Supporting Awareness in Shared Workspaces Using Relevance-Dependent Event Notifications

Uwe Rauschenbach

Rostock University
Department of Computer Science
Computer Graphics Research Group
D-18051 Rostock
Germany

Email : urausche@informatik.uni-rostock.de
WWW : http://www.informatik.uni-rostock.de/~urausche

Abstract. A shared workspace is a groupware system which allows multiple users to access objects contained in it. The system needs to display notifications about the activities of co-workers to each user if they are relevant for his/her work. This paper suggests a concept to model events in a shared workspace. Based on that, a mechanism for specifying the relevance of event notifications is proposed and a filtering mechanism is introduced. Several presentation techniques are described which are capable of presenting event notifications at different relevance levels. Finally, a prototype implementing part of the model is presented, and some critical remarks regarding the extensibility of the system LinkWorks™ are made.

Keywords: CSCW, Groupware, User Interfaces, Multi User Systems, Awareness

1 Introduction

Nowadays, groupware systems like LinkWorks™ are commercially available and broadly used to support co-operation in geographically distributed working groups. One application class are virtual shared workspaces, which support the simultaneous or asynchronous access of multiple users to a set of shared objects. However, in most existing systems the user is not sufficiently informed about events happening in the workspace due to the actions of other users (e.g., changing a shared document). Event notifications offer the capability to overcome this lack of awareness, thus helping to create and maintain a shared context for the work process going on. On the other hand, the user can be easily overloaded by a flood of notifications.

This paper suggests a concept to model events in the context of a shared workspace. Based on that, a mechanism for specifying the relevance of event notifications is proposed and a filtering mechanism is introduced. Several presentation techniques are described which are capable of presenting event notifications at different relevance levels. Finally, some critical remarks regarding the extensibility of the system LinkWorks™ are made, and a prototypical implementation of some of the concepts is described.

2 Modelling Events and Notifications

Motivation. Groupware systems allow co-operation and communication. *Awareness* information can support the user in these activities by creating and maintaining a shared context. We define awareness to be the „use of implicitly existing information channels with the goal to capture past and present activities of co-operation partners in the current working context".

In the mode of *synchronous awareness*, information must be provided to the user about the current work of her/his colleagues, their availability for communication etc. Here, detailed information about parallel activities of other users has to be presented as far as it is relevant for the own work. Contrasting, the *asynchronous mode of awareness* (often called *catch-up mode*) supports the user who has left the system for a while in catching up quickly with the work of others. There, a *summary of past activities* has to be presented.

The presentation of event notifications is a means to support both modes of awareness. In order to cater for uncoupled awareness[1] and to filter the flood of notifications, a notion of context has to be used. As the relevance of a notification determines its presentation, the (potential) receiver of a notification must be able to control the relevance of existing notifications as well as of notifications about events in the future.

In existing research and commercial systems, the presentation of event notifications according to a user-definable relevance is not supported. DIVA [SOH94] or BSCW [APP96], for instance, present event notifications, but the presentation is the same for each user. LinkWorks™ [DEC94] shows notifications about changes in a very intrusive way using modal dialogues, thus interrupting the user's work. Although this mechanism can be turned on and off for individual objects as well as for a part of the folder hierarchy, it neither gives information about the object affected nor it allows follow-up actions.

Shared workspace, context. Using the desktop metaphor, we assume a hierarchically structured object system to store the shared objects and want to call the objects which form this hierarchy containers. As users often group their work objects by task in containers, we use a container as a work *context*. Furthermore, we define a shared workspace to be a container object containing *user objects* which represent the users who are allowed to enter the workspace and *work objects* which are the containers (folders) and documents the users work with. When we enrich such a shared workspace with functionality for providing awareness information (described in this paper) and communication facilities (planned), a sort of a simple collaborative virtual environment is formed.

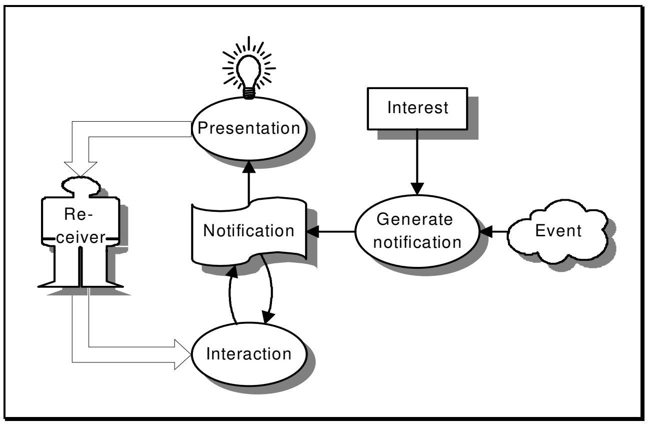


Figure 1: Relation between events, interest, and notifications

Events, Interest and Notifications. To provide a flexible and user-controllable notification mechanism, we must distinguish between *events* and *notifications*. Each user is free to register *interest* in a particular event in order to determine how a notification about this event is presented to her/him. Thus, our model consequently separates *events* from event *notifications*. While an event models all the information about one state change of the system necessary to provide awareness of that change and is stored once, multiple notifications about that event can be directed to multiple receivers to inform them about that change if it is relevant to them. Table 1 lists the fields of an event tuple.

Questions of the user	**Data to be captured**
WHAT happened?	event type
WHEN did it happen?	time stamp
WHERE did it happen?	affected object, parent object (context)
WHO did it do?	event initiator
WHICH further information exists?	optional parameters

Table 1: Questions of the user and corresponding event entry fields

[1] to be aware of events happening in other contexts than the one the user is currently working in

Events and notifications are linked by the *interest* of the potential receiver which can be specified using *interest tuples*. As interest is concerned with future events, an interest tuple contains *pattern* fields to be matched against the fields of each event on occurrence, and *relevance* fields which determine the relevance of the notification to be generated if a match is found. The relevance of a notification (see section 4) has two functions. First, it determines the *presentation* technique to use. Second, the relevance controls the *propagation* of the notification in the context hierarchy. Propagation of notifications means that notifications are forwarded to contexts in higher levels of the context hierarchy, starting from the context of event occurrence, until a context is reached where the notification is presented[2]. This mechanism enables to provide awareness of events raised somewhere down in the context hierarchy in higher-level contexts. While interest supports specifying the relevance of notifications about future events, *lifecycle operations* support the alteration of the relevance of existing ones and the retrieval of additional information. These operations include

- Lowering the importance of a notification (marking the notification as „read“)
- Snoozing the notification to get a reminder at a later time
- Confirming (deleting) the notification
- Getting details about the event reported
- Displaying the object affected by the event reported or performing follow-up actions on it (e.g., reading a mail whose arrival has been reported)
- Creating a communication channel to the initiator of the event reported

Lifecycle operations can be carried out either explicitly by the user or automatically by the system when certain criteria are met[3]. In order to prevent information overload, a *thinning concept* has been provided which removes all but the latest of multiple notifications about events which have the same type and affect the same object.

3 Presenting Notifications

Characteristics of the Presentation Techniques. As stated above, the presentation of a notification depends on its relevance. Consequently, different presentation techniques had to be designed. To derive design criteria, we developed an intuitive classification of presentation techniques. For our purposes, we distinguish between *properties* and *capabilities* of a presentation technique, where the former determine the latter as well as have to meet certain user's needs. As properties of a technique, we consider its *obtrusiveness*, its *context relation*, its *perceiveability* and its *degree of detail*. Depending on these properties, the capability of a technique to *attract attention* and its *interactivity* are considered to be of importance.

Based on this, we introduce the *urgency* of a presentation technique as a criterion to decide which technique to use for a notification based on its relevance. Rules are used to determine the mapping between both. The ideal presentation technique maintains the relation to the object affected by the event, has a high potential to attract attention, a rich set of interactive features and a low degree of obtrusiveness. Unfortunately, this ideal presentation technique does not exist. Instead, each presentation technique is a trade-off between its potential to attract attention and its interactivity on the one hand and its obtrusiveness on the other hand. High potential to attract attention comes always at the cost of high obtrusiveness, which is only accepted by the user for notifications with a high relevance.

Presentation Techniques for Notifications. We defined five steps of urgency and accordingly developed five presentation techniques, which can be seen as an extension of the conventional desktop metaphor. The presentation technique with the lowest urgency is *not to present* the notification. For most notifications, the technique with the next higher urgency is used, which we called *object-coupled presentation*. This technique presents some of the event attributes (e.g., time stamp or type) as graphical attributes of the icon representing the object which is affected by the event. For this purpose, a consistent extension of the graphical alphabet of the LinkWorks icons is proposed, which relies on altering the colour of certain parts of the object icon or of overlaid special symbols in order to represent the time stamp of the event (colour) and its type (icon part or overlay). *Context-coupled presentation* is the next

[2] the so-called notification context

[3] e.g., confirming the notification about an „document object changed“ event when the user opens this document for reading

level of urgency, which presents the event notification within the context of event initiations, i.e. as a button bar in the folder window which represents the context for the event. *Non-modal global presentation* displays this button bar in a global window, which is always visible and not dependent on the event context, thus allowing an easy, non-obtrusive overview over globally-relevant events. The technique having the highest urgency, *modal global presentation*, uses a modal dialogue box to present a detailed compilation of relevant events, offering rich interactivity and still maintaining the context relation. Colour plates showing screen shots of the presentation techniques can be found at the end of this paper.

4 A Prototypical Solution

Prototype Architecture. To prove the feasibility of our concepts, we implemented a prototypical shared workspace system under MS-Windows™ 3.1 using the API of the groupware system LinkWorks™. Our experiences showed, however, that the extensibility of LinkWorks™ is not sufficient to realise the concepts outlined above. We discovered the following serious shortcomings:

- Structured, dynamic attributes were needed to store lists of interest entries, events and notifications associated with an object. Such attributes are used by LinkWorks™ internally, but cannot be defined via the API.
- The usage and extension of the existing notification mechanism of LinkWorks™ is not possible via the API.
- LinkWorks™ methods concerned with the generation of user interface elements can not be extended or overwritten using the LinkWorks™ API.

Thus, we had to implement all user interface functionality for our proof-of-concept prototype from scratch using a user interface class library. An external database was used to store interest, events and notifications. LinkWorks™ served only as an object store, for meta data management and for user authentication.

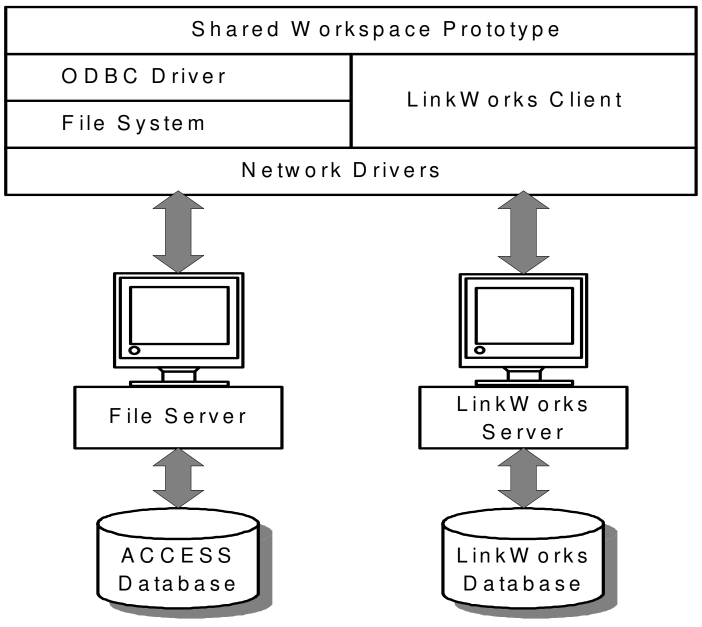


Figure 2: Architecture of the prototype

Data model. For supporting the tasks of collaborative office work, we selected a set of event classes which is shown in Table 2. When looking at events, we must distinguish between *atomic events* and *activities*. Where the former represent a single state change in the system (e.g., the creation of a new document) at a *point* in time, the latter (e.g., a session of a user in the system) are *processes* which have a *time duration* greater than zero and can generate a series of state changes. Thus, an activity can be modelled by a series of events - at least by the pair *start event – end event*.

Event/activity type	**Semantics**
CreateEvent	Create object in a container
DeleteEvent	Delete object in a container
MoveToEvent	Move object into a container
MoveFromEvent	Take object out from a container
ModifyEvent	Modify document object
RenameEvent	Rename object
LoggedInActivity	Presence of a user in the shared workspace
EditActivity	User is editing document object
ReadActivity	User is editing document object
SystemEvent	System message

Table 2: Event/activity classes an their semantics

For our implementation, we define an *event entry* to be a tuple $e := (t, y, p_y, u, o, o_y)$, which consists of the event time t, the event type y, the optional parameters p_y (depending on y), the event initiator u, the event object o and the parent object o_y[4].

An *interest pattern* must be able to describe future events which may be of interest to a particular user, and it must specify the initial relevance of the notification to be generated if such an event occurs. We specify an interest pattern as a tuple: $i{:=}(o, U_o, y, exp(p_y), u_o, w, b, c_m)$; being o the object which will be affected by the interesting event, y the type of the event, U_o the set of users who might initiate the event, $exp(p_y)$ a logical expression on the parameters of y, u_o the user specifying interest, w the importance and c_m the notification context of the notification to be generated (see below). The Boolean flag b indicates whether to apply the interest pattern only to the first matching event[5] or to all matching events.

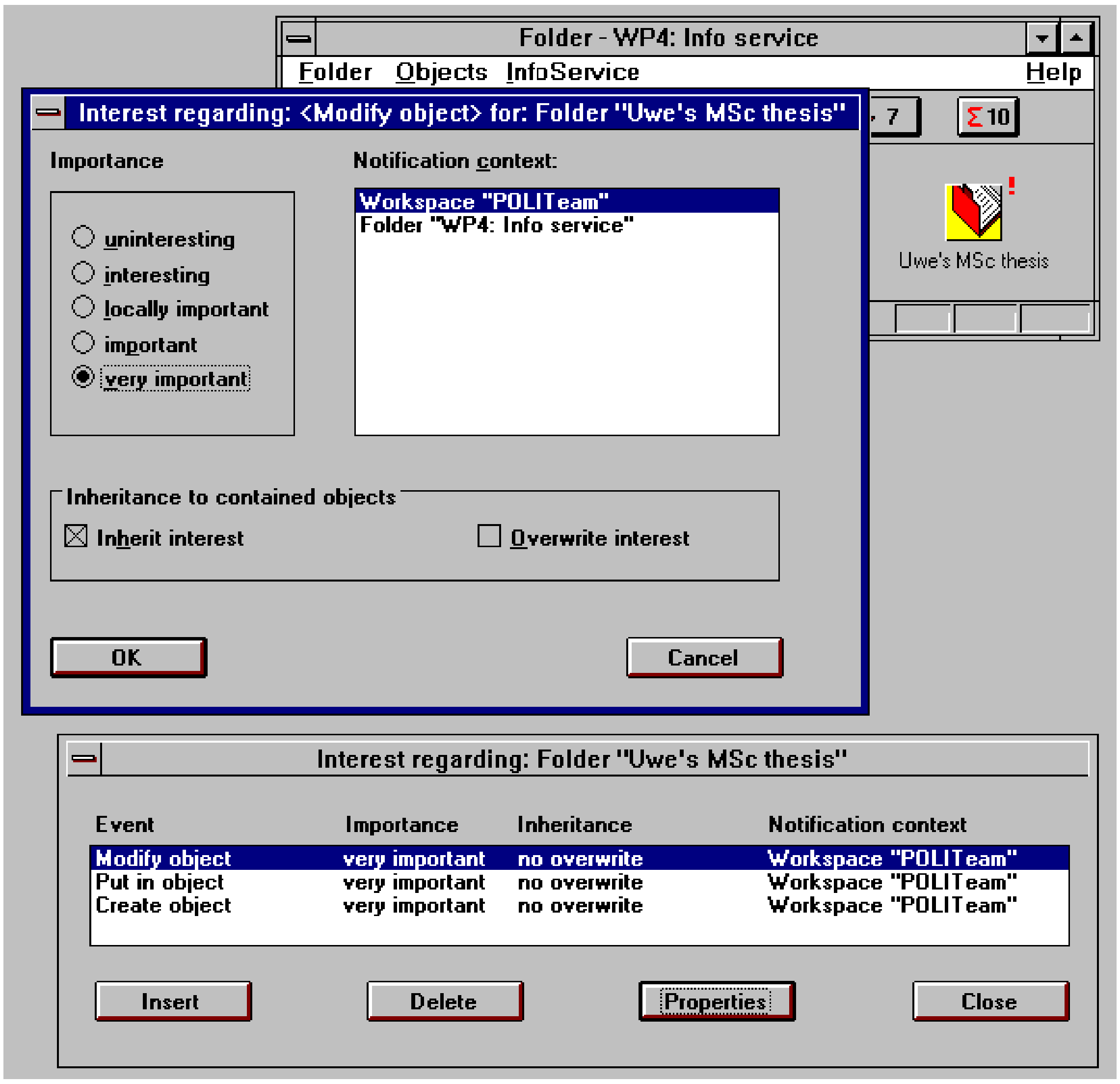
A notification is modelled as: $n{:=}(e, r, op, u)$ with the event e, the relevance r, the receiver u and the possible interactions op. Implicitly, the sender of the notification is the initiator of e. The relevance of a notification $r{:=}(w, c_m, s, t)$ consists of the importance w and the notification context c_m (which are initialised from the interest pattern) and the information status s and the snooze time t (which are needed for realising lifecycle operations). The relevance of a notification determines its presentation.

Specifying interest. Each user can specify interest in any event regarding any object or subtree of the container hierarchy. When interest is specified for a subtree of the container hierarchy (i.e. for a container which contains objects), the user can choose between overwriting existing interest entries attached to objects of the subtree (overwriting inheritance) or only applying the interest pattern to objects without individual user-defined interest (non-overwriting inheritance). This provides a mechanism for setting user-specific default interest patterns. Figure 3 shows an example of specifying interest using non-overwriting inheritance for a subtree of the container hierarchy.

When looking up interest after an event occurred, a three-step mechanism is used. First, a user defined interest pattern attached to the object affected by the event (object interest pattern) is considered. Second, the container hierarchy is traversed towards the root container in order to find user defined defaults. These either overwrite the user-defined interest (overwriting inheritance) or are only considered when no object interest pattern was found (non-overwriting inheritance). If this process does not find a resulting interest entry which can be applied, the default entry of the class of the object affected by the event is used.

[4] The parent object is only stored with event types which alter the relationship between parent and child objects, e.g., deleting an object or moving an object to a different folder

[5] For instance, when user A sends a mail message to user B, he/she might want to be notified that A read it. In this case, B is only interested in the first *read* event, not in subsequential ones regarding the same message.



Interest regarding: <Modify object> for: Folder "Uwe's MSc thesis"

Importance

- ○ uninteresting
- ○ interesting
- ○ locally important
- ○ important
- ● very important

Notification context:

Workspace "POLITeam"
Folder "WP4: Info service"

Inheritance to contained objects

☒ Inherit interest ☐ Overwrite interest

Interest regarding: Folder "Uwe's MSc thesis"

Event	Importance	Inheritance	Notification context
Modify object	very important	no overwrite	Workspace "POLITeam"
Put in object	very important	no overwrite	Workspace "POLITeam"
Create object	very important	no overwrite	Workspace "POLITeam"

Figure 3: Specifying interest

Generating and distributing notifications. When an event occurs, for each user a notification about that event is generated according to his/her user defined interest or a default interest. The relevance is initialised using the fields w and c_m from the interest tuple. Notifications are propagated upward in the container hierarchy in order to present an overview of events that happened lower hierarchy levels. As this can lead to high volumes of data and to information overload, thinning of the propagated notifications is necessary. For an overview it is considered sufficient to store only the most recent notification plus the total number of notifications in the according subtree per container object and event type. Detail can be provided to the user on demand.

Figure 4 shows the data flow of the generation and distribution of a notification about an event showing the event initiator (local user) and a remote notification receiver. In a groupware system with potentially long time delays it is crucial to separate between local and remote echo. We do this separation by replicating the mechanism for the propagation of notifications. When the local user initiates an event, a notification to the event initiator is generated and propagated locally. It is presented to the local user with low urgency (i.e., the colour of the affected object icon changes) using the normal presentation mechanism. As no network access is needed, the response time is reasonably short.

After that, the notifications to be stored in the database are generated. For doing so, the interest of each potential receiver (i.e. each user who has access to the workspace) is determined by the interest lookup process described above. According to the interest patterns found, notifications are generated, propagated and stored in the database. In order to distinguish between primary and propagated notifications, propagated notifications are flagged accordingly. The remote client accesses the notifications in the database either by a polling or a messaging mechanism, retrieves and presents them.

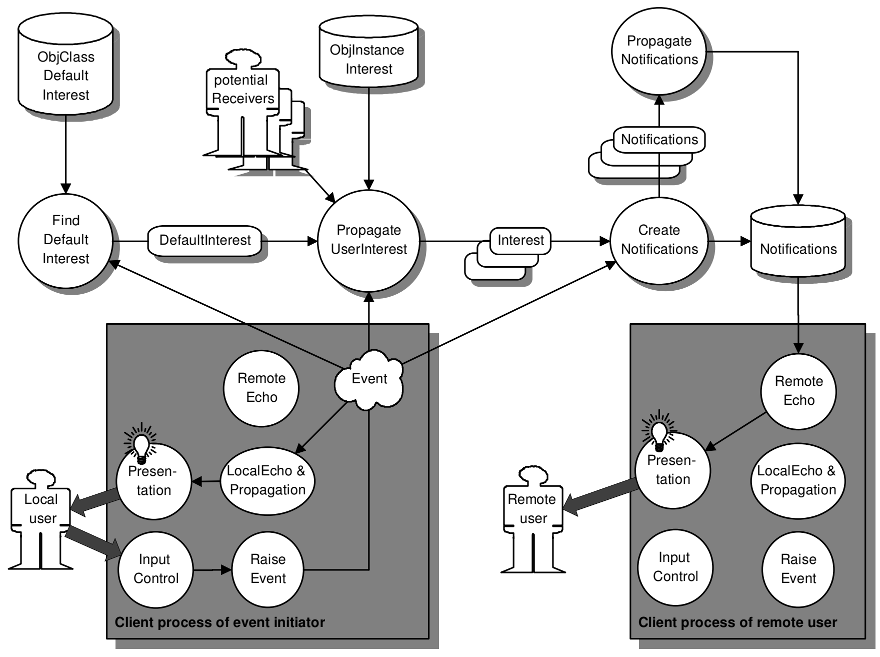


Figure 4: Generation and distribution of a notification

Lifecycle and presentation of notifications. The concept of graded urgency of presentation techniques has already been discussed in section 3. But how is a presentation technique selected given a notification with a certain relevance? A notification can be presented at all levels of urgency up to a maximum level according to the state it has reached in its lifecycle (see below). That means, if a notification is presented using a technique with high urgency, it is too presented at all urgency levels which are lower. We call the presentation technique with the highest urgency for a particular notification the *maximum presentation technique* for that notification. If a notification has been generated, its maximum presentation technique solely depends on its importance. In other words, a notification will never be presented with a more urgent technique than its importance suggests. Table 3 shows which maximum presentation technique is selected according to the importance of a notification.

Importance	Maximum presentation technique
unimportant	None
interesting	Object coupled
locally important	Context coupled
important	Non-modal global
very important	Modal global

Table 3: Presentation depending on the importance of a notification

Each notification has a lifecycle which can be represented as a state transition diagram. The states in the lifecycle are determined by the values of the fields *snooze time (t)* and *information status (s)* in the relevance tuple of a notification. These values can be changed interactively by the user (see, e.g., Colour plate 4) in order to control the urgency of the presentation of existing notification. The snooze time is used to postpone the display of a notification to a later time and thus realising a remembrance function. The information status stores whether or not the user has marked the notification as „seen“. Furthermore, the relevance tuple contains a field called *notification context* (c_m *)*. This field controls the modal global presentation – it is used to determine in which subtree of the context hierarchy (i.e., container) the notification is relevant. For instance, a notification about an event that happened somewhere down in the container hierarchy could be relevant in the root node of the hierarchy, i.e., the workspace itself, or in a folder which is contained in the workspace.

The above additional attributes of a notification are considered as follows:

1) A notification is only presented using modal global presentation if its notification context is open (i.e., the corresponding window is opened on the screen). Thus, the notification context can be seen as an information filter.
2) Notifications which are marked as „seen“ are presented object-coupled at maximum.
3) Snoozed notifications are presented are presented object-coupled at maximum from the time of postponing until the expiry of the snooze time interval.
4) The propagated copies of a notification are never presented globally. This does not override 2) and 3).

Notifications about events directed to the event initiator are presented object-coupled at maximum regardless of their importance. They serve as local echo of his/her actions.

Multi-threaded dialogue. According to [LUC94], the user interface of a groupware system should be asynchronous, that means, a long network access should only block the objects affected, but not the whole application. The LinkWorks™ API executes all calls asynchronous, thus both supporting and requiring an asynchronous user interface. Figure 5 shows the possible states of a user interface object and the transitions between these.

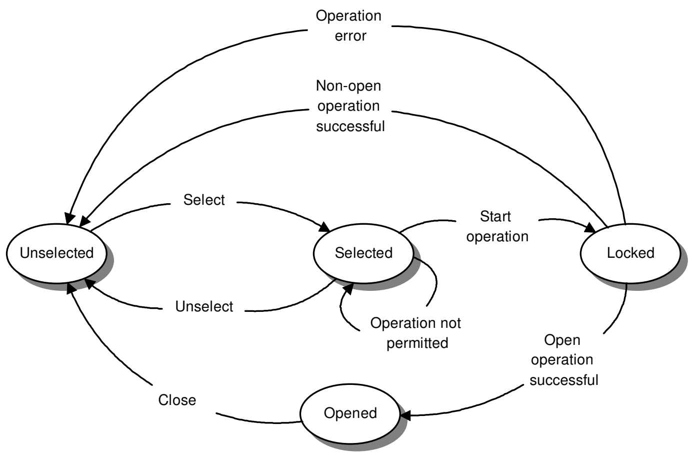


Figure 5: State transitions of objects in a multi-threaded user interface

When an operation is to be performed on an object, it has to be selected first (operation *Select*). After having selected all objects, the user can activate the operation he/she intends to apply to the objects. As LinkWorks executes the operation sequentially on the selected objects, they have to be locked in order to prevent the user from selecting more objects or deselecting objects while the operation is running. To distinguish between multiple groups of selected objects, the *Lock* operation attaches a *selection ticket* to each selected object which determines the group the selected objets belongs to. While the operation is now executed, the user is enabled to select other objects than the locked ones and execute operations on them. If the operation fails or generates an atomic event (e.g., renaming an object), the object state turns to *unselected* upon completion. If the operation is an activity, the state of the object is turned to *open* until the activity is terminated by a closing operation.

5 Conclusions

This paper investigated opportunities for the relevance-dependent presentation of notifications about events in a shared workspace. We found that such notifications offer the potential to enhance the efficiency of co-operative work in virtual shared workspaces, but have to be filtered according to some notion of relevance in order not to overload the user with information.

We developed a model which supports the generation and presentation of event notifications. The model consequently separates events fom notifications. Notifications can have different relevance which is determined by interest patterns either defined by the user or taken from a set of defaults. The receiver can interact with the notification in order to change its relevance or to get further information about the event reported. Notifications are filtered using contexts and are propagated from the context of the event occurrence to higher-level contexts. In the prototypical implementation, a context was considered a folder in a folder hierarchy.

The presentation of a notification depends on its relevance. Five presentation techniques with graded urgency have been proposed which are extensions of the conventional desktop metaphor. Here, the focus has been on the extension of the graphical alphabet of the LinkWorks (TM) icons.

Finally, we presented a prototypical implementation of our concept. Some problems of the realisation using LinkWorks™ have been identified. We discussed the mechanism used to specify interest, the process of notification generation and distribution, selection rules for a presentation technique according the relevance of the notification to be presented and a solution which supports a multi-threaded dialogue.

Future work could focus on the dynamic, semi-automatic generation of interest patterns and on the use of different media (e.g., sound) and metaphors (e.g., 3D environments) for the presentation. Another task could be the reimplementation of the prototype in order to gain independence from LinkWorks™, to include synchronous and asynchronous communication facilities between users and to support mobile users which connect to the system via wireless modems.

Acknowledgements

Parts of this work have been carried out in the POLITeam project at VW-GEDAS. The author wishes to thank VW-GEDAS for the support in the work presented. Special thanks to Heidrun Schumann from Rostock University for encouraging words and many valuable comments on my work, to Wolfgang Prinz, Ludwin Fuchs and Markus Sohlenkamp from GMD Bonn for fruitful discussions and last but not least to my former colleagues at VW-GEDAS.

References

[APP96] Appelt, W.; Busbach, U.: *„The BSCW System: A WWW based Application to Support Cooperation of Distributed Groups“*, to be presented at WET ICE 96: Collaborating on the Internet: The World-Wide Web and Beyond, Stanford University, June 19-21, 1996

[BER93] Berlage, T.; Genau, A.: *„A Framework for Shared Applications with a Replicated Architecture“*, in: Proceedings of ACM Symposium on User Interface Software and Technology, (1993)

[DEC95] Digital Equipment Corporation: LinkWorks™ Version 3.0.7 Manuals, 1995

[DOU92a] Dourish, P.; Bellotti, V.: *„Awareness and Coordination in Shared Workspaces“*, in: Proceedings of ACM CSCW '92 Conference on Computer-Supported Cooperative Work, (1992), pp. 107-114

[DOU92b] Dourish, P.; Bly, S.: *„Portholes: Supporting User Awareness in a Distributed Work Group“*, in: Proceedings of CHI '92, ACM, (1992), pp. 541-547

[FUC94] Fuchs, L; Prinz, W.: *„Supporting User Awareness with Local Event Mechanisms in the GroupDesk System“*, to be published, Source: GMD, Schloß Birlinghoven, 53754 St Augustin, email: prinz@gmd.de

[LUC94] Lucas, P.; Schneider, L.: *„Workscape: A Scriptable Document Management Environment“*, in: Conference Companion CHI '94, ACM, (1994), Seite 9-10

[NEW92] Newman-Wolfe, R. E. et al.: *„Implicit Locking in the Ensemble Concurrent Object-Oriented Graphics Editor“*, in: Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work, (1992), pp. 265-272

[RAU95] Rauschenbach, U: *„Relevanzabhängige Darstellung von Meldungen über Benutzeraktivitäten in einem gemeinsamen Arbeitsbereich“*, Diplomarbeit (MSc Thesis), Universität Rostock, Fachbereich Informatik, 1995

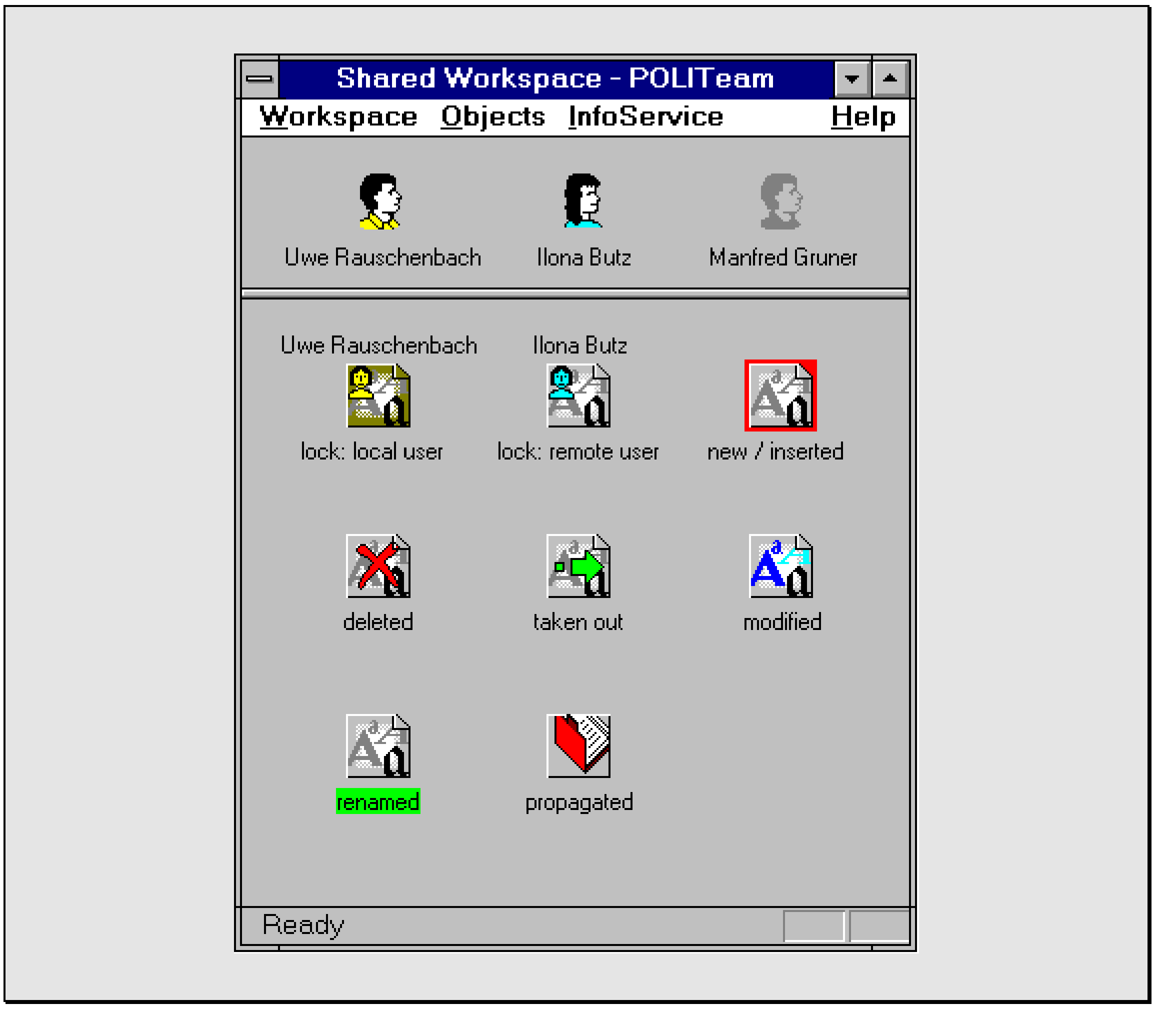
[SOH94] Sohlenkamp, M.; Chwelos, G.: *„Integrating Communication, Cooperation and Awareness: The DIVA Virtual Office Environment“*, in: Proceedings of CSCW '94, ACM, (1994), pp. 331-343

[SOH95] Sohlenkamp, M.: *„Awareness and State Change Visualization in Multi-User Environments“*, Draft, to appear in: Arbeitspapiere der GMD (Gesellschaft für Mathematik und Datenverarbeitung)

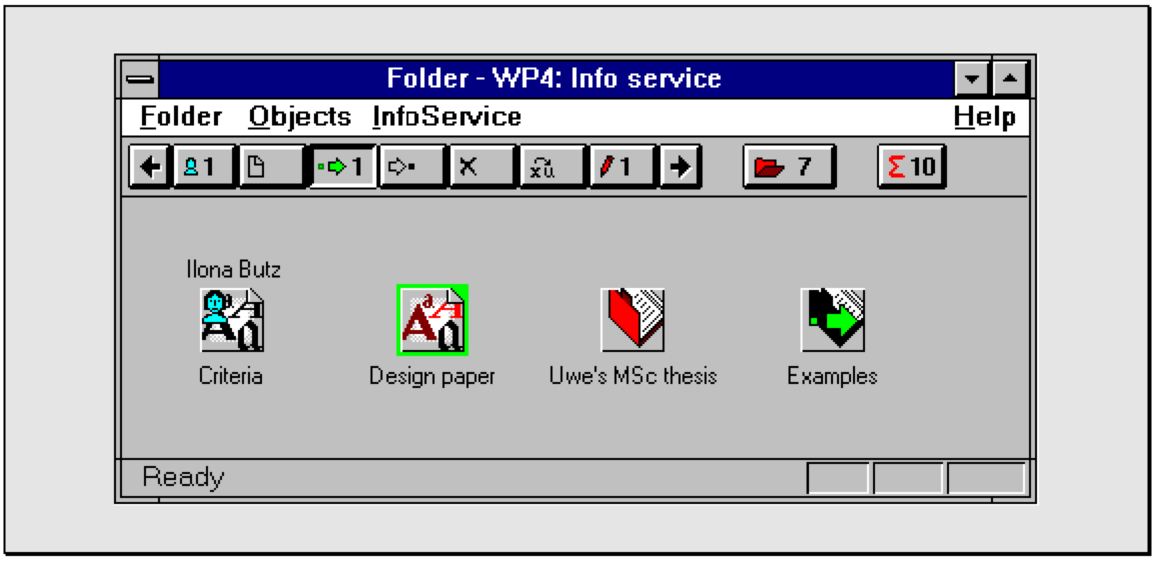
[TAN94] Tang, J. C.; Rua, M.: *„Montage: Providing Teleproximity for Distributed Groups“*, in: Proceedings of CHI '94, pp. 37-43

[TOL94] Tollmar, K. et al.: *„The Collaborative Desktop. An Environment for Computer Supported Cooperative Work“*, in: Conference Companion CHI '94, ACM, (1994), pp. 23-24

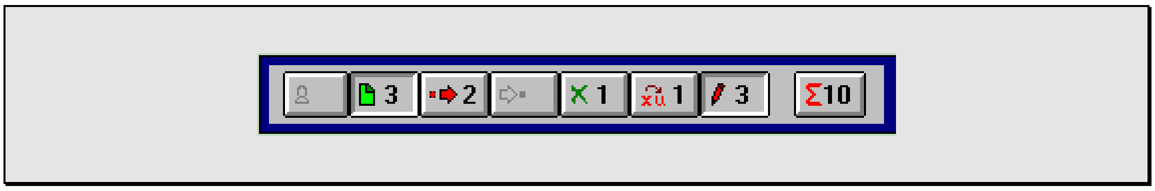
Colour Plates



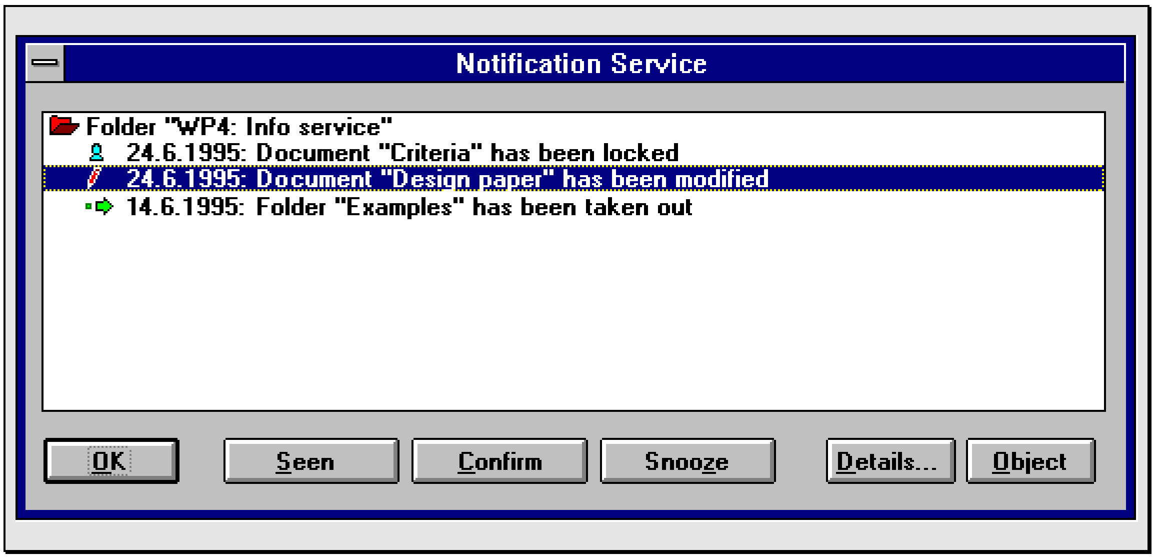
Colour Plate 1: Object-coupled presentation



Colour Plate 2: Context-coupled presentation



Colour Plate 3: Non-modal global presentation


Notification Service

Folder "WP4: Info service"
24.6.1995: Document "Criteria" has been locked
24.6.1995: Document "Design paper" has been modified
14.6.1995: Folder "Examples" has been taken out

Colour Plate 4: Modal global presentation