Natural Modelling of Interactive Applications

Fabio Paternò, Marco Volpe

ISTI-CNR, Via G.Moruzzi
1 56126 Pisa, Italy
`fabio.paterno@isti.cnr.it`
`http://giove.isti.cnr.it/`

Abstract. This paper presents an approach, and the associated environment, aiming to support designers to intuitively model interactive applications. The goal is to make modelling activity more natural. The approach is based on a transformation able to convert hand-drawn visual model sketches on boards into task-model specifications that are further editable and analysable through automatic visual tools. A first test of the environment has been carried out and has provided useful suggestions about how to improve it.

1 Introduction

Design and development of interactive software systems still require considerable effort. In natural interaction the goal is to ease the user interaction by providing techniques that make the interaction with a system similar to how humans usually interact. This has raised further interest in new interaction modalities and multi-modal interfaces in general. This natural interaction paradigm can also be applied to the design and development cycle in order to ease building interactive software systems. Some work has already been dedicated to obtaining natural programming [12], which aims to support programming through languages understandable by people without \specific programming skills. On the one hand, natural development implies that people should be able to work through familiar and immediately understandable representations that allow them to easily express relevant concepts, and thereby create or modify applications. 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 precise −but more difficult to develop− descriptions.

Thus, integrating informal and structured specifications is a key issue. The usability of development environments can benefit from using multiple representations with various levels of formality. In fact, at the beginning of the design process many things are vague and unclear, so it is hard to develop precise specifications from scratch, especially because a clear understanding of the user requirements is a non-trivial activity. The main issue is how to exploit personal intuition, familiar metaphors and concepts to obtain/modify a software artefact. Examples of informal input for more structured representations are textual scenarios [15] and sketches on boards [8]. For example, non-programmer users feel comfortable with sketch-based systems that allow them to concentrate on concepts by exploiting natural interactions, instead of being distracted by cumbersome low-level details required by rigid symbolisms. Such systems are generally able to recognise graphical elements and convert them into formats that can be edited and analysed by other software tools.

The use of hand-drawn sketches for supporting the design cycle has already been considered in several works but, to our knowledge, it has not been considered for supporting the development of HCI models, such as task models. Software Design Board [17] has addressed the issue of supporting a variety of collaboration styles but in creating UML diagrams. SketchiXML [4] is a tool that allows the creation of sketches representing how the user interface should appear and convert them into logical concrete descriptions of the user interface that can then be manipulated for further processing through other tools. The Calì library [5] has been used to develop JavaSketchIt, a tool able to recognise user-drawn sketches representing user interfaces and generate the corresponding Java implementation.

Our work has a different goal: to provide an environment that makes it possible to interpret informal sketches and translate them into the associated task model descriptions that can be used for various purposes, including development of ubiquitous interfaces. In particular, one of the main contributions is in supporting a transformation able to recognise the semantic and syntactical structure of a ConcurTaskTrees task model [14] starting with some basic symbols recognised from hand-drawn sketches. This is part of a more general effort aiming to support end user development, which can be defined as a set of methods, techniques, and tools that allow users of software systems, including non-professional software developers, at some point to create, modify or extend a software artefact. In particular, our ultimate goal is to create environments that allow people with different backgrounds to easily create task models of ubiquitous interfaces, which can be used for prototyping the corresponding interfaces for the various devices supporting them. Indeed, a tool (TERESA [11]) already exists that is able to support a number of transformations to generate user interfaces adapted to different interaction platforms and modalities starting with logical descriptions of task models. The transformation from task descriptions to the user interface for a given platform exploits intermediate conceptual user interface descriptions and takes into account the interaction resources of the target platform. One advantage of this approach is that it allows designers to focus on the conceptual aspects without having to deal with a plethora of low-level implementation details of the potential devices. Thus, designers can be people without programming skills.

In this area the introduction of visual modeller tools represented an important step forward in the adoption of model-based approaches but it is not enough because many people still find this activity difficult and tedious, even when supported by such visual tools. Thus, research has started aiming to provide further support. Possible solutions have considered the use of textual scenarios where names are automatically associated with objects and verbs with tasks [16], or the use of vocal interfaces supported by some natural language processing [2]. In other cases the use of reverse engineering techniques applied to either user interface implementations [3] [13] or logs of user interactions [7] has been considered.

In this paper we present a novel solution based on the analysis of hand-drawn sketches. In particular, we first introduce the overall architecture of the proposed environment, then we discuss how the sketches are interpreted and how the result of the first low-level recognition is used to obtain the task model specification. The results of a first test are reported along with some useful suggestions for improvements that have emerged. Lastly, conclusions and indications for future work are presented.

2 The Architecture of the Environment

A good deal of work has been dedicated to developing tools able to convert hand-written sketches into digital formats or to support hand-written sketches of user interfaces and then convert them into corresponding specifications or implementations. However, so far no work has been dedicated to the development of environments able to recognise task models from hand-written sketches on board. This is an important goal because often the modelling work is time-consuming and a bit tedious, and such support can ease this phase. In addition, often the modelling work is the result of an interdisciplinary discussion carried out with the support of boards, where the involved actors can easily draw, modify, and cancel the elements composing the model. In designing our environment we have considered two previously available tools: the Mimio board [9] and the CTTE (ConcurTaskTrees Environment) tool [10]. The former is a device able to detect the strokes drawn with special pens on the board using ultrasound signals. It uses a high-resolution ultrasonic position capture system consisting of a capture bar, colour-coded marker sleeves and an electronic eraser. The capture bar is a ultrasonic tracking array positioned along the upper left edge of the whiteboard. The capture bar connects to a personal computer. The electronic marker sleeves transmit an ultrasonic signal to the capture bar, which triangulates the pen's position on the board as the user writes. The only change users must make is to be sure they use the electronic eraser to make corrections, since Mimio cannot capture changes made with a standard eraser or with one's fingers. The Mimio system captures each move of a marker or stylus on the whiteboard surface as digital data that expresses vector strokes over time. The information detected can be stored in various electronic formats, including SVG, the language for vectorial graphical representations in Web environments. The CTTE environment is a publicly available tool that supports the possibility of editing and analysing task models specified in the ConcurTaskTrees notation. The features supporting editing are multi-fold (direct manipulation of the model structure, layout improvement, lining up of tasks at the same level, …) as well as those supporting analysis (interactive simulator, metrics calculation, comparison of models, …). The tool can import/export specifications in an XML-based description. Consequently, as shown in Figure 1 (rectangles represent processes and hexagons represent their outputs), the new environment that we have designed has to support the transformation from the SVG description of the hand-written diagram into a XML specification of the task model that can be imported in the tool supporting further editing and analysis.

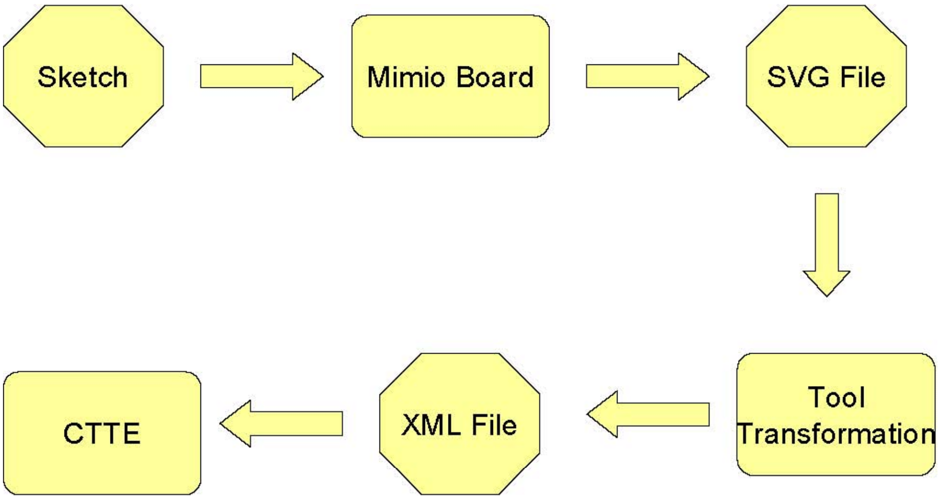


Figure 1: The Architecture of the environment.

Thus, one main component of our environment is a transformation able to take as input the low-level descriptions of the hand-drawn sketches and build the logical and syntactical structure of the corresponding task model. In next sections we discuss the solution that has been identified for this purpose.

3 Recognizing hand-written sketches

As we have introduced in the foregoing, our environment has to provide a transformation able to take the description of a hand-written sketch and convert it into a specification of the task model that can be imported in a tool for further editing or analysis or used for development. To this end, we have not implemented the transformation able to take the elementary drawings and identify the corresponding symbols because a number of tools have already been developed for this purpose. Rather, we have designed a transformation able to take the set of elementary symbols associated with users strokes, recognise the corresponding elements of the task model and identify the associated structure in order to obtain a specification that can be imported in the task modelling environment. In particular, in order to recognise the basic symbols we have analysed two possible solutions: the Rubine [15] and Calì [5] libraries. Both of them have advantages and disadvantages. In Rubine's approach the set of basic graphical symbols recognisable is modifiable but it is only able to recognise graphic symbols drawn through a single stroke. In Cali it is possible to recognise graphical symbols that are drawn through multiple strokes but the set of recognisable gestures is fixed and it also requires information regarding when each stroke has been drawn. Thus, we have decided to use the Rubine's algorithm because it does not put any limitation in terms of graphical symbols that can be recognised and does not require temporal information regarding the various gestures (our electronic board does not provide such temporal information). Figure 2 shows an example of use of the electronic board for drawing a task model. It allows designers to freely draw and modify sketches of the visual specification.

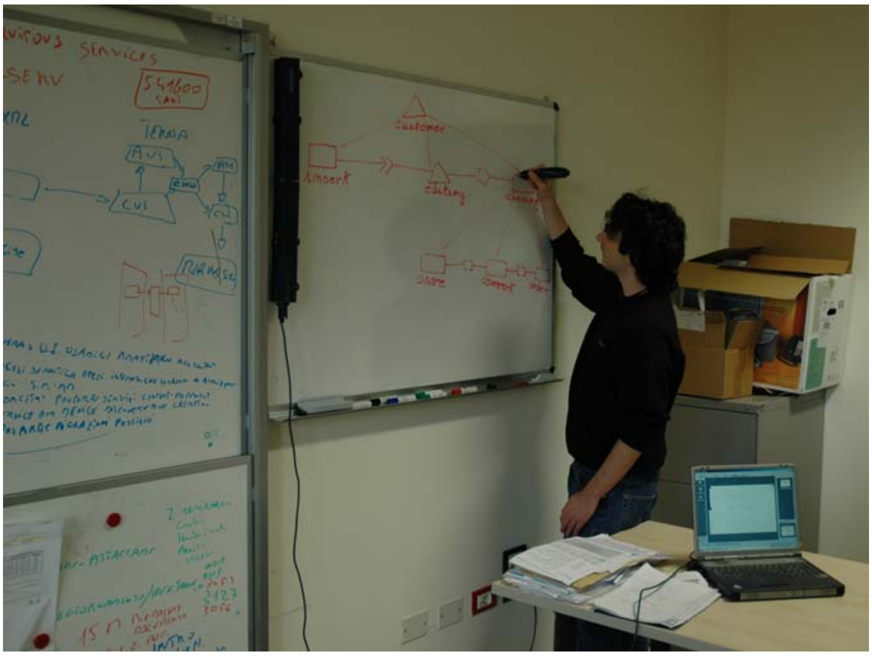
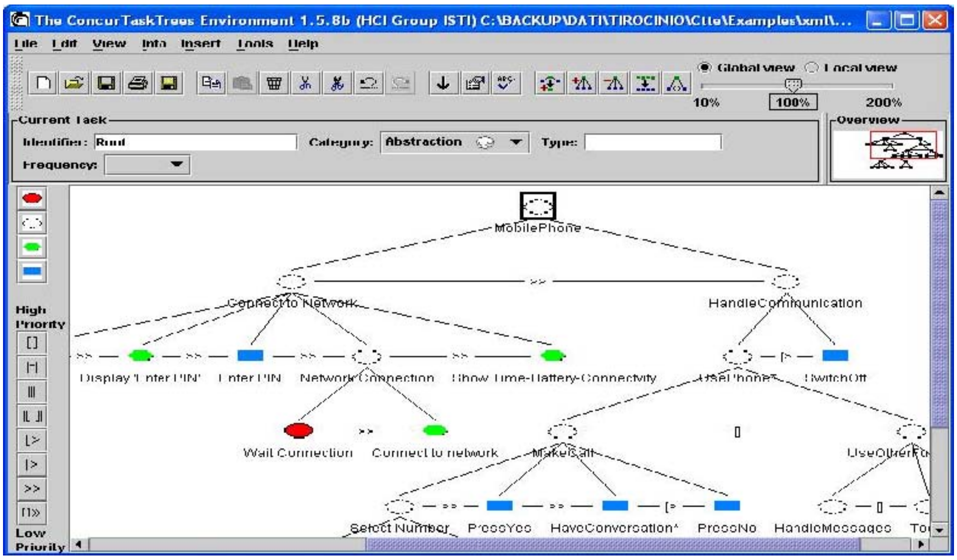


Figure 2: An example of use of the proposed environment.

4 The Transformation from sketches to models

The main visual elements in ConcurTaskTrees are the task names, the graphical symbols indicating how task performance is allocated, the symbols associated with temporal operators and the lines indicating task decomposition and task siblings. Figure 3 shows the user interface of the CTTE environment with an example of task model specification. One feature of the notation is to have different icons to represent task allocation: user (only internal cognitive activities), system (automatically performed), interaction, abstract (higher level tasks whose subtasks are allocated differently). Such icons can be represented in the tool in two ways selectable by the user: either images conveying the type of allocation or geometrical shapes that are useful when people draw task models on paper or board. The model in Figure 3 uses icons in the latter representation. In the new environment, we have simply changed the representation of abstract tasks from dotted ovals to triangles, which are easier to draw.



The ConcurTaskTrees Environment 1.5.8b (HCI Group ISTI) C:\BACKUP\DATI\TIROCINIO\Ctte\Examples\xml\...

● Global view ○ Local view

Current Task

Identifier: Root

Category: Abstraction

Type:

Frequency:

Figure 3: An example of CTTE representation of the task model.

In order to perform the transformation, we have first carefully identified the set of basic symbols that should be recognised (they are represented in Figure 4). They are the graphical shapes representing task allocation, the lines connecting tasks, the symbols representing the temporal operators and the letters used for the task identifiers. In particular, the temporal operators are represented by composing five elementary symbols (“[“, “]”, “>”, “|”, “=”). Regarding the lines for representing the structure of the tree we have considered four types of lines: vertical lines, two types of oblique lines, and horizontal lines. For each elementary graphical symbol only one method for drawing has been defined. For example, in the case of a triangle, the user has to start with the vertex located at the top and then draw in the anticlockwise direction.

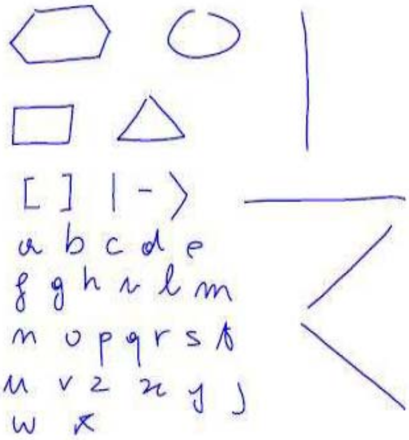
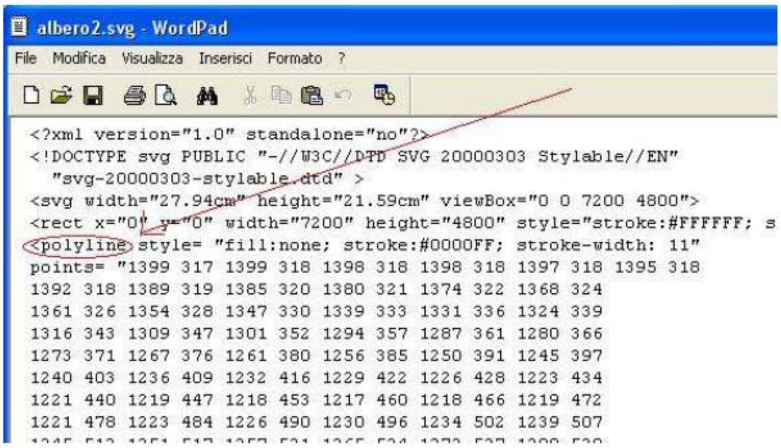


Figure 4: The set of basic symbols recognised.

In order to apply the transformation there is first a training phase during which at least ten examples of each basic graphical symbol to be recognised are provided and analysed through the Rubine algorithm. We have used a Java implementation of this algorithm, which has been made publicly available by the GUIR group at University of Berkeley [6]. This algorithm is limited to the recognition of the basic symbols associated with each stroke type. To this end, the algorithm associates mathematical representations to the main geometrical features of the symbol, then it applies a function to identify which of the possible symbols is closest to the one currently considered.

The SVG file generated by the Mimio board is rather simple. Even if SVG supports various constructs, the files generated mainly contain only various polyline instances. Each polyline is associated with one stroke representing a single graphical element. The Rubine algorithm identifies the element associated with the graphic symbol detected by analysing the previously provided example set of each possible element and evaluating what the most similar is. Then, our algorithm performs a number of further processing steps necessary to recognise the elements of the task model and its structure. Task names are identified by analysing the positions of the characters recognised and grouping those which are located closest to each other. A similar process is followed to recognise the representations of temporal operators. A task is associated with each symbol instance representing task allocation, and the closest name recognised is associated to this instance.



```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 20000303 Stylable//EN"
  "svg-20000303-stylable.dtd" >
<svg width="27.94cm" height="21.59cm" viewBox="0 0 7200 4800">
<rect x="0" y="0" width="7200" height="4800" style="stroke:#FFFFFF; s
<polyline style= "fill:none; stroke:#0000FF; stroke-width: 11"
points= "1399 317 1399 318 1398 318 1398 318 1397 318 1395 318
1392 318 1389 319 1385 320 1380 321 1374 322 1368 324
1361 326 1354 328 1347 330 1339 333 1331 336 1324 339
1316 343 1309 347 1301 352 1294 357 1287 361 1280 366
1273 371 1267 376 1261 380 1256 385 1250 391 1245 397
1240 403 1236 409 1232 416 1229 422 1226 428 1223 434
1221 440 1219 447 1218 453 1217 460 1218 466 1219 472
1221 478 1223 484 1226 490 1230 496 1234 502 1239 507
```

Figure 5: An excerpt of a SVG specification generated by the Mimio software.

Then, our algorithm first identifies the root task in the hierarchical structure, which is that positioned in the highest part of the board. In order to recognise the structure of the model we divide the lines between those useful to identify parent/child relations (vertical and oblique ones) and those for identifying sibling nodes (horizontal lines). Once a vertical or oblique line has been recognised, our algorithm looks for the task closest to the top and the bottom, which will be the parent and the child task, respectively. Likewise, two sibling tasks are identified when a horizontal line is recognised. Once the model has been built, it is coded into the corresponding XML-based specification, which can be imported into the CTTE environment.

In order to perform its processing our tool takes as input the SVG file created by the electronic board and its software. It also supports previewing of the hand-drawn representation represented through SVG. Then, it asks for the set of examples to use to identify the main features of each basic symbol to recognise. Such input is used to calculate the task model structure. Figure 6 shows an example of the SVG representation and the corresponding task model imported in the CTTE environment.

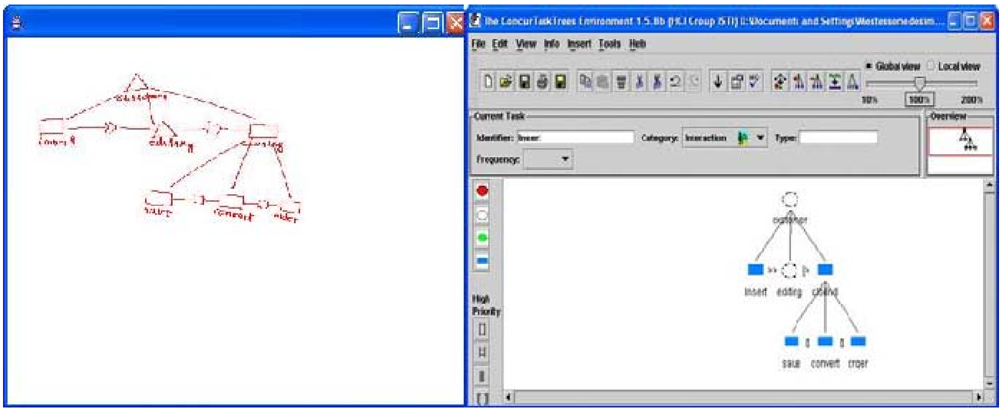


Figure 6: SVG representation of hand-drawn model and its transformation.

5 A First Test

Once we had designed and created a first prototype, we tested it in order to assess the level of reliability and correctness of the transformation supported. In practise, we performed two types of test: one aiming at revealing whether the tool provides correct results if it uses a set of examples provided by another user; the other one to highlight the level of reliability in recognising the task model. Overall, the test involved 14 users recruited in the Institute or among friends.

In the first test different people created a set of hand-written examples for each symbol. Then, they had to create one task model and the tool was applied using the different sets of examples available in order to understand whether the recognition ability was affected by the person who drew the examples used by the algorithm. Thus, we compared the results when the set was provided by the same person who provided the examples or a different one. The test showed that when the task model was drawn by the same person almost everything was correctly recognised, whereas some problems came up in the case of a different person. In particular, there were some errors in recognising some characters (this was deemed a minor issue since if a character is not correct in a task name it can be easily correct later on), sometimes hexagons were replaced with triangles and some problems were detected in recognising horizontal lines.

In the other test, the 14 users had to draw a predefined task model on the board. Due to time constraints, before the exercise they received little information regarding the type of board used and how they had to draw the basic symbols in a single stroke. They were asked to exploit all the space available in the board by distributing all the elements of the visual model. The results were then analysed at three levels:

- If the XML specification was generated; this would not have been possible in the event that some tasks were not recognised or were confused with other elements or in the event there was confusion between horizontal and vertical or oblique lines;
- If the hierarchical structure was recognised (mainly the parent/children relations);
- if the temporal relations were recognised, this implied recognising the horizontal lines and the temporal operators.

While the first two levels were completely correct in the large majority of the cases (75% and 62%), some serious issues were detected in the third level. The problem was due to a lack of reliability in the recognition of horizontal lines, whereas the symbols representing temporal operators were recognised in the majority of cases. However, this issue can be addressed in various ways and solved. First of all, the subjects were requested to draw temporal relations by also drawing two segments indicating to what tasks they were associated, thus obtaining a representation identical to that supported in the CTTE tool. Some modellers pointed out that when they draw task models on the board or on paper they do not actually draw such lines, they just put the temporal operator symbol and its position clearly indicates to what tasks it refers to. In addition, if we still want to keep the horizontal lines, then it would be sufficient to add pre-processing to identify first the various types of lines (vertical, horizontal, and oblique) using a solution similar to that supported by the CALI' environment.

Conclusions and Future Work

Modelling, as well as various phases in the design cycle, is sometimes a tedious activity. The introduction of visual tools has made the work of designers more efficient than when only pencil-and-paper is used but they still require considerable effort. There is a need for environments able to capture more immediately the representations resulting from a discussion or analysis of a possible design solution, such as those drawn on whiteboards.

The application of the natural interaction paradigm to modelling can provide useful results. While tools for providing recognition of low-level graphical symbols from hand-drawn sketches already exist, so far no proposal has addressed the conversion of these representations into task models able to support design of interactive systems.

In this paper we have presented a solution able to convert hand-drawn task models on an electronic board into an XML-based ConcurTaskTrees specification. A first prototype has been implemented and tested with a number of users. The results are encouraging and show the feasibility of the approach. Solutions for further improving the reliability of the recognition process have already been identified. Since there are tools (TERESA) able to support generation of ubiquitous interfaces starting with task model descriptions, this solution opens up the possibility of obtaining environments in which even people without programming skills can more easily design such interfaces.

Future work will be dedicated to supporting real-time transformation of hand-drawn sketches on board into desktop visual representation and extending the possibility of the environment in such a way as to capture both hand-written sketches and vocal description in various multi-modal combinations as input for creating the corresponding task model.

Acknowledgments

This work has partially been supported by the SIMILAR Network of Excellence on MultiModal User Interfaces (www.similar.cc).

References

1. Berti, S., Paternò, F., Santoro C., "Natural Development of Ubiquitous Interfaces", Communications of the ACM, September 2004, pp.63-64, ACM Press.
2. Berti, S., Paternò, F., Santoro C., Natural Development of Nomadic Interfaces based on Logical Descriptions, in End User Development, Springer Verlag, H.Lieberman, F.Paternò, W.Wulf (eds.), forthcoming.

3. Bouillon, L., Vanderdonckt, J. and Souchon, N., 2002. “Recovering Alternative Presentation Models of a Web Page with VAQUITA”, Proceedings of CADUI'02, Valenciennes, pp.311-322, Kluwer.
4. Coyette, A., Faulkner, S., Kolp, M., Limbourg, Q., Vanderdonckt, J., SketchiXML: Towards a Multi-Agent Design Tool for Sketching User Interfaces Based on UsiXML, Proc. of 3rd Int. Workshop on Task Models and Diagrams for user interface design TAMODIA’2004 (Prague, November 15-16, 2004), pp. 75-82.
5. Fonseca, M.J., Jorge, J.A.. Using Fuzzy Logic to Recognize Geometric Shapes Interactively. Proceedings of the 9th Int. Conference on Fuzzy Systems (FUZZ-IEEE 2000). San Antonio, USA, May 2000 Available from: http://immi.inesc.pt/projects/cali/publications.html
6. GUIRlib, available at http://guir.berkeley.edu/projects/guirlib/
7. Hudson S., John B., Knudsen K., Byrne M., 1999. A Tool for Creating Predictive Performance Models from User Interface Demonstrations, *Proceedings UIST’99*, pp.93-102.
8. Landay J. and Myers B., "Sketching Interfaces: Toward More Human Interface Design." In IEEE Computer, 34(3), March 2001, pp. 56-64.
9. Mimio Board, http://www.mimio.com
10. Mori, G., Paternò, F., Santoro, C., (2002), “CTTE: Support for Developing and Analysing Task Models for Interactive System Design”, IEEE Transactions on Software Engineering, pp.797-813, August 2002 (Vol. 28, No. 8), IEEE Press.
11. Mori, G., Paternò, F., Santoro, C., Design and Development of Multi-Device User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, August 2004, Vol.30, N.8, pp.507-520, IEEE Press.
12. 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
13. Paganelli, L., Paternò,F., 2003. A Tool for Creating Design Models from Web Site Code, International Journal of Software Engineering and Knowledge Engineering, World Scientific Publishing 13(2), pp. 169-189.
14. Paternò, F., Model-based Design and Evaluation of Interactive Applications, Springer Verlag, ISBN 1-85233-155-0, 1999.
15. Rubine D., Specifying Gestures by Example. Computer Graphics, vol. 25, no. 4, Luglio 1991.
16. Tam, R.C.-M., Maulsby, D., and Puerta, A., “U-TEL: A Tool for Eliciting User Task Models from Domain Experts”, *Proceedings IUI’98*, ACM Press, 1998
17. Wu J., Graham, N., The Software Design Board: A Tool Supporting Workstyle Transitions in Collaborative Software Design, in *Proceedings of Engineering for Human-Computer Interaction and Design, Specification and Verification of Interactive Systems,* LNCS 2844, Springer Verlag, 92-106, 2004.