IST PROGRAMME

Action Line: IST-2002-8.1.2

End-User Development

Empowering people to flexibly employ advanced information and communication technology

Contract Number IST-2001-37470

D1.1+D1.2
Research Agenda and Roadmap for EUD

Editors:

Fabio Paternò (ISTI-CNR),

Markus Klann (FhG-FIT),

Volker Wulf (FhG-FIT)

Summary

This document presents a roadmap and a research agenda for the end-user development field. The goal is to identify promising action lines in this area for the EC VI Framework and the broader scientific community.

December 2003

Table of contents

Executive Summary

Most European citizens have become familiar with the basic functionality and interfaces of computers. However, while some substantial progress has been made in improving the way users can access interactive software systems, developing applications that effectively support users' goals still requires considerable expertise in programming that cannot be expected from most European citizens. Thus, one fundamental challenge for the coming years is to develop environments that allow people without particular background in programming to develop their own applications or modify existing ones, with the ultimate aim of empowering people to flexibly employ advanced information and communication technologies, especially in the framework of upcoming ambient intelligence environments.

We think that over the next few years we will be moving from *easy-to-use* (which has yet to be completely achieved) to *easy-to-develop interactive software systems*. Some studies report that by 2005 there will be 55 million end-user developers, compared to 2.75 million professional users [BAB00]. End-User Development (EUD) in general means the active participation of end-users in the software development process. In this perspective, tasks that are traditionally performed by professional software developers are transferred to the users. Thus, the users need to be specifically supported in performing such tasks. In particular, end-user development 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 or modify a software artefact.*

End User Development has an impact at various levels, and future research has to address all of them:

- *Programming paradigms and languages*: different approaches to programming can have an impact on how easily end-users can access and use software. Many formalisms exist, which were all invented by computer scientists to serve their needs for describing the various situations they encounter. In such activities, computer scientists are both the providers and the users of the formalisms. All such notations have an underlying programming paradigm, which is a model of how programs will be interpreted and executed by the computer. Logical programming, functional programming, object-oriented programming, parallel programming, spreadsheets, script languages are examples of such paradigms, all of which need to be revised to support EUD.
- *Methods, environments and tools*: they should be revised in order to allow people with little background in programming to be able to create or modify software artefacts. This requires environments able to provide natural support for the tasks that users intend to perform, and to effectively represent and manipulate the objects necessary to this purpose. In addition, EUD calls for a better integration of development and documentation.
- *Architectural issues*: there are aspects important for end user development that can have an impact on the underlying software architecture, for example, when addressing flexible environments supporting multiple-user cooperation.

- *Interaction techniques*: new techniques developed in the human-computer interaction field for representing and analysing large information spaces can provide beneficial results in creating easier to use development environments.
- *Application domains*: each domain has its own language and set of important concepts. The domain experts would like to have environments where they can directly manipulate such concepts through familiar metaphors without having to learn computer-oriented languages.
- *Organizational and social issues*: the work context can have an impact on how people organise the development process. Users of end-user programmable and tailorable systems (e.g., CAD systems, spreadsheets) will often be organised (directly or indirectly) in help networks and local hierarchies. The goal of such organisation is to help those who are not programmers to modify their applications or to create new ones. Common middle-level roles in such an organisational structure are "super users". These are experts in the field who are also knowledgeable about end-user programming and tailoring. In addition, they need to be sociable and willing to help other people to get their job done.

A number of application domains seem particularly amenable to end user development environments. An example is given by home applications. The home has the potential to become one of the most popular application domains for information technology. Thus, it may be the home where the need for end user development will be particularly important. First research contributions in this area have been put forward in order to support downloading logical descriptions of the state and functioning of appliances and then generating the corresponding interfaces for controlling such appliances. Another particularly interesting application domain is the design environment, usually supported by CAD systems, in manufacturing enterprises, with evident impact on improving financial/quality aspects of their development process. In the scientific domain there is a great deal of interest in end user development, for example in biology, in order to allow scientific experts to tailor applications to their information needs. Enterprise resource planning (ERP) is one of the most important software areas for the European industry. Recently, leader companies in the market have realised the importance of end user concepts that allow various types of users of large ERP software to modify it in order to obtain systems more suitable for their actual needs. Over the past years, we have seen a significant change in users' expectations of business applications. Traditional ERP applications were very much centred around one single functional area, and the dominant user scenarios were data entry, reporting, and ERP work flow. Such a simplified user model is not sufficient for modern business solutions. In these systems, the user is an active knowledge worker who needs communication and analysis tools, content management, and ad-hoc collaborative workflow and the ability to tailor the system to her own needs. At the same time, it is important to dramatically simplify the customisation process and enable business experts and end users to modify the software themselves without the need for IT consultants.
In parallel, the ever-increasing introduction of new types of interactive devices poses new challenges for developers, even end user developers, who have to create or modify applications taking into account the features of the devices that can be used to access the functionality, which may range from small, handheld devices to large, wall-size, interactive areas.

1. Background for Research in End-User Development

1.1 Motivations

The widespread penetration of interactive software systems has raised an increasing need for better environments for building applications. Most European citizens have become familiar with the basic functionality and interface of computers. In addition, some results from the research in human-computer interaction and usability have started to penetrate certain product markets, thus improving the levels of usability.

However, while some substantial progress has been made in improving the way users can access interactive software systems, developing applications that effectively support users' goals still requires considerable expertise in programming that cannot be expected from most European citizens. Thus, one fundamental challenge for the coming years is to develop environments that allow people without particular background in programming to develop their own applications or modify existing applications, with the ultimate aim of empowering people to flexibly employ advanced information and communication technologies, especially within the upcoming ambient intelligence environments.

We think that over the next few years we will be moving from *easy-to-use* (which has yet to be completely achieved) to *easy-to-develop interactive software systems*. Some studies report that by 2005 there will be 55 million end-user developers, compared to 2.75 million professional users [BAB00]. This is a multidisciplinary area that needs contributions from Human-Computer Interaction, Software Engineering, Computer Supported Cooperative Work, Artificial Intelligence, Ethnography, Cognitive Psychology and others.

From the area of technology development, see Figure 1, we learn that, amongst other performance indicators, the storage capacity and connectivity bandwidth increase rapidly. By having more storage and high bandwidth it becomes possible to deliver large amounts of content to a wide variety of devices. This could bring forward the situation of content overload to consumers. In terms of storage capacity for example, we see the emergence of high capacity optical storage media (today up to 22 Giga Byte) small enough to be integrated in many devices including portable systems. Connectivity is being supported by many different standards going from short-range wireless (low power) to full in-home networks for streaming high quality multimedia content. In order to cope with this potential content overload, more functionality (such as different query and content management methods) will be introduced. The danger of this approach is that users will spend more time on operating devices than actually enjoying the content they want.

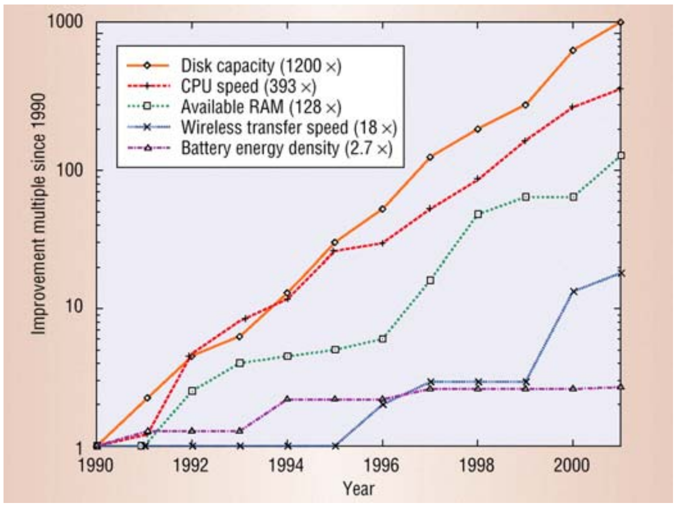

Figure 1: Moore's law for trends in storage, CPU, memory, wireless connectivity and battery technology (see [D03]).

In order to address such issues, interactive systems should have some form of intelligence. However, given the fundamental need of users to be in control, end-user development becomes important to provide users with the ability to modify or program the behaviour of their intelligent environment. This is important for all the companies whose business is devoted more towards the mass market, an area where programming skills are not so extended. So, in order to achieve the goal, the devices must be easy and fast to personalize, and they must provide facilities to manage both push and pull information, such as filters.

In addition, despite advances in mobile device technology and wireless infrastructures (e.g. UMTS, GPRS, Wireless LANs, and Bluetooth) European manufactures of the latest mobile devices and network providers are unable to exploit consumer and business to business opportunities. Inherent to this deficiency is the issue of application development, which is costly, complex as well as lacking usability know-how in the technology/application domain.

This situation is depicted in Figure 2, showing that the individual productivity to create software increases substantially slower than the performance/cost provided by hardware and the amount of software present in products, hence leading to a productivity gap.

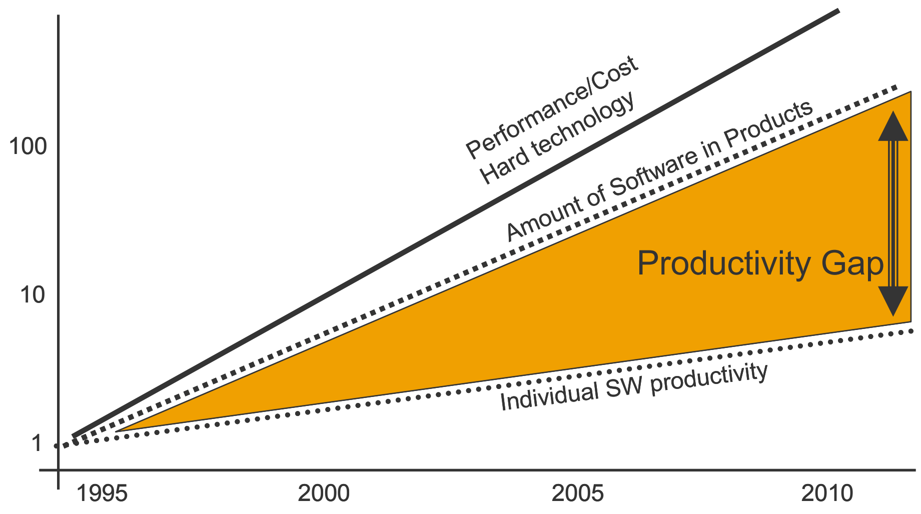

Figure 2: Software Productivity Gap (SO 2.3.2.3 / IST)

EUD can be a strategic solution to bridge the productivity gap by allowing end users to directly implement some additional features important to accomplish their tasks.
Given that users' tasks, users' qualifications, and organizational environments are differentiated and dynamically changing, there is a big need to make software flexible within its application context. EUD will provide non-professional programmers with the tools to adapt software within its application context. Since the European work force is well educated in comparison to most international competitors, one can assume that European workers will be particularly able to make use of EUD tools and methods to improve their business processes. EUD will lead to better work processes by making the software more flexible during use and thus enabling innovative non-anticipated appropriation processes [OH97], [PW99]. Moreover, EUD supports processes of Integrated Organization and Technology Development where the introduction of a highly flexible software is connected to processes of organizational and personnel development [WR95]. So EUD can be seen as a key enabler for the development of innovative IT-based business processes and a higher productivity of labour.
Moreover, EUD is important to the development of the information society on a political level. The information society is characterized by the fact that computer networks will become the leading media. Other traditional media will be integrated on these networks. However, the creation of content and the modification of the functionality of these infrastructures is difficult for non-professional programmers. Therefore, in many sectors of society it is possible to find a division of labour between those who produce and those who consume. This development has the negative social effects of exclusion and deactivation. Gerhard Fischer calls these phenomenon "gift wrapping culture". In dealing with this phenomenon, EUD allows citizen to actively participate in the information society.

Some end-user development oriented techniques have already been adopted by software for the mass market such as macros in MS-Word or some programming-by-example techniques in MS-Excel. However, we are still quite far from their systematic adoption. The interactive capabilities of new devices have created the potential to overcome the traditional separation between end users and software developers. New environments able to seamlessly move between using and programming (or customizing) can be designed.

One of the goals of end-user development is to reach closeness of mapping: as Green and Petre [GP96] said: "The closer the programming world is to the problem world, the easier the problem-solving ought to be… Conventional textual languages are a long way from that goal". Even graphical languages often fail to provide immediate understanding of representations for the developers. The need for end user development is clearly emerging.

1.2 Definition

End-user development means, in its most general sense, the active participation of end-users in the software development process. In this perspective, tasks that are traditionally performed by professional software developers are transferred to the users. They need to be specifically supported in performing these tasks. The range of active user participation in the software development process can range from providing information about requirements, use cases and tasks, including participatory design, to end-user programming. In other words, EUD also means to provide end-users with the possibility of creating or modifying a software artefact when the system is already released and people are using it. At the first EUD-Net workshop held in Pisa a definition of End User Development was identified: "*End User Development is a set of activities or techniques that allow people, who are non-professional developers, at some point to create or modify a software artefact*".

This definition has been sharpened in the network discussions by a slight but important modification: "*End User Development is 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 or modify a software artefact".* The new definition aims to highlight that we focus on users of software systems and that non-professional developer is a role rather than a definitive classification. Thus, it can happen that the same person in different contexts can act in some cases as professional developer and in others as a non-professional developer.

As it has been pointed out by Costabile et al. [CFL03], two types of end-user activities can be identified:

1. Activities that allow users, by setting some parameters, to choose among alternative behaviours (or presentations or interaction mechanisms) already available in the application; such activities are usually called parameterisation or customisation or personalization.
2. Activities that imply some modification through any programming paradigm, thus creating or modifying a software artefact. Examples of programming paradigms are: programming by demonstration, programming with examples, visual programming, macro generation, script languages.

Mørch [M97] has discussed that the second group of activities can be further divided into integration and extension. *Integration* goes beyond customisation by allowing users to add new functionality to an application, without accessing the underlying implementation code: users tailor an application by linking together predefined components within or across the application. *Extension* refers to the case in which the application does not provide, by itself or by its components, any functionality that accomplishes a specific user need, thus adding a new functionality generates a radical change in the software. In the above categorization, there are instances that cut across categorical boundaries.

Examples of activities belonging to the first type (according to Costabile et al.'s classification) are:

- *Parameterisation*. It is intended as specification of unanticipated constraints in data analysis. In this situation, observed very often, the user wishes to guide a computer program by indicating how to handle several parts of the data in a different way; the difference can just lie in associating specific computation parameters to specific parts of the data, or in using different functionalities available in the program.
- *Annotation*. The users write comments next to data and results in order to remember what they did, how they obtained their results, and how they could reproduce them.

Examples of activities belonging to the second type are:

- *Modelling from the data*. The system supporting the user derives some models from observing interaction data.
- *Programming by demonstration*. Users show examples of property occurrences in the data and the system infers from them a (visual) function.
- *Indirect interaction with application objects*. As opposed to direct manipulation, a command language can be provided to script objects.
- *Incremental programming*. It is close to traditional programming, but limited to changing a small part of a program, such as a method in a class. It is easier than programming from scratch.
- *Extended annotation*. A new functionality is associated with the annotated data. This functionality can be defined by any technique previously described

We believe that EUD more properly involves the second set of activities (according to Costabile et al.'s classification) because in the first set only use of software is actually performed: no real modification is made on the software when the user can just enter information according to predefined formats.

1.3 Requirements

Software practices – including use, design, development and maintenance – seem to change character around adaptable systems. As tailoring interfaces allows the user to change the program, the border between use and design gets blurred. As use, tailoring, adaptation, maintenance and development activities get intertwined, they have to be coordinated in a different way. Traditional borders defining a project in this manner are often too rigid. Applications must be tailorable, adaptable by their users to meet the changing requirements. End-user development also means the adaptation and further development of software in response to individual preferences, changing co-operative work practices as well as developing business practices. This may in the future be reconceptualised as long-term, evolutionary application development.

Software is more and more used in co-operative settings mediating between different work practices and integrating a range of data sources. Especially in so-called data centred businesses – in such diverse areas as telecommunication, municipal service provision, banking and insurance – software becomes part of a technical infrastructure intrinsically interlaced with the work and business practices it supports. Neither can be changed without changing the other as well.

Model-based approaches can be useful for end-user development because they allow people to focus on the main concepts (the abstractions) without being confused by many low-level details. Through meaningful logical abstractions it is also possible to support participation of end-users already in the early stages of the development process. Optimally, model-based software development is to be combined with prototype-oriented development. Like programming, modelling requires the availability of suitable and usable languages and supporting tools to be effective. Visual modelling languages have been identified as promising candidates for defining models of the software systems to be produced. UML and related tools such as Rationale Rose are the best known examples. They inherently require notions of abstractions and should deploy concepts, metaphors, and intuitive notations that allow professional software developers, domain experts, and users to communicate ideas and concepts. This requirement is of prominent importance if models are not only to be understood, but also used and even produced by end users. Language usability needs to be empirically analysed.

An important issue is the integrated description of system functionality and user interaction. Fundamentally, this calls for identifying task allocation between the user and the system and relating user and system views in an integrated design process, i.e., integrated modelling of user interface and system functionality. Nevertheless, separation of aspects to be addressed within the integrated model needs to be achieved by defining (role-oriented) partial models as views of the overall model. This raises issues of consistency management in models.

Naturally, end-user development requires that the tasks intended to be performed by end users have to be addressed and solved beforehand on the more technical, underlying levels. For example, dynamic reconfiguration of system components (e.g. customisable interoperability) by end users must be supported by underlying component models and system architectures that technically enable this interoperability and component reuse. Only after these operations have been realized on a technical level, can such tasks be effectively performed by end users on a more abstract level.

The trade-off between expressiveness and usability is a general concern in the area of end-user development. For example, there are several direct manipulation environments that allow easy assembling of applications. Simple functionality can thus be easily constructed using the graphical interface. However, more sophisticated functionality and complex behaviour that go beyond simple applications or early prototypes require the use of a scripting language that is typically integrated with the visual elements of the graphical user interface, thus requiring specific programming skills.

End-user development has some important effects on other, more technical levels of software. Administration of customisable systems is far more complex than dealing with mostly standardized configurations and implementations. Component-based development raises issues of standardized interfaces, interoperability, etc. Adaptability in general requires ensuring program correctness and not invalidating other required properties. Therefore, adequate means for defining semantics and analysing system properties in the presence of adaptation are needed to restrict possible modifications.

Tailoring interfaces have to present the flexible aspects and their manipulation in a way that is comprehensible for their users. The design of the tailoring features should provide the user with a gentle slope of complexity, so that on the one hand, beginners can feel safe making simple changes, while experts or super users, on the other hand, have powerful tools to implement their designs. Additional support comprises help systems, the possibilities of documenting the results of tailoring and the domain-specific test features.

1.4 Dimensions for EUD

Sutcliffe and others [SLM03] have identified three dimensions to describe EUD environments (see Figure 3). The first dimension is the scope. Some systems are developed with the intention of supporting users in a narrow domain of expertise. Programmable queries on protein structures is an example in BioInformatics. Such applications are task- and domain-specific. On the other hand, many EUD environments are intended to be general-purpose tools that can be applied to a wide variety of problems. In a similar manner to expert programming languages, EUD systems vary in scope from specific to general.

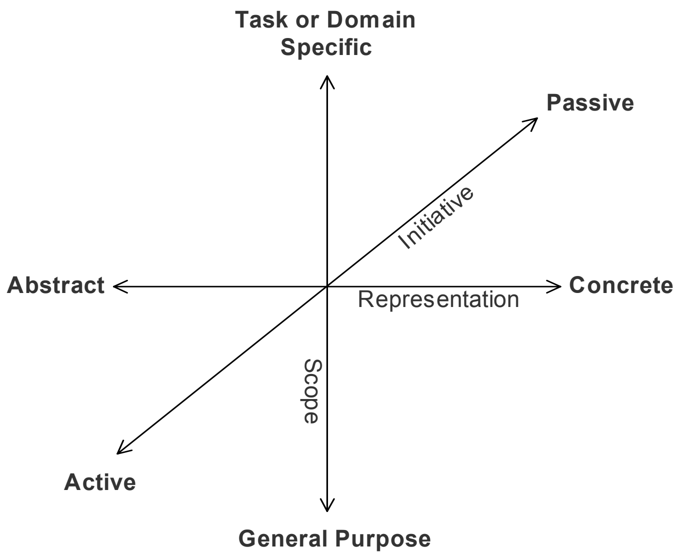

Figure 3: Dimensions of end-user development

The second dimension concerns the representations used for communicating with the user. Communication may use natural language and natural user actions; alternatively, a more formal language may be used which the end user has to learn. The modality of communication is also involved; for instance, instructions might be given by drawing intuitively understood marks and gestures (e.g. Palm Pilot), or manipulating a set of physical objects (e.g. turning a set of dials to program a washing machine). More formal communication can be achieved by symbolic text or diagrammatic languages; the formal syntax might be spoken, although this is unlikely. This dimension may be described as a range from abstract (non-natural) to concrete (natural) representations. Representations could be assessed for naturalness and other properties such as changeability, comprehensibility, etc., using cognitive dimensions [GP96]. The key psychological trade-off for the naturalness dimension is the learning burden imposed on the end user by any artificial language versus the errors in interpretation that may arise from ambiguities inherent in less formal means of communication.

System initiative forms the third dimension for EUD. Systems might leave initiative completely with the user and just provide a means of instructing the machine. At the other extreme, intelligent systems infer the user's wishes from demonstrated actions or tracking user behaviour, and then take the initiative to create appropriate instructions or behaviour. In between are systems that provide users with development tools but constrain their actions so that only intelligible or appropriate instructions are given. System initiative may also be mixed (see [KEOW03]), so in Domain Oriented Design Environments [F94], the system is mainly passive but it does embed critics which take the initiative when the system spots the user making a mistake. The dimensions are not completely orthogonal. For instance, natural communication in English implies some system initiatives in interpreting and disambiguating users' instructions, either automatically or by a clarification dialogue. Further dimensions may be added in the future; for instance, the concepts or subject matter represented by an EUD environment. The dimensions can be used to assess the psychological implications of different EUD approaches; however, we also need to consider the effort of development and user motivation.

2. Classification of Research Lines for End User Development

End user development has an impact at various levels in the software development:

- *Programming paradigms and languages*, different approach to programming can have an impact on how easily end-users can access and use them. Many formalisms exist, which were all invented by computer scientists to serve their needs for describing the various situations they encounter, or to satisfy certain software engineering properties. In that activity, computer scientists are both the providers and the users of the formalisms. Each of these formalisms has an underlying programming paradigm that is a model of how programs will be interpreted and executed by the computer. Logical programming, functional programming, object-oriented programming, parallel programming, spreadsheets, script languages are examples of such paradigms.
- *Methods, environments and tools*, should be revised in order to allow people with little background in programming to be able to create or modify software artefacts. In addition, EUD calls for a better integration of development and documentation.
- *Architectural issues*, there are aspects important for end user development that can have an impact on the underlying software architecture. For example, when flexible environments supporting multiple-user cooperation are addressed.
- *Interaction techniques*, new techniques developed in the human-computer interaction field can provide beneficial results in creating easier to use environments. Then, it would be important to define when each EUD technique can be useful.
- *Application domains*, each domain has its own language and set of important concepts. The domain experts would like to have environments where they can di-

rectly manipulate such concepts through familiar metaphors without having to learn computer-oriented languages.

- *Organizational factors such as the* division of labour, pattern of coordination, or reward schemes have an important impact on how end users organise the development process. Organizations should offer sufficient resources and incentives to end users to encourage their development activities. Important roles in encouraging tailoring activities are "super users", “gardener”, or “local experts”. These actors are domain experts and also knowledgeable about end-user programming. In addition, they need to be social and willing to help other people to get their job done.
- *Social issues*, users of end-user programmable and tailorable systems (e.g., CAD systems, spreadsheets) often belong (directly or indirectly) to help networks. These informal social aggregates support those who are not programmers to modify their applications or to create new ones. Cultural factors, e.g. a tailoring culture, increase the likelihood for the emergence of these help networks [CH90].

3. Promising Research Lines

In the Description of Work we identified three general research areas as those able to provide the best ingredients to obtain effective end-user development environments:

- ***Adaptive, adaptable and innovative interaction techniques.*** Environments that help users to interact with their applications by dynamically modifying their behaviour and functionality while taking into account various aspects: user behaviour, external environment, tasks to perform, interaction device and so on.
- ***Visual modelling and multimedia environments for rapid component developmen*t.** High-level environments that allow users to focus on conceptual aspects and then support rapid prototyping of corresponding user interfaces taking into account the strong usability limitations of current visual modelling environments such as those based on UML.
- ***Cooperative end user programming.*** Environments that help end users to support each other in programming, to share their programs and modify shared programs.

While after the network we can confirm the importance of such areas, we are now also able to provide a more detailed discussion of promising research lines in these areas.

3.1 Natural Development

The work in Myers’ group aims to obtain natural programming [PM96], meaning programming through languages that work in the way that people who do not have programming experience would expect. We intend to take a more comprehensive view of the development cycle, thus not limited only to programming, but also including requirements, designing, modifying, tailoring, … Natural development implies that people should be able to work through familiar and immediately understandable representations that allow them to easily express and manipulate relevant concepts, and thereby create or modify interactive software artefacts. On the other hand, since a software artefact needs to be precisely specified in order to be implemented, there will still be the need for environments supporting transformations from intuitive and familiar representations into more precise, but more difficult to develop, descriptions.

Often the initial model is the result of brainstorming by either a single person or a group. Usually people start with some paper or whiteboard sketches. This seems an interesting application area for intelligent whiteboard systems [LM01] or augmented reality techniques able to detect and interpret the sketches and convert them into a format that can be edited and analysed by desktop tools.

Most end-user development will probably benefit from the combined use of multiple representations that can have various levels of formality. The possibility of developing through sketching can be highly appreciated in order to capture the results of early analysis or brainstorming discussions. Then, there is the issue of moving the content of such sketching into representations that can more precisely indicate what artefact should be developed or how it should be modified.

A similar approach is followed when people try to use informal descriptions in natural language such as scenario descriptions for obtaining more structured representations. An example can be found in [PM99] where starting from informal scenarios it is shown how to use the information that they contain to obtain more general task models.

Recent years have seen a large adoption of visual modelling techniques in the software design process (example of CASE-tools supporting visual modelling languages such as the UML are Rationale Rose, Together, Magic Draw, Enterprise Architect, Poseidon for UML), but there are also research environments publicly available such as CTTE [MPS02]). However, we are still far from visual representations that are easy to develop, analyse and modify, especially when realistic case studies are considered. The application and extension of innovative interaction techniques ([BMA01]), including those developed in information visualization (such as semantic feedback, fisheye, two-hand interactions, magic lens, …), can noticeably improve their effectiveness.

3.2 Multimodality

The technological evolution is also enabling a wide adoption of multimodality, even in Web environments. To address multimodality in web-based applications the World Wide Web Consortium's Multimodal Interaction Working Group is currently extending and merging existing web technologies in order to cover different interaction modalities as well as multiple devices and dynamic device usages. The Multimodal Interaction Framework [MMI-FW] defines the markup languages used in the data flow between the components of multimodal applications as well as for the description of concrete interfaces. For creating interfaces, the Multimodal Interaction Framework discriminates between generation of output modes, styling them and rendering them on a device. The framework proposes existing standards such as XHTML, SVG and VoiceXML [VXML] as output of the styling phase, but is not limited to them. When retrieving user input, the framework uses EMMA [EMMA], a markup language used to represent human input to applications. It is designed to allow multiple interpretations of the same input and to collect metadata from input processors for later evaluation. An integration step disambiguates these interpretations in order to recognize the input in the framework.

In particular, vocal interaction can play an important role in this respect as well. Support for vocal interaction is mature for the mass market. Its support for the Web has been standardised by W3C [A01]. The rationale for vocal interaction is that it enables the development of applications suitable for both Internet and wireless communication, it makes practical operations more natural and faster, and it enables multi-modal applications (graphic and/or vocal).

There are many potential applications: information retrieval (provide information about news, sport, traffic, weather, …); e-commerce (order tracking, call centre, financial application, …); telephone services (personal voice dialling, teleconference, …); unified messaging (e-mail systems, personal organizers..); CAD applications, etc.

In this context, the concept of Pragmatic Web by Repenning [RS03] can provide an important contribution: instead of the traditional "click the link" browser-based interfaces, agents capable of multi-modal communication might be very useful to provide access to Web-based information. Agent communication methods include facial animation, speech synthesis, and speech recognition and understanding. End users will instruct agents to transform information in highly customised ways. Agents will work together to combine information from multiple Web pages, access information autonomously or triggered by voice commands, and represent synthesised information through multi-modal channels.

3.3 Authoring Environments for Ubiquitous Computing

Several years have passed since researchers started to think about ubiquitous computing. This challenge raises many interesting issues in computer science [W99] providing the possibility of accessing many types of devices in different environments. One important issue is how to design and develop interactive applications that can be accessed through different devices from various environments while preserving usability. A related question is whether even end users can develop such types of applications.

In a paper on the future of user interface tools [MHP00], the authors indicate that the wide platform variability *encourages a return to the study of some techniques for device-independent user interface specification, so that developers can describe the input and output needs of their applications, so that vendors can describe the input and output capabilities of their devices, and so that users can specify their preferences. Then, the system might choose appropriate interaction techniques taking all of these into account*. The basic idea is that instead of having separate applications for each device that exchange only basic data, there is some abstract description and then an environment that is able to suggest a design for a specific device that adapts to its features and possible contexts of use.

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

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

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

Various solutions have been proposed for this purpose. They vary according to a number of dimensions. For example, the automation level can be different: a completely automatic solution can provide meaningful results only when the application domain is rather narrow and consequently the space of the possible solutions regarding the mapping of tasks to interaction techniques is limited. More general environments are based on the mixed initiative principle: the tool supporting the mapping provides suggestions that the designer can accept or modify. An example is the TERESA environment [MPS03] that provides support for the design and development of nomadic applications, which can be accessed through different types of interaction platforms.

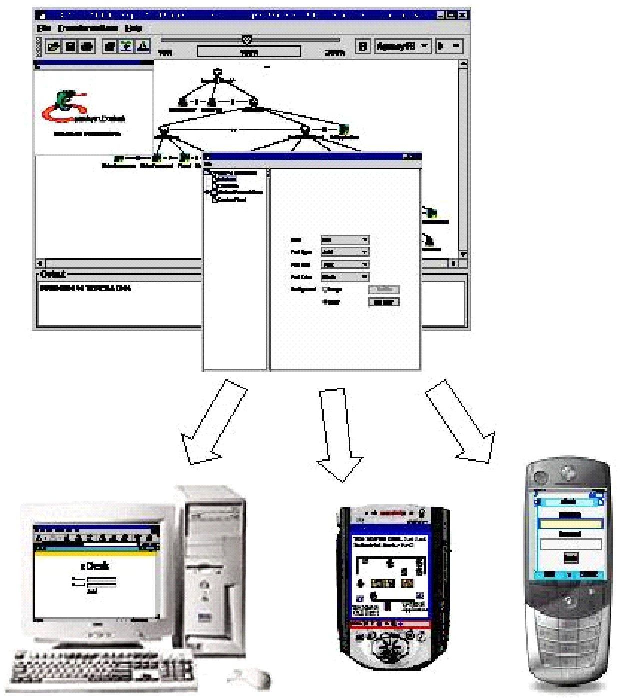

Figure 4: The TERESA Approach.

The TERESA tool (http://giove.cnuce.cnr.it/teresa.html) offers a valid support to the multi-context interface realization. The development starts with an analysis of the user task model. TERESA processes the task model to produce consistent and adapted versions of the interface according to the features of the device at hand. The output is a fast prototype of the final interface for the various devices considered (see Figure 4).

3.4 Component-based Development

Environments for component-based development give another important support in this respect. Software components and component-based design have received much attention in the software engineering and application development communities over the past years. Software components allow systems to be built by starting from high-level reusable building blocks instead of writing program statements in a general purpose programming languages. One of the great promises of composition is that it has the potential to be performed at runtime (i.e., when the system is in use). Connecting two components only requires 'glue code' (i.e., a high-level script) that records the connections between the components. However, the integration of software components by end-users to make new applications is far from trivial.

One goal is the provision of effective, interactive, high-level environments that allow users to focus on conceptual aspects and support rapid prototyping of corresponding user interfaces. Usability aspects are to be accounted for, and an adequate level of abstraction should be visually represented. Component-based development e.g. reusing components and web services ("component engineering and reuse") and pattern-based development should be supported by innovative techniques. Such domain-oriented environments should support end-users in visually designing and developing applications. Design tools are to be complemented by tools for automatic code generation or assembly of applications from components. Effective visual and/or diagrammatic languages are to be defined that account for both cognitive dimensions and issues of language engineering.

A critical bottleneck is that end users need to know what interface methods are defined on the various components and how they must be called to realise the integration of two components. Interestingly, a model for software component integration is Lego toy construction. Lego provides great flexibility in how two components can be coupled together. By keeping interfaces (connection points) general, each brick can connect to many other bricks (of different shapes). This generality is approached in software by method interfaces that cater to many combinational needs. However, the cost of generality (advantageous for component developers) is paid at the expense of end-user mastery because the connection points will often not have intuitive (domain-specific) names and may require parameters to be specified so that they can be used in many combinations.

3.5 Patterns

There is a growing interest in the possibility of using patterns [GHJV95] in user interface design, development and evaluation. For example, they can be a good way of capturing and communicating design solutions that can be reused in many areas [PM97]. In general terms they can be used for both descriptive and prescriptive purposes. In the former case they capture a recurrent aspect (it can be a recurrent behaviour or system structure or presentation structure) whereas in the latter case they can be used to indicate a solution for a recurring problem.

In the end-user development field there are many candidates for their application [ET97], for example, embodying HCI guidelines as patterns or using patterns for process and organisation design. In particular, in the identification of high-level environments it is important to pay attention to patterns of elements that users often need in order to associate them with high-level constructs.

3.6 Cognitive Support for End User Development

Another aspect to consider is the psychology of programming that indicates what important psychological aspects can have an impact on this activity. Whatever formalism or programming language is used, the underlying paradigm has a great influence on what programmers can express and with what ease. However, there is little information available on the cognitive features of programming paradigms. As both users and designers of those paradigms, computer scientists are usually more interested in exploring the formal properties of paradigms and formalisms, and the related software engineering issues, than in understanding why they prefer such or such paradigm. However, it is a fact that paradigms are more or less easy to learn and apply depending on the task. A better understanding of the underlying cognitive issues would be as important for end-user programming as a better understanding of the properties of interaction styles is for user interface design. The graphical programming environment that was provided with the first versions of Lego's Robotic Invention System is a good example. Its visual language is clear and easy to manipulate for children.

If end-user programming is to get a wide audience, it is important that the cognitive properties of programming paradigms are well known: for what situations are they better suited? What are their limitations? Otherwise, the industry might create frustration among users by choosing paradigms that lead them to create programs that they do not understand well.

Once this understanding has been achieved, it would be important to create critique environments able to analyse user support on the basis of its cognitive impact.

3.7 Architectural Concepts for Flexible Systems

The need for flexible environments has implications also at the architectural level. One example occurs when we consider adaptivity for ubiquitous computing. Adaptive environments help users to interact with their applications by dynamically modifying their behaviour and functionality while taking into account various aspects: user behaviour, external environment, tasks to perform, interaction devices and so on. In this area it is of particular interest to design applications able to address the many possible use environments, on-the-fly dynamic configuration of interaction devices and the rapidly increasing availability of many types of devices (ranging from small phones to large flat displays, including embedded computers in cameras, cars...). This development will continue and computers will start to vanish into the environment, and computational power and networking capabilities will then become ubiquitous.

This engenders the need for context-dependent applications that can be supported by both adaptive and adaptable techniques. When the system is adaptable it can be tailored (manually) by the end users to fit their needs, work practices, business goals, etc. The results will enhance user competence and awareness of the system, allowing for personal adaptations, with the creation of new functionalities and user interface features. An important aspect is that adaptations should be as unobtrusive as possible (not interfering with the task itself). Thus, more work is needed on user modelling and how it can improve efficiency and effectiveness in end-user programming.

Systems will by then be organised in intelligent environments that require new interaction metaphors and methods of control. Well-known explicit interaction devices, such as mouse and keyboard are not necessarily available, rendering user interfaces that rely on them inappropriate. Beside more natural interaction techniques (e.g. speech and gesture), mobile and static sensors will be used in intelligent environments to support implicit user interaction.

One of the main challenges for the success of ubiquitous computing is the design of personalised user interfaces and software that supports easy access to relevant information and that is flexible enough to handle changes in user context and availability of resources.

3.8 Application domains (home applications, ...)

A number of application domains seem particularly suitable for end user development environments. An example is given by home applications. In their homes people will interact with more and more electronic devices. This means that the house potentially can become one of the most popular applications for information technology. Thus, it can be a domain where the need for end user development will be particularly important. First research contributions in this area have been put forward. An example is PUC (Programmable Universal Controller) [NMH02]: a system that supports downloading logical descriptions of appliances' states and functions and then generation of the interface for controlling such appliances. However, this approach does not allow end users to change the resulting interfaces.

Another particularly interesting application domain is certainly the design environment, usually supported by CAD systems, in manufacturing enterprises with evident impact on improving financial/quality aspects of their development process. Designers as end-users, who have deep knowledge of their specific environment and that are not professional developers, must be supplied with visual development tools in order to formalise 'programmatic' solutions to their needs. In the scientific domain there is a lot of interest in end user development. For example in biology, experience acquired at the Pasteur Institute during several years indicates that in the field of biology software there are many local developments in order to deal with daily tasks, such as managing data, analysing results, or testing scientific ideas. Moreover, it is worth mentioning that many biologists have no or very limited programming skills, and yet feel the need of modifying the application they use to better fit their needs.

ERP is one of the most important software areas for the European industry. Again, leading companies in the market have recently realised the importance of end user concepts that allow various types of users of large ERP systems to modify the software in order to obtain systems more suitable for their actual needs. Over the past years, we have seen a significant change in the expectation of business applications. Traditional ERP applications were very much centred around one single functional area and the dominant user scenarios were data entry, reporting, and ERP work flow. This simplified user model is not sufficient for modern business solutions like Customer Relationship Management, Human Capital Management, Knowledge Management, and Supplier Relationship Management. In these systems, the user is an active knowledge worker who needs communication tools, analytics, content management, and ad-hoc collaborative workflow and the capability of tailoring the system to her own needs. At the same time, the total cost of ownership (TCO) of ERP software becomes the main competitive argument. TCO can only be reduced by dramatically simplifying the customisation process and by enabling business experts and end users to modify the software themselves without the need of hiring IT consultants or IT-administrators. Already today, Enterprise Portals offer the personalization or creation of custom-made web pages and reports. Last but not least, companies such as SAP see a shift into a service-based architecture of business applications that may result in a new application development paradigm in which traditional coding is replaced by orchestration of existing enterprise services. Service composition including generation of User Interfaces may become an activity of business experts using simplified development environments with pre-packaged semantics. Considering these changes in the user model of ERP software, such companies see an increasing relevance of End User Development within their products.

Another application domain is that related to systems supporting data intensive businesses like telecommunication, e-government or banking. Computer applications become integrated in infrastructures connecting different work practices within and across organisational borders. The flexibility of such infrastructures is of strategic importance when developing new services. Often the need to redevelop part of the computer support to accommodate business or organisational development prohibits the development at all. Tailorable systems and domain specific end user development provide a competitive advantage.

A survey questionnaire administered to several parties, both from academia and industry, also outside EUD-Net, indicated that office, home, and research are considered the most promising application domains for EUD [CP03]. Other application domains, not listed in the questionnaire, were also pointed out: education (indicated by most people), decision analysis, and medical domain.

3.9 Cooperative End-User Programming

Cooperative end-user programming involves environments that help end users to support each other in programming, to share their programs and modify shared programs. Given the fact that users typically have very different skills and interests in tailoring or programming, there are many different divisions of labour with regard to these activities. Therefore, it is important to provide technical features which support cooperative end-user programming. An important aspect of this is to develop annotation and manipulation tools that act on partial designs, allowing users to customise software directly in individual or cooperative working environments. In some cases it will also be important to consider that the cooperation will occur across groups of people with various levels of expertise. There are two types of cooperation in cooperative EUD: cooperation among end-users themselves and cooperation between developers and end-users. In order to perform a local adaptation of some software system, local developers need to know something about its design and in that sense they have an "indirect communication" or cooperation with developers. Optimally, from an EUD perspective, a direct communication link would be provided between developers and end-users, but this is not often possible (expensive, inaccessible) and instead the communication becomes indirectly supported by intermediate representations [MM00]. In systems that have undergone several versions of local adaptations the cooperation may return to cooperation between end-users (as local developers).

3.10 Flexibilizing Software Development

End-user development and tailoring changes the temporal, personal and organisational borders between design, development and use. The use of software systems will interlace with adaptation and further development by both domain experts and software engineers. Agile methods [C02] address similar issues. The agile approach focuses on delivering business value early in the project lifetime and being able to incorporate late-breaking changes in requirements by accentuating the use of rich, informal communication channels and frequent delivery of running, tested systems, all the while devoting due attention to the human component of software development. Proponents of the agile approach say that these practices lead to more satisfied customers and a superior success rate of delivering high quality software on time.

End user development can be expected to have a strong impact on software engineering as it requires agility of development methods, that is to say methods that are more people oriented than process oriented and emphasize flexibility and adaptability over full description.

3.11 Tailoring Cooperation

Tailorable systems provide users with the possibility of adapting existing system functionality at runtime. This is meant for non-professional IT-people, since they are the foremost experts on their own work, in order for them to adapt computer support to fit their specific needs and the dynamics of the workplace. Tailoring in such contexts is itself a cooperative activity [TB94]. Changes must be deliberated before any implementation can take place, fit with the existing software design and be accepted in the joint workplace. Tailored artefacts and programs are likely to be shared [HK91], but this is no panacea and requires specific consideration by developers of tailorable systems. The technique of sharing software artefacts has already proven itself in developer communities (e.g. open source development). However, for this to be a success in non-professional user communities, it requires the end user developers to document their contributions, such as with annotations and design rationale. Also, mechanisms have to be provided to find and access appropriate artefacts, such as naming and classification conventions, access rights and adequate views. The trustworthiness of a given artefact must be accessible to users by way of e.g. learning about the artefact's creator or its previous history of use. The possibilities and limitations of tailorability have to be designed and presented in a way that is comprehensible not only for computer professionals, but also for end users [MM00], and the design of the tailoring features of an application must be adapted not only with the system's user interface, but also with the use, technical and developmental contexts [DL03].

3.12 Software Engineering for End Users

The definition of end-user development is based on the differences between end users and professional programmers and software engineers. There are differences in training, in the scale of problems to be solved, in the processes, etc. However, there are some similarities. Some of those similarities are to be found in the life cycle of the developed software artefacts. For instance, managing the successive versions of a piece of software will most probably become a problem for end users. Version management is already a problem with word processor documents. Reports or letters are often written in several phases: a businessman will write successive versions of a contract and has it proofread by all parties; a home user will reuse the same letter year after year when sending his or her tax report, and just changes the figures in the letter. Smart or appropriately educated users learn a simple technique aimed at helping them manage the successive versions: assigning a number to each version.

However, one cannot expect an end user to apply the techniques provided by the software engineering field. Software engineering methods and tools require knowledge of concepts and techniques that end users do not have. They imply the use of methods and tools that require specific training. They probably consume more time than an end user is willing to afford, etc. In addition, not all problems from software engineering are equally important for end users: team development techniques are most probably beyond end users' needs. Consequently, an interesting line of research consists in identifying new sets of techniques and tools that would be the counterpart of software engineering for end users: software crafting.

The following research directions are fertile for end-user software development:

- Determining what theoretical and empirical studies of the problems addressed by software engineering can be transposed to end user development, as well as why and how.
- Studies to identify possibly existing problems that are specific to end user development and are thus not addressed by software engineering
- Research on methods and tools that would address the previously identified problems in ways that are suitable to end users: "lightweight methods", tools to support them, and appropriate user interfaces taking into account end-users tasks and activities.

For example, debugging should be made easier in end user development given that it is a phase that requires a lot of effort. More generally, software engineering is traditionally concerned with the creation and modification of software artefacts. End-user development has basically the same objective but with the users being developers. Thus, the central question of software engineering issues in EUD is how EUD relates to software engineering. This question also refers to the distinction of end-user developers on the one hand and software engineers and professional programmers on the other hand in the definition of end-user development coined during the network discussions. Therefore, EUD research that is oriented towards software engineering first has to examine the interrelationships of end-user development and software engineering. Interesting lines of future research can then be how software engineering practices can be adapted and applied to facilitate end-user development, and, vice versa, what impact end-user development has on the software engineering discipline. These questions can be addressed on the levels of concepts, languages, methods, and tools.

From a software engineering perspective, end-user development in general means the active participation of end-users in the software development lifecycle. Active end-user participation may span across the whole software life cycle. It can range from providing information about requirements, use cases and tasks, via participatory design (e.g. based on the use of visual modelling languages) and end-user programming (e.g. programming by example or programming by demonstration) to adaptation, tailoring, modification, and evolution. Tasks that are traditionally performed by professional software developers are to be accomplished by users. Tasks that are specific to end-user development will have to be integrated. Particular attention must be paid to post-release changes by users, their anticipation and the preparation of the software and its development process to accommodate them, and their impact on software qualities like maintainability and interoperability. In HCI, it is observed that using the system changes the users, and as they change they will use the system in new ways. As a consequence, systems must be designed so that they can evolve to accommodate those users' needs that cannot be anticipated in the requirement phase or that are newly determined because of user evolution [CF03].

End-user developers need to be specifically supported in performing their tasks. It is an ultimate goal to define the range of end-user participation in software development in terms of generic and domain-specific, end-user oriented software development methods and processes, and to build supporting development tools and environments.

4. Roadmap for Research in End User Development

To outline a potential roadmap for research in EUD, we focus on three intertwined lines of research: software architectures, interfaces, and support for collaboration. Starting with the current status of EUD, we discuss what research activities could reasonably be carried out until 2005, what status would then be reached, and how research could continue until 2010. As for the predictions for 2010 and 2020, they are obviously rather general in nature and do not yet include concrete recommendations for research activities. Rather, they deal with the impact of EUD on the Information Society, state possible applications and societal aspects and make guesses on what EUD goals might realize at the corresponding time.

With regard to software architectures a number of different approaches exist in research (e.g. agent-based, component-based, rule-based). However, empirical knowledge on suitability in real-world settings is still insufficient. A mayor challenge is to refine existing approaches and developing new architectures suitable for both system and user evolution. The combination with conventional architectures and the adaptation with the respective development processes have to be investigated. Moreover, one has to look into the suitability of different EUD-architectures to support run-time changes with a gentle slope of complexity. Case studies need to show the suitability of these frameworks for different application domains. Furthermore, patterns of decomposition have been developed.

With regard to interfaces, research has been carried out on various interface techniques, e.g. Augmented Reality, Tangible User Interfaces. Among others, interfaces need to be designed to make end-users aware of existing EUD-functionality and allow them to perform safely experiments. Use of alternative interfaces for EUD is beginning to emerge. Knowledge on cognitive suitability and design criteria is still insufficient. Therefore, empirical studies on EUD-interfaces need to be conducted. New UI-techniques, e.g. supplementing adaptability with adaptive context-sensitive system behaviour, need to be developed. Experiences from real-world applications need to be gathered. In the near future, research has to focus on EUD-systems which combine adaptability with adaptivity (e.g. based on user- and context-models). Moreover, interfaces need to be developed in order to make EUD-functionality available with a gentle slope of complexity. Innovative interfaces should be able to check consistency and signal errors. Design guidelines should become available for different classes of interface problems.

Collaborative aspects have been taken up in research as a key element to EUD (e.g. gardening-metaphor). However, empirical knowledge on collaborative EUD is still insufficient and implementations of collaborative EUD-functionality are only in their beginning. Therefore, concepts for collaborative EUD have to be developed. Standards for describing EUD artefacts have to be worked out to make them exchangeable (e.g. with regard to quality, recommendations, purpose). Based on such standards, EUD-artefacts can be described and placed into repositories for sharing. Software agents should become able to recommend suitable artefacts available from such repositories.

Looking towards 2010, we believe that architectural frameworks, decomposition-techniques, patterns, interfaces, and tools for collaboration support can exist in a consolidated and integrated manner. End Users are supported by tools for exploring, testing and assessing while carrying out EUD activities. Cases of best practices have been documented to explain EUD as an activity that is embedded in social networks. Concepts to acquire EUD-skills are integrated into educational curricula. EUD starts becoming an important aspect of applications in most domains: education, scientific research (e.g. bioinformatics), business (CAD, ERP, GIS), and domestic domains.

Towards 2020, substantial adaptability has become a property of all newly developed software systems. Adaptable software-systems have penetrated into all domains, e.g. business, leisure, home, culture. Most people have skills in EUD. EUD has become an important activity for most jobs and for the majority of people. A high level of adaptivity in all devices is a big part of what is called ‘Ambient Intelligence’. While EUD has gained central importance in the application of information technology, the users are not more aware of these features. EUD has become an integral aspect of their appropriation of IT.

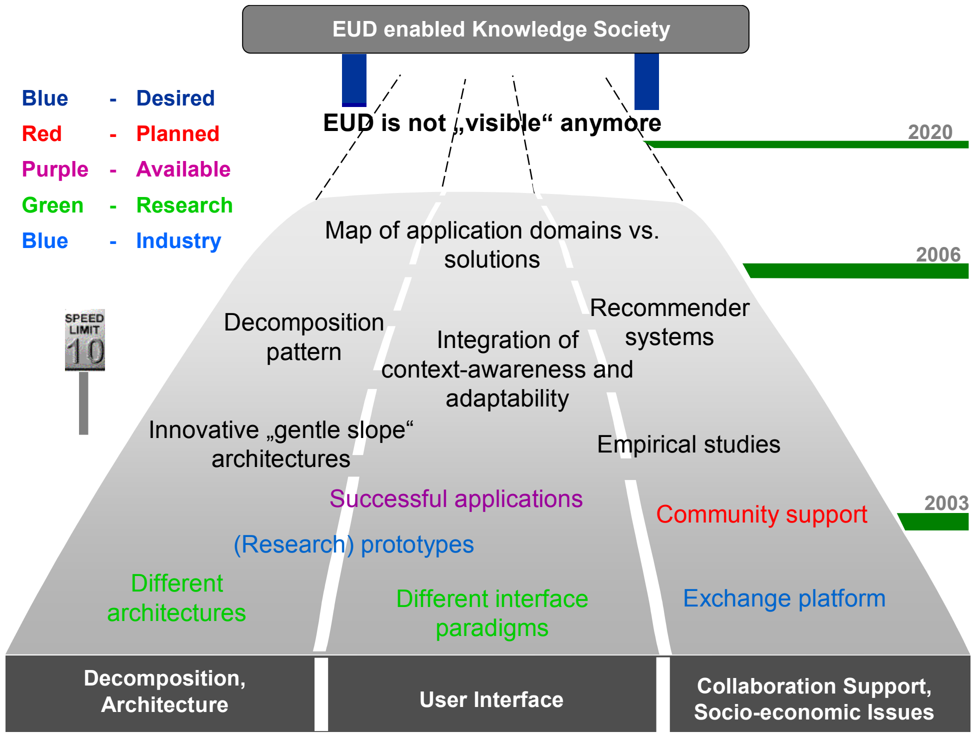
EUD enabled Knowledge Society
Blue - Desired
Red - Planned
Purple - Available
Green - Research
Blue - Industry
EUD is not „visible“ anymore
2020
Map of application domains vs. solutions
2006
SPEED LIMIT 10
Decomposition pattern
Recommender systems
Integration of context-awareness and adaptability
Innovative „gentle slope“ architectures
Empirical studies
Successful applications
Community support
2003
(Research) prototypes
Different architectures
Different interface paradigms
Exchange platform
Decomposition, Architecture
User Interface
Collaboration Support, Socio-economic Issues

Figure 5: The future of End-User Development.

Conclusions

Discussion of the research agenda during the lifetime of the network has been very rich and therefore difficult to summarise. After a discussion of the motivations and requirements most relevant to end user development identified so far, this report outlines some promising research lines that, if adequately supported, can provide important results in obtaining effective end user development environments, which is one strategic goal for the European Information Society.

As Figure 5 shows, the road to achieving an EUD enabled knowledge society is still long, even if prototypes, mainly research ones, have already appeared. Particularly challenging areas that will be addressed in the near future are decomposition patterns, integration of context-awareness and adaptivity, and recommender systems. The ultimate goal is to provide users with non-intrusive, "invisible" support for their developments.

Acknowledgements

All EUD-Net members who provided comments and suggestions are acknowledged for their help to improve this document.

References

[A01] Ken Abbott, Voice Enabling Web Applications: VoiceXML and Beyond. ISBN: 1893115739, APress L. P., 2001

[BMA01] Beaudouin-Lafon M., Mackay E., Andersen P., at al., CPN/Tools: A Post-WIMP Interface for Editing and Simulating Coloured Petri Nets. Proceedings ICATPN 2001. pp.71-80, Springer Verlag LNCS N. 2075.

[BAB00] Barry W. Boehm, Chris Abts, A. Winsor Brown, Sunita Chulani, Bradford K. Clark, Ellis Horowitz, Ray Madachy, Donald J. Reifer, and Bert Steece, Software Cost Estimation with COCOMO II, Prentice Hall PTR, Upper Saddle River, NJ, 2000.

[C93] A. Cypher, ed. Watch What I Do MIT Press, 1993**.**

[CH90] Carter, K.; Henderson, A.: Tailoring Culture, in: Hellman, R.; Ruohonen, M.; Sorgard, P. (eds): Proceedings of the 13th IRIS, Reports on Computer Science and Mathematics, No. 107, Abo Akademi University 1990, pp. 103 - 116

[C02] A. Cockburn. Agile Software Development. Addison Wesley. 2002.

[CF03] M.F. Costabile, D. Fogli, G. Fresta, P. Mussio, A. Piccinno, “Building environments for End-User Development and Tailoring”, IEEE Symposia on Human Centric Computing Languages and Environmnets, Auckland, New Zeeland, October 28-31, 2003.

[CFL03] Costabile, M.F.; Fogli, D.; Letondal, C.; Mussio, P.; Piccinno, A.: Domain-Expert Users and their Needs of Software Development, paper at Special Session on EUD, UAHCII Conference, Crete, June 2003.

[CP03] M.F.Costabile and A.Piccinno, Analysis of EUD Survey Questionnaire, D4.2, October 2003.

[D03] De Ruyter B., Challenges for End-User Development in CE devices, paper at Special Session on EUD, UAHCII Conference, Crete, June 2003.

[DL03] Dittrich, Y., and Lindeberg, O. (2003): ‘Designing for Changing Work and Business Practices.’ in: N. Patel (ed.) Adaptive Evolutionary Information Systems. Idea Group Publishers, 2003.

[EMMA] http://www.w3.org/TR/emma/

[ET97] Erickson, T., Thomas, J., “Putting it all together: pattern languages for interaction design.” *Proceedings CHI97*, Extended Abstracts*,* 1997.

[F94] Fischer, G. (1994). Domain-Oriented Design Environments. *Automated Software Engineering*, *1*(2), 177-203.

[GHJV95] Gamma, E., Helm, R., Johnson, R., Vlissides, J., *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison Wesley, 1995.

[GP96] T.R.G. Green, M. Petre; Usability analysis of visual programming environments: a 'cognitive dimensions' framework, in J. Visual Languages and Computing, Vol.7, N.2, pp.131-174, 1996.

[HK91] Henderson, A., and Kyng, M. (1991): ‘There is no place like Home: Continuing Design in Use.’ in J. Greenbaum and M. Kyng: Design at Work, Lawrence Erlbaum Associates 1991, pp. 219-240.

[Ka01] Kahler H.: Supporting Collaborative Tailoring, Ph.D.-Thesis, Roskilde University, Denmark, Roskilde, 2001.

[KEOW03] Klann, M.; Eisenhauer, M; Oppermann, R.; Wulf, V. (2003): Shared initiative: Cross-fertilisation between system adaptivity and adaptability. UAHCII’03, Crete, June 2003.

[LM01] Landay J. and Myers B., "Sketching Interfaces: Toward More Human Interface Design." In IEEE Computer, 34(3), March 2001, pp. 56-64.

[Ma90] Mackay, W.E.: Users and customizable Software: A Co-Adaptive Phenomenon, PhD Thesis, MIT, Boston (MA), 1990

[MM00] Mørch, A.I. and Mehandjiev, N.D., Tailoring as Collaboration: The Mediating Role of Multiple Representations and Application Units. Computer Supported Cooperative Work 9(1), 75-100.

[MMI-FW] http://www.w3.org/TR/mmi-framework/

[MPS02] Mori G., Paternò F., Santoro C., CTTE: Support for Developing and Analysing Task Models for Interactive System Design, *IEEE Transactions in Software Engineering,* pp. 797-813, August 2002 (Vol. 28, No. 8), IEEE Press.

[M97] Morch. A., Three Levels of End-User Tailoring: Customization, Integration, and Extension. In M. Kyng & L. Mathiassen (eds.), *Computers and Design in Context*, (51-76). The MIT Press, Cambridge.

[MHP00] Myers, B., Hudson, S., Pausch, R. *Past, Present, Future of User Interface Tools*. Transactions on Computer-Human Interaction, ACM, 7(1), March 2000, pp. 3-28.

[MPS03] Mori G., Paternò F., Santoro C., "Tool Support for Designing Nomadic Applications", Proceedings ACM IUI'03, Miami, pp.141-148, ACM Press.

[Na93] Nardi, B. A.: A Small Matter of Programming - Perspectives on end-user computing, MIT-Press, Cambridge et al., 1993.

[NMH02] Nichols, J., Myers, B.A., Higgins, M., Hughes, J., Harris, T.K., Rosenfeld, R., Pignol, M. "Generating Remote Control Interfaces for Complex Appliances," in *UIST 2002*. Paris, France: pp. 161-170.

[OH97] Orlikowski W. J.; Hofman J. D., "An Improvisational Model for Change Management: The Case of Groupware Technologies"; in: Sloan Management Review (Winter 1997); 1997; pp. 11-21.

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

[PM96] Pane J. and Myers B. (1996), "Usability Issues in the Design of Novice Programming Systems" TR# CMU-CS-96-132. Aug, 1996. http://www.cs.cmu.edu/~pane/cmu-cs-96-132.html

[PM97] Paternò, F., Meniconi, S., "Patterns for Dialogue Representations", *Proceedings International Workshop on Representations in Interactive Software Development*, pp. 73-81, 1997.

[PM99] Paternò F., Mancini C., Developing Task Models from Informal Scenarios, Proceedings ACM CHI'99, Late Breaking Results, pp.228-229, ACM Press, Pittsburgh, May 1999.

[PW99] Pipek V.; Wulf V.: A Groupware's Life, in: Proceedings of the Sixth European Conference on Computer Supported Cooperative Work (ECSCW '99), Kluwer, Dordrecht 1999, pp. 199 – 219

[RS03] Repenning A., Sullivan J., "The Pragmatic Web: Agent-Based Multimodal Web Interaction with no Browser in Sight", Proceedings INTERACT 2003, IOS Press, pp.212-219.

[SLM93] Sutcliffe A., Lee D.& Mehandjiev N., Contributions, Costs and Prospects for End-User Development, Proceedings HCI International 2003.

[VXML] http://www.w3.org/TR/voicexml20/

[W99] Weiser M., Some computer science issues in ubiquitous computing, ACM Mobile Computing and Communications Review, Volume 3, Number3.
[WR95] Wulf V.; Rohde M.: Towards an Integrated Organization and Technology Development; in: Proceedings of the Symposium on Designing Interactive Systems, 23. - 25.8.1995, Ann Arbor (Michigan), ACM-Press, New York 1995, pp. 55 – 64.
[Wu99] Wulf V.: „Let's see your Search-Tool!" – On the Collaborative Use of Tailored Artifacts, in: Proceedings of GROUP '99, ACM-Press, New York 1999, pp. 50 – 60.