Aspects of End User Tailoring: People, Tools, Tailoring

Anders I. Mørch
Department of Information Science
University of Bergen,
N-5020 Bergen, Norway
+47 55584118
anders@ifi.uib.no

ABSTRACT
A novel approach to conceptualizing computer applications – by aspects – is presented. The approach is proposed to simplify end-user tailoring. Different aspects of an application (user interface, rationale, and program code) are tailored separately, and the interface between adjacent aspects is updated automatically by the computer.

A tailorable application, BasicDraw, built and conceptualized as a collection of three-aspect application units has been evaluated with users in a video-recorded experiment. The results show that it is easier to tailor the interface than it is to tailor the program code. Recommendations for tailoring of code are given.

Keywords
Information appliances, tailorable applications, aspects of application units, levels of tailoring, empirical evaluation.

INTRODUCTION
End user computing is systems development undertaken by users to further develop an existing system to needs that were not accounted for in the original system [20]. This is often accomplished by writing task-oriented scripts and macros in high-level programming languages for off-the-shelf computer applications, such as word processors, spreadsheets, and drawing packages [21].

Studies has shown that a majority of word processor users (92%) do tailor their applications in various ways, such as setting parameters in preference forms to customize the system [24]. Many users find this difficult and one reason for this, according to Page et al., is that tailoring tools are not well integrated into the systems.

In this paper I report on an approach to organizing computer applications to simplify tailoring: conceptualizing them as a collection of orthogonal *aspects*. The notion of aspect is motivated in this section, and further developed in the remainder of the paper. The anticipated users of the approach are future developers of tailorable applications.

Aspects
An aspect is a singular facet of a complex (multi-faceted) interactive system. It is characterized by being physically connected to (one or more) adjacent aspects, but it is not part of any other aspect in the conventional sense of a part/whole decomposition of a complex system. Aspects are different from parts in that they are organized in what I call orthogonal aspect hierarchies rather than part/whole composition hierarchies. The notion of aspect has taken inspiration from Simon's notion of "nearly decomposable" systems [25] as well as gestalt psychology and language philosophy. The latter influence is illustrated by the Escher drawing shown in Figure 1 [8]. It is further explained below.

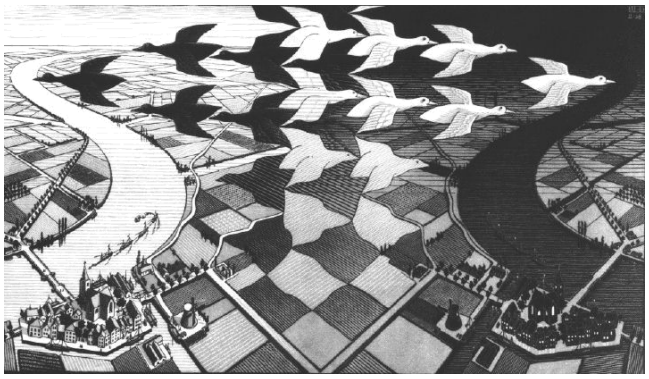

Figure 1: Organizing complexity as a collection of "nearly decomposable" aspects. M.C. Escher *"Day and Night."*

Each of the three aspects in the drawing: white-bird, black-bird, and landscape can be studied by itself without reference to the others, and each aspect provides an interface (boundary) to the others that is more or less well-defined.

The *asymmetry* of aspects is seen as a model for the asymmetries found in interactive systems. It cannot easily be modelled by conventional part/whole composition hierarchies. The Escher drawing thus provides a clue for how to (re)conceptualize and organize interactive systems for the purpose of tailoring them. Furthermore, I propose a solution for how to implement the model: identifying the relevant aspects of a system, addressing each aspect by itself, and creating well-defined interfaces between adjacent aspects that are managed by the computer.

The rest of this paper builds support for these claims and is organized as follows. Users of advanced information appliances are identified. Household appliances is suggested as a model for future information appliances. A tailorable application is viewed as a collection of instances of a type of information appliance referred to as an application unit. Tools to tailor different aspects of an application at the granularity of the application unit have been built. A tailorable application BasicDraw has been evaluated in a laboratory experiment. Results from the experiment are reported.

Information Appliances

Most users of computer applications are not interested in tailoring the applications but in using them to accomplish useful tasks, such as writing reports and sending messages by electronic mail. MacLean et al. [16] present data that shows that there is not a sharp distinction between users and programmers as is often thought. The authors characterize the range of users from programmers to regular users as lying along a "tailoring slope," and including handymen and tinkers in between. Handymen and tinkers have acquired tailoring skills that are not typical of regular users, but they are not perceived as programmers.

Mackay [15] has observed that many users are not able to tailor the system they use without help from more capable users, referred to as translators. Translators are able to take a user's problem and translating it into a solution. The kind of tailoring that is most frequently performed is referred to as *customization*: setting default preferences for user interface layout and configuration options. The majority of users consider customization to be useful [15, 24].

It has since been hypothesized that tailoring should be extended to reach below the user interface and allow different levels of a system to be tailorable as well [19], and that the next generation computer application is not a desktop application but an *information appliance* [22]. These two hypothesis, which at first may seem unrelated, form the starting point for the rest of this paper.

The majority of today's desktop applications are office applications [10]. An information appliance, on the other hand, is physically smaller than today's desktop applications, and it is primarily intended to be used in the home, integrated with media appliances, such as television, video, and telephone.

I propose that kitchen appliances be used as the model for information appliances since kitchen appliances are some of the most usable and useful tools we have today. Throughout this paper I use a can-opener as an example to illustrate my argument (Figure 2). The can-opener is seen as a simple interactive system: a unit of three orthogonal aspects: interface, rationale, and mechanism. Each of the aspects can be (conceptually) taken apart and studied by itself and is physically linked to other aspects by (more or less) well-defined interfaces. Furthermore, the can-opener serves a model for the structure of an application unit, which is the term I use for an information appliance [18]. Application units are further described in a later section.

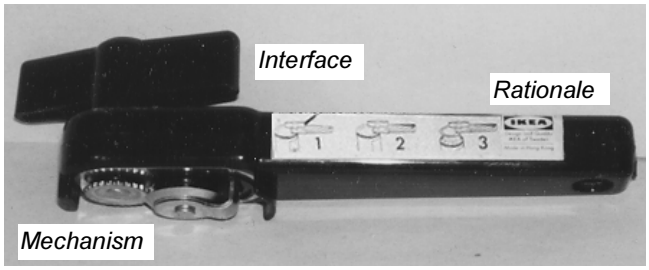

Figure 2: The IKEA can-opener, a kitchen appliance, is seen as a unit of three orthogonal aspects. It serves a model for the structure of an application unit (a kind of information appliance).

Although current kitchen appliances are relatively easy to use, they are not easy to modify. First, they are difficult to modify because they are not meant to be modified. Second, they are difficult to modify because the building blocks they are made of are static (black boxes). Computer applications, on the other hand, have a greater potential for modifiability because they are built out of dynamic building blocks (software programs) that can be changed by rewriting the program code.

RELATED WORK

Programmable Applications

Programmable applications [5] and design environments [6, 9] are classes of computational environments that give an end-user support in modifying and extending these environments. Modifiability is supported by access to high-level, domain-oriented building blocks (analogous to LEGO bricks), and extensibility is supported by writing small programs in dynamic programming languages (e.g., LISP and JAVA).

A goal of programmable applications is to support programming not in terms of low-level machine instructions but in terms of high-level, task-oriented building blocks and to elevate programming to an activity that leverages the use of the applications. A comprehensive discussion of some motivational issues behind programmable applications is given in [20].

In field studies of end-user programming behavior, Nardi and her colleagues [21] have observed that users of programmable applications, such as spreadsheets and CAD systems, are able to master the formal notations embedded in these systems (scripting languages) when the notations match the tasks the user wishes to perform. This is in part because users are already familiar with formal notations from other activities, such as knitting and baseball score keeping [21].

End User Participation

Involving end-users as first-class participants together with developers and managers in projects where new technology

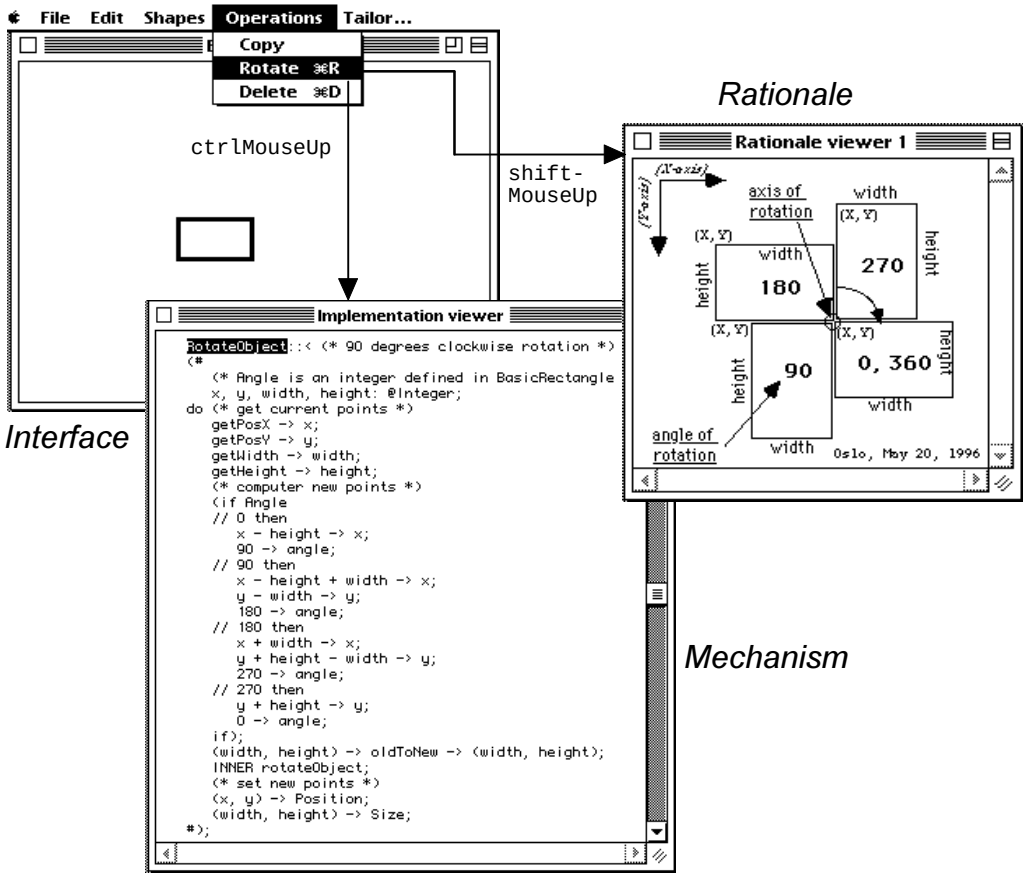

Figure 3: The BasicDraw application gives the user access to the three aspects of an application unit: Interface, Rationale, and Mechanism (program code), and the aspects are linked by eventhandlers (two displayed). The screen image shows the three aspects of the "Rotate" application unit. Tailoring will typically require making changes to each of the three aspects since they are partly interdependent.

is introduced is a well-established principle in Scandinavia [23]. Users are the foremost experts on their own work and therefore need to be involved in decisions regarding the introduction of new technology at their workplace. Furthermore, new technology will almost always change existing ways of working [23].

Ehn and Kyng [3] suggest a "tool perspective" on development and use of computer systems. This perspective emphasizes that end-users should be active participants in the design process rather than passive suppliers of data to be used in formal specifications. The authors recommend that users participate in the making of design specifications by using tools and materials they are familiar with from work and other everyday activities.

Bjerknes and Bratteteig [1] propose an "application perspective" to involve end-users in design. The application perspective puts technology in the context of use, i.e., viewing it from the users' work tasks. These tasks are different from the formal procedures encoded in the technology they use to accomplish the tasks. On this basis the authors recommend that computer systems be built by studying the use situation and of finding new and creative ways to use existing technology to influence and challenge the development of technical tools.

Scandinavian end-user participation has primarily been concerned with the involvement of end-users in the early stages of systems development, i.e., during analysis, design and prototyping. The perspective presented this paper is that end-user participation should not stop once a system has been installed, but continue to involve end-users in future enhancements as well [2, 20]. This is addressed in this work by combining the tool perspective [3] and the application perspective [1]. The next section describes how I interpret the tool perspective and the following section how I interpret the application perspective.

TOOLS FOR TAILORING

User Interface Objects

The interface of a computer application is composed of user interface objects. These objects mediate interaction between a user and an interactive system in order to complete work tasks, such as writing reports, tabularizing data, creating diagrams, and making presentation slides. These are typical tasks carried out with today's computer applications. The actual mechanism that transform user

input (keyboard press, mouse action, etc.) to system functionality is the eventhandler. User interface objects with eventhandlers was first introduced in the Xerox Star computer [12].

The transition from using an application to tailoring it can be supported by extending the eventhandler mechanisms of conventional interface objects to allow for additional events to be processed, each being associated with a specific level of tailoring. This tailor-oriented perspective on user interfaces has lead me to reconceptualize the notion of user interface object and to propose an alternative unit of use, the Application Unit.

Application Units

A computer application, in the context of this paper, is seen as a collection of application units [18], and each application unit is defined as a set of orthogonally integrated aspects. The application unit is modelled after the structure of the kitchen appliance shown in Figure 2.

The three aspects of an application unit are: (1) interface, (2) rationale, and (3) mechanism. Interface is the user interface objects and hence the aspect the user is primarily concerned with. Rationale is documentation for *how to use* an application unit as well as *what* its mechanism does and *why* it does it. Mechanism is the program code that makes it do it.

To tailor an application requires modifying one or more of the three aspects of its application units. Access to the various aspects is accomplished by holding down a modifier key (`option/alt`, `shift`, or `ctrl`) while performing the normal interaction gesture (pressing or releasing a mouse button) on an interface object. This is illustrated by the two arrows marked `ctrlMouseUp` and `shiftMouseUp` in Figure 3.

When an aspect has been modified, the other aspects, and the interface between modified aspects may have to be updated. Whereas the latter can, to a large degree, be automated by the computer, the former is a design activity that has to be done manually by the end-user. When tailoring the user interface, for example, the tailor may need to make changes to the program code as well, keeping it compatible with the interface changes, and vice versa. In either case, the rationale may have to be updated. How to link them up with each other again is accomplished by the computer (saving the changes, compiling the code, etc.).

Tailoring the user interface may also require different skills of the end-user than the skills required when tailoring the mechanism. To tailor the mechanism, for example, the end-user needs to be concerned about writing code in an efficient manner as well as effectively utilizing previously written code. On the other hand, to tailor the interface, the tailor needs to be concerned about how to best support user tasks, and user tasks are different from mechanisms (tasks are typically informal whereas mechanisms must follow the rules of a formal grammar).

Each aspect provides a view of the application that requires its own set of tailoring tools. To provide a unified notion of tailoring across aspects, tailoring tools are referred to as *editors,* and there is one editor per *level* of tailoring (each level corresponding to an aspect). The correspondence between aspects and levels is shown in Figure 4.

Tailoring levels		Aspects
Customization	.....	Interface
Integration	.....	Rationale
Extension	.....	Mechanism

Figure 4: Relationship between application unit aspects [18], and tailoring levels [19].

EXAMPLE: TAILORING BASICDRAW

BasicDraw is a tailorable drawing program created by the author. Its drawing functionality is similar in scope to the functionality of small drawing editors found in word processors and presentation programs. It allows the user to create basic geometrical shapes such as rectangles, ovals, and triangles, and to manipulate them by copying, moving, scaling, and rotating them. These tasks are frequently performed by computer users when they create diagrams in written reports and visual presentations The tasks are perceived to be simple by most computer users. It is not simple, however, to tailor a drawing program to adapt it to new needs.

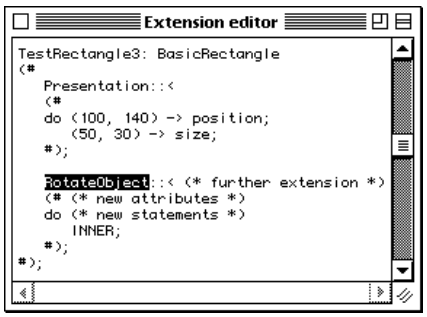

Figure 5: Tailoring program code (mechanism) in an extension editor. New code is added as extensions to old code (see Figure 3). The old code cannot be discarded.

Levels of Tailoring

There are three levels of tailoring supported by BasicDraw: customization, integration, and extension [19]. Customization is the activity of changing the interface aspect by setting parameters of interface objects in a *presentation editor*. Integration is the activity of changing the rationale aspect by writing informal textual descriptions, drawing uninterpreted diagrams, and pasting pictures in a *drawing editor*. Extension is the activity of changing the mechanism aspect by writing new program code in an *extension editor* (Figure 5). None of the old code in BasicDraw can be discarded. The reason for this is safety: to prevent users from accidentally destroying mechanisms that already works in the application.

EMPIRICAL EVALUATION

BasicDraw has been evaluated with end-users in an informal user study in order to assess the usability of its tailoring tools. The following section describes the study (task, users, procedure) and gives a summary of some results. A full description of the results can be found in [20].

Task

The aim of the study was to assess the users ability to locate and tailor a selected set of application units in the BasicDraw application. Two simple but representative exercises were devised for this purpose. The first was to make a "rectangle" into a "square" and the second to improve the "rotate" command. An abbreviated description of the second exercise is shown in Figure 6.

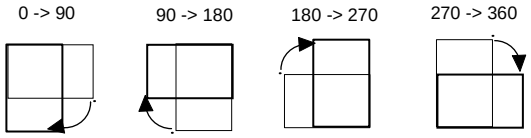

Figure 6: One of the exercises in the experiment was to improve the "rotate" command of BasicDraw according to the diagram shown here (see Figure 3 for the original design). The diagram was supplemented by a textual description (not shown).

The goal of the exercise was not to test the users' ability to create advanced application functionality but to compare their individual differences with respect to tailoring at the different levels (i.e., which levels are easier than others; what tools are needed, etc.). Only the results related to tailoring by extension are discussed in this paper since it caused the most difficulties for the users. A discussion of tailoring by customization and integration are found in [20].

Users

Twelve users participated in the experiment. They were recruited by e-mail sent to all students and some faculty members at the Department of Informatics at the University of Oslo. The majority of respondents (67%) were from the social informatics group. Two were female, ten were male. All users received a compensation of 100 Kroner (about $15) for volunteering. The participants had *some* prior knowledge of programming: each of them had taken an introductory course in object-oriented programming. Some of the users (42%) reported that it had been more than one year since they last wrote a program. Others reported more programming experience: about half of the users (58%) had taken a course in BETA programming.

Procedure

BasicDraw is implemented in the BETA programming language [19], and tailoring by extension required the users to write some program code in BETA. Below is a program solution to exercise II (which was to improve the "rotate" command) created by one of the users (user #9). The exercise required the users to extend the `RotateObject` method of `TestRectangle3`, which is a subclass of `BasicRectangle` (see Figure 4). The amount of program code that each user wrote varied from 1 to 22 lines for each application unit, and each exercise had about 2-3 application units (primarily shapes and commands). Not all applications units required tailoring by extension.

```
TestRectangle3:BasicRectangle
(# Presentation::<
   (#
   do (100, 140) -> position;
      (50,30) -> size;
   #);
   RotateObject::< (* further extension *)
   (# (* new attributes *)
   do (* new statements *)
      (if Angle
      // 90 then
         x + width -> x;
      // 180 then
         y + height -> y;
      // 270 then
         x - width -> x;
      // 0 then
         y - height -> y;
      if);
      INNER;
   #);
#);
```

In the boldface code above the user has copied the if-statement from the parent method (see Figure 3) and created new statements for each of the four conditions of "angle of rotation." Each statement provides some additional mechanism which, upon execution has the effect of moving the rectangle in the direction specified by the diagram when compared to the original design.

The main method used for data collection was "thinking aloud" [7, 14]. Users were asked to "think aloud" and verbalize their thoughts as they interacted with the system to tailor it. The interaction was recorded on video-tape. Before the recording started, the users were given a warm-up exercise to practice thinking aloud. During the recorded exercises both the computer interface and the spoken interactions were captured on tape. The tapes were later analyzed to identify usability problems and to identify patterns of recurrent tailoring behavior. Some of the tapes (the ones judged to be most representative) were transcribed by hand. An excerpt from one of those tapes is reproduced below:

> *Now we are in a lying-box position. When it is rotated it becomes 270 degrees.*
>
> *Let me see, ...*
>
> *What has happened here must be changed by pushing it to the left towards the one we had at 180.*
>
> *And to do that we keep the Y the same, but for the X we subtract the Width.*

X minus Width is put into X.

Then we have the last one. If Angle is 0, then ...

It means that we have been at 270 and we have to "lift" it up again, or subtract Height from the Y.

This excerpt is part of the protocol of user #9, and thus corresponds with the program code shown in the text above. It gives an example of recurrent behavior: "inventing" informal concepts to explain program behavior. These informal concepts (underlined in the above text) were not suggested by the evaluator nor the problem description given to them. Instead, the users invented them to explain how the old system operated and how they wanted the new system to behave.

Although informal, these concepts would almost always correspond (one-to-one) with the formal concepts (program code) they were writing. Part of the above protocol shows that the user is reading aloud as he writes the code in the editor.

I interpret the protocol data to mean that end-users prefer to articulate requirements of computer systems not in terms of a formal specification language but in terms that include a large portion of informality (incomplete and changing terminology), and that the informal elements need to be integrated with formal elements associated with the implementation language. The former interpretation confirms previous studies that show that successful (democratic) user involvement in systems development requires users to have access to tools and materials they can relate to [4]. In addition I have shown that the informal approach can be integrated with a formal approach [20], by viewing a system as a set of orthogonal aspects.

Findings

All twelve users were able to complete the two exercises, which included locating the three aspects of a selected set of application units (menus, menu items, and shapes), and then tailoring the application units at one or more of the three levels of complexity: customization (interface), integration (rationale), and extension (mechanism). Some of the users received help by the evaluator when they were tailoring at the implementation level.

After the experiment, qualitative data was collected by informal interviews and questionnaires. The data indicates that tailoring differs at the various levels: it was more difficult to tailor the mechanism (extension) than it was to tailor the user interface (customization), and the difficulty of tailoring rationale (integration) was somewhere in between the other two.

None of the users reported difficulties when tailoring the interface. The main difficulties users reported when tailoring rationale were how to design it and how to map design content (i.e., the design) to corresponding concepts in the code. These results are further reported in [20]. The main difficulties users had when writing program code are displayed in the diagram of Figure 7. It is a multiple-choice question taken from one of the questionnaires and presents answers to the question of what the users thought was most difficult about programming.

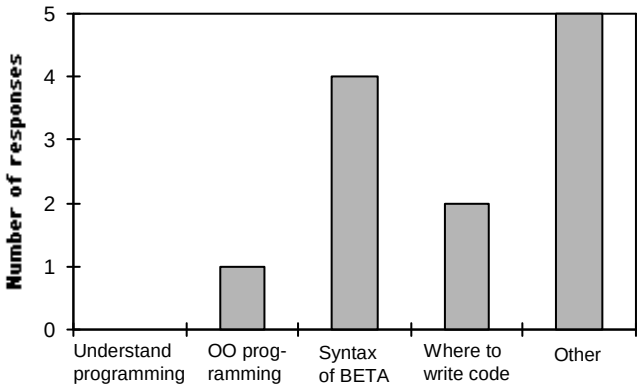

Figure 7: Answers to a question about what was most difficult about programming.

None of the users reported that they thought it was difficult to understand programming. Only one user (8%) reported difficulty with understanding object-oriented programming. Four users (33%) reported difficulty with BETA's syntax, and two users (17%) had difficulty understanding where to write the code. Five of the users (42%) chose the "other" option. Their difficulties were related to understanding the flow of control from superclass to subclass, and of finding out what variables were available and what values they had.

Questions about programming	**med**	**rang**	**N**
9. How useful was it to look at/copy from old code when writing new code? (1 = totally useless; 4 = sometimes useful; 7 = very useful)	6.5	5-7	12

Figure 8: Median with range of answers to a rank-ordered question to measure the usefulness of look at/copy from old code.

The data in Figure 8 are the answers provided by the users to a related (rank-ordered) question. It shows that users thought it was very useful (average score of 6.5/7) to have access to old code in order to look at it and to copy from it when writing new code. The users would actively read and copy from the old code during the exercises that required programming. The code defined the mechanisms they had to work with, and included variables and algorithms from immediate superclasses. This confirms previous studies of software reuse that show that to have access to old code for copy-paste-and-modify is useful during reuse of object-oriented programs [e.g., 13].

DISCUSSION

Is it realistic to think that future users of information appliances will be programmers, albeit programming in the

small? If not, should we still be seeing an application as being partitioned into aspects? These are some of the questions I have attempted to answer in this paper.

First, I consider programming to be among the last tools to resort to when tailoring an application. In a previous paper I have advocated a gradual transition into the complexity of an application, a transition that should only go as deep as getting a tailoring-problem solved [19].

Second, by reconceptualizing the basic unit of an interactive system to be not an application but an application unit, the complexity of an interactive system has been reduced by partitioning it into smaller, self-contained units. The latter was made possible in part by organizing them as three orthogonal aspects that mutually reference each other. Tailoring an application unit will in most cases be much simpler than tailoring an application since most application units are smaller than applications, and more or less independent of each other. This require fewer lines of code to be written. By making application units domain-oriented, a large portion of the code that has to be written can be reused from other application units.

BETA's syntax was perceived by some users to be difficult to master (see Figure 7). Nevertheless, only one user thought that a scripting language would improve end-user tailoring. I interpret this to mean that the dividing line between what can be expected of end-users without any knowledge of programming and what can be expected of end-user programmers is not, as often perceived, related to the distinction between small languages (with a "little" syntax) and full-fledged languages (like BETA), but rather related to the distinction between high-level, task-oriented languages, on the one hand, and low-level, computer-oriented languages, on the other hand. This confirms previous field studies carried out by Nardi [21].

What specific programming language one chooses to use, I claim, should not be a main concern in end-user computing, but rather how well the computational mechanisms provided by the language map to the tasks that users wish to perform. The BETA code available to the users in the experiment was deliberately made to be task-oriented. This was possible since the code defines mechanisms of application units, and application units are, by definition, task-oriented.

RECOMMENDATIONS

My study has shown that end-users with introductory level knowledge of programming are able to participate in further development of computer applications by tailoring them at different levels of complexity without being professional programmers. End-user tailoring brings programming closer to users by integrating tailoring tools into generic applications in a way that makes small programming changes have a perceivable effect on the user. This view requires developers of tailorable applications to reconceptualize applications as collections of aspect-oriented application units.

OPEN ISSUES

Some issues for further investigation are:

- Is it possible to do "deep" tailoring without a programming language?
- Should we aim at higher-level (more domain-oriented) programming languages?
- How can we integrate application units without an application framework?

CONCLUSION

I have reported on the results of a recent Ph.D. dissertation. It proposes a novel approach for conceptualizing computer application (by aspects), and models the smallest usable unit of an application (application unit) in terms of kitchen appliances. I have further built a generic drawing program as a collection of application units and integrated tailoring tools into it. The tailoring tools operate at the granularity of the application unit. Tailoring of application units has been evaluated in an experiment with twelve college-level users. I conclude from the experiment that tailoring below the interface is viable and necessary in order to involve users in further development of computer systems. This require some knowledge of programming, but the learning curve can be lowered by starting from already built application units. Incrementally changing them by simple program extensions will create perceivable effects for the end-user.

ACKNOWLEDGMENTS

The research reported in this paper was conducted as part of the author's Ph.D. dissertation at the University of Oslo. I thank members of the Systems Development group at the Department of Informatics for challenging discussions. I thank Barbara Wasson for useful comments on the first version of this paper.

REFERENCES

1. Bjerknes, G., and Bratteteig, T. The Application Perspective: An Other Way of Conceiving System Development and Edp-based Systems. *Proceedings IRIS-7 Seventh Scandinavian Research Seminar on Systemeering* (Helsinki, Finland, 1984). Helsinki School of Economics, 204-225.
2. Braa, K. Priority Workshops as a Springboard for User Participation in Redesign Activities. *Proceedings COOCS'95 Conference on Organizational Computing Systems* (Calif., 1995), ACM Press, xx-yy.
3. Ehn, P., and Kyng, M. A Tool Perspective on Design of Interactive Computer Support for Skilled Workers. *Proceedings IRIS-7 Seventh Scandinavian Research Seminar on Systemeering* (Helsinki, Finland, 1984). Helsinki School of Economics, 211-242.
4. Ehn, P. and Kyng, M. Cardboard Computers: Mocking-it-up or Hands-on the Future. In J. Greenbaum and M.

Kyng (eds.) *Design at Work: Cooperative Design of Computer Systems*. Lawrence Erlbaum, Hillsdale NJ, 1991, 169-195

5. Eisenberg, M. Programmable Applications: Interpreter Meets Interface. *SIGCHI Bulletin* 27, 2 (1995), 68-83.
6. Eisenberg, M., and Fischer, G. Programmable Design Environments: Integrating End-User Programming with Domain-Oriented Assistance. *Proceedings of CHI'94* (Boston MA, April 1994), ACM Press, 431-437.
7. Ericsson, K.A., and Simon, H.A. *Protocol Analysis: Verbal Reports as Data.* The MIT Press, Cambridge MA, 1984.
8. Escher, M.C. *The Graphic Work of M.C. Escher.* Ballantine Books, New York NY, 1971.
9. Fischer, G., and Girgensohn, A. End-User Modifiability in Design Environments. *Proceedings of CHI'90* (Seattle WA, April 1990), ACM Press, 183-19.
10. Greenbaum, J. *Windows on the Workplace: Computers, Jobs, and the Organization of Office Work in the Late Twentieth Century*. Monthly Review Press, New York NY, 1995.
11. Henderson, A. and Kyng, M. There's No Place Like Home: Continuing Design in Use. In J. Greenbaum and M. Kyng (eds.). *Design at Work: Cooperative Design of Computer Systems.* Lawrence Erlbaum, Hillsdale NJ, 1991, 219-240.
12. Johnson, J., Roberts, T.L., Verplank, W., Smith, D.C., Irby, C.H., Beard, M. and Mackey, K. The Xerox Star: A Retrospective. *IEEE Computer* 22, 9 (September 1989), 11-26.
13. Lange, B.M. and Moher, T.G. Some Strategies of Reuse in an Object-Oriented Programming Environment. *Proceedings of CHI'89* (Austin TX, May 1989), ACM Press, 69-73.
14. Lewis, C. Using the "Thinking-aloud" Method in Cognitive Interface Design. Research Report RC 9265. T.J. Watson Research Center, Yorktown Heights NY, 1982.
15. Mackay, W. Users and Customizable Software: A Co-Adaptive Phenomenon. Ph.D. thesis. Sloan School of Management. Massachusetts Institute of Technology, Cambridge MA, 1990.
16. MacLean, A., Carter, K., Lovstrand, L., and Moran, T. User-Tailorable Systems: Pressing the Issues with Buttons. *Proceedings of CHI'90* (Seattle WA, April 1990), ACM Press, 175-182.
17. Madsen, O.L., Møller-Pedersen, B., and Nygaard, K. *Object-Oriented Programming in the BETA Programming Language*. Addison-Wesley, Wokingham UK, 1993.
18. Mørch, A. Application Units: Basic Building Blocks of Tailorable Applications. *Proceedings of EWHCI'95* (Moscow Russia, July 1995) Lecture Notes in Computer Science 1015. Springer-Verlag, Berlin, 45-62.
19. Mørch, A. Three Levels of End-User Tailoring: Customization, Integration, and Extension. In M. Kyng and L. Mathiassen (eds.). *Computers and Design in Context*. The MIT Press, Cambridge MA, 1997.
20. Mørch, A.I. Method and Tools for Tailoring of Object-oriented Applications: An Evolving Artifacts Approach. Ph.D. thesis. Dept. of Informatics, University of Oslo, April 1997.
21. Nardi, B. *A Small Matter of Programming: Perspectives on End User Computing.* The MIT Press, Cambridge MA, 1993.
22. Norman, D.A. *Taming Technology.* Available as http://cogsci.ucsd.edu/~norman/DNMss/TamingTech.html, 1997.
23. Nygaard, K. Program Development as a Social Activity. *Proceedings of Information Processing 86* (1986). North-Holland, 189-198.
24. Page, S.R., Johnsgard, T.J., Albert, U., and Allen, C.D. User Customization of a Word Processor. *Proceedings of CHI'96* (Vancouver BC, April 1996). ACM Press, 340-346.
25. Simon, H.A. *The Sciences of The Artificial.* Second Edition. The MIT Press, Cambridge MA, 1981.