Tailoring by Integration of Domain-specific Components:  
The Case of a Document Search Tool

Volker Wulf, Helge Kahler, Oliver Stiemerling, Markus Won

Volker Wulf  
University of Siegen  
Institute for Information Systems  
Germany  
wulf@fb5.uni-siegen.de

Helge Kahler  
International Institute for Socio Informatics  
Germany  
helge.kahler@iisi.de

Oliver Stiemerling  
ecambria systems GmbH  
Germany  
os@ecambria-systems.com

Markus Won  
University of Bonn  
Department of Computer Science III  
Research Group on HCI and CSCW (ProSEC)  
Germany  
won@cs.uni-bonn.de

keywords: tailorability, component-based development, evolutionary prototyping, file search

ABSTRACT

In this paper we describe the evolutionary design and implementation of a search tool for files in shared workspaces used within an off-the-shelf groupware product. The design is based on the assumption that a useful generic search tool must be highly tailorable which is achieved by applying an innovative software architecture allowing to assemble components during runtime. In order to understand people’s searching methods in shared workspaces and to support the design, we employed interviews and workshops with users as well as a field test to understand the users’ needs. During the design process we developed a series of prototypes that were then evaluated by office workers. Consequently, the process described and the lessons learned extend from searching in files as a case via tailorbility of software as an answer to the resulting requirements to component architecture as a way to implement this tailorbility. The results derived from the treatment of these interrelated aspects constitute the core and value of this paper.
INTRODUCTION

In many repository-based systems (file storage, WWW, etc.) searching is one of the most important activities. Especially if these repositories are organized and used by more than one individual, searching needs to be supported technically. While the issue of searching for files in the PolTeam project¹ was the starting point of our research, we considered tailorability of software to be one of the possible solutions for multiple and changing user requirements for a software supporting the search for files within a group’s workspace. On the level of software technology, ‘components’ appeared to be a feasible option for implementing tailorable applications.

This paper is an account of our work on a component-based tailorable search tool and describes the lessons learned. While one might argue that such a concept fails to focus on a single key issue, we believe that our case benefits from the intertwinement of these three levels. They deserve and require the joint treatment in the paper that we addressed to them in the process. Our research was guided by two interrelated questions: The overall question how can we design for searching in groupware is concerned with software-technical as well as procedural aspects and issues of the user interface and the underlying functions. The more specific question how well are components suited for runtime tailorability relates to a particular solution for designing for searching in groupware. The given answer does not only contribute to the overall question but constitutes a value of its own for a range of applications.

The goal of this paper is twofold: Firstly, it is descriptive - we describe the longsome process of different steps in developing a search tool used in a groupware setting. Secondly, the paper is prescriptive in that it reflects on the process and depicts the lessons learned. By following these two lines we aim at providing a deeper understanding of what we did - of what relevance it has for searching, tailorability and component architectures as well as their relation and we aim at providing support for others facing similar design challenges.

The step-by-step development of the search tool is accompanied by the participation of users and accomplished in an evolutionary way by implementing several consecutive prototypes of search tools, each based on empirical and theoretical insights. The empirical insights have been derived from the evaluation of the previous version of the search tool as well as from empirical work conducted before the introduction of the current version. The theoretical insights stem from related literature where their relation to our problems sometimes only became clear during the process.

The structure of the paper reflects this process: After an account of the state of the art in searching, tailoring, and component architectures, the process of the development of three search tools is described in chronological order including methodical issues, like the form of user involvement chosen, technical issues and empirical results. The paper is concluded with a discussion of what we have learned and answers to our research questions.

STATE OF THE ART

In this section the state of the art of searching for files, tailoring software and component-based software architectures is provided.

¹ The PolTeam pilot project dealt with the groupware support of distributed governance as a reaction to the relocation of parts of the German government from Bonn to Berlin.
Searching for Documents

Searching in computer systems is one of the most fundamental tasks that has to be performed while working with a computer. In a distributed system, which is used for collaborative work on jointly used and changed documents, searching for files as well as searching within the files is crucial for the system’s success, mainly because good search results are vital to handle the huge data amount of such systems. Within an individual workspace well-defined organizational guidelines can help to cope with data repositories. Modern file systems offer different techniques for organizing data. In collaborative settings such guidelines are hard to manage, they have to be negotiated. In many cases users are tempted to follow their individual working needs rather than compromises found by the group. Then technical support by effective search engines, being flexible enough to support diverse ways to search, is needed. Another important issue in the discussion on collaborative work is to keep in mind that the success of any searching activity is highly dependent on the collaborative working experience of the people performing the search and of the respective software settings, particularly access control.

Several empirical studies deal with file organization in the electronic office (cf. Malone 1983; Suchman and Wynn 1984; Barreau and Nardi 1995; Rao et al. 1994). These studies observe the organization of electronic documents from a single user and static perspective. In recent years, for many people the production of documents became a collaborative activity supported by networked computers (cf. Wulf 1997). This constitutes a particular need for research on aspects of the organization and search of files in shared workspaces.

Tailoring Software

Our early experiences with the search tool in the POLTEAM setting showed a wide range of requirements from different persons in different work settings using the system. Meeting different and even mutually exclusive requirements can be achieved by making the used software tailorable. Tailorability is a software attribute that allows changing certain aspects of the software to meet different users’ needs.

Tailorability is a desirable software feature for several reasons:

- The diversity of requirements posted by different individuals and organizations must be taken into account when implementing, buying, or using generic software that is supposed to fit different settings. What may be appropriate for a Federal Ministry with hundreds of employees might not work at all for a shoe-selling small enterprise using a software system.

- The uncertainty about the exact work practices and procedures even in the perception of the workers makes it necessary to leave room for alternative ways of performing tasks (Trigg 1992). In many cases there is not only one way to perform a certain task, as each user has a different way of working with a software.

- The dynamics of individual and organizational work requires software to change over time. The structures of work organization and collaboration may vary considerably in relatively short time periods.

Tailoring software encompasses different dimensions, e.g. the initiator, actor, object, aim, time, and scope that should be considered already at the time of system design (cf. Stiemerling et al. 1997). It is widely agreed that tailorability is one of the major future challenges in the design of interactive systems (e.g. MacLean et al. 1990, Bentley & Dourish 1995, JCSCW 2000). Henderson and Kyng (1991) consider tailoring to be an activity that
continues design in use. All of the contributions stress that the discussion about tailoring should not only be lead in terms of technical measures, but that tailoring software is an activity that is deeply rooted in personal habits and preferences as well as socio-organizational circumstances and dynamics.

A closer look at organizational aspects of tailoring is taken in some papers dealing with the sharing of tailored files (Mackay 1990; Nardi 1993). Trigg & Bødker (1994) point out the fact that the possibility to exchange tailored artifacts can have a standardizing effect. While these findings are not explicitly related to groupware, they involve group activity to tailor software used by a group. The papers emphasize the importance of local experts, who know the work practice well enough to provide adequate tailoring. Carter and Henderson (1990) claim the necessity for a “tailoring culture” within an organization because tailoring not only poses technical problems, but should be considered as a relationship rather than a property. MacLean et al. (1990) go beyond observing only by introducing the “Buttons” system into part of an organization by means of which tailored objects can be sent around and modified. The above-mentioned diversity of requirements poses a particular challenge for tailoring software to a group’s needs: on the one hand the diverse requirements of the different members of the group must be somehow combined or moderated into a convention or standard for group usage; on the other hand it should still be possible for a group member to tailor within the limits of the convention.

Nevertheless we have to point out the disadvantages of tailored software. First of all individualized software is not as easily maintainable as standardized systems. Every customization performed hinders the software support within large organizations. Furthermore, the acquisition of software becomes harder as colleagues are using differently tailored tools. Different instances of a highly tailorable software can differ to a great extent. Thus, the iterative learning process taking place in the daily work may be hampered.

**Component Architectures**

In order to put tailorability into practice software architectures have to be developed that allow for runtime flexibility to avoid a breakdown in the work processes of work. Since some time, component-based architectures are available for software developers (e.g. with the JAVA BEANS model). Our research focuses on the question whether specific component-based architectures can enable end-users to modify existing software in order to fit their needs or create new functions by assembling existing pieces. In this way component-based architectures should become a means to alleviate the climbing of the steep “tailorability mountain” (MacLean et al. 1990; Wulf and Golombek 2001).

Component oriented programming in general (see Szyperski and Pfister 1996) is motivated by the successes of classical engineering disciplines, like electronics or mechanical engineering, have had with building complex artifacts from standardized components (e.g. transistors, resistors, cogs, or screws). Taking into account this motivation, a software component can be defined as “a unit of composition with contractually specified interfaces and explicit context dependencies only. Components can be deployed independently and are subject to composition by third parties.” (Szyperski and Pfister 1996, p. 130).

Components have been successfully employed to support the design of graphical user interfaces. Application builders like BORLAND JBUILDER (component model: JAVA BEANS, see JavaSoft 1997) often provide generic visual design elements (e.g. buttons, text-boxes, combo-boxes), which are configured and composed during the design
process to yield domain-specific applications. The notion of components, however, has been applied to areas of software engineering other than GUI-design, as well.

If the architecture consists of multiple layers of nested components (hierarchical component architecture), tailoring operations are possible at several different levels of abstraction and complexity. Components on the higher levels of the hierarchy could be closer in semantics to the application domain (e.g. the bookkeeping component of a business software package), while components further down the hierarchy could be more technically oriented (e.g. the TCP/IP-protocol component). Thus, a hierarchical component architecture could provide appropriate tailoring levels for both, a bookkeeper and a system administrator.

In our work (Stiemerling 2000, Stiemerling and Cremers 1998, Won 1998) we are investigating the use of components for tailorability of complex and distributed applications after initial development. Component architectures are quite attractive for tailoring, because they support a number of different tailoring interfaces, from simple parameterization (Henderson and Kyng 1991), over visual programming (Nardi 1993) to programming by modification of examples (Nardi 1993, Mørch 1997). Particularly runtime composition by users or local experts can support tailoring activities, because its effects are immediately visible and there is no need to engage into activities like compiling - generally performed by experienced system administrators only – or rebooting a machine, which is often considered to be a breakdown of the flow of work. Using standard components in combination with domain-specific component frameworks then could serve as a basis for a domain-specific language (DSL 97).

**POLITEAM: THE CONTEXT OF THE SEARCH PROBLEM**

The POLITEAM project was a collaborative software development project in which the target organizations required technical support for distributed collaboration. The main function of the POLITEAM system was to supplement paper work processes with electronic work processes in a government ministry. To accomplish this, POLITEAM offered a shared workspace, electronic circulation folders and e-mail functionality (cf. Prinz and Kolvenbach 1996). An already existing groupware system (LINKWORKS by DEC) was chosen and adapted to specific user and situation requirements (see Klöckner et al. 1995). A collaborative and evolutionary approach was used in the design, allowing modifications to be made over time (cf. Wulf 1997).

Obtaining the user requirements was accomplished with the help of users at two different government organizations: a German Federal Ministry (referred to as FM in the paper), and the State Representative Office of a German state (referred to as SR in the paper), both located in Bonn. In the FM, in the department where the system was installed, we found varied employee roles: one unit leader, six ministry employees (responsible for specific content areas of the ministry), and three typists in their own service unit. In the SR Body about 30 people represented the interests of their state in the federal government’s legislation process. The organizational structure of the body mainly consisted of sections that represented state ministries. Most of the sections just consisted of the section manager. Before the introduction of POLITEAM, these sections were supported by three typists, who worked at the typing pool. Like in the FM, the employees collaborated using the shared workspace and e-mail.
Among the multitude of challenges that POLITEAM had to face, dealing with such a complex issue as the introduction of groupware in large administrations, the searching for files was of particular interest. This is due to the fact that the administrative work is centered on text documents that are often processed by different people. While this can sometimes be handled by sequential workflow mechanisms, the collaboration often is too unstructured for this rigid kind of support. Therefore the possibility for electronic search offers a benefit particularly for the less structured collaborative parts of administrative work. Besides, searching files electronically within, and possibly even beyond common workspaces, raises many questions regarding access rights and the ambiguity of visibility and privacy. In order to meet the resulting requirements, we developed a tailorable search tool for the participating users in the course of the POLITEAM project. The search tool has been developed in three major steps: Taking a participatory and evolutionary approach, we first developed Search Tool 1 using Microsoft Visual Basic (VB) as a means for prototyping to meet general needs with a default setting for a generic search tool (cf. Kahler 1996). In a second step we used the evaluation of Search Tool 1 for a reimplementation in Java. The resulting Search Tool 2 allowed end-users and organizations to tailor the tool to their particular search requirements (cf. Won 1998). Finally, Search Tool 3 (cf. Engelskirchen 2000) provided mechanisms to support collaborative tailoring activities and the exploration of the tailoring functionality. Figure 1 gives an overview on the design process.

Figure 1: Steps in the process of designing several versions of our search tools

SEARCH TOOL 1: INVOLVING USERS

The basic version of LINKWORKS included a tool that allowed the user to search for any object independent of its actual location within the system. Discussions with users revealed that this search tool was not well enough
designed to be used by our two government organizations involved, so we decided not to use the original search tool in our partner organizations, but to redesign it.

**Requirements from the Field**

In order to identify the basic functionality of a groupware search tool and possible enhancements, we conducted 10 interviews with interview partners from four different organizations, lasting about 30-45 minutes each. Besides the interviews, in this first step of the search tool’s redevelopment, workshops discussing searching were held with a group of users in the FM. Several aspects of a group-related tool for electronic search in a shared workspace proved to be of importance:

**Privacy issues**

On the one hand users wanted to have full access to all the documents stored in the system, on the other hand they demanded that “their” documents should not be accessible for other users. In the role of an actively searching person, the interviewees pleaded for an almost unlimited access to electronic search, arguing that this was helpful and necessary for collaboration and adequate for team work. They were much more privacy alert when they took the role of a person affected by someone else’s electronic search. In this case, many of the interviewees said they felt uncomfortable knowing that everyone could have an insight in their folders and considered this as an unwanted intrusion. This “I want to see yours but you may not see mine” attitude obviously requires technical and social mechanisms for mediation.

Beyond these individual preferences, the informal ways of collaboration are specific for an organization. In the FM searching on other people’s desk was a taboo except for the head of department and one other person who was responsible for the registry. The head of the department of the FM mentioned, that considering the different organizational standards about standardization, there is a need to be able to set the borders for tailorability for the search tool so that the tailoring of the tool didn’t exceed reasonable limits. This, however, is the key point for tailorability: What reasonable means cannot be foreseen when specifying a generic software particularly for work in groups?

**User interface**

The users had different preferences about what to do with found documents (e.g. create a copy or create a link). Furthermore they did not agree upon the way the search results should be ordered (e.g. by name or by date). Considering this, there was a need to think about options and limits for individual and organizational tailorability of the search tool.

**Implementation**

We employed Microsoft Visual Basic in order to be able to quickly develop the first version of the new search tool. The input dialog of Search Tool 1 considers the most frequent requirements from the interviews and workshops. There are several input fields that allow the user to specify the search request. The output dialog of Search Tool 1 takes privacy aspects into account by dividing the output into files found on one’s own desk, someone else’s desk, or in the registry. A link from the found items can be created in one’s own workspace.
This first prototype did not incorporate all the demands of the interviewees and the participants of the workshops for two reasons: First, we wanted to force the privacy discussion, so we only split the search result into three groups. Second, we just allowed creating links on the found objects. The first prototype of the search tool was presented only a short time after the introduction of LINKWORKS and most of the users had no experience with groupware systems or even computer systems at all until then.

**Evaluation**

Search Tool 1 was now presented to ten project members (users and user advocates) during a workshop. The discussions verified existing requirements and helped identifying new demands. Besides, our user advocates, who supported the pilot users during their daily work, added some requirements to a new version of the search tool.

**User interface**

Most of the users asked for a spatial separation of the search request panel and the displaying components within one window so they could see their search request while viewing their search results. In Search Tool 1 we used two windows, one allowing the input of several search criteria and another displaying the results of the search. Therefore, the new search tool should have at least two components: an input component and a display component, which have to be displayed in the same window.

The users had no common idea about the representation of the result. Some users wanted to have exactly one window where all the objects found in the system were shown. Others would have liked the search result to be divided into several subsets. The found objects then should be sorted by certain criteria. For instance, some users wanted to see the documents in two separated windows: one for documents which lay on their own virtual desktop and another for the rest. Others could imagine that the documents were ordered by name or date of creation.

**Search criteria**

All users had different preferences how to search for an object. For instance, some users searched for a document by its name, others by the date of the last change or they searched for the document’s owner. Therefore no simple way to decide, which of the possible search criteria should be asked for by our search tool, could be determined.

As tailorability is one way to meet these different requirements, we decided to implement a new version of our search tool that was supposed to offer several aspects of tailorability. Therefore, Search Tool 2 was implemented by using component-based architectures.

**SEARCH TOOL 2: TOWARDS COMPONENT-BASED TAILORABILITY**

In this section, we describe how we used a component architecture to create a tailorable search tool. The idea was to use the established work on components and wiring diagrams to connect these in order to enable users and local experts to create, modify and test search tools during runtime.

As described in the section *State of the Art*, component architectures are well-suited for the design of tailorable applications. Therefore, the implementation of Search Tool 2 was based on the JavaBeans component model.
Furthermore, we developed a runtime and design environment in which compositions can be changed dynamically during runtime by adding or removing components or changing the connections between them.

In the case of Search Tool 2, a layered architecture (see section State of the Art) is used for experienced users to develop their own search tool by selecting and combining up to about twenty components, whereas a beginner would take just two nested components which can contain many atomic components and combine them to a simple search tool. This approach is described below in more detail.

**Introductory Workshop**

Our experiences from Search Tool 1 provided already quite a good idea about searching in groups. With respect to our aim of providing components for building different search tools, we needed a better understanding of what a reasonable deconstruction of a search tool into components would be. This was one of the topics of a full-day workshop (workshop 1 of Search Tool 2) we held with 9 persons from the FM and SR. The analysis of several other search tools had shown, that a first approach to a composition would consist of two distinguished parts related to the chronological sequence of searching for electronic files in a shared workspace. First, a search is performed according to a specific search request, then the results are presented and can be used for further work. Most of the common search engines are divided similarly into two parts. The workshop confirmed this and provided further hints, particularly for different in- and output switches taking into consideration e.g. age and name of files.

The division into two parts, seen from the user’s perspective, results in the division into three parts that we provided: the latter can be deducted by looking at the three types of components described below. Here the output components are distinguished from the so-called flow components that are used to perform a search and to do some work on the search results, like splitting it into two output streams (switch) or getting some more attributes (e.g. date of creation, last change) of them.

**Implementation**

Search Tool 2 has been developed by taking into account the diverse requirements that evolved during the above mentioned workshop. As a basis for our implementation, we chose the JAVA BEANS component model (see Java Soft 1997) that allows for dynamical binding of components.

**The components**

Based on the results of workshop 1 of Search Tool 2 described above, we decomposed Search Tool 1 into several different components that have been divided into three categories: input components, data flow components, and output components (cf. figure 2 and 3).
The top of figure 2 shows some of the implemented input components. These components are used to specify the search and to trigger an action. For instance, one can enter the name of the objects that are to be found and start a search.

The bottom of figure 2 depicts some flow components including the search engine. This component has to deal with the input parameters generated by the input components and to build up the search results. The search engine connects the search tool with the LINKWORKS database via the application-programming interface. It transfers the users’ inquiry to the database and receives a list of retrieved documents. Two other components, that can be used to prepare the visualization of the search result, are the name switch component and the location switch component. These components are used, if the result of a search has to be split respecting a condition that has been specified beforehand (“+” meaning condition is true). The name switch divides the resulting files by their names (e.g. starting with A-M or starting with N-Z). The location switch divides the resulting files by their location within the system (own desktop vs. other desktops). In order to get a more refined presentation of the result, one could imagine many more different switches.

Figure 3 shows some output components. The display window shows the files found. The user can select, which of the document’s attributes are shown and how the retrieved documents are ordered. Other components, that can be used to visualize the search result, are the counter component (“Documents found” – added in Search Tool 3
shows the number of found objects and serves to get some information about files on others’ desktops without showing any of their attributes) or the info, link and copy components that allow to show additional information, create a link to the file, and to copy a found file. In the following, we will describe the composition of a search tool using these components and we will illustrate their interaction. For the sake of simplicity, we will now concentrate on six atomic components: start button, name input component, search engine, location switch, result counter and result list (see figure 4).

![Figure 4: Simple search tool example](image)

To connect the components with each other we implemented “wiring-instructions” that define how the application is to be composed. In order to compose the components, the tailoring environment allows for wiring operations that support connecting two different types of ports. Empty circles indicate input ports, full circles output ports. In order to support users in wiring the components appropriately, input and output ports, that can be connected, are presented in the same color (in this paper, they are shown in the same gray scale). The semantics of the components are understood best by regarding the simple example in figure 4.

The search engine is triggered by the start button. In order to specify the search query, the value is entered in the name input component. The search results are fed into the location switch that is parameterized to channel all documents found on one’s own virtual desktop into the right result list, while the number of documents found elsewhere, is shown in the counter window on the left (the parameterization of result switches, buttons, and the search engine is not shown here).

A more complex example using the layered architectures is given below (cf. figure 6).
The runtime and tailoring environment

We developed a prototype of an integrated runtime and tailoring environment that serves as basis for the deployment of component based application (see figure 5). These applications are defined by a set of implemented components.

![Search Tool](image)

Figure 5: The integrated runtime and tailoring environment in runtime mode

When the application is started within the environment, the default wiring-instructions are read first. According to these, the necessary components are instantiated and connected. For a more detailed discussion of the tailoring environment see Won (1998). Here we want to abstract from these technicalities and focus on how the environment is perceived by the end user.
If the application is running on a window-based operating system, on a regular basis (the whole environment is implemented in Java and thus platform independent), nothing out of the ordinary should be visible, except some way to leave the “use-mode” and enter the “tailoring-mode”. If the user decides, that the application needs tailoring, he or she enters the “tailoring mode”. Figure 6 shows the tailoring-mode for one specific instance of the search tool. Here, we want to focus on the general concepts of the tailoring environment. The little dots at the edges of the components represent the ports, i.e. the interfaces for interaction with other components. As described above, the search tool employs a flow-oriented metaphor (data flow through the application), we have input ports (empty circles) and output ports (full circles). The lines indicate the way the components are wired to each other.

The user now can do everything but change the implementation of the components that are delivered in binary form. He or she can delete components, instantiate new ones (by choosing a component from the toolbox menu), change the wiring, or change the hierarchical structure. These operations ensure flexibility of the approach. Whether the flexibility offered is satisfying, of course, depends on the set of available application components.

**Evaluation**

In order to evaluate the design of the component-based search tool and its integrated runtime and tailoring environment, we held another workshop (workshop 2 of Search Tool 2). Here, we were mainly interested if the
users were willing and able to handle the tailoring functions of the new environment and if the existing version of the component-based search tool could satisfy the different user requirements.

This workshop was held at the University of Bonn’s research lab. Eleven participants joined the workshop. Four of them were employees of the SR – a section manager, an administrative clerk, a secretary and a clerk who provided local support to other users. Moreover, three user advocates – two working with the SR and one working with the FM – participated in workshop. The other participants were members of the PoliTEAM project involved with the design of the search tool. The discussions that occurred during the workshop have been documented by the project members and transcribed. The quotations presented in the paper are taken from this transcription and translated from German by the authors.

In the beginning of workshop 2 of Search Tool 2, one of the user advocates gave a short presentation. Comparing Search Tool 1 with the new one, he gave a survey on the new functionality. When introducing the integrated runtime and tailoring environment, the user advocate referred to the “Lego”-metaphor. Then one of the designers gave a presentation on different search tools and the tailoring environment on a computer, where a LinkWorks client was installed. The search tool that was active at the beginning of the presentation was a simple tool that just allowed searching on ones own desktop by specifying three search attributes. Activating the tailoring menu, the users could select among three different alternatives of the search tool. These alternatives had been assembled beforehand and were represented in the menu by a term that tried to express their features. From this selection menu it was possible to switch into the tailoring environment where a new search tool could be tailored by connecting the existing components. After the presentation the four users from the SR were asked to apply the tailoring environment and build a new search tool by themselves. All of the four users were willing to experiment with the tailoring environment. Receiving some support by the project members, they were able to construct search tools of different complexities. In the following, we will give an overview on the aspects discussed during the workshop.

Understanding component architectures

As described above, Search Tool 2 consists of several components as well as a runtime and tailoring environment. This approach was clear to all users. Additionally, even the hierarchical constructions were understood without any problems by more experienced users.

Technical issues

Looking at the functionality provided by the components, the users wished for a couple of further design requirements during the workshop.

- Full text search

  Concerning the input attributes for the search engine, the users asked to be able to search for arbitrary words inside the documents. Often, they could not remember any of the given document attributes but a key word inside the text.

- More expressive display components
The users also asked for a new type of display window for objects found. Some required that this component should display the location of a found object.

- **Showing the content of the document**

Having found objects the users required being able to access them directly. In the prototype as well as in the original search tool they can just create a copy of or a link to such a document. Links and copies have to be accessed via another window. This leads to additional efforts in interacting with the system.

- **Switches**

The prototype offered just two switches: one to distinguish between the different desks a document was found on, and one to distinguish between different object names. Appreciating the concept of separate display windows, the users required further switches.

The section manager who has to handle similar issues periodically was asking for a switch that would display the documents which are older than three months in a window separated from a window that shows documents which were written more recently. He also asked for a switch that would allow displaying objects in different windows depending on the question in which of his own folders these objects were found. He said that he was often searching for a document referring to just one of the political areas he is responsible for. As he stored such documents in one folder with corresponding subfolders, he just wanted to see documents found in that folder. Concerning the electronic registrar, one of the administrative clerks suggested to be able to distinguish between the folders of the Bundestag (first chamber of parliament) and the Bundesrat (second chamber of parliament). Moreover, one user suggested that it would be nice to distinguish between those documents that are on one's own desktop and those that are just accessible via document sharing.

- **More expressive descriptions and additional context help**

The users' tailoring experience during the workshop generated the following requirements to improve the tailoring environment. First of all, the users asked for a better description of the different components they could select from. In the prototype the developers had given the components names that were hardly comprehensible to the users. Furthermore, the users asked for more appropriate names and for icons in the select boxes that were supposed to symbolize the meaning of individual components as well as whole search tool alternatives.

Apart from a more intuitive naming the users requested a context sensitive quick-info delivering more comprehensive explanations about the behavior of individual components or search tool alternatives. Moreover the users would have liked a textual explanation popping up as soon as they touched one of the input ports of a component, to indicate which other component could be connected to this port.

- **Context sensitive behavior**

As the select box displaying all components available in the system is very complex, the users had problems localizing the appropriate components during the workshop.
Collaborative issues

In the course of the workshop some users started discussing about how to carry out the tailoring tasks collaboratively. Circumstances requiring searching typically involve time pressure and do not leave time for tailoring to most users. The user providing local support was very enthusiastic about the tailoring environment. During the workshop the users already discussed how the tailoring work could be divided among them. Pointing to her colleague who provided local support, the administrative clerk suggested the following division of labor: “The alternatives are good for us. The assembly-mode is for you.” Assuming such a division of labor, the colleague providing system support required an additional tool to distribute newly assembled search tools among the users. He argued that his job would become much easier with such a tool.

Exploration

Another issue that arose during the discussion, was the necessity to explore newly assembled search tools. The user acting as local system support asked to be able to test newly created search tools: “I need to know whether these things do what they are supposed to do.” Nevertheless, the exploration of a tool that involves searching documents owned by others can cause disturbances to the other users. A statement of the secretary made clear that she would carefully select those users who would accept such a disturbance.

Even tolerant colleagues will probably not accept permanent interruptions due to other people’s testing. Therefore, a test environment allowing users to explore their newly assembled or selected search tools without affecting other users would probably encourage tailoring, especially in collaborative settings. Such a test environment should also cope with the users’ worries to break existing artifacts. In that sense, the user providing local support asked for an “undo”-command to be able to recover the old search tool if he should have made a mistake when modifying it.

Summary of evaluation

The evaluation of Search Tool 2 showed the need for tailoring features in a search tool in general. Groupware settings are that complex that different users have different requirements according to their position, experience, and work style. From the technical point of view this flexibility can be reached by the approach of component-based tailoring. However, our users understood the basic ideas of component-based architectures but they were not able to tailor their tools on the spot and they were even less capable of assembling new tools. This is caused by the complex user interface. Workshop 2 of Search Tool 2 was the source of many new ideas that can help developing a new more easy-to-handle tailoring interface and additional functionality facilitating tailoring. The last issue we focused on was to explore if the components, we had implemented so far, could be used for designing all the required search tools. Due to the component-based approach allowing for building software out of mostly independent modules, users were able to identify missing components. Furthermore, those components could be added due to domain-specific requirement without changing the existing ones.

2 This approach can be seen in other collaboration contexts, too. In (Kahler 2001) cooperative tailoring of a word processor is described. This work focuses on the exchange of tailored artifacts. So there are experienced users which fine-tune applications and others who can use them.
SEARCH TOOL 3: FIELD TEST

Based on the experiences with Search Tool 2 we built Search Tool 3 that not only contained enhanced functionality but was used within the SR for several weeks (cf. Engelskirchen 2000 for a detailed description of all aspects of Search Tool 3).

Setting

For Search Tool 3 the component language and the tailoring environment have been extended. In the following we carried out a field study with three users in the SR. Other users of the SR were asked to provide search permission on their documents eventually needed by the participants of the field test.

We presented the new search tool environment in a workshop in which the three users, one user advocate and three designers participated. After this, it was introduced for the field test. In the following two weeks, the users were continuously supported by a user advocate. Also, project members visited each user at least twice for a 60 - 120 minutes time span. During these prearranged visits project members encouraged tailoring activities related to the users’ search tasks. The tailoring process and the emerging problems were observed, written notes were taken during the observation and transcribed directly after the visit.

At the end of the observation period, a last extension of the search tool was introduced that included a simulated search in the groupware environment. A few days after installing this new version, we carried out semi-structured interviews with the users. The interviews covered the following issues related to the tailoring environment: patterns of collaborative tailoring, usage of textual documentation (manuals, help functions, annotations), occasions and means to experiment with applications, and further design requirements. The interviews lasted about 60 minutes and were carried out at the users’ workplaces. Written notes were taken during the interviews, a transcription was carried out immediately afterwards. Shortly after the interviews we copied all the tailored artifacts for analysis.

Implementation

In the following all extensions of Search Tool 3 are described (cf. Wulf 1999).

• Some new components

  We have implemented some new components, e.g. more expressive display components and a counter component capable of counting the found objects without displaying them. Regarding the privacy discussion this component allows for finding an object without seeing it. Thus one can search for an object only to get to know about its existence.

• More expressive descriptions of the components

  As described above the users encountered problems when selecting the basic components appropriately. As these basic components were labeled by rather design-oriented names, the users found it difficult to select the appropriate components out of a linear list in which the components were itemized. We tackled the problem in three different ways: First, we tried to find more meaningful names for the individual
components in collaboration with the users. Second, we added icons to the presentation of the list’s components that resembled the visual presentation of the components at the interface. Third, we classified the components into four different types and used this classification scheme as an additional hierarchy in the toolbox menu. Moreover, we implemented a context sensitive select box providing only those components that could be connected to the active port.

- **Hypertext-based Help**

  We developed a hypertext-based help menu for the search tool window and the toolbox window. The help texts of the tool-box covered all elementary components by a brief explanation of up to six sentences depending on their complexity. Screen shots were added where necessary.

- **Search Token**

  For privacy reasons we decided to allow searching only on everyone’s own desk unless a person put a special search token in a folder, which then allowed others to search this folder. This visible token meets users requirements to always be aware of the fact that someone else may search in one of their folders.

- **Ability to share tailored artifacts**

  Search Tool 3 allows users sharing different customized variants of search tools by saving them in a shared workspace. There, they can be deleted, renamed or copied. It is also possible to annotate these artifacts (see below).

- **Annotations: The facility to describe tailored artifacts**

  The initial workshop and the field test showed that users are hardly able to deduce the meaning of all components just from their names and the way they are classified. Hence, we generated possibilities to describe the functionality textually. Features that allow describing components and tailored artifacts have to take into account the different actors who produce this documentation (cf. figure 7).
An exploration mode

In a real-life setting, it is difficult for users trying to find out how an unknown search tool works, when search permissions have to be explicitly granted. Users who try out a new search tool that unexpectedly does not find any documents on other users’ desks, have difficulties to judge whether this outcome is due to a wrong understanding of the tool or missing search-permissions on others’ desks. Therefore, we decided to extend the search tool environment by an exploration mode. For privacy reasons the final version of the exploration mode included the possibility to explore a simulated data space that did not belong to a person but contained artificial data generated particularly to support people in exploring the search tool’s functionality.

Figure 8 shows a screen shot of the exploration mode. The window on the top right allows populating the simulated desktops with experimental data. The other two windows (search tool window at the bottom and tool box window on the top left) show the tailoring environment in exploration mode. The windows look exactly like the original ones with the exception of the background color. The search tool presented in the search tool window operates on the experimental data visible in the top right window.
Figure 8: The exploration mode

Evaluation

Structuring Components and Tailored Artifacts

Search Tool 3 included some new components with more expressive descriptions. During the field test we found that these features improved the users’ ability to select elementary components. Still, when the components were invisible during the search tool’s usage or their functionality was rather complex, it turned out to be difficult to communicate their meaning by a name or an icon (i.e.: it was difficult to find appropriate names and icons for switches). Besides, not all of the users conceived the components’ classification scheme that we used in order to establish an additional hierarchy level in the menu. So they suggested abandoning the additional level in the hierarchy of the menu and applying it just as a means to structure the linear list. Given a list of all in all 17
elementary components, this was a viable solution. Nevertheless, if a component based tailoring language consists of considerably more elementary components, the former approach needs to be pursued, and that may lead to the mentioned problems. A practical approach to solve this problem would be a tailorable menu structure. Yet, if each user could modify the structure individually, this might lead to collaboration problems.

During the field test, the naming of tailored artifacts caused problems, too. For instance, the clerk from the public relations department used her own convention to name search tools she had modified. This convention was not well understood by the other users. In order to encourage collaborative use of tailored artifacts, common naming conventions are essential. The classification of tailored artifacts may lead to further problems. Right now, there are just two linear lists for the compound components and the search tool alternatives. These lists are in alphabetical order according to their names. Nevertheless, with an increasing number of these artifacts, individually or collectively tailorable classification schemes seem to be indispensable.

**Shared Tailored Artifacts and the Division of Labor**

Workshop 2 of Search Tool 2 revealed that the users liked the idea of being able to share tailored artifacts if there was a local expert responsible for these activities. During the field test, the users therefore welcomed the implemented sharing mode. The two none-expert users appreciated being provided with high quality tailored artifacts. The local expert, however, stated in his final interview that he would feel a bit uneasy if any tailored artifacts became publicly available, and thus, other users could see when and what he had tailored. He asked for private stores where he could keep his experimental artifacts.

**Annotations and Help**

The field test showed that users are hardly able to deduce the meaning of all components just from their names and the way they are classified. Hence, we generated possibilities to describe the functionality textually. Features that allow describing components and tailored artifacts, have to take into account the different actors who produce this documentation.

As programmers created the elementary components and the tailoring environment, we developed a hypertext-based help menu for the search tool window and the toolbox window. The help texts of the tool box covered all the elementary components by a brief explanation of up to six sentences depending on their complexity. Screen shots were added where necessary.

The field study furthermore revealed that the local expert was the only one who used the help menu at least sporadically. All users indicated difficulties in finding the access point to activate the help-texts and the location of the desired explanation in the hypertext presentation.

Other than with the predefined elementary components, users generated compound components and full search tools themselves. Thus, the description of these artifacts had to be carried out by them. As the textual documentation of design rationales imposes extra burden and is therefore often omitted (cf. Grudin, 1996), we tried to provide as much technical support as possible. We have implemented an annotation window that consists of five different text fields: “name”, “creator,” “origin”, “description”, and “remarks” (cf. figure 7). The “name” field is automatically marked whenever a tailored artifact is created. In the “creator” field the user who builds a tailored artifact can fill in his name. In the “origin” field, a reference is generated automatically in case a tailored
artifact has been created by modifying an existing one. In the “description” field the creator should clarify the function of the component. In case a compound component is derived from an existing one, the original description is copied automatically and put in Italics. The “remark” field can contain further comments. Contrary to the help texts, the annotations were accessible directly from the display of the respective tailored artifact.

During the field test annotations were used more frequently than the help-texts, which was probably caused by an easier access mode and the richer information structure. The users liked the information structure of the annotation window. The documentation of the creator’s name was important to them for four reasons: First, knowing about the creator’s typical search tasks helps them to understand the functioning of the tailored artifact. Second, the creator’s name is an important information for judging the quality of a tailored artifact. Third, it allows contacting the creator for further information. Fourth, the documentation of his name gives the creator a chance to let the organization know about his efforts. The users found the “origin” field helpful as it recorded parts of the tailoring history, and thus, eased understanding the functioning of an artifact. The “description” and “remark” fields were perceived being essential to increase understanding and were almost always filled in during the field test. Nevertheless, the way they were filled in was often regarded problematic. Especially the usage of abbreviations and incomplete sentences caused considerable problems. Thus, here again user groups carrying out collaborative tailoring activities need to develop appropriate conventions.

**Exploration**

Discussing the concept of exploration during the interviews, the user providing local support found the exploration environment useful to test whether a search tool really does find what it was supposed to find. The other clerk was quite reserved towards this concept because to her it seemed too complex to handle. The efforts to create experimental data and to handle the different roles appropriately seemed too high for her compared to the benefits: checking whether a given search tool is doing what it is supposed to do.

While observing users’ tailoring habits we found many occasions when building and experimenting interleaves. For instance, users modified a given tool to better understand its functioning and that of some of its parts, or they carried out minor modifications in an existing tool and experimented with that. Therefore, we believe that building tailored artifacts and experimenting with them should be both supported in the exploration mode.

We extended the exploration mode in a way that experimenting not only with the tailored artifacts but also with the tailoring functions was possible. In order to build an explorable tailoring environment, we applied the concepts ‘experimental data’, ‘neutral mode’ (action is not really performed, only explained) and ‘freezing points’ (defined status of a file that you can return to after experimenting). Whenever a user decides to switch to the exploration mode of the tailoring environment, a new window pops up. It has a specific frame color and contains those windows of the tailoring environment that were active before starting the tailoring mode (search tool and tool box). These windows behave regularly, except for those functions that allow modifying the state of the original search tool environment (e.g. store search tool, rename search tool). These functions are put into neutral mode; so the state transition following their execution is not carried out but described textually. The exploration mode comes up with a copy of the search tool that was active before. Playing the role of experimental data for the tailoring environment, this tool can be modified by means of the tailoring functions. If users decide to leave the exploration mode, they are asked whether they want to store or abandon the outcome of their
explorative activities. In any case, the users return to the tailoring environment containing the search tool that was active before they started the exploration mode.

**DISCUSSION & CONCLUSIONS**

In this paper, we described the process of developing three consecutive prototypes of a search tool to be used for search in a shared workspace. The process included user participation of different kinds at various steps of the development process. We learned much about the searching for files within groups, the chances and limits of making such a search tool tailorable for individual as well as group needs and the chances and limits of component architectures to do so. Several essential insights have been gained from the process as a whole. We are now able to provide some answers to our research questions *how can we design for searching in groupware* and *how well are components suited for runtime tailorability*. While the overall need for tailorability is out of question, the study itself cannot be generalized easily.

The technical basis of component-based architectures enables and enhances flexibility of software. On the one hand the component paradigm allows for independent development of single domain-specific components that are then to be integrated with multiple purpose components. On the other hand the idea of composing software due to personal needs was quite well understood. The question whether any application can be decomposed in the same way as we did with the search tool, still remains open. However, our experience indicates that those applications that are characterized by a data flow structure, seem to be good candidates.

Designing for searching in groupware encompasses several dimensions. During the *process* of design, the combination of having users participate and developing several versions of the search tool evolutionarily proved to be effective: Each workshop, interview and evaluation provided us with insights that could not have been won by top-down one-step design. Since we did not only want to support particular organizations but also wanted to show the general feasibility of our approach, the process was rather costly with respect to time and personnel. However, since we provided evidence that our approach was effective, subsequent endeavors to construct tailorable software with limited functionality can also be efficient. Nevertheless, they will still have to bear in mind that providing a flexible organization with adequate software support always is a laborious and ongoing effort.

Search Tool 3 as the final *product* of the design process included much functionality, all of which was implemented due to requirements from users or improvements derived from literature survey. The resulting complexity led to a differentiation in using the functions:

- few of the functions are used frequently by many users, e.g. looking for or trying out new search tools from the search tool pool;
- some are used sometimes by many users, e.g. exploration of an unknown search tool in case a task or a group changes or the written explanation of the search tool is not sufficient;
- some others again are only used by few people, e.g. the possibility to create new search tools from scratch or perform severe modifications on an existing search tool.
This is encouraging in two ways: Firstly, the fact that the functions were used at all and that there were
differentiated patterns of usage, proved that there really was a need for a tailorable search tool. Secondly, and
more important, it shows that the functionality we added, resulted in more than just the sum of the parts: the
different options and levels of using, annotating, exploring, modifying and constructing search tools led to a
variety of options for users to tailor. Thus, they could move up to a plateau of the “tailorability mountain”
(MacLean et al. 1990) that suits their abilities and needs with the option to engage in more tailoring activity by
talking to a local expert. This was also supported by our layered architecture allowing for the assembly of search
tools on different levels of granularity and in fact supported the evolution of a local expert (cf. Won et al. 2005).

There are two ways in which the fact is taken into account that the search tool was meant to work in a
collaborative setting to find files in a shared workspace. On the one hand the participating users were the objects
of other’s search activities. Therefore there was a request for mechanisms to protect privacy. In order to meet this
request we implemented an option to find objects without being able to display them, a search token marking a
folder to allow this folder to be searched, and the possibility to explore a simulated data space rather than the real
shared workspace in order to explore how a search tool works. On the other hand the users were subjects of the
search. Not only did they require efficient mechanisms for their individual search but there was also the need to
provide mechanisms for collaboration regarding search activities. Therefore, our search tool environment
provided mechanisms to share, distribute and annotate search tools for others to use or modify, thus supporting
weak forms of collaboration. The workshops show that stronger forms of collaboration where users meet and
discuss the tailoring of a search tool, are generally supported by the possibility to tailor and particularly by the
runtime environment, where ideas that have been discussed can easily be turned into a tailored search tool and be
tested. We assume that such a tool for easy tailoring may serve as a medium to encourage groups discussing
group standards that then can be shared. The systematization of customizations resulting from a collaborative
tailoring process would then contribute to common norms and conventions needed for collaborative work. Again,
one must bear in mind that this eventually rewarding activity of collaborative tailoring requires willingness and
patience.

Sharing tailored search tools seems most helpful for small work groups with a rather similar work context. Being
able to explore functionality in groupware context with real or simulated data on the other hand becomes more
important the more users are involved: The complex interdependencies of different users and their privacy and
other settings require more than a textual description of functionality. However, this rather complex form of
supporting users’ understanding always has to be accompanied by simpler forms like tutorials, context-sensitive
help texts and a colleague to ask.

Our specific research focus was on how well components are suited for runtime tailorability. For quite a while it
has been demanded to provide tailorability not only on a surface level but to embed it deeper into systems. Using
a component architecture to realize runtime tailorability proved to be a good choice. The process of developing a
search tool showed that using a component architecture suited for runtime tailorability was a feasible way to meet
multiple requirements for such a search tool. The relative simplicity of the task consisting of input, searching, and
output facilitated identifying an adequate level of modularization and the relevant components. For more
complex tasks, the identification of useful modulization patterns will be much more difficult but, as we believe,
also possible - provided that there is some guidance for users and some time for them to have experts and
solutions evolve (cf. Stevens and Wulf 2002). The possibility to represent the connections visually provided for a significant reduction of complexity. Considering the difficulties that some users encountered in assembling a search tool during our workshops and the field test, we doubt that other forms of modularization (e.g. blocks of program code) would have been manageable to all but very few skilled users. Being able to modify components and their wiring during runtime was vital for the acceptance of the search tool construction set. If there had not been such a tight integration between using and tailoring the search tool, the threshold to create, modify or test a search tool would have been critically high. The component-based approach also allows for defining domain-specific languages in general, which can be achieved by combining basic components (buttons, input boxes, etc.) with domain-specific ones (DSL 1997). Together they then build the component framework from which composition can be build by end-users.

There is also evidence that such a component-based approach can be enhanced and possibly be used for the design of a variety of systems. The strict separation between search tool specific components and the design of the runtime environment makes it comparatively easy to enhance the prototypes by further components. Therefore we think that the design and runtime environment can easily be reused: it is technically possible to create a multitude of different tailorable tools and applications. However, considering the simple structure of a search tool, it remains an open question at which level the creation of different kinds of complex tailorable software with these forms of components is no longer reasonable.

The process described above and the lessons learned encourage us to continue research that combines theoretical and empirical evidence to come up with suggestions that can make a difference to software designers, introducers, and users.

REFERENCES


DSL 97, 1st ACM SIGPLAN workshop on domain specific languages, January 18th, 1997.


Tailoring by Integration of Components: The Case of a Document Search Tool

JCSCW - Computer Supported Cooperative Work (CSCW): The Journal of Collaborative Computing, Vol. 9, Issue 1, 2000, Special Issue on Tailorable Systems and Cooperative Work


Stiemerling, O.; Kahler, H.; Wulf, V.: How to make software softer - designing tailorable applications. In: Proceedings of the DIS ’97, pp. 365-376


Trigg, R.; Bødker, S.: From Implementation to Design: Tailoring and the Emergence of Systematization in CSCW, in: Proceedings of CSCW ’94, pp. 45-54


Wulf, V.: “Let’s see your search-tool - Collaborative use of tailored artifacts in groupware. Proceedings of GROUP ’99, pp. 50-59