We are IntechOpen, the world's leading publisher of Open Access books Built by scientists, for scientists

4,800
Open access books available

122,000
International authors and editors

135M
Downloads

Our authors are among the

154
Countries delivered to

TOP 1%
most cited scientists

12.2%
Contributors from top 500 universities

CLARIVATE ANALYTICS BOOK CITATION INDEX INDEXED

WEB OF SCIENCE™

Selection of our books indexed in the Book Citation Index in Web of Science™ Core Collection (BKCI)

Interested in publishing with us?
Contact book.department@intechopen.com

Numbers displayed above are based on latest data collected.
For more information visit www.intechopen.com

29

Evaluation of the interface prototypes using DGAIU abstract representation models

Susana Gómez-Carnero and Javier Rodeiro Iglesias
University of Vigo
Spain

1. Introduction

The user interface is determinant for the success or failure of a software application. Its development is demanding as referring to time and costs of production. Myers and Rosson (Myers & Rosson, 1992) determine—according to inquiries on developers—that the efforts employed in building the interface of an interactive software rate around the 48%, whereas Garthner Group (Garthner Group, 1994) situate them around the 70%. Hence, optimizing the quality of the user interface by means of an effective design becomes vital so as to obtain its maximum acceptability.

The acceptability of a user interface is measured according to three main points:

1. ¿**Is the user interface kind to the user view?** In this case, the acceptability would be linked to the aesthetical properties of the interface. The load memory of the user or its functional correctness is not considered. The main point so far is that the user interface remains friendly and kind to the user's view.
2. **¿Does the user interface do what it has to do?** Testing whereas the user interface remains useful to accomplish successfully the tasks it was created for, both in the sense of its capacity to do what the user wants, and with respect to the user's capacity to do what the interface wants. This second point is closely related to the tasks that a user will have to perform over the user interface.
3. **¿Is the use of the user interface easy for the user?** Here, the usability term is introduced (Shackel, 1986) (Eason, 1988) (ISO, 1992) (ISO, 1993) (Nielsen, 1993) (IBM, 1993). If a user considers that the use of the user interface is too complicated, the user may not accept this user interface even though it may still remain efficient or aesthetically kind to the user.

The assessment of the functionality of these criteria, then, is really complicated. This remains so because of the means whereby the assessment has generally being carried out, mainly drawing on subjective appreciations through polls or inquiries on experts' opinions or questionnaires made to the users (Molich & Nielsen, 1990) (Preece & Rogers, 1994) (Nielsen, 1993) (Wharton, 1994). Therefore, and given the subjectivity of the user interface assessment in user interface qualitative evaluation techniques, it would be interesting to reach a more direct and discrete method of assessment in order to avoid the 'inaccuracy' of subjective perception. A possible solution for this would be that the user got to interact with a prototype of the user interface generated from a specification model obtained in a previous

phase of analysis of its requirements. Hence, the user could obtain a more rigorous view of the user interface, while helping the early identification of possible problems before time and money are ineffectually spent for the industry. Most of the evaluation methods are conceived after the creation of the software. Due to the costs of software development, then, it seems reasonable to suggest as necessary doing the evaluations previously to any software implementations.

There exists a great amount of research on the above mentioned criteria, but most of it is in theoretical state and isn't yet available for implementation. In fact, after the examination of different representation techniques in order to study whether these criteria can apply to complex interfaces—and thus derive an objective manner for their assessment—the conclusion is that these criteria remain somewhat unpractical and incomplete. Therefore, we present here an alternative notation that allows representing the user interface in an abstract manner while paying attention to its components, to its visual presentation (in graphic terms) and to its interaction with the user.

In this chapter, we present the part of the notation which represents the functionality of the user interface, so that the outcome will be a user interface that allows us to do semiautomatic assessments minimizing the developmental and evaluative costs. In section 2, we introduce the part of the notation corresponding to user interface behaviour representation. Section 3, then, proceeds to the notation that has to be used for test evaluation. In section 4 the EAU tool is introduced, which will allow users a dynamic assessment of the functionality of the user interface through an interactive simulation. And finally, in section 5, we expose the conclusions drawn out of this chapter's research.

2. DGAUI Representation

There is literature on user interface representation models. Reviewing research on different models for user interfaces, and focusing on those that proposed the visual representation mode (Gómez-Carnero, 2008), we here consider that the nature of the problems deriving from these models makes it necessary to come across an alternative solution. The proposed DGAUI representation is premised over the fact that the visual user interface is not a continuous structure. In fact, it argues that the visual user interface is made of discrete finite elements known as user interface components that define the interface as a composition of individual elements. These user interface components follow a topological hierarchy, so that one component may be contained within another (Rodeiro, 2001).

The notation for the definition of the visual user interface allows:

- To define the visual user interface components, with standard graphical primitives if the component has visual representation in the user interface; or else to determine properties if the component is for input information or simply a container of other user interface components.
- To determine the topological composition of visual user interface components so as to create the visual user interface with which the user interacts in a given moment.
- To represent the dialogue between the different components within the user interface, signalling the events to which a component reacts to and the manner in which the other visual user interface components respond to this.

The best choice in structuring the notation is XML. This will allow creating a DTD that will ease the parsing of notation structures.
Paying attention to the semantics of the notation, the first part conveys the initial representation of the visual user interface, while the second communicates the representation of the structure of any of the states that the user interface experienced derived of the interaction of its components—including as well the transitions between states. Given that the nature of the two parts in the notation is so discordant, we divide the notation in two DTD. The first (called **DGAUI-DEF**) consists of a detailed definition of each of the user interface components that constitute an interface. This separation is effective for the reusability of component definitions in other visual user interface representations. The second DTD (called **DGAUI-INT**) depends on the first, for this second notation is calculated from the first. The DGAUI-DEF contains all the states that a visual user interface can reach. This set of states can be calculated from the initial representation of the user interface. The initial state is made of the properties introduced into the user interface components' definition. After the onset state, a series of possible individual events that may take place over a user interface component or over the components in a state are simulated, and the changes produced on the components will determine a new state (one that already exists or one that is identified as new) and a transition between the actual state and the new state. This is the application of the concept the state diagrams for a user interface but generates from interaction on individual user interface components. This notation is oriented to the state of the visual user interface components instead of being oriented to the state of the whole visual user interface. An overall state of the visual user interface would be obtained from the combined states of the different components in a visual user interface.
Thus, the notation conveys the separation between the presentation and the behaviour of the user interface. The presentation is located in the representation definition while the functionality is located on the states and the transitions between states. These transitions between states are calculated using hypothetical user actions (events) on the visual user interface components and using also events of the system.
By **interface state** we understand any of the conjunctions of the visual user interface components that, according to the value of its properties, can be reached through the interaction with a user in a given moment.
For the definition of the notation we consider that:

- The user actions are not arbitrary.
- The set of visual user interface states are finite and can be described and evaluated.
- A visual user interface state depends on the components that form it and their properties.
- A state is a moment in the visual user interface in which the visual user interface is waiting for a user's action, and where the visual user interface does not change while the user is not interacting with it.

Thus, each state is characterized by the value that the components of an interface acquire according to the following four properties:

- *Visible*: Indicating the visibility of the component. Visible (T) or not Visible (F) on screen.
- ***Active*:** Indicates if the component responds to a user's actions (T) or not (F). If the component has Activo (F) in a state it doesn't exist transition to other state caused by this component.

- *InfI*: This property activates the input of data associated to the component. If the component property has the value of True it will accept data given by the user.
- *InfO*: The output of data associated to the component is activated. If this property has value True the component will show the data sent from the "core" of the application to the user.

Events are user's actions over input hardware devices on the system. These are detected by the system which will respond to them according to the behavioural patterns set for the system to follow. An event is a single user action; for example, drag and drop is the combination of three single actions or events: click, move and release. Similarly, we can define **pre-conditions** and **post-conditions** for the events. If an event is defined using the notation of a visual user interface component and it has no pre-condition, the changes in other components can take place only if the event over this component is produced. For example, in the event RightClick over ComponentTwo the condition is "ComponentOne:Activo(T)" (the property Activo of componentOne has value True); here, the user action over ComponentTwo will not be performed if the value of ComponentOne in property Activo is F. For post-conditions, though, we need to define the values of the properties that must be satisfied in order to reach the next state, that is, we need to define the values of the transitions.

The notation does not limit the events that can be defined for interaction. The HCI engineer can define the events that she/he considers necessary and communicate their meaning to the workgroup. Some examples of basic events that we use are:

- *LeftClick:* click over left Mouse button.
- *RightClick*: click over right Mouse button.
- *ReLeClick*: Release left Mouse button.
- *ReRiClick*: Release right Mouse button.
- *MouseOn*: Mouse pointer over a component.
- *Key(key of keyboard)*: keyboard keys combination.

It is possible to calculate the events from the states with DGAUI from the events of the components in the visual user interface. Thus, if we know that a component is affected by an event, we can build an oriented labelled state graphic of the user interface and establish which will be its following state. The vertexes are the states of the visual user interface and the labelled arcs are the transitions between states. There are two particularly important states for the functionality of the interface:

- *Initial state*: a vertex in which all the associated arcs are exit arcs and in which there are no entrance arcs which may possibly be reached without passing through the initial state.
- *Final State*: a vertex in which all the associated arcs are input arcs and where there is none of exit. The existence of two or more vertexes that only have input arcs is symptomatic of an anomaly in the visual user interface, for it will correspond to two different Final States.

A set of possible derivative states may be obtained from the initial state if a user's action is performed over it. This is done by applying the events over the visual user components with property Activo(T) and by making the associate changes of interaction on the other visual user components. The rest of the states can be obtained from this first set of derivative states

by applying the same processes to each one of the states, until reaching a final state where any visual user component has value True in the properties Active and Visible.
It is possible to identify transitions or arcs, labelled with a component and the event that is applied in order to trigger the state, from previously identified states, during the obtaining stage in the process of state creation in the user interface.
Two states are equal, and therefore, the same state, if all their visual user interface components have the same values in properties Active, Visible, InfI and InfO. The components that belong within a state are the ones that have some function within the state. A visual user interface component belongs to a state if it has some functionality. This functionality may derive from a visual appearance that conveys relevant information about the interface to the user (in this case the component property Active has value True). Or else, the functionality may derive from the fact that the visual user interface component causes changes on the properties of other components when an event is triggered. In any case, one of the advantages of this notation system is that it allows the modification of the visual properties of a component without causing any variations in its behaviour (the user cannot see the component but its behaviour is maintained along intermediate states).
If the modification of the appearance of a visual user interface component is so that it changes the functionality of the component, this would result in the configuration of a different visual user interface component. The interpretation of the appearance of the component in a visual user interface by the user must be unique for each visual user interface component and must also help to identify and clarify its functionality to the user. Otherwise, the component will remain ambiguous and the visual user interface design will be wrong.
This situation is common in interactive models that use interactors for the specification of components. The specification of the interactor is given or defined to specify the different states that may be reached by a component through its interaction with a user. According to the traditional specification of interactors, there is a definite functionality and a unique appearance for each state interactor. There is no model considering multiple renderings or visualizing options for a unique given interactor state. This is so because the specification centres on the dialogue of the interactor instead of on its visual appearance.
In the DGAUI proposal, there exists the possibility of different appearances for the components in the visual user interface—counting on their being triggered by a user's actions by means of visual operations—but the behaviour of the visual user interface component is always the same. Visual operations are, for example, resizing or changes on the size of visual user interface components. By resorting to the DGAUI, the consistence of the visual interface is implicitly maintained, for two visual user interface components created with the same appearance should follow the same behavioural pattern. And even if the appearance of the visual user interface component is altered or modified as the result of a personal choice made by the user, this modification will not affect the behaviour of the visual user interface component.
The DGAUI proposal does not consider the representation of abstract information in the application because this work is oriented to the early stages in the prototyping of the application. The participation of visual user interface components as elements that may allow the user to choose the input and output of information on the user interface is defined including in version 3.04 the domain definition of data or the mask for text input. If a user's action changes dramatically the appearance of a visual user interface component, then this

new appearance would be a new visual user interface component, and hence, a different visual user interface state.
If the visual user interface is correct there should only exist a state in which all the visual user interface components would have the properties Active and Visible with value False (in the final state). Likewise, the initial state of the user interface can be unequivocally identified from the representation of the visual user interface components in the DGAUI-DEF, through an examination of their properties.
In order to simplify its comprehension, here follows a description of the DTD DGAUI-DEF.

Item	Description
<!ELEMENT Composicion (Componente+)>	The composition indicates the topologic relation among the components contained within a containing component.
<!ELEMENT Componente (Alineado?, Equiespaciado?, Subcomponentes)> <!ATTLIST Componente Nombre CDATA #REQUIRED >	Each component is identified by a name and contains a group of subcomponents which facilitates the identification of the alignment characteristics and the equidistant spacing for each subcomponent.
<!ELEMENT Alineado (#PCDATA)> <!ATTLIST Alineado opcion (iz | de | su | in) #REQUIRED >	It signals that the contained components are aligned. There are various possibilities: right or left alignment, or else top or bottom alignment, all of which occur at a given distance in pixels.
<!ELEMENT Equiespaciado (Nombre+)> <!ELEMENT Nombre (#PCDATA)>	It highlights that the contained components are equidistant from one another and from the borders of the containing component.
<!ELEMENT Subcomponentes (Cont+)>	Grouping of the subcomponents in the containing component.
<!ELEMENT Cont (#PCDATA)> <!ATTLIST Cont InfI CDATA #IMPLIED InfO CDATA #IMPLIED >	The component introduces information given by the user (InfI). The component shows information from the application (InfO).

<!ELEMENT Descripcion (Grafico* | Texto* | Enumeracion*)*>	The description of each component in the interface signals that it can be a graphic, a textual component, or a graphic defined by enumeration.
<!ELEMENT Grafico ((Rectangulo | Linea | Circulo | Elipse | Poligono), EstiloLinea?, AnchoLinea?, ColorLinea?, ColorRelleno?, Posición?, Tamano?, Datos?)> <!ATTLIST Grafico Nombre CDATA #REQUIRED Visible (t | f) #REQUIRED Activo (t | f) #REQUIRED InfI (t | f) #IMPLIED InfO (t | f) #IMPLIED >	The graphic component is defined by its name, a primitive, the style, wideness and color of its line, its filling color, its position and size, as well as the type of data that may get into the system or out of the system through such a component.
<!ELEMENT Rectangulo (Coordenada, Coordenada)>	The rectangle derives from two coordinates that determine their diagonal.
<!ELEMENT Linea (Coordenada, Coordenada)>	The line derives from two coordinates that determine its ends.
<!ELEMENT Circulo (Coordenada, Radio)> <!ELEMENT Radio (#PCDATA)>	The circle derives from a coordinate that determines the circle's centre and from a number that indicates its radio.
<!ELEMENT Elipse (Coordenada, Coordenada, AnguloInicio, AnguloFin)> <!ELEMENT AnguloInicio (#PCDATA)> <!ELEMENT AnguloFin (#PCDATA)>	The ellipsis derives from the main rectangle within the ellipsis and from two angles: an onset angle and an ending angle.
<!ELEMENT Poligono (Coordenada, Coordenada, Coordenada+)>	The polygon derives from the coordinates which determine its vertexes.
<!ELEMENT Coordenada (Px, Py)> <!ELEMENT Px (#PCDATA)> <!ELEMENT Py (#PCDATA)>	Every coordinate derives from a given position for the *X* axis and another for the *Y* axis.

<!ELEMENT EstiloLinea EMPTY> <!ATTLIST EstiloLinea Estilo (continua | discontinua) #REQUIRED >	The study of the line admits two values: continuous and discontinuous.
<!ELEMENT AnchoLinea (#PCDATA)>	Pixel size of the primitive line.
<!ELEMENT ColorLinea (#PCDATA)>	Line color in format RGB XXXXXX
<!ELEMENT ColorRelleno (#PCDATA)>	Filling color in format RGB XXXXXX
<!ELEMENT Posicion (Fija | Relativa)> <!ELEMENT Fija (Coordenada)> <!ELEMENT Relativa (OpCRel?, Coordenada)> <!ELEMENT OpCRel (Centrado | (Justificado, Justificado?))> <!ELEMENT Centrado EMPTY> <!ATTLIST Centrado Tipo (h | v | a) #REQUIRED > <!ELEMENT Justificado (#PCDATA)> <!ATTLIST Justificado Tipo (de | iz | su | in) #REQUIRED >	The position of a component determines its location within a containing component or within the device. This position may be fixed or relative, signalled by topologic restrictions in the area OpCRel; in any case, this position derives from a coordinate that signals its left top position (with respect to its containing component or else to the visualizing device) The component is centered according to its containing component. The centering may be horizontal, vertical or both. Alignment of the component on the right, left, top or bottom sides of a containing component according to a given number of pixels.

<!ELEMENT Tamano (Valorx, Valory)> <!ATTLIST Tamano Tipo (fijo | relativo) #REQUIRED > <!ELEMENT Valorx (#PCDATA)> <!ELEMENT Valory (#PCDATA)>	Size of the component within the containing component or within the visualizing device.
<!ELEMENT Datos (Tipo?)> <!ELEMENT Tipo (#PCDATA)> <!ATTLIST Tipo Longitud CDATA #IMPLIED RangoInf CDATA #IMPLIED RangoSup CDATA #IMPLIED Decimales CDATA #IMPLIED >	Conditions to be met by the input and output of data in the interface; attributes may vary depending on the type of data.
<!ELEMENT Texto (Txt, Fuente, TamanoFuente, ColorFuente, EstiloFuente, Posicion, Tamano)> <!ATTLIST Texto Nombre CDATA #REQUIRED Visible (t | f) #REQUIRED Activo (t | f) #REQUIRED > <!ELEMENT Txt (#PCDATA)> <!ELEMENT Fuente (#PCDATA)> <!ELEMENT TamanoFuente (#PCDATA)> <!ELEMENT ColorFuente (#PCDATA)> <!ELEMENT EstiloFuente (#PCDATA)>	The textual component is defined by the text that it presents, its type, size and font color; besides, it also conveys the characteristics of size and position. The size can be fixed or relative.
<!ELEMENT Enumeracion (Fichero, Posicion, Tamano)> <!ATTLIST Enumeracion Nombre CDATA #REQUIRED Visible (t | f) #REQUIRED Activo (t | f) #REQUIRED > <!ELEMENT Fichero (#PCDATA)>	The graphic components by enumeration contain the folder's route that contains the corresponding graphic's image.

<!ELEMENT Dialogo (ItemDialogo*)>	The dialogue about the components in the interface is formed by.
<!ELEMENT ItemDialogo (Precondiciones?, Respuesta)> <!ATTLIST ItemDialogo Elemento CDATA #IMPLIED Evento CDATA #REQUIRED >	Each ItemDialogo is made of the component from which the dialogue derives (Element), the event that triggers it, the preconditions that must be met, and the answer that such event triggers on the component.
<!ELEMENT Precondiciones (Precondicion+)> <!ELEMENT Precondicion (#PCDATA)> <!ATTLIST Precondicion Visible (t | f) #IMPLIED Activo (t | f) #IMPLIED InfI (t | f) #IMPLIED InfO (t | f) #IMPLIED>	The preconditions indicate the condition/s that must be met for the dialogue to be produced.
<!ELEMENT Respuesta (Cambio+)> <!ELEMENT Cambio (#PCDATA)> <!ATTLIST Cambio Visible (t | f) #IMPLIED Activo (t | f) #IMPLIED InfI (t | f) #IMPLIED InfO (t | f) #IMPLIED ProcID CDATA #IMPLIED >	The answer indicates the changes occurred in the properties of other components.

Table 1. DTD DGAIU-DEF

The visual user interface component definition, the topological composition, and the dialogue between components are constants for a visual user interface. The information that is introduced for each state of the visual interface refers to the values of the properties in the visual user interface components.

Once the states which a visual user interface goes through are obtained we use a state graph (multidigraph) to represent the whole set of transitions between states. In the state graph, the vertexes correspond to the visual user interface states and the arcs represent the transitions from one state to another. The arcs are labelled with the name of the visual user interface component and the event that causes the transition.

The XML document (DGAUI-INT) contains the following information:

- *The Topological Composition* of the visual user interface components contained in other visual user interface components.
- *The Information about the evolution of the visual user interface.* All the visual user interface states are defined by the description and properties of its components.

The initial state is obtained from the description of the components in the visual user interface and the rest of other possible states are obtained as part of an automatic process.

- *Set of transitions between states.* This is obtained during the automatic process of states identification.

Thus following, the detailed description of the DTD DGAUI-INT is given:

Item	Description
<!ELEMENT Composicion (Componente+)>	The composition is the same as the one noted in DTD DGAUI-DEF .
< !ELEMENT Estados (Estado+)*>	
<!ELEMENT Estado (Descripcion?)> <!ATTLIST Estado Numero CDATA #REQUIRED >	Each state is formed by a state number and the description of the components that make that state; that is, those components that are visible or active or those components with any of the properties of InfI o InfO valued as T.
<!ELEMENT Descripcion (Grafico* | Texto* | Enumeracion*)*>	The description of each component in the interface indicates that it can be a graphic, a textual component, or a graphic defined by enumeration.
<!ELEMENT Grafico ((Rectangulo | Linea | Circulo | Elipse | Poligono), EstiloLinea?, AnchoLinea?, ColorLinea?, ColorRelleno?, Posición?, Tamano?, Datos?)> <!ATTLIST Grafico Nombre CDATA #REQUIRED Visible (t | f) #REQUIRED Activo (t | f) #REQUIRED InfI (t | f) #IMPLIED InfO (t | f) #IMPLIED >	The graphic component is defined by its name, its primitive, the style, wideness and color of the line, its filling color, its position and size, and the kind of data that can get into the system out of the system through it.
<!ELEMENT Rectangulo (Coordenada, Coordenada)>	The rectangle derives from the coordinates that determine its diagonal.

<!ELEMENT Linea (Coordenada, Coordenada)>	The line derives from two coordinates that determine its ends.
<!ELEMENT Circulo (Coordenada, Radio)> <!ELEMENT Radio (#PCDATA)>	The circle derives from a coordinate that determines its centre and a number that indicates its radio.
<!ELEMENT Elipse (Coordenada, Coordenada, AnguloInicio, AnguloFin)> <!ELEMENT AnguloInicio (#PCDATA)> <!ELEMENT AnguloFin (#PCDATA)>	The ellipsis derives from the main rectangle within the ellipsis and two angles: an onset angle and an ending angle.
<!ELEMENT Poligono (Coordenada, Coordenada, Coordenada+)>	The polygon derives of the coordinate, which determines its vertexes.
<!ELEMENT Coordenada (Px, Py)> <!ELEMENT Px (#PCDATA)> <!ELEMENT Py (#PCDATA)>	Each coordinate derives from a position in the *X* axis and another in the *Y* axis.
<!ELEMENT EstiloLinea EMPTY> <!ATTLIST EstiloLinea Estilo (continua | discontinua) #REQUIRED >	The style of line admits two values: continuous and discontinuous.
<!ELEMENT AnchoLinea (#PCDATA)>	Size in pixels of the primitive line.
<!ELEMENT ColorLinea (#PCDATA)>	Line color in format RGB XXXXXX
<!ELEMENT ColorRelleno (#PCDATA)>	Filling color in format RGB XXXXXX
<!ELEMENT Posicion (Fija | Relativa)> <!ELEMENT Fija (Coordenada)> <!ELEMENT Relativa (OpCRel?, Coordenada)> <!ELEMENT OpCRel (Centrado | (Justificado, Justificado?))>	The position of a component determines its location within a containing component or within the device. This position may be fixed or relative, signalled by topologic restrictions in the area OpCRel; in any case, this position derives from a coordinate that signals its left top position (with respect to its containing component or else with respect to the visualizing device)

<!ELEMENT Centrado EMPTY> <!ATTLIST Centrado Tipo (h | v | a) #REQUIRED > <!ELEMENT Justificado (#PCDATA)> <!ATTLIST Justificado Tipo (de | iz | su | in) #REQUIRED >	The component is centered with respect to its containing component. The centring can be horizontal, vertical or both. Alignment of the component on the right, left, top or bottom side of its containing component a given number of pixels.
<!ELEMENT Tamano (Valorx, Valory)> <!ATTLIST Tamano Tipo (fijo | relativo) #REQUIRED > <!ELEMENT Valorx (#PCDATA)> <!ELEMENT Valory (#PCDATA)>	Size of the component within the containing component or within the visualizing device.
<!ELEMENT Datos (Tipo?)> <!ELEMENT Tipo (#PCDATA)> <!ATTLIST Tipo Longitud CDATA #IMPLIED RangoInf CDATA #IMPLIED RangoSup CDATA #IMPLIED Decimales CDATA #IMPLIED >	The conditions to be met by the input and output data in the interface; attributes may vary depending on the type of data.
<!ELEMENT Texto (Txt, Fuente, TamanoFuente, ColorFuente, EstiloFuente, Posicion, Tamano)> <!ATTLIST Texto Nombre CDATA #REQUIRED Visible (t | f) #REQUIRED Activo (t | f) #REQUIRED > <!ELEMENT Txt (#PCDATA)> <!ELEMENT Fuente (#PCDATA)> <!ELEMENT TamanoFuente (#PCDATA)> <!ELEMENT ColorFuente (#PCDATA)> <!ELEMENT EstiloFuente (#PCDATA)>	The textual component is defined by the text that it presents, its kind, size and font colour; besides, it also conveys the characteristics of size and position. The size can be fixed or relative.

<!ELEMENT Enumeracion (Fichero, Posicion, Tamano)> <!ATTLIST Enumeracion Nombre CDATA #REQUIRED Visible (t | f) #REQUIRED Activo (t | f) #REQUIRED > <!ELEMENT Fichero (#PCDATA)>	The graphic components by enumeration contain the folder's route that contains the corresponding graphic's image.
<!ELEMENT Transiciones (Transicion*)>	
<!ELEMENT Transicion (Precondiciones?)> <!ATTLIST Transicion EstadoInicial CDATA #REQUIRED Elemento CDATA #IMPLIED Evento CDATA #REQUIRED EstadoFinal CDATA #REQUIRED Alcanzable (t | f) #IMPLIED >	Each transition is made of an original transition state, the component (element) over which the event takes place, the event which triggers the transition, and the indicator that signals whether the transition is feasible.
<!ELEMENT Precondiciones (Precondicion+)> <!ELEMENT Precondicion (#PCDATA)> <!ATTLIST Precondicion Visible (t | f) #IMPLIED Activo (t | f) #IMPLIED InfI (t | f) #IMPLIED InfO (t | f) #IMPLIED >	The preconditions indicate the condition/s that must be met for the transition from one state to another to occur.

Table 2. DTD DGAIU-INT

3. Definition of the User Tests

Once the visual user interface is defined, we can proceed to create a notation that will define the tests that must be carried out on a given visual user interface to test its liability. The aim is to record as many parameters and actions as possible and necessary during a user's interaction with a user interface prototype. The DGAUI provides a description of the appearance of the components and the states that may be obtained from a standard rendering device. It also provides renderings of the user tasks and the states that a user can derive by deploying the visual user interface. The first step, then, in automating the evaluation of the visual user interface, and as signalled above, will be to define a notation that may allow us to describe the atomic parts of the evaluation.

As occurred in the case of the DGAUI-DEF, we will use XML in structuring the notation. With this, we will create a DTD that will allow us an easy parsing of notation structures. It is possible to define as many evaluations as it is desired. Each evaluation is formed by a set of user tasks that have to be performed, and the following information should be provided for each of the user tasks:

- A description. (A textual description of the user task for documentation)
- The parameters that have to be assessed and recorded during the evaluation process. These may be *Time Parameters*, for instance, the total amount of time used by the user in accomplishing the task; the time elapsed until the user starts interacting with the task; average time in between events; time elapsed until the first mistake by the user takes place, etc. Other parameters may be *counter parameters*, that is, the number of user events; the number of user mistakes during the evaluation task; the number of times that an error takes place; etc. A final possible parameter to quantify may be the *error parameter*, identifying and controlling types of user mistakes/errors; for example, which is the most frequent user mistake or which is the most frequent mistake or error in a state.
- The different states that a visual user interface will go through during the accomplishment of a task, including a description of the transitions that are generated between states, highlighting the event that must be carried over a given component for the transition to take place. In the prototype that the user will manipulate it will be clearly defined which are the possible states and the resulting transitions.

The description in full of the DTD is as follows:

Item	Description
<!ELEMENT DPU (Prueba)>	Interface test description
<!ELEMENT Prueba (Descripcion,Interfaz,Tareas)> <!ELEMENT Descripcion (#PCDATA)> <!ELEMENT Interfaz (#PCDATA)>	The test is defined departing from a description of the very test, of the identification of the interface over which the test is done and the limitation of the activities that the user will have to perform over it.
<!ELEMENT Tareas (Tarea+)> <!ELEMENT Tarea (Descripcion, Parametros, Estados)>	Each task that the user has to perform as part of the test includes its own descriptions, the parameters that will be measured for the fulfilment of the test and the states to be measured by these parameters.

<!ELEMENT Parametros (Parametro*)> <!ELEMENT Parametro (Nombre, Tipo, Estado_Inicial, Estado_Final)>	The parameters are identified by a name and belong to a type.
<!ELEMENT Nombre (#PCDATA)>	Name of the parameter
<!ELEMENT Tipo (Tiempo|Contador|Error)> <!ELEMENT Tiempo EMPTY> <!ATTLIST Tiempo Caracteristica (TTotal | TPrimerEvento | TMedioEntreEventos | TMedioReaccion | TPrimerError) #REQUIRED > <!ELEMENT Contador EMPTY> <!ATTLIST Contador Caracteristica (NumEventos | NumErrores | NumOcuError) #REQUIRED > <!ELEMENT Error (Descripcion?, ID_Estado?, Evento?, Componente?)> <!ATTLIST Error Caracteristica (IdError | EstadoError | ErrorMasFrec) #REQUIRED > <!ELEMENT ID_Estado (#PCDATA)>	The parameters can be of three different types: Time: referring to temporal aspects. Counter: referring to quantitative values related to the user's actions. Error: referring to failures occurred during the task's performance.
<!ELEMENT Estado_Inicial (#PCDATA)>	State in which the assessment of the parameter begins
<!ELEMENT Estado_Final (#PCDATA)>	State in which the assessment of the parameter ends.
<!ELEMENT Estados (Estado*)> <!ELEMENT Estado (Parametros?, Transicion)>	
<!ELEMENT Transicion (Estado_Inicial, Estado_Final, Componente, Evento)> <!ELEMENT Componente (#PCDATA)> <!ELEMENT Evento (#PCDATA)>	The transition is identified by the onset state, the component over which the event must be produced, the event that has to take place and the final state derived from its production

Table 3. DTD DPU

In the following lines, we provide a detailed description of each ot the parameters that can be assessed:

Time

- Total time (TTotal): time that the user devotes in implementing the task.
- Time Firs Event (TPrimerEvento): Time that the user devotes in accomplishing an event the first time that she/he has contact with the interface.
- Average time between events (TMedioEntreEventos): Time that the user spends in between two successive events.
- Average time of reaction (TMedioReacción): Time that the user takes in accomplishing a new event after making a mistake.

Counter

- Number of events (NumEventos): number of successful and failed events that the user needs to accomplish the task.
- Number of mistakes (NumErrores): total number of mistakes that were generated during the task.
- Number of appearance of the same mistake (NumOcuError): number of times that each mistake has taken place.

Error

- Error identifier(IdError): textual identification of the error.
- Error State(EstadoError): state in which the error has taken place.
- Most frequent error (ErrorMasFrec): the most frequent error in the test.

4. Evaluator for the user's actions (EUA)

The EAU tool allows the user to dynamically evaluate the visual user interface usability. This evaluation is carried out trough an interactive simulation of the interface. From the abstract notation of DGAUI (DGAUI-INT) we can build the visual appearance of the states in the interface and simulate possible user actions over the components. With de EAU notation it becomes possible to define the user tasks that a user can try. The simulation reproduces the visual appearance of the interface following the user tasks described in section 3.

Through the interaction of the user with the simulator, a great amount of information is recorded according to the parameters set forth in section 3. This information is stored in a data base for its future study and analysis. Hence, this tool allows the HCI engineer to define as many assessments as may be thought necessary, obtaining quantitative information about the actual impact of a user on the visual user interface. Normally, the HCI engineer explains to the user which are the aims to be reached while evaluating the visual user interface by means of the prototype. Then, with the information obtained, the HCI engineer can determine if the interface has any problems before starting its coding.

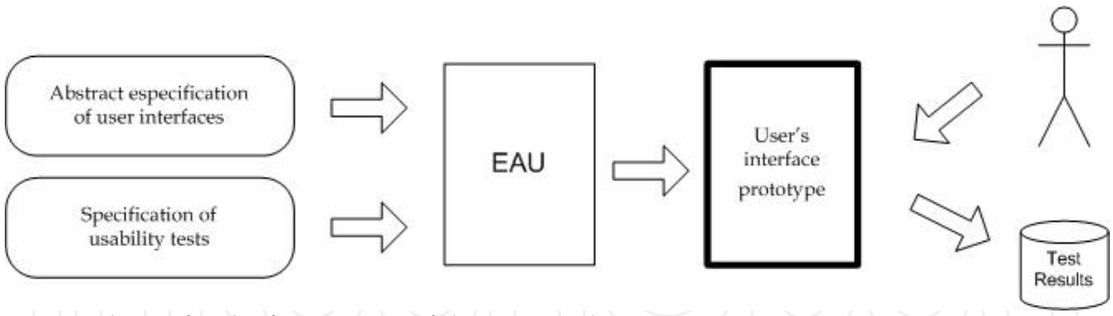

Fig. 1. Schema for the functioning of the EAU tool.

The EAU tool shows a simple interface with two basic functionalities:

- Load Interface: which allows the selection of an XML file that contains the visual user interface description (DGAUI-DEF and GDAUI-INT) and which generates the visual appearance of the states in the user interface.
- Test Interface: necessary in order to evaluate the visual user interface, by selecting the individual user task definitions that complete the evaluation (AEU XML file). By means of these definitions, and through the configuration of the parameters in order to have access to the data base, the simulation is executed. The information about each user's interaction with the prototype is stored in the data base for study.

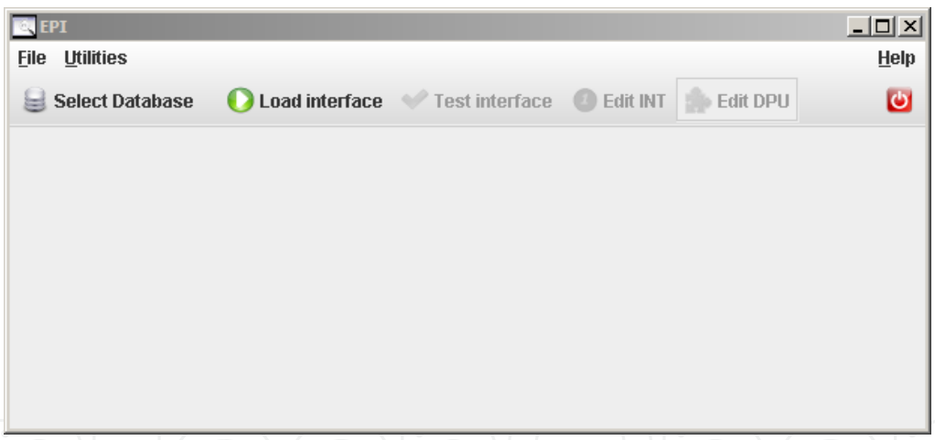

Fig. 2. Main window for the application.

Figure 3 Shows a visual user interface state generated by the EAU tool from a DGAUI description. The interface example corresponds to an average text processor.

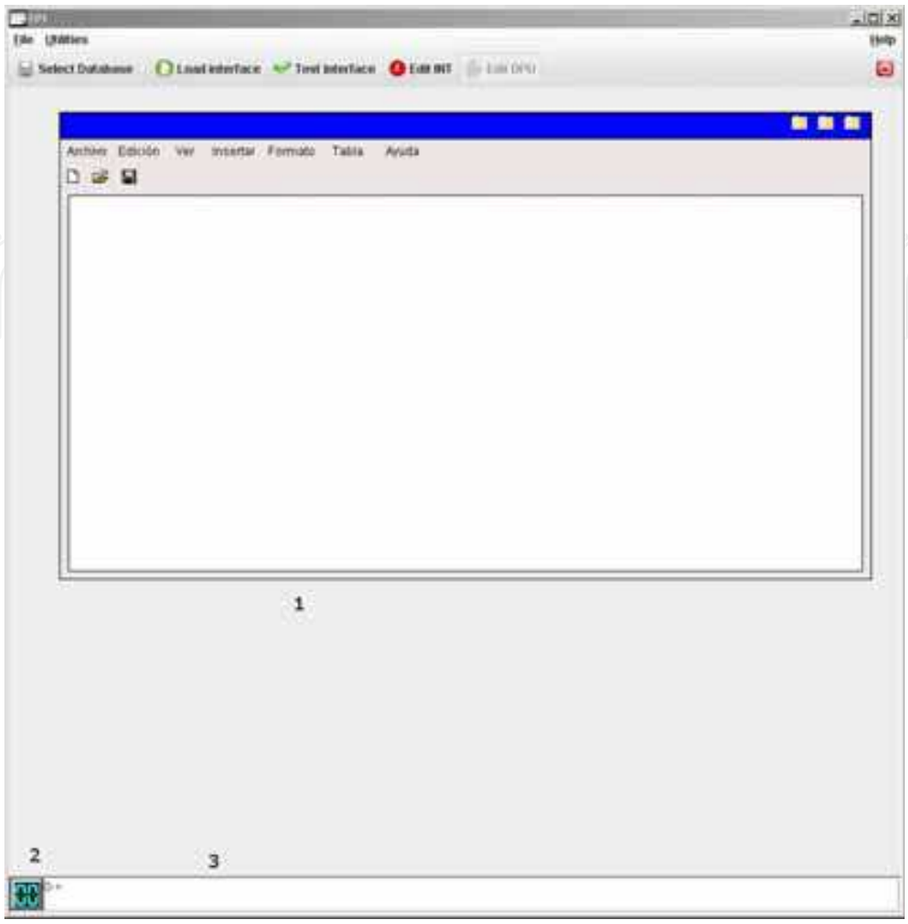

Fig. 3. Text processor prototype.

Zone 1: Shows the interface.
Zone 2: Shows the state that the interface is in.
Zone 3: Shows the record of states that the interface has gone through.

We will here take as an example the definition of a test that will assess the accomplishment of a task consisting on the selection of the option New in the File menu. We will evaluate the following parameters: Total time devoted for the execution of the task; number of events; time in which the first event takes place; average time in between events and total number of errors produced throughout the task.

The defining code is as follows:

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!DOCTYPE DPU SYSTEM "DPU.dtd">
<DPU>
<Prueba>
 <Descripcion>Prueba1</Descripcion>
 <Interfaz>Editor</Interfaz>
```

```
<Tareas>
 <Tarea>
  <Descripcion>Selection of the option New in the File menu</Descripcion>

  <Parametros>
  <Parametro>
    <Nombre> Total time devoted for the execution of the task </Nombre>
    <Tipo>
     <Tiempo Caracteristica="TTotal"/>
    </Tipo>
    <Estado_Inicial>0</Estado_Inicial>
    <Estado_Final>10</Estado_Final>
  </Parametro>
  <Parametro>
    <Nombre> Number of events </Nombre>
    <Tipo>
      <Contador Caracteristica="NumEventos"/>
    </Tipo>
    <Estado_Inicial>0</Estado_Inicial>
    <Estado_Final>10</Estado_Final>
  </Parametro>
  <Parametro>
    <Nombre> Time in which the first event takes place </Nombre>
    <Tipo>
     <Time Caracteristica=" TMedioEntreEventos "/>
    </Tipo>
    <Estado_Inicial>0</Estado_Inicial>
    <Estado_Final>10</Estado_Final>
  </Parametro>
  <Parametro>
    <Nombre> Average time in between events </Nombre>
    <Tipo>
     <Time Caracteristica=" TMedioEntreEventos "/>
    </Tipo>
    <Estado_Inicial>0</Estado_Inicial>
    <Estado_Final>10</Estado_Final>
  </Parametro>
  <Parametro>
    <Nombre> Total number of errors produced throughout the task </Nombre>
    <Tipo>
     <Contador Caracteristica="NumErrores"/>
    </Tipo>
    <Estado_Inicial>0</Estado_Inicial>
    <Estado_Final>10</Estado_Final>
  </Parametro>
 </Parametros>
```

```
  <Estados>
   <Estado>
    <Transicion>
     <Estado_Inicial>0</Estado_Inicial>
     <Estado_Final>2</Estado_Final>
     <Componente>Marc_Archivo</Componente>
     <Evento>MouseOn</Evento>
    </Transicion>
   </Estado>
   <Estado>
    <Transicion>
     <Estado_Inicial>2</Estado_Inicial>
     <Estado_Final>9</Estado_Final>
     <Componente>Marc_Archivo_Selec</Componente>
     <Evento>LeftClick</Evento>
    </Transicion>
   </Estado>
   <Estado>
    <Transicion>
     <Estado_Inicial>9</Estado_Inicial>
     <Estado_Final>10</Estado_Final>
     <Componente>Menu_Archivo_op_nuevo</Componente>
     <Evento>MouseOn</Evento>
    </Transicion>
   </Estado>
  </Estados>
 </Tarea>
</Tareas>
</Prueba>
</DPU>
```

Once the test is finished, the results would appear on the screen.

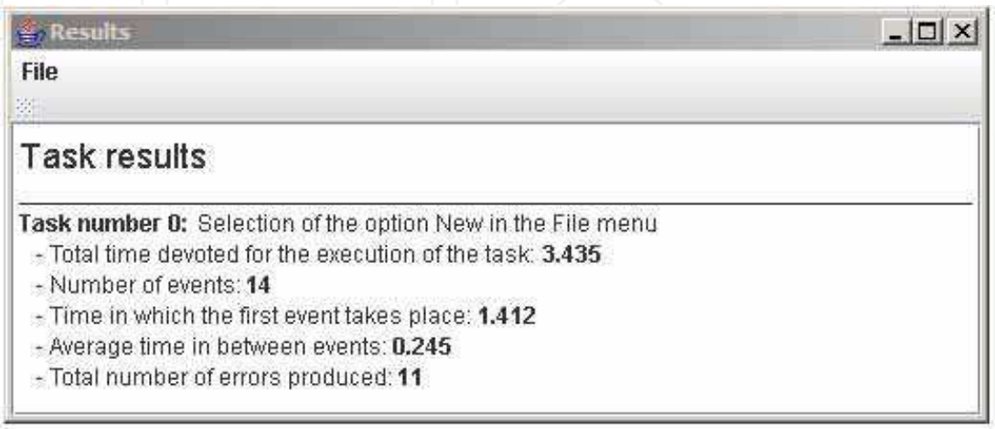

Fig. 4. Window showing the results of the test in EAU

5. Conclusions

In this chapter we have presented an abstract representation of user interfaces particularly designed for visual interactive systems. The focus of this representation has been the visual aspect of the user interface because, for the user, it remains the most important part. It is through the interaction with the appearance of a user interface that the user obtains information from the user interface, and reacts to it.

Another aspect explored in this essay is the relevance of the appearance of the behaviours in the visual user interface components (with varying sizes and positions). Such variations allow for the same state of a component to render different functions.

We have proved that it is possible to describe a set of interface user tasks in a notation; similarly, we have shown how to automatically generate a prototype with which to assess the user interface behaviour with its users and its acceptability by them in a quantitative manner.

6. References

Eason, K. (1988). Information Technology and Organizational Change. Taylor and Francis, London.

Gartner Group (1994). Annual Symposium on the Future of Information Technology, Cannes 7-10 November 1994.

Gómez Carnero, S. (2008). Sistematización de la validación de interacción del usuario sobre la visualización en interfaces de usuario usando especificación abstracta. PhD Thesis. Universidad de Vigo. March 2008.

IBM (1993). IBM Dictionary of Computing. McGraw-Hill.

ISO (1992). Software product evaluation quality characteristics and guidelines for their use.

ISO (1993). Ergonomics Requirements for Office Work with Visual Displays Terminals: Guidance and Usability.

Molich R. y Nielsen J. (1990). Heuristic evaluation of user interfaces. Proceedings of ACM CHI 1990. Seattle, WA, April 1990, Pág. 249-256, 1990.

Myers B. A. y Rosson M. B.(1992). Survey on user interface programming. CHI'92 Conference Proceedings on Human Factors in Computing Systems (Bauersfeld P.,Bennett J. y LYNCH G. eds.), page. 195-202. ACM Press, Nueva York, NY.

Nielsen, J. (1993). Usability Engineering. Academic Press, London.

Nielsen J. and Mack R. L. (1994). Usability Inspection Methods. John Wiley and Sons, New York, NY.

Preece, J., Rogers, Y., Sharp, H., Benyon, D., Holland, S. And Carey Tom. (1994). Human-Computer Interaction. Addison-Wesley Publishing, Reading, Mass.

Rodeiro Iglesias, J. (2001). Representación y análisis de la componente visual de la interfaz de usuarios. PhD Thesis. Universidad de Vigo. September 2001.

Shackel, B. (1986). Ergonomics in designing for usability. In M. D. Harrison and A. Monk, editors. People and Computers: Designing for Usability. Cambridge University Press.

Wharton C. (1994). The cognitive walkthrough method: a practitioner's guide. Usability Inspection Methods (NIELSEN J. y MACK R. L. eds.). John Wiley & Sons, New York, NY, Page. 105-140, 1994.

Human-Computer Interaction
Edited by Inaki Maurtua

ISBN 978-953-307-022-3
Hard cover, 560 pages
Publisher InTech
Published online 01, December, 2009
Published in print edition December, 2009

In this book the reader will find a collection of 31 papers presenting different facets of Human Computer Interaction, the result of research projects and experiments as well as new approaches to design user interfaces. The book is organized according to the following main topics in a sequential order: new interaction paradigms, multimodality, usability studies on several interaction mechanisms, human factors, universal design and development methodologies and tools.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Susana Gomez-Carnero and Javier Rodeiro Iglesias (2009). Evaluation of the Interface Prototypes Using DGAIU Abstract Representation Models, Human-Computer Interaction, Inaki Maurtua (Ed.), ISBN: 978-953-307-022-3, InTech, Available from: http://www.intechopen.com/books/human-computer-interaction/evaluation-of-the-interface-prototypes-using-dgaiu-abstract-representation-models

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen