Human–Computer Interaction — INTERACT'99
Angela Sasse and Chris Johnson (Editors)
Published by IOS Press, © IFIP TC.13, 1999

Plasticity of User Interfaces: Framework and Research Agenda

David Thevenin & Joëlle Coutaz

CLIPS-IMAG, BP 53, 38041 Grenoble Cedex 9, France.

{david.thevenin,joelle.coutaz}@imag.fr

Abstract: This paper introduces the notion of plasticity, a new property of interactive systems that denotes a particular type of user interface adaptation. Plasticity is the capacity of a user interface to withstand variations of both the system physical characteristics and the environment while preserving usability. Typically, a 'plastic' electronic agenda would run both on a workstation and on a hand-held computer without requiring a complete system redesign and re-implementation. We present a generic framework inspired by the model-based approach, for supporting the development of plastic user interfaces. Within this framework, a plastic user interface is specified once and serves multiple sources of physical variations. The goal is to guarantee usability continuity under variations in physical constraints while minimizing development and maintenance costs. This framework is illustrated with two simple case studies. Preliminary results and the state of the art in HCI open a new research agenda for the design and development of plastic user interfaces.

Keywords: user interface adaptation, plasticity, user interface, development process, usability.

1 Introduction

The need for ubiquitous access to information processing, the success of new consumer devices such as pocket computers and wireless networks, the availability of large electronic boards as well as the development of immersive caves, offer new challenges to the HCI software community. In particular, user interfaces need to accommodate the variability of a large set of interactional devices without leading to costly development efforts. For example, an electronic agenda should run both on a workstation and on a hand-held computer without requiring a complete system redesign and re-implementation. In this article, we introduce the notion of plasticity to denote this particular type of user interface adaptation.

In the next section, we define the coverage of plasticity within the problem space of adaptation. We then present a generic framework inspired from a model-based approach, for supporting the development of plastic user interfaces. This framework is illustrated with simple case studies. Preliminary results and the state of the art in HCI open a new research agenda for the design and development of plastic user interfaces.

2 Adaptation and Plasticity

2.1 Adaptation

In HCI, adaptation is modeled as two complementary system properties: adaptability and adaptivity (Browne et al., 1990; IFIP WG2.7, 1996). Adaptability is the capacity of the system to allow users to customize their system from a predefined set of parameters. Adaptivity is the capacity of the system to perform adaptation automatically without deliberate action from the user's part. Whether adaptation is performed on human requests or automatically, the design space for adaptation includes three additional orthogonal axes (see Figure 1): target, means, and time.

The target for adaptation This axis denotes the entities for which adaptation is intended: adaptation to users, adaptation to the environment, and adaptation to the physical characteristics of the system (e.g. interactional devices such as a mouse).

The means of adaptation This axis denotes the software components of the system involved in adaptation: typically, the system task model, the rendering techniques, and the help subsystems, may be modified to adapt to the targeted entities. The system task model is the system implementation of the user task model specified by experts on human factors. The rendering techniques denote the observable presentation and behavior of the system. The help subsystems include help about the system and help about the task domain.

The temporal dimension of adaptation Adaptation may be static (i.e. effective between sessions) and/or dynamic (i.e. occurring at run time).

A more comprehensive classification scheme can be found in (Dieterich et al., 1994) motivated by the detailed analysis of intelligent user interfaces

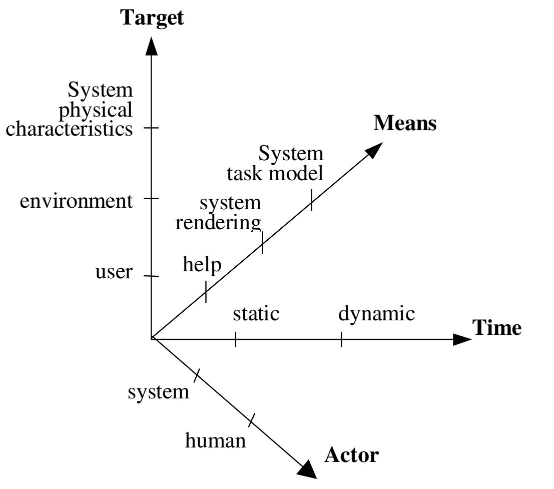


Figure 1: A design space for adaptation at a high level of reasoning.

2.2 Plasticity

The term 'plasticity' is inspired from the property of materials that expand and contract under natural constraints without breaking, thus preserving continuous usage. By analogy, *plasticity* is the capacity of a user interface to *withstand variations* of both the system physical characteristics and the environment while *preserving usability*. In addition, a plastic user interface is specified once to serve multiple sources of physical variations, thus *minimizing development and maintenance costs*. Plasticity may be static and/or dynamic. It may be achieved automatically and/or manually.

Within the design space of adaptation presented in Figure 1, plasticity is characterized in the following way:

1. Along the *target* axis, plasticity is concerned with the variations of the system physical characteristics and/or the environment. It does not, therefore, cover adaptation to users' variations.

2. Along the *means* axis, plasticity involves the modification of the system task model and/or of the rendering techniques.

3. The *temporal* and *automaticity* axes are left opened.

Technically, platform independence of code execution is not a sufficient condition to support plasticity. Virtual toolkits, such as the Java abstract machine, offer very limited mechanisms for the automatic reconfiguration of a user interface in response to variations of interactional devices. All of the current tools for developing user interfaces embed an implicit model of a single class of target computers (typically, a keyboard, a mouse and at least a 640x480 color screen). As a result, the rendering and responsiveness of a Java applet may be satisfactory on the developer's workstation but not necessarily usable for a remote Internet user. Therefore, platform independence is not sufficient to guarantee *usability continuity*. In addition, the iterative nature of the user interface development process, as well as code maintenance, make it difficult to maintain consistency between the multiple target versions.

In the absence of appropriate software tools, we need a framework for reasoning about the design and development of plastic user interfaces.

3 A Framework for Plasticity

Going one step further than the slogan for portability (i.e. "write it once and run it multiple times"), a framework for supporting plasticity should allow developers to *specify it usable and produce it multiple times*.

Model-based user interface generators, which promote the specification of high level models, provide an approach consistent with our requirements: they aim at usability while stressing computational reification. Our framework, inspired from such generators, should be viewed as an added-value to the models and methodological guides provided by the seminal design methods in HCI (e.g. MUSE (Dowell & Long, 1989), ADEPT (Johnson et al., 1993), or MAD (Scapin & Pierret-Golbreich, 1990). Our framework is not intended as a substitute to the models advocated by the design methods. Instead:

- It builds upon the models that work well in user interface design and user interface development.

- It improves existing models to satisfy the requirements imposed by plasticity.
- It explicitly introduces new models that have been overlooked or ignored so far.

Figure 2 shows the seven components of our framework. A detailed description is given in (Thevenin, 1998).

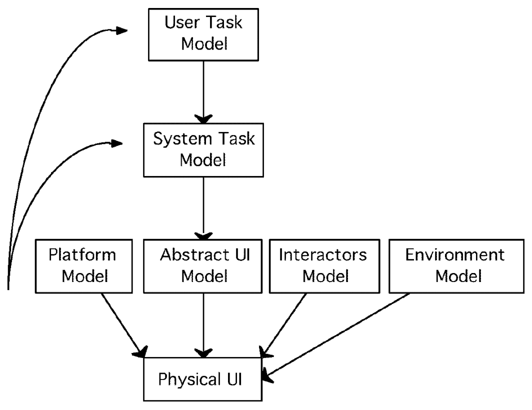


Figure 2: A framework for the development of plastic. Arrows denote models dependencies.

The *User Task Model* is a formal or semi-formal transcription of the real world activities. It results from the activity analysis performed by human factor specialists.

The *System Task Model* describes how the work tasks could be achieved with the introduction of the designed system. This model is informed by the User Task Model. The rules that govern the derivation of the User Task Model into a System Task Model are not well formalized yet. One can find in (Gamboa Rodriguez & Scapin, 1997) some early attempts at making the transformation explicit.

The *Abstract User Interface* is the canonical expression of an abstract rendering of the domain concepts and domain functions in conformance to the System Task Model. At this level of abstraction, rendering is interactor independent. It is similar in spirit to the notion of common information representation proposed in (Hüther & Rist, 1998). The notion of Abstract User Interface will be discussed further in Section 5.

The *Interactors Model* describes the interactors available for the physical rendering of a particular user interface. An interactor is an independent communicating agent characterized by an internal state and a perceivable state. It is capable of producing events and processing events caused by a user or by any component of the system. Graphical user interface widgets, speech sentences are examples of interactors. The interactor model has been used in a variety of ways including system architecture modeling (Coutaz et al., 1996), presentation mapping validity (Doherty & Harrison, 1997), and user interface conformity with system task modeling (Markopoulos et al., 1997).

The *Platform Model* describes the physical characteristics of the target platforms. To our knowledge, platform modeling has been considered implicitly as the pervasive workstation. Plasticity needs the explicit expression of the target platforms in terms of quantified physical resources. A platform model should include the interactional devices available (e.g. mouse, screen, video cameras), the computational facilities (e.g. memory and processing power), as well as the communicational facilities (e.g. bandwidth rate of the communication channels). Although Mackinlay et al. (1990) address input devices only, their theory provides sound foundations for modeling the physical resources of a system.

The *Environment Model* specifies the context of use. The scope of the notion of environment is still unclear in the literature. Within the context of our framework, the environment roughly covers objects, persons and events that are peripheral to the current task but that may have an impact on the system and/or the user's behavior. For example, a mobile phone that knows that it has entered a public area will automatically switch ringing signal from sonic to vibrating. Salber in (Salber & Abowd, 1999) gives an early attempt at defining the notion of environment along with software components that support the development of 'context sensitive' interactive systems.

The *Physical User Interface* results from the constraints resolution and/or constraints propagation expressed in the Platform Model, the Abstract User Interface Model, the Interactors and the Environment Models.

Finally, the models in relation to the physical world (i.e. the Platform, the Interactors, and the Environments Models) may in turn have an impact on the System Task and the User Task models. For example, users needs for reading email from a mobile phone may not be different from the requirements for processing email on a standard workstation. Therefore, the derivation of a plastic user interface (as any user interface) does not necessarily follow a pure top-down design process.

The framework of Figure 2 sets the foundations for the development of plastic user interfaces. To be operational, the models it involves must be structured and formalized. Next section provides preliminary requirements and heuristics in that direction.

4 Requirements and Heuristics

An early analysis of an electronic agenda has led us to formulate a number of requirements and heuristics in relation to the User Task Model, the System Task Model, and the Interactors Model. The electronic agenda runs both on a Palm Pilot and personal workstations (i.e. PC and Macintosh).

4.1 The User Task Model

With regard to the User Task Model, we advocate the following recommendation:

> "Domain concepts involved in a task should be ranked according to their level of importance in the task domain as well as according to their degree of centrality for this particular interaction point."

For example, in the agenda application, the day of the month and the month of the year are *first class objects*. Conversely, the day of the week (e.g. whether the 15th of October is a Wednesday or a Friday) is not necessary to carry most frequent tasks. It is a second class object. The *degree of centrality* of a concept denotes the extent to which that concept is part of the environment for that particular task or is used as a central item for carrying the task. Typically, the location of use of an agenda is not central for executing the task but may modify the way the task is carried out.

User and system task models and notations have no provision for specifying the relative importance and periphery of domain concepts. A notable exception, however, is the task-related information analysis presented in Sutcliffe (1997) and in TKS (Johnson et al., 1995). The authors make an explicit distinction between "the information required" and "the information needed" by the user to carry a task.

The availability of *first class and second class concepts*, as well as the *centrality* of concepts can help the rendering process using general heuristics. An example of such a heuristic is:

> "First class objects should be observable whereas second class objects may be browsable if observability cannot be guaranteed due to the lack of physical resources."

Another example is:

> "Peripheral objects may be browsable whereas central objects should be observable."

Observability is the capacity of the system to make perceivable the concepts relevant for carrying a particular task. Browsability allows the user to make a concept perceivable when this concept is not directly observable (IFIP WG2.7, 1996). In the agenda example, a date is displayed as "Wednesday October 15,1998" on a workstation. It is rendered as "15 Oct 98" on the Palm Pilot whose screen resources are scarce. Here, the day of the week, which is a second class object, is not observable.

4.2 The System Task Model

The System Task Model, which describes how the work tasks are achieved with the system, includes domain-dependent tasks and articulatory tasks. An *articulatory task* is domain independent. As such, it does not appear in the User Task Model and it does not modify the values of the domain concepts. In general, an articulatory task is induced by the physical apparatus and the rendering techniques of the system. For example, the lack of screen space may introduce extraneous tasks for accessing a sequence of screen contents that would otherwise be presented at once on a large display. Scrolling and navigating are typical articulatory tasks.

In the context of plasticity, browsability, when used for accessing second class objects, generates articulatory tasks. For example, opening the agenda at a certain date requires selecting the month and the day in the calendar. On the PC version, the calendar is always visible whereas on the Palm Pilot, the calendar must be opened on demand. The 'GoTo' command is an articulatory task introduced in the System Task Model of the Palm Pilot version to access the calendar, a browsable domain concept.

Our early analysis of simple case studies calls for the following recommendation:

> "Generic articulatory tasks should be identified in a systematic way and inserted appropriately in the System Task Model."

4.3 The Interactors Model

As mentioned in Section 3, interactors have been used for addressing distinct issues and purposes. As a result, a multiplicity of formal representations has been developed for specifying interactors. Although these modeling techniques are convenient for their particular context of use, none of them addresses the problem of plasticity explicitly. This leads us to formulate the following additional recommendation:

> "Interactors should specify the abstract data types they are able to handle. They should also be able to evaluate their

appropriateness as well as their rendering cost."

The abstract data type description provides the information necessary to perform the mapping between a domain concept and a set of candidate interactors. The modeling techniques of Mackinlay (1986) and Roth & Mattis (1990) have paved the way in this direction but for graphical non interactive presentation only. Fischer (1997) addresses a similar problem for direct manipulation without considering multi-modal interaction.

The appropriateness of an interactor denotes its user-centered effectiveness at rendering a particular concept for a particular task. Because of constraints imposed by the lack of physical resources, a concept may not be rendered optimally with regard to the designer's intent or to the user's requirements. A number of mapping rules may be found in Vanderdonckt (1995) for graphics user interfaces and in Sutcliffe (1997) and Bernsen (1994) for multimedia and multi-modal interaction. The capacity of an interactor to evaluate its appropriateness serves two purposes: It guides the selection process for concept rendering and it permits to quantify the continuity or the degradation of the system usability.

The rendering cost of an interactor expresses the quantity of physical resources needed for its instantiation. As discussed in Coutaz et al. (1995), an interactor belongs to a modality. A modality m is defined as the couple $m = < r, d >$ where r denotes a representational system, and d the physical I/O device used to convey expressions of the representational system. Both r and d can be characterized with a rendition cost Cr and Cd. For example, the display footprint is a way to express the rendition cost of a graphical interactor. Temporal footprint, i.e. the time to enunciate a vocal output message, expresses the rendition cost of a speech output interactor. The rendering cost drives the selection of the final interactors set for presenting domain concepts.

Figure 3 shows the thumb index interactor used on the PC for setting the visualization mode of the agenda, whereas the Palm Pilot version deploys a tiny set of three icons. The visualization mode is an enumerated data type (day, month, and year) that can be mapped to a set of three icon interactors. Both sets use the same physical output device (i.e. the screen) but two different representational systems (text and graphics) whose rendition costs Cr (their display footprint) are different. In the example of the date ("October 1998" vs. "Oct 98") the same representational system is used (text) using distinct pragmatic values that convey the same semantics at a lower footprint cost.

Having presented general requirements and heuristics for the components of our framework, we now discuss the principles for producing a Physical User Interface from an Abstract User Interface and an Interactors model.

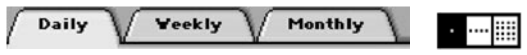


Figure 3: On the left, the thumb index used for the workstation. On the right, the icons set for the Palm Pilot.

5 From Abstract to Physical User Interfaces

5.1 Overall Principles

The notion of Presentation Unit (PU) used in TRIDENT (Vanderdonckt, 1995) sets the foundations of a canonical representation for the Abstract Interface Model. A PU is a hierarchical structure that models information containers such as workspaces, as well as elementary units that correspond to domain concepts at the appropriate level of granularity. In addition to the structural static description of a PU, PU's have a behavior and are linked by navigational relationships. These relationships reify task ordering as expressed in the System Task Model. The Physical User Interface is obtained by associating an interactor to every PU. Additionally, an interactors configuration is performed for PU's composed of sub-PU's.

For the selection of interactors, we advocate a two step process:

- Step 1: Correspondence matching between PU's and interactors.
- Step 2: Correspondence matching between the interactors selected in Step 1 and the physical resources provided by the system.

Correspondence matching between PU's and Interactors is based on consistency checking between the information flows that the PU's and the Interactors support respectively. An information flow is characterized with attributes including:

- Direction of information: one way input, one way output, two ways input and output,
- Information data type: type, domain of values, cardinality, etc ((Mackinlay, 1986; Roth & Mattis, 1990) for additional attributes). For a PU, the information data type corresponds to the domain concept supported by the PU. For an

interactor, the information data type denotes the abstract domain the interactor is able to render.

For example, the graphic thermometer and the bar chart interactors, the graphic and the audio real number, all support information flows that are compatible with PU Temperature. Interactors, such as the check box and the radio button, are not consistent with this PU. They are discarded in step 1 of the selection process.

Correspondence matching between the interactors selected in step 1 and the physical resources of the system, is based on constraints resolution between the rendering costs of the interactors ($< Cr, Cd >$) and the availability of the physical resources of the computer system. For example, the thermometer and the bar chart interactors both require a screen ($Cd =$ Screen) whereas, for the real number, $Cd =$ (Screen or Loudspeaker). In addition, their representational cost Cr ($Cr =$ Screen footprint) is higher than that of the graphic real number. On a mobile phone with no screen or with a small screen, the audio/graphic real number satisfies the constraints.

We have applied this simple process to MMS, a simple media space.

5.2 MMS

MMS conveys the level of activity of the community members of our lab. It currently runs on standard workstations but with varying window sizes as a means to simulate different physical screen sizes. The activity level is computed from the use of the mouse and the keyboard. Image differences from video cameras could also have been used as a source of input.

In this example, a single compound PU contains three elementary PU's: the purpose of the system, a quit command, the list of users currently connected to MMS, their activity level and the name of their workstation. All of the concepts are first class objects except the name of the workstation the user is currently connected to. Table 1 shows the information flow of the elementary PU's.

Concept	Direction	Data Type	Card	Domain
Person name	Out	ASCII	1	Undefined
Workstation name	Out	ASCII	1	Undefined
Level of activity	Out	Integer	1	[0, 10] step 1
Quit command	In	ASCII	1	{Quitter}
System Purpose	Out	ASCII	1	Undefined

Table 1: Information flow of the PU's for MMS.

The interactors that satisfy the PU's information flow include: the string interactor for the purpose of the system, as well as for the users and workstations names; the button interactor fits the quit command; the integer, the bar chart and the plot interactors match the level of activity. In our system, interactors are selected dynamically according to the size of the window. This window which, in the current implementation, simulates a physical screen, corresponds to the compound workspace PU. The sequence in Figure 4 shows the rendering of MMS according to screen space variations. Figure 5 illustrates the internal representation used to compute the final rendering. Each node is characterized with a cost and the set of matching alternatives for rendering a PU. An Oval node corresponds to an interactor decorated with its rendering cost. Diamond nodes represent interactors composition with layout rules. Its rendering cost is a linear combination of its siblings cost. Rectangle nodes denote rendering Alternatives. The final representation is computed using a constraint resolution algorithm based the rendering costs and physical resource availability.

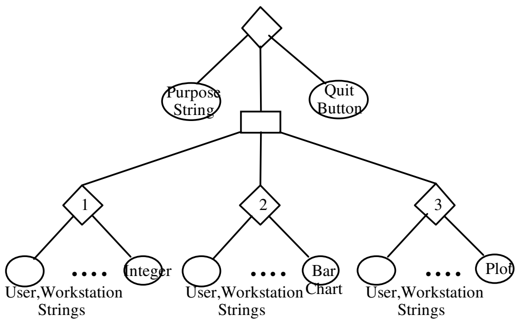
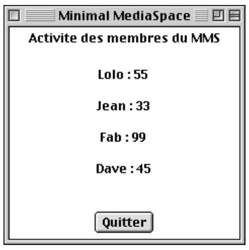


Figure 5: Internal representation of the solution space for rendering MMS.

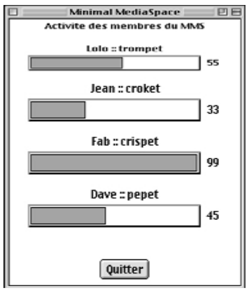
6 Conclusion

In this paper, we have introduced the notion of plasticity for addressing the portability of interactive systems in relation to usability continuity. Building on the model-based approach, we propose a framework that structures the problem space and provides the foundations for a solution space using high level specifications. Such descriptions trigger the appropriate design questions and open the way to the automatic or semi-automatic generation of plastic user interfaces. We have illustrated the feasibility of the approach with simple case studies: an off-the shelf electronic agenda, and MMS a media-space that conveys the level of activity of the community members of our lab. These preliminary results need to be completed with a more thorough analysis and formalization in order to make our framework operational.



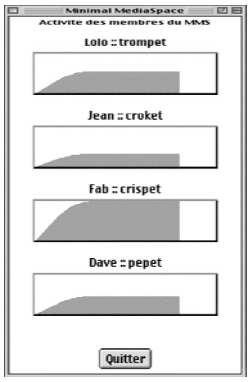
(a)

Rendering of MMS when using a small window. Only users' name and their level of activiy are observable using an integer value. Since workstation names are second class objects, they are not observable.



(b)

Rendering of MMS using bar charts when using a medium range window size.



(c)

Rendering of MMS when using a window large enough to accommodate plot interactors.

Figure 4: Dynamic rendering of MMS in response to screen space variations.

References

Bernsen, N. O. (1994), "Foundations of Multimodal Representations: A Taxonomy of Representational Modalities", *Interacting with Computers* **6**(4), 347–71.

Browne, D., Totterdell, P. & Norman, M. (eds.) (1990), *Adaptive User Interfaces*, Computer and People Series, Academic Press.

Coutaz, J., Nigay, L. & Salber, D. (1996), Agent-Based Architecture Modeling for Interactive Systems, *in* D. Benyon & P. Palanque (eds.), *Critical Issues in User Interface Engineering*, Springer-Verlag, pp.191–209.

Coutaz, J., Nigay, L., Salber, D., Blandford, A., May, J. & Young, R. (1995), Four Easy Pieces for Assessing the Usability of Multimodal Interaction: The CARE Properties, *in* K. Nordby, P. H. Helmersen, D. J. Gilmore & S. A. Arnessen (eds.), *Human–Computer Interaction — INTERACT'95: Proceedings of the Fifth IFIP Conference on Human–Computer Interaction*, Chapman & Hall, pp.115–20.

Dieterich, H., Malinowski, U., Kühme, T. & Schneider-Hufschmidt, M. (1994), "State of the Art in Adaptive User Interfaces". http://www.cc.gatech.edu/computing/classes/ cs8113d_94_fall/homepage.html.

Doherty, G. & Harrison, M. (1997), A Representational Approach to the Specification of Presentations, *in* M. Harrison & J. Torres (eds.), *Design, Specification and Verification of Interactive Systems'97*, Springer-Verlag, pp.273–90.

Dowell, J. & Long, J. (1989), "Towards a Conception for an Engineering Discipline of Human Factors", *Ergonomics* **32**(11), 1513–35.

Fischer, M. (1997), A Framework for Generating Spatial Configurations in User Interfaces, *in* M. Harrison & J. Torres (eds.), *Design, Specification and Verification of Interactive Systems'97*, Springer-Verlag, pp.225–41.

Gamboa Rodriguez, F. & Scapin, D. (1997), Editing MAD* Task Descriptions for Specifying User Interfaces at both Semantic and Presentation Levels, *in* M. Harrison & J. Torres (eds.), *Design, Specification and Verification of Interactive Systems'97*, Springer-Verlag, pp.193–208.

Hüther, M. & Rist, T. (1998), "Entering a shared Information Space through Heterogeneous Communication Devices", http://www.dfki.uni-sb.de/imedia/workshops/ecai98/#Program.

IFIP WG2.7 (1996), *Design Principles for Interactive Software*, Chapman & Hall.

Johnson, P., Johnson, H. & Wilson, S. (1995), Rapid Prototyping of User Interfaces Driven by Task Models, *in* J. M. Carroll (ed.), *Scenario-Based Design: Envisioning Work and Technology in System Development*, John Wiley & Sons, pp.209–46.

Johnson, P. W., S., M. & P., Pycock, Y. (1993), ADEPT-Advanced Design Environment for Prototyping with Task Models, *in* S. Ashlund, K. Mullet, A. Henderson, E. Hollnagel & T. White (eds.), *Proceedings of INTERCHI'93*, ACM Press/IOS Press, p.66.

Mackinlay, J. (1986), "Automating the Design of Graphical Presentations of Relational Information", *ACM Transactions on Graphics* **5**(2), 110–141.

Mackinlay, J., Card, S. K. & Robertson, G. (1990), "A Semantic Analysis of the Design Space of Input Devices", *Human–Computer Interaction* **5**, 145–90.

Markopoulos, P., Johnson, P., & Rowson, J. (1997), Formal Aspects of Task-based Design, *in* M. Harrison & J. Torres (eds.), *Design, Specification and Verification of Interactive Systems'97*, Springer-Verlag, pp.209–24.

Roth, S. & Mattis, J. (1990), Data Characterization for Intelligent Graphics Presentation, *in* J. C. Chew & J. Whiteside (eds.), *Proceedings of CHI'90: Human Factors in Computing Systems*, ACM Press, pp.193–200.

Salber, D. & Abowd, G. (1999), The Context Toolkit, *in* M. G. Williams, M. W. Altom, K. Ehrlich & W. Newman (eds.), *Proceedings of CHI'99: Human Factors in Computing Systems*, ACM Press, pp.434–41.

Scapin, D. & Pierret-Golbreich, C. (1990), Towards a Method for Task Description: MAD, *in* L. Berlinguet & D. Berthelette (eds.), *Proceedings of Work with Display Unit '89*, Elsevier Science, pp.371–80.

Sutcliffe, A. (1997), "Task-Related Information Analysis", *International Journal of Human–Computer Studies* **47**(2), 223–57.

Thevenin, D. (1998), *Interfaces Homme — Machine Autoconfigurables*, DEA Informatique, UJF.

Vanderdonckt, J. (1995), Knowledge-Based Systems for Automated User Interface Generation; The TRIDENT Experience, Technical Report RP-95-010, Fac. Univ. de N-D de la Paix, Inst. d'Informatique, Namur.

Human–Computer Interaction — INTERACT'99
Angela Sasse and Chris Johnson (Editors)
Published by IOS Press, © IFIP TC.13, 1999

Author Index

Human–Computer Interaction — INTERACT'99
Angela Sasse and Chris Johnson (Editors)
Published by IOS Press, © IFIP TC.13, 1999

Keyword Index