From Model-based to Natural Development

Fabio Paternò

ISTI-CNR
Via G.Moruzzi, 1 – 56100 Pisa - Italy
f.paterno@cnuce.cnr.it

Abstract
Model-based approaches aim to support development through the use of meaningful abstractions in order to avoid dealing with low-level details. Despite this potential benefit, their adoption has mainly been limited to professional designers. This paper discusses how they should be extended in order to obtain environments able to support real end-user development, by which designers can develop or modify interactive applications still using conceptual models but with a continuous support that facilitates their development, analysis, and use.

1 Introduction
One fundamental challenge for the coming years is to develop environments that allow people without particular background in programming to develop their own applications. The increasing interactive capabilities of new devices have created the potential to overcome the traditional separation between end users and software developers. Over the next few years we will be moving from *easy-to-use* (which has yet to be completely achieved) to *easy-to-develop interactive software systems*. Some studies report that by 2005 there will be 55 million end-users, compared to 2.75 million professional users (Boehm et al., 2000).
End-user development in general means the active participation of end-users in the software development process. In this perspective, tasks that have traditionally been performed by professional software developers are transferred to the users, who need to be specifically supported in performing these tasks. New environments able to seamlessly move between using and programming (or customizing) can be designed. Some end-user development oriented techniques have already been added to software for the mass market, such as the adaptive menus in MS-Word or some programming-by-example techniques in MS-Excel. However, we are still quite far from their systematic adoption.
At the first EUD-Net workshop held in Pisa a definition of End User Development was established: "*End User Development is a set of activities or techniques that allow people, who are non-professional developers, at some point to create or modify a software artefact*". One of the goals of end-user development is to reach closeness of mapping: as Green and Petre put it (Green and Petre, 1996): "The closer the programming world is to the problem world, the easier the problem-solving ought to be.… Conventional textual languages are a long way from that goal". Even graphical languages often fail to furnish immediately understandable representations for developers. The work in Myers' group aims to obtain natural programming (Pane and Myers, 1996), meaning programming through languages that work in the way that people who do not have programming experience would expect. We intend to take a more comprehensive view of the development cycle, thus not limited only to programming, but also including requirements, designing, modifying, tailoring, ….. Natural development implies that people should be able to work through familiar and immediately understandable representations that allow them to easily express and manipulate relevant concepts, and thereby create or modify interactive software artefacts. On the other hand, since a software artefact needs to be precisely specified in order to be

implemented, there will still be the need for environments supporting transformations from intuitive and familiar representations into more precise, but more difficult to develop, descriptions.

2 Model-based Development

Model-based approaches can be useful for end-user development because they allow people to focus on the main concepts (the abstractions) without being confused by many low-level details. Through meaningful logical abstractions it is also possible to support participation of end-users already in the early stages of the development process.

In traditional software engineering, the Unified Modelling Language (UML) (OMG, 2001) has become the de facto standard notation for software models. UML is a family of diagrammatic languages tailored to modelling all relevant aspects of a software system; and methods and pragmatics can define how these aspects can be consistently integrated. Like programming, in order to be effective, modelling requires the availability of suitable and usable languages and support tools. Visual modelling languages have been identified as promising candidates for defining models of the software systems to be produced. UML and related tools such as Rationale Rose or ArgoUML are the best-known examples. They inherently require abstractions and should deploy concepts, metaphors, and intuitive notations that allow professional software developers, domain experts, and users to communicate ideas and concepts. This requirement is of prominent importance if models are not only to be understood, but also used and even produced by end users.

The developers of UML did not pay a lot of attention to how to support the design of the interactive components of a software system. Thus, a number of specific approaches have been developed to address the model-based design of interactive systems. Since one of the basic usability principles is "focus on the users and their tasks", it became important to consider task models. The basic idea is to focus on the tasks that need to be supported in order to understand their attributes and relations. Task models can be useful to provide an integrated description of system functionality and user interaction. This calls for identifying task allocation between the user and the system, and relating user and system views in an integrated design process, i.e., integrated modelling of user interface and system functionality. Then, the development of the corresponding artefact able to support the identified features should be obtainable through environments able to identify the most effective interaction and presentation techniques on the basis of a set of guidelines or design criteria.

Various solutions have been proposed for this purpose. They vary according to a number of dimensions. For example, the automation level can be different: a completely automatic solution can provide meaningful results only when the application domain is rather narrow and consequently the space of the possible solutions regarding the mapping of tasks to interaction techniques is limited. More general environments are based on the mixed initiative principle: the tool supporting the mapping provides suggestions that the designer can accept or modify. An example is the TERESA environment (Mori, Paternò and Santoro, 2003) that provides support for the design and development of nomadic applications, which can be accessed through different types of interaction platforms.
In order to move from model-based to natural development we have identified three key criteria that should be supported and will be discussed in the next section: integrated support of both informal and formal representations; effective representations highlighting the information of interest; and possibility of developing software artefacts from either scratch or an existing system.

3 Criteria for Natural Development Environments

In this section we discuss what design choices should be pursued to extend model-based approaches in order to obtain natural development environments.

3.1 Integrating informal and structured representations

Most end-user development would benefit by the combined use of multiple representations that can have various levels of formality. At the beginning of the design process many things are obscure and unclear. It is hard to develop precise specifications from scratch. In addition, there is the problem of clearly understanding what user requirements are. Thus, it can be useful to use the results of initial discussions to feed the more structured parts of the design process. In general, the main issue of end-user development is how to use personal intuition, familiar metaphors and concepts to obtain or modify a software artefact, whose features need to be precisely defined in order to obtain consistent support for the desired functionality and behaviour. In this process we should address all the available multimedia possibilities. For example, support for vocal interaction is mature for the mass market. Its support for the Web is being standardised by W3C (Abbott, 2001). The rationale for vocal interaction is that it makes practical operations quicker and more natural, and it also makes multi-modal (graphic and/or vocal) interactions possible. Vocal interaction can be considered both for the resulting interactive application and for supporting the development environment.

In CTTE (Mori, Paternò and Santoro, 2002), to support the initial modelling work we provide the possibility of loading an informal textual description of a scenario or a use case and interactively selecting the information of interest for the modelling work. To develop a task model from an informal textual description, designers first have to identify the different roles. Then, they can start to analyse the description of the scenario, trying to identify the main tasks that occur in the scenario's description and refer each task to a particular role. It is possible to specify the category of the task, in terms of performance allocation. In addition, a description of the task can be specified along with the logical objects used and handled. Reviewing the scenario description, the designer can identify the different tasks and then add them to the task list. This must be performed for each role in the application considered. Once designers have their list of activities to consider, they can start to create the hierarchical structure that describes the various levels of abstractions among tasks. The hierarchical structure obtained can then be further refined through the specification of the temporal relations among tasks and the tasks' attributes and objects. The use of these features is optional: designers can start to create the model directly using our editor, but such features can be useful to ease the modelling work. U-Tel (Tam, Maulsby, and Puerta, 1998) provides a different type of support: through automatic analysis of scenario content, nouns are automatically associated with the objects, and verbs with the tasks. This approach provides some useful results, but it is too simple to be generalised. For example, in some cases a task is defined by a verb followed by an object.

Another useful support can be obtained starting with the consideration that often the initial model is the result of brainstorming by either one single person or a group. Usually people start with some paper or whiteboard sketches. This seems an interesting application area for intelligent whiteboard systems (Landay and Myers, 2001) or augmented reality techniques able to detect and interpret the sketches and convert them into a format that can be edited and analysed by desktop tools. The possibility of developing through sketching can be highly appreciated in order to capture the results of early analysis or brainstorming discussions. Then, there is the issue of moving the content of such sketching into representations that can more precisely indicate what artefact should be developed or how it should be modified.

3.2 Providing Effective Representations

Recent years have seen the widespread adoption of visual modelling techniques in the software design process. However, we are still far from visual representations that are easy to develop, analyse and modify, especially when large case studies are considered. As soon as the visual model increases in complexity, designers have to interact with many graphical symbols connected in various ways and have difficulties analysing the specification and understanding the relations among the various parts

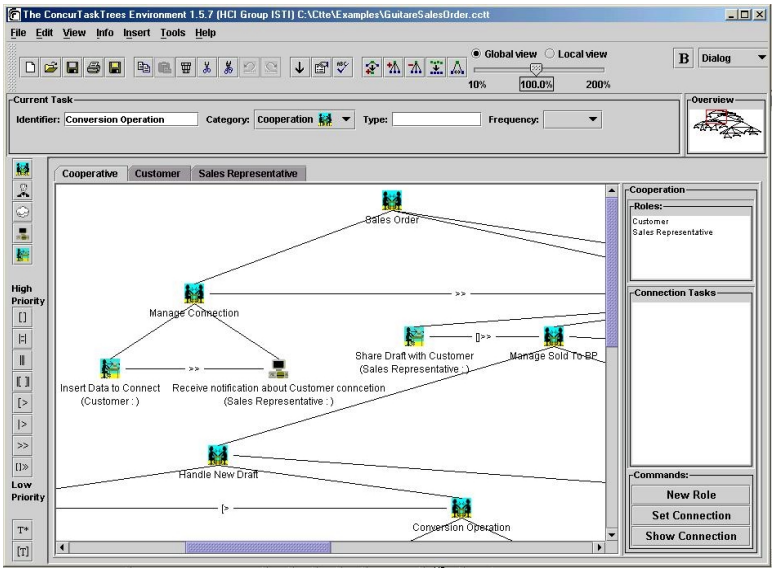

Figure 1: How CTTE provides focus+context representation.

CTTE provides support through some representations that ease the analysis of a model. It furnishes a focus+context representation of the model (see Figure 1). So, it is possible to have a detailed view of a part of the model in the main window (only a part can be displayed when large models are considered) but it still provides a small overview pane (in the right-top part) that shows the designer exactly where the part in the large window is located in the overall model. The representation in the overview pane is limited to show the hierarchical structure without indicating the task names and the icons that represent how the tasks are allocated. Identifying relations between the focus and the context window is facilitated by the hierarchical structure of the model.

In addition, an interactive system is a dynamic system so designers need to specify how the system can evolve: the sequential constraints, the possible choices, the interrupting activities and so on. When people analyse a specification, including the instances of the various temporal operators, they may have problems understanding the actual resulting dynamic behaviour. To this end, interactive simulators showing the enabled activities and how they change when any of them is performed can allow a better understanding of the actual behaviour specified.

In addition, the application and extension of innovative interaction techniques, including those developed in information visualization (such as semantic feedback, fisheye, two-hand interactions, magic lens…), can noticeably improve the effectiveness of the environments aiming to provide different interactive representations depending on the abstraction level of interest, or the aspects that designers want to analyse or the type of issues that they want to uncover.

3.3 Transformation-based environments

The starting point of development activity can often vary. In some cases people start from scratch and have to develop something completely new; in other cases people start with an existing system (often developed by somebody else) and need to understand the underlying conceptual design in order to modify it or to extend it to new contexts of use. Thus, a general development environment should be able to support a mix of forward (from conceptual to concrete) and reverse (from concrete to conceptual) engineering processes. This calls for environments that can support various transformations able to move among various levels (code, specification, conceptual description) in both a top-down and bottom-up manner and to adapt to the foreseen interaction platforms (desktop, PDA, mobile phones, …) without duplication of the development process.

While TERESA is an example of forward engineering, WebRevenge (Paganelli and Paternò, 2002) is an example of reverse engineering: it is able to take the code of a Web site implemented in HTML and reconstruct the corresponding task model through an analysis of the interaction techniques, tags and links existing. The combination of TERESA and WebRevenge allows a process where a designer takes an existing Web site for desktop systems, reconstructs its logical model through Web Revenge, analyses and modifies it in order to redesign the application for a mobile device and generates a new version of the application for the different platform with the support of TERESA.

4 Conclusions and acknowledgments

This paper provides a discussion of how model-based design should be extended in order to obtain natural development environments. After a brief discussion of the motivations for end user development, we set forth the criteria that should be pursued in order to obtain effective natural development environments. This can be achieved by extending traditional model-based approaches. In order to make the discussion more concrete, a specific environment for model-based design (CTTE) has been considered. We gratefully acknowledge support from the European Commission through the EUD-Net Network of Excellence (http://giove.cnuce.cnr.it/eud.html).

5 References

Boehm, B. W., Abts, C., Winsor Brown A., Chulani, S., Clark, B. K., Horowitz, E., Madachy, R., Reifer, D. J., and Steece, B., (2000). Software Cost Estimation with COCOMO II, Prentice Hall PTR, Upper Saddle River, NJ.

Green, T.R.G., Petre M. (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework, in J. Visual Lang. and Computing, Vol.7, N.2, pp.131-174.

Landay J. and Myers B., (2001). Sketching Interfaces: Toward More Human Interface Design. In IEEE Computer, 34(3), March 2001, pp. 56-64.

Mori G., Paternò F., Santoro C., (2002). CTTE: Support for Developing and Analysing Task Models for Interactive System Design, *IEEE Transactions in Software Engineering,* pp. 797-813, August 2002 (Vol. 28, No. 8), IEEE Press.

Mori, G., Paternò, F., Santoro, C., (2003) Tool Support for Designing Nomadic Applications, Proceedings Intelligent User Interfaces '03, pp.141-148, ACM Press, 2003.

OMG (2001). Unified Modeling Language Specification, Version 1.4, September 2001; available at http://www.omg.org/technology/documents/formal/uml.htm

Paganelli, L., Paternò, F., (2002) Automatic Reconstruction of the Underlying Interaction Design of Web Applications, Proceedings SEKE Conference, pp.439-445, ACM Press, Ischia.

Pane J. and Myers B. (1996), "Usability Issues in the Design of Novice Programming Systems" TR# CMU-CS-96-132. Aug, 1996. http://www.cs.cmu.edu/~pane/cmu-cs-96-132.html

Tam, R.C.-M., Maulsby, D., and Puerta, A., (1998). U-TEL: A Tool for Eliciting User Task Models from Domain Experts, *Proceedings IUI'98*, ACM Press, 1998.