

## A Top-Down Visual Approach to GUI development

ROSANNA CASSINO, GENNY TORTORA, MAURIZIO TUCCI, GIULIANA VITIELLO  
Dipartimento di Matematica e Informatica  
Università di Salerno  
Via Ponte don Melillo – 84084 Fisciano  
ITALY

*Abstract:* - In this paper, we propose an advance in the field of computer-aided development of interactive applications, which consists in a methodology that integrates the advantages of graph-based design with a visual construction of software applications using component assembly mechanisms. We have exploited such technique enhancing the visual approach to develop the graphic and interactive features of an application interface presented in *TAGIVE* (**T**ool for the **A**ided **G**eneration of **I**nteractive **V**isual **E**nvironments). The top-down development approach prevents possible incorrectness and incompleteness error. The visual development model for both static and dynamic aspects makes more intuitive the design and the implementation. This has the effect to support novice programmers in the development of software applications while reducing expert designers' workload.

*Key-Words:* - user interface design, graphical environments, interactive applications, event management, correctness and completeness.

### 1 Introduction

Traditionally, general-purpose programming languages, such as C and C++, have been employed by skilled programmers to develop user interfaces. In recent years a large number of software tools have been proposed to reduce time and effort needed to develop interactive systems, providing support to rapid GUI prototyping as well as to system development [5]. However, most of such tools still lack a clear-cut separation of concerns between the user interface design and the development of the underlying interactive application. As a matter of fact, apart from a set of common widgets with predefined interactive behaviors, the implementation of the system dynamic behavior upon user's interaction, remains a difficult task, which requires programming experience.

The present research aims to extend the degree of support that GUI developers receive from a user interface development environment. We present a methodology to design and implement interactive visual environments, simple to use, flexible and which requires a limited knowledge in the field of graphical programming and a short training period. *TAGIVE* (**T**ool for the **A**ided **G**eneration of **I**nteractive **V**isual

**E**nvironments) is the prototype system that implements the proposed technique. The tool was first introduced in [1] and has now been enhanced with a module for the event management, which better supports a visual approach to the construction of interactive applications. When developing an interactive application, a directed labeled graph is used to provide a top-level design of the interactive flows characterizing the application. Then, starting from that top-level design, the tool is able to customize the event handler module, so that all the features related to an event (i.e., the source object, the action performed on it, and its effect on the application itself) can be easily detailed on a usable form-based interface. The unified framework characterizing *TAGIVE* integrates the advantages of graph-based design with a visual construction of software applications using component assembly mechanisms.

The static features and the dynamic behaviors of the target application are formally specified at a high abstraction level, in terms of a grammar model, named Attributed SR-Action grammars, introduced in [2] to describe interactive visual languages. Such grammars consist of rules that model the system transitions upon

user interaction, adequately capturing the dynamics and evolution of the system. Such a formal specification of the interactive visual application is useful to perform automatic checks on correctness and completeness of the development process<sup>1</sup>.

The rest of this paper is organized as follows. Section 2 describes the proposed methodology to design interactive visual applications and presents an overview of the architecture of the current system prototype. Section 3 shows how TAGIVE supports the suggested methodology on a sample interactive application. Section 4 explains how correctness and completeness in the developed application are ensured by the system. Section 5 contains a discussion on related work and Section 6 presents some concluding remarks.

## 2 The two-level design approach

Before describing the top-down development methodology characterizing TAGIVE, we need to explain the terminology used. For the purpose of our work, an *interactive visual application* is defined as a set of *scenes* and a set of *external components*, related to each other on the basis of the interactions performed. A *scene* is made up of elementary components representing interface widgets and arranged on the scene according to a certain layout. Each *elementary component* appearing in a scene is said to be *dynamic*, if some event is associated with it or *static* otherwise. *External components* are instead referred to any file that can be invoked by some dynamic component of a scene (e.g., image files, text files, video clips, sound files, web pages, etc.). Fig. 1 sketches the structure of an interactive visual application. Such a model is exploited in the system in order to allow the two-level approach in the application design and development. The former level is focused on the between-scene or scene-to-external component interactions, while the latter details such interactions in terms of the dynamic components composing each scene, with the possibility to switch from one level to the other at any time during the development process.

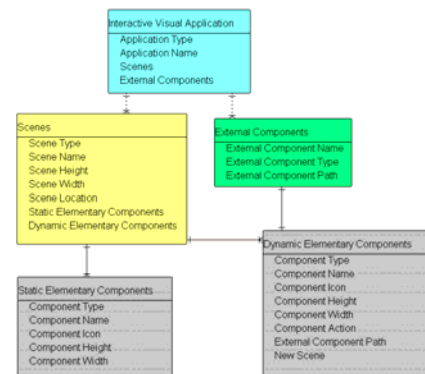


Fig. 1 - Structure of an interactive visual application.

The design methodology employed in the system integrates the advantages of a graph-based design technique with the benefits coming from a visual construction of applications using component assembly mechanisms. Such integration is aimed at supporting the development of interactive visual applications by directly relating the design phase to the implementation phase. As a matter of fact, a graph-based design approach is used to build the “application map”, in terms of a top-level transition diagram representing all the possible interaction paths. Indeed, oriented and connected graphs are a very good means to design and examine the number of scenes and the best interaction paths of an interactive visual application. For example, in an e-learning application, a designer may build the application map, so that any student shall eventually cross a certain scene, or reach a certain scene through established paths. The idea underlying the proposed methodology is that the application map directly guides the development of the interactive application. The implementation is based on a component assembly technique applied to the different scenes and on an event-handling mechanism used to implement inter-scene interactions. The generated interactive visual application prototype is expressed in terms of an XML-based language, which provides the additional benefits of a portable and version resilient application.

Fig. 2 shows an overview of the architecture of the current system prototype. The two levels of the development process are implemented by means of the Map Editor and the Scene Editor, respectively. The double-ended arrow connecting the two modules indicate that an iterative process is possible when developing the interactive application, which allows for a desirable incremental approach based on user’s feedback.

<sup>1</sup> A description of the formalism is out of the scope of the present paper, but the reader who wishes to get further insight on such aspect may contact the authors, who would be happy to provide the complete formal definition of the visual languages those grammars are able to specify, and examples of interactive systems formally specified.

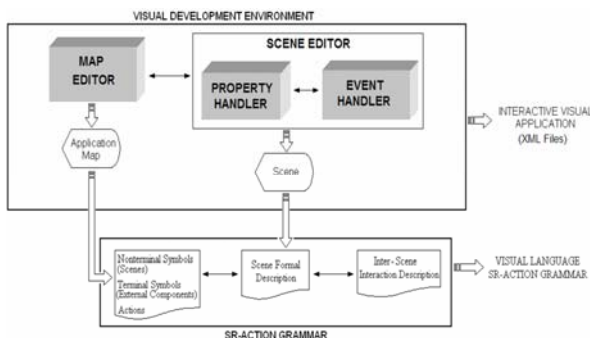


Fig. 2 - The proposed system architecture <sup>2</sup>.

As shown in the figure, besides the main development environment, the system exhibits a formal grammar environment. In fact, as a “side effect” of the overall development process, it also generates the formal specification of the developed interactive visual application, in terms of the SR-Action grammar model. The use of a grammatical formalism allows us to express the interactive application in terms of a formal visual language, specified by a complete set of syntactic and semantic rules which precisely describe the structuring of any scene and the dynamic mechanism characterizing the associated interactions. Such a formal specification of the interactive visual application is useful to perform automatic checks on correctness and completeness of the development process. However, for the sake of brevity, we have decided not to describe the grammar formalism and its benefit here, but rather focus on the Visual Development Environment characterizing the system prototype. In the following section we outline its basic architectural components and provide a description of the whole development process on a simple example of interactive visual application.

### 3 The visual application development

We illustrate how the designer may use the TAGIVE to build an interactive visual environment, in a guided and fully visual manner, on a sample interactive application. Suppose the designer wants to develop an application whose storyboard is sketched in Fig. 3. Its main scene represents a classroom where certain interaction paths can start. In particular, in the Teacher’s Desk scene an action performed on the

central teacher image should cause a video clip to play, whereas an action performed on the bottom-right button should cause a transition to the Student View scene. Once there, the user may traverse the exit door to terminate the application, or the right-hand door to move to the School Library scene. In the School Library, if an action is performed on the central librarian, a new video clip about the borrowing policies is triggered. If a click is performed on the switch button on the left of the door, the spotlight at top of the scene is switched off. Finally, if the door of the library is traversed, the initial classroom scene (i.e., Teacher’s Desk) is reached again.

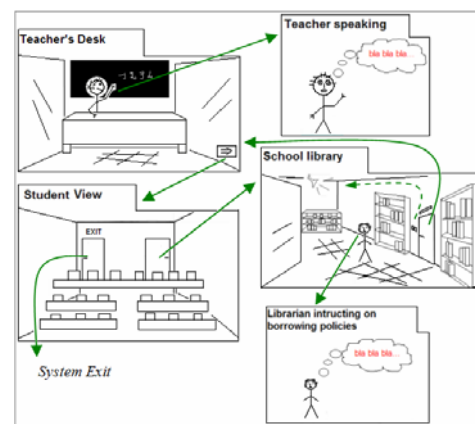







Fig. 3 - The storyboard of a sample application.

Starting from the design process, the designer draws the structure of the application by operating within the application map, adding, modifying or deleting nodes and connections between them. This is done within the Map Editor (see Fig. 4), which provides the graph editing work area (the Work Panel) that allows the designer to include nodes in the graph, which visually represent the scenes or external components of the application, and directed edges, which represent the interaction relationships between them<sup>3</sup>. Thus, the map nodes could represent containers like panels, and frames, or objects like multimedia clips, sounds, pictures, and web pages. All such components are represented by icons which can be selected from the Components List, provided by the editor. As for the edges, they are labeled using items taken from the Action List, which contains the actions that can be

<sup>2</sup> The shape  indicates a tool module. The shape  indicates a graphical visualization unit of an interactive visual application. The shape  indicates formal specification documents. The arrow  denotes the interconnection between tool modules. The arrow  denotes the production of an output.

<sup>3</sup> Presently, we are facing the problem of keeping effective the map visualization, as the number of the nodes and the the number of edges increase. We plan an improvement of the Map Editor module to manage several view levels of the “sub-maps” representing the subsystems composing the overall application.

used to define the interactions between pairs of scenes or between scenes and external components. Moreover, cycling edges are used to define intra-scene interaction. Thus, for example the click performed on the spotlight switch in the School Library scene, is represented in Fig. 4 by a loop transition on the corresponding node.

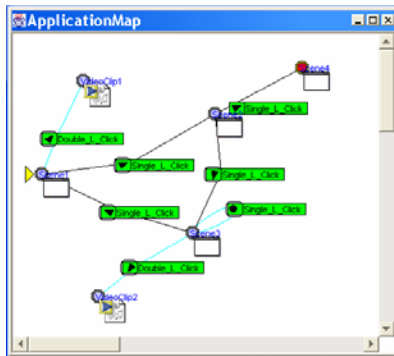


Fig. 4 - The application map.

At formal level, in this phase, TAGIVE identifies the start symbol, the terminal, the non terminal symbols and the set of actions which will characterize the interactive behaviour of the application being designed. Such elements will be used by the system to define the initial form of the underlying SR-Action grammar [2]. Moreover, for each edge in the map, the system keeps track of the corresponding source node and, afterwards, uses such information to perform automatic checks for correctness/completeness when the designer terminates the language specification task. When the application map is finished, for each node representing a scene, the designer exploits the Scene Editor to add the necessary elementary components to the corresponding container. In particular, for each scene, in a visual manner the designer selects the basic frame or the panel that represents a scene, from the Containers palette of the Scene Editor and then, by the Property Handler frame shown on the left, he/she specifies the corresponding physical attributes (e.g., the background colour or image, the size, and the name.). From the Elementary Components palette, he/she drags visual objects to be positioned in the frame (i.e., widgets such as buttons, labels, menus, etc...) and manages the associated properties from the Property Handler frame. Fig. 5 shows the construction of the first scene of the example. The second and the third scenes are developed analogously.



Fig 5. - Implementation of the scene Teacher's Desk.

Each elementary component appearing in a scene is said *dynamic* if the designer associates an action to it. In this perspective, to guarantee correctness in the development process we propose a top-down technique to control the insertion of actions in any scene. In particular, the designer establishes at the map level the actions that will be allowed within a scene. At the lower-level (the phase of the scene development), those actions will be the only feasible actions the designer may associate to each dynamic component in order to detail the related event. To do so, he/she first selects the Events item from the Control Panel dropdown list of the Scene Editor. As a result, the Event Handler displays a form, by which the designer can specify the desired interaction in a visual manner. As an example, Fig. 6 shows the visual implementation of the event corresponding to the transition from Student View to School Library.

In particular,

- The first field in the Event Handler form allows to select the Source Object (i.e., the dynamic component receiving the action) from the list of elementary components featuring in the scene. In the example, the source object is the door1 label of the scene Student View.
- The second field, named Target Object, is used when the action causes changes of attributes in some elementary component of the same scene, which is not the case for the considered action.



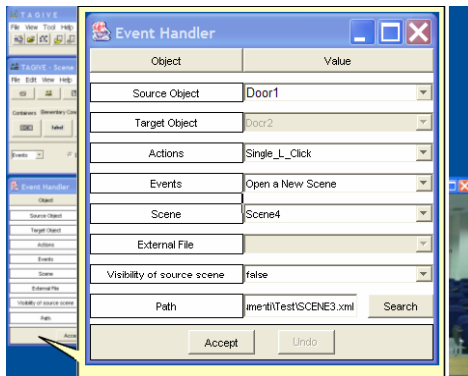


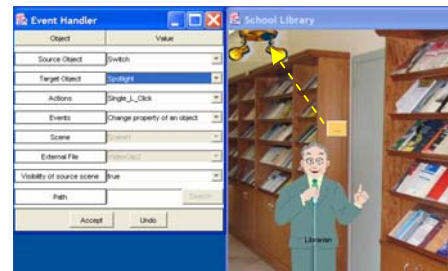
Fig. 6 –The “Open a New Scene” event.

- The Actions field allows to select the action from the list of possible actions associated with the chosen source object. In the example, a “Single\_L\_Click” action may be associated to the label door1 of the Student View.
- By means of the Events field, the designer can indicate the type of event associated with the action, selecting among “Open a New Scene” (as in Fig. 6), “Change Property of an Object”, and “Launch an External File”.
- The field Scene indicates the target scene that is reached from the source scene, when the action is performed.

If the type of event associated with an action is “Launch an External File”, the designer specifies the corresponding path, possibly activating the Search button of the Path field. Due to space limits, we do not show all the images corresponding to the given example, but a complete demo of the system can be downloaded from our web site at the URL <http://www.dmi.unisa.it/people/vitiello/>.

In the application map shown in Fig. 4 a loop transition labelled “Single\_L\_Click” goes out from the Scene3 node.

Then, at the development lower level the designer can to manage an event corresponding to a change of properties of an object in the same scene. In this case, he/she selects the option “Change Property of an Object” in the Events field of the Event Handler form. Let us consider again the example of the single click on the switch button in the School Library, which causes the spotlight to be switched off. Actually, such event corresponds to the change of the icon image representing the spotlight at the top of the scene. Fig. 7 shows two steps that allow to manage the implementation of this type of event in a totally visual manner.



(a)



(b)

Fig. 7 –The “ Change Property of an Object” event.

The designer first selects the source object, the target object, whose properties should be modified, the action and the type of event to manage (see Fig. 7.a). Then the designer clicks on the Accept button and the system displays the Property Handler corresponding to the target object. Once he/she sets the new property values, he/she presses the Apply button to confirm the event management specification (see Fig. 7.b). If the designer wants to modify the properties also of other objects in the scene, he selects another element in the Target Object field and remakes the process described before.

#### 4 Correctness and completeness checks

The described methodology aids the designer in the development of an interactive visual application guaranteeing correctness and completeness of the process and preventing possible errors from the first stages of design. In particular, the presence of nodes not connected to the others in the graph underlines scenes or files expected but not achievable in the visual environment. Again, edges representing interactions, which have not been specified at the lower level, will be highlighted in the map.

The formal specification of an interactive visual application is useful to perform automatic checks on correctness and completeness of the development process. As a matter of fact, the two phases in the construction of an application, namely the production of the Application Map and the detailed composition

of its scenes, correspond to two different phases of the formal specification. In particular, starting from the Application Map, the set of nonterminal symbols representing scene nodes, the set of terminals representing external file nodes and the set of actions representing edges, are generated. Moreover, the association of each edge in the map with the corresponding source and target nodes is internally stored. Such information is later exploited by *TAGIVE* to directly control the lower level of the construction, i.e. the composition of each scene and the management of the events occurring in it. This allows the system, throughout the development phase, to prevent incorrectness and to check for completeness of inter-scene interactions.

It is worth noting that apparent nondeterministic situations may arise from two edges labelled by the same action which stem out of one node in the map, e.g., two “Single\_L\_Click” edges stemming out of the same scene node. In such a case, *TAGIVE*, would prevent the designer from associating the same action twice with the same dynamic component in the source scene. Thus, once a “Single\_L\_Click” action has been associated with a button in the scene, and the corresponding event has been managed, any other attempt to associate a “Single\_L\_Click” action with that button would fail. Of course, the second “Single\_L\_Click” action may instead be associated to any other dynamic component in the scene.

## 5. Related work and final remarks

Several graphical toolkits and user interface development environments are today available to support developers in the construction of interactive visual applications. However, with most of those tools, the specification of the interactive runtime behaviour of the system is still a cumbersome and tedious task.

Myers was one of the first researchers who felt that software developers should be better supported in the specification of the system dynamic behavior. In [3] he proposed a model for handling input devices in interactive applications, by means of objects (named *interactors*) which encapsulate the details of the related events. However, little work has been done since then in that direction. Recently, more sophisticated model-view-control architectures have been employed with UIDE environments, requiring the specification of a dialogue control module for each UI component included in the interface design. *MOBI-D* (Model-Based Interface Layout Editor) is one of the

first systems conceived to provide designers with greater flexibility than traditional toolkits, by allowing developers to define interface models as organized, reusable, and integrated computational units [4]. However, no underlying formal specification is employed to control the whole development process.

The *TAGIVE* system we have described comes from the integration of two kinds of User Interface Management Systems (UIMS), those based on object languages and those based on grammars. As described in [3], object language based UIMS use a natural approach to the interface considering its elements as pairs of objects that interact, whereas grammar based UIMS allow to specify the interface through a set of rules in a grammar of a formal language. Thanks to the combination of the two approaches, *TAGIVE* turns out to be *flexible*, and the components of the realized applications are *easy to reuse*. Moreover, *high abstraction level of specification* is automatically achieved for the generated visual language. Indeed, the developer is allowed to implement the interactive visual application visually and then, in automatic manner, the corresponding grammar is produced, by which *TAGIVE* controls completeness of the provided design.

### References:

- [1] R. Cassino, G. Tortora, M. Tucci, G. Vitiello, “A Tool for the Aided Generation of Interactive Visual Environments”, 2003 International Conference on Distributed Multimedia Systems (DMS'2003) - Florida International University.
- [2] R. Cassino, G. Tortora, M. Tucci, G. Vitiello, “SR-Task Grammars: A Formal Specification of Human Computer Interaction for Interactive Visual Languages”, 2003 Symposium on Visual Languages and Formal Methods (VLFM '03).
- [3] B. A. Myers, “User interface software tools”, *ACM Transactions on Computer-Human Interaction*, 2, 1, March 1995, pp. 64-103.G.
- [4] R. Puerta, E. Cheng, T. Ou, and J. Min, "MOBILE: User-Centered Interface Building," *Procs. CHI99: ACM Conference on Human Factors in Computing Systems*. Pittsburgh, 1999, pp. 426-433, ACM Press.
- [5] B. Shneiderman, C. Plaisant, *Designing the User Interface: Strategies for Effective Human-Computer Interaction*, 4/E, Addison-Wesley, 2005.