Dipartimento di Informatica
Università degli Studi di Bari

DE_VISU

Programma di ricerca (cofinanziato dal MURST, esercizio 2000)

Specifica, Progetto e Sviluppo di Sistemi Interattivi Visuali

Progettazione e Sviluppo User-Centred di Ambienti Visuali: il caso di VLDesk

RAPPORTO TECNICO: N.03 ANNO 2002

BA-R06 **30 Maggio 2002**

Sommario

VLDesk è un ambiente integrato che supporta il progettista di linguaggi visuali in tutte le fasi del processo di sviluppo di ambienti di programmazione visuali. VLDesk è un prototipo sviluppato dall'Università di Salerno. In questo rapporto si riportano i risultati della valutazione di usabilità di VLDesk. Inoltre, si rivede il processo di sviluppo Component_Based utilizzato da VLDesk, per integrare il ciclo di vita con metodi di usabilità, al fine di ottenere ambienti visuali usabili.

Titolo ricerca unità	L'usabilità nel progetto di sistemi interattivi
Codice	BA-R06
Data	30 Maggio 2002
Tipo di prodotto	Rapporto tecnico
Unità responsabile	BA
Unità coinvolte	BA - SA
Autori	Maria Francesca Costabile, Gennaro Costagliola, Rosa Lanzilotti, Grazia Minardi, Michele Risi.
Autore da contattare	Maria Francesca Costabile Dipartimento di Informatica Università degli Studi di Bari Via Orabona 4, 70125 Bari, Italia costabile@di.uniba.it

1. Introduction

It is common opinion that the effective use of a visual language requires a visual programming environment to support the given language by enabling the drawing and the processing of the visual language sentences [Bur95][Fer]. Visual programming environments for specific apllication domains are usaually implemented by using ad hoc techniques. On the other hand, by analyzing the charactestics of Visual Languages (VLs), it is possibile to determine several common aspects. In particular, VLs are syntactically described by graphical objects (visual symbols), and by the relations holding between the objects involved into visual sentences. The formalization of such characteristics allows the definition of development processes and tools appropriate for the implementation of visual languages belonging to different application domains.

VLDesk is a system for the automatic generation of visual programming environments. This system provides a “Desk” where a developer can find an integrated set of tools supporting the whole process of visual language development. VLDesk is composed by several components, these components are described in the section 3.

In this document we report the results of a usability evaluation of the component of VLDesk, called Symbol Editor.

The approach used to design and develop the VLDesk System is User-Centred methodology, in which the final users are involved from the very beginning of the planning stage. With this kind of methodology, a system is designed iterating a design-implementation-evaluation cycle [Cha01].

This report is organized as following. In the section 2 the usability evaluation is discussed. In the section 3, briefly describes the Visual Language Desk System (VLDesk) for the automatic generation of visual programming environments. In the section 4 describes the User-centred design. Finally, in the section 5 illustrates development methodology used by VLDesk and revise it with respect to [Cos02] in order to take usability into account.

2. Usability Inspection

The cost of the maintenance of big software systems is very expensive due to reasons related to the usability engineering, such as: frequent requests for changes by users, overlooked tasks, lack of understanding of users’ requirements, and insufficient user-analyst communication and understanding. A proper usability engineering methodology, applyed to the software life-cycle, will prevent most such problems and thus substantially reduce cost in the overall software life-cycle. In spite of the poor attention so far devoted to usability by software engineers, it is now widely acknowledged that usability is a crucial factor of the overall quality of interactive applications [Nie93].

In order to evaluate usability, various methods have been developed. The most significant are user-based methods and inspection methods. User-based methods mainly consist of user testing, in which usability properties are assessed by observing how the system, or the prototype of the system, is actually used by some representative of real users performing real tasks. Usability inspection methods involve expert

evaluators only, who inspect the user interface in order to find out possible usability problems, provide judgements based on their knowledge, and make recommendations for fixing the problems and improving the usability of the application.

Inspection methods are cost-effective since they only requires the work of the experts, while user testing require much more resources. Inspection methods include: heuristic evaluation, cognitive walkthrough, formal usability inspection, guidelines reviews [Cha01].

In our case, to evaluate VLDesk system we have used the heuristic evaluation.

In Appendix 1, we report, in Italian the report of our inspection, in which found problems and their possible solutions are described.

3. The Visual Language Desk System

Visual Language Desk (VLDesk) is a system for the automatic generation of visual programming environments, supporting all the phases of the Component-Based Visual Environment Development (CB-VED) for designing and implementing VLs by automatically generating visual programming environments. This system provides a "Desk" where a developer can find an integrated set of tools supporting the whole process of visual language development. The main components of VLDesk are: the *Symbol Editor*, the *Visual Production Editor*, the *Textual Grammar Editor* and the *Language Inspector* [Cos02].

The *Symbol Editor* supports the language developer in drawing the language visual symbols and allows him/her to associate the syntactic proprieties to each of them. The symbols of a given language are collected into a dictionary.

The *Visual Production editor* is a visual component supporting the language designer in the visual grammar specification, by drawing the grammar rules. Such a specification produces a YACC-like definition of the visual environment. Since it does not completely describe the syntax and semantics of the given language the designer can further refine it by the *Textual Grammar Editor*. The lexical analyser part for the symbol is automatically generated by selecting the symbol from the dictionary.

In the case of hierarchical visual language [Cos] VLDesk allows a designer to use the *Language Inspector* tool to easily navigate through the specifications of the component visual language. A hierarchical visual language is a language whose vocabulary contains symbols that can be annotated with textual or visual sentences from the same or another visual language .

Once the language specification has been provided, the tool automatically generates an integrated Visual Programming Environment (VPE), composed by a visual sentence editor and a compiler for processing them.

The VLDesk system maintains a repository of visual languages previously developed that can be reused.

4. User-Centred Design

One of the main requirements of the information technology society is to design for universal access. Designers must allow the human users to focus on the task at hand and not on the means for doing that task.

We need methods and technique to help designers change the way they view and design products, methods that work for the users' needs and abilities.

One of the approaches in this direction is user-centred design, which has already proven as a key factor for leading towards the development of successful interfaces. User-centred design implies that final users are involved from the very beginning of the planning stage, and identifying user requirements becomes a crucial phase.

Involving users from early stages allows basing the system core on what is effectively needed. Poor or inadequate requirement specifications can determine interaction difficultes, including lack of facilities and usability problems.

The main principles of user-centered design are:

1. analyse users and tasks;
2. design and implement the system iteratively through prototypes of increasive complexity;
3. evaluate design choices and prototypes with end users.

User-centred approach requires understanding reality: who will use the system, where, how and to do what. In this way it is possible to avoid serious mistakes and to save re-implementation time, since the first design is based on empirical knowledge of user behavior, needs and expectations. Collecting user information is not an easy task, even discussing with users, since users often neglect aspects that they erroneously consider not important.

Many different techniques can be applied for collecting user information, among them direct observation, interviews and questionnaires. Direct observation means observing the users while they carry out their tasks at their workplace. It is the most reliable and precise method for collecting data about users, especially valuable for identifying user classes and related tasks.

Interviews collect self-reported experience, opinion and behavioral motivations. They are essential to finding out procedural knowledge as well as problems with currently used tools. Interviws cost a bit less than direct observation, because they can be shorter and easier to code.

Finally, questionnaires can be handed out and collected by untrained personnel allowing to gather from various users a huge quantity of data at low cost. Questionnaires provide an overview on the current situation as well as specific answers.

At the end of the design cycle, summative evaluations are run. They test of the final system with actual users performing real tasks in their working environment. A summative evaluation should be considered as the very last confirmation of the correctness of the hypotheses stated during the design process.

5. Modifying the VLDesk Methodology

In the Figure 1 is showed the modified methodology to use during the system development. The first step consists to collection the user requirements with the techniques described above, during a meeting with user that will use the system. The analyst and the user work together in a kind of participatory design [May99] in

order to establish the characteristichs of the visual environemnt. They identify the visual languages involved in the final environment and their interrelationships. For each language the analyst sketches the graphical aspect of the symbols and of the visual sentences. In this phase the analyst makes use the VLDesk to produce a first visual environment mock-ups prototype [Som96]. This prototype facilities the communication between the user and the analyst to discuss about the syntax and the semantics of the single languages. After, this first prototype is evaluated with other users, through the inspection usability to discover if the user requirements are meet.

If the prototype doesn't comply with the users needs and expectations it is necessary to back in the prototyping phase and a new prototype is designed and again evaluated. This iterative process ends up when user requirements are meet.

While, if the prototype are comply, the developer details the requirements in the Visual Environment Specification document, where it specifies the user requirements and the relationships between the different visual languages of the visual environment.

So a new prototype is made according to the Visual Environment Specification document. In the following, a new prototype evaluation is made. As we described above, after that the evaluation is acceptable the developer can produce the final Visual Environment.

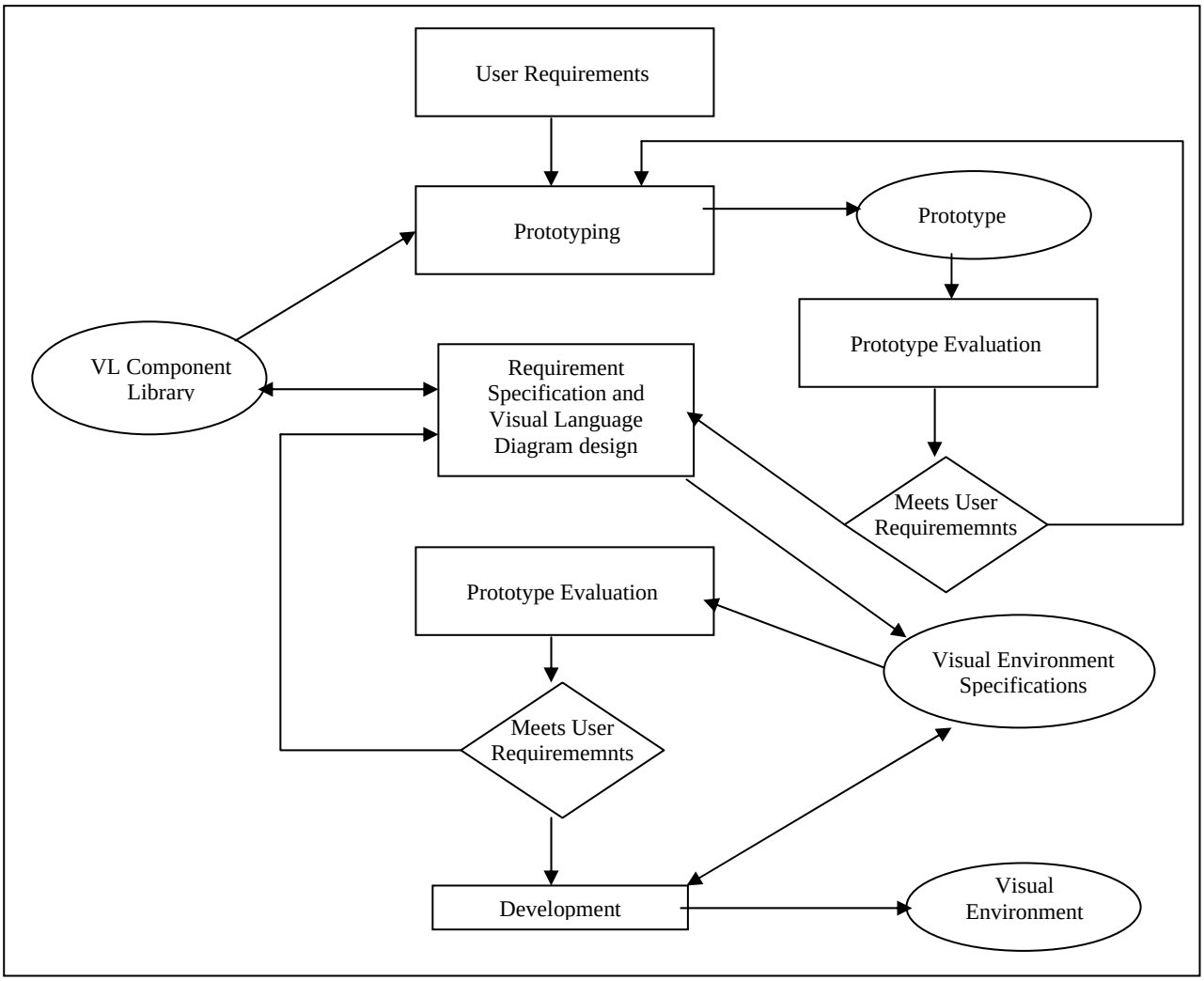

Figura 1 The Component Based Visual Environement Development Process

References

[Bur95]	M. Burnet, A. Goldberg and T. G. Lewis, “Visual Object-Oriented Programming”, Manning Publications Co., 1995.
[Cha01]	S.K. Chang, “Software Engineering & Knowledge Engineering”, Vol. 1 Fundamentals, World Scientific, 2001, pp. 179-192.
[Cos]	G. Costagliola, A. De Lucia, S. Orefice and G. Polese, “A Classification Framework for the Design of Visual Languages”, to be published in the Journal of Visual Languages and Computing, in print.
[Cos02]	G. Costagliola, A. De Lucia, R. Francese, M. Risi and G. Scanniello, “A Component-Based Visual Environement Development Process”, in proc. SEKE 02, July 15-19, 2002, Ischia, Italy, pp. 327-334.
[Fer]	F. Ferrucci, G. Tortora and G. Vitiello, “Visual Programming”, in Wiley Encyclopedia of Software Engineering, J. J. Marciniak E. D., 2nd Edition, in print.
[May99]	D. J. Mayhew, “The usability Engineering Lifecycle”, Morgan Kaufmann Publisher, inc., San Francisco, California, 1999.
[Nie93]	J. Nielsen, “Usability Engineering”, Academic Press, Cambridge, MA, 1993.
[Som96]	I. Sommerville, “Software Engineering”, Addison Wesley, 1996.

Appendix 1. Valutazione di usabilità del Symbol Editor di VLDesk

Rapporto della valutazione

N° Problema	Locazione	Problema	Principio Violato	Soluzione
1	Interfaccia principale del VLDesk	All'apertura del sistema appaiono troppe finestre, queste confondono subito l'utente poiché non sa da dove cominciare.	Orientamento	Sarebbe opportuno permettere all'utente di scegliere le operazioni che vuole eseguire e così aprire le relative finestre.
2	Interfaccia principale del VLDesk	Nella finestra principale del VLDesk, sulla barra principale non ci sono I tooltip a tutte le icone presenti.	Coerenza; Minimizzare il carico di memoria dell'utente.	E' preferibile inserire i tooltip su tutte le icone.
3	Symbol Editor	Non è chiaro come inserire un simbolo.	Minimizzare il carico di memoria dell'utente; Parlare il linguaggio degli utenti.	Sarebbe conveniente inserire nel menù "File", della finestra principale, la voce "New Symbol". Inoltre, sarebbe necessario inserire anche un'icona sulla barra principale per velocizzare l'operazione.
4	Symbol Editor	Nella barra "Tools" sono assenti i tooltip.	Minimizzare il carico di memoria dell'utente.	E' preferibile inserire i tooltip.
5	Menù Tools del Symbol Editor	Nel menù "Tools" la voce "Physical Palette" non dà la possibilità di chiudere la palette "Tools".	Minimizzare il carico di memoria dell'utente.	Fare in modo che tramite la voce "Physical Palette" presente nel menù "Tools" sia possibile aprire e chiudere la relativa barra degli strumenti.
6	Symbol Editor	Se si clicca su "Both" i pulsanti "Logical" e "Physical" rimangono attivi.	Coerenza	Sarebbe opportuno che dopo aver attivato "Both" anche "Physical" e "Logical" apparissero attivati.
7	Symbol Editor	Se si clicca sul pulsante "Grid" scompare la griglia ma il pulsante sembra essere non selezionato.	Minimizzare il carico di memoria dell'utente.	E' opportuno far cambiare lo stato dei pulsanti quando selezionati.

8	Symbol Editor	Per una migliore visualizzazione dei simboli è consigliabile disegnare nel quadrato più scuro che appare al centro del foglio. Questo non è chiaro.	Minimizzare il carico di memoria dell'utente.	Sarebbe opportuno fare apparire un tooltip che indichi tale zona.
9	Symbol Editor	Non è chiara la differenza fra "Attaching Region", "Area Box" e "Indirect Area".	Minimizzare il carico di memoria dell'utente; Parlare il linguaggio degli utenti.	Utilizzare dei termini più espliciti per l'utente.
10	Menù Edit del Symbol Editor	"Select All" nel menù "Edit" non funziona.	Prevenire errori.	E' opportuno controllare che tutte le funzioni che vengono inserite siano attive.
11	Symbol Editor	Dopo aver inserito del testo è impossibile effettuare dei cambiamenti.	Minimizzare il carico di memoria dell'utente.	Sarebbe opportuno che l'utente abbia la possibilità di cambiare il testo già inserito, senza dover cancellare quello precedentemente scritto e inserirne uno nuovo.
12	Symbol Editor	Dopo aver inserito delle annotazioni ad un linguaggio, queste appaiono sempre nel box "Static Annotation Text" anche se si seleziona un altro linguaggio.	Minimizzare il carico di memoria dell'utente; Prevenire errori.	Sarebbe necessario che quando si seleziona un altro linguaggio appaiono le relative annotazioni.
13	Finestra Annotation	Nella finestra "Annotation" l'help non funziona.	Aiuto.	Fare in modo che l'help sia sempre disponibile e nel caso non ci fosse rendere non attivo il pulsante.
14	Finestra Annotation	Dopo aver inserito un'annotazione se si clicca invio per aggiornare l'informazione il tasto non funziona, inoltre non consente nemmeno di andare a capo.	Coerenza; Minimizzare il carico di memoria dell'utente; Shortcut	Far in modo che l'utente esperto possa velocizzare il proprio lavoro utilizzando l'invio da tastiera per far memorizzare l'informazione immessa.
15	Symbol Editor	Nella palette di "Box Area" non ci sono tooltip, inoltre il puntatore non cambia forma.	Coerenza; Feedback.	Sarebbe opportuno che una volta che l'utente ha inserito l'oggetto logico e clicca sull'icona del puntatore, questo riprenda la relativa forma.

16	Symbol Editor	“Box Area” nella barra degli strumenti standard e “Area Box” per la palette dei colori hanno nomi diversi anche se si riferiscono allo stesso argomento.	Coerenza.	Controllare i tooltip sulle icone che si riferiscono allo stesso argomento.
17	Symbol Editor	La voce “Load Symbol” permette di aprire la finestra “Open Formato VLS” questo lascia intendere che è possibile aprire un file.	Minimizzare il carico di memoria dell’utente; Coerenza.	Sarebbe opportuno usare gli stessi nomi per identificare le stesse funzioni.
18	Symbol Editor	Sulla finestra principale del Symbol Editor, l’Help non è presente ma il pulsante è attivo.	Minimizzare il carico di memoria dell’utente; Coerenza.	Se non esiste l’help è bene che il pulsante non sia attivo.
19	Symbol Editor	La linea curva presente nella palette “Attaching Region” non realizza la connessione logica curva tra due simboli, ma diventa un quadrato pieno.	Prevenire errori; Coerenza.	E’ necessario che l’utente riesca a disegnare l’elemento che ha selezionato.
20	Symbol Editor	Alcuni connettori logici sembrano essere pieni ma quando vengono disegnati sono vuoti.	Prevenire errori; Coerenza.	E’ necessario che l’utente riesca a disegnare l’elemento che ha selezionato.