. At this present level (*Level 1*) creating the three ball *complex components* (*CC_Ball_i*, *i = {1, 2, 3}*) each of them reduces by one the number of user events to be considered. Six user events exist (*mouse click*) (*A'1*, *A'2*, *A'3*, *a1*, *a2* and *a3)* (Fig. 3) which may be triggered over six different *complex components*. Exemplifying for the (*A'1)* event, this represents a mouse click on the (*CC_Ball_1*) *complex component* which is responsible for 2 visual transitions (*TB_1* or *TB_2*). These two transitions are triggered by user interaction with the (*normal*) visual state or with the (*selected*) visual state, respectively. In the case of the fields, for each *complex component* only one visual state may receive user interaction (e.g. to click in (*CC_Field_1*) is equivalent to

click in (*CC_Field_1_N*) with (*a1*) user event). At the present level (*Level 1*) the user may trigger 6 user events over 6 *complex components*, reducing in 3 the number of user events to be managed.

4.3.2 Visual States and Transitions: Level 1

After 6 user events occurring over 6 *complex components* had been identified is important to detail the *self* and *delegate events* that proceed in result of user interaction with those *complex components* producing transitions between visual states. (Figure 3) represents components visual transitions (*TB_1*, *TB_2*, *TB_3* and *TF*) that occur in result of *self* and *delegate events* (and *delegate actions*) possible to be triggered by user or by *complex components*. The interface functionality does not become complete, just using the 6 *complex components* (individually) and it is necessary to establish the missing links between *complex components*. The dashed arrows in gray represent *delegate events/actions* necessary to be enabled in order to the game interface become fully functional (each arrow corresponds to one *delegate action* triggered by a *complex component* and one *delegate event* received by another *complex component*). Each ball and each field receives a user event (*mouse click*). In the case of the balls, (*TB_1*) or (*TB_2*) transitions (represented by 6 black arrows) are triggered after user interaction (*A'1*, *A'2* or *A'3*) which produces a *self event* over a ball.

For each ball, (*CC_Ball_i_N*) visual state can change to (*CC_Ball_i_S*) visual state, or the reverse, through (*TB_1*) or (*TB_2*) visual transitions, respectively. These transitions are triggered by user interaction (*A'i*) over a ball visual state (*CC_Ball_i_k*, where $i = \{1, 2, 3\}$ and $k = \{N, S\}$) and can be represented by the function:

$$A'i = \begin{cases} TB\_1 \ if \ k = N \\ TB\_2 \ if \ k = S \end{cases} \quad (1)$$

In the case of the fields, at this moment, doesn't occur any visual transition triggered by user interaction. (*TF*) transitions still inactive because they depend on relations to be established between components, external to the fields. In the particular example of this game interface, at this level (*Level 1*) the functionality of each one of these 6 *complex components* doesn't allows yet triggering *delegate events* (not internally nor externally). Still considering the information provided by (Fig. 3) is verifiable that besides 6 user events (*A'1*, *A'2*, *A'3*, *a1*, *a2* and *a3*) another 30 (15/15) *delegate events/actions* produce (*TB_2* and *TB_3*) transitions (indicated over gray dashed arrows). (*TF*) transitions are triggered by (*a1*, *a2* and *a3*) user events. Each ball receives 1 *user event* (corresponding to 2 *self events*), receives 5 *delegate events* and enables 2 *actions* in other components. In the case of the fields, each one receives 1 user event (corresponding to 1 *self event*), receives 0 *delegate events* and triggers 2 *actions* on other components.

4.3.3 Completing Visual Interface: Level 1

Until now, the analysis on creating the game visual interface using the 6 *complex components* (individually) demonstrates that the interface functionality is not complete because some transitions are only activated by *delegate events* triggered at a higher level of *complex component* abstraction or directly by the user interface designer, who

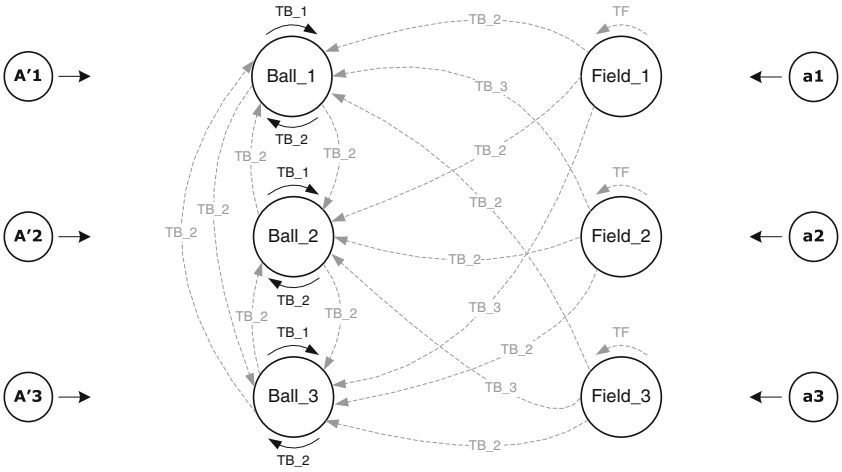

Fig. 3. Visual transitions of 6 *complex components* (*Level 1*)

has to manually establish the missing links between *complex components*. At this first *complex component* level, using 6 *complex components* without completing their functionality, some other invalid visual states and visual transitions could occur. Considering the game rules (in which only one ball can be selected at a time) and the possible user events over a ball, the following task was set: assuming that *A'j* represents an user event (*mouse click*) over a Ball (*CC_Ball_j, j = {1, 2, 3}*) which can have one of the three possible visible states (*normal (N)*, *selected (S)* or *correct (C)*) identified as *CC_Ball_j_k*, where *k = {N, S, C}* and *j = {1, 2, 3}* a new (*BT*) task is defined, based on possible transitions:

$$BT(j,i) = \begin{cases} TB\_1 \ if \ k = N \wedge j = i \\ TB\_2 \ if \ k = S \wedge j = i \\ TB\_2 \ if \ k = S \wedge j \neq i \end{cases} \quad (2)$$

The (*BT*) task guarantees that just one ball can be selected at a time and allows changing (*CC_Ball_i*) visual states, according with game rules. When the user selects a ball (*CC_Ball_j, j = {1, 2, 3}*) that means (*BT*) task will verify the ball visual state, comparing it with all the balls visual state (*CC_Ball_i, i = {1, 2, 3}*):

- If the user clicks on a ball represented by CC_Ball_j and that is in (normal) state, the (BT) task executes a (TB_1) transition, which allows that ball to change his visual state from (normal) to (selected);
- If the user clicks on a ball and that already is in (selected) state the (BT) task executes a (TB_2) transition;
- For the other balls the (BT) task executes (*TB_2*) transitions for each ball that is in (*selected*) state.

The interface designer needs to establish relationships between the 3 *complex components* (balls) and the other 3 *complex components* (fields) and to ensure that: after

a ball is selected, by clicking in the respective field, both components change to the (*correct*) visual state. Otherwise, in case the selected ball does not match the field the user has clicked, the ball is deselected. Therefore, according with the game objective (to make the correct match of the selected ball with the correct field) the following (*FT*) task was set. Considering that *aj* represents an event (*mouse click*) over the field identified by (*CC_Field_j_N, where j = {1, 2, 3}*):

$$FT(j,i) = \begin{cases} TF \wedge TB\_3 \; if \; \text{CC\_Field\_j\_N} \rightarrow \text{CC\_Ball\_i\_S} \\ TB\_2 \quad if \; \text{CC\_Field\_j\_N} \nrightarrow \text{CC\_Ball\_i\_S} \end{cases} \tag{3}$$

The (*FT*) task allows changing the interface visual state by activating new (*correct*) visual states (if the user makes a correct choice) or changing a ball (*selected*) visual state to the (*normal*) visual state (if the user makes an incorrect choice). After one ball be on its (*selected*) visual state, the user selects one of the 3 fields (*mouse click*) (*CC_Field_j, j = {1, 2, 3}*) which have one (*normal*) visual state identified as (*CC_Field_j_N, where j = {1, 2, 3}*). This (*FT*) task allows executing the following transitions:

- (*TB_3*) and (*TF*) transitions, if the field in which the user has clicked corresponds to the correct ball already selected (CC_Ball_i_S). That means, if the selected ball corresponds to the chosen field, both complex components (CC_Ball_i and CC_Field_j) change to the correct visual state (CC_Ball_i_C and CC_Field_j_C);
- (*TB_2*) transition, if there is no correct correspondence between the selected ball and the chosen field, which means that the ball on (*selected*) visual state changes to its (*normal*) visual state.

The correct connection will be validated by the (*FT*) task. The (*CC_Field_j*) (*TF*) transition results from a *self event* triggered by user interaction with one field. The (*CC_Ball_i*) (*TB_3*) transition is not directly related with user interaction, but is triggered by a *delegate event* received from the field correctly chosen by user. Both (*TF* and *TB_3*) transitions are simultaneously triggered and correspond both to the ball and the field final visual states (the *correct* states). Thus, the interface designer achieves the complete game interface functionality through establishment of missing connections between the 6 *complex components*. Following will be presented the interface representation considering the use of two *complex components*.

4.4 Complex Components: Level 2

At this abstraction level, 2 *complex components* are created (*CC_Balls* and *CC_Fields*). Each one aggregates 3 *complex components* defined in the previous level (*Level 1*): (*CC_Balls* = {*CC_Ball_i, i = 1, 2, 3*} and *CC_Fields* = {*CC_Field_i, i = 1, 2, 3}*). Each one (*CC_Balls*) and (*CC_Fields*) *complex component* acts as a container and represents examples of components with more than one visual state and more than one interaction. At this abstraction level is considered that the user may interact (*mouse click*) with 2 *complex components* representing the balls and the fields. Two user events exist (*mouse click*) (*A* and *a)* which may be triggered over the 2 *complex components* (*CC_Balls* and

CC_Fields). Therefore, at this level, a single user event acts on (*CC_Balls*) component which corresponds to click in one of the 3 interaction areas and whose action may trigger *self* or *delegate events*. The analogy is identical in the case of (*CC_Fields*). At this level, looking at the way *complex components* are structured, initially 2 new components exist, independent between each other (*CC_Balls* and *CC_Fields*). After the 2 *complex components* (*CC_Balls* and *CC_Fields*) have been created, the interface designer will verify that it can use them to complete the visual interface. For the interface functionality becomes complete (in order to obtain all the possible global visual transitions) it becomes necessary to make the connection between the 2 *complex components*, manually establishing the missed links between these 2 components, through their *delegate events/actions*. One possible way to complete this game interface could be using the rules established by (*FT*) task, previously defined, which establish the necessary links between balls and fields.

4.5 Complete Complex Component

After using the *complex component abstraction process* previously detailed, it was verified that is not possible to represent a complete and functional user interface through one *complex component* (has it could be the optimal result of this process). This happens because the final interface cannot fully respect the *Criterion 3* concerned with the *CC abstraction process*. Thus, the *final interface* is obtained through a *complete complex component* (*CCC*) which represents a complete visual object composed of related visual components, absolutely independent of any other interface components and in this case the complete game interface.

5 Reducing Complexity by Using Complex Components

In this section we intend to analyze the interface game representation process using two or six *complex components*. At (Level 1) above described, and considering the representation of six *complex components*, just the internal visual changes between states of each *complex component* are available. These visual changes result from transitions between the *normal*, the *selected* and the *correct* states. It is responsibility of the interface designer to complete the final representation of the game, establishing the missing links between the six *complex components*, through their *delegate events/actions*. At this level, the nine *self events* related to the six user events that can be triggered over the six *complex components* are already represented. It remains to establish the representation of 15 *delegate events* and the 15 (related) *delegate actions*, to the interface functionality becomes complete. To achieve that, the interface designer may implement the (*BT*) and the (*TF*) tasks which complete the missing functionality. In the second abstraction level, using two *complex components*, one of the interface functionality conditions, which requires that only one ball could be selected at a time, is already established. Therefore, the interface designer just needs to ensure proper connection between the balls and the fields. To achieve that, he must establish the representation of 9 *delegate events* and the 9 (related) *delegate actions*, in order to

Table 1. Using *simple components* versus *complex components*

number of	**level 0**	**level 1**	**level 2**
components	15 (SC)	6 (CC)	2 (CC)
visual states	20	20	20
global visual transitions	48	48	48
user events	9	6	6
self events	×	9	9
simple events	60	×	×
delegate events/actions	×	30	18
total events	**60**	**36**	**24**

complete the interface functionality. The interface designer can overcome this by implementing the (*TF*) task, which will complete the interface functionality.

In summary, comparing the representation of the game interface using six or using two *complex components* versus using *simple components*, it is verifiable that in both situations the complete functionality of the game is obtained, with the 20 visual states and the 48 possible transitions being represented. At (*Level 0*) above described, is necessary to create 60 single events, while considering the (*Level 1*) or (*Level 2*) of abstraction only 30 and 18 *delegate events/actions*, respectively, need to be established (Table 1). Also, a reduction on the number of user events necessary to be considered (less 3 user events) is obtained when comparing between using *simple components* and 6 or 2 *complex components*.

6 Conclusions

A test bed user interface was specified as a set of *SCs* [8] and as a set of *CCs* (as described in this paper). In both situations, the user understands the interface as a unique entity. Concerning the user interface specification, a simplification is observed when using the proposed *CCs* compared with the use of *SCs*. This simplification can be observed at:

- *Visual presentation*: the process of changing a *SC* is individually done. In the case of a *CC*, the visual state change process is encapsulated and internally done through the *SEs* available in the *CC*;
- *Component composition*: the possibility to group *SCs* into a *CC* and also to group *CCs* into other *CCs* allows to establish their positions with a single operation;
- *Component dialog*: the encapsulation provided by *CCs* allows to group components into other more *CCs*, but with more functionality. When a *CC* is used it will successively activate events, not only on the present *CC*, but also on the *CCs* contained in it. Therefore, there is a proliferation of events inherited from the *CCs* inside of it.

The apparent complexity perceived from the vast number of events involved is actually simplified by using *CCs*. Since the whole interaction process between *CCs* becomes totally transparent to the interface designer, the number of events that the interface designer has to control is reduced. This reduction is obtained as a consequence of the unneeded control of the interface designer over the *DAs* behaviour inside *CCs*. The designer just needs to verify the *SEs* (which arrive to the *CC*) and to identify the *DEs* in other *CCs*. From the interface abstraction analysis described in this paper, it was possible to verify that as the user interface abstraction increases through *CCs* encapsulation, more simplified becomes the user interface design, due to: (a) reduced number of needed components; and (b) simpler components to be used, due to the reduced number of events that the interface designer has to control.

Acknowledgments. This work was partially 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. Alencar, P., et al.: A Formal Approach to Design Pattern Definition & Application. University of Waterloo, Ontario (1995). http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.51.5451
2. Alencar, P., Cowan, D., Lucena, C.: A logical theory of interfaces and objects. IEEE Trans. Software Eng. **28**(6), 548–575 (2002)
3. Carr, D.: Specification of interface interaction objects. In: CHI 1994 – ACM Conference on Human Factors in Computer Systems, pp. 372–378 (1994)
4. Duke, D., Harrison, M.: Abstract interaction objects. Comput. Graph. Forum **12**(3), 25–36 (1993)
5. Duke, D., Faconti, G., Harrison, M., Paternó, F.: Unifying views of interactors. In: Proceedings of the Workshop on Advanced Visual Interfaces, pp. 143–152, 1–4 June 1994
6. Faconti, G., Paternó, F.: An approach to the formal specification of the components of an interaction. In: Eurographics 1990, North-Holland, pp. 481–494 (1990)
7. Teixeira-Faria, P.M., Iglesias, J.R.: 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)
8. Iglesias, J.R., 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)
9. Savidis, A.: Supporting virtual interaction objects with polymorphic platform bindings in a user interface programming language. In: International Workshop on Rapid Integration of Software Engineering Methods (RISE 2004), pp. 11–23. Springer Verlag, Luxembourg (2005)
10. Pinheiro da Silva, P.: User interface declarative models and development environments: a survey. In: Paternó, F. (ed.) DSV-IS 2000. LNCS, vol. 1946, pp. 207–226. Springer, Heidelberg (2001)