Advice-Giving Systems for Selecting Interaction Objects

Jean Vanderdonckt
Université catholique de Louvain, Institut d'Administration et de Gestion
Place des Doyens, 1, B-1348 Louvain-la-Neuve (Belgium)
E-mail: vanderdonckt@qant.ucl.ac.be

Abstract

In the development of interactive systems, the selection of interaction objects refers to the process of identifying usable widgets for each piece of data forming the system according to various parameters. This process can become repetitive and tedious for data intensive systems where data is abundant. To overcome this shortcoming and to provide assistance to designers who are responsible for conducting this process, two types of advice-giving systems for selecting interaction objects have been developed. SELECTVISION *consists of a system where parameters are input manually and the possible selected interaction objects are displayed interactively.* SEGUIA *consists in a Knowledge-Based System where the selection process can be conducted either automatically after providing a file of parameters or in a mixed-initiative approach where both the system and the designer are interacting to conduct the process. Both systems share a common internal representation of usability guidelines for selecting interaction objects expressed as a selection tree of rules. This structure has been shown to be very observable and easy to process, while preserving some maintainability.*

1. Introduction

In the development of interactive systems, the selection of interaction objects (IOs) refers to the process of identifying the most usable widget, such as an edit box, a list box, or a drop-down box, for each data item specified in the data model for input/display in the future application [1]. This selection may involve three parameter types:

1. Meta-data expressing some semantics of the data item such as data type (e.g., real), domain definition (e.g., unknown domain, known domain, or a mixed domain having both values together), choice type (simple choice or multiple choice), number of possible values (Npv), orientation (e.g., vertical or horizontal).
2. User parameters, such as user experience level, user preferences (e.g., for selection or for input).
3. Physical environment parameters, such as screen constraints, data density (e.g., low or high).

Since these parameters can be numerous and complex to manage to select any usable IO for any data item, it seems sound to organize the selection process into a practical format that is easy to manipulate and to browse. For this purpose, a set of selection guidelines have been organized into `IF...THEN...` *production rules* assembled in a production system [2,8]. The possible contexts of the selection process are described in the antecedent as a conjunction of simple conditions expressed on *n* parameters whose satisfaction fires the execution of the subsequent:

$$\bigcup_{i=1}^{n} parameter_i = value_i \Rightarrow Selected\ IO = usable\ IO$$

This execution may in turn generate a new context allowing another rule to be fired until all candidate rules are exhausted. For example, rule 2 in fig. 1 selects radio buttons to input/display one possible value between the three known values of a real data item in a low density environment.

```
IF Data type=real AND Interaction=input/display AND
Choice type=simple AND Domain definition=unknown,
THEN Selected IO=profiled unilinear edit box        (Rule 1)
IF Data type=real AND Interaction=input/display AND
Choice type=simple AND Domain definition=known AND
Nvp ∈ [2,3] AND Density=low,
THEN Selected IO=radio button                       (Rule 2)
IF Data type=real AND Interaction=input/display AND
Choice type=simple AND Domain definition=known AND
Nvp ∈ [2,3] AND Density=low,
THEN Selected IO=radio button                       (Rule 3)
...
```

Figure 1. Some excerpts of the production system.

It is important to note here that the selection guidelines are not design rules. Design rules generally express in a straightforward format a physical selection which is not necessarily based on some usability experimental results. Design rules are typically found in corporate style guides. Conversely, the selection guidelines are here primarily based on theoretical assumptions combined with empirical results [2,4]. Selection guidelines have been established for nine data types: hour, date, Boolean, graphical, integer, real, alphanumeric, sound, and video.

As selection guidelines are based on such results, it is fundamental that their expression does not depend on a particular target computing platform where the future interactive system will be executed. To ensure some independence of selection rules with respect to physical considerations, their consequent will select an *Abstract Interaction Object* (AIO) rather than a *Concrete Interaction Object* (CIO) [8]. For this purpose, a taxonomy of AIOs has been defined with abstract attributes, events, and primitives that allow instantiation in a particular computing platform, once this is chosen.

2. Internal representation of production rules

Conducting a selection process on a production system does not assume a particular inspection ordering since the control flow is not determined by the ordering production rules are coded. They are fired as soon as the current context matches their antecedent. Although production rules are easy to process by an automata and are not redundant, their linear formal expression makes them somewhat hard to maintain by a designer. Moreover, they cannot be observed visually. To overcome these shortcomings, another internal representation is needed.

A *bulleted list* can be examined by writing each non terminal condition of the rule antecedent in an indented way and by reserving the last indented line for each terminal condition related to the rule consequent. A similar bullet is consistently assigned to each line. For example, fig. 2 represents a bulleted list equivalent to fig. 1. The bullet symbolic makes the diagram observable and easy to maintain (modifying a simple condition can be operated one line at a time). However, the browsing flow is basically sequential and the diagram size dramatically expands as the number of such simple conditions increases.

1) Choice type = simple
   a) domain definition = unknown: profiled unlinear edit box
   b) domaine definition = known
      - $2 \leq Npv \leq 3$
        - Density = low: radio buttons
        - Density = high: option box
      - $4 \leq Npv \leq N_{mag}$
        - Density = low: radio buttons in a group box
        - Density = high: option box
      - $N_{mag} < Npv \leq T_m$
        - Density = low: selection list
        - Density = high: drop-down selection list
      - $Npv > T_m$
        - Density = low: scrolling selection list
        - Density = high: drop-down scrolling selection list

Figure 2. A bulleted list representing production rules.

In fig. 2, N_{mag} represents a cut-off constant deciding where partial view of values is decided. It is currently assigned to the Magical Number as $N_{mag} \equiv 7$ [2]. T_m represents a cut-off constant deciding when fast scrolling is need to manage an important amount of values. It is currently assigned to the Tullis' constant as $T_m \equiv 50$ [7].

A *normal flowchart* can be drawn with one lozenge having two truth values (yes/no) for each condition in the rule antecedent and with one rectangle specifying which AIO to select as the rule consequent. Although the condition chaining is straightforward to understand, to browse and to maintain thanks to binary choices, this normal flowchart remains visually loaded: the number of lozenges is equal to the number of non terminal conditions minus 2 and the number of rectangles is equal to the total number of selectable AIOs. A *reduced flowchart* can be obtained from the normal flowchart by branching to common subparts and by rearranging, grouping conditions to minimize the size. Therefore, the more compact the flowchart becomes, the more manageable it becomes, but the less modifiable it also becomes. Verifying completeness and consistency remains a hard task to carry out.

A *decision table* can graphically express production rules into four quadrants as follows [3,5]:

1. The upper left quadrant states all possible conditions.
2. The upper right quadrant contain entry points depicted by ■ to denote any condition combination.
3. The lower left quadrant enumerates the possible AIOs to be selected for the current context.
4. The lower right quadrant provides flags depicted by ■ to denote the selected AIOs in the column.

Although the condition ordering in the upper part makes the diagram very readable in any random path, conditions do not overlap and preserve their non-redundancy. Such a decision table is quite observable and maintainable. However, the lower right quadrant is a very sparse matrix mainly due to the single selected AIO per column. This sparsity can be reduced by factorizing common AIO attributes such as drop-down, scrolling, multiple choice, editable attributes (fig. 3). Alternate solutions are supported by numbering: for instance in fig. 3, **(1)** depicts a selection list with multiple choice, whereas **(2)** depicts a Boolean selection list. **(1/2)** is specifying that both AIOs can be selected equally. This reduction does not entail the modifiability of any line or column in the quadrants nor does it decrease the visual capacity to show to designers a visual portrait of rules. However, such a reduced table forces the algorithm to process all selectable AIOs to reach a final solution so that no supplemental characteristic will be missed. This complexity is particularly demanding when the amount of selection guidelines is huge (363 in our case) or when the selectable AIOs with various attributes are numerous (up to 50 in our case).

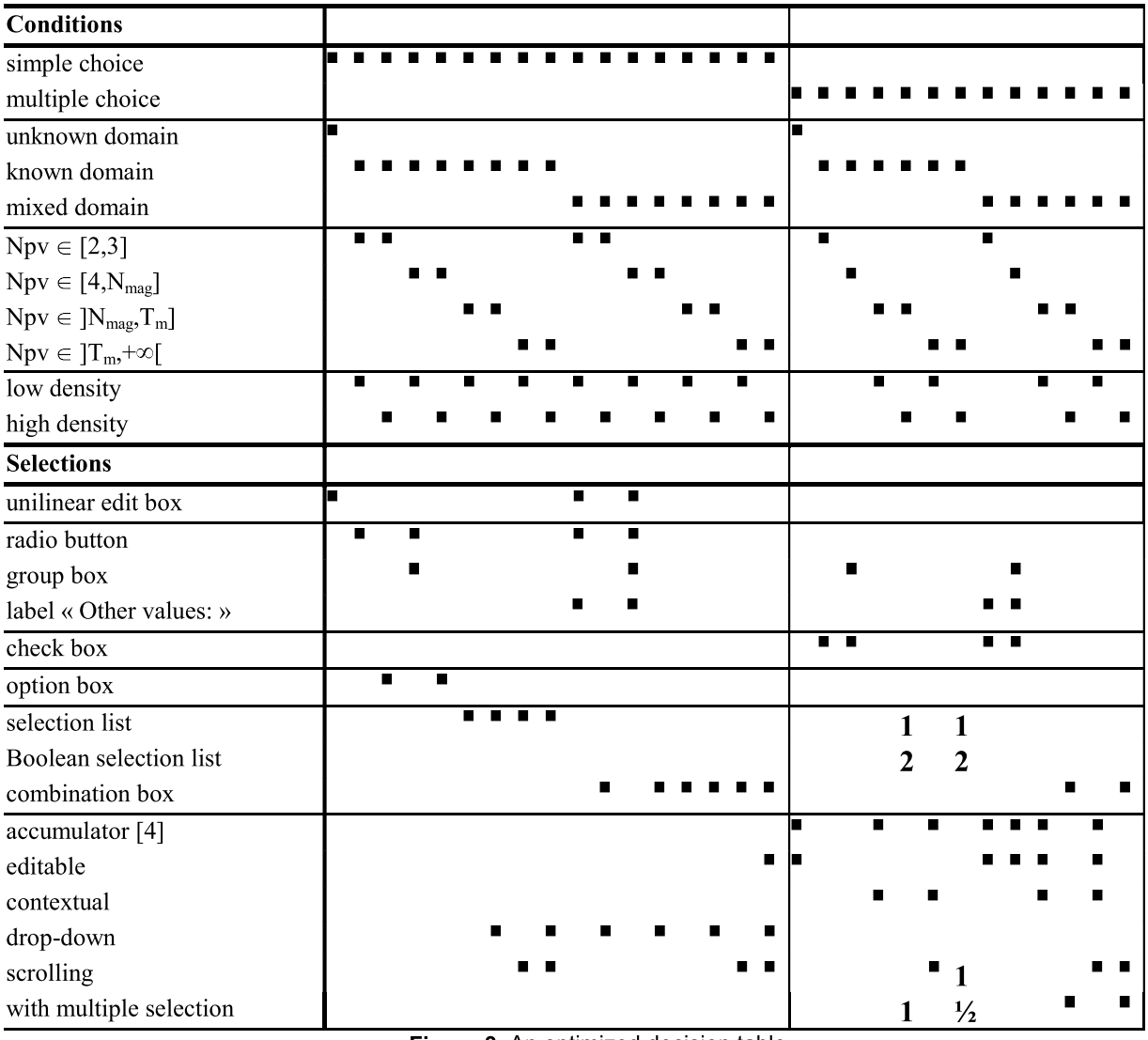

Figure 3. An optimized decision table.

A *decision tree* consists of a graph beginning with a node called the *root* whose descending nodes only possess one single preceding node [5,6]. We refer to a *selection tree* as a decision tree where two types of nodes are found:

1. *Branching nodes* evaluate a simple condition on a single parameter to determine a subsequent path to follow.
2. *Conclusion nodes* are leaf nodes selecting a supposed-to-be usable AIO or a combination of AIOs according to the evaluated conditions.

Any selection tree can be drawn horizontal, vertical or oblique. Differences between line types linking condition, while they are in rake (⊣), in a pencil of rays (⋖) or in line (|) are not significant [6]. Conversely, the orientation of selection trees positively influences spatiality: oblique selection trees spread the amount of conditions along two dimensions, thus minimizing the size (fig. 4). Decision trees are easily understood by designers [6]. They can dynamically follow the selection logic. Therefore, experts as well as non-experts are able to actively participate in defining and using rule chaining. Selection trees allow visual programming so that it is no longer necessary to master programming languages to express selection rules. Therefore, selection trees are kept as probably the most appropriate internal format for manipulating selection guidelines in an observable way, while preserving maintainability.

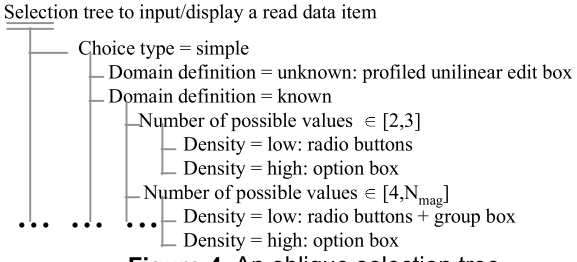

Figure 4. An oblique selection tree.

3. SELECTVISION, a manual system

SELECTVISION is a manual advice-giving system manipulating selection trees. The current values of parameters are manually input in adequate fields in the control panel reproduced in fig. 5. The recommended AIO is then automatically selected and appears in the bottom-right square. Once a particular AIO has been selected, by double-clicking on the field, the designer can graphically visualize why this particular AIO has been selected, thus providing some explanation of the reasoning behind the selection of the AIO.

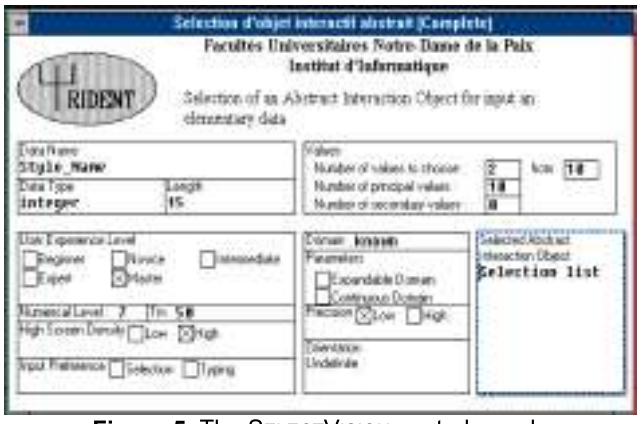

Figure 5. The SELECTVISION control panel.

The path from the root of the global selection tree to the leaf node holding a selected AIO is highlighted in blue and can be followed on demand in both full and partial views. The designer is guided throughout the selection tree as the selected AIO is surrounded in the selection tree (fig. 6). This advice-giving system is manual since it only displays the name of selected AIOs. On one hand, this information can be useful for any computing platform. On the other hand, this information is not connected to any user interface development environment, thus losing the selection results. Anyway, this manual system is useful for learning how to select usable AIOs and how the selected AIO can be influenced by playing with the current parameter values. For example, if an integer with simple choice among 9 possible values must be input in a continuous domain with low precision, then:

- If orientation is vertical, the AIO can be a scroll bar.
- If orientation is horizontal, the AIO can be a scale.
- If orientation is circular, the AIO can be a pie diagram.
- If orientation is undefined, the AIO can be a scroll bar.

SELECTVISION supports such an exploration as any value modification is automatically propagated in the decision tree, thus leading to a possibly different leaf node. This mechanism seems particularly appreciated by designers who want to understand the precise conditions leading to a particular AIO. However, the loss of information captured in the software decreases its attractiveness. In the case of data intensive systems, it is obvious that repeating this manual process for each data item can be interpreted as a cumbersome approach. This regret mainly motivates the need for a system that automates most of this process and that allows the direct contribution of the selection process into an exploitable user interface development environment. Designers seem much more encouraged to specify the parameter values of each data item when they know that the selection process will produce results that are straightforwardly to reuse.

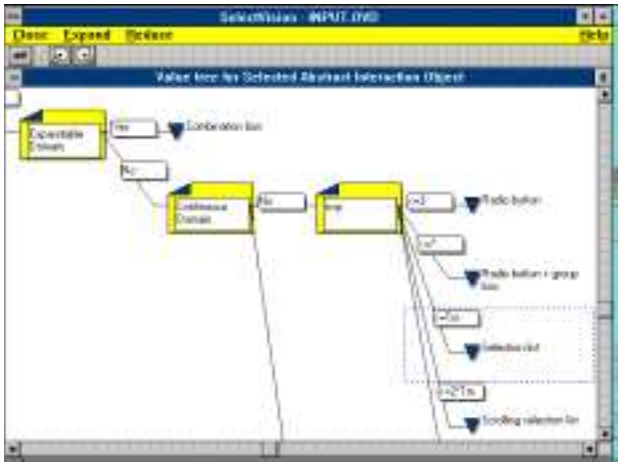

Figure 6. A partial view of the selection tree.

4. SEGUIA, an automated or computer-aided system

SEGUIA ("Système Expert pour générer la 'user interface' automatiquement") consists in a knowledge based system (KBS) implemented on top of an expert system shell (AION/DS for Windows, from Trinzic Software) aimed at generating a user interface presentation according to a model-based approach. In this technique, a connection is established between two models:

1. A *source model*: an information model containing a hierarchical structure of data items manipulated in the interactive system. Each data item is specified throughout the values of the parameters that have been used before. For this purpose, an object-oriented model of the seven data types supported has been created. Each data item therefore consists in an instance of this object-oriented model (fig. 7).
2. A *destination model*: a presentation model containing a hierarchical description of AIOs that will form the presentation of a future user interface. For this purpose, a second object-oriented model of possible AIOs has been created with a set of methods holding their abstract primitives and a set of slots holding their abstract attributes (fig. 8).

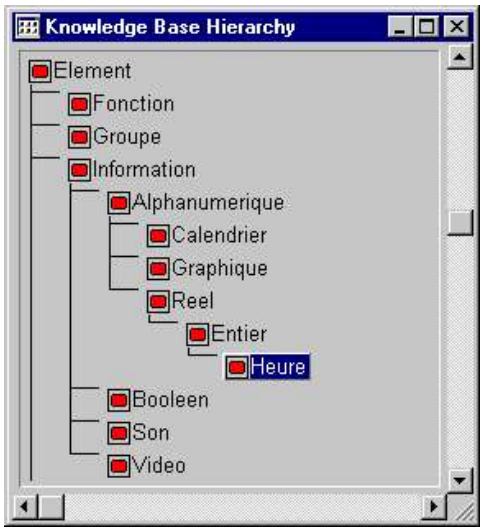

Figure 7. Supported data types in the information model.

For instance, fig. 8 reproduces the definition of the combination box as an AIO.

Figure 8. An AIO definition in the presentation model.

High level abstract primitives, such as creating, deleting, opening and closing a combination box are specified in the methods definition. High level abstract attributes, such as dimensions, locations and selection types are specified in the slots definition. Only high level abstract primitives and attributes are specified in this model. Other low level primitives, such as scrolling, highlighting, selecting a value, and other low level attributes, such as colors, texture, border type, have not been retained here : although they are required when defining a concrete presentation, they are not needed at the level of an abstract presentation. Moreover, some slots such as the location (X and Y slots in fig. 8) are not fed into the selection process. They will be added when the abstract presentation is mapped onto a concrete presentation.

The selection tree introduced in section 2 and implemented in SEGUIA (fig. 9) is responsible for establishing the connection between each data item in the information model to one or several AIOs in the presentation model according to the parameters captured in the information model. The basic idea of this mapping involves (1) examining the parameter values of each data item in the information model, (2) selecting one or several AIOs by browsing the selection tree according to the values, (3) creating AIO instances in the presentation model according to the results of the selection process. This procedure is repeated for each data item appearing in the hierarchy, thus forming a second hierarchy in the presentation model.

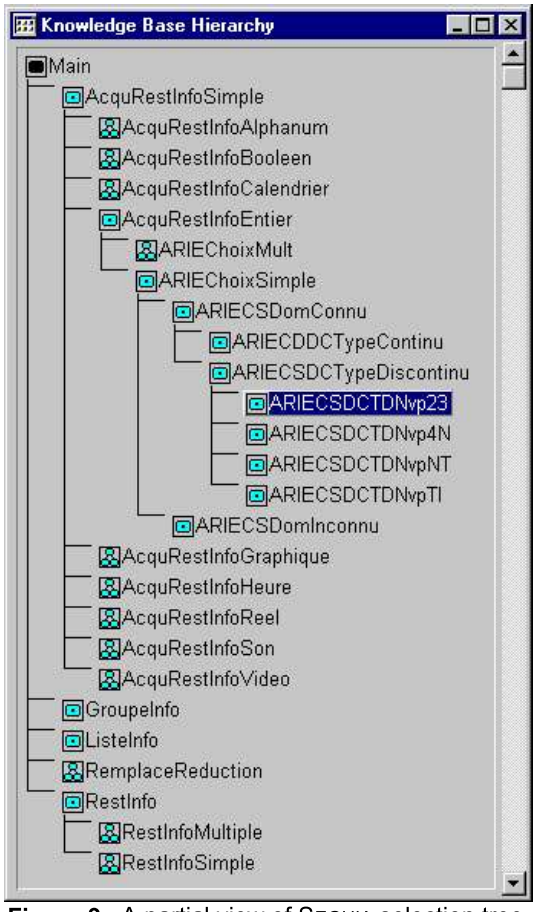

Figure 9. A partial view of SEGUIA selection tree.

Fig. 9 shows the selection tree expanded at a particular node: the one to select an AIO for the simple choice of an integer data item having a known domain of 2 or 3 possible values in a discrete range.

SEGUIA is built on top of an expert system shell that contains an inference engine, which is able to process the selection tree with built-in facilities with three chainings:

1. *Forward chaining*: the selection tree is browsed step by step forward from the root to leaf node according to the parameter values. This chaining is mainly used in the selection process to generate a presentation.
2. *Backward chaining*: the selection tree is browsed step by step from any node to the parent node, provided that the initial node is not the root. This chaining is mainly used when some parameter values should change. Designers found this feature especially useful as it enables them to stop the selection process at any step, to go back one or several steps in the selection tree as when no AIO alternatives are considered usable at the current node. This chaining can also be used when a designer wishes to understand all parameter values needed to select a particular AIO. If a same AIO can be selected at different places in the selection tree, these circumstances can be displayed.
3. *Bi-directional chaining*: the selection tree is browsed step by step in a combination of both forward and backward chaining, thus enabling designers to go back and forth to explore AIO alternatives in the selection tree.

A second important feature of SEGUIA is that the selection process can be conducted according two approaches:

1. *Automated generation of a presentation*: the values of **all** required parameters need to be provided in the system for each data item. The selection process is then launched and achieved without any other human mediation or interruption. A hierarchy of AIOs is finally generated with the forward chaining.
2. *Computer-aided design of a presentation*: only the **known** values of parameters are provided in the system for each data item. The selection process is then conducted interactively with the designer. The selection tree is browsed as far as possible. The system then prompts the designer to explore the alternatives offered at the current node. The designer is then supported in traversing the selection tree according to the bi-directional chaining described above. A hierarchy of AIOs is finally generated when a leaf node is reached for data item or when the designer decides to select a dedicated AIO.

The AIO hierarchy can be transformed into a CIO hierarchy once a target environment has been chosen. This transformation is applied by a set of mapping rules [8]. Currently, SEGUIA only supports the Microsoft Windows environment since the current rules support the transformation of AIO to CIO belonging to the Microsoft Visual Basic V5.0 environment.

5. Conclusion

Designers differ in the extent to which they appreciate automated generation or computer-aided design of a presentation. In the context of data intensive systems, they tend to start with the computer-aided design approach as they learn by interacting with the system and by controlling the selection process. Once they master the selection process, they are more likely to use the automated generation to avoid repeating the same process. Routine tasks, such as selecting AIOs for a large number of data items in a data intensive system, should be automated. Human control is still necessary to handle unanticipated situations. When such control is needed, SEGUIA suggests the bi-directional chaining with a visual presentation of the selection tree to see where the designer is located. Therefore, a balance is needed between automation that releases a designer from repetitive tasks and human control that provides them with responsibility and accomplishment. Although SEGUIA contains a cut and paste facility to allow reuse of previously used parameter values or even complete data specification, this support for reusability is felt to be poor. We are investigating other methods [9].

Acknowledgments
The author would like to warmly thank UIDIS'99 anonymous reviewers and Norman Paton for his support.

6. References

[1] D. de Baar, J.D. Foley, and K.E. Mullet, "Coupling Application Design and User Interface Design", *Proc. of CHI'92*, ACM Press, New York, 1992, pp. 259–266.

[2] F. Bodart and J. Vanderdonckt, "On the Problem of Selecting Interaction Objects", *Proc. of HCI'94*, Cambridge University Press, Cambridge, 1994, pp. 163–178.

[3] D. Gettys, "IF You Write Documentation, THEN Try a Decision Table", *IEEE Transactions of Professional Communication*, PC-29, Vol. 4, 1986, pp. 61–64.

[4] T.J. Johnsgard, S.R. Page, R.D. Wilson, and R.J. Zeno, "A Comparison of Graphical User Inteface Widgets for Various Tasks", *Proc. of HFS'95*, Human Factors & Ergonomics Society, Santa Monica, 1995, pp. 287–291.

[5] B. Moret, "Decision Trees and Diagrams", *ACM Computing Surveys*, Vol. 14, No. 4, 1982, pp. 593–623.

[6] G.H. Subramanian, J. Nosek, S.P. Raghunathan, and S.S. Konitkar, "A comparison of the Decision Tables and Tree", *Communications of the ACM*, Vol. 35, No. 1, January 1992, pp. 89–94.

[7] T.S. Tullis, "The Formatting of Alphanumeric Displays: A Review and Analysis", *Human Factors*, Vol. 25, 1983, pp. 657–682.

[8] J. Vanderdonckt and F. Bodart, "Encapsulating Knowledge for Intelligent Automatic Interaction Objects Selection", *Proc. of InterCHI'93*, ACM Press, New York, 1993, pp. 424–429.

[9] J. Vanderdonckt and P. Berquin, "Towards a Very Large Model-Based Approach for User Interface Development", *Proc. of UIDIS'99* (Edinburgh, 5-6 September 1999), IEEE Computing Society Press, Los Alamitos, 1999.