By Anders I. Mørch, Gunnar Stevens, Markus Won, Markus Klann, Yvonne Dittrich, *and* Volker Wulf

Component technology can provide the reuse option to the end user given a workable model and implementation platform.

COMPONENT-BASED TECHNOLOGIES FOR END-USER DEVELOPMENT

COMPONENT-BASED SOFTWARE DEVELOPMENT (CBSD) INVOLVES multiple roles. Framework builders create the infrastructure for components to interact; developers identify suitable domains and develop new components for them; application assemblers select domain-specific components and assemble them into applications; and end users employ component-based applications to perform daily tasks [7].

There is room for a fifth role in CBSD—that of end-user developers positioned between application assemblers and end users. These end-user developers are able to tailor applications at runtime because they have both domain expertise and technical know-how, but they are not perceived as programmers. They would interact with applications to adjust individual components by tailoring techniques [5], and modify existing assemblies of components to create new functionality [8]. Furthermore, they can play a critical role when large-scale component-based systems have to be redesigned or evolved as a result of changing requirements [1].

We have adopted techniques from software engineering and HCI, going back to their pioneering visions, and interpreted them in a user-oriented direction. Our goal is to provide a components approach to end-user development (EUD). The work we present here addresses

ILLUSTRATION BY HAL MAYFORTH

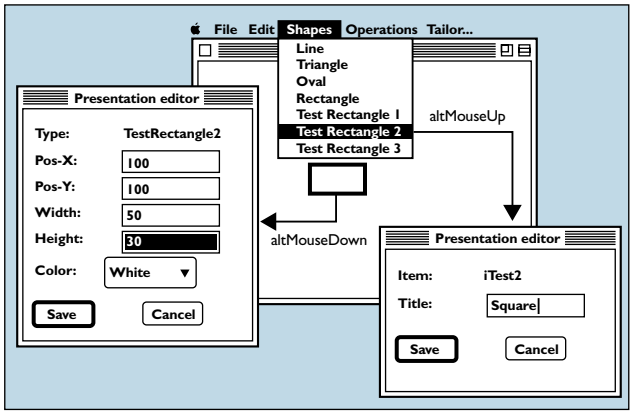

Figure 1. Gentle slope to customization in a drawing program. Direct activation is accomplished by holding down a modifier key while performing the normal interaction gesture on a user interface object. All components (menus, menu items, and graphical shapes) can be customized in the same uniform way. The saved data is stored in an initialization file and reinstated when the program is restarted [5].

how to ease the transition between using an application and tailoring it at different levels of complexity. We propose new metaphors and techniques for selection and assembly of components.

McIlroy [4] recognized the need for component-based systems in software engineering by adopting techniques developed in the manufacturing industry, such as black boxing, subassemblies, and variability. Black boxing would ensure a component had a well-defined interface, built and tested according to a specification that could be documented in a parts catalogue. Subassemblies referred to collections of components that could be combined with other components, and variability referred to the ease with which models or sizes of standard components could be selected [4]. He expected software systems to be constructed by combining reusable, off-the-shelf components, such as transistors, capacitors, and integrated circuits combined on a circuit board.

Kay [2] pioneered the ideas of domain-specific components (GUI objects) and end-user empowerment with computer applications. He made an analogy with the painters' palette and canvas and suggested application developers should provide end users with domain- specific design environments that would empower them to create a wide range of products within selected domains, such as musical compositions and circuit design diagrams [2]. The domain specificity of these environments would allow users to gain better control over their applications because the users' tasks were not automated, but left open-ended for them to explore the domains with their expertise and creativity.

The techniques of black boxing, variability, component-based subassemblies, and design environments have been widely accepted in software engineering and HCI and proved influential for CBSD. For example, JavaBeans components can be integrated by direct manipulation techniques in visual builders embedded in integrated development environments (IDEs), and the parameters of these components can be modified in property sheets. However, these environments are built for professional developers, resulting in the following shortcomings from the EUD perspective:

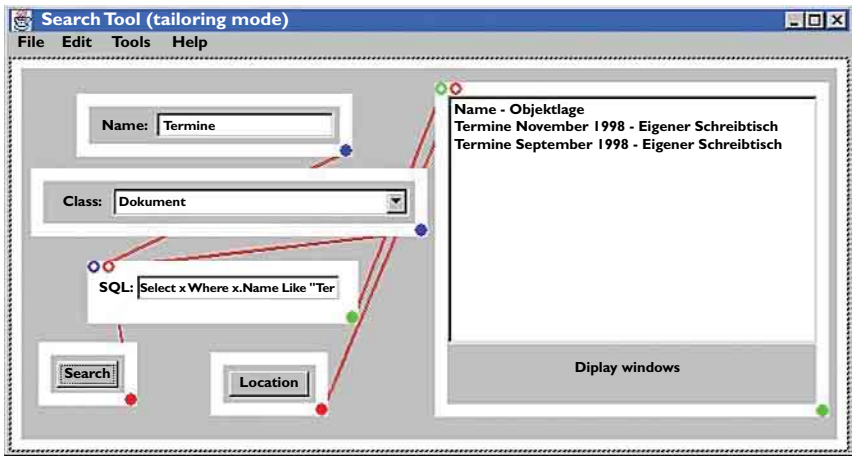

Figure 2. The application here allows the user to search for documents in different sections of a groupware. In tailoring mode both visual and non-visual components are displayed. These components can be grouped into assemblies and integrated by connecting input and output ports. Empty circles indicate input ports and filled circles indicate output ports.

- There is a discrepancy between using an application (runtime) and modifying it with an IDE (design time);
- Metaphors that provide meaningful abstractions for end users, allowing them to break up applications into suitable components and subassemblies are missing;
- There is little support to gradually learn to build or modify software components.

Bridging the Gap between Use and Programming

The learnability issue is addressed by the concept of "gentle slope," that is, to modify a computer application through its user interface end users should only need to increase their knowledge by an amount

proportional to the complexity of the modification [3]. Intermediate techniques, such as parameterization (end-user customization) and integration of high-level building blocks were suggested to bridge the gap between use and programming. Two techniques we have experimented with are direct activation [9] and different levels of tailoring [5]. Direct activation means that tailoring functionality should be accessible from the use context when the need for tailoring occurs (see Figure 1 for an example).

Mørch [5] proposed three levels of tailoring to gradually bridge the gap: customization, integration, and extension. In the context of component-based tailorability, customization means to modify the parameters of existing components, integration means to create or modify assemblies of components, and extension means to create new components by writing program code. The integration of components is more complex than customization. When mastered it provides a powerful middle layer between customization and programming. For example, most IDEs requires application assemblers to know the components' interface names when connecting two components, which are associated with the methods of the components' constituent objects. However, tools to support the integration and reconfiguration of existing components are important to provide in EUD environments. Employing nested component structures can further differentiate this middle layer, with sublayers providing the end users with increasingly more control over their applications [8].

To address these issues, we used the FlexiBeans component model[1] and the corresponding FreEvolve platform for computer-supported collaborative work [6]. This platform provides an application programming interface (API) for integrating tailoring functionality with software components. Based on this API, several visual tailoring environments were built allowing end-user developers to modify an application by reassembling components at runtime without deep programming knowledge [8].

To visualize the ports and to provide meaning to the act of connecting two components the visual component integration formalism was created [8] (see Figure 2). A connection between two components is only valid if the types and names of ports match. We believe this is a viable approach if the

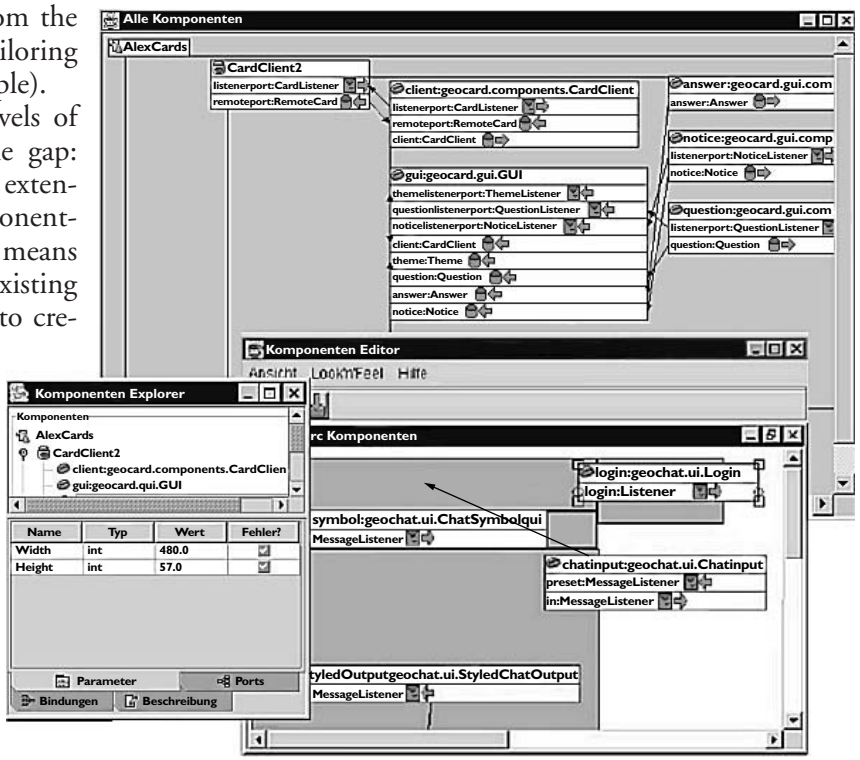

Figure 3. The TailorClient- allows for modifying a composition in three different views. The upper view displays only the visual components and allows for resizing and positioning them. The other views present the interaction between components (bottom right) and an abstract tree view of the nesting structure (bottom left). The views are all synchronized, meaning any change made to the composition in either of the views is immediately updated in the other views.

port semantics cater to both human and computer requirements: aligning with users' understanding of how two components should interact and being programmable on the computer. Understanding the semantics of integrating components in this way can help end users understand the capability of black-boxed components (such as, what they can and cannot do). In order to allow for a gentle slope in this environment, the FreEvolve platform also provides hierarchically nested component structures. Composing higher-level components requires mastering a lower level of complexity than is needed for lower-level ones. Lower-level components can be composed into an assembly and stored as a higher-level component. Higher-level components can be

[1]The FlexiBeans component model is based on JavaBeans. Extensions were made in order to incorporate the specific requirements of EUD. For instance, the FlexiBeans component model allows for typed and named event-based communication. Furthermore, an additional pull mechanism has been created to render the composition of components in a user-friendly manner.

integrated by the same mechanisms as lower-level ones and thus be reused in multiple configurations.

To test the viability of this approach to component integration and EUD several prototypes were developed, including a chat tool and a search tool—a component-based groupware application with an integrated tailoring environment. The tailoring environment is directly accessible from the application by a single mouse click from its use mode. We wanted to see if this environment allowed a seamless transition from use to tailoring. In the tailoring mode all components (whether visible or nonvisible at runtime) as well as the connections between components are visible (see Figure 2). The findings from the study indicate the approach works well when the application has a high ratio of visual to nonvisual components, since the context switch is minimal. Without this high ratio the tailoring interface becomes cluttered with objects and users will have difficulty navigating.

To address the space contention problem we created a new tailoring environment. In the TailorClient [8] assemblies of components behind the screen are presented by means of additional views (Figure 3). Furthermore, the composition can be checked for integrity. We have integrated two integrity checking approaches—one is based on constraints attached to the components; the other analyzes the events passed among the components of an assembly [8].

Challenges for Future Work

Component-based technologies are a promising direction for further work in EUD as it allows end users to tailor existing applications by assembling high-level components. However, EUD must be planned for as part of the CBSD life cycle. Framework and component builders must develop functionality designed for tailorability, that is, components supporting direct activation of tailoring tools, that are domain-oriented, and allow experimentation without hazardous side effects. Customization operations should allow sizes and models (variability options) to be user-defined for the most frequently used components. Finally, providing one intermediary level of technological support, such as high-level and nested components, might not be enough to create a gentle learning slope. We anticipate multiple levels of intermediate techniques to flourish in the future. The following issues represent some areas for future work:

- How to select components not provided with the application environment without requiring the user to write programs or move outside of the use context;
- How to combine different levels of tailoring, such as combining customization and extension with the integration of existing components; and
- How to support cooperation among different users who have different qualifications, skills, interests, and resources to carry out tailoring activities. C

References

1. Dittrich, Y., Lindeberg, O. Designing for changing work and business practices. *Adaptive Evolutionary Information Systems.* N. Patel, Ed. Idea Group Publishing, Hershey, PA, 2002.
2. Kay, A.C. Microelectronics and the personal computer. *Scientific American.* (Sept. 1977), 231–244.
3. MacLean, A., Carter, K., Lövstrand, L. and Moran, T. User-tailorable systems: Pressing the issue with buttons. In *Proceedings of the Conference on Human Factors in Computing Systems.* (Apr. 1990), 175–182.
4. McIlroy, M.D. Mass produced software components. *Software Engineering—NATO Science Committee Report.* P. Naur and B. Randell, Eds. (Garmisch, Germany, 1968), 138–155.
5. Mørch, A.I. Tailoring tools for system development. *Journal of End User Computing 10,* 2, (Spring 1998), 22–30.
6. Stiemerling, O. Component-Based Tailorability. Ph.D. thesis (unpublished). University of Bonn, Germany, 2000.
7. Vitharana, P. Risks and challenges of component-based software development. *Commun. ACM 46*, 8 (Aug. 2003), 67–72.
8. Won, M., Stiemerling, O. and Wulf, V. Component-based approaches to tailorable systems. *End-User Development.* H. Lieberman, F. Paternò, and V. Wulf, Eds. Kluwer Academic, 2004 in press.
9. Wulf, V., Golombek, B. Direct activation: A concept to encourage tailoring activities. *Behaviour & Information Tech. 20,* 4 (2001), 249–263.

Anders I. Mørch (anders.morch@intermedia.uio.no) is an associate professor of informatics at InterMedia, University of Oslo, and is an adjunct professor in the Department of Information Science and Media Studies at the University of Bergen, Norway.
Gunnar Stevens (stevens@fb5.uni-siegen.de) is a research associate in the Department of Information Systems at the the University of Siegen, Germany.
Markus Won (won@cs.uni-bonn.de) is a research associate in the Department of Computer Science at the University of Bonn, Germany.
Markus Klann (markus.klann@fit.fraunhofer.de) is a research associate at the Fraunhofer Institute for Applied Information Technology (FhG-FIT), Sankt Augustin, Germany.
Yvonne Dittrich (ydi@itu.se) is an associate professor in the Department of Design and Use of IT at the IT University in Copenhagen, Denmark, and the Department of Software Engineering and Computer Science at the Blekinge Institute of Technology, Sweden.
Volker Wulf (wulf@fb5.uni-siegen.de) is an associate professor in the Department of Information Systems at the University of Siegen, and a senior researcher at the Fraunhofer Institute for Applied Information Technology, Sankt Augustin. He also heads the International Institute for Socio-Informatics, Bonn, Germany.