AUTOMATIC GUI GENERATION FOR WEB BASED INFORMATION SYSTEMS

Nicola Aloia, Cesare Concordia, Maria Teresa Paratore
Institute of Information Science and Technologies
Italian National Research Council
Via Moruzzi, 1 - CNR Research Area, Pisa (56100) Italy
nicola.aloia@cnuce.cnr.it

ABSTRACT

The HCI subsystem is an essential component of an Information System, whose success inside an organization strongly depend on user acceptance. A good UI design considering usability and accessibility is to the base of a system implementation that could be quite complex, particularly when a user interaction with different devices is required. The cost of such implementation, especially for large system, could be relevant, so any tool or method for automatic (or semi-automatic) GUI generation would be welcomed. In Web Based Information Systems, data entry Uis, interacting with Data Manager are a significant part of the whole HCI subsystem and usually implement a limited set of metaphors to allow reuse of components in different contexts. For this kind of UIs the complexity of automatic generation can be reduced. In this paper we first present a review of the existing open-source tools for automatic GUI development, then, we present a case study of GUI automatic generation in a Public Access WIS context.

KEYWORDS

Human Computer Interaction, WWW/Internet Applications, WWW/Internet Case studies, Web Engineering.

1. INTRODUCTION

The evolution of network technologies has been a major factor in improvements of Information Systems (IS) technologies over last years. Particularly, technologies stemming from Internet have achieved today commodity status in building Web-based ISs (WIS)[4]. The ISs' renovation has involved all its components: from data storage to workflow management and so. Considering the HCI subsystem's point of view many improvements have been reached, mainly because of the evolution of graphic browser and their ability to implement platform neutral rendering languages, but also thanks to introduction of powerful "multi platform frameworks" allowing developers to build "mobile thin clients". Only few years ago ISs' features like "multi-client" access or nomadic connection, were largely theoretical, today, thanks to the Internet technologies they are common requirements for ISs. Despite of technological evolution many issues regarding HCI are today still opened, rather some of them like accessibility have raised in importance especially in ISs managed by public organizations whose aim is to inform and/or provide services to citizens. In such a context, many attempts can be found in literature [5] and are currently being in place to build tools for automatic generation of cross platform user interfaces. However our experiences persuaded us that obtained results aren't good enough. For some kinds of UIs, anyway, issues can be simplified, hence acceptable results can be obtained: this is the case of user interfaces for data entry, which are a relevant component in the development of a WIS 's interaction subsystem, as far as development times and costs are concerned.

2. OPEN SOURCE TOOLS FOR AUTOMATICALLY DEVELOPMENT OF GUIS: A REVIEW

Usually, once the application domain of a IS has been fixed, the first step for a GUI designer is to choose a *metaphor* [1] on which base the design of data entry GUIs. That is the designer must define the fundamental concepts, terms, and images by which command/control are communicated and information will be recognized, understood, and remembered by IS users. Starting from the defined metaphor a set of basic graphic components is designed; these objects will be combined to obtain all the desired user interfaces. The capability to describe these interfaces in terms of their "atomic" components by means of a declarative language that allows their automatic generation (or semi-automatic generation with poor human intervention) even only for the structural part seems to be, for data entry UIs, not only feasible, but also essential in a project's economy. This subject has been interesting both open-source and commercial software world. Here we focus our attention to open-source projects devoted to GUIs development for applications based on Java technologies. A certain number of such projects, some of which based on XML, are available at the moment. These tools are quite different each other and they can meet a wide range of needs.
If the developer's priority is to avoid Swing or SWT and work with very lighter APIs, he or she should consider open source projects such as "Thinlet" or "LwVCL" [http://www.thinlet.com, http://zaval.org]: these projects provide the developer with packages whose dimensions are not greater than 150 KB, but this "lightness" is paid in terms of a low number of functionalities and widgets. Another framework, which is alternative to Swing in GUI's developement for Java applications, is SWT [http://www.eclipse.org]. The goal of this project is to allow developers to build user interfaces, which look and behave exactly as the user's OS native interfaces do. The main drawback of SWT libraries is that they are not currently supported by all operating systems (like MacOS X); moreover, they miss some of the features and widgets implemented in Swing.
A very interesting project seems to be Luxor-XUL: differently from the previously described ones, this project offers libraries which are not alternative to Swing. Luxor's packages comprise classes for describing Swing graphical elements by means of XML; this makes Luxor particularly easy to be used by Java developers. Using Luxor, the amount of code used for GUIs' developement is drastically reduced and in the same time the updating of existing interfaces is simplifies, because in order to modify the look of a GUI one simply has to modify the text document describing the GUI itself. For our pourposes we chose Luxor-XUL among all the open source projects previously described (see below). This choice has been mainly suggested from the necessity to work with previously developed Java code using Swing libraries. Luxor's high flexibility and its easiness of use further convinced us.

2.1 The XUL language

XUL has been developed inside Mozilla open-source project [www.mozilla.org]; designer's goal was to obtain a new tool to build Web applications GUIs, which could interact with web widely used rendering languages, such as HTML, CSS etc. XUL was developed to build graphical applications rather than format documents and build web pages; hence it is complementary towards rendering formalisms such as HTML or DHTML. Using a runtime engine called Gecko, developers can implement multi-platform GUIs, that is, GUIs whose rendering is device independent.

XUL is based on the concept that graphical interfaces are entities, which are autonomous from applications, and proposes itself as a tool that can advantage both developers and users.

For example, using XUL for developing Netscape 7's and Mozilla's graphical interfaces makes it possible that these browsers present the same look and feel under different operating systems and can be easily customized by users. If a user wants to change the look and feel of his web browser, he simply has to download one of the skins available on line, without modifying the rendering engine's code. A skin (a.k.a. theme) is an element of a graphical user interface that can be changed to alter the look of the interface without affecting its functionality. Each skin is compounded of XUL files, XML files and images that need to be installed in a specific directory to work properly. A graphical user interface can be given a high level view as a set of elements hierarchically organized and represented by means of XML formalism.

2.2 Luxor-XUL

Luxor-XUL (luxor-xul.sourceforge.net) is an open source project whose goal is to integrate the features of Mozilla's XUL with Java's portability, in order to obtain cross platforms GUIs. Using Luxor, developers

can describe Swing graphical elements using tags and attributes; a Java-Swing GUI can thus be completely described in terms of XUL and XML files: XUL files are used to describe graphical elements, while XML files are used to initialize particular objects, such as trees, lists and tables. In a XUL document designers describe the GUI from a "hierarchical" point of view; hierarchical relationships between graphical elements are well represented by the document's tree structure. Luxor needs a similar structure to be built and permanently stored in memory; to attain this, the JDOM package is used (JDOM is an open-source API for creating, parsing and modifying XML documents). In Figure 1 is depicted how Luxor interacts with a Java application: once the XUL description of the interface is complete, Luxor classes need to be instantiated in order to build the GUI and button actions need to be implemented.

Unfortunately, in current release, Luxor-XUL lacks some of Mozilla's XUL potential concerning GUIs' design; in Mozilla's XUL there is a wide set of tags and attributes for describing graphical components and their layouts, while in Luxor this set is quite smaller.

The main problems we encountered during our testing concerned layout management: Luxor's current release (Beta 8) doesn't supply tags and attributes to deal with complex layouts (such as Swing gridBagLayout), nor does it allow to manage styles properly, even if a style attribute exists. Other problems arose from the fact that there were no attributes to describe certain graphical elements, so we needed to modify some of Luxor's classes in order to introduce new tags.

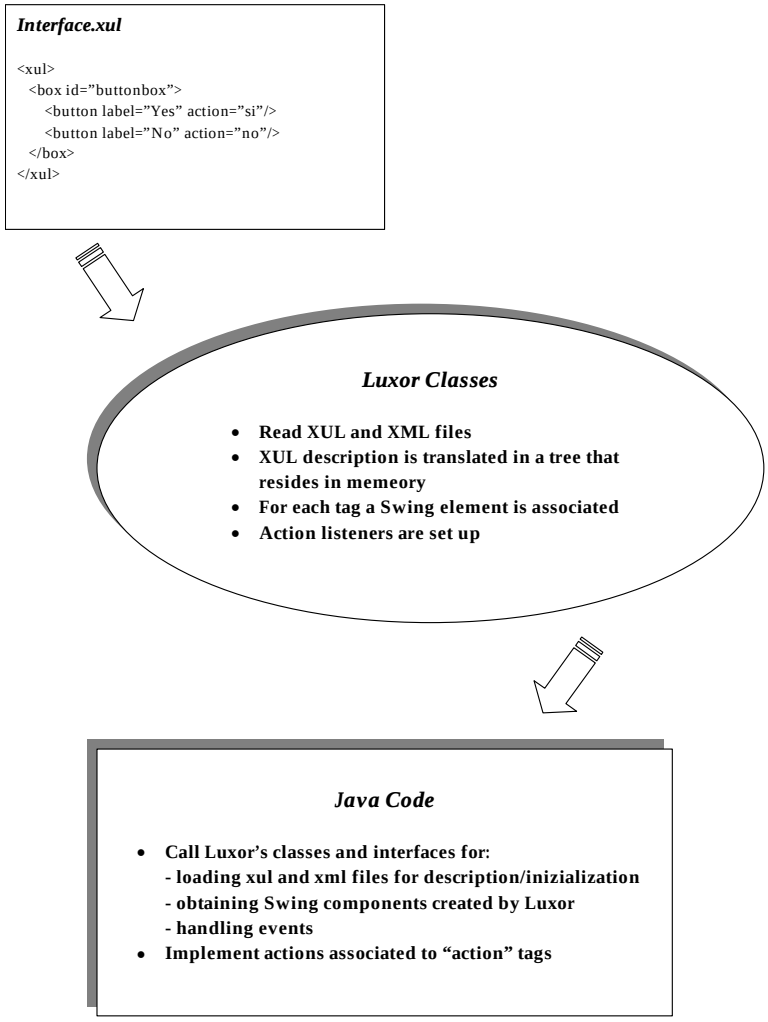


Figure 1 Luxor XUL

3. CASE STUDY: GUIS IN SVS PAWIS

The “SVS information system” is a Public Access Web-based Information System (PAWIS), which runs at “Servizio Sviluppo Sostenibile”, a division of Italian Environment Ministry located in Rome. Its aim is to promote and coordinate nation-wide public programs and initiatives on environmental awareness, education and training. Since almost all departments of “Servizio Sviluppo Sostenibile” are involved in the WIS, the computer system component (built using Java technology) must provide several different tools to cover administrative, research, management and, above all, citizen’s information needs. The HCI subsystem architecture has been designed keeping in mind the following guidelines:

- In order to let user perceive SVS system, distributed over the Internet, as a unified whole, the “desktop metaphor” [2] has been adopted.
- Building computer systems supporting complex Information Systems requires designers to take care of life cycle of implemented processes. When an automated process is not valid anymore, for any reason, the computer system must be changed as consequence. Therefore computer system should be designed as modular as possile to allow automated processes (including UIs) to be changed according to evolution of organization’s needs. To deal with this issue an API of “atomic” graphic and non-graphic components for UIs has been implemented, user interfaces (mainly data entry UI) are implemented combining these components.

In the implementation phase we’ve tested tools for automatic GUI generation; while results are encouraging for simple UIs there are many open issues (see below) for more complex GUI. In current release of SVS system few UIs are obtained through a declarative description of components parsed at runtime to build the interface, in next paragraph a use case is presented.

3.1 Use case: automatic generation of a UI in SVS

In SVS WIS there are many data entry UI. Every data entry interface allows users (with right permissions) to read, insert or change information, sending queries or update requests to the data manager. As use case we show in this paragraph as Luxor XUL framework has been adopted to build “Find” interfaces. In this interface a main panel contains two frames disposed horizontally: in the upper frame a tabbed panel allows to enter search criteria (filters), in the bottom frame, a table is placed in order to show the search results (the answer of the query). The Find UI is used for searching all categories of information in SVS. For each different category the number of tabs in the upper panel and their content vary, as well as the structure of the result set shown in the bottom frame. Adding or removing graphical objects from the panels thus obtains different layouts. Luxor XUL has been used to combine graphical elements in such a way to create GUIs on the fly for the different entities.

As we said above, the set of Luxor’s tags can be enlarged quite easily. This is a very useful skill when we are dealing with a large number of GUIs sharing some features, as it leads to a strong reduction of code. The first step has been to create SVS’s built-purpouse tags. Figure 2 shows what we mean: the panel for data search and visualization is associated to tags <infeasearch>, <filterpanel> and <resultpanel>.

Element <infeasearch> is used to build a orizontal splitpanel, while <filterpanel> and <resultpanel>, by means of the source attribute, allow to load the frames into the splitpanel (source refers to the id given for the frames in XUL description). Implementing different Find UIs becomes at this point very simple: for each SVS entity, the file standardpanel.xul is loaded at run-time, with appropriate values for the source attributes. We will opportunely create directories in which XUL files for the description of all the “filterpanels” and “resultpanels” will be stored.

Each time, the correct descriptions of the upper and bottom frames must be loaded: particularly concerning the table for result presentation, it is necessary to set up a mechanism for fetching the correct data (i.e. the answer of the data base) and the correct header. Actually, in order to organize data in a table, Luxor-XUL provides for a tag, which refers to a definite XML file to build a Swing JTable element.

```
//tab_enti.xml
<?xml version="1.0"?>
<TAB_ENTI>
```

```
      <ENTE>
            <ID>EN001</ID>
            <DENOMINAZIONE>Ente name</DENOMINAZIONE>
            <TIPOLOGIA>Type name</TIPOLOGIA>
            <SEDE>Located</SEDE>
      </ENTE>
      …….....
                  // Following: similar descriptions for the other table
rowz
      ..........
  </TAB_ENTI>
```

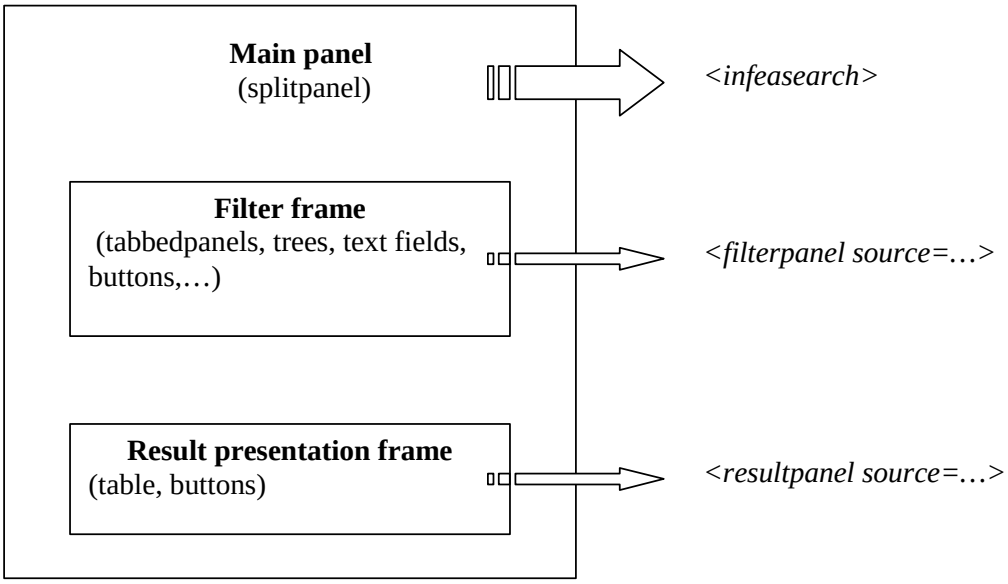


Figure 1

Let's assume we want to implement a Find GUI for the category "Enti" (Agencies). Figure 4 schematizes the interface's generation process: the figure shows the file standardpanel.xul in which the interface is described, XUL files for describing upper and bottom panels, and the GUIs they produce, which compose the final result.

Run time construction of the JTable for result presentation is obtained by means of the tags <colgroup> and <col> (inside result_enti.xul) and the attributes select and heading, which define column headings; data to be shown are indicated by the following fragment of code:

```
<datagrid datasource="data/tab_enti.xml" select="/TAB_ENTI/ENTE">.
```

The *datasource* attribute refers to the file tab_enti.xml in the "data" subdirectory, while select="/TAB_ENTI/ENTE" indicates the node to consider inside the document for taking data to be shown inside the table columns.

To realize how simple it is to modify structure and layout of a GUI, consider Figure 3: this figure shows a search panel for the category "Enti" in which only three tabs are needed and checkboxes and radiobuttons are used instead of lists.

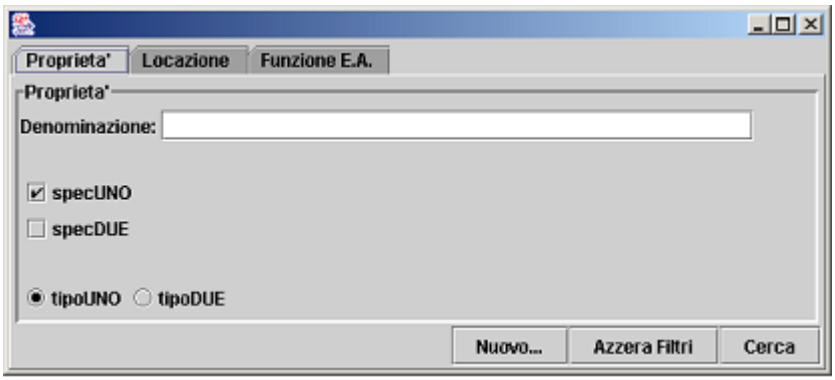


Figure 2

To attain this result we modified the description file filter_enti.xul: we eliminated the lines of code describing the tabs we didn't want to show any more and then we modified some lines in the description of the first "tabpanel". We wanted to change the way to represent the user's choices: the following schema shows the changes we had to do in filter_enti.xul in order to obtain this:

Description	Code

Choices described by means of combo boxes	`<tabpanel id="proptab">`
	`<label value="Tipologia:"/>`
	`<choice id="COMBO1"map="SCELTA_PROP1"/>`
	`<label value="Specifica:"/>`
	`<choice id="COMBO2" map="SCELTA_PROP2"/>`
	`</tabpanel>`

Choices described by means of checkbox and radio buttons	`<tabpanel id="proptab">`
	`<checkbox id="CK1" label="specUNO"`
	`checked="true"/>`
	`<checkbox id="CK2" label="specDUE"/>`
	`<choice list=" SCELTA_PROP2" type="radio"/>`
	`</tabpanel>`

3.2 Advantages and open issues

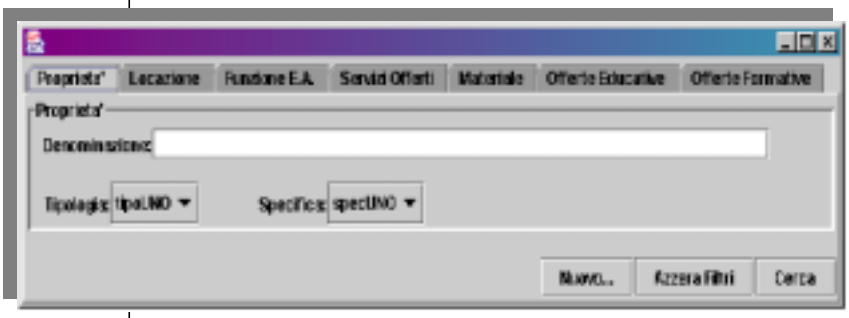
As we repeatedly stated, the use of a language such as XUL is very helpful in developing user interfaces, as it allows writing a considerably lower amount of code, thus saving time. The XUL language developed by Mozilla team is quite powerful, but it works only in a Mozilla/Netscape context. Luxor XUL open source project attempts to extend the benefits of XUL to other applications, in particular to applications based on Java technologies. Unfortunately, Luxor's goals aren't fully achieved: Mozilla's XUL layout capabilities haven't been achieved yet, and the number and functionalities of the available widgets are not comparable (at least in Luxor's latest release) with Java-Swing. Furthermore, neither Mozilla's nor Luxor's XUL seem to be suitable to be used in a multi-client context, where the same interfaces need to be displayed on many different devices. During our experience with Luxor XUL, we appreciated the language's simplicity of use and the fact that there was no need to modify Java code when we wanted to modify the UIs. The lower amount of code required should improve the whole IS's performances; anyway we tested XUL on a very restricted number of GUIs, so this issue will be better analyzed in future works. According to Luxor's designers, the lack of tags and attributes to properly manage complex styles and layouts should be solved in Luxor's future releases. Wider sets of widgets should also be provided.
One step further would be the development of a graphical framework, provided with a "widget repository". The repository should contain not only the widgets associated to standard tags, but also the user defined ones. In a framework like that, developers should be able to build UIs interactively without writing code (widgets could be added by means of drag and drop) and look at intermediate results.

```
// standardpanel.xul

<xul>
        <infeasearch>
            <filterpanel source="filter_enti.xul"/>
            <resultpanel source="result_enti.xul"/>
        </infeasearch>
</xul>
```

```
// filter_enti.xul

<xul>
<map id="SCELTA_PROP1">
          <entry key="UNO" value="tipoUNO"/>
         <entry key="DUE" value="tipoDUE"/>
</map>
<map id="SCELTA_PROP2">
          <entry key="UNO2" value="specUNO"/>
         <e  ntry key="DUE2" value="specDUE"/>
</map>
<tabbox>
<tabs>
<tab label="Proprieta'"/>
<tab label="Locazione"/>
<tab label="Funzione E.A."/>
<tab label="Servizi Offerti"/>
<tab label="Materiale"/>
<tab label="Offerte Educative"/>
<tab label="Offerte Formative"    />
</tabs>
<tabpanels>
<tabpanel id="proptab">
....//segue descrizione del pannello
</tabpanel>
<tabpanel id="loctab">
…..//segue descrizione del pannello
</tabpanel>
<tabpanel id="funtab">
…
…
</tabpanels>
</tabbox>
</xul>
```




```
//result_enti.xul

<xul>
   <vbox id="RISRIC">
      <groupbox>
      <caption label="Risultato ricerca:"/>
          <datagrid datasource="data/tab_enti.xml" select="/TAB_ENTI/ENTE">

               <colgroup>
                   <col select="ID" heading="Id"/>
                    <col select="DENOMINAZIONE" heading="Denominazione"/>
                    <col select="TIPOLOGIA" heading="Tipologia"/>
                    <col select="SEDE" heading="Sede"/>
                </colgroup>
          </datagrid>
      </groupbox>
            <hbox>
               <separator/>
               <button labe  l="Elimina" />
               <spacer/>
               <button label="Mostra" />
               <spacer/>
               <button label="Apri" />
            </hbox>
   </vbox>
</xul>
```

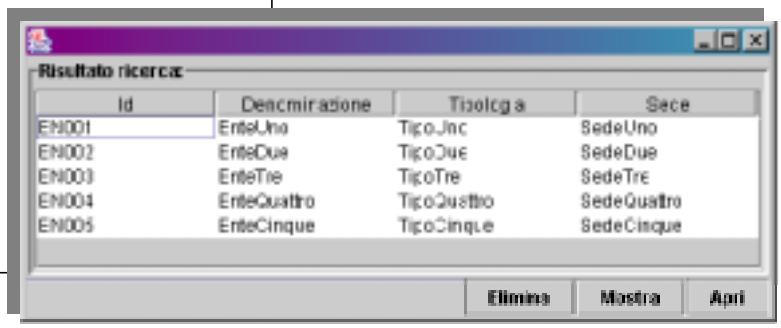


Figure 4

4. CONCLUSION

This paper discuss issues in automatic GUI generation for Web-based Information Systems, paying attention to the set of interfaces allowing users to interact with data management system, a.k.a. data entry UIs. After a survey on main open source projects, an interesting framework (Luxor XUL) is described highlighting its main features and drawbacks. This framework has been used in implementing the Public Access WIS “SVS System”. The second part of the paper is devoted to present the HCI subsystem of this PAWIS as case study, detailing the part regarding automatic GUI generation. Even if a little set of GUIs are built using this framework, and many issues remains opened, we believe that results are encouraging and we’re planning to enlarge set of automated GUIs for next system release.

REFERENCES

[1] Baeker, R. M., Grudin, J., Buxton, W. A. S., Greenberg, S. 1995, *Readings in Human-Computer Interaction: Toward the Year 2000*, Second Edition, Morgan Kaufmann Publishers.

[2] Preece, J., Rogers, Y., SHARP, H., Benyon, D., Holland, S., Carey, T., 1994, *Human-Computer Interaction*, Addison Wesley

[3] [2] N. Aloia, C. Concordia, F. Furfari, V. Miori 2001: *Caratteristiche, problematiche e tecnologia dei sistemi informativi basati sul Web*, Rivista di Informatica AICA, vol. XXXI n.2 settembre 2001 (pp. 95-108)

[4] N.Aloia, T.Bendini, F.Paternò, C.Santoro 1998: *Design of Multimedia Semantic Presentation Templates: Options, Problems and Criteria of Use*, Proc. of AVI '98 International Workshop on Advanced Visual Interfaces, ACM Press (pp.205-215), L’Aquila Italy, maggio 1998

[5] G. Abowd, J. P. Bowen, A. Dix, M. Harrison, R. Took 1989: *User Interface Languages: A Survey of Existing Methods* Oxford University Computing Laboratory (1989)