User-Centered Point of View to End-User Development

Philipe Palanque

LIIHS-IRIT, Université Paul Sabatier
118, route de Narbonne,
31062 Toulouse Cedex, France
palanque@irit.fr

Rémi Bastide

LIIHS-IRIT, Université Toulouse 1
place Anatole France,
31042 Toulouse Cedex, France
bastide@irit.fr

Abstract

This paper proposes a user centered point of view for the definition and the construction of end user development environments. Taking, as input, the seven stages of actions of Norman's action theory we propose a set of guidelines for easing end user development. The main criteria for the language are "types of application covered" and "closeness of the language with respect to the application domain". The main criterion for end user development environment is the "continuous and permanent feedback" proposed by the environment. These criteria are then exemplified on PetShop environment that aims at building highly interactive applications providing using the Interactive Cooperative Objects language (an object-oriented, distributed and concurrent programming language).

1 Introduction

Designing computer-based applications belongs to the type of human activities that is higly demanding on the user's side. In this paper we propose to use Norman's activity theory as a tool for investigating where the main difficulties can occur, and thus to provide design rules for notations and environments to support user's activities and reduce difficulties.
Bringing a user centered point of view to programming environment has as already been studied (Green 1990, Blackwell 1996) but we believe that new development in the field of software engineering can enhance previous results.
The paper is structured as follows. Section 2 introduces briefly Norman's model and presents how development activities relates to it. Section 3 is devoted to ways of reducing gaps both during execution and interpretation. Section 4 presents how the ICO visual language and its CASE tool PetShop provide mechanisms to reduce such gaps.

2 Norman's Action Theory and EUD

Figure 1 (left-hand side) presents the seven stages of action and is extracted from (Norman 1998). Right-hand side of the figure represents the execution path i.e. the set of activities that have to be carried out by the user in order to reach the goal. Left-hand side represents the set of activities that have to be carried out by the user in order to interpret the changes resulting from his/her actions in the real world.

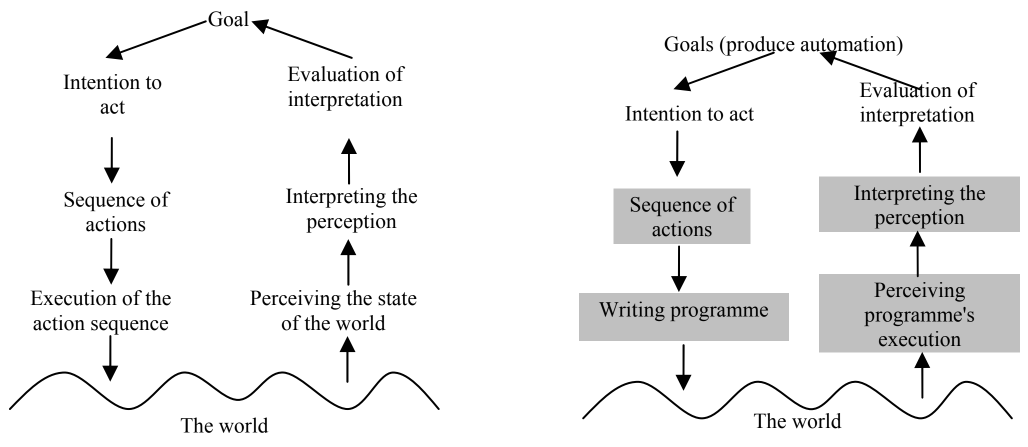
Goal
Intention to act
Evaluation of interpretation
Sequence of actions
Interpreting the perception
Execution of the action sequence
Perceiving the state of the world
The world
Goals (produce automation)
Intention to act
Evaluation of interpretation
Sequence of actions
Interpreting the perception
Writing programme
Perceiving programme's execution
The world

Figure 1: Seven stages of action (Norman 2001) (left)
Customization of Norman's model to EUD (right)

Figure 1 (right hand side) shows the refinement of specific activities from previous model to programming activities. The two main stages we are focusing on are:

- On the execution side, the activity of going form an intention to its actual transcription into some program,
- On the perception side, the activity of perceiving program execution and interpreting this perception.

3 Reducing Gulfs

There are several ways of reducing gulfs with respect to the models above. We investigate here two different and separated ways of reducing the gaps. The first one is related to the execution gulf and the second one is related to the evaluation gulf. Both of them are presented in next sections.

3.1 Reducing Execution Gulf

While designing a programming language, the designer must bear in mind that there is a tradeoff between the expressive power of the language and its "usability" [1] by the programmer. Due to space reasons, this very important aspect of usability of the language is not discussed in the paper.
One of the ways of making execution easier is to make description language closer to the application domain, thus making the concepts in the language close to the objects in the real word. This is of course much more important when end user programming is concerned as the end user is by definition highly connected to the real word. In fact having the end user writing the program directly avoids some classical potential usability problems when the vocabulary (used by the users in their everyday work) is different from the one used on the user interface by the computer scientists.
However, making the language closer to the application domain reduces its applicability to other domains. This has clearly been a tendency both in classical programming languages and in end

[1] For a concrete and measurable definition of usability of EUP languages please refer to (de Souza et al. 2001)

user development languages. Computer science has always been trying to provide generic programming languages that could be applied to a wide range of systems. This is a basic requirement for a programming language dedicated to professional programmers, as, for sake of efficiency, programmer's skills and knowledge must cover several potential application domains. On the other side, a lot of end user programming environments have been dedicated to specific domains such as simulation (Cypher & Smith 1995), geographic information systems (Traynor C. & Williams 1997) or air traffic control applications (Esteban et al. 1995), among others.
As a conclusion to this section, it is possible to reduce the execution gulf by providing programming languages in which constructs are close to the application domain. This is only true as the end users, by definition, only deals with applications that are heavily related to their activities and do not consider building applications for other users.

3.2 Reducing Evaluation Gulf

A way of reducing the evaluation gulf is to make easier the relationship between program edition and program execution. Usually these two activities are made available to the user in a modal way: first writing the program and then (usually in a different context) execute the program. Making these two activities separated and modal introduces difficulties in both perceiving the program execution and interpreting its behaviour (Butler et al. 1998).
In order to reduce this gulf we propose the support of these two activities in parallel within the programming environment. We have developed such a programming environment called PetShop. Its architecture as well as its use on simple case study is presented in section 4.

4 ICOs and PetShop

In previous work we have defined an object oriented distributed, concurrent and visual language called Interactive Cooperative Objects (ICO) (Bastide & Palanque 1999). This language is dedicated to the construction of highly interactive distributed applications. It is to be used by expert programmers with skills in: formal description techniques, object oriented approaches, distributed and interactive systems. Even though the programmers were not end users of the applications to be constructed, our goal was to increase usability of the language by providing ways of reducing evaluation gulf.

In order to show on a concrete example the concepts introduced above we have decided to use a simple case study in the field of components engineering for interactive applications.

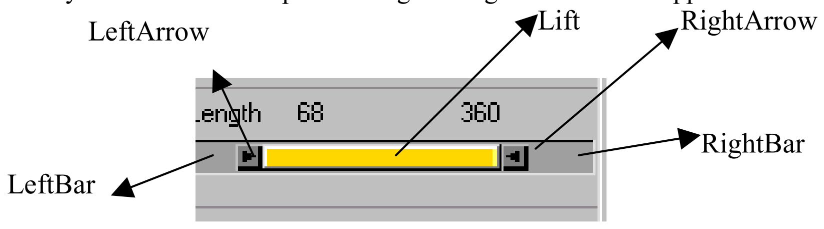

Figure 2: A simple application: a range slider (Ahlberg & Shneiderman 94)

A range slider is a basic interactive component that allows user to select values within a range (between a lower and an upper bound). The Range Slider belongs to the hybrid category of interactors as it can be manipulated both in a discrete and continuous way. This kind of compound quite complex interactors are more and more used in interactive applications and companies building user interface toolkits (such as Microsoft and Ilog) have already invested in component technology. However, the more complex the components, the less reliable they are.

4.1 Language

Using the ICO language it is possible to describe the entire behaviour of such an interactor. We only present here a subset of this description i.e. its behaviour (see Petri net model in Figure 3). The complete description of the case study can be found in (Navarre et al. 2000).
This behavioural description models the set of events the range slider can react to (mouse up, mouse move and mouse down), the set of states it can be in (the distribution of tokens in the places (ellipses) Petri net) and the set of actions the range slider can perform (the transitions (rectangles) in the Petri net).
One of the problem of building such a concurrent program is, first to understand its behaviour and then to understand whether this behaviour is similar to the one that was expected.

4.2 PetShop Environment

By providing a way of having both program execution and program edition at a time PetShop allows for rapid prototyping, iterative and modeless construction of applications (Navarre et al. 2001). Figure 3 presents a snapshot of Petshop at runtime. The small window on top of the picture corresponds to the execution of the visual program (the Petri net) in the bigger window underneath.

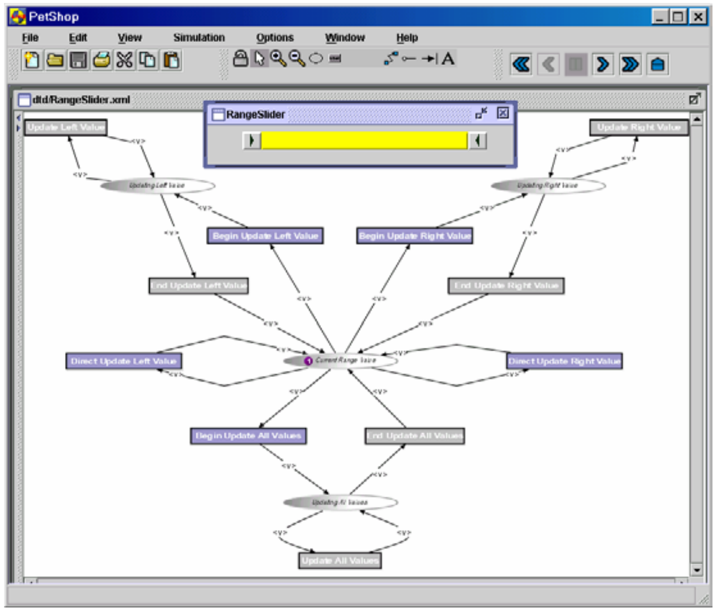

Figure 3: Integrated program edition and execution within PetShop

Thus, at a time, the programmer can interact with the application (use the range slider), see the impact of his/her action on the behaviour of the visual program. Another possibility is to modify the visual program and see immediately the impact on the program execution.
For instance, on Figure 3, the transition "begin Update Left Value" is associated to the left button of the range slider. If, by modifying the Petri net, the programmer makes this transition

unavailable (for instance by adding an input place without any token in it) then the button will immediately appeared as disabled (greyed out) and acting on it will have no effect.

5 Conclusions and Perspectives

This paper has proposed the use of Norman's model to understand the potential difficulties encountered by programmers. We have also shown how a programming environment (such as PetShop) that proposes program edition and execution in a modeless way could reduce those difficulties.

We have already conducted some early evaluation of ICO language and PetShop as part of the Mefisto LTR Esprit Project. However, in order to quantify and confirm the results of this early evaluation more usability tests should be conducted. This will be done in the framework of a military funded project starting in January 2003. This is a very important issue as pointed out in (De Souza et al. 2001).

6 References

Ahlberg C. & Shneiderman B. The Alphaslider: A Compact and Rapid Selector. ACM SIGCHI conference on Human Factors in Computing Systems, CHI'94, Boston, pp. 365-371. 1994.

Bastide R. & Palanque P. A Visual and Formal Glue between Application and Interaction. International Journal of Visual Language and Computing, Academic Press Vol. 10, No. 5, pp. 481-507. 1999.

Blackwell A. Metacognitive theory of visual programming: what do we think we are doing ? IEEE workshop on visual languages, pp. 240-246. IEEE computer society, 1996.

Butler R., Miller S., Potts J. & Carreño. A formal method approach to the analysis of mode confusion. In proceedings of 17th AIAA/IEEE Digital Avionics Systems Conference, 1998.

Cypher A. & Smith D.C. Kidsim: end user programming of simulations. Proceedings of ACM SIGCHI CHI 95 conference, Denver, USA. pp. 27-34, 1995.

de Souza C., Barbosa S. & da Silva S. Semiotic engineering principles for evaluating end-user programming environments. Interacting with Computers, vol. 13, pp. 467-495. Elsevier. 2001

Esteban O., Chatty S. & Palanque P. Visual construction of Interactive Software. IFIP conference on Visual Database, Lausane, Switzerland. pp. 145-160, Chapman et Hall, 1995.

Green T. Cognitive dimensions of notations. People and computers V, Cambridge University Press, R. Winder & A Sutcliffe (Eds), 1990.

Navarre D., Palanque P., Bastide R. & Sy O. A Model-Based Tool for Interactive Prototyping of Highly Interactive Applications. 12th IEEE, International Workshop on Rapid System Prototyping ; Monterey (USA). IEEE ; 2001.

Navarre D., Palanque P., Bastide R. & Sy O. Structuring interactive systems specifications for executability and prototypability. 7th Eurographics workshop on Design, Specification and Verification of Interactive Systems, DSV-IS'2000; Springer Verlag LNCS. n° 1946. 2000.

Norman D. (1998). The design of everyday things. MIT press 1998.

Traynor C. & Williams M. A study of end-user programming for geographic information systems. Seventh workshop on empirical studies of programmers, pp. 140-156. 1997.