In: Intercting with Computer (in press)

Exploration Environments: Supporting Users to Learn Groupware Functions

Volker Wulf

ProSEC – Department of Computer Science III
University of Bonn
Römerstr. 164, 53117 Bonn, Germany

Explorative learning plays a major role when users face new functionality. Nevertheless, the multi-user character of groupware makes explorative learning more difficult. Users are often unable to understand the way certain functions work because they cannot perceive the effects of the functions' execution. This problem gets more severe with tailorable groupware. Therefore, we propose exploration environments as an additional feature to support users in self-directed learning. Looking at three tailorable groupware tools, we show how exploration environments can be realized. To generalize our findings, we develop a model which describes the user interface of tailorable groupware. Based on this model, we compare the design of the three tools and present general guidelines for the implementation of exploration environments. Finally, we report about the results of a workshop in which a groupware tool containing exploration environments has been evaluated.

1 Introduction

The term groupware is defined in different ways in the literature (cf. Johansen 1988; Ellis, Gibbs, and Rein 1991, Johnson-Lenz and Johnson-Lenz 1991). We will speak about groupware in a general sense being multi-user applications, which support communication and cooperation among their users (cf. Wulf 1997a). Due to the differentiation and dynamics of the supported communication and cooperation processes, these applications need to be tailorable, and therefore, users can adapt applications during use to different tasks, personal preferences and group standards (cf. Henderson and Kyng 1991; Malone, Lai and Fry 1992, Oberquelle 1994; Bentley and Dourish 1995, Wulf and Rohde 1995; Kahler et al. 1999a).

Tailorable groupware is only useful if users learn to adapt it. In psychology learning is defined as a modification of behavior based on experiences (cf. Lefrancois 1972). The process of learning a computer application modifies the user's behavior whenever the functionality of the system supports his task at hand. In case of tailorable applications, learning enables the user to carry out the corresponding operations, whenever an adaptation seems necessary to him.

There are different ways users may learn the functionality of a computer application. On the one hand the learning process can be structured externally. This is typically the case with computer courses, seminars, and manuals. On the other hand users can learn an application by self-directed learning activities where they explore a system's functionality according to their actual needs (cf. Dutke 1994, pp. 146).

Based on empirical investigations, Carroll and Mack (1983 and 1984) argue that exploration plays a major role in learning an application. As there is often a mismatch between the learning goals of the users and the externally imposed structure, users tend to rely strongly on self-directed activities. Experimental studies in laboratory settings indicate that explorative learning is often more efficient than externally directed learning activities (Kamouri et al. 1986; Greif and Janikowski 1987; Frese et al. 1988). Qualitative research investigating how users learn to tailor an application stresses the importance of exploration, as well. Mackay (1990, pp. 172) finds that experimentation plays a major role when asking interviewees about the most important information resource when they are tailoring. According to Oppermann and Simm (1994) users learn to tailor mainly by trial and error. Nevertheless, the design of the user interface of tailoring environments provides quite some barriers to self-directed learning. According to Mackay (1990, p. 165) users fear to break things. Oppermann and Simm (1994) find that the effects of tailoring activities are difficult to

perceive by the users.

The following will deal with support for users in learning tailorable groupware in a self-directed way. To tackle this problem, we will present the concept of exploration environments. The concept will be developed by generalizing from three design cases in which learning support for tailorable groupware has been implemented.

The paper is structured as follows: In the subsequent section we survey the state of the art concerning technical features which support self-directed learning. These features have been developed for single user applications and are not well suited to support learning of multi-user applications. In section three we present three tailorable groupware tools which have been developed according to user requirements within a cooperative design project, the POLITeam project. To provide a terminology to speak about tailorable groupware in general terms and to generalize the findings of this paper, we develop a model of the user interface of tailorable groupware (section four). The three tools presented in section three are used to illustrate this model.

Within the POLITeam project it turned out that users had problems in learning to handle the tailorable tools which they required. Based on the state of the art presented in section two, we have developed innovative features which support self-directed learning in multi-user applications. In section five we describe for each of the tailorable tools how these innovative features, called exploration environments, were realized. Comparing the three design cases by means of the user interface model, we develop in section six the general concept of an exploration environment for tailorable groupware. Section seven presents the results of an evaluation study dealing with one of the implementations of this concept. The paper ends by discussing potentials and limitations of the concept of exploration environments.

2 Technical Support for Self-directed Learning

Technical features to support self-directed learning of a groupware's functionality have not been developed in the CSCW community. Nevertheless, since the early 80s exploration has been an important research topic in the HCI community. Concepts for an interface design which promotes self-directed learning have been discussed for single user applications. Paul (1994, pp. 156) gives a survey on these concepts. He distinguishes between two learning activities: overviewing the existing functionality and experimenting with certain functions, when users try to apply a system to carry out a task at hand. During the overview phase the user investigates the given functionality. Survey functions present all functions of an application according to certain classification schemes. Histories of interaction allow users to survey those functions which they have executed before. Scenario machines display a recorded sequence of activities which gives examples how users can apply certain functions. By contrary, when experimenting the user tries out which state transition follows the execution of a certain function. Undo-functions allow to reset the execution of a function. Freezing points allow to define certain system states in advance to which the user can come back after having tried out other functions. Experimental data are especially created to encourage experiments with certain functions. A neutral mode replaces the execution of a function by a textual description of the effects of its execution (cf. Darlington, Dzida, Herda 1983; Yang 1990; Abowd and Dix 1992; Berlage 1994; Paul 1994).

The overview phase basically requires for single user applications the same technical support as for groupware. By contrary, experimenting with groupware requires in most cases new technical concepts. In single user applications users can typically perceive the actual state of an application and the state transition resulting

from the execution of a function. In groupware, the actual state of an application and the effects of a function's execution are often invisible or only partly visible for the users (e.g.: the access rights granted to somebody else cannot be perceived by the owner). Only in case an application follows the WYSIWIS principle (What You See Is What I See) in a very strict manner (cf. Johansen 1988), users can perceive the effects of a function's execution on the side of the other user. Moreover, groupware functions being activated by different users can lead to interdependencies, which are hard to perceive just from the perspective of the activator. This problem gets even more difficult if the corresponding functions are heterogeneous with regard to the different users' interfaces. Heterogenity is caused by either a different design of the functionality on the different users' interfaces or a tailorable design of the functionality on the different users' interfaces. In case the design of the functionality differs on the various users' interfaces, it is hard to learn about interdependences between the functions. In case the functions on the various users' interfaces are tailorable, the state of the application's functionality is unstable and requires continuous learning efforts.

In the CSCW literature, concepts have been developed which enable users to reset the effects of a function's execution even if other users have manipulated the data afterwards (cf. Ellis, Gibbs and Rein 1991; Prakash and Knister 1994; Ressel et al. 1996). These approaches support the exploration of an application in case the application follows the WYSIWIS principle and there is not any interdependence between the execution of the functions on the different users' interfaces. In case these preconditions are not given, these approaches do not support self-directed learning because users cannot perceive the effects of a function's execution on other users' interfaces.

Some groupware applications allow users to take different identities while they are logged-in at the same terminal. Being logged-in under different identities a user can switch between the different interfaces. Such a feature supports self-directed learning, because users can try to find out about the state transitions on the other users' interface. For security reasons it is not acceptable to apply this feature to real users' accounts. That means, one could think of creating dummy users for each real user of the system to support self-directed learning activities. Nevertheless users may have problems in dealing with the complexity resulting from handling two full interfaces. Taking such an approach it is impossible to reduce this complexity or to implement specific features which support the learning process. In next section we develop specific features, which support self-directed learning of groupware.

3 Three Tailorable Groupware Tools

The POLITeam project was a software development project where the application partners required technical support for distributed cooperation. The project began in May 1994 and ended in December 1998. The main function of the POLITeam system is to supplement paper work processes with electronic work processes in one German federal ministry and in different bodies of a Northern German state government. POLITeam offers shared workspaces, electronic circulation folders and E-mail functionality (cf. Wulf 1997b; Prinz, Mark and Pankoke 1998; Pipek and Wulf, 1999). A cooperative and evolutionary approach was applied to redesign an already existing groupware system (LinkWorks by DEC) according to specific user, organization and situation requirements. Tailorability played a major role during the development of the system (cf. Stiemerling, Kahler and Wulf 1997).

In the following we will present three tailorable groupware tools which have been

developed in the context of the POLITeam project. These tools are tailorable in different ways. Taking these tools as examples, we will develop an approach to model the user interface of tailorable groupware applications.

3.1 A Tailorable Search Tool

The original version of LinkWorks contained a search tool which allowed users to search for any object independent of its actual location within the system. Discussions with users revealed that this design was inappropriate, because it threatened users' privacy and allowed users to manipulate someone else's documents (cf. Kahler 1996). The problem was enlarged by the fact that it was not possible for the owner of a document to specify other users' search rights appropriately (cf. section 3.2). To satisfy the users' different requirements the original search tool was redesigned in a tailorable way and a more intuitive manner to specify search rights was developed (cf. Wulf 1999a).

To reach a high degree of tailorability, the search tool was dismantled into different types of components. On the contrary to the traditional application of components in software engineering (e.g. Banavar et al. 1998), a tailoring environment was developed which allowed for runtime composition of the components (cf. Stiemerling and Cremers 1998).

The search tool consists of six types of elementary components.[1] Specification components allow building an inquiry mask containing just those search attributes which the user typically needs. Taking the example of the search tool presented in figure 1, four different specification components are displayed in the upper two rows. The search button is a component of which activation finishes the specification of the

[1] Won (1998) and Engelskirchen (2000) give a more detailed description of the tailoring language.

inquiry and starts the search engine. Looking at the search tool presented in figure 1, one can find it being the first element in the third row. On its right there is the search engine. It connects the search-tool with the LinkWorks database via the application-programming interface. The search engine transfers the inquiry of the users to the database and receives a list of retrieved documents. Different result switches allow to subdivide the retrieved documents and transfer them to distinct display windows. In figure 1 one finds the result switch "Name" which subdivides the search results according to the first letter of the documents' names. Two different windows allow the displaying of the retrieved documents either in a normal window presenting the documents' names and further attributes or in a window which counts the amount of documents retrieved and just presents their number. In figure 1 there are two windows of the first type which display the subdivided search results. The left window contains those documents with a starting letter from "A" to "M", in the other window those starting with "N" to "Z". Control buttons implement different modes in dealing with retrieved documents. Either a link or a copy of the document can be created on the user's desktop or the document can be accessed directly. In figure 1 these buttons are located above each window. The search tool presented in figure 1 allows to create links and copies of documents displayed in each of the windows.

To assemble these components, wiring functions are given, which allow to connect the different types of ports. Connections between components are indicated by dark lines between the input and output ports (cf. figure 1). To support users in distinguishing between input and output ports, input ports are empty circles, output ports full circles. To support users in wiring the components appropriately, input and output ports, which can be connected, are presented in the same color.

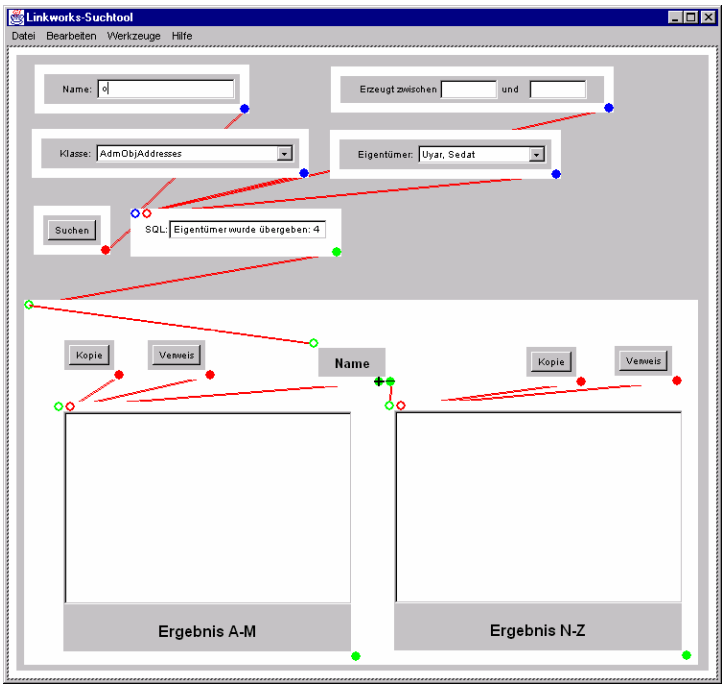

Figure 1: The search tool window in tailoring mode

The tailoring environment allows to generate compound components, which are an assembly of other components wired together. Figure 1 presents a search tool which has been composed of two compound components: one compound component for search (composition of specification components, the search engine and the start button) and another one for display (composition of result switches, windows and control buttons). To build such a search tool, users just need to select from the given set of compound components and wire search and display components by means of a single connecting line. Such tailoring activities are less complex than building a whole search tool from elementary components.

On the lowest level of tailoring complexity, the users can just choose between alternative search tools. A search tool is a specific compound component built either from elementary components or from lower level compound components. So the search tool which has been assembled in figure 1 can be stored and later be reused.

3.2 A Tool to Specify Access Rights

The original version of LinkWorks allows users to specify access rights by means of profiles. Chosen for a certain object (e.g. document), these profiles determin which user or user group could carry out certain functions on an object or on an object class. These profiles are generated by system administrators and allow to group users only according to the formal organization hierarchy modeled in the tool. Moreover, the users up in the formal hierarchy are automatically allowed to read the objects created by the users positioned lower in the hierarchy. Since this implementation of access rights was not flexible enough to support ad-hoc groups and endangered the privacy on the lower hierarchy levels, we decided to define just three elementary access profiles in LinkWorks. And we developed a new tool to specify access rights.

The new tool is integrated into the LINKWORKS system, replacing the old access control lists with a statement-based approach. Tailoring statements are similar to production rules in expert systems. A statement consists of a condition and an action part. The condition part of the statement describes the situation in which the object is accessed. The condition can be formalized by the following attributes: accessing user or user group, the activated function (search, read, delete, modify, create link, delete link, and create statements), the accessed objects or object classes, the object location (a folder containing objects), roles concerning an electronic circulation folder (initiator, actual holder, etc.), and the time of access. The action part of the statement describes whether in the described condition the function should be executed or not. Statements are also used for meta-tailoring (cf. Dewan and Shen 1998) in the sense that they specify which user is allowed to create a statement that concerns certain objects or object classes (cf. Wulf, Stiemerling and Pfeifer 1999).

Users define new access rights by creating new statements and deleting or

changing old ones. Figure 2 shows the screen used to input or edit statements. In the upper part of the screen, a user chooses the relevant attributes of the statement's condition part from the selector boxes. In the middle part of the screen he defines the action part of the statement. This is done under the label "Decision" by selecting either the alternative forbidden or allowed. In the lower part of the screen ("Rule in plain text") the rule as it is specified above is presented in plain text. In the example given in figure 2 a statement to control the access to the document "Text" has been created. The rule states that whenever the user "Oliver" tries to have access to this object (condition port) the access will be allowed (action port). The presentation in plain text is updated, whenever the user changes the formal definition of the statement above.

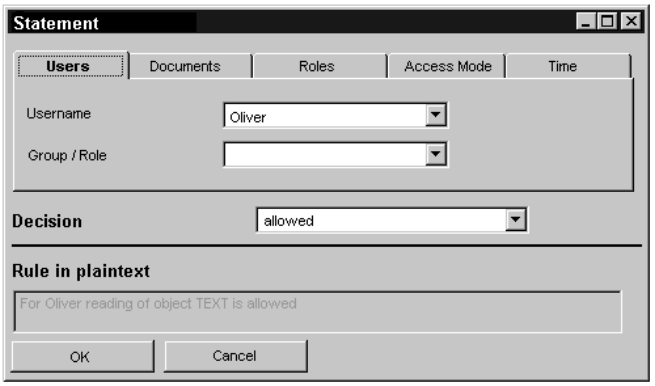

Figure 2: The screen for editing statements

A major problem during tailoring access rights by means of statements is the handling of inconsistencies. An inconsistency occurs if the condition parts of more than one statement is met by an actual situation of usage and the action parts of these statements prescribe different system behavior. There are two classes of inconsistencies: inclusions (if the condition part of one statement is completely included in the condition part of the other statement) and orthogonality (if the two conditions do not have an inclusion relationship). The tool solves inconsistencies

resulting from an inclusion relationship by applying a weighing algorithm which gives priority to the more specific statement. Orthogonality is solved by weighing the attributes used in the definition of the condition of each statement differently (e.g. user, user group, object, object class, date, etc.). The weights are chosen in such a way that a unique value for every relevant statement is calculated. In case of inconsistency, the weighing algorithm leads to a sorted list of statements, which allows to resolve inconsistencies (i.e. the weights of the attributes which match the current situation are added up).[2] The weighing algorithm can be tailored by modifying the weights of the different attributes of the condition. Moreover, the weight of certain statements can be specified manually by the users to give, for instance, important general statements highest priority (cf. Wulf, Stiemerling, and Pfeifer 1999). To support users in handling weighed statements, a specific display mode is implemented (cf. section 4.3, figure 5).

3.3 Tailorable Filters of an Awareness Service

In the context of the POLITeam project it turned out that users require support to tailor their word processors cooperatively. Therefore, we extended the MS-Word application by a tool, which allows sending or sharing of those artifacts which could be tailored by users (e.g. document templates or toolbars). Each user has a private workspace to store tailored artifacts and a mailbox to receive tailored artifacts sent by other users. Furthermore, a shared workspace is provided to which users can publish tailored artifacts from their local workspace and from which they can copy tailored artifacts to their local workspaces. This tool is developed in VBA (Visual Basic for Applications) and integrated in the MS-Word menu bar (cf. Kahler et al. 1999b).

---

[2] Stiemerling (1996) and Stiemerling and Cremers (1998) describe the weighting algorithm in more detail.

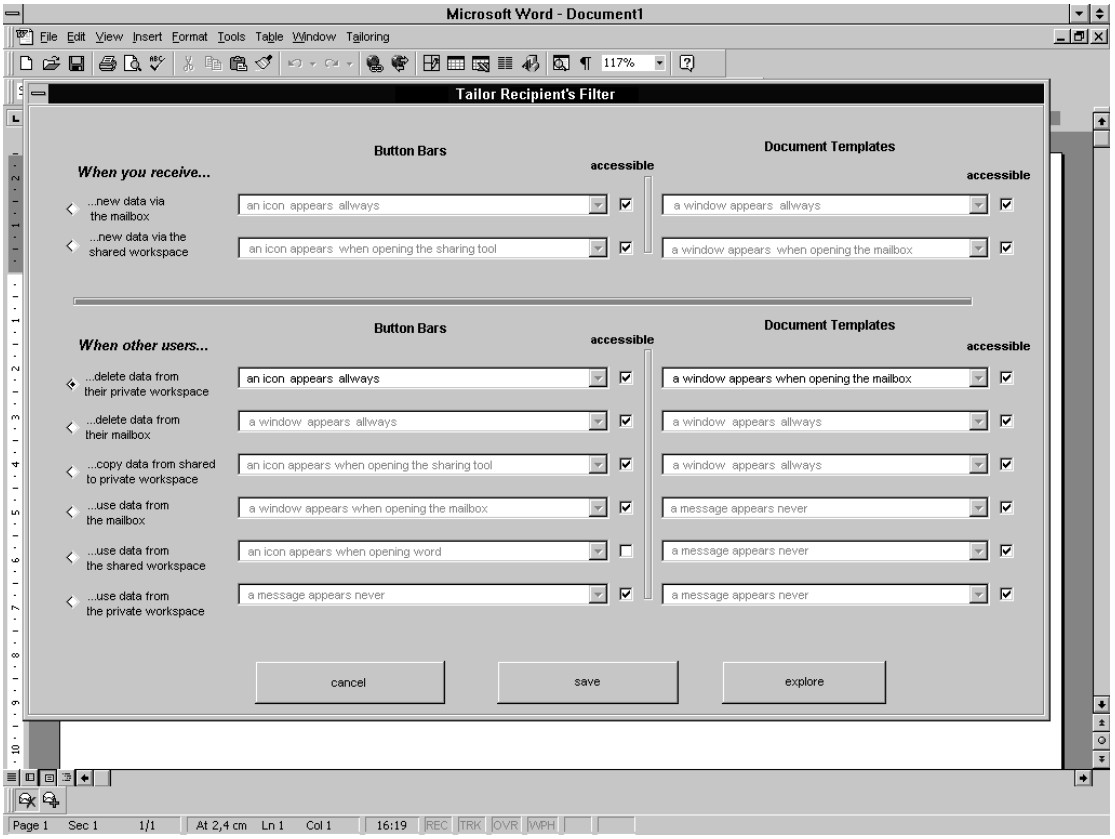

Figure 3: Window allowing to tailor the recipient's filter

To inform users about events concerning the shared artifacts, an awareness service is implemented in the tool. Eight different types of events are recorded automatically, distributed among the users and displayed at their interfaces. To avoid that users being overloaded by useless messages and to protect their privacy, the awareness service is tailorable by means of two filters. The recipient's filter allows selecting the relevant events to be displayed at the interface. Therefore, users can specify which type of event (of the given list of types of events) concerning which type of shared artifact (button bar, document template) should be displayed at their interface. Moreover, they can define situations (when starting Word, when starting the sharing tool, when opening their private mailbox) in which these events should be displayed and they can

select intensities (pop up window, icon presentation in the status bar, or not at all) for the presentation of these events.

The list of the eight event types is displayed in the left column of the tailoring window (cf. figure 3). The two types of shared artifacts are presented in the top raw of the window presented in figure 3. Concerning this 2 x 8 matrix a plain text description of the selected display situation and intensity can be found.

To ease the tailoring process, we provide two tailoring functions of different levels of complexity. On the first level, users can select for each type of events an interest profile. This profile can be chosen from a list of given profiles. It indicates in which situation and at what intensity an event will be displayed. On the second level of complexity the users can generate new interest profiles by combining display situations with intensities. Figure 3 presents the window which allows to tailor the recipient's filter on the first level. The selected interest profiles are displayed in the white boxes.

The producer's filter allows the user to decide which of the events created by themselves should be published and to whom. To tailor this filter we provide again two different levels of tailoring complexity. On the first level the users can decide which event should be distributed to which user or user group. On the second level the users can generate new groups of users. So the recipients can display only those events, which were published to them by the producer. The windows to carry out the tailoring activities are built up similar to the one of the recipient's filter.

4 Modeling the User Interface of Tailorable Groupware

Given the three examples of groupware tools, we will develop a model of the user interface of tailorable groupware applications. Such a model should allow describing

on an abstract level those aspects of a tailorable groupware application which are relevant to the users. The user interface model will provide a basis to generalize findings related to the design of these applications.

4.1 Functions, Mechanisms and Data

We assume that the user interface of an interactive application consists of functions and mechanisms. Groupware like any other interactive application provides functions and mechanisms to allow users to manipulate data.

Normal and Triggered Functions

Functions are the aspects of the interface the user can interact with. Their execution changes the state of the application. We can distinguish between normal and triggered functions (cf. Oppermann and Simm 1994). Normal functions are typically represented at the interface of an application, activated by an input of the user and executed in the way prespecified by a tailoring function or specified during the activation. Triggered functions are not represented at the interface of an application. They are executed by the occurrence of a prespecified event which is created internally or which results from the interaction with another application. The clock of the operating system or the execution of another function can produce such an event, for instance. Following the event, the triggered function is executed in the way prespecified by the tailoring function.

Tailoring environments consist of one or more normal functions. These tailoring functions allow to adapt other functions (normal or triggered ones) by creating or modifying persistent artifacts (tailored artifacts). On the contrary to the other functions the execution of these functions does not modify the data handled by an application but the functionality of an application. Together with the ordinary

functions the tailoring functions shape the functionality of an application.

An example of a normal function is the ”search function”. The users specify the inquiry during its activation. The tailoring environment allows to define the inquiry mask and the way retrieved documents are presented. By contrary, access control is a function which is triggered by the activation of normal functions, like read or search. It is executed according to the prespecified access rights.

Mechanisms

Functions are the parts of the user interface which can be activated (in case of normal functions) or at least tailored (in case of triggered functions) by the users. Mechanisms are those aspects of the user interface, which are implemented in a way that they cannot be activated or tailored by the user. They are similar to triggered functions in the sense that they get executed by the occurrence of a prespecified event which is relevant to the users. Nevertheless, the link between event and execution is implemented by the designer rigidly and inaccessible to the user. For instance, many aspects of the screen display of applications are not tailorable by the users (e.g. colors of the different ports of the search tool, textual description of the different events in the awareness service). We call such aspects of a user interface mechanisms.[3] Those aspects of the interface, which can be tailored (e.g. position of the specification boxes or the display windows in the search tool) are triggered functions. So the distinction between a mechanism and a triggered function is made by the designer when he decides which aspects of the application will be tailorable.

Local and Global Data

Single user applications contain data, which are not exchanged among the users.

---

[3] Any module in the program code could be seen as a feature because it is executed due to an internal event (call of the module). Nevertheless, our definition takes a user centered perspective. The generating event and the

These data are local to the producer. By contrary, a groupware application contains data, which can be exchanged between the users by means of functions or mechanisms. We distinguish between sending and sharing mode (cf. Wulf 1997a). In the sending mode the producer hands over the ownership of the data by making them accessible to the recipients. Before sending, the data are local to him. After receiving, they are local to the recipient, the producer cannot access them anymore. By contrary in the sharing mode the producer publishes the data and makes them accessible to the recipient. After sharing the data the producer can still access them. The data are shared between user and recipient. Access rights allow to create different categories of shared data.

4.2 Extending the IFIP-Model

On the base of the discussion on User Interface Management Systems (UIMSs), the „IFIP-model“ and the „Seeheim-model“ are early attempts to model those aspects of an application which are relevant to the users. The Seeheim model divides the user interface into the following components: presentation, dialog control and the application interface model (cf. Green 1985). Dzida (1983) comes up with a similar model, which differentiates among the input/output interface, the dialog interface, the tool interface and the organization interface. In both of the models functions, mechanisms and data can be attributed to one of the different components or interfaces.

Input/Output Interface

The input interface mediates between the physical input activities of the user and their formal representation within the system. Therefore, rules are defined, which

---

execution of the mechanism must make sense to the users.

specify how to input commands. Mechanisms or triggered functions, belonging to this interface, read the physical input devices and convert the raw input data into the form required by the other interfaces. Mechanisms or triggered functions belonging to the output interface display the actual state of an application in a way that it corresponds to the perceptive faculties of the user. So this interface determines how the output generated by the other interfaces is presented at the output devices. The input/output interface corresponds mainly to the Seeheim model's presentation component.[4]

Dialog Interface

The dialog interface guides the user interaction with the tool and the organization interfaces. It serves mainly two purposes. Physical restrictions of the input/output interface do not allow to present all functions of the deeper interfaces at the same time to the user. Hence, the dialog interface implements strategies in which temporal order, and by means of which dialog function users can have access to individual functions of the other interfaces (cf. Rauterberg 1995). Moreover, users are supported by the dialog interface when they apply individual functions of the tool and organization interfaces. These dialog functions or mechanisms provide the information to understand functions of the other interfaces, to control the execution of these functions, to inform users about errors when activating those functions, and to provide technical support to recover from an error state. Contrary to this wider understanding of the dialog interface, the Seeheim model's dialog control component rather focuses on the first of these two aspects. The error recovery functions are, for instance, subsumed under the "application interface model" component.[5]

---

[4] After having defined the input interface, the distinction between triggered and normal functions can be made more clearly. Normal functions are executed due to an event generated by the input interface while triggered functions are generated due to an event generated by one of the other interfaces.

[5] The IFIP-model's broader understanding provides the basis to formulate principles for the human centered design of the dialog interface. These principles have meanwhile become an international standard (cf. ISO 9241 Part 10).

Tool Interface

The tool interface represents those aspects of the functionality which support users in carrying out their primary tasks with the application. It represents the semantics of an application. The IFIP as well as the Seeheim model were originally developed to describe UIMSs which were understood as standard modules apart from the rest of an application's functionality. Therefore, both of these models understand the tool interface and the application interface model component as a module situated between the UIMS and the rest of the application. As we do not believe in the premises of UIMSs and are rather more interested in the analytic than in the constructive value of user interface models, we define the tool interface more broadly. All functions and mechanisms of single users' applications, which provide support for the users' primary task, form the tool interface of an application. Due to the IFIP model's further differentiation, we restrict the tool interface to the functions and mechanisms which manipulate local data and are not relevant for communication and cooperation among users.

Organization Interface

The IFIP model contains additionally the organization interface, which covers all the rules determining the relation between the individual task of a user and those of other users. Defining this interface, Dzida (1983, p. 6) mentions especially the division of labor, work processes and rules of cooperation. Contrary to the other interfaces, Dzida understands the organizational interface socio-technically. To describe the user interface of systems which support communication and cooperation more precisely, Oppermann et al. (1992) have differentiated the organization interface. They distinguish between a technical and a non-technical aspect. The technical aspect

contains those parts of an application, which support communication and cooperation among users. We call the functions and mechanisms on this interface global ones. The non-technical aspect deals with the organizational and social embedment of the application. By introducing the technical aspect of the organization interface as an own entity, the tool interface gets restricted to single user functions and mechanisms which are not relevant for communication and cooperation among users. We call the functions and mechanisms of the tool interface local ones.

The technical aspect of the organization interface can be differentiated again. We distinguish between real global and supportive global functions (cf. Höller 1993, Wulf 1997a). Contrary to local functions and mechanisms, supportive global ones play a role in the technically mediated communication and cooperation according to the sending mode. They support users before sending and after receiving data. Nevertheless, their execution leads to state transitions only on the activator's local data. On the contrary the state transitions following the execution of real global functions or mechanisms deal with the sending of data or affect the state of shared data. Thus, it can be perceived at the output interface of other users, as well.

Tailoring environments

Applications may provide tailoring environments for functions on each of the different interfaces. A tailoring environment is typically activated by a dialog function and contains functions to create or administrate tailored artifacts (cf. Oberquelle 1994). These tailored artifacts, referring to the different interfaces are presented in figure 4 by the variable names: $T_{I/O}$, T_D, T_{TL} and T_{TO}. Their state determines the execution of the tailorable functions.

The set of functions which allow to build a tailored artifact is forming a tailoring language. A tailorable function may be equipped with different tailoring

environments. A tailoring environment may contain languages of different levels of complexity. Henderson and Kyng (1991, pp. 226) distinguish among three levels of complexity:

- choosing among alternatives of anticipated behavior,
- constructing new behavior from existing pieces,
- altering the artifact (i.e. reprogramming).

Certain tailoring languages may allow to create artifacts which modify other languages. These tailored artifacts are called T_T in figure 4. Their state influences the execution of other tailoring functions. Thus, tailoring environments can be designed in a layered way. We speak about a layered structure of the tailoring environments in case of:

- different languages contribute to the tailoring of the same normal or triggered function on various levels of complexity,
- languages of higher complexity create tailored artifacts, which modify or extend the less complex languages (cf. Wulf 1999a).

Layered languages are advantageous in dealing with different levels of tailoring expertise among the users, encouraging individual learning and stimulating cooperative tailoring activities (cf. MacLean et al. 1990; Nardi and Miller 1991, Nardi 1993).

Overview over the Extended IFIP Model

Figure 4 presents the extended IFIP model which can be used to describe the user interface of groupware. The interfaces of two users and the non-technical aspects of their organization interface are presented. As tailoring environments - especially for users without programming skills - should be designed consistently with the ordinary functions (cf. MacLean et al. 1990, Oberquelle 1994), we do not distinguish between

the ordinary functions and the ones of tailoring environments on the level of the input/output and the dialog interfaces.[6] On each of the different interfaces, we have represented the functions relevant to the user. Their execution influences the state of the data belonging to this interface. The execution of the tailorable functions leads to different state transitions depending on the corresponding tailored artifact. These artifacts are created, modified and deleted within the corresponding tailoring environments. State transitions resulting from the execution of real global functions affect the interface of other users as well. Therefore, the technical aspects of the organization interface are connected in the same way as the other interfaces on each of the user's side.

[6] At that point our approach to model the user interface of tailorable groupware differs from the one proposed by Oberquelle (1994).

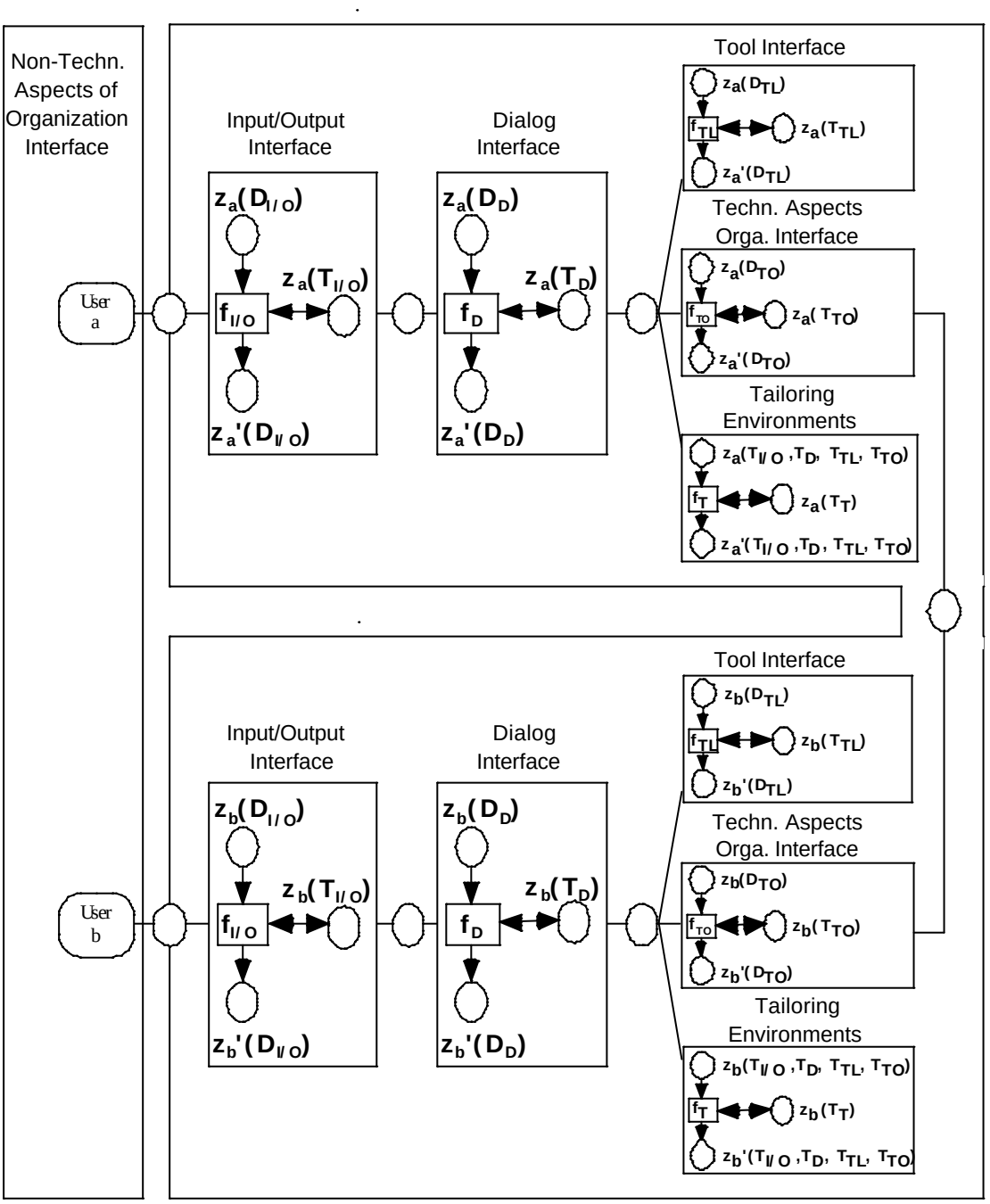

Figure 4: Extended IFIP-Model describing the user interface of tailorable groupware

4.3 Applying the Interface Model

Next, we want to apply the given terminology to describe the tools mentioned in section 3. We will pay special attention to those aspects of the input/output and dialog interface which support users in tailoring these tools.

Search Tool for Groupware

A search tool for groupware implemented in the described way is a normal function of the organization interface.[7] As it works on shared data it is a real global function. The effects of the search tool's execution on shared data are influenced by the access control function. The access control is triggered by the execution of the search tool.

The tailoring environment is layered, and therefore, it provides functions to select among different search tool alternatives, to build search tools from compound components and to assemble compound components or search tools from elementary components. Newly created compound components and search tools extend the set of given alternatives on the less complex layers.

The dialog interface of the tailoring environment influences the way how the different functions can be activated. The alternative search tools can be activated from one of the menu bars of the search tool. This menu bar also contains an item which allows to switch into the assembly mode. In the assembly mode the wires connecting the components of the actual search become visible and can be edited. Moreover, a toolbox appears which contains the elementary and compound components listed in two distinct menu bars. Thus, this mechanism of the dialog interface provides the users first of all with a tailoring environment which contains a list of alternative search tools. A second environment offers them complex tailoring functions. Help functions describe elementary functions while annotations describe compound components and search tool alternatives.

The output interface contains mechanisms which display the different components, ports, and wires. Figure 1 indicates for example that compound components are

[7] In case the search tool would work only on the local data of the activator, it would belong to the tool interface.

presented by means of a white frame underlying the elementary components grouped together. Wires between components are displayed as lines between the respective ports. A triggered function of the output interface allows the users to name newly created compound components and search tools. These names are used to represent these artifacts in the menus from which they can be selected. The input interface determines for instance how compound components are created and how wires between components can be generated and deleted.

A Tool to Specify Access Rights

The tool which allows the users to specify access rights in groupware is a tailoring environment. It allows to determine the behavior of the access control function. The access control function is triggered by those normal functions which try to access shared data. Therefore, it is a real global function. The tailoring environment contains functions which allow to create, edit and delete statements. It is not layered because there is only one tailoring language given.

The dialog interface of the tailoring environment supports users in understanding the statements referring to a certain object. With regard to an object, users can activate a menu item in LinkWorks which displays the relevant access statements in a window (cf. Figure 5). The statements are listed according to the strategies for inconsistency handling described before. The statement on the top of the list is the highest rated statement, i.e. the most specific statement. The last statement is the least rated one, i. e. the most general statement. This window contains also the access points to activate the three tailoring functions.

Moreover, the window contains the access point for the undo and redo functions.[8] Pressing the undo button the users can reset the last modification for the displayed set

of statements. A further activation of the undo function resets the second last of their tailoring activities. The redo function allows to reset the last execution of the undo function.

A mechanism of the output interface presents the relevant statements in plain text. It generates the text automatically from the formal data structures representing the statements internally (cf. figure 5). The input interface determines, for instance, how to select a statement from the list to be modified.

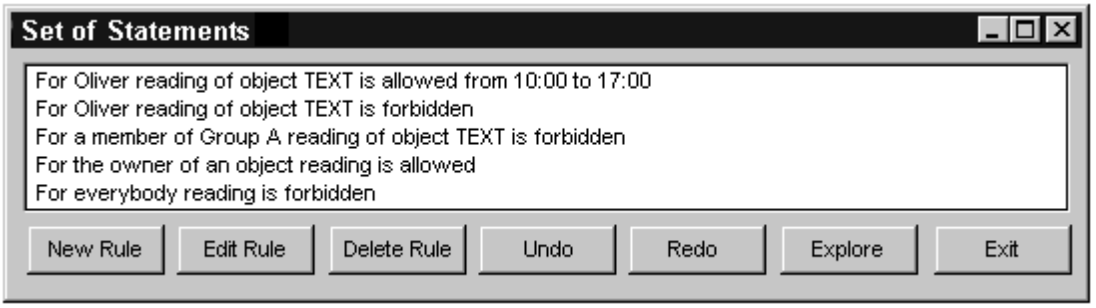

Figure 5: Window listing the relevant access statements

Tailorable Filters of an Awareness Service

The awareness service, which distributes the events among the different users, is a mechanism, because the users cannot tailor it. It is triggered whenever the producer activates one out of the set of prespecified functions. It is a real global mechanism because it is essential in sending the events. The producer's filter is a function triggered by the awareness service whenever an event is generated. It is a real global function, because it determines which event is distributed. The recipient's filter is a function, triggered by the awareness service whenever an event is to be delivered. It is a supportive global function, because it operates on local data (the incoming events).

As already mentioned in section 2, both of these triggered functions are equipped with a layered tailoring environment. The consumer's filter can be tailored by

---

[8] Note that LinkWorks has not implemented application-wide undo/redo functions.

selecting among prespecified interest profiles. The interest profiles can be created by means of another tailoring language. The producer's filter can be tailored by selecting recipients or groups of recipient to whom the events are distributed. The groups of recipients can be tailored by means of another tailoring language.

Mechanisms of the dialog interface support users in finding the tailoring environments which define the filters. The access point for the tailoring environment of the producer's filter changes its color briefly whenever an outgoing event is created. The tailoring environment of the recipient's filter is accessible via the presentation of the events at the interface. The access points of the second level tailoring functions are placed into the window which allows to activate the first level tailoring functions. Pull down menus with which the users can select the interest profile (in case of the recipient's filter) and the groups of users (in case of the producer's filter) are extended by an item representing the higher level tailoring function. Choosing this item the users activate the window to specify the second level tailoring functions.

Another mechanism of the first level tailoring functions' dialog interface allows the users to get a survey on the actual configuration of the filter. It displays the configuration of all eighteen different events in one window. The window which shows the selected interest profiles of the recipient's filter is shown in figure 3. The producer's filter is designed similarly.

A triggered function of the output interface suggests names for newly created interest profiles automatically. Nevertheless a user can modify this name manually. It is used to represent the interest profile at the recipient's filter's first level tailoring function. Another triggered function of the output interface allows users to name newly created groups of recipients. These names are used to represent these groups at

the first level tailoring function of the producer's filter.

5 Exploration Environments

So far we have described three tailorable groupware tools. The extended IFIP-model allows us to describe these tools by means of a common terminology. In the following, we will hint to problems users experienced in handling the three groupware tools described before. Dealing with these problems, we present different approaches to support users self-directed learning of these tools. These approaches can be seen as different realizations of the concept of exploration environments for groupware.

5.1 Exploring the Search Tool

The search tool was introduced to a body of the Northern German State government in Bonn. During the field test it turned out that users had problems understanding the functioning of unknown search tools, because they could not work out the exact meaning of certain components (e.g. output ports of switches) or the outcome of their interplay. The problem to explore tailored search-tools became even more difficult due to the interplay between the search tool and the access rights on shared data. Users had difficulties to judge whether the outcome of an inquiry was due to a wrong understanding of the tool or missing search permissions (cf. Wulf 1999a). Therefore, we decided to extend the search tool by an exploration environment.

The exploration environment of the search tool allows to explore the use of search tools as well as to tailor search tools and compound components. The exploration environment is presented by means of a window which has a specific frame color. It contains the search tool which was active before starting the exploration environment. The exploration mode is indicated by a different background color of the search tool.

Moreover, a document browser is part of the exploration environment. To explore a search tool, other users' documents and desktops have to become visible and accessible. Such an option cannot be realized with other users' real data and desktops, since it would violate their privacy. Consequently, the document browser allows creating other users' simulated desktops populated with experimental data. The experimental data are equipped with access rights. These access rights can be manipulated within the document browser. In the exploration environment the search tool is executed on the data structure given in the document browser.

Figure 6 shows a screen shot of the exploration environment. The big window in the middle represents the document browser. To ease the users' understanding, the browser's menu is designed similarly to the main menu of LinkWorks. Nevertheless, users can activate only those functions of the menu, which allow to create or to delete documents and those which allow to modify access rights. There are further buttons which allow to populate the simulated desktops more efficiently (e.g. creation of a set of documents at random).

The lower window represented in figure 6 shows the search tool in assembly mode. The toolbox is presented in the upper right corner (note the different background color). Both of these windows behave regularly except for those functions which allow to modify the state of the original search tool (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. Nevertheless, in the exploration environment the search tool can be modified by means of the tailoring functions. Thus, users can modify the given search tool and create compound components.

The button on top of the window allows to leave the exploration environment. If a

user decides to do so, he is asked whether he wants to store or abandon the outcome of the tailoring activities in the exploration mode. In case he decides to store a newly designed search tool, he is asked whether he wants to store the experimental data generated by the document browser, as well. If he does so, the next time the exploration environment of the search tool will be populated with these experimental data from the beginning.

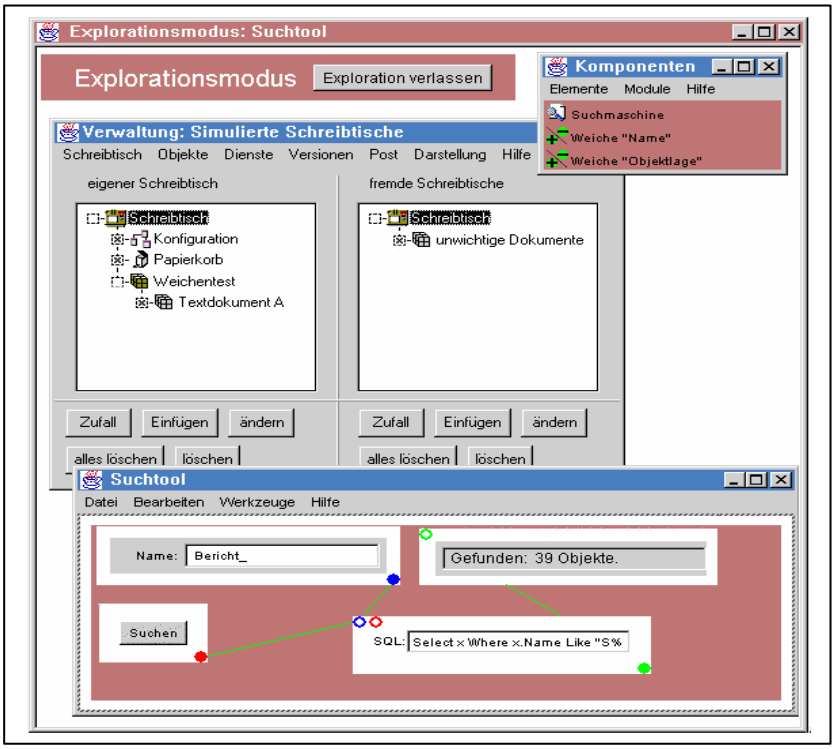

Figure 6: Exploration environment of the search tool

The user can also store a newly generated compound component when leaving the exploration environment. In this case he can decide to store the search tool in which the compound component was embedded and the experimental data which he may have created to test this search tool together with the compound component. Then, these data become part of the environment which can be activated to explore this compound component. The full search tool executed on the experimental data may provide a characteristic example of how to use the respective compound component.

Users can test the search-tool example and deduce the functioning of the compound component from the outcome of these experiments. Besides they can find out about the particularities of the compound component by replacing it with another one of the same type. After leaving the exploration environment, the user returns to the search tool environment which was active before he started the exploration.

As the exploration environment allows to use and to tailor full search tools, compound components and elementary components, one can distinguish at least four scenarios on how to use the exploration environment. First it can be used to explore a given search tool. Second it can be used to explore a compound component or an elementary search tool component. Third it can be used to tailor search tools, and fourth, it can be applied to tailor compound components. We will briefly describe how to apply the exploration environment in the different scenarios.

In the first scenario a user may want to find out whether the search tool presented in figure 6 is finding documents only on his own desktop or also on other users' desktops. The user can switch into the exploration environment by selecting this option from the "help" menu bar whenever the search tool is active. When the user switches into the exploration environment a new window pops up (outer window in figure 6). It contains a second window in which the search tool to be explored is displayed. Moreover, the user can open the document browser to find out on which data the search tool is operating in the exploration environment. Now he can execute an inquiry and compare the search results which the experimental data given in the document browser. Then he can either create or delete documents or change the access rights in the document browser. The user can refine his understanding of the way a certain search tool works by executing it on different sets of experimental data.

In the second scenario the user may want to know how an elementary or a

compound component works. After selecting a single component, he can switch into the exploration environment from the help menu. Here again two new windows pop up. The search tool window contains the component to be explored which is now embedded into a full search tool. Moreover, the user can have access to the experimental data by means of the document browser. By trying the search tool and exchanging the individual components the user can find out about the meaning of these components. For instance, he may exchange the component to be explored against one of the same type to find out differences with regard to the outcome of the search inquiries.

In the third and fourth scenario the users tailor new search tools or compound components in the exploration environment. Therefore, the tool box is included in the exploration environment (cf. figure 6). Like in the search tool's tailoring mode, users can select components or compound components, place them into the search tool window and connect them by creating lines between their ports. A compound component is generated by tearing a frame around a certain subset of components and store it under a specific name. A new search tool is generated by storing the result of such tailoring activities under a certain name. To test newly created search tools and compound components experimental data can be generated and stored together with these artifacts.

5.2 Exploring the Access Control

Due to the problems of the original implementation of the access control (cf. section 3.2), we prespecified only three access profiles in LinkWorks among which the users could select. During their use it turned out that users had problems to understand the access rights which were specified by two of the three profiles.

Therefore, they selected almost always the one profile which did not impose any access restrictions on other users. Moreover, it turned out that one of the other two profiles was tailored in a faulty way by the project team in one of the state ministries. As it took a lot of effort to test such a profile from the perspective of the different users involved, the profile was not sufficiently checked after being tailored. The mistake became visible by accident after months of use in one of the state ministries(cf. Wulf 1999c). Thus, supporting exploration seems to be a crucial function for any tool which allows to specify access rights in groupware.

While the statement-based approach allows users to tailor the access rights even beyond the selection of prespecified access profiles, an automatism is implemented to handle inconsistent statements. This automatism has to be understood by the users, as well. Therefore in our approach an environment to explore the interplay of the different access statements is of special importance.

Contrary to the search tool, the exploration environment of the access control function does not contain any tailoring functions. So there is only one scenario for the use of the exploration environment. Given a certain set of relevant access statements the user can explore their meaning. If he wants to tailor the access statements, the user has to leave the exploration environment.

As access control is a triggered function, its existence is visualized via the object to which the access statements refer. Concerning such an object, users can activate a dialog window which lists the relevant statements (cf. section 4.3). The button which allows to access the exploration environment is placed in this window (cf. figure 5). Thereafter, a new window appears in which the activator can describe a hypothetical situation of use. In the new window the activator can specify another user, a user group or one of the roles with regard to the electronic circulation folder (lower left

window in figure 7). Concerning this hypothetical situation the activator can test the access rights resulting from the interplay of the given statements. Based on this situation, the underlying window lists in its upper part those statements which define the access rights to the object prespecified when the exploration environment is started. To clarify the meaning of partly inconsistent statements the window is extended furthermore. In its lower right part, it describes the other users' access rights in plain text concerning all the seven functions mentioned in section 3.2 (cf. figure 7). So the user is informed about the access rights which are valid in the hypothetical situation. Now, the user may modify the hypothetical access situations to find out which access rights to the preselected object are specified by the given access statements concerning these situations.

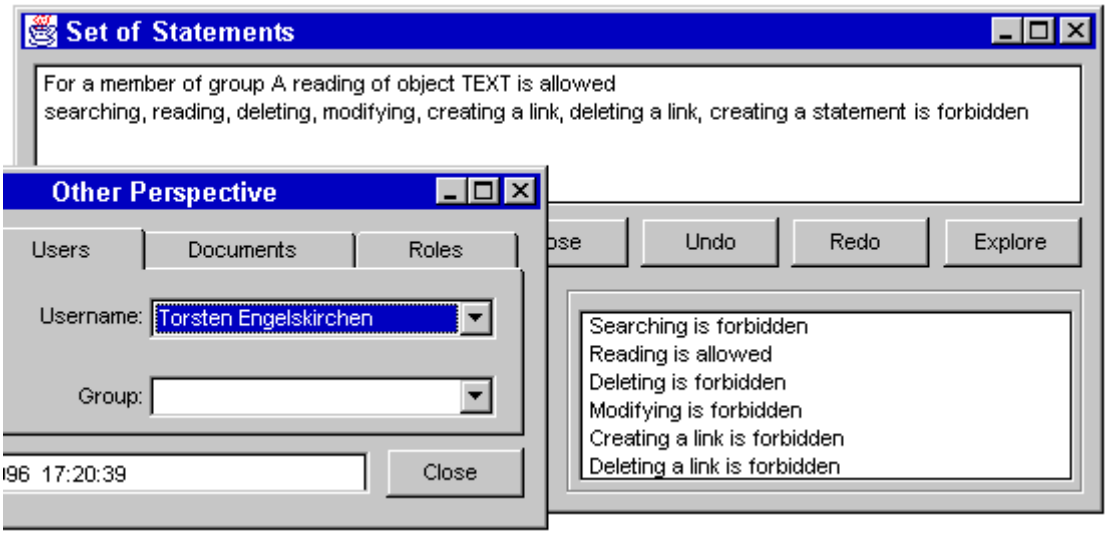

Figure 7: Exploration environment allowing to evaluate the set of tailoring statements from different perspectives

Beyond the exploration of the access rights of the preselected object or object group, the activator can also decide to explore the access rights of another object of

his electronic workspace. By selecting a new object or object group the activator can find out about the access rights from the perspective of other users, user groups or roles.

Having left the exploration environment, the user is back to the window shown in figure 5. Now he can create, delete or edit statements or use the undo/redo functions to tailor the given set of statements. Then he can open the exploration environment again.

5.3 Exploring the Tailorable Filters of the Awareness Service

The awareness service of the tool to exchange tailored artifacts has not been evaluated in the context of the POLITeam project. In this project, Fuchs (1997) has implemented and evaluated an awareness service for a workspace to share documents. Contrary to our design he just realized the recipient's filter. Nevertheless the specification of this filter was more complex, because there were more event types and more complex display situations given. Evaluating this application in the federal ministry, he found that users had problems to understand why given specifications of the recipient's filter led to the display of certain events (cf. Fuchs 1997, p. 155). While these findings result from the use of a single filter, support for exploration seems to be even more crucial in case two tailorable filters interact.

Similar to the tool to tailor access rights, we have realized exploration environments for the producer's and the recipient's filters, which do not contain tailoring functions. However, each filter can be modified by two layered tailoring functions. So there are two different exploration environments for each filter and there are different scenarios on how the two exploration environments for each filter could be used. In the following we will describe mainly how the explorations environments

for the recipient's filter works. The exploration environments of the producer's filter work in a similar way.

The recipient's filter is a triggered function. The display of the incoming events is designed in a way that the first level tailoring function can be activated directly (cf. section 4.3). We equipped the window of the first level tailoring function with a button to enter the exploration environment for this filter. Thus, whenever the users have tailored the filter they can test the outcome of these activities.

After the exploration environment is activated, it comes up a window which asks the user to specify an event. The distribution and display of this event will be simulated in the exploration environment. Next a new window appears which is subdivided into two parts (cf. figure 8). These parts differ in background color and title. The left part represents the simulated workspace of another user. It is titled: "Producing Events - Other Users' View". The right part represents the activator's workspace. It is titled: "Receiving Events - Own View". There are also some gray fields in the window, which contain functions to describe and control the exploration environment (e.g. exit the exploration environment, return to the event specification).

By means of a blinking arrow next to the corresponding pull down menu, the activator is asked to attribute the left side of the window to one of the other users. To do so, the pull down menu offers a list of all the users registered in the application.

In this way the activator creates an event on the side of the other user. Now the different functions and mechanisms of the awareness service are visualized automatically. For the visualization we use the metaphor of water dropping through two funnels (cf. figure 8). A drop of water symbolizes an event, the gravity indicates the awareness service and the two funnels symbolize the filters. Each funnel can be either open or closed. Text marks describe the different stages of the event handling.

The water drop passes the first funnel if the producer's filter of the selected user allows to publish this event. It passes the second funnel if the recipient's filter of the activator allows to display the event. If the event is eliminated at one of the filters this fact is symbolized at the corresponding funnel and described textually.

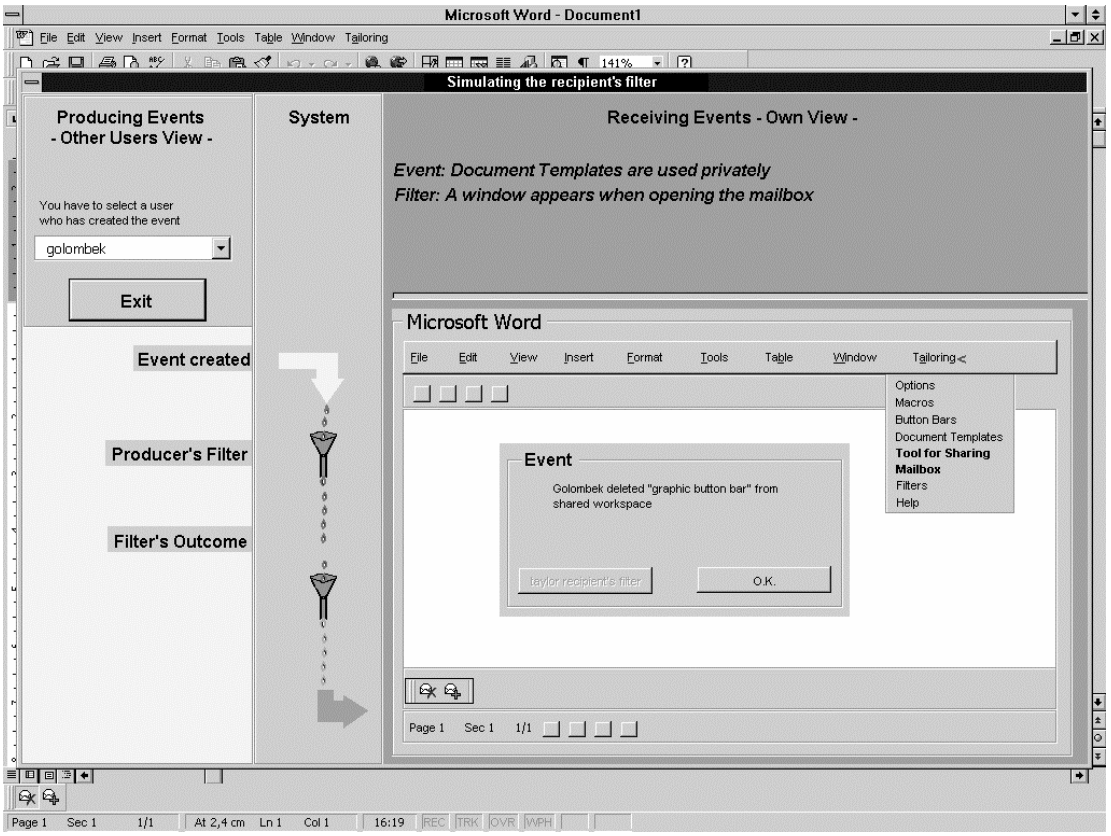

Figure 8: Environment to explore the recipient's filter

In case the event has passed both of the filters a simulated MS-Word environment is opened on the right side of the window (activator's side). This environment simulates the activator's functionality. Most of the menu bars are not available except the one which allows to open the tool to share tailored artifacts and the mailbox to receive sent ones. By means of a blinking arrow next to the corresponding menu item, the activator is hinted to the fact that he can check out the different situations in which

he might be notified about an incoming event. In the simulated MS-Word environment the events are displayed with the intensity prespecified in the recipient's filter.

When allowing users to explore the display of the events generated by other users, we have to take the specification of the producer's filter into account. As this filter is a function on the side of the other user, in the real application it is not accessible by the recipient. In exploration mode we have tackled the resulting problem in two ways. First, any user who specifies his filters can set a flag, which indicates whether these specifications are accessible to other users for the purpose of exploration. In the screen representing the recipient's filter these flags can be set for any event type and are placed behind the corresponding interest profiles (cf. figure 3). If a user allows to explore the relevant part of his producer's filter, the exploration mode applies these data to determine the behavior of the first funnel. If the filter is not accessible for exploration, a question mark appears in the first funnel. However, the exploration environment lets the event pass the producer's filter and transmits it to the recipient's filter. Finally, when the event is displayed in the exploration environment, the missing information about the specification of the producer's filter is indicated textually.

Having explored how an event generated by another user is displayed on his screen, the user can create new events originated by other users. By investigating how different events are displayed at his simulated interface, the user can learn about the way a certain configuration of the recipient's filter works. Of course, the user can leave the exploration environment at any time. In this case he returns to the window where the recipient's filter is tailored.

In the second scenario a user is equipped with another exploration environment to learn about the effects of newly created interest profiles. If a user wants to add or edit

an interest profile with regard to a certain event, he can activate the second level tailoring function directly (cf. section 4.3). To test newly created interest profiles an exploration environment can be activated in this window. It is implemented similarly to the one described in the first scenario. Nevertheless, the activator does not have to specify the event created by the other user. It is already known to the system, because the interest profiles are created for each type of event individually. Thus, the activator can start directly by selecting the other user whose event production is simulated in the exploration environment. After leaving the exploration environment the activator returns to the window where the interest profiles can be created.

In the third scenario another exploration environment is applied to learn about the effects of the producer's filter on the distribution of events. Like the recipient's filter, the producer's filter is a triggered function whose execution has been made visible to the users. In this state the first level tailoring function can be activated directly (cf. section 4.3). We equipped the window of the first level tailoring function with a button to enter the exploration environment for this filter.

After the user activates the exploration environment, a window appears which is again subdivided into two parts. The left part represents the activator's tool for sharing tailored artifacts. The right part represents the workspace of another user. As in the case of the recipient's filter, there are some gray fields in the window, which describe the exploration environment and contain functions for its control.

By means of blinking arrows next to the corresponding pull down menu, the activator is asked to select an event to be created and a recipient to whom the event will be distributed. Now the simulated sharing tool appears. A button is indicated by a blinking arrow of which activation creates the selected event. By pressing this button the user creates the prespecified event in the exploration environment. The metaphor

of water dropping through two funnels is applied again to visualize the influence of the two filters and the awareness service on the event distribution. The producer's filter is of special interest, because the activator has tailored it. Concerning the recipient's filter, the other users can decide whether their specification is accessible for exploration. The handling of missing access is similar to the one in the exploration environment of the recipient's filter.

If the event has passed the two filters the activator can move into the right part of the window and activate another window where the MS-Word application of the other user is simulated. In this window the activator can try out when and how the event is displayed on the side of the recipient. Afterwards, the activator can choose to learn about the effects the current specification of the producer's filter has on the distribution of events to other recipients. He just has to specify another event type and another recipient and to follow its distribution. Alternatively he can push a button to leave the exploration environment.

In the fourth scenario the user wants to find out which effect a newly created group of users has on the distribution. Contrary to the second level tailoring function of the recipient's filter (deferring interest profiles), the definition of a user's group seems to be more easily to understand for the users. Therefore, we decided not to build a specific exploration environment for this second level tailoring function. Nevertheless, users can explore the effects of a newly created user group by means of the exploration environment of the producer's filter. After having specified a new user group, the user can step back to explore the producer's filter as it is described in the third scenario.

6 Exploration Environments for Tailorable Groupware

Obviously ordinary functions belonging to the organization interface are difficult to explore. To approach this problem, we have developed three examples how to support users in exploring tailorable groupware. In this section we will generalize these findings. Using the terminology of the extended IFIP model, we will develop general guidelines for the design of exploration environments in groupware.

The existence of triggered functions and mechanisms on the organization interface is often hard to perceive on the user's own interface. Functions existing on the other users' interface are hidden anyway. Users can often assume that a real global function exists on the other users' interfaces when looking at perceivable state transitions on sent or shared data. However, supportive global functions on other users' interfaces are completely hidden.

Moreover, the effects of executing a function are hard to understand if only seen from the perspective of one user. This problem is worsened if the state transition following the execution of a function results from an interplay with functions tailored or activated on different users' interfaces. For instance, supportive global functions can only be triggered by the execution of other user's real global functions or global mechanisms. Therefore, these functions can only be explored if the user has access to the interface of other users. Exploration environments allow to make hidden functions and mechanisms visible and to simulate the interfaces of other users to try these functions out.

The exploration of tailoring functions poses a special problem. The tailored artifact resulting from their execution is only meaningful to users with regard to the tailorable function. Therefore, it is necessary to execute the tailorable function when a user tries to explore the tailoring function. In case of layered tailoring functions, second level

functions influence the first level ones which again influence the ordinary function. Thus, the design of exploration environments has to find appropriate solutions to tackle these cases.

After clarifying requirements for exploration environments on an abstract level, we will look at the concrete design of this functionality which belongs to the dialog interface but simulates aspects of all the other interfaces of the application.

First we discuss how to design the access to these functions. Mechanisms of the dialog interface should make the exploration environment directly accessible whenever a corresponding normal function is activated. As triggered functions are not represented at the user interface, the dialog interface should be designed in a way that the exploration environments can be activated directly from the corresponding first-level tailoring function. Direct activation of the environments can be realized in two ways. Either the access point of the exploration environment is placed closely related to one of the corresponding normal function, or there exists a consistent way how to deduce the activation of the exploration environment from the one of the corresponding normal function (cf. Wulf 1999b).

Second, we discuss the design of tailoring environments. Exploration environments have to simulate the original design of the explorable functions concerning all three aspects of their users interfaces. While the technical aspects of the organization interface require a close similarity between the original and the simulated function, it is essential to differentiate between the simulated and the original ones on the input/output and dialog interfaces.

The key function of exploration environments for groupware is the possibility to choose the perspective of other users' interfaces. This functionality is typically realized by selecting from a list of all relevant users, user groups or roles registered in

the application.

Depending on the function to be explored, the interface of the other users may be realized in a descriptive or an experimental way. In case of the access control in groupware, the functions of the other users' interface, which allowed to have access to the selected object, were not implemented in the exploration environment, but the effects of their execution were described in plain text. The plain text was automatically generated from the specification of the access rights. By contrary, in case of the search tool and the awareness service the relevant functions of the other users' interface were realized in the exploration environment. Thus, the users could experiment with the interdependencies of the different functions.

The first approach can be seen as the implementation of a neutral mode with regard to the effects which a function's execution has on other users' interfaces (cf. section 2). It is a viable solution if the state transition resulting from the execution of a function on the side of the other user can be easily expressed in plain text. This is the case with the access rights in groupware. The effect of the access statements on the seven relevant functions on the other user's side can be expressed rather easily in plain text (see figure 7). Except for the function to select another user's perspectives, there is no other function or data needed in the exploration environment. The main challenge, when one designs exploration environments in a descriptive way, is the output interface. A mechanism has to be developed which generates the plain text dynamically according to the actual state of the tailorable function. Such a description has to contain all the relevant preconditions and consequences of a function's execution.

The second approach requires the realization of the relevant functions and of their user interfaces in the exploration environment. To explore the dependency among

these functions, experimental data must be given in the exploration environment. Therefore, further functions may have to be implemented in the exploration environment, which allow to generate experimental data (e.g. the document browser in case of the search tool). The functions of the exploration environment should carry out the same state transitions on the experimental data as the original functions would carry out on the original data.

When designing an exploration environment for a specific function, one has to decide which other functions or mechanisms of the organization interface should be included. This decision has to be made concerning the other ordinary functions and the different layers of the corresponding tailoring functions. As to other ordinary functions we believe that the tailoring environment should be kept as small as possible. With regard to a triggered function only those other functions and mechanisms should be included in the environment which can generate events leading to the function's execution (e.g. in case of the recipient's filter, the producer's filter and the awareness service seem to be relevant). With regard to a normal function, it should include only those other functions which influence the function's execution (e.g. in case of the search tool the triggered access control function).

Whether tailoring functions should be included in the exploration environment of the tailorable functions is still an open question (in case of the search tool we have included all tailoring functions, in case of the awareness filters we did not include the second level ones in the exploration environments of the first level tailoring functions). If the tailoring functions are included, the exploration environments become more complex because there are different scenarios on how to apply the same environment. In case they are excluded, additional exploration environments for each layer of the tailoring languages have to be created. Such an approach requires

concepts on how to explore artifacts tailored on higher levels of complexity. These artifacts have to be places in a technical context to enable the users to explore their behavior (e.g. further components to create a whole search tool or preselected event types to test an interest profile).

The exploration environment should be implemented as close as possible to the dialog interface of the original application. Nevertheless, the fact that many functions of the original application are not part of the exploration environment should become instantly visible to the user. Their access points can be disabled (e.g. in the case of the MS-Word simulation) or they can be put into neutral mode (e.g. like in case of the search tool). Moreover, it can be helpful to visualize hidden state transitions following the execution of triggered functions or mechanisms (e.g. like the dropping water metaphor in case of the awareness service). Especially if the exploration environments are more complexly structured, additional mechanisms can be helpful to guide the users' exploration (e.g. the blinking arrows in case of the awareness filters indicate the sequence of activities required).

The exploration environment should be implemented as close as possible to the input/output interface of the original application. Nevertheless, it has to differ at least in three points. First, it is necessary to express the differences between simulated interfaces of the other users and the one of the activator. We did this by means of colors and descriptive text. Second, space and means to display the additional functions of the exploration environment have to be found (such functions are, for instance, the selection of different perspectives, the creation of experimental data or the exit from the exploration environment). Third, spatial limitations of the output interface may require to simplify original windows and to sequentialize the output of the exploration environment by means of different windows.

Finally, we will discuss how to design the exit from an exploration environment. A button to exit the tailoring environment should always be visible within the exploration environment. Whenever this function is activated the system should be set back to the state before the exploration environment was started. In case there are not any tailoring functions included in the exploration environment, this approach is easy to implement. If the users may create experimental data or tailored artifacts, they have to be asked whether they want to store them before leaving the tailoring environment (cf. section 5.1).

7 Empirical Evaluation

Having clarified the concept of exploration environments, one has to evaluate whether this concept supports users' learning. This question requires empirical investigations. Therefore, we have carried out a field test and an experimental study to deal with this question.

The results of the experimental study indicate that exploration environments have a positive effect on self-directed learning activities. Taking the awareness service as an example of a tailorable application (cf. sections 3.3, 4.3 and 5.3), we compared the learning activities of two groups of users. The one was equipped with exploration environment while the other was not. It turned out that users whose tool was equipped with exploration environments had a better understanding of the tailorable awareness service and carried out tailoring tasks more efficiently (cf. Wulf 2000).

While the description of this experimental study is beyond the scope of this paper, we will report about the results of a workshop where the search tool's exploration environment was evaluated. This workshop was the final event of a cooperative design process in which the tailorable searchtool was developed together with users of

a body of the political administration (cf. Wulf 1999 a). The workshop was held at the research lab of the University of Bonn. Four users of the government body and three researchers took part in the workshop. Three of the four users participated before in the search tool's design process. One of the participants of the workshop acted as a local expert in supporting other users of the body, while the others were normal users.

The workshop took about two hours. It started by a brief talk of one of the researchers giving an introduction to the component-based search tool and its exploration environment. Afterwards there was a short presentation of the search tool's functionality and its exploration environment. And then the different users were asked to tailor the search tool, e. g. they had to choose among alternative search tools or to modify given tools. Having finished tailoring, the users were asked to explain how the tailored tools function. At the end of the workshop we had a general discussion.

To understand given search tools, the users applied the exploration environment. The local expert found it, for instance, helpful to explore a search tool which contained a result switch and two display windows. By looking at the experimental data and trying out different queries, he corrected a misunderstanding on how the switch subdivides the search results and came up with an appropriate perception. In similar ways the users arrived at predicting the functioning of different search tool alternatives. Nevertheless they needed occasionally some help in handling the exploration environment. With regard to the experimental data they rather used the stored ones than creating new ones. This may be due to the fact that we had equipped the exploration environments already with data fitting to the learning tasks. One of the users had problems to understand the additional functions allowing to populate the simulated desktops efficiently.

Discussing the design of the exploration environments, the users found the specific background colors of the windows which belonged to the exploration mode essential to understand on which data they were actually searching. The users found it helpful that there was a resemblance between the menu of the document browser and the one of the original LinkWorks application. It eased their orientation within the exploration environment. The fact that most of the menu items were put into neutral mode was well perceived, as well.

During the exploration of one of the search tool alternatives, a user suggested to realize the exploration environment in a descriptive way. Trying to understand a search tool which divides the search results by means of a switch into two windows, he came up with the following idea. A text accessible via the context menu should describe which type of documents will be displayed in each of the windows. He found such a solution more easily usable than the given exploration environment. This remark seems to indicate a desire to get equipped with less complex variations of exploration environments. Nevertheless the automatic generation of textual descriptions is a rather complex problem in case there are flexible search permissions and different switches subdividing the search results into various windows. Moreover, it is questionable whether users will be able to understand all the implications of such a text.

In the final discussion, we asked the users whether and how they would apply exploration environments in their daily work. Two of the users stated that they prefer the exploration environments from reading help menus (“ I rather like to play around than read.”) or asking colleagues (“You do not want to show to the others that you do not understand it.”). The local expert stressed that he would apply exploration

environments whenever he had built new search tools. This way he could test whether the tools do what they are supposed to do.

The other two users who were in general less skillful in handling the LinkWorks application were also less convinced of the usefulness of exploration environments. One of them stated that it would depend on the support provided to her by other members of the organization. As the exploration environment looked rather complex to her, she felt uneasy to handle it herself. There was a general agreement among the participants of the workshop that the handling of these exploration environments would be too difficult for some of their colleagues. (" That would be too difficult for Mr. R.").

These results indicate that the exploration environments implemented in case of the search tool are obviously best suited for those users who have reached already a certain level of skills in handling computers.

8 Conclusion

We have presented three tailorable groupware tools. To support explorative learning of each of these tools we have implemented additional features, which simulate the system behavior at other users' interfaces. To describe the users' problems and to generalize the design-related finding, we have developed a model of the user interface of tailorable groupware. This model was applied to formulate design guidelines for exploration environments in groupware.

The design of the tailoring environments presented in this paper was based on user requirements coming up in the POLITeam project. The design solutions presented in section 5.1 and 5.3 have been evaluated empirically. In this paper we have focussed on the qualitative evaluation of the search tool's exploration environment by means of

a workshop. Further work is required to investigate on the factors which influence the use of exploration environments, and on their effects on processes of self-directed learning. The presented design options have to be evaluated in more detail. Moreover, we need to find out how different types of users handle the additional complexity contained in the tailoring environments. We may need to look for exploration support in groupware which is more appropriate for unskilled users. Exploration environments reduce already this complexity in comparison to applications which support learning by allowing to log-in under different identities (cf. section 2). In the latter case, the users have to deal with two complete interfaces and create their experimental data themselves while in our case only those functions, essential for learning, need to be handled. Nevertheless, alternative approaches and further simplifications should be investigated to support self-directed learning of tailorable groupware.

Exploration environments as they are proposed in this paper, allow individual users to experiment with groupware functions. Thus, they do not need to ask other users for access to their interfaces. Assuming that the learning styles of individual users differ considerably, we believe that this will lead to less mutual disturbances. However, the use of exploration environments diminishes occasions for collective learning processes. These processes are nevertheless crucial when users learn tailorable applications (cf. Mackay 1990, Nardi 1993, Trigg and Bødker 1994). Therefore, the effect of the usage of these exploration environments on collective learning processes has to be investigated, as well.

Nevertheless, due to the fact that exploration environments are realized in groupware, the multi-user character of these applications can be used to support cooperative learning. For instance, users could be made aware of other users who experiment with the same function. Thus, they could either meet face-to-face or

exchange their experiences via application immanent communication channels.

Though the concept of exploration environments is applicable to a wide range of groupware functions, some preconditions have to be met. First, the behavior of the function to be explored needs to be deterministic. This precondition is typically given within computer applications. Nevertheless, it would not be wise to develop an exploration environment for the video or audio transmit function of a video conference, in case it uses, for instance, the internet as a transport medium. As the audio and video transfer rate is not deterministic, it does not make sense to simulate these functions. Second, the behavior of the functions to be explored should not depend on real-time input of the different users. As the input interfaces of the different users can only be accessed sequentially, it is not possible to simulate the effects of synchronous users' input. For instance, locking functions of synchronous group-editors cannot be explored in the way proposed here. Third, only those groupware applications can be equipped with exploration environment which input/output interface allows to simulate the input and output of other users. While this requirement is typically given with computer screens and keyboards, with hardware-oriented output interfaces like telephone or telefax it is often not met.

In these cases it is not possible to realize exploration environment. In many other cases we believe that the concept of exploration environments will become an important means to support users in learning about groupware functions in general and tailorable ones in particular.

9 References

Abowd, G. D.; Dix, A. J.: Giving undo attention, in: Interacting with Computers, Vol. 4, No. 3, 1992, pp. 317-342

Banavar, G.; Doddapaneni, S.; Miller, K.; Mukherjee, B.: Rapidly Building Synchronous Collaborative Applications By Direct Manipulation, in: Proceedings of CSCW '98, ACM-Press, New York, 1998,

pp. 139 - 148

Bentley, R.; Dourish, P.: Medium versus Mechanism. Supporting Collaboration Through Customisation, in: Marmolin, H.; Sundblad, Y.; Schmidt, K. (Hrsg.), Proceedings of the Fourth European Conference on Computer Supported Cooperative Work - ECSCW '95, Kluwer, pp. 133-148

Berlage, T.: A selective Undo mechanism for graphical user interfaces based on command objects, in: Transactions on Computer-Human Interaction, Vol. 1, No. 3, 1994, pp. 269-294

Carroll, J. M.; Mack, R. L.: Actively learning to use a word processor, in: Cooper, W. E. (ed.): Cognitive aspects of skilled typewriting, Springer: New York, 1983, pp. 259 – 281

Carroll, J. M.; Mack, R. L.: Learning to use a word processor by doing, by thinking, by knowing, in: Thomas, J. C.; Schneider, M. L. (eds): Human factors in computer systems, Ablex: Norwood, NJ 1984, pp. 13 – 51

Darlington, J.; Dzida, W.; Herda, S.: The role of excursions in interactive systems, in: Journal on Man-Machine Studies, Vol. 18, 1983, pp. 101-112

Dewan, P.; Shen, H.: Flexible meta access-control for collaborative applications, in: Proceedings of CSCW '98, ACM-Press, New York, 1998, pp. 247 - 256

Dutke, S.: Mentale Modelle: Konstrukte des Wissens und Verstehens, VAP, Göttingen 1994

Dzida, W.: Das IFIP-Modell der Benutzerschnittstelle, in: Office Management, Sonderheft 31, 1983, pp. 6 – 8

Ellis, C.A., Gibbs, S.J. and Rein, G.L.: Groupware. Some Issues and Experiences, in: Communications of the ACM, Vol. 34, No. 1., 1991, pp. 39 - 58

Engelskirchen, T.: Exploration anpaßbarer Groupware, M.Sc. Thesis, Department of Computer Science III, University of Bonn, 2000

Frese, M.; Albrecht, K.; Altmann, A.; Lang, J.; Papstein, P. V.; Peyerl, R.; Prümper, J.; Schulte-Göcking, H.; Wankmüller, I.; Wendel, R.: The effects of an active development of the mental model in the training process: experiment al results in a word processing system, in: Behaviour and Information Technology, Vol. 7, No. 3, 1988, pp. 295-304

Fuchs, L.: Situationsorientierte Unterstützung von Gruppenwahrnehmung in CSCW-Systemen, PhD-Thesis, Department of Computer Science, University of Essen, 1997

Gantt, M.; Nardi, B. A.: Gardeners and gurus: Patterns of Cooperation among CAD users, in: Proceedings of CHI '91, May 3-7, 1991, Monterey, CA, pp. 107 – 117

Green, M.: Report on Dialog Specification Tools, in: Pfaff, E. (ed.): User Interface Management Systems, Springer: Berlin, pp. 9 – 20 (quoted from: Edmonds, E. (ed.): The seperable User Interface, Academic Press: London 1992, pp. 195 – 211)

Greif, S.; Janikowski, A.: Aktives Lernen durch systematische Fehlerexploration oder programmiertes Lernen duch Tutorials, in: Zeitschrift für Arbeits- und Organisationspsychologie, Vol. 3, 1987, pp. 94-99

Henderson, A.; Kyng M. (1991): There's No Place Like Home. Continuing Design in Use. in: Design at Work, Lawrence Erlbaum Associates, Publishers, pp. 219-240

Höller, H.: Kommunikationssysteme - Normung und soziale Akzeptanz, Vieweg: Braunschweig u.a. 1993

Howes, A.; Paynes, S. J.: Supporting exploratory learning, in: Proceedings of INTERACT'90, North-Holland, Amsterdam, S. 881 – 885

ISO 9241: Ergonomic requirements for office work with visual display terminals (VDTs) Part 10: Dialogue Principles

Johansen, R.: Current User Approaches to Groupware, in: Johansen, R. (ed): Groupware, Freepress,

New York 1988, pp. 12-44

Johnson-Lenz, P.; Johnson-Lenz, T.: Post-mechanistic groupware primitives: rhythms, boundaries and containers, in Greenberg, Saul (eds): Computer-supported Cooperative Work, Academic Press, London, 1991, pp. 271-293

Kahler, H.: Developing Groupware with Evolution and Participation: A Case Study, in: Proceedings of the Forth Biennial Conference on Participatory Design (PDC´96), Boston, MA, Nov. 13 - 15, 1996, pp. 173 – 182

Kahler, H.; Mørch, A.; Stiemerling, O.; Wulf, V.: Tailorable Systems and Cooperative Work, Special Issue of Computer Supported Cooperative Work: The Journal of Collaborative Computing, 1999a (in press)

Kahler, H.; Stiemerling, O.; Wulf, V.; Höpfner, G.: Gemeinsame Anpassung von Einzelplatzanwendungen, in: Proceedings of Software-Ergonomie '99, Teubner, Stuttgart, 1999, pp. 183 - 194

Kamouri, A. L.; Kamouri, J.; Smith, K. H.: Training by exploration: facilitating the transfer of procedural knowledge through analogical reasoning, in: Journal of Man-Machine Studies, Vol. 24, 1986, pp. 171-192

Lefrancois, G. R.: Psychological Theories and Human Learning: Kongors Report, Wadsworth: Belmont, Ca 1972

Mackay, Wendy E.: Users and customizable Software: A Co-Adaptive Phenomenon, PhD-Theses, MIT, Boston (MA) 1990

MacLean, A.; Carter, K.; Lövstrand, L.; Moran, T: User-tailorable Systems: Pressing the Issue with Buttons, in: Proceedings of the Conference on Computer Human Interaction (CHI '90), April 1-5, 1990, Seattle, Washington, ACM-Press, New York 1990, pp. 175 –182

Malone, Th. W.; Lai, K.-Y.; Fry, Ch.: Experiments with Oval: A Radically Tailorable Tool for Cooperative Work. in: Procceedings of CSCW '92, ACM-Press, New York, 1992, pp. 289-297

Mørch, A.: Method and Tools for Tailoring of Object-oriented Applicatios: An Evolving Artifacts Approach, PhD-Thesis, University of Oslo, Department of Computer Science, Research Report 241, Oslo 1997

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

Nardi, B. A.; Miller, J.: Twinkling lights and nested loops: Distributed problem solving ans spreadsheet development, in: International Journal of Man Machine Studies, vol 34., pp. 161 – 184

Oberquelle, H. (1994): Situationsbedingte und benutzerorientierte Anpaßbarkeit von Groupware. in: Hartmann, A. et al. (eds), Menschengerechte Groupware, Stuttgart, pp. 31-50

Oppermann, R.; Murchner, B.; Reiterer, H.; Koch, M.: Software-ergonomische Evaluation. EVADIS II. Walter de Gruyter , Berlin 1992

Oppermann, R.; Simm, H.: Adaptability: User-Initiated Individualization, In: Oppermann, R. (ed.): Adaptive User Support – Ergonomic Design of Manually and Automatically Adaptable Software. Hillsdale, New Jersey 1994: Lawrence Erlbaum Ass.

Paul, H.: Exploratives Agieren, Peter Lang, Frankfurt/M 1994

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

Prakash, A.; Knister, M. J.: A framework for undoing action in collaborative systems, in: ACM-Transactions on Computer-Human Interaction, Vol. 1, No. 4, 1994, pp. 295-330

Prinz, W.; Mark, G.; Pankoke, U.: Designing groupware for Congruency in Use, in: Proceedings of CSCW'98, ACM-Press, New York, 1998, pp. 373 - 382

Rauterberg, M.: Ein Konzept zur Quantifizierung software-ergonomischer Richtlinien, PhD-Thesis, Deportment of Computer Science, University of Zürich, Zürich 1995

Ressel, M.; Nitsche-Ruland, D.; Greunzhäuser, R.: An integrating transformation-orineted approach to concurrency control and undo in group editors, in: Proceedings of CSCW '96, ACM-Press, New York 1996, pp. 288-297

Stiemerling, O.: Anpaßbarkeit in Groupware - ein regelbasierter Ansatz M.Sc. Thesis, Department of Computer Science III, University of Bonn 1996

Stiemerling, O.; Cremers, A. B.: Tailorable Component Architectures for CSCW-Systems, in: Proceedings of the 6th Euromicro Workshop on Parallel and Distributed Programming, Jan 21-24, 1998, Madrid, Spain, IEEE Press, pp. 302-308

Stiemerling, O.; Kahler, H. and Wulf, V.: How to Make Software Softer - Designing Tailorable Applications. In Proceedings of 2nd Conference on the Design of Interactive Systems, Amsterdam (NL), ACM Press 1997, pp. 365-376

Trigg, R. and Bødker, S. 1994: From Implementation to Design: Tailoring and the Emergence of Systematization in CSCW, in: Proceedings of CSCW´94, ACM-Press, New York, pp. 45 – 55

Won, M.: Komponentenbasierte Anpaßbarkeit von Groupware, M.Sc. Thesis, Department of Computer Science III, University of Bonn 1998

Wulf, V.: Konfliktmanagement bei Groupware, Vieweg, Braunschweig und Wiesbaden 1997a

Wulf, V.: Storing and Retrieving Documents in a Shared Workspace: Experiences from the Political Administration. In: Howard, S.; Hammond, J.; Lindgaard, G. (eds): *Human Computer Interaction: INTERACT 97*, Chapman & Hall 1997b, pp. 469-476

Wulf, V: „Let's see your Search-Tool!" - Collaborative use of Tailored Artifacts in Groupware, to appear in: Proceedings of GROUP '99, ACM-Press, New York 1999a, pp. 51 - 59

Wulf, V.: On Search for Tailoring Functions: Empirical Findings and Implications for Design, in: Proceedings of OZCHI ´99, November 28 - 30, Wagga Wagga, Australia, 1999b, pp. 105 - 111

Wulf, V.: „Why did that happen?" – Building Appropriate Mental Models on Groupware Functions, in: Bullinger, H.-J.; Ziegler, J.: Human-Computer Interaction: Communication, Cooperation and Application Design, Lawrence Erlbaum, Mahwah, 1999c, pp. 338 - 342

Wulf, V.: Design for Self-directed Learning – Evaluating the Concepts of Direct Activation and Explorative Execution, Habilitation-Theses, University of Hamburg, 2000

Wulf, V.; Rohde, M.: Towards an Integrated Organization and Technology Development; in: Proceedings of the Symposium on Designing Interactive Systems, 23. - 25.8.1995, Ann Arbor (Michigan), ACM-Press, New York 1995, pp. 55 – 64

Wulf, V.; Stiemerling, O.; Pfeifer, A.: Tailoring Groupware for Different Scopes of Validity. In: Behaviour & Information Technology, 1999, Vol. 18, No. 3, pp. 199 - 212

Yang, Y.: Current Approaches & new Guidelines for Undo-Support Design, in: Proceedings of INTERACT'90, North-Holland, Amsterdam, pp. 543 - 548