Looking for the Right Toolkit at the Right Time

Fabio Paternò
CNR-ISTI, HIIS Laboratory
Via Moruzzi 1, 56124 Pisa, Italy
fabio.paterno@isti.cnr.it

ABSTRACT
I have been working for several years in the area of tools for supporting design, development, and evaluation of interactive systems. In this position paper following the suggested format, I report on my personal view on the workshop topics based on such experience.

Author Keywords
End user development, cross-device user interfaces, model-based design.

ACM Classification Keywords
H.5.2 [Information Interfaces and Presentation]: User Interfaces - Interaction styles.

THREE CHALLENGES/OPPORTUINITIES

End User Development

The design and development of flexible software able to match the many possible users' needs is still a difficult challenge. It is almost impossible to identify all the requirements at design time and, in addition, such requirements are not static since user needs are likely to change/evolve over time, and designers have to consider the wide variability of the possible contexts of use. Indeed, the explosion of mobile technologies has made it possible for people to access their applications from a variety of contexts of use that differ in terms of available devices, things, and services, and that require specific actions when various types of events occur. Thus, it is not possible to guarantee a complete fit between the initially designed system and actual needs at any given time. As a result, it is important to design interactive software through methods and tools capable of dynamically and quickly responding to new requirements without spending vast amounts of resources, and which are able to consider that boundaries between design-time and run-time have become more and more blurred. Achieving this can increase the impact of software development companies since their applications can more easily penetrate many markets thanks to their ability to be customized directly by domain experts.

End-User Development (EUD) [1] approaches can help to solve such issues by enabling the possibility of customizing software applications by domain experts that have not experience in programming. Users' backgrounds can vary from management, engineering, construction, education, research, health, insurance, sales, administration or other areas. On the one hand, such users share a common requirement for software to support their common tasks, which may vary rapidly, and some of them cannot even be anticipated at design time, but discovered only during actual use. On the other hand, current slow software development cycles and the lack of domain knowledge on the part of software developers are limitations to addressing the requirements of the different users. Thus, EUD can reduce time and costs needed for customizations and increase their quality by avoiding potential misunderstandings between final users and developers.

Cross-device User Interfaces

The increasing availability of various types of devices in our daily life is often a missed opportunity since current applications are limited in supporting seamless task performance across them. Users often perceive device fragmentation around them rather than an ecosystem of devices that supports their activities. In order to address such issues a number of frameworks, platforms, and authoring environments have been proposed, mainly in research environment. The goal is to facilitate design and development of multi-device user interfaces. Responsive design is not enough to address such issues since its basic assumption is that the user at a given time interacts with only one device, such device may vary over time, and thus it provides the possibility of easily modifying the presentation mainly according to the screen size. In cross-device design such assumption no longer holds, and the goal is to allow developers to create applications that allow users to interact with many devices at the same time, and the various parts of the user interfaces distributed across the different devices can keep their state synchronized.We can distinguish various types of multi-device user interfaces depending on the features that they support: migratory user interfaces are able to dynamically migrate from one device to another in order to follow users' movements while preserving their state; distributed user interfaces allow users to interact with an application through multiple devices at the same time; cross-device user interfaces are distributed user interfaces, with the additional capability to synchronise their state, so that the interactions through some element in one device update the state of the corresponding elements (if any) in another device. Such categories are not mutually exclusive, so for example it is possible to have user interfaces that are both migratory and cross-device.

Model-based User Interfaces

Model-based approaches [2] have been proposed to manage the increasing complexity derived from user interfaces in multi-device environments. They have also been considered in W3C for standardization in order to ease their industrial adoption. The main idea is to provide a small general conceptual vocabulary to support user interface developers in the design process. The resulting logical descriptions can then be transferred into a variety of implementation languages with the support of automatic transformations, thereby hiding the complexity deriving from the heterogeneous languages and devices, and sparing developers the need to learn all the details of such implementation languages. Thus, they can be useful to obtain more accessible applications as well since they make more explicit the semantics and the role of the various user interface elements, which is also important for access through assistive technologies.

One important contribution in the area of intelligent user interfaces was Supple [3], which provided a tool able to consider aspects related to the user and the device at hand for generating a personalized version of the interactive application. However, the combined explosion of mobile technologies and Internet of Things has posed new challenges to address since there is a variety of contextual aspects to consider when generating interactive applications. More recently, a generative approach with semantic interaction descriptions for obtaining user interfaces for smart things was proposed [4] but again it assumed some previous knowledge of the smart things to address, and thus lacked the ability to support customization for dynamic contexts of use.

Other authors have more focused on how the use of model-based UI development approaches can improve the experience of users interacting with context-dependent, multi-platform applications. Recent contributions [5] presented a UI development framework for ambient applications integrated with a user modelling system, in order to provide usability predictions during early development stages.

One issue in model-based techniques is that because heuristics are often involved in the generation process, the connection between specification and final result can be problematic to understand and control. Programmers must also learn a new language for specifying the models, which raises the threshold of their use. However, the importance of such approach is demonstrated by its adoption, to some extent, by a widely used languages as HTML 5. Indeed, one of the main HTML 5 features is the introduction of more semantics tags, which better express the purpose of the possible user interface elements. On the other hand, HTML 5 is still limited with respect to the potentialities of model-based approaches since it mainly considers only graphical user interfaces, while languages such as MARIA [6] can be used also for generating multimodal user interfaces [7] as well.

THREE SUCCESSFUL TOOLKITS

The Context Toolkit

The Context Toolkit [8] aimed at facilitating the development and deployment of context-aware applications. The Context Toolkit was among the earliest supports for developing context-enabled applications by providing a library to facilitate integration with sensors. It was one of the first attempts to support developers in creating applications able to consider not only the events that occur on the screen but also those that are generated in the surrounding environment.

The authors defined context as environmental information that is part of an application's operating environment and that can be sensed by the application. It consisted of context widgets and a distributed infrastructure that hosted the widgets. Context widgets are software components that provide applications with access to context information while hiding the details of context sensing. In the same way GUI widgets insulate applications from some presentation concerns, context widgets insulate applications from context acquisition concerns. To summarize, the main features of the Context Toolkit were: encapsulation of sensors; access to context data through a network API; abstraction of context data through interpreters; sharing of context data through a distributed infrastructure; storage of context data, including history; basic access control for privacy protection.

It initially considered a limited set of events and led to meld the context awareness code with the application. More recently, the Context Toolkit has been augmented with support to facilitate development and debugging of context-dependent applications [9]. Programming abstractions, called Situations, expose an API supporting both developers and designers to provide application adaptivity without coping with low-level code.

Scratch

Scratch [10] is a visual programming language developed by the MIT Media Lab, which targets students, scholars, and teachers parents to easily create animations, games, and similar applications. Thus, it represents a useful tool for a range of educational and entertainment purposes., and it can be a way for introducing to the more advanced world of computer programming. Scratch represents an example of tool for end user development environment based on the jigsaw puzzle metaphor, in particular in creating interactive applications with multimedia content. In this metaphor, each software components is seen as a piece of a puzzle and the shapes of the various pieces provide the cognitive hints needed to understand the possible compositions. As such, non-expert users can easily associate each puzzle piece with the component it represents. While this metaphor supports

more complex configurations than the ones supported by the pipeline metaphor, one disadvantage is that it has constrained capability for limited expressiveness. Indeed, the pieces of the puzzle have a limited number of interfaces (i.e. sides), thereby restricting the set of possible programming expressions. AppInventor [11] has then exploited such metaphor to support the development of functionalities triggered by events on an app user interface. While in Scratch and AppInventor the puzzles pieces are used to represent low-level programming constructs, a different approach has been proposed in Puzzle [12], in which it has been adopted to support development of Internet of Things applications on smartphones:, and the elements are associated with high-level functionalities that can also control various actuators. Thus, Puzzle has been designed to facilitate the composition of various pieces through a touch interface for a screen with limited size. Each possible puzzle piece represents a high-level functionality that can be composed, and its shape and colours indicate the number of inputs and outputs and the information type of information that they can communicate. Thus, the tool provides a usable solution but is limited to the composition of functionalities for which a puzzle piece has been provided.

Supple

One important contribution in the area of intelligent user interfaces was Supple [3], which provided a tool able to consider aspects related to the user and the device at hand for generating a personalized version of the interactive application.

Supple uses decision-theoretic optimization to automatically generate user interfaces adapted to a person's abilities, devices, preferences, and tasks. In particular, SUPPLE can generate user interfaces for people with motor and vision impairments and the results of our laboratory experiments show that these automatically generated, ability-based user interfaces significantly improve speed, accuracy and satisfaction of users with motor impairments compared to manufacturers' default interfaces. It takes a functional specification of the interface, the device-specific constraints, a typical usage trace, and a cost function. The cost function is based on user preferences and expected speed of operation. SUPPLE's optimization algorithm finds the user interface, which minimizes the cost function while also satisfying all device constraints.

The SUPPLE authors then focused on how to exploit SUPPLE in order to support disabled users, for example, by automatically generating user interfaces for a user with impaired dexterity based on a model of her actual motor abilities. More generally, we can consider adaptation useful for both permanent and temporary disabilities. An example of temporary disability is when the user has to move fast and interact with a graphical mobile device. Thus, the user's visual attention cannot be completely allocated to the interaction.

BRIEF OVERVIEW OF MY PAST WORK

The goal of my research work has been to bring human values such as usability and accessibility in the design and development of interactive technologies.

At the beginning, my main research area was model-based design of interactive applications. I developed the ConcurTaskTrees notation for specifying task models and also designed an associated environment (CTTE) to support the development and analysis of task models specified through this notation, which has been widely used in various industries and universities in various parts of the world (with 26000 downloads). The tool has been applied in several application domains including ERP, interactive safety-critical systems (for example in air traffic control[1]), and the language has been the input for a W3C standard in the area (http://www.w3.org/TR/task-models/). I also worked on the design of the MultiModal TERESA and MARIA languages and the associated tools, whose main purpose is to support designers of multi-device, multi-modal interactive applications starting with user interface logical descriptions.

A considerable amount of work has also been dedicated to mobile guides. I designed the first museum mobile guide in Italy for the Marble Museum. Then, various solutions for this type of guide have been further designed, some in collaboration with local museums, which exploit various technologies for location awareness, collaboration and multimodal interaction.

From this type of experience I have broaden my interests to ubiquitous interactive systems. In particular, to address issues related to multi-device environments by proposing original solutions for migratory and cross-device user interfaces, which allow seamless access through a variety of devices ranging from wearable to large public displays, and dynamic allocation of interactive components across them. I also edited and wrote part of a book on Migratory Interactive Applications for Ubiquitous Environments published by Springer Verlag.

In parallel, another research topic in which I have played a pioneering role is end-user development, in which area I coordinated a European Network of Excellence (EUD-net). I also co-edited (together with Henry Lieberman from MIT, and Volker Wulf from University of Siegen) one of the best-known books on End User Development (widely cited), and carried out various research studies in the area. In this area I have actively worked at the design of various authoring environments and tools, such as Puzzle for intuitively editing interactive applications from a smartphone, or a mashup tool for creating new Web applications by composing existing components using the

[1] https://www.eurocontrol.int/ehp/?q=node/1617

familiar copy-paste interaction across them. More recently, I have focused on how to personalize context-dependent applications through trigger-action rules [13].

SUGGESTIONS FOR TOPICS FOR DISCUSSION

I see room for discussion at the workshop from different perspectives. One perspective is to consider the main technological trends and how they impact on the design of user interface toolkits. For example, we are in the Internet of Things time, which means more and more surrounded by various types of sensors, objects, devices and services that can be associated dynamically to meet user's expectations. Are current toolkits able to sufficiently support the design and development of applications for such environments ?

Another perspective is to analyse trends and solutions in this area through the threshold/ceiling criteria (the "threshold" is how difficult it is to learn how to use the toolkit, and the "ceiling" is how much can be done using the toolkit) [14]. Various toolkits have been criticized because they have high threshold and low ceiling, which means they require considerable effort for learning how to use them, and then in the end they support limited functionalities. What approaches should be considered to invert this situation and obtain low threshold / high ceiling solutions ?

One further perspective is to focus on the main attributes that user interface toolkits should have, and thus they can be useful to design an evaluation framework for toolkits. Such attributes are usually important both for the toolkits and the applications that they allow developers to obtain. Example of attributes that seem particularly relevant are: coverage, consistency, interoperability, usability, customizability, extensibility, and scalability.

REFERENCES

1. H. Liebermann, F. Paternò, V. Wulf. "End-User Development", Springer, Dordrecht, 2006
2. J.Fonseca (Ed.) (2010) Model-Based UI XG Final Report, W3C Incubator Group Report 2010.
3. Krzysztof Gajos and Daniel S. Weld. SUPPLE: automatically generating user interfaces. In IUI '04: Proceedings of the 9th international conference on Intelligent user interface, pages 93-100, New York, NY, USA, 2004. ACM Press.
4. Simon Mayer, Andreas Tschofen, Anind K. Dey, and Friedemann Mattern: User interfaces for smart things--A generative approach with semantic interaction descriptions, ACM Transactions on Computer-Human Interaction (TOCHI) 21 (2), 12, 2014.
5. Marc Halbrügge, Michael Quade, Klaus-Peter Engelbrecht, Sebastian Möller, Sahin Albayrak, Predicting user error for ambient systems by integrating model-based UI development and cognitive modelling Proceedings Ubicomp'16, pp.1028-1039, ACM Press
6. F. Paternò, C. Santoro, L.D. Spano, "MARIA: A Universal Language for Service-Oriented Applications in Ubiquitous Environment", ACM Transactions on Computer-Human Interaction, Vol.16, N.4, November 2009, pp.19:1-19:30, ACM Press,
7. M Manca, F Paternò, C Santoro, LD Spano. Generation of multi-device adaptive multimodal web applications. International Conference on Mobile Web and Information Systems, 218-232
8. D. Salber, A. K. Dey, and G. D. Abowd: The Context Toolkit: Aiding the Development of Context-Enabled Applications. CHI 1999: 434-441
9. A. K. Dey, and A. Newberger: Support for context-aware intelligibility and control. CHI 2009: 859-868
10. M. Resnick, J. Maloney, A. Monroy-Hernandez et al., "Scratch: programming for all," Communications of the ACM, vol. 52, no.11, pp. 60–67, 2009.
11. App Inventor MIT, 2012, http://appinventor.mit.edu/.
12. Danado J., Paternò F., Puzzle: Puzzle: A Mobile Application Development Environment using a Jigsaw Metaphor, Journal of Visual Languages and Computing, 25(4), pp.297-315, 2014.
13. Giuseppe Ghiani, Marco Manca, Fabio Paternò, Carmen Santoro: Personalization of Context-dependent Applications through Trigger-Action Rules. ACM Transactions on Computer-Human Interaction (ACM TOCHI), 2017.
14. B. A. Myers, S. E. Hudson and R. Pausch: Past, Present and Future of User Interface Software Tools, ACM Transactions on Computer Human Interaction. March, 2000. 7(1). pp. 3-28.