Hindawi Publishing Corporation
ISRN Software Engineering
Volume 2013, Article ID 532659, 11 pages
http://dx.doi.org/10.1155/2013/532659

Review Article

End User Development: Survey of an Emerging Field for Empowering People

Fabio Paternò

CNR-ISTI, HIIS Laboratory, Via Moruzzi 1, 56124 Pisa, Italy

Correspondence should be addressed to Fabio Paternò; fabio.paterno@isti.cnr.it

Received 25 February 2013; Accepted 26 April 2013

Academic Editors: C. Calero, K. Framling, O. Greevy, A. Lastovetsky, and C. Rolland

The purpose of this paper is to introduce the motivations behind end user development, discuss its basic concepts and roots, and review the current state of art. Various approaches are discussed and classified in terms of their main features and the technologies and platforms for which they have been developed. Lastly, the paper provides an indication of interesting possibilities for further evolution.

1. Introduction

One important trend in software technology is that more and more interactive applications are being written not by professional software developers but by people with expertise in other domains working towards goals supported by computation. Statistics from the US Bureau of Labor and Statistics predicted that by 2012 in the United States, there would be fewer than 3 million professional programmers but more than 55 million people using spreadsheets and databases at work and many writing formulas and queries to support their jobs [1]. More recently, a July 2011 Gartner report indicated that nonprofessional developers will build at least 25 percent of new business applications by 2014. Computer programming, almost as much as computer use, is becoming a widespread, pervasive practice. Such trends were already identified some years ago [2] and are becoming more and more evident.

End-User Development (EUD) can be defined as a set of methods, techniques, and tools that allow users of software systems, who are acting as non-professional software developers, at some point to create, modify or extend a software artefact [3]. End users have specific goals in their own domains, which are not related to software development. The users that we consider here are people who have some basic technological knowledge but are not professional programmers.

There are various motivations for EUD: professional developers lack the domain knowledge that end users cannot easily convey when communicating requirements for a new application, and regular development cycles are too slow to meet the users' fast changing requirements. However, since end users usually lack the training of professional software developers, it is simply not possible to use the traditional development approaches for EUD.

Currently available applications only realize a fraction of EUD's potential and still suffer from several flaws, limiting EUD's important contribution to empowering users as active citizens of the information society. The success of the Web 2.0 is a clear indication of how people would like to be more active and creative in the information society. However, Web 2.0 is mainly limited to user-generated content, while people often also would like to change the behaviour, functionality and accessibility of their applications.

Topics related to EUD have already been investigated to some extent in recent years. For example, [4] provided an interesting overview of the state of art in end-user software engineering. They provided an analysis involving phases in software engineering, such as debugging [5] and testing, which are not considered in this paper since they have not stimulated particular interest from an EUD viewpoint. On the other hand, their work did not properly address recent developments in EUD, such as EUD for the Web and mobile applications, which are extensively dealt with in this paper because such new developments have to consider novel aspects. For example, in recent years we have witnessed an increasing number of applications developed through mobile

devices. This involves specific requirements for developing applications. Indeed, early EUD approaches mainly focused on static desktop applications. However, more and more people use a variety of devices (desktop, tablet, smartphones, etc.) to access and manipulate their applications in evermore varying environments, and this raises new problems for end-user development that are discussed in the following.

In general, 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 is a need for environments supporting transformations from intuitive but sometimes ambiguous representations into more precise, but more difficult to understand, specifications. Model-based approaches to the design of interactive systems [6–10] are used to support the development through the use of meaningful abstractions to avoid dealing with low-level details. Despite such potential benefits, their adoption has mainly been limited to professional software designers and developers. While domain experts are often familiar with modelling techniques in their work, they often have difficulties in manipulating models of interactive software applications. Therefore, new solutions are necessary which enhance the nature of abstract models, representations, and computational languages.

This paper is organised as follows. It first provides a short indication of the early work in the area, then discusses key concepts that characterise EUD and provide concrete examples of how they have been addressed. Then, some of the currently important emerging approaches involving the Web and ubiquitous applications are described and discussed, and lastly some conclusions and indications for possible further developments are provided.

2. Early Work

With the advent of mass market personal computing with graphical user interfaces and tools for personal productivity, there was soon a need to create environments that allowed people without a programming experience to create their own applications. Various approaches were exploited for this purpose. HyperCard was an environment for Apple Macintosh that was the first successful hypermedia system before the emergence of the World Wide Web. It combined database features in "cards" that supported clickable regions that could link to another card, or execute some functions. This combination of features—simple form layout, database capabilities and ease of programming—led to widespread use in many different roles. *Programming-by-demonstration* (PBD), sometimes called *programming-by-example*, is another programming technique whereby the user creates an application by performing the steps as an example. Thus, the user demonstrates an example of what the program should do, from which the programming environment infers a more general application supporting the desired behaviour. One of the first contributions in this area was Peridot [11], which allowed the user interface designer to draw a picture of what the user interface should look like and to interact with it. While this was happening, Peridot created a code for the interface and its connections to actual application programs. The code produced was not simply a transcript of the designer's actions, but parameterized procedures, in which parts of the interface could depend on the parameter values. To this end, Peridot used some simple artificial intelligence techniques to infer how the graphics and mouse should change based on the actual values of the parameters. Some PBD systems are able to deductively infer the entire program, while others deduce what they can, and ask the user for help for the rest [12]. Eager [13] supported PBD by detecting looping behaviours on HyperCard; this was obtained by searching through a person's interaction history for events similar to the current action. PBD-based tools have been considered for creating animations and many other kinds of programs [14, 15]. One problem with PBD has been representing the final program in a form useful to the users to enable them to review, test, and debug the program. Thus, PBD is often used in combination with visual or textual languages.

Fischer and Girgensohn [16] introduced the concept of *end-user modifiability in design environments* in order to ease the construction and modification of artefacts designed within the environment, thus making it possible to modify systems in order to allow them to support tasks that the designer did not foresee. This concept was then generalised into the concept of *meta-design* aiming to allow users to act as designers and be creative, thus putting in their hands the tools rather than the object of the design. This approach can be useful to obtain environments that are flexible and able to evolve because they cannot be completely designed prior to use.

Customization techniques [17–19] soon raised a good deal of interest with the aim of helping users obtain application versions more suitable for their tasks. Mørch [20] identified three levels of tailoring: customizing existing functionality; integrating functionality available elsewhere, and extending a system with new functionality created by end users. More recently, such tailoring techniques have been investigated in the context of component-based applications [21].

One approach that has received a considerable amount of attention has been *visual programming*. The idea is to exploit two-dimensional graphical representations to facilitate development. In general, this trend has been successful to simplify development though this has not always resulted in novice users being able to understand how to develop their own applications. In visual programming, dataflow visual language is one paradigm that has been often applied: its basic idea is to associate icons to high-level functionalities that are important for the specific domain experts. Usually such icons have some output and input ports representing the input and the output data, and the development of the application then consists in interactively composing instances of such icons by indicating where they receive input from and where they send the results of their processing. One of the first environments applying such paradigm was Cantata [22] a visual language for Khoros, an environment that was used mainly for image processing applications. Another similar

example was reported in [23], which considered an environment to specify, analyse, and execute visual programs for geographical information systems (GIS) in order to support professionals without a strong programming background who wanted to access GIS. The adoption of a visual language approach was useful in order to hide the plethora of basic GIS functions, while providing ready-to-use tools to solve users' tasks. This visual environment provided users with higher level interfaces: it was based on the module concept, which was conceived as a software building block that implements a solution to a general basic task and is presented to the user through an interactive frame. Complex GIS queries could be carried out by interconnecting modules into flow networks, using a direct manipulation approach. More recently various tools have provided graphical representations for designing dataflow connections between components, for example, Yahoo! Pipes (http://pipes.yahoo.com/), which provides a composition tool to aggregate, manipulate, and mashup content from around the Web. Like Unix pipes, simple commands can be combined together to create an output that meets user needs, such as combining many feeds into one, then sort, filter, and translate it. More generally, visual programming languages soon raised a number of issues related to their scalability [24].

Spreadsheets were amongst the first major EUD programming environments made possible by these innovations [25], beginning with VisiCalc, then continuing with Lotus 1-2-3 and Excel. A number of environments have adapted the successful spreadsheet style of end-user programming to other domains (e.g., [26, 27]).

3. Key Concepts

Generally speaking, effective design tools should make it easy for novices to get started (*low threshold*) but also possible for experts to work on increasingly sophisticated projects (*high ceiling*) [28]. The low threshold means that the interface should not be intimidating and should give users immediate confidence that they can succeed. The high ceiling means that the tools are powerful enough to create sophisticated, complete solutions. Too often tools that enable development of interactive applications may be quite hard to learn (they do not have a low threshold). Instead, they focus on providing numerous powerful features so that experts can assemble results quickly. One main goal in EUD is to decrease the conflict between application complexity and learning effort [29]. In general, complex programming languages can address a wide range of problems but thereby also increasing the learning burden on the user. On the other hand, easy-to-learn languages are often domain specific and limit the development possibilities. The EUD aim is to allow for a gentle learning curve, which means to provide environments whereby each incremental increase in the level of customizability only requires the user to devote an incremental amount of effort. This contrasts with most systems, which present "walls" where the user must stop and learn many new concepts and techniques to make further progress. Thus, novel metaphors are used to reduce the learning burden by reducing the conceptual distance between actions in the real world and the more abstract concepts presented in programming languages. In general, such graphical metaphors are closer to real life and thus motivate users to learn and use it in daily work practices [29]. In order to allow users to provide for complex behaviours, a gentle learning curve is supported through different layers of abstraction that can be created.

The increasing popularity of Web 2.0 technologies characterised by environments in which users can directly provide and manipulate the application content shows how people without programming background like to have more control on their applications. Furthermore, there are several other programming solutions for specific problems in leisure time, as demonstrated by tools such as semantic Web browsers [30] and process tools [31]. However, these graphic tools have a limited range of expression. More powerful EUD tools rely on scripting languages, but these present users with a considerable learning burden. Furthermore, macros, formulae, and scripts are prone to errors [32].

Over time, Fischer has elaborated and refined the concepts of DODEs (Domain-Oriented Design Environments), which provided a graphical design language and a set of design patterns, with critic and advisor experts to guide the design process [33, 34]. However, first DODEs required considerable configuration or seeding with knowledge before development could proceed and they were limited to specific domains. AgentSheets was developed as a more general EUD technology to implement DODEs [35–37]. AgentSheets provides an agent-oriented development environment with rule-based templates for specifying agent behaviour and a toolkit for creating graphical worlds in which agents interact. It also uses a spreadsheet metaphor. In these systems, program objects occupy cells in a grid and interact with the objects in neighbouring cells. AgentSheets has been used to develop several different types of applications, including mobile information systems. However, the agent rule-based specification style does not easily scale to more complex procedural programming in business domains. A similar agent rule-based development environment specifically aimed at children as end users, developed by Pane and Myers et al. [38, 39], used rule-based specification templates coupled with explanation facilities and limited programming by demonstration. The programming-by-example tools researched by Lieberman [40] were able to infer more complex user intentions from demonstrated actions in the graphical worlds. However, Lieberman concluded that mixed initiative EUD, combining demonstration with explicit user-driven instruction, was the best way forward. Lieberman's "Goal Oriented Web Browser" [30] is an example of such a PDB system. Another approach is called sloppy programming and is based on interpretation of pseudonatural language instructions, instead of formal syntactic statements [31], thus increasing adaptability of code for nonprogrammers.

Often EUD approaches support users in composing and customizing sets of available basic elements developed by programmers. Such basic elements are represented by and composed through intuitive metaphors, such as the jigsaw in which the basic elements correspond to

the pieces to compose or iconic data flow representations in which the icons correspond to the basic elements. Google App Inventor [41] expresses the process of building applications in a way similar to Scratch [42], by which traditional programming is performed by combining jigsaw puzzle pieces. This metaphor was also used in "Playing with the Bits" [43], where users snap available components together using a jigsaw puzzle metaphor.

Collaboration is a key feature when it comes to the end-user creation of programs because designing solutions for complex problems often requires more knowledge than a single person possesses, leading to social creativity. This makes it important for groups of end-user programmers to have suitable tools to support their collaborative programming tasks, such as those researched in the field of End-User Development. Open source communities are an important reference point in terms of collaboration processes for the design of new EUD environments, even if open source is usually oriented to expert programmers. In addition, in comparison with groups of professionals, end-user groups are very heterogeneous as members do not have the same technical background and sometimes even their social backgrounds are different. The users that EUD targets are non-professional developers: people, who usually are domain experts, have some basic technological knowledge but are not professional programmers.

Mutual development [44], *co-development* [45–47], and *participatory design* [48] refer to activities in which end users are involved in system design but may or may not be involved in its actual coding. For example, the use of storyboards, scenarios, and interactive prototyping are accepted approaches for encouraging participatory design [49]. In a study of user engagement over seven years, Letondal and Mackay [50] used a workshop-based approach to foster collaboration among biologists, bioinformaticians, and computer scientists. There is a need for novel approaches to make a participation that encourages finding a common ground in participatory development of end-user tools as well as collaborative development of applications.

End-user development can benefit from using multiple representations with various levels of formality. At the beginning of the design process, many ideas are unclear, so it is hard to develop precise specifications. The main issue of end-user development is how to exploit personal intuition, familiar metaphors, and concepts to facilitate design exploration, and subsequently specification and/or modification of software artefacts. Examples of informal representations to start to express what the interactive application should do are textual descriptions [51] and graphical sketches [52, 53]. For example, nonprogramming users feel comfortable with sketch-based systems that allow them to concentrate on main concepts by exploiting natural interactions allowing them to easily modify the produced descriptions, instead of being distracted by cumbersome low-level details required by rigid formal languages. Such systems must be able to recognise graphical elements in the sketches and convert them into formats that can be edited and analysed to generate computational instructions or infer them from PBE interaction.

Regarding the use of *natural language descriptions in connection with End-User Development*, Tam et al. [51] described a system for the elicitation of user-task models. In particular, their methodology allows for a domain expert to complete the elicitation process by providing textual descriptions of examples of specific interaction scenarios. Interacting with the expert, the system then identifies the corresponding tasks, domain objects, and actors. More recently, Liu and Lieberman [54] introduce the notion of "Programmatic Semantics" to represent the mapping between NL structures and basic programming language structures. For example, noun phrases are interpreted as data structures, verbs as functions, and adjectives as properties. A similar task is addressed in [55], illustrating an approach for translating keyboard commands (such as "click search button" or "Arial 10 point font") into an executable code. As a general comment, in their attempt to better understand NL commands, all developments reported so far impose severe limitations on the expressive power of the communication process by forcing users to communicate through a "controlled" language, and thereby reducing the amount of linguistic analysis, usually limited to simple steps of information extraction from the text. The iPhone Siri provides a small example of what can be done in this direction by using a natural language user interface to answer questions, make recommendations, and perform actions by delegating requests to a set of Web services.

Pane and others [38] followed a different approach in order to make it easier the development of interactive applications. They started with general principles and heuristics developed in the field of HCI that can be applied to help address such issues, such as Nielsen's heuristic evaluation including recommendation such as be consistent, keep it simple, speak the user's language, prevent errors, help the user get started, and so forth. The cognitive dimensions framework [56, 57] lists additional criteria that can be used to evaluate design alternatives in programming systems, such as closeness of mapping, viscosity, hidden dependencies, imposed guess-ahead and visibility, and so forth. Pane and others conducted a pair of studies to examine the language and structure that children and adults used before they have been exposed to programming. In these studies, they presented programming tasks to nonprogrammers, who then had to solve them on paper. The tasks covered a broad set of essential programming techniques and concepts, such as control structures, storage and data manipulation. Some of the features observed in these studies were that an event-based or rule-based structure was used, where actions were taken in response to events; aggregate operators (acting on a set of objects all at once) were used much more often than iterative operators (which act on each object in the set individually); datastructures were avoided by using content-based queries. Then, they developed a new environment (HANDS) based on such results that represents the computation as a character sitting at a table, manipulating cards that hold the program data. This model substitutes familiar ideas for the common but completely unfamiliar von Neumann machine model

of computation. The HANDS language provides operators that closely match those observed in the studies performed. It uses an event-based style of programming and provides queries and aggregate operators to allow more concise high-level expressions for tasks that require the assembly of many primitives in other languages.

4. EUD for Web Applications

The Web is the most common user interface and can beaccessed through any type of device. The Document Object Model (http://www.w3.org/DOM/) provides a common interface to interactively manipulate Web applications, thus enabling the possibility of building interactive tools able to exploit such features. An interesting point is that such tools allow end users to manipulate Web applications without involving the original developers. Consequently, while first EUD tools mainly focused on desktop graphical applications, in recent years a considerable amount of work has been carried out to apply the EUD approach to Web environments.

Scaffidi investigated the use of scenario-based requirements [58] and proposed novel techniques for data validation [59]. A simple proposal was EzWeb Enterprise Mashup [60], a platform that allows the composition of a set of gadgets for service developers. The resulting environment is still oriented to professional developers. The programming-by-example approach has been implemented in Web environments through different mechanisms. Nichols and Lau [61] describe a system that allows users to create a mobile version of a Website through a combination of navigating through the desired portion of the site and explicitly selecting content. Macías and Paternò [62] take a similar approach, in which users directly modify the Web page source code. These modifications are used as a specification of preferences, which are then generalized and applied to other pages on the same site through the support of model-based user interface description languages. Toomim et al. [63] allow users to select example data from Websites and automatically generate a range of user interface enhancements. Lin and others [64] have proposed a system (Vegemite) using direct manipulation and programming-by-demonstration techniques to automatically populate tables with information collected from various Websites. They have addressed a class of ad hoc mashups, where users want to quickly combine data from multiple Websites in an incremental, exploratory fashion, often in support of a one-time or infrequently performed task. Their tool allows constructing a data table from anywhere and then running scripts that add columns to that table based on the data in each row.

Chickenfoot [65] is an extension of Mozilla Firefox, which allows users to modify Web pages without knowing HTML and using a tool (the browser) familiar to most. Its main characteristics are the following: it is directly executed in the browser so that the users can immediately realise what they are editing because they immediately see the result; it uses a syntax based on words that should be widely known by users, such as "click" and "enter"; it allows users to describe the components through a small set of intuitive commands (e.g., click, enter, pick, keypress, and go). Instead, CoScripter [66] is a system that allows users to record, share, and automate tasks to perform in the Web and provides a repository where the scripts created are shared. It was inspired by Chickenfoot, which enabled end users to customize Web pages by writing simplified JavaScript commands that used keyword pattern matching to identify Web page components. CoScripter uses similar heuristics to label targets on Web pages, but it uses natural language representation for scripts that require less effort than Chickenfoot's JavaScript—based language. In CoScripter scripts are recorded as natural language scripts that can be modified by the user without having to understand a programming language. In detail, it consists of two main parts: a centralised repository of scripts and a Firefox extension that facilitates creating and running a script. The two work together: users access the repository to select a script, which can then be executed in the extension either step by step or automatically to completion. ActionShot [67] was a successive tool consisting in an extension to the Firefox Web browser built on top of the CoScripter Web recording/playback platform in order to automatically detect repetition in Web usage logs and enable retroactive authoring by allowing users to manually search for, extract, edit, and rerun previous actions. For this purpose, ActionShot provides interfaces to facilitate browsing and searching through this history, sharing portions of the history through established social networking tools such as Facebook, and creating scripts that can be used to repeat previous interactions at a later time.

The existence of a tremendous amount of Web content, which is not always in a form that supports end users' needs, has motivated Marmite [68]. This tool allows users to select some operators, place them in a data flow representation, and view the current state of the data corresponding to a particular operator in a table, which shows what the data looks like after it has passed through the operator. However, Marmite is able to manage only a small set of pre-defined data types. In addition, it has been implemented as a Firefox plug-in, thus limiting its applicability to this browser only. A different tool allowing users to combine interactive components from different Web sites and build new ones as well is presented in [69]. This solution is browser independent thanks to the use of an intermediate proxy/mashup server, and enables people without programming knowledge to create mashups composed of Web components selected directly from existing Web applications by establishing communication among components originally belonging to different applications.

Recently, one trend to encapsulate there has been functionalities in Web services that can be accessed through the Internet through their operations and parameters. In the area of EUD for service-based interactive applications, d.mix [70] supports the development of applications based on Web services through a site-to-service map: the user can navigate annotated Web sites and select relevant elements. Through a site-to-service map, d.mix generates the code including the Web service calls that yield results corresponding to the user's selection. Such-code can then be modified through a wiki. The platform also makes available some examples that can be further edited, also in this case some programming knowledge is required to be able to exploit the tool features. Another

approach to such issues has been proposed by Nestler et al. [71] through a tool for rapid development of composite applications using annotated Web services starting with the WSDL service operation descriptions and exploiting Web services annotations providing suggestions for user interface development, when available. However, this tool is limited to create simple form-based user interfaces on desktop systems and still requires some familiarity with Web services technology. The authors identified some general guidelines to consider when supporting EUD for this type of application: hide programming code and technical details from the users; use abstraction layers, visual representations, and metaphors to facilitate and realize the WYSIWYG approach; concentrate on the most important aspects that require knowledge or input from the user when modeling the logic of the desired application; implement common UI guidelines to produce service-based applications of high usability.

5. EUD for Mobile Applications

Recent years have witnessed the rapid growth of the use of mobile devices to access interactive applications. Limited work has been dedicated to EUD for mobile applications. Previous approaches for desktop applications cannot be simply reproposed as they are, given the specific characteristics of mobile devices: they are becoming ever richer in terms of sensors, such as accelerometers, GPS, and that can be exploited during the interactions, and the limited screen size in which applications are accessed requires a careful and specific design for presenting content and interaction elements in order to avoid usability problems such as tedious activities in zooming in and out for viewing the desired piece of information or touch/based interactions that select the wrong elements.

The first EUD environments to create applications for mobile devices have mainly targeted desktop environments: they assume that people use the desktop for developing the application, which is then deployed in the mobile device, thus implying a rigid division between design time and run time. Examples of domains of desktop EUD environments targeting mobile applications have been tourism, museum guides [72, 73], and home applications [74, 75]. Akesson et al. [76] presented a user-oriented framework to ease the reconfiguration of ubiquitous domestic environments. The support, running on a tablet PC, adopts a paradigm based on jigsaws.

A visual strategy for developing context-aware applications was proposed in [77]. Such a system, called iCAP, allows end-users to design application prototypes by defining elements (objects, activities) and rules (associations between actions and situations). The rules are graphically edited through basic operations like dragging the defined elements onto rule sheets. Another framework to support people without programming experience is eBlocks [78]: it facilitates the creation of customized sensor-based systems and the configuration of condition tables.

Carmien and Fisher [79] describe a framework for customizing mobile applications to help people with cognitive disabilities. A graphic editor, intended to be used by the caretakers, facilitates the management of the task-support scripts for helping the disabled. The reported evaluation of the editing environment, called MAPS-DE, revealed that the caretakers appreciated the possibility of customizing the prompting system for the needs of individuals with specific disabilities. Hull et al. [80] provided a set of template applications for tourism. Ghiani et al. [72] have developed an environment that allows customization of mobile solutions for museum guides, performed mainly on desktop systems, and it also allows the generation of application versions for stationary systems with large screens. Floch [81] describes the initial design of a city guide that can be tailored by end users in order to include information from different service providers according to the visitor's position and visiting purpose.

Collapse-to-zoom [82] was a technique for viewing Web pages on small screen devices by interactively removing irrelevant content selected through end user gestures performed with a pen on the mobile device screen. Thus, it can be considered an approach to interactively customizing desktop Web applications for mobile access. A similar approach but extended with the possibility of preserving the client-side state of the application even when dynamically migrating it to a mobile device is presented in [83]. The basic idea is that users access a Web application through a desktop system in order to perform some interaction, and then, when they have to move, they can migrate the application to a mobile device in which they can continue their task from the point they left off. In addition, the users can interactively select the parts of the Web application they want to migrate, thus customising a mobile version on-the-fly. This is obtained through some scripts dynamically inserted in the original Web application by a migration server. The application state that can be migrated to the target device includes that of the interactive forms, cookies, Javascript variable and other aspects.

Contributions for mobile EUD have addressed aspects, such as parameterization of the mobile terminal [84], frameworks to support mobile authoring and execution [85], creation of UIs through sketching or by adding interactive techniques in the touch screen [86].

Desktop EUD environments lack the advantages of enabling end users to create applications opportunistically while on the move. Recent advances in smart phones in terms of connectivity, processing power, and interaction resources have enabled the creation of mobile EUD environments. Puzzle [87, 88] supports editing on a touch-based smartphone by using the jigsaw metaphor to convey the concepts of connecting high-level functionalities, and a solution, inspired by the work of Cuccurullo, Francese et al. [89, 90], using the colours to indicate the associated data types, thus providing intuitive cues to help the users to correctly connect the outputs and inputs of the jigsaw pieces.

Each interaction platform has specific features that determine its limitations and make it suitable to perform some tasks [91]. Indeed, even if the computational resources of mobile devices (e.g., smartphones) are growing, they are still more limited than those of desktop systems; on the

other hand, they have a number of sensors and features that desktop systems do not support. Watching a long video or making a flight reservation are typical examples of tasks more suitable for devices with large screens. On the other hand, location-based tasks such as showing the route from the current position to a hotel are more suitable for mobile devices. Thus, users do not use all devices in the same way and tend to assign different roles to devices both by choice and by necessity. A recent study [92] highlighted that most of consumers' time is spent in front of a variety of interactive devices, which can be used both sequentially (i.e., by moving from one device to another) and simultaneously (i.e., using more than one device at the same time). Thus, it can be useful to have authoring environments for multidevice applications. One proposal in this area was Damasks [93], which supports the use of sketches, design patterns [94], and layers to generate desktop, mobile, and vocal Web applications. The layers are used to indicate whether the various parts of the interactive application description should be supported by all the platforms or only by one specific platform. MARIAE [95] supports a larger set of platforms through the use of device-independent languages, it supports an abstract language and refinements for various target platforms (desktop, mobile, vocal, multimodal, etc.). However, such environments are generally still more appropriate for software professionals rather than endusers. Moreover, they have yet to consider multimodal interaction at development time and need to be better integrated with social support in order to allow the users to share comments, examples and suggestion. In addition, systematic empirical validation is necessary in order to understand the best solutions for the deployment of EUD environments.

6. Discussion

In general, various dimensions can be used to compare the various approaches to EUD for interactive applications. One is the *generality of the approach*: whether the approach is specific for one application domain or can be exploited in various, or even all, domains. Indeed, we have seen approaches that have targeted specific domain experts, such as care givers [79], biologists [50], museum curators [72], while approaches such as App Inventor [41] are not application domain dependent and have been used for various application types. Other EUD tools are able to support applications for various domains as long as they are developed through a specific technology, such as most of the tools we have discussed for Web technologies. Another relevant dimension is the *coverage of the main interactive application aspects*: some approaches focus only on developing the interactive part, and others aim to cover also the functional parts. For example, Denim is a tool focusing more on the interactive part while d.mix and the ServFace builder aim to provide support for developing access to functionalities implemented as Web services as well.

A further differentiating aspect is *whether and how abstractions are used to hide the implementation details*. For example, the jigsaw metaphor is an abstract representation of the structure of an application and its components, whereby the components corresponding to each jigsaw piece depend on the environment exploiting such metaphor. It has been exploited in different ways in different environments:

(i) in Puzzle [88], the jigsaw pieces are associated with high-level functionalities developed by programmers so that end users need only to compose them without knowing how they were implemented;

(ii) App Inventor [41] addresses the application development at a more detailed granularity thus asking the end-user developers to use jigsaw pieces representing low-level programming constructs and specify what should be done when low-level events occur.

Thus, App Inventor provides more flexibility in the development than Puzzle but requires more programming knowledge.

Another type of approach is represented by programming-by-example: it does not exploit abstractions and requires users to provide some specific concrete sequences of interactions, which are then generalised to implement the desired application behaviour.

7. Possible Future Evolutions

While recent years have seen a considerable increase in novel proposals to address EUD issues, there is still a lot of work to be done to realise its full potential, and the continuous on-going technological evolution poses new challenges and opportunities to EUD.

According to Gartner Inc., context-aware technologies will involve $96 billion of annual consumer spending worldwide by 2015. Thus, there is increasing need for platforms for general management of applications that can be composed by end users, which will also offer the possibility to *customize such compositions according to the context of use*. Some early examples in this direction have recently become available in the marketplace. Tasker [96] is an Android app that allows users to perform context-sensitive actions based on simple event-trigger rules. The user is in charge of creating the context-sensitive rules. However, it is still too limited in terms of types of applications that can be developed, but a start nonetheless and moreover demonstrates the utility of this type of contribution. Locale [97] is another Android app that allows users to create situations specifying conditions under which the user's phone settings should change. Even the latest mobile operating system versions have some small context-aware improvements (e.g., alerts that launch on location). A more structured approach is proposed in IVO [98], which aims to support users in building context-aware applications by creating workflows that determine the application behaviour when a specified context is detected. In the workflow description, it is possible to indicate what events and conditions can trigger the various activities. Despite the progress made, there is still a need for more general solutions, ready for wide use for developing context-aware applications in areas of major societal interest. Such solutions should provide authoring tools that should be executable in various

types of interactive devices (desktop, tablet, and mobile) and able to easily manipulate user-customizable context-dependent adaptation rules (often represented in terms of events/conditions/actions). Only recently have researchers started to investigate how people develop through their mobile phones, in particular, the ways that end users programmatically use mobile phones' special hardware (e.g., GPS, accelerometer, and gyroscope) for practical everyday purposes [99], and there is still a lot to understand in this perspective. The goal is to bridge the gap between what technologies can provide and what end users require to realize a full vision of ubiquitous computing [100].

Natural interaction aims to make the interaction with software environments more similar to interactions among people. Recent technological advances are making this vision more and more possible. For example, the vocal modality is much better supported and is now used in various applications in the mass market (e.g., Google Voice, iOS SIRI). The multimodal aspects can refer to combined use of voice and graphical direct manipulation techniques, even with gestures, in order to make expressing user intentions more intuitive. It will be useful to investigate novel metaphors, intelligent advisors, and multimodal user interfaces exploiting more recent post-WIMP interaction techniques [101, 102]. New approaches synthesising natural language, sketching, and graphical representations manipulatable by gestures in intuitively understandable models could realise fully *natural development* [103]. End users will be enabled to create or modify interactive applications while using conceptual models and multimodal design languages with intelligent support that facilitates their development, analysis, and use. Intelligent critics and advisors can assist end-users with problem solving in the design and creation of software artefacts. The purpose of the intelligent advisors tools is to make such development environments more effective, by exploiting some mechanisms able to monitor/capture end user behaviour and learn from it.

Social support will become more and more important in end-user development activities as well, also given the availability of platforms, such as Facebook, connecting millions of people online. Social networks can become useful tools to share and discuss the design results among user communities (ActionShot [67] is one of the first examples in this direction). In addition, crowdsourcing concepts can be studied and developed to support end-users community development. In this way, people who need a solution can pose the problem to a wide community who share similar interests and see whether someone can provide a suitable solution for it. Crowdsourcing and social networks need to be further explored in order to conduct needs analysis and collect feedback from end users as well as tools to support social end-user development by which people can easily share problems and associated solutions, together with their underlying rationale. Some initial work in this direction has been carried out. Nebeling and others [104] have presented an approach for the lightweight development of Web information systems based on the idea of involving crowds in the underlying engineering and design processes. The approach is designed to support developers as well as nontechnical end users in composing data-driven Web interfaces in a plug-and-play manner. To enable this, they introduce the notion of crowdsourced Web site components whose design can gradually evolve as the crowd associates the components with more data and functionality.

Another possibility is to investigate new interaction paradigms in development activities. One promising approach in this context is tangible interfaces [105]. They are user interfaces in which a person interacts with digital information through the physical environment. Investigation of this type of approach already started several years ago; for example, the project on programmable bricks [106] has produced results that allow children to build and program even robots and served as inspiration for commercial products, such as LEGO MindStorms, but with the advent of the Internet of Things is becoming evermore interesting and actual.

References

[1] C. Scaffidi, M. Shaw, and B. Myers, "Estimating the numbers of end users and end user programmers," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '05)*, pp. 207–214, Dallas, Tex, USA, September 2005.

[2] B. W. Boehm, C. Abts, A. Winsor Brown et al., *Software Cost Estimation with COCOMO II*, Prentice Hall PTR, Upper Saddle River, NJ, 2000.

[3] H. Lieberman, F. Paternò, and V. Wulf, Eds., *End-User Development*, Human Computer Interaction Series, Springer, New York, NY, USA, 2006.

[4] A. J. Ko, R. Abraham, L. Beckwith et al., "The state of the art in end-user software engineering," *ACM Computing Surveys*, vol. 43, no. 3, article 21, 2011.

[5] A. J. Ko and B. A. Myers, "Designing the whyline: a debugging interface for asking questions about program behavior," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI '04)*, pp. 151–158, April 2004.

[6] F. Paternò, *Model-Based Design and Evaluation of Interactive Applications*, Springer, New York, NY, USA, 2000.

[7] J. M. C. Fonseca, Ed., "W3C model-based UI XG final report 2010," May 2010, http://www.w3.org/2005/Incubator/model-based-ui/XGR-mbui-20100504/.

[8] F. Paternò, C. Santoro, and L. D. Spano, "MARIA: a universal, declarative, multiple abstraction-level language for service-oriented applications in ubiquitous environments," *ACM Transactions on Computer-Human Interaction*, vol. 16, no. 4, article 19, 2009.

[9] R. Jacob, L. Deligiannidis, and S. Morrison, "A Software model and specification language for non-WIMP user interfaces," *ACM Transactions on Computer-Human Interaction*, vol. 6, no. 1, pp. 1–46, 1999.

[10] P. Szekely, "Retrospective and challenges for model-based interface development," in *Design, Specification and Verification of Interactive Systems*, Eurographics, pp. 1–27, Springer, Vienna, Austria, 1996.

[11] B. A. Myers and W. Buxton, "Creating highly-interactive and graphical user interfaces by demonstration," *Computer Graphics (ACM)*, vol. 20, no. 4, pp. 249–258, 1986.

[12] A. Cypher, *Watch What I Do: Programming by Demonstration*, The MIT Press, Cambridge, Mass, USA, 1993.

[13] A. Cypher, "Eager: programming repetitive tasks by example," in *Proceeding of the CHI Conference on Human Factors in Computing Systems (CHI '91)*, pp. 33–39, ACM Press, New Orleans, La, USA, 1991.

[14] B. A. Myers, *Creating User Interfaces by Demonstration*, Academic Press, San Diego, Calif, USA, 1998.

[15] B. A. Myers, J. Goldstein, and M. A. Goldberg, "Creating charts by demonstration," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI '94)*, pp. 106–111, April 1994.

[16] G. Fischer and A. Girgensohn, "End-user modifiability in design environments," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90)*, pp. 183–192, 1990.

[17] W. E. Mackay, "Patterns of sharing customizable software," in *Proceedings of the ACM Conference on Computer-Supported cooperative work (CSCW '90)*, pp. 209–221, ACM Press, Los Angeles, Calif, USA, 1990.

[18] A. MacLean, K. Carter, L. Lövstrand, and T. Moran, "User-tailorable systems: pressing the issues with buttons," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '90)*, pp. 175–182, ACM Press, Seattle, Wash, USA, 1990.

[19] B. Nardi, *A Small Matter of Programming*, MIT Press, Cambridge, Mass, USA, 1993.

[20] A. Mørch, "Three levels of end-user tailoring: customization, integration and extension," in *Computers and Context*, M. Kyng and L. Mathiassen, Eds., pp. 51–76, MIT Press, Cambridge, Mass, USA, 1997.

[21] V. Wulf, V. Pipek, and M. Won, "Component-based tailorability: enabling highly flexible software applications," *International Journal of Human Computer Studies*, vol. 66, no. 1, pp. 1–22, 2008.

[22] J. R. Rasure and C. S. Williams, "An integrated data flow visual language and software development environment," *Journal of Visual Languages and Computing*, vol. 2, no. 3, pp. 217–246, 1991.

[23] F. Paternò, I. Campari, and R. Scopigno, "The design and specification of a visual language: an example for customising geographic information systems functionalities," *Computer Graphics Forum*, vol. 13, no. 4, pp. 199–210, 1994.

[24] M. M. Burnett, M. J. Baker, C. Bohus, P. Carlson, S. Yang, and P. van Zee, "Scaling up visual programming languages," *Computer*, vol. 28, no. 3, pp. 45–54, 1995.

[25] D. Bricklin, B. Frankston, and D. Fylstra, "VisiCalc, software arts," 1979, http://www.bricklin.com/history/intro.htm.

[26] M. Burnett, S. Yang, and J. Summet, "A scalable method for deductive generalization in the spreadsheet paradigm," *ACM Transactions on Computer-Human Interaction*, vol. 9, no. 4, pp. 253–284, 2002.

[27] J. A. Johnson, B. A. Nardi, C. L. Zarmer, and J. R. Miller, "Ace. Building interactive graphical applications," *Communications of the ACM*, vol. 36, no. 4, pp. 41–55, 1993.

[28] B. A. Myers, S. E. Hudson, and R. Pausch, "Past, present and future of user interface software tools," *ACM Transactions on Computer Human Interaction*, vol. 7, no. 1, pp. 3–28, 2000.

[29] G. Fischer, E. Giaccardi, Y. Ye, A. G. Sutcliffe, and N. Mehandjiev, "Meta-design: a manifesto for end-user development," *Communications of the ACM*, vol. 47, no. 9, pp. 33–37, 2004.

[30] A. Faaborg and H. Lieberman, "A goal-oriented web browser," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI '06)*, pp. 751–760, April 2006.

[31] G. Little, T. A. Lau, A. Cypher, J. Lin, E. M. Haber, and E. Kandogan, "Koala: capture, share, automate, personalize business processes on the web," in *Proceedings of the 25th SIGCHI Conference on Human Factors in Computing Systems 2007 (CHI '07)*, pp. 943–946, May 2007.

[32] M. Burnett, A. Sheretov, B. Ren, and G. Rothermel, "Testing homogeneous spreadsheet grids with the "what you see is what you test" methodology," *IEEE Transactions on Software Engineering*, vol. 28, no. 6, pp. 576–594, 2002.

[33] G. Fischer, "Domain-oriented design environments," *Automated Software Engineering*, vol. 1, no. 2, pp. 177–203, 1994.

[34] G. Fischer, K. Nakakoji, and Y. Ye, "Metadesign: guidelines for supporting domain experts in software development," *IEEE Software*, vol. 26, no. 5, pp. 37–44, 2009.

[35] A. Repenning and A. Ioannidou, "Agent-based end-user development," *Communications of the ACM*, vol. 47, no. 9, pp. 43–46, 2004.

[36] A. Repenning and A. Ioannidou, "What makes end-user development tick? 13 design guidelines," in *End-User Development*, H. Lieberman, F. Paternò, and V. Wulf, Eds., Human Computer Interaction Series, pp. 51–85, Springer, New York, NY, USA, 2006.

[37] A. Repenning and J. Sullivan, "The Pragmatic Web. Agent based multimodal web interaction with no browser in sight," in *Proceedings of the Conference on Human-Computer Interaction (INTERACT '03)*, IOS Press, 2003.

[38] J. F. Pane, B. A. Myers, and L. B. Miller, "Using HCI techniques to design a more usable programming system," in *Proceedings of the IEEE Symposia on Human Centric Computing Languages and Environments (HCC '02)*, pp. 198–206, 2002.

[39] B. A. Myers, J. F. Pane, and A. Ko, "Natural programming languages and environments," *Communications of the ACM*, vol. 47, no. 9, pp. 47–52, 2004.

[40] H. Lieberman, *Your Wish Is My Command. Programming by Example*, Morgan Kaufmann, Academic Press, New York, NY, USA, 2001.

[41] App Inventor MIT, 2012, http://appinventor.mit.edu/.

[42] M. Resnick, J. Maloney, A. Monroy-Hernández et al., "Scratch: programming for all," *Communications of the ACM*, vol. 52, no. 11, pp. 60–67, 2009.

[43] J. Humble, A. Crabtree, T. Hemmings et al., "'Playing with the Bits' user-configuration of ubiquitous domestic environments," in *UbiComp 2003: Ubiquitous Computing*, A. K. Dey, A. Schmidt, and J. F. McCarthy, Eds., Lecture Notes in Computer Science, Springer, Berlin, Germany, 2003.

[44] R. Andersen and A. Morch, "Mutual development: a case study in customer-initiated software product development," in *Proceedings of the 2nd International Symposium on End-User Development*, vol. 5435 of *Lecture Notes in Computer Science*, pp. 31–49, Springer, Siegen, Germany, 2009.

[45] M. F. Costabile, D. Fogli, P. Mussio, and A. Piccinno, "End-user development: the software shaping workshop approach," in *End User Development*, H. Lieberman, F. Paternò, and V. Wulf, Eds., Human-Computer Interaction Series, pp. 183–205, Springer, Berlin, Germany, 2006.

[46] M. F. Costabile, A. Piccinno, D. Fogli, and A. Marcante, "Supporting interaction and co-evolution of users and systems," in *Proceedings of the Working Conference on Advanced Visual Interfaces (AVI '06)*, pp. 143–150, May 2006.

[47] M. F. Costabile, P. Mussio, L. P. Provenza, and A. Piccinno, "Supporting end users to be co-designers of their tools," in

Proceedings of the 2nd International Symposium on End-User Development, vol. 5435 of *Lecture Notes in Computer Science*, pp. 70–85, Springer, Siegen, Germany, 2009.

[48] S. Kuhn and M. J. Muller, "Participatory design—introduction to the special section," *Communications of the ACM*, vol. 36, no. 6, pp. 24–28.

[49] K. Bødker, F. Kensing, and J. Simonsen, *Participatory IT Design: Designing for Business and Workplace Realities*, MIT Press, Cambridge, Mass, USA, 2004.

[50] C. Letondal and W. E. Mackay, "Participatory programming and the scope of mutual responsibility: balancing scientific, design and software commitment," in *Proceedings of the 8th Participatory Design Conference Artful Integration: Interweaving Media, Matrials and Practices (PDC '04)*, pp. 31–41, July 2004.

[51] R. C. M. Tam, D. Maulsby, and A. R. Puerta, "U-TEL: a tool for eliciting user task models from domain experts," in *Proceedings of the 1998 International Conference on Intelligent User Interfaces (IUI '98)*, pp. 77–80, January 1998.

[52] J. A. Landay and B. A. Myers, "Sketching interfaces: toward more human interface design," *Computer*, vol. 34, no. 3, pp. 56–64, 2001.

[53] A. Coyette, S. Kieffer, and J. Vanderdonckt, "Multi-fidelity prototyping of user interfaces," in *Proceedings of the 11th IFIP TC13 International Conference on Human-Computer Interaction (INTERACT '07)*, vol. 4662 of *Lecture Notes in Computer Science*, pp. 149–162, Springer, Rio de Janeiro, Brazil, September 2007.

[54] H. Liu and H. Lieberman, "Programmatic semantics for natural language interfaces," in *Proceedings of the ACM Conference on Human Factors in Computing Systems, (CHI '05)*, Portland, Ore, USA, April 2005.

[55] G. Little and R. C. Miller, "Translating keyword commands into executable code," in *Proceedings of the 19th Annual ACM Symposium on User Interface Software and Technology (UIST '06)*, pp. 135–144, October 2006.

[56] T. R. G. Green and M. Petre, "Usability analysis of visual programming environments: a 'cognitive dimensions' framework," *Journal of Visual Languages and Computing*, vol. 7, no. 2, pp. 131–174, 1996.

[57] A. F. Blackwell and T. R. G. Green, "A cognitive dimensions questionnaire optimised for users," in *Proceedings of the 12th Annual Meeting of the Psychology of Programming Interest Group*, A. F. Blackwell and E. Bilotta, Eds., pp. 137–152, 2000.

[58] C. Scaffidi, A. Cypher, S. Elbaum, A. Koesnandar, and B. Myers, "Using scenario-based requirements to direct research on web macro tools," *Journal of Visual Languages and Computing*, vol. 19, no. 4, pp. 485–498, 2008.

[59] C. Scaffidi, B. A. Myers, and M. Shaw, "Fast, accurate creation of data validation formats by end-user developers," in *End-User Development*, V. Pipek, M. B. Rosson, B. de Ruyter, and V. Wulf, Eds., vol. 5435 of *Lecture Notes in Computer Science*, pp. 242–261, Berlin, Germany, 2009.

[60] J. Soriano, D. Lizcano, M. A. Canas, M. Reyes, and J. J. Hierro, "Fostering innovation in a mashup-oriented enterprise 2.0 collaboration environment," in *Proceedings of the SIWN International Conference on Adaptive Business Systems (ICABS '07)*, pp. 62–669, Chengdu, China, 2007.

[61] J. Nichols and T. Lau, "Mobilization by demonstration: using traces to re-author existing web sites," in *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI '08)*, pp. 149–158, January 2008.

[62] J. A. Macías and F. Paternò, "Customization of Web applications through an intelligent environment exploiting logical interface descriptions," *Interacting with Computers*, vol. 20, no. 1, pp. 29–47, 2008.

[63] M. Toomim, S. M. Drucker, M. Dontcheva, A. Rahimi, B. Thomson, and J. A. Landay, "Attaching UI enhancements to websites with end users," in *Proceedings of the ACM Conference on Human Factors in Computing Systems*, pp. 1859–1868, 2009.

[64] J. Lin, J. Wong, J. Nichols, A. Cypher, and T. A. Lau, "End-user programming of mashups with vegemite," in *Proceedings of the 13th International Conference on Intelligent User Interfaces (IUI '09)*, pp. 97–106, February 2009.

[65] R. C. Miller, M. Bolin, L. B. Chilton, G. Little, M. Webber, and Y. Chen-Hsiang, "Rewriting the web with chickenfoot," in *No Code Required: Giving Users Tools to Transform the Web*, pp. 39–62, Elsevier, Burlington, Mass, USA, 2010.

[66] G. Leshed, E. M. Haber, T. Matthews, and T. Lau, "CoScripter: automating & sharing how-to knowledge in the enterprise," in *Proceedings of the 26th Annual CHI Conference on Human Factors in Computing Systems (CHI '08)*, pp. 1719–1728, April 2008.

[67] I. Li, J. Nichols, T. Lau, C. Drews, and A. Cypher, "Here's what i did: sharing and reusing web activity with ActionShot," in *Proceedings of the 28th Annual CHI Conference on Human Factors in Computing Systems (CHI '10)*, pp. 723–732, April 2010.

[68] J. Wong and J. I. Hong, "Making mashups with marmite: towards end-user programming for the web," in *Proceedings of the 25th SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*, pp. 1435–1444, May 2007.

[69] G. Ghiani, F. Paternò, and L. D. Spano, "Creating mashups by direct manipulation of existing web applications," in *End-User Development*, vol. 6654 of *Lecture Notes in Computer Science*, pp. 42–52, Springer, Berlin, Germany, 2011.

[70] B. Hartmann, L. Wu, K. Collins, and S. R. Klemmer, "Programming by a sample: rapidly creating web applications with d.mix," in *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST '07)*, pp. 241–250, October 2007.

[71] T. Nestler, A. Namoun, and A. Schill, "End-user development of service-based interactive web applications at the presentation layer," in *Proceedings of the 3rd ACM SIGCHI Symposium on Engineering Interactive Computing Systems (EICS '11)*, pp. 197–206, June 2011.

[72] G. Ghiani, F. Paternò, and L. D. Spano, "Cicero designer: an environment for end-user development of multi-device museum guides," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 5435, pp. 265–274, 2009.

[73] A. Celentano and M. Marek, "An end-user oriented building pattern for interactive art guides," in *End-User Development*, M. Costabile, Y. Dittrich, G. Fischer, and A. Piccinno, Eds., vol. 6654 of *Lecture Notes in Computer Science*, pp. 187–202, Springer, Berlin, Germany, 2011.

[74] A. F. Blackwell and R. Hague, "AutoHAN: an architecture for programming the home," in *Proceedings of the IEEE Symposia on Human-Centric Computing Languages and Environments*, pp. 150–157, September 2001.

[75] A. F. Blackwell, "End-user developers at home," *Communications of the ACM*, vol. 47, no. 9, pp. 65–66, 2004.

[76] K. P. Akesson, A. Crabtree, P. Hansson et al., "'Playing with the Bits' User-Configuration of Ubiquitous Domestic Environments," in *Proceedings of the 5th International Conference on*

Ubiquitous Computing (UbiComp '03), vol. 2864 of *Lecture Notes in Computer Science*, pp. 256–263, 2003.

[77] A. K. Dey, T. Sohn, S. Streng, and J. Kodama, "iCAP: interactive prototyping of context-aware applications," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 3968, pp. 254–271, 2006.

[78] S. Cotterell and F. Vahid, "A logic block enabling logic configuration by non-experts in sensor networks," in *Proceedings of the Extended Abstracts on Human Factors in Computing Systems (CHI '05)*, pp. 1925–1928, 2005.

[79] S. P. Carmien and G. Fischer, "Design, adoption, and assessment of a socio-technical environment supporting independence for persons with cognitive disabilities," in *Proceedings of the 26th Annual CHI Conference on Human Factors in Computing Systems (CHI '08)*, pp. 597–606, April 2008.

[80] R. Hull, B. Clayton, and T. Melamed, "Rapid authoring of mediascapes," Tech. Rep. HPL-2004-154, 2004.

[81] J. Floch, "A framework for user-tailored city exploration," in *End-User Development*, vol. 6654 of *Lecture Notes in Computer Science*, pp. 239–244, Springer, Berlin, Germany, 2011.

[82] P. Baudisch, X. Xie, C. Wang, and W. Y. Ma, "Collapse-to-zoom: viewing web pages on small screen devices by interactively removing irrelevant content," in *Proceedings of the Annual ACM Symposium on User Interface Software and Technology (UIST '04)*, pp. 91–94, October 2004.

[83] G. Ghiani, F. Paternò, and C. Santoro, "On-demand cross-device interface components migration," in *Proceedings of the 12th International Conference on Human-Computer Interaction with Mobile Devices and Services (Mobile HCI '10)*, pp. 299–307, September 2010.

[84] U. Tuomela, I. Kansala, J. Hakkila, and J. Mantyjarvi, "Context-Studio? Tool for personalizing context-aware applications in mobile terminals," in *Proceedings of the Australasian Computer Human Interaction Conference (OzCHI '03)*, p. 292, Nokia Research Center, 2003.

[85] J. Danado, M. Davies, P. Ricca, and A. Fensel, "An authoring tool for user generated mobile services," in *Proceedings of the 3rd Future Internet Conference on Future Internet (FIS '10)*, A. Berre, A. Gomez-Pérez, K. Tutschku, and D. Fensel, Eds., pp. 118–127, Springer.

[86] J. Seifert, B. Pfleging, E. Bahamóndez, M. Hermes, E. Rukzio, and A. Schmidt, "Mobidev: a tool for creating apps on mobile phones," in *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services (MobileHCI '11)*, pp. 109–112, ACM, 2011.

[87] J. Danado and F. Paternò, "A prototype for EUD in touch-based mobile devices," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '12)*, pp. 83–86, 2012.

[88] J. Danado and F. Paternò, "Puzzle: a visual-based environment for end user development in touch-based mobile phones," in *Human-Centered Software Engineering*, vol. 7623 of *Lecture Notes in Computer Science*, pp. 199–216, 2012.

[89] S. Cuccurullo, R. Francese, M. Risi, and G. Tortora, "MicroApps development on mobile phones," in *End-User Development*, M. Costabile, Y. Dittrich, G. Fischer, and A. Piccinno, Eds., vol. 6654 of *Lecture Notes in Computer Science*, pp. 289–294, Springer, Berlin, Germany, 2011.

[90] A. De Lucia, R. Francese, M. Risi, and G. Tortora, "Generating applications directly on the mobile device: an empirical evaluation," in *Proceedings of the International Working Conference on Advanced Visual Interfaces (AVI '12)*, pp. 640–647, 2012.

[91] D. Dearman and J. Pierce, "It's on my other computer, computing with multiple devices," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '08)*, pp. 767–776, ACM Press, Florence, Italy, 2008.

[92] Google Research Report, "The new multi-screen world: understanding cross-platform consumer behavior," 2012, http://services.google.com/fh/files/misc/multiscreenworld_fi- nal.pdf.

[93] J. Lin and J. A. Landay, "Employing patterns and layers for early-stage design and prototyping of cross-device user interfaces," in *Proceedings of the 26th Annual CHI Conference on Human Factors in Computing Systems (CHI '08)*, pp. 1313–1322, April 2008.

[94] J. Borchers, *A Pattern Approach to Interaction Design*, Wiley, Chichester, UK, 2001.

[95] F. Paternò, C. Santoro, and L. D. Spano, "Engineering the authoring of usable service front ends," *Journal of Systems and Software*, vol. 84, no. 10, pp. 1806–1822, 2011.

[96] Tasker, http://tasker.dinglisch.net/.

[97] Locale, http://www.twofortyfouram.com/.

[98] V. Realinho, T. Romão, and A. E. Dias, "An event-driven workflow framework to develop context-aware mobile applications," in *Proceedings of the 11th International Conference on Mobile and Ubiquitous Multimedia (MUM '12)*, article 12, ACM Press, 2012.

[99] B. Athreya, F. Bahmani, A. Diede, and C. Scaffidi, "End-user programmers on the loose: a study of programming on the phone for the phone," in *Proceedings of the IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC '12)*, pp. 75–82, 2012.

[100] S. Holloway and C. Julien, "The case for end-user programming of ubiquitous computing environments," in *Proceedings of the FSE/SDP Workshop on the Future of Software Engineering Research (FoSER '10)*, pp. 167–171, November 2010.

[101] M. Beaudouin-Lafon, "Instrumental interaction: an interaction model for designing post-WIMP user interfaces," in *Proceedings of the Conference on Human Factors in Computing Systems "The Future is Here" (CHI '00)*, pp. 446–453, April 2000.

[102] R. J. K. Jacob, O. Shaer, A. Girouard et al., "Reality-Based interaction: a framework for post-WIMP interfaces," in *Proceedings of the 26th Annual CHI Conference on Human Factors in Computing Systems (CHI '08)*, pp. 201–210, April 2008.

[103] S. Berti, F. Paternò, and C. Santoro, "Natural development of nomadic iterfaces based on conceptual descriptions," in *End-User Development*, pp. 143–160, Springer, 2006.

[104] M. Nebeling, S. Leone, and M. C. Norrie, "Crowdsourced web engineering and design," in *Web Engineering*, vol. 7387, 2012, pp. 31–45, Springer, Berlin, Germany.

[105] O. Shaer, N. Leland, E. Calvillo-Gamez, and R. Jacob, "The TAC paradigm: specifying tangible user interfaces," *Personal and Ubiquitous Computing*, vol. 8, no. 5, pp. 359–369, 2004.

[106] M. Resnick, "Behavior constrction kits," *Communications of the ACM*, vol. 36, no. 7, pp. 64–71, 1993.