

**Universität Rostock**  
**Fachbereich Informatik**  
**Institut für Computergrafik**



**Relevanzabhängige Darstellung  
von Meldungen über Benutzeraktivitäten  
in einem gemeinsamen Arbeitsbereich**

Diplomarbeit

**vorgelegt von:** Uwe Rauschenbach  
geboren am 05. Dezember 1968 in Demmin

**Betreuer:** Prof. Dr. Heidrun Schumann, Universität Rostock  
Martin Klose, VW-GEDAS, Berlin

**Abgabedatum:** 15. September 1995

## **Kurzfassung**

Ein gemeinsamer Arbeitsbereich ermöglicht es mehreren Benutzern, auf die darin befindlichen Objekte zuzugreifen. Dabei muß das System jedem Benutzer die Aktivitäten der Kooperationspartner darstellen, soweit sie für dessen Arbeit relevant sind. Nach einem Überblick über Arbeiten zum Thema Groupware-Benutzerschnittstellen untersucht die vorliegende Arbeit Möglichkeiten der relevanzabhängigen Darstellung von Meldungen über Benutzeraktivitäten in einem gemeinsamen Arbeitsbereich. Ein Modell für Meldungen wird entworfen, und Techniken zur Präsentation von Meldungen mit abgestufter Dringlichkeit werden vorgeschlagen. Schließlich wird ein Prototyp vorgestellt, der einen Teil des Modells implementiert.

## **Abstract**

A shared workspace 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. After reviewing related work, this paper investigates opportunities for the relevance-controlled presentation of notifications about user activities in a shared workspace. A model for notifications is designed and techniques with graded urgency for the presentation of these notifications are proposed. Finally, a prototype implementing part of the model is presented.

## **Keywords**

CSCW, Groupware, User Interfaces, Multi User Systems, Shared Workspace, Awareness

# **CR-Klassifikation**

**H.2.4** Distributed Systems

**H.4.1** Office Automation

**H.5.2** User Interfaces

- Screen design
- Interaction styles

**H.5.3** Group and organization interfaces

- Asynchronous interaction

**K.8.3** Application packages

- Data communications

---

# Inhaltsverzeichnis

---

<b>1</b>	<b>Einleitung .....</b>	<b>7</b>
<b>2</b>	<b>Groupwaresysteme - ein Überblick.....</b>	<b>9</b>
2.1	Allgemeines .....	9
2.2	Klassifikation von Groupwaresystemen .....	9
2.2.1	Zeit .....	10
2.2.2	Ort .....	10
2.2.3	Kopplung .....	10
2.2.4	Kollaborationsunterstützung.....	11
<b>3</b>	<b>Anforderungen eines Groupwaresystems an die Benutzerschnittstelle.....</b>	<b>12</b>
3.1	Allgemeines .....	12
3.2	Vom Single-User-System zum Groupware-System.....	13
3.3	Awarenessinformation zur Unterstützung des Benutzers .....	14
<b>4</b>	<b>Beschreibung des Projektumfeldes und der Problemstellung.....</b>	<b>17</b>
4.1	Das Projekt POLITeam.....	17
4.2	Das System LinkWorks .....	18
4.3	Das Problem: Meldungen im gemeinsamen Arbeitsbereich.....	20
4.3.1	Zielstellung der vorliegenden Arbeit .....	20
4.3.2	Ist-Zustand .....	20
4.3.3	Sollkonzept .....	21
4.3.4	Weiteres Vorgehen.....	21
<b>5</b>	<b>Ein Modell für Meldungen.....</b>	<b>22</b>
5.1	Grundlagen.....	22
5.1.1	Motivation.....	22
5.1.2	Voraussetzungen .....	23
5.1.3	Überblick über das Modell.....	25
5.2	Ereignisse, Interesse und Meldungen.....	26
5.2.1	Ereignis .....	26
5.2.2	Interesse .....	30
5.2.3	Meldung .....	35
5.3	Erzeugen und Propagieren einer Meldung.....	45
5.3.1	Erzeugen und Verteilen einer Meldung .....	45
5.3.2	Propagieren und Neupropagieren von Meldungen .....	47
<b>6</b>	<b>Darstellung von Meldungen.....</b>	<b>52</b>
6.1	Allgemeines .....	52
6.2	Charakterisierung von Präsentationstechniken .....	52
6.2.1	Eigenschaften einer Präsentationstechnik.....	53

6.2.2	Fähigkeiten einer Präsentationstechnik.....	54
6.2.3	Die Dringlichkeit der Präsentation.....	55
6.3	Präsentationstechniken für Meldungen.....	56
6.3.1	Technik 1: Keine Präsentation.....	56
6.3.2	Technik 2: Objektgebundene Präsentation.....	56
6.3.3	Technik 3: Kontextgebundene Präsentation.....	65
6.3.4	Technik 4: Nicht-modale kontextübergreifende Präsentation.....	68
6.3.5	Technik 5: Modale kontextübergreifende Präsentation.....	69
6.4	Relevanzabhängige Auswahl der Präsentationstechnik.....	71
6.4.1	Prinzip.....	71
6.4.2	Der Auswahlmechanismus.....	72
<b>7</b>	<b>Vorstellung eines Prototypen.....</b>	<b>73</b>
7.1	Allgemeines.....	73
7.2	Ziele der Erstellung des Prototypen.....	73
7.3	Machbarkeitsanalyse auf der Basis von LinkWorks.....	74
7.3.1	Allgemeines.....	74
7.3.2	Anforderungen der geplanten Architektur.....	74
7.3.3	Aufgetretene Probleme.....	75
7.4	Beschreibung der realisierten Lösung.....	76
7.4.1	Systemarchitektur.....	76
7.4.2	Einbindung in die Klassenstruktur von LinkWorks.....	77
7.4.3	Programm-Architektur.....	78
7.4.4	Datenbankeinbindung.....	81
7.4.5	Asynchrone Ausführung von Aktionen auf LinkWorks-Objekten.....	84
7.4.6	Trennung von lokalem und entferntem Echo.....	86
7.4.7	Spezifizieren von Interesse.....	88
7.4.8	Erzeugen von Meldungen.....	89
7.4.9	Konsumieren von Meldungen.....	92
7.4.10	Präsentationstechnik „Farbcodierung“.....	95
7.4.11	Präsentationstechnik „Meldebox“.....	97
7.5	Wertung.....	98
<b>8</b>	<b>Schlußbetrachtungen.....</b>	<b>100</b>
	<b>Literaturverzeichnis.....</b>	<b>102</b>
	<b>Abbildungsverzeichnis.....</b>	<b>107</b>
	<b>Tabellenverzeichnis.....</b>	<b>109</b>
	<b>Anlage A: Glossar.....</b>	<b>110</b>
	<b>Anlage B: Funktionsbeschreibung des Prototypen.....</b>	<b>115</b>
	<b>Anlage C: Szenario.....</b>	<b>129</b>
	<b>Anlage D: Farbtafeln.....</b>	<b>137</b>
	<b>Selbständigkeitserklärung.....</b>	<b>145</b>
	<b>Thesen.....</b>	<b>146</b>

---

# Abkürzungsverzeichnis

---

CMC	<u>C</u> omputer- <u>m</u> ediated <u>c</u> ommunication
CSCW	<u>C</u> omputer- <u>s</u> upported <u>c</u> o-operative <u>w</u> ork
WYSIWIS	<u>W</u> hat <u>y</u> ou <u>s</u> ee <u>i</u> s <u>w</u> hat <u>I</u> see
ODBC	<u>O</u> pen <u>D</u> atab <u>a</u> se <u>C</u> onnectivity
MFC	<u>M</u> icrosoft <u>F</u> oundation <u>C</u> lasses
SDI	<u>S</u> ingle <u>D</u> ocument <u>I</u> nterface
MDI	<u>M</u> ultiple <u>D</u> ocument <u>I</u> nterface

---

# 1 Einleitung

---

In den letzten Jahren ermöglichten Fortschritte in der Software-, Hardware- und Netztechnologie die Schaffung immer komplexerer Computersysteme. Von der Unterstützung einzelner Benutzer vollzog sich im Zuge der fortschreitenden Vernetzung der Schritt zu Multiuser-Systemen. Dadurch wurde die Grundlage für rechnerunterstützte kooperative Prozesse geschaffen. Der multidisziplinäre Forschungszweig CSCW<sup>1</sup> und die Softwarekategorie Groupware entstanden auf dieser Grundlage Mitte der achtziger Jahre. Als Geburtsstunde gilt die erste CSCW-Konferenz 1986 in Austin, USA<sup>2</sup>.

Heute sind erste Groupwaresysteme bereits kommerziell verfügbar und werden u.a. eingesetzt, um kooperative Arbeit in räumlich verteilten Gruppen zu unterstützen. Eine Möglichkeit dazu sind virtuelle gemeinsame Arbeitsbereiche, die es mehreren Benutzern ermöglichen, auf darin befindliche Objekte zuzugreifen. Hierbei tritt in existierenden Systemen oft das Problem auf, daß ein Benutzer über für ihn relevante Aktivitäten von Kooperationspartnern nur unzureichend informiert wird. Eine häufig gestellte Frage ist zum Beispiel: „Welche Objekte wurden geändert, seit ich sie das letzte Mal gelesen habe?“ Entsprechende Meldungen können dieses Informationsbedürfnis befriedigen, andererseits kann deren Flut den Benutzer in seiner Arbeit stören.

Die vorliegende Arbeit versucht, eine Lösung für das oben skizzierte Problem aufzuzeigen. Nach einem allgemeinen Überblick über das Themenfeld Groupware-Benutzerschnittstellen wird das Problem eingegrenzt und ein Lösungskonzept vorgestellt. Ausgehend von Ereignissen, die für einen Benutzer relevant sein könnten, wird ein Modell zum Spezifizieren von Relevanz und darauf aufbauend ein Modell für Ereignismeldungen entwickelt. Es werden Präsentationstechniken für diese Meldungen und ein Mechanismus zur Auswahl einer Präsentationstechnik anhand der Relevanz einer Ereignismeldung vorgeschlagen. Schließlich wird eine erste prototypische Implementierung der entwickelten Konzepte präsentiert.

---

<sup>1</sup> CSCW - Computer-supported co-operative work

<sup>2</sup> Vgl. [PET92]

Als Vorbedingung für das entwickelte Konzept wird angenommen, daß die Benutzer in einem gemeinsamen Arbeitsbereich in einer kooperativen Atmosphäre und nicht mit der Absicht der Erlangung einer Vorteilsposition gegenüber anderen Benutzern arbeiten.

Die Arbeit ist Teil des Groupware-Projektes POLITeam im BMFT-Verbundprojekt POLIKOM.

---

## 2 Groupwaresysteme - ein Überblick

---

### 2.1 Allgemeines

Groupwaresysteme sind Softwaresysteme, die kooperative Arbeit unterstützen. Petrovic<sup>1</sup> unterscheidet die folgenden Funktionsbereiche:

- Electronic Mail
- Gruppen-Terminkalender
- Co-Autorensysteme
- Elektronische Meeting-Systeme
- Gruppenwissensbasen
- Intelligente Agenten.

Der in dieser Arbeit näher betrachtete gemeinsame Arbeitsbereich kann in den Funktionsbereich Co-Autorensysteme eingeordnet werden. Im folgenden sollen Groupwaresysteme klassifiziert werden.

### 2.2 Klassifikation von Groupwaresystemen

In der aktuellen wissenschaftlichen Literatur findet man die folgenden vier Dimensionen zur Klassifizierung von Groupware. Die ersten drei Dimensionen können gleichzeitig zur Klassifizierung von kooperativer Arbeit genutzt werden.

- Synchron vs. asynchron (*Zeit*)
- Lokal vs. entfernt (*Ort*)
- Gekoppelt vs. individuell (*Kopplung*)
- Collaboration aware vs. collaboration transparent<sup>2</sup> (*Kollaborationsunterstützung*)

---

<sup>1</sup> Vgl. [PET92]

<sup>2</sup> Vgl. [DEW91]

### 2.2.1 Zeit

Von *synchroner* Arbeit spricht man, wenn die Kooperationspartner gleichzeitig (parallel) mit dem System arbeiten. Beispiele sind das UNIX-Programm *talk* oder das Hypermedia-Autorensystem *CoMediA*<sup>1</sup>. *Asynchrone* Arbeit heißt, daß das System von den Kooperationspartnern zeitversetzt benutzt wird. Das bekannteste Beispiel hierfür dürfte *Email* sein.

### 2.2.2 Ort

*Lokal* heißt, daß sich die Kooperationspartner am selben Ort befinden, *entfernt*, daß sie sich an unterschiedlichen Orten befinden. Zusätzlich soll der Begriff *virtuell lokal* für Systeme benutzt werden, die (mehr oder weniger perfekt) die Illusion (*Teleproximity*) schaffen, daß sich alle Kooperationspartner im selben virtuellen Raum befinden. Ein Beispiel für ein lokales System ist das Meeting Support System *CoLab*<sup>2</sup>, ein entferntes System ist beispielsweise *Email*, und ein virtuell lokales System ist *Portholes*<sup>3</sup>.

### 2.2.3 Kopplung

Kopplung macht nur Sinn bei synchronen, (virtuell) lokalen Systemen. *Eng gekoppelte* Arbeit stellt allen Benutzern den gleichen Bildschirmausschnitt zur Verfügung, jeder sieht dasselbe. Diese Technik wird auch als *WYSIWIS*<sup>4</sup> bezeichnet. *Lose gekoppelte Arbeit* ermöglicht es, daß unterschiedliche Benutzer gleichzeitig unterschiedliche Teile desselben Objektes bearbeiten. *Individuelle (entkoppelte)* Arbeit verbirgt die Aktivitäten anderer Kooperationspartner vor dem jeweiligen Benutzer. Meist sichert ein Sperrmechanismus die Integrität des bearbeiteten Objektes.

Undo/Redo-orientierte Systeme wie *GINA*<sup>5</sup> unterstützen auch asynchrone, entkoppelte Arbeit. Die asynchron erstellten Versionen des bearbeiteten Dokuments werden unter Nutzung des Undo/Redo-Mechanismus gemischt. Das Mischen ist jedoch eine synchrone, eng gekoppelte Aktivität, da vom System nicht lösbare Konflikte von den Kooperationspartnern in der Diskussion gelöst werden müssen.

---

<sup>1</sup> Vgl. [SAN92]

<sup>2</sup> Vgl. [STE88]

<sup>3</sup> Vgl. [DOU92b]

<sup>4</sup> WYSIWIS - What you see is what I see

<sup>5</sup> Vgl. [BER93]

## 2.2.4 Kollaborationsunterstützung

Groupwaresysteme können entweder *collaboration aware* oder *collaboration transparent* arbeiten.

Ein System, das *collaboration aware* arbeitet, ermöglicht die zeitgleiche Bearbeitung eines Objektes durch mehrere Benutzer. Das kann z.B. durch Nutzung eines History-Konzeptes mit selektiven Undo/Redo-Mechanismen<sup>1</sup> erreicht werden. Weiterhin ist es möglich, wahlweise in öffentlichen oder privaten Arbeitsbereichen zu arbeiten und den Kopplungsgrad der gemeinsamen Arbeit variabel zu konfigurieren. Beispiele für diese Systemklasse sind GINA und GroupDesign<sup>2</sup>.

Ein System, das *collaboration transparent* arbeitet, ermöglicht keine gleichzeitige Bearbeitung eines Objektes durch mehrere Benutzer. Durch exklusives Sperren des bearbeiteten Objektes wird zu jedem Zeitpunkt nur einem Benutzer Schreibzugriff gewährt, Modifikationen müssen immer sequentiell erfolgen. Als Kopplungsgrad kommen daher nur individuelle (vollständig entkoppelte) oder streng gekoppelte Arbeit in Frage. Während bei ersterer verschiedene Objekte von mehreren Benutzern parallel bearbeitet werden können, können bei letzterer alle Benutzer den gleichen Bildschirmausschnitt sehen (*WYSIWIS*), jedoch hat zu jedem Zeitpunkt immer nur ein Benutzer Schreibrecht. Das im Projekt POLITeam verwendete System LinkWorks<sup>TM</sup> gehört zur dieser Klasse.

---

<sup>1</sup> Vgl. [BER93], [BEA92] und [PRA92]

<sup>2</sup> Vgl. [BEA92]

---

## 3 Anforderungen eines Groupwaresystems an die Benutzerschnittstelle

---

### 3.1 Allgemeines

Systeme zur Unterstützung kooperativer Arbeit (Groupware) sind sehr komplex und weisen viele Zusammenhänge auf, die sich aus den Aktionen mehrerer Benutzer ergeben. Diese Aktionen ändern den Zustand des Systems bzw. der Objekte im System und können zeitgleich (synchron) oder zeitversetzt (asynchron) ablaufen. Information über diese Zustandsänderungen kann wesentlich zum Entstehen und Erhalten eines gemeinsamen Kontextes für die kooperative Arbeit beitragen. Sie sollte daher an der Benutzerschnittstelle dargestellt werden. Heutige Systeme (z.B. LinkWorks) leisten das nicht oder nur unzureichend<sup>1</sup>.

Eine weitere Herausforderung bei der Gestaltung einer Benutzerschnittstelle für ein Groupwaresystem ergibt sich aus der Verteilung von Datenhaltung und Datenzugriff in heterogenen Netzwerken mit u.U. langen Antwortzeiten. Um eine zügige Arbeit mit dem System zu ermöglichen, ist eine asynchrone Benutzerschnittstelle<sup>2</sup> mit folgender Charakteristik notwendig:

- wenn für einzelne Objekte gerade ein Netzzugriff erfolgt, sind Interaktionen mit anderen Objekten weiterhin möglich
- konsequente Trennung von lokalem (sofortigem) und entferntem (ggf. verzögertem) Echo<sup>3</sup>

Aus der Forderung zur Darstellung von Information über die Aktivitäten anderer Benutzer entstand in den letzten Jahren der Forschungszweig Awareness innerhalb der CSCW-Forschung. Dabei geht es um die Aspekte der Erzeugung, Verteilung und

---

<sup>1</sup> Siehe Abschnitt 4.3.2.

<sup>2</sup> Vgl. auch [LUC94]

<sup>3</sup> Vgl. [BEA92]

Darstellung o.g. Information sowie um Zugriffsrechte<sup>1</sup> auf selbige. In Abschnitt 3.3 wird auf den Awareness-Aspekt von Groupwaresystemen genauer eingegangen.

## 3.2 Vom Single-User-System zum Groupware-System

In einem Single-User-System gehen alle Änderungen an den Objekten des Systems von einem Benutzer aus, der also immer den aktuellen Zustand kennen sollte. Konflikte in Zugriffsrechten gibt es nicht. Die konventionelle Desktop-Metapher unterstützt diese Arbeitsweise durch Abbilden des persönlichen Schreibtisches auf dem Rechner.

In einem Multiuser-System können, gesteuert von Zugriffsrechten und einem Locking-Mechanismus, mehrere Benutzer zeitversetzt Änderungen an denselben Objekten im System vornehmen. Daraus ergibt sich für den einzelnen Benutzer ein Informationsbedarf über die Aktionen anderer Benutzer (vgl. Tabelle 3.1).

System-kategorie	Single User System	Multi User System	Groupware System
Merkmale	<ul style="list-style-type: none"> <li>• Alle Änderungen gehen von einem Benutzer aus.</li> </ul>	<ul style="list-style-type: none"> <li>• Änderungen an Objekten zeitversetzt durch mehrere Benutzer,</li> <li>• Zugriffsrechte,</li> <li>• Locking.</li> </ul>	<ul style="list-style-type: none"> <li>• Enthält über Multi User Systeme hinausgehende Mechanismen zur Kooperationsunterstützung.</li> <li>• Ermöglicht Kommunikation<sup>2</sup> und ggf. synchrone gemeinsame Bearbeitung von Objekten.</li> <li>• Stellt Awarenessinformation dar.</li> </ul>
Beispiele für den Informationsbedarf des Benutzers		<ul style="list-style-type: none"> <li>⇒ Welche Zugriffsrechte haben ich und andere?</li> <li>⇒ Welche Dokumente wurden verändert, seit ich sie das letzte Mal gelesen habe?</li> <li>⇒ Wer hat eine bestimmte Änderung durchgeführt?</li> <li>⇒ Haben meine Kooperationspartner ausreichende Zugriffsrechte auf Objekte, die ich erstellt habe?</li> </ul>	<ul style="list-style-type: none"> <li>⇒ Sind meine Kooperationspartner für eine Kommunikation verfügbar?</li> <li>⇒ Haben sie Nachrichten hinterlassen?</li> </ul>

**Tabelle 3.1:** Systemkategorien, ihre Merkmale und der Informationsbedarf eines Benutzers

<sup>1</sup> Vgl. hierzu [ROH95]. Zugriffsrechte auf Awarenessinformation sind nicht Gegenstand dieser Arbeit.

<sup>2</sup> CMC - Computer-mediated communication

Systeme, die die kooperative Arbeit unterstützen (Groupware), ermöglichen darüber hinaus die Kommunikation und die synchrone Bearbeitung eines Objektes durch mehrere Benutzer. In Zukunft werden Groupwaresysteme auch immer stärker Awarenessinformation darstellen und so dem sich ergebenden Informationsbedarf der Benutzer über die Aktivitäten von Kooperationspartnern Rechnung tragen.

### 3.3 Awarenessinformation zur Unterstützung des Benutzers

Groupwaresysteme ermöglichen Kooperation und Kommunikation. Awarenessinformation unterstützt den Benutzer bei diesen Aktivitäten, schafft und erhält den Kontext für die gemeinsame Arbeit. Die Begriffe werden im Rahmen vorliegender Arbeit wie folgt definiert:

**Kommunikation:** Expliziter Aufbau von Kanälen zum Nachrichtenaustausch zwischen mindestens zwei Menschen.

**Kooperation:** Gezieltes Ausführen von Aktionen auf gemeinsam genutzten Objekten zum Erreichen eines Zieles.

**Awareness:** Nutzung implizit vorhandener Informationskanäle mit dem Ziel der Erfassung vergangener und gegenwärtiger Aktivitäten anderer Kooperationspartner im aktuellen Arbeitsumfeld.

Der Begriff *Awareness* kommt aus dem Englischen (*to be aware of sth.* - sich einer Sache bewußt sein) und bezeichnet das bewußte Erfassen der Aktivitäten anderer.

Awarenessinformation kann, wenn sie sinnvoll eingesetzt wird, die Effektivität kooperativen Arbeitens über große Entfernungen in virtuellen gemeinsamen Arbeitsbereichen erhöhen. Tabelle 3.2 gibt einen Überblick über typische Aktivitäten in einem gemeinsamen Arbeitsbereich. In Tabelle 3.3 werden die Benutzeraktivitäten dargestellt, die dem Gebiet der Awareness zugeordnet werden können.

	Synchron	Asynchron
Kommunikation	Kommunizieren in Echtzeit	Hinterlassen von Notizen für andere
Kooperation	Gleichzeitige Bearbeitung eines Objektes	Zeitversetzte Bearbeitung eines Objektes
Awareness	Was tun andere zur Zeit?	Was haben andere während meiner Abwesenheit getan?

**Tabelle 3.2:** Aktivitäten und Informationsbedarf eines Benutzers in einem gemeinsamen Arbeitsbereich<sup>1</sup>

<sup>1</sup> Vgl. [SOH94]

Synchronous Awareness	Asynchronous Awareness
<ul style="list-style-type: none"> <li>• Grob erfassen, was Kooperationspartner gerade tun</li> <li>• Feststellen, wer für eine Kommunikation verfügbar ist</li> <li>• Eigene Verfügbarkeit festlegen</li> <li>• Festlegen, welche Information über eigene Aktivitäten andere sehen dürfen</li> <li>• Feststellen, ob gemeinsam genutzte Objekte gerade von anderen bearbeitet werden, neu erzeugt, gelöscht oder in einen anderen Bereich verschoben wurden</li> </ul>	<ul style="list-style-type: none"> <li>• Feststellen, ob / wann / von wem gemeinsam genutzte Objekte geändert wurden</li> <li>• Feststellen, ob andere Nachrichten hinterlassen haben</li> </ul>

**Tabelle 3.3:** Benutzeraktivitäten, die dem Gebiet der Awareness zugeordnet werden können <sup>1</sup>

Nach Fuchs und Prinz kann Awarenessinformation wie folgt klassifiziert werden:

	Synchron	Asynchron
Gekoppelt	Was passiert zur Zeit im aktuellen Arbeitsbereich?	Was hat sich im aktuellen Arbeitsbereich seit dem letzten Zugriff verändert?
Ungekoppelt	Welche für mich relevanten Ereignisse geschehen zur Zeit in anderen Bereichen des Systems?	Welche für mich relevanten Ereignisse geschahen während meiner Abwesenheit in anderen Bereichen des Systems?

**Tabelle 3.4:** Klassifizierung von Awarenessinformation <sup>2</sup>

Im Mode *synchronous awareness* muß dem Benutzer an der Benutzeroberfläche dargestellt werden, was seine Kollegen momentan tun, welche Objekte sie manipulieren, ob sie für eine Kommunikation erreichbar sind usw. Es kommt hier darauf an, *über parallele Tätigkeiten anderer Benutzer ständig detailliert informiert zu werden*, soweit sie für die eigene Arbeit relevant sind.

Im Gegensatz dazu steht der Mode *asynchronous awareness*<sup>3</sup>. Hierbei will ein Benutzer, der das System für eine Weile verlassen hat, den aktuellen Stand der Arbeiten und die Änderungen seit seiner letzten Sitzung möglichst schnell erfassen. Er ist also an einer *Zusammenfassung vergangener Tätigkeiten* interessiert.

Awarenessinformation kann zur Überwachung mißbraucht werden. Es wird daher ein Konzept benötigt, das dem Benutzer ermöglicht, den Zugriff auf Information über seine Aktivitäten zu steuern, um diese Gefahr auszuschließen. Dieses Konzept kann nicht Gegenstand der vorliegenden Arbeit sein. Interessenten seien auf [ROH95] (Transparenz und Aushandelbarkeit) oder [BOR91] (Symmetry and Control) verwiesen.

<sup>1</sup> In Anlehnung an [SOH94]

<sup>2</sup> Vgl. [FUC94]

<sup>3</sup> Dieser Mode wird häufig auch als *Catchup* (Wiedereinstieg) bezeichnet.

Um Awarenessinformation in die Benutzerschnittstelle zu integrieren, ist eine Erweiterung der konventionellen „Desktop-Metapher“ um Techniken zur Präsentation von Awarenessinformation notwendig. Diese Erweiterung wird im Abschnitt 5 beschrieben.

---

## 4 Beschreibung des Projektumfeldes und der Problemstellung

---

### 4.1 Das Projekt POLITeam

Das Projekt POLITeam ist im BMFT-Verbundprojekt POLIKOM angesiedelt, welches vor dem Hintergrund des Regierungsumzugs von Bonn nach Berlin die Entwicklung von Systemen fördert, die die Zusammenarbeit von Menschen in großen, geographisch verteilten Organisationen unterstützen. Das System POLITeam wird gemeinsam von der VW-GEDAS Berlin, der GMD Bonn und der Universität Bonn entwickelt und beim Bundesjustizministerium, dem Justizministerium des Landes Mecklenburg/Vorpommern mit Zweigstellen in Bonn, Brüssel und Tallin sowie der AUDI AG als Pilotanwender getestet.

Im Projektantrag wird folgendes Projektziel gesteckt: „Als Projektergebnis wird ein einsatzreifes System zur Gruppenkoordination, Vorgangssteuerung und Archivierung entstehen, das sich durch eine neuartige Konzeption und Funktionszusammenstellung auszeichnet. Das System zielt gleichermaßen auf die öffentliche Verwaltung, vertreten durch Ministerien und nachgeordnete Behörden, und auf Anwendungen im industriellen Bereich ab.“

Das System POLITeam soll folgende Teilfunktionalität aufweisen:

- Vorgangssteuerung  
=> ermöglicht die Weiterleitung von Sammlungen von Dokumenten (Vorgängen) auf einem vorbestimmten Weg innerhalb der Organisation.
- Gemeinsame Arbeitsbereiche  
=> stellen einer Gruppe von Benutzern Werkzeuge, Dokumente und Strukturierungsmittel bereit, die diese gemeinsam nutzen können.
- Benachrichtigungs- und Informationsdienst  
=> informiert Benutzer über relevante Ereignisse im System, die Aktivitäten anderer Benutzer und über die Historie von Dokumenten.

- Registratur und Archivierung  
=> bereitet Dokumente für die Recherche nach gewissen Kriterien vor und archiviert sie.
- Organisationsmanagement  
=> beschreibt die Struktur der Organisation.
- Technisches Management  
=> unterstützt die Administration des Systems.

Als Groupware-Basis für das System POLITeam kommt das Groupwaresystem LinkWorks™ der Firma Digital Equipment Corporation zum Einsatz.

## 4.2 Das System LinkWorks

LinkWorks ist ein verteiltes, plattformübergreifendes System. Unterstützt werden Mail, Workflows (Objektläufe) und gemeinsamer (jedoch zeitversetzter) Zugriff auf Objekte. Objektläufe leiten Objekte auf einem vorbestimmten Weg von Benutzer zu Benutzer weiter und stellen eine Erweiterung der Mailfunktionalität dar.

Die Voraussetzung für den gemeinsamen Zugriff auf ein Objekt ist, daß jeder zugriffsberechtigte Benutzer einen Verweis auf dieses Objekt erhält. Die Semantik dieser Verweise ist mit Hardlinks im Betriebssystem UNIX vergleichbar. Auf einen Verweis kann wie auf das Originalobjekt zugegriffen werden. Jede Änderung der Attribute eines Objektes über einen Verweis wirkt sich auf alle anderen Verweise auf dieses Objekt aus. Nach dem Erstellen eines Verweises auf ein Objekt sind Originalobjekt und Verweisobjekt nicht mehr zu unterscheiden. Das Verweisobjekt kann jetzt einem anderen Benutzer zur gemeinsamen Nutzung geschickt werden. Es kann diesem danach nicht mehr entzogen werden, er muß es explizit löschen. Beim Löschen eines Verweises wird das Originalobjekt erst mit dem Löschen des letzten Verweises gelöscht. Der Objektname eines Verweisobjektes wird an der Benutzeroberfläche hellblau hinterlegt dargestellt.

Das LinkWorks-Userinterface bildet eine Büroumgebung nach und basiert auf der Desktop-Metapher. Alle Arbeitsmittel sind Objekte. Grob können die abstrakten Klassen *Container*-, *Werkzeug*- und *Dokumentobjekt* unterschieden werden, von denen konkrete Klassen abstammen, wie zum Beispiel *Ordner*, *Reißwolf* oder *Textdokument*. Ein LinkWorks-Containerobjekt kann weitere Objekte enthalten. Es gibt verschiedene Containerobjektklassen, wie *Ordner*, *Fach*, *Schrank* und *Schreibtisch*. Mit Hilfe von Containerobjekten können alle Objekte in einer *Objekthierarchie* strukturiert werden. Auf dem Schreibtisch befindliche Werkzeugobjekte dienen zur Ausführung von

generischen Arbeitsaufgaben, wie zum Beispiel dem Suchen oder Löschen eines Dokuments. Dokumentobjekte enthalten die Dokumente, in deren Bearbeitung die Arbeitsaufgabe des Benutzers besteht. Mit jedem Dokumentobjekt ist ein Werkzeugobjekt assoziiert, das die Bearbeitung des Dokuments ermöglicht. Dieses befindet sich nicht auf dem Schreibtisch, sondern wird beim Bearbeiten des Objektes gestartet.

Die Benutzer sind in einem Organigramm strukturiert. Die Stellung eines Benutzers in der Organisationsstruktur bestimmt seine Zugriffsrechte auf die Objekte im System. Ein rudimentärer Benachrichtigungsdienst<sup>1</sup> informiert über Änderungen. In Farbtafel 4.1<sup>2</sup> ist der LinkWorks-Schreibtisch eines Benutzers abgebildet.

Das System verfügt über eine integrierte objektorientierte Programmiersprache (LinkWorks Class Programming) und eine C-Schnittstelle nach außen (LinkWorks APO). Mit Hilfe dieser Schnittstellen und des Werkzeugs LinkWorks Workbench kann das System um benutzerdefinierte Klassen mit frei definierbaren Attributen und Methoden erweitert werden. Die Methoden können auch aus externen dynamischen Bibliotheken aufgerufen werden. Der Prototyp eines gemeinsamen Arbeitsbereiches<sup>3</sup> wurde als benutzerdefinierte Klasse realisiert.

Die Daten werden in einer verteilten relationalen Datenbank auf Serverrechnern verwaltet. Ein Zellenkonzept unterstützt die Verteilung und Skalierung des Systems. LinkWorks unterscheidet zwischen *Objektdateien*, die in der Datenbank der Objektzelle gespeichert sind und auf die bei jedem Objektzugriff erneut zugegriffen wird, *Dokumenten*, die den Inhalt der Dokumentobjekte (z.B. ein Word-Dokument) darstellen und als Files im sogenannten Pool-Bereich des Servers der Objektzelle im Filesystem liegen, und *Managementdaten*, die in der Datenbank der systemweit eindeutigen Managementzelle liegen und über ein Caching-Konzept zu den Objektzellen und den Client-Rechnern repliziert werden.

Client-Rechner übernehmen die Darstellung und die Interaktion mit dem Benutzer. Objekte können mit Applikationen (Werkzeugen) bearbeitet werden, die auf den Client-Rechnern installiert sind. LinkWorks ermöglicht die Einbindung beliebiger Werkzeuge.

---

<sup>1</sup> Siehe Abschnitt 4.3.2 für eine kritische Darstellung.

<sup>2</sup> Die Farbtafeln befinden sich aus drucktechnischen Gründen am Abschnittsende ab Seite **Fehler!**  
**Textmarke nicht definiert..**

<sup>3</sup> Siehe Abschnitt 7.

## 4.3 Das Problem: Meldungen im gemeinsamen Arbeitsbereich

### 4.3.1 Zielstellung der vorliegenden Arbeit

Teilfunktionalitäten des Systems POLITeam sind der gemeinsame Arbeitsbereich und der Benachrichtigungs- und Informationsdienst. In einem gemeinsamen Arbeitsbereich laufen Aktivitäten mehrerer Kooperationspartner zeitgleich oder zeitversetzt ab. Für eine effektive Kooperation ist es wünschenswert, Meldungen über Aktivitäten und Verfügbarkeit anderer Benutzer (Awarenessinformation) darzustellen, um einen gemeinsamen Kontext für die kooperative Arbeit zu schaffen und zu erhalten. Das ist die Funktion des Benachrichtigungs- und Informationsdienstes.

Awarenessinformation kann sehr umfangreich werden und im Extremfall zu einer Flut von Meldungen führen, die den Benutzer mehr stören als unterstützen. Deshalb ist es notwendig, diese Information zu filtern. Dazu muß ein Mechanismus zum Spezifizieren der Relevanz einer Meldung gefunden werden. Die Relevanz einer Meldung steuert dann ihre Darstellung mit einer ihrer Wichtigkeit entsprechenden Präsentationstechnik im Arbeitskontext des Empfängers.

### 4.3.2 Ist-Zustand

Der Benachrichtigungsdienst im bestehenden System LinkWorks ist ein Beispiel für ein schlechtes Design. Farbtafel 4.2 skizziert ein Beispiel für den Ist-Zustand. Hier hat ein Benutzer Interesse an den Änderungen der Objekte im Schrank „POLITeam“ angemeldet. Bei jeder Änderung eines beliebigen Objektes in diesem Schrank erscheint nun eine modale Meldebox, deren Meldungstext keinen Rückschluß auf das geänderte Objekt zuläßt. In Farbtafel 4.2 wurde das Dokument „Designpapier“ modifiziert.

Die Schwachstellen des existierenden Benachrichtigungsdienstes in zusammengefaßter Form:

- LinkWorks stellt alle Meldungen in einer modalen Meldebox dar, die den Benutzer in seiner laufenden Tätigkeit unterbricht und die er schließen muß, um weiterarbeiten zu können.
- Die Relevanz einer Meldung kann nur für Änderungen an Objekten und nur in den beiden Stufen „anzeigen“ oder „nicht anzeigen“ vom Benutzer eingestellt werden. Dazu wird das Attribut „Änderungen anzeigen“ eines Objektes gesetzt. Das wird durch ein kleines Ausrufezeichen rechts vom Objekticon dargestellt (siehe Farbtafel 4.2).

- Es ist unmöglich, aus einer Ereignismeldung das Objekt zu ermitteln, auf das sich die Meldung bezieht. Das System zeigt jedes Ereignis innerhalb der Objekthierarchie als Änderung desjenigen Objektes an, dessen Attribut „Änderungen anzeigen“ gesetzt ist und das beim Durchlaufen der Objekthierarchie vom geänderten Objekt aufwärts zur Wurzel als erstes passiert wird.
- Es ist nicht möglich, direkt aus dem Fenster der Ereignismeldung Folgeaktionen zu starten, wie beispielsweise das Lesen einer Mail, deren Ankunft gemeldet wurde, oder ein direktes Anzeigen eines geänderten Objektes.
- Die Relevanz einer existierenden Meldung kann nicht geändert werden. Nach dem Quittieren einer Ereignismeldung durch Schließen der Meldebox ist die Meldung nur auf explizite Nachfrage über das Ereignisbuch erreichbar.

### 4.3.3 Sollkonzept

Dieser Zustand soll verbessert werden. Folgende Designziele werden dabei verfolgt:

- Das Interesse an einem Ereignis ist abgestuft vom Nutzer spezifizierbar.
- Die Ereignismeldungen werden durch Präsentationstechniken mit abgestufter Dringlichkeit dargestellt.
- Das Verfolgen von Ereignismeldungen innerhalb der Containerhierarchie ist möglich.
- Aus einer Ereignismeldung können Folgeaktionen gestartet werden.
- Die Relevanz einer Ereignismeldung kann abgestuft geändert werden.

Dazu sind die folgenden Teilprobleme zu lösen:

- Entwicklung eines Modells für Ereignisse und Meldungen
- Untersuchung von Möglichkeiten zum Spezifizieren der Relevanz einer Meldung für einen bestimmten Benutzer
- Entwickeln von Präsentationstechniken mit unterschiedlicher Dringlichkeit
- Darstellung der Meldungen abhängig von ihrer Relevanz
- Implementation eines ersten Prototypen zur Veranschaulichung einiger Erkenntnisse

### 4.3.4 Weiteres Vorgehen

Im folgenden Abschnitt wird ein Ereignis-Interesse-Meldungs-Modell entwickelt, das anhand von Benutzerinteresse an Ereignissen Meldungen mit zugeordneter Relevanz erzeugt. Danach werden Techniken zur Darstellung der Meldungen diskutiert und schließlich ein Prototyp beschrieben, der einige der vorgestellten Ideen veranschaulicht.

---

# 5 Ein Modell für Meldungen

---

## 5.1 Grundlagen

### 5.1.1 Motivation

Wie in Abschnitt 3 dargestellt, können Meldungen über die Aktivitäten von Kooperationspartnern die Effektivität kooperativen Arbeitens erhöhen. Um die Ereignismeldungen zu filtern, muß jeder Benutzer selbst festlegen können, wie relevant eine Meldung über ein bestimmtes Ereignis für ihn ist. Das muß sowohl für existierende Meldungen als auch für Meldungen über zukünftige Ereignisse möglich sein. Die Relevanz bestimmt, ob und wie eine Meldung dargestellt wird.

Der oben genannte Aspekt wird in existierenden Systemen nicht oder nur unzureichend beachtet. DIVA<sup>1</sup> beispielsweise stellt zwar Ereignismeldungen dar, jedoch erfolgt die Darstellung für jeden Benutzer in der gleichen Art und Weise. Dem Autor ist kein System bekannt, das Ereignismeldungen abhängig von einer benutzerdefinierbaren Relevanz darstellt.

Daher soll in diesem Abschnitt ein neuartiges Modell für Ereignismeldungen entwickelt werden, das die in Abschnitt 4.3.3 aufgestellten Forderungen erfüllt. Es zeichnet sich insbesondere durch die konsequente Entkopplung der Ereignismeldungen von den Ereignissen aus. Dadurch kann jeder Benutzer die Relevanz einer an ihn gerichteten Ereignismeldung sowohl vor Eintreten des Ereignisses spezifizieren als auch nachträglich modifizieren, ohne mit anderen Benutzern in Konflikt zu geraten.

---

<sup>1</sup> Vgl. [SOH94]

### 5.1.2 Voraussetzungen

Bevor der Autor mit der Entwicklung eines Modells für Ereignisse, Interesse und Meldungen beginnt, will er zwei wichtige Voraussetzungen für ein solches Modell einführen: die Objektklassen, mit denen man die Grundfunktionalität eines gemeinsamen Arbeitsbereiches modellieren kann, und den Begriff des Kontextes.

#### a) Objektklassen zur Modellierung eines gemeinsamen Arbeitsbereiches

In Abbildung 5.1 ist der Baum der Objektklassen dargestellt, die zur Modellierung eines gemeinsamen Arbeitsbereiches benötigt werden und die Grundlage für die Einführung des Ereignis-Interesse-Meldungs-Modells bilden. Die Objektklassen ergeben sich aus den im Projektantrag dargelegten Anforderungen an die Grundfunktionalität des Gemeinsamen Arbeitsbereiches.

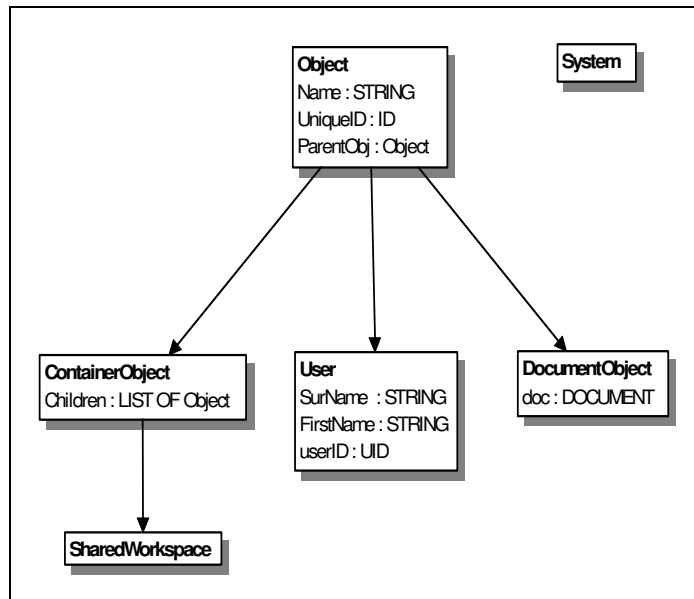


Abbildung 5.1: Objektklassen im gemeinsamen Arbeitsbereich

Die Klasse *Object* ist die Basisklasse für alle anderen Klassen. *ContainerObject* stellt einen Container zur Verfügung, der Objekte der Klassen *ContainerObject* und *DocumentObject* enthalten kann und somit als Strukturierungsmittel dient. Durch das Ineinanderschachteln von Containerobjekten entsteht die *Containerhierarchie*. Diese ist ein Baum, dessen Knoten Containerobjekte sind und dessen Wurzelobjekt der gemeinsame Arbeitsbereich ist. Ein Container kann auch Objekte enthalten, die keine Container sind<sup>1</sup>. Für eine Containerhierarchie, in der einige Knoten Nicht-Containerobjekte enthalten, soll der allgemeinere Begriff der *Objekthierarchie*<sup>2</sup> verwendet werden. Die Objekthierarchie ist ein Baum, dessen Nicht-Blattknoten Containerobjekte und dessen Blattknoten Objekte einer beliebigen Klasse sind. Ein *ContainerObject* kann *geöffnet* oder *geschlossen* sein. Geöffnet bedeutet dabei, daß ein Fenster auf dem

<sup>1</sup> Das kann beispielsweise ein Dokumentobjekt sein.

<sup>2</sup> Die Objekthierarchie darf nicht mit einer Klassenhierarchie verwechselt werden. Während in ersterer die Knoten von *Objektinstanzen* gebildet werden und die Kanten die Beziehung „Objekt ist in Containerobjekt enthalten“ darstellen, sind bei letzterer die Knoten *Objektklassen*, und die Kanten repräsentieren die Beziehung „Objektklasse erbt von Elternobjektklasse“.

Bildschirm existiert, das die Icons der im Container enthaltenen Objekte darstellt. Geschlossen heißt, daß kein solches Fenster existiert.

*SharedWorkspace* modelliert den gemeinsamen Arbeitsbereich und ist ein *Container-Object*, das zusätzlich Benutzer enthalten kann. Diese werden als Objektklasse *User* dargestellt. Objekte der Klasse *DocumentObject* schließlich enthalten die Dokumente, in deren Bearbeitung die kooperative Tätigkeit besteht. Die Klasse *System* kann ebenfalls Quelle von Ereignissen (wie z.B. Fehlermeldungen) sein.

## b) Kontext

Ein Kontext stellt eine Arbeitssituation dar. Eine vollständige Kontextbeschreibung kann beliebig detailliert sein, wie zum Beispiel: „Benutzer A editiert Dokument X, das sich im Container Y befindet. Gleichzeitig modifiziert Benutzer B eine Kopie dieses Dokuments, die sich im gleichen Container befindet“. In diesem Kontext kann eine Meldung über die Aktivität von B für A eine hohe Relevanz aufweisen. Jedoch ist eine derartige Kontextbeschreibung sehr schwierig umzusetzen. Ein Grund dafür ist der hohe Aufwand des Benutzers für die Spezifizierung eines Kontextes. Es müßten der Zustand von Objekten, die Handlungen von Benutzern, Beziehungen zwischen Objekten und die Vorgeschichte der Objekte betrachtet werden.

Im Rahmen des Projektes POLITeam wird eine Lösung benötigt, die sich durch eine einfache Durchschaubarkeit und einen geringen Aufwand für den Benutzer auszeichnet. Aus Gesprächen mit den Anwendungspartnern und aus eigenen Erfahrungen mit der Nutzung des Systems LinkWorks im Projektteam ist bekannt, daß die Dokumentobjekte im System meist nach inhaltlichen Kriterien in Containerobjekten angeordnet werden. Der POLITeam-Projektordner in LinkWorks wurde beispielsweise nach Arbeitspaketen untergliedert. Die Anwendungspartner in den Ministerien halten alle zu einem Vorgang gehörenden Objekte in einer elektronischen Laufmappe. Daher faßt der Autor einen Kontext vereinfacht als ein Containerobjekt auf, das die zur Erledigung einer Arbeitsaufgabe notwendigen Objekte enthält. Die Containerhierarchie bildet somit eine Struktur einander übergeordneter Kontexte. Meldungen können innerhalb dieser Hierarchie in übergeordnete Kontexte weitergeleitet werden und dort einen Überblick über die Geschehnisse in untergeordneten Kontexten vermitteln.

Ein Containerobjekt kann - wie im vorigen Abschnitt dargestellt - zwei Zustände annehmen: geöffnet und geschlossen. Diese Zustände sollen auch für Kontexte gelten. Meldungen lassen sich einem Kontext zuordnen, in dem sie relevant sind. Das ist der sogenannte *Meldekontext*. Die Meldung wird angezeigt, wenn der Benutzer ihren Meldekontext geöffnet hat. Neben dem Meldekontext gibt es noch den *Entstehungs-*

*kontext.* Das ist der Container, in dem das gemeldete Ereignis auftritt, also in dem sich zum Ereigniszeitpunkt das Ereignisobjekt befindet.

### 5.1.3 Überblick über das Modell

Das Modell trennt konsequent zwischen einem Ereignis und der Meldung über dieses Ereignis an einen Empfänger. Die Entkopplung geschieht über das Interesse des Benutzers an dem Ereignis. Die initiale Relevanz der Ereignismeldung wird von diesem Interesse bestimmt. Abbildung 5.2 gibt einen Überblick über das Zusammenspiel der Modellelemente.

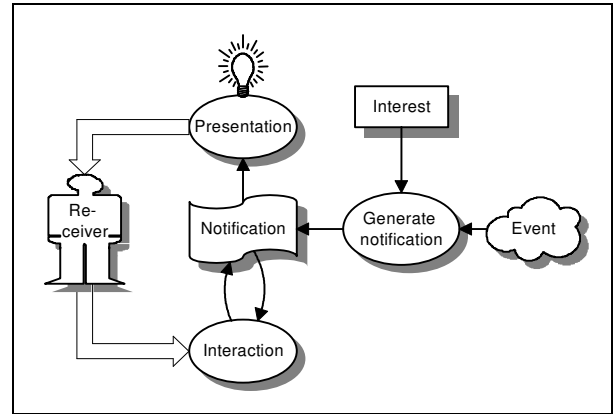


Abbildung 5.2: Überblick über das Modell

Wenn ein Ereignis auftritt, so wird geprüft, ob es Benutzer gibt, die Interesse an dem Ereignis angemeldet haben. Ist das der Fall, erzeugt das System für jeden dieser Benutzer eine Meldung, deren Relevanz abhängig vom jeweiligen Interesse ist. Die Meldung wird dem Benutzer mit einer Darstellungstechnik präsentiert, die der Relevanz der Meldung entspricht. Durch Interaktion mit der Meldung kann ihre Relevanz nachträglich vom Empfänger geändert werden, was sich auf die Art der Darstellung auswirkt. Damit wird das Reagieren auf die Meldung ermöglicht.

In das geschilderte Grundmodell lassen sich einfach Filter zur Implementation von Zugriffsschutzmechanismen integrieren. Ein Ereignisfilter könnte z.B. nur die Ereignisse durchlassen, die der Benutzer auch freigeben möchte. Ein

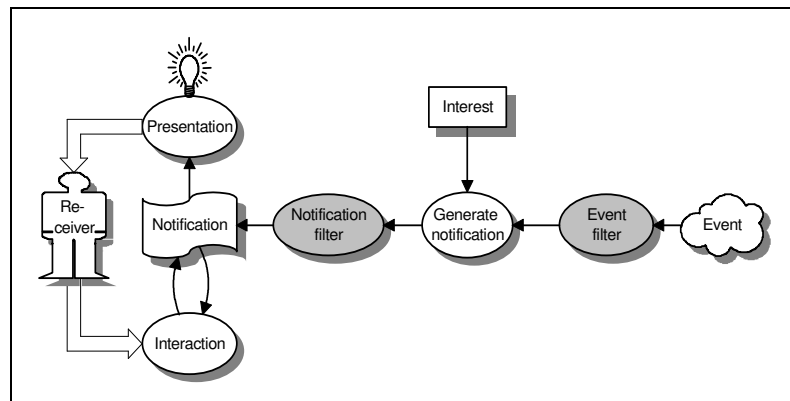


Abbildung 5.3: Erweiterung der Meldungserzeugung um Filter

Meldungsfilter erlaubt das Ausschließen von bestimmten Empfängern für eine Meldung. Auf die Filter (siehe Abbildung 5.3) wird im Rahmen der Diplomarbeit nicht weiter eingegangen.

## 5.2 Ereignisse, Interesse und Meldungen

### 5.2.1 Ereignis

#### a) Ereignis und Aktivität

Ein Ereignis ist eine Zustandsänderung eines Objektes zu einem bestimmten Zeitpunkt, also mit einer Zeitdauer von Null. Eine Aktivität ist ein Prozeß mit einer Zeitdauer größer als Null, der seinerseits den Zustand von Objekten ändern, also Ereignisse erzeugen kann. Jede Aktivität wird durch ein Start- und ein Endereignis begrenzt. Demnach kann man eine Aktivität durch Ereignisse beschreiben.

Awarenessinformation beinhaltet sowohl Ereignisse als auch Aktivitäten. Aktivitäten (z.B. aktuell im System aktive Benutzer) sind nur im synchronen Modus interessant. Ereignisse, die keine Start- bzw. Endereignisse sind, interessieren in beiden Awarenessmodi.

#### b) Der Ereignisbegriff

Da ein Ereignis den Zustand der gemeinsam genutzten Objekte und der Benutzer im Arbeitsbereich ändert, ist es der primäre Träger von Awarenessinformation. Beim Eintreten eines Ereignisses müssen die entsprechenden

Fragen des Benutzers	Zu erfassende Daten
WAS ist passiert?	Ereignistyp
WANN ist es passiert?	Ereigniszeitpunkt
WO ist es passiert?	Ereignisobjekt, ggf. Elternobjekt
WER hat es getan?	Ereignisinitiator
Welche weitere Information gibt es?	weitere Parameter

Daten erfaßt und gespeichert werden. Welche Daten sind nun zu speichern? Tabelle 3.3 gibt eine Übersicht über zum Gebiet der Awareness gehörige Benutzeraktivitäten. Daraus kann der Informationsbedarf eines Benutzers bezogen auf Ereignisse in einem gemeinsamen Arbeitsbereich abgeleitet werden. Er ist in Tabelle 5.1 dargestellt. Diese Vorüberlegungen legen die Speicherung eines Ereignisses als Tupel nahe. Der Autor definiert auf dieser Grundlage die Menge aller Ereignisse wie folgt als kartesisches Produkt:

$E : T \times Y \times P_y \times U \times O \times O_c$ , mit den Größen:

- E - Menge der Ereignisse
- T - Zeit
- Y - Menge der Ereignistypen
- $P_y$  - vom Ereignistyp abhängige Parametermenge
- U - Menge der Benutzer
- O - Menge der Objekte
- $O_c$  - Menge der Containerobjekte

Damit ist ein *Ereignis*  $e$  ein Tupel, das aus den Feldern *Ereigniszeitpunkt*  $t$ , *Ereignistyp*  $y$ , vom Ereignistyp abhängigen optionalen *Parametern*  $p_y$ , *Ereignisinitiator*  $u$ , *Ereignisobjekt*  $o$  und *Elternobjekt*  $o_y$  besteht:

$$e := (t, y, p_y, u, o, o_y)$$

Der Ereignistyp drückt die Art der durch das Ereignis repräsentierten Zustandsänderung aus. Die Parameter enthalten abhängig vom Ereignistyp zusätzliche Information über die Zustandsänderung.

In Abhängigkeit vom Wert des Feldes  $u$  können *Objekt-* und *Systemereignisse* unterschieden werden. Erstere beziehen sich auf die Objekte im gemeinsamen Arbeitsbereich, letztere drücken Zustandsänderungen des Groupwaresystems aus<sup>1</sup>. Das Feld  $u$  ist bei einem Systemereignis leer und bezeichnet bei einem Objektereignis den Benutzer des Arbeitsbereiches, von dem das Ereignis ausgelöst wurde. Die Felder  $o$  und  $o_y$  identifizieren die Kombination Objekt und Elternobjekt, auf die sich das Ereignis bezieht. Bei den meisten Ereignistypen ergibt sich das Elternobjekt des Ereignisobjektes implizit aus der Stellung des Objektes  $o$  in der Objekthierarchie. Daher ist auch  $o_y$  ein optionaler Parameter. Relevant ist  $o_y$  nur beim Löschen und Herausnehmen eines Objektes, da durch diese Operationen die ursprüngliche Strukturbeziehung zwischen Eltern- und Kindobjekt aufgehoben wird.

Der Wertebereich für den Ereignistyp  $Y$  hängt von der Klasse des Objektes ab, d.h., nicht alle Ereignistypen sind bei allen Objektklassen möglich.

In einem gemeinsamen Arbeitsbereich relevante Ereignis- bzw. Aktivitätstypen und ihr Bezug auf Benutzer und Objektklassen sind in Abbildung 5.4 dargestellt.

Ereignis- bzw. Aktivitätstyp	Semantik
CreateEv	Objekt in Containerobjekt erzeugen
DeleteEv	Objekt in Containerobjekt löschen
MoveToEv	Objekt in Containerobjekt hineinlegen
MoveFromEv	Objekt aus Containerobjekt herausnehmen
ModifyEv	Objektdokument verändern
RenameEv	Objekt umbenennen
LoggedInActivity	Anwesenheit des Benutzers im Arbeitsbereich
EditActivity	Objektdokument ist zum Bearbeiten geöffnet
ReadActivity	Objektdokument ist zum Lesen geöffnet
SystemEvent	Systemereignis

Tabelle 5.2: Semantik der Ereignistypen

Tabelle 5.2 enthält die Semantik der Ereignistypen. Sie entstammen einer Analyse bereits vorhandener Ereignistypen in LinkWorks und wissenschaftlicher Publikationen<sup>2</sup>.

<sup>1</sup> Das könnte z.B. das Auftreten eines Fehlers oder das bevorstehende Herunterfahren des Systems sein.

<sup>2</sup> Vgl. insbesondere [FUC94]

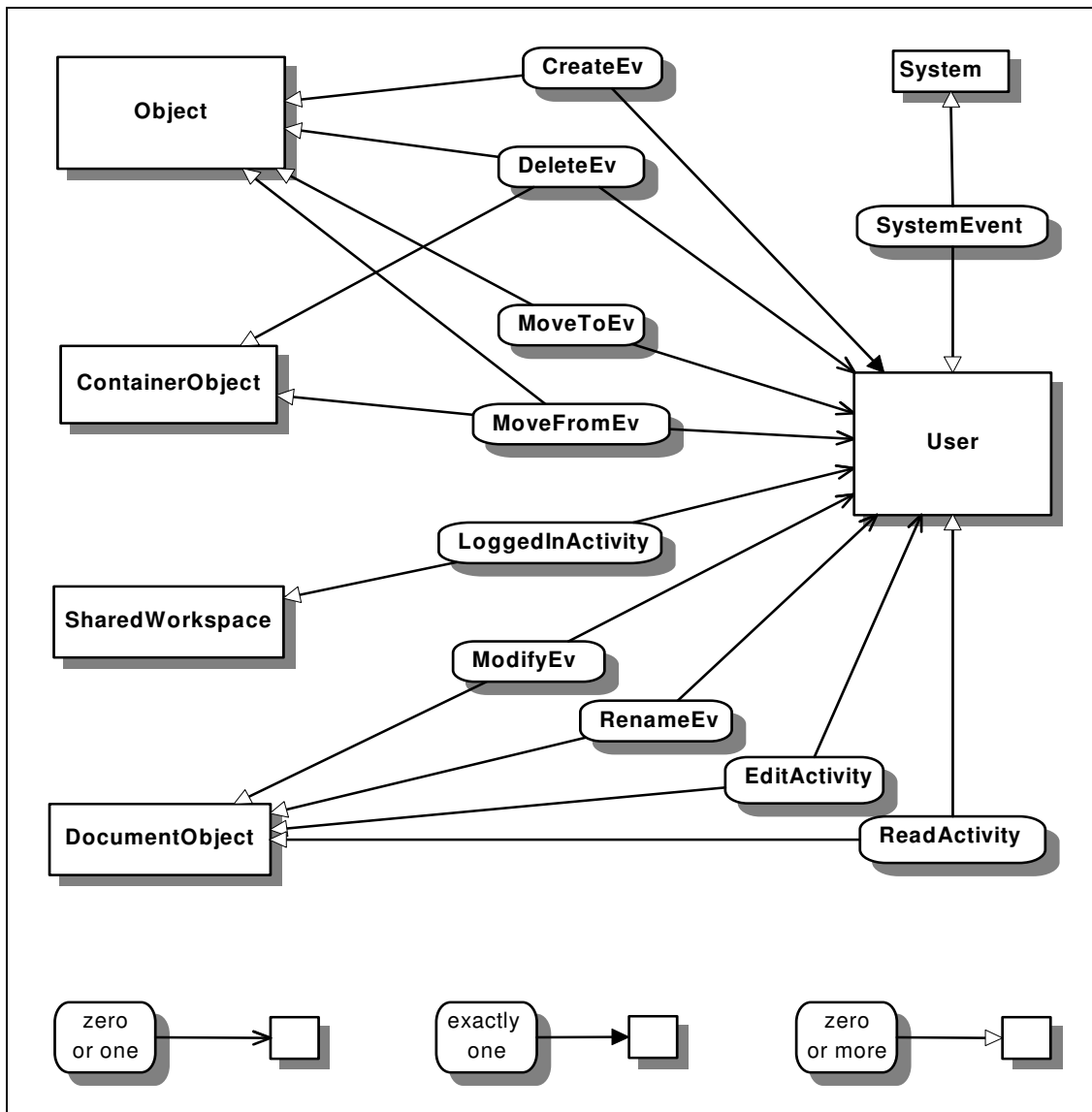
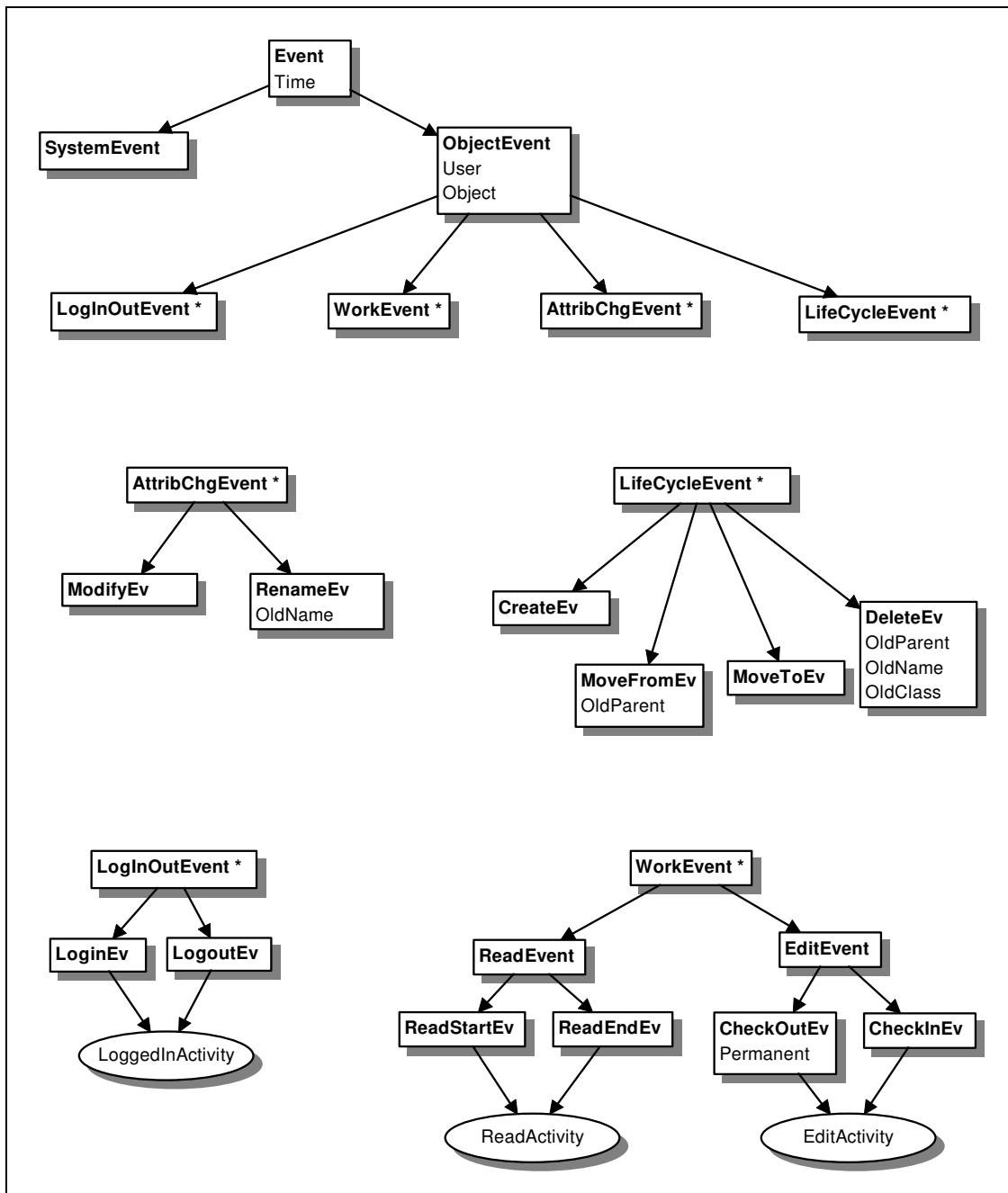


Abbildung 5.4: Ereignis- bzw. Aktivitätstypen und ihr Bezug auf Benutzer und Objektklassen

### c) Ereignisklassen

Softwaretechnisch betrachtet kann ein Ereignis als Objekt modelliert werden. Der Ereignistyp wird dann durch die Objektklasse dargestellt, die restlichen Felder des Tupels als Attribute der Klasse. Ein Ereignis wird immer von einer Methode des betroffenen Objektes generiert. Diese trägt für die korrekte Initialisierung der Attribute des Ereignisses Sorge. In Abbildung 5.5 ist der Baum der Ereignisklassen einschließlich der von Start- und Endereignissen begrenzten Aktivitäten dargestellt.

Abbildung 5.5: Ereignisklassen<sup>1</sup>

<sup>1</sup> Abstrakte Klassen enden auf „Event“, konkrete auf „Ev“. Ein Stern in einem Blattknoten eines Baumes zeigt an, daß eine verfeinerte Darstellung dieses Knotens existiert.

## 5.2.2 Interesse

### a) Der Interessebegriff

Das Modell trennt konsequent Ereignisse und Ereignismeldungen. Beim Erzeugen einer Ereignismeldung erfolgt die Trennung über das Interesse eines Benutzers an dem Ereignis. Interesse stellt entsprechend der in Tabelle 5.1 aufgelisteten Fragen des Benutzers ein Muster dar, das bei der Meldungserzeugung ausgewertet wird und die Relevanz der Meldung bestimmt. Dieses Muster kann - wie ein Ereignis - als Tupel dargestellt werden. Es enthält Felder, die den Fragen des Benutzers entsprechen und die bei der Auswertung mit den korrespondierenden Feldern des Ereignistupels verglichen werden können. Weiterhin sind Felder enthalten, die die Relevanz<sup>1</sup> der zu erzeugenden Meldung ausdrücken. Wenn das Interessemuster auf das Ereignis „paßt“, so ergibt sich die Relevanz der zu erzeugenden Meldung aus diesen Relevanzfeldern.

Formal definiert der Autor nach diesen Vorüberlegungen die Menge aller Interessemuster wie folgt:

- $I: O \times \mathbf{P}(U) \times Y \times \text{Exp}(P_y) \times U \times W \times B \times O_c$ ; mit den Größen
- I - Menge aller Interessemuster
  - O - Objektmenge
  - U - Benutzermenge
  - $\mathbf{P}(U)$  - Potenzmenge der Menge U
  - Y - Menge der Ereignistypen
  - $\text{Exp}(P_y)$  - Menge der logischen Ausdrücke auf der Parametermenge des Ereignistyps
  - W - Menge der möglichen Wichtigkeiten
  - B - Menge der booleschen Werte
  - $O_c$  - Menge der Containerobjekte.

Ein Interessemuster  $i$  ist dann ein Tupel der folgenden Form:

- $i := (o, U_o, y, \text{exp}(p_y), u_o, w, b, c_m)$ ; mit den Größen
- $o$  - Objekt, auf das sich das interessante Ereignis bezieht (Ereignisobjekt)
  - $U_o$  - Menge der Benutzer, die das Ereignis auslösen
  - $y$  - Ereignistyp
  - $\text{exp}(p_y)$  - logischer Ausdruck auf Parametern des Ereignisses
  - $u_o$  - Benutzer, der Interesse anmeldet
  - $w$  - Wichtigkeit
  - $b$  - Flag für einmalige / mehrmalige Anwendung des Interesses
  - $c_m$  - Meldekontext.

<sup>1</sup> Die Relevanz einer Meldung wird im Abschnitt 5.2.3 (b)) beschrieben.

Das Einmaligkeitsflag dient zur Modellierung von Interesse, das nur beim erstmaligen Eintreten eines Ereignisses besteht. Eine Einsatzmöglichkeit ist die Lesebestätigung einer Nachricht. Hier interessiert den Absender nur, ob der Empfänger die Nachricht gelesen hat oder nicht, also erlischt das Interesse nach dem erstmaligen Eintreten des Ereignisses „Nachrichtenobjekt gelesen“.

Die Wichtigkeit drückt den Grad des Interesses eines Benutzers an einem Ereignis aus und bestimmt die Dringlichkeit der Präsentation der entsprechenden Meldung, also die Auswahl der Präsentationstechnik. In Abschnitt 6.2.3 werden fünf mögliche Dringlichkeitsstufen für Präsentationstechniken abgeleitet. Um für den Benutzer eine Transparenz der Zuordnung zu gewährleisten, sollte die Abbildung der Wichtigkeit auf die Präsentationstechnik eine 1:1-Abbildung sein. Damit werden fünf Wichtigkeitsstufen benötigt. Tabelle 5.3 listet die möglichen Werte für die Wichtigkeit auf. Für ihre Semantik wird auf Abschnitt 6.4.2 verwiesen.

Wichtigkeitsstufen
unwichtig
interessant
lokal wichtig
wichtig
sehr wichtig

**Tabelle 5.3:** Wichtigkeit

Der Meldekontext ist ein Containerobjekt, das dem Objekt übergeordnet ist, an dem Interesse angemeldet wird. Der Arbeitsbereich ist der globale Meldekontext. Wenn die Wichtigkeit hoch genug ist, erfolgt bei geöffnetem Meldekontext die Präsentation mit einer dringlichen Technik, bei geschlossenem Meldekontext mit einer weniger dringlichen Technik.

Wichtigkeit und Meldekontext bestimmen zusammen die initiale Relevanz einer erzeugten Ereignismeldung.

Ein Interessemuster  $i$  „paßt“ auf ein Ereignis  $e$ , wenn der folgende logische Ausdruck wahr ist:

$$i.o = e.o \wedge e.y = i.y \wedge e.u \in i.U_o \wedge i.exp(e.p_y).$$

In diesem Fall wird an den Benutzer  $i.u_o$  eine Meldung über das Ereignis  $e$  mit der Wichtigkeit  $i.m$  und dem Meldekontext  $i.c_m$  gesandt. Falls das Einmaligkeitsflag des passenden Interessemusters gesetzt ist, wird es danach verworfen.

Für jeden Ereignistyp kann eine Grundwichtigkeit automatisch festgelegt werden. Dazu dient die Spezifizierung von objektklassenbezogenen Default-Interessemustern. Diese können systemweit und zusätzlich benutzerdefiniert angegeben werden. Da ein Default-Interessemuster losgelöst von der Containerhierarchie existiert, ist es nicht möglich, in diesem Muster einen Default-Wert für den Meldekontext anzugeben. Deshalb wird der gemeinsame Arbeitsbereich, in dem sich jeweils das Ereignisobjekt befindet, als Default genutzt. Der Autor erarbeitete zwei verschiedene Arten der Spezifizierung und Nutzung

von Default-Interessemustern: den *ergänzenden Ansatz* und den *einschränkenden Ansatz*.

Beim ergänzenden Ansatz hat ein Default-Interessemuster dieselbe Form wie ein normales Interessensmuster. Nur wenn der Benutzer kein Interesse für ein Ereignis spezifiziert hat, wird ein solches Default-Muster genutzt.

Beim einschränkenden Ansatz dient ein Default-Interessemuster der Einschränkung des benutzerdefinierbaren Interesses. Es werden also zuerst alle Default-Muster ausgewertet, bevor das benutzerdefinierte Interesse zum Zuge kommt. Dieses wird nur berücksichtigt, wenn es nicht mit den angegebenen Grenzwerten kollidiert. Einschränkende Default-Interessemuster enthalten eine initiale, eine minimale und eine maximale Wichtigkeit. Das ermöglicht es, die Wichtigkeit von Mengen von Ereignissen zu initialisieren und zu bestimmen, inwieweit diese durch benutzerdefinierte Muster verändert werden kann. Formal kann die Menge der einschränkenden Default-Interessemuster wie folgt beschrieben werden:

$I_D : K \times Y \times \text{Exp}(P_y) \times W \times W \times W \times B$ , mit den Größen

$I_D$	- Menge der einschränkenden Default-Interessemuster
$K$	- Menge der Objektklassen
$Y$	- Menge der Ereignistypen
$\text{Exp}(P_y)$	- Menge der logischen Ausdrücke auf der Parametermenge des Ereignistyps
$W$	- Menge der möglichen Wichtigkeiten
$B$	- Menge der booleschen Werte

Ein Default-Interessemuster  $i_D$  ist dann ein Tupel der folgenden Form:

$i_D := (k, y, \text{exp}(p_y), w_i, w_{\min}, w_{\max}, b)$ ; mit den Größen

$k$	- Objektklasse
$y$	- Ereignistyp
$\text{exp}(p_y)$	- logischer Ausdruck auf Parametern des Ereignisses
$w_i$	- initiale Wichtigkeit
$w_{\min}$	- minimale Wichtigkeit
$w_{\max}$	- maximale Wichtigkeit
$b$	- Einmaligkeitsflag

Die initiale Wichtigkeit dient zum Initialisieren des Wichtigkeitsfeldes. Minimale und maximale Wichtigkeit schränken die Freiheit beim Angeben der benutzerdefinierten Wichtigkeit ein. Das Einmaligkeitsflag aus dem Default-Muster initialisiert das Einmaligkeitsflag des konkreten Interessensmusters.

## b) Interesseeintrag

Die Realisierung eines Interessemusters kann beispielsweise als Eintrag in einer Interesse-Liste eines Objektes erfolgen. Ein solcher Interesseeintrag speichert das Interesse eines Benutzers an Ereignissen bezogen auf dieses Objekt.

Bei den Ereignistypen „Objekt erzeugen“ und „Objekt in ein Containerobjekt hineinlegen“ existiert das vom Ereignisobjekt zum Zeitpunkt des Angebens von Interesse noch nicht oder liegt einem anderen Elternobjekt. Daher wird hier das Interesse am Ereignis nicht für das potentielle Ereignisobjekt angegeben, sondern für das potentielle Elternobjekt.

## c) Propagieren von Interesse

Ein einfacher Weg, Objekte in einem Kontext (Container) mit gleichen Interessemustern zu versehen, ist die *Propagierung* von Interesse. Dabei wird Interesse an einem Ereignis eines bestimmten Typs innerhalb der Objekthierarchie nach unten weitergeleitet. Es gibt zwei Möglichkeiten: Propagierung *mit Überschreiben* und *ohne Überschreiben* existierender Muster. Je Interessemuster wird dazu ein Feld hinzugefügt, das anzeigt, ob das entsprechende Muster nicht, mit oder ohne Überschreiben propagiert werden soll.

Das Propagieren erfolgt als Teil der Ermittlung des Interesses eines Benutzers an einem auftretenden Ereignis. Zunächst wird nach einem passenden Interessemuster für das Ereignisobjekt gesucht. Für die Ereignisse „Objekt erzeugen“ und „Objekt in ein Containerobjekt hineinlegen“ beginnt die Suche beim Elternobjekt des erzeugten oder hineingelegten Objektes. Das Ergebnis ist entweder ein passendes oder ein leeres Interessemuster. Mit diesem wird das Ergebnismuster initialisiert. Als nächstes wird versucht, ein für das Ereignis passendes Interessemuster beim nächsthöheren Objekt in der Containerhierarchie zu finden. Ist das nicht erfolgreich, so wird das Ergebnismuster beibehalten. Im anderen Fall gibt es in Abhängigkeit vom Propagierungsfeld des gefundenen Musters drei Möglichkeiten:

1. Das Interesse soll nicht propagiert werden.  
→ Das Ergebnismuster wird beibehalten.
2. Das Interesse soll *ohne* Überschreiben propagiert werden.  
→ Das Ergebnismuster wird mit dem neuen Muster überschrieben, falls ersteres leer ist.
3. Das Muster soll *mit* Überschreiben propagiert werden.  
→ Das Ergebnismuster wird mit dem neuen Muster überschrieben.

Diese Schrittfolge wird mit dem neuen Ergebnismuster solange beim jeweils nächsthöheren Objekt in der Containerhierarchie wiederholt, bis das Wurzelobjekt erreicht ist. Das weitere Vorgehen unterscheidet sich in Abhängigkeit vom gewählten Default-Ansatz.

Beim *ergänzenden* Ansatz wird zunächst geprüft, ob das Ergebnismuster immer noch leer ist. Nur in diesem Fall wird ein Default-Interessemuster gesucht. Die Suche beginnt bei den benutzerdefinierten Defaults der Klasse des Ereignisobjektes. Falls hier kein passendes Muster vorhanden ist, wird die Suche bei den systemweiten Defaults der Klasse fortgesetzt. Ist die Suche auch hier erfolglos, so wird sie bei den benutzerdefinierten Defaults der Elternklasse des Ereignisobjektes weitergeführt. Das setzt sich fort, bis entweder ein Muster gefunden oder die Wurzel des Klassenbaumes erreicht wurde.

Beim *einschränkenden* Ansatz erfolgt die Suche eines Defaults in jedem Fall. Das Durchlaufen des Klassenbaumes ist identisch mit dem ergänzenden Ansatz. Unterschiede gibt es in der Behandlung der Einschränkungen. Die minimale und maximale Wichtigkeit des systemweiten Defaults schränken die minimale und maximale Wichtigkeit des benutzerdefinierten Defaults ein und bilden Grenzen für die initiale Wichtigkeit. Das resultierende Wichtigkeitsintervall (der Durchschnitt der beiden Intervalle) schränkt seinerseits die Wichtigkeit eines gefundenen Ergebnismusters ein. Ist dieser leer, wird er mit den Werten aus dem resultierenden Default-Muster initialisiert.

#### **d) Implizites Interesse**

Das Spezifizieren von Interesse kann direkt durch Benutzereingaben oder automatisch als Teil einer Benutzeraktivität erfolgen. Im ersten Fall liegt explizites, im zweiten Fall implizites Interesse vor. Implizites Interesse ist eine Möglichkeit, den Aufwand des Benutzers beim Spezifizieren von Interesse zu senken. Dabei erzeugt das Starten einer Aktivität ein Interessemuster, das mit dem Ende der Aktivität wieder gelöscht wird. Beim Lesen eines Dokuments wäre zum Beispiel implizites Interesse an parallel durchgeführten Änderungen sinnvoll.

### 5.2.3 Meldung

#### a) Der Meldungsbegriff

Eine Meldung ist eine Information über den Eintritt eines *Ereignisses*, die eine gewisse *Relevanz* hat, über bestimmte *Interaktionsmöglichkeiten* verfügt und an einen *Benutzer* gerichtet ist. In Abhängigkeit von der Relevanz einer Meldung wird die Art ihrer *Präsentation* bestimmt. Präsentation ist die Darstellung einer Meldung mit einer mehr oder weniger dringlichen *Präsentationstechnik*<sup>1</sup>. Um dem Benutzer einen Überblick über die Geschehnisse in untergeordneten Kontexten zu ermöglichen, werden Meldungen in der Containerhierarchie *propagiert*. Das heißt, sie werden aus dem Kontext ihrer Entstehung in übergeordnete Kontexte weitergeleitet.

Der Autor beschreibt die Menge aller Meldungen als kartesisches Produkt:

$$M : E \times R \times \mathbf{P}(Op) \times U, \text{ mit den Größen}$$

M	- Menge aller Meldungen
E	- Menge aller Ereignisse
R	- Menge der möglichen Relevanzen
Op	- Menge aller Interaktionsmöglichkeiten
$\mathbf{P}(Op)$	- Potenzmenge der Menge Op
U	- Menge aller Benutzer

Eine Meldung ist dann ein Tupel der Form:

$$m := (e, r, op, u)$$

Die Komponenten *Ereignis*  $e$ , *Relevanz*  $r$ , *Empfänger*  $u$  und *Interaktionsmöglichkeiten*  $op$  beschreiben eine *Meldung*  $m$ . Der Absender der Meldung ist der Initiator des Ereignisses  $e$ . Die Interaktionsmöglichkeiten ergeben sich aus dem Ereignistyp und der Relevanz. Ihre Verfügbarkeit an der Benutzeroberfläche ist von der Interaktivität<sup>2</sup> der Präsentationstechnik abhängig, die ihrerseits anhand der Relevanz der Meldung ausgewählt wird<sup>3</sup>.

Wenn ein Ereignis auftritt, wird für jeden Benutzer abhängig von seinem Interesse an dem entsprechenden Ereignis eine Meldung generiert. Die Relevanz wird entsprechend des Interesses des Empfängers initialisiert. Eine Meldung über eine *Aktivität* kann erzeugt werden, indem die Meldung bei Auftreten des Startereignisses erzeugt und bei Auftreten des Endereignisses gelöscht wird.

<sup>1</sup> Präsentationstechniken werden im Abschnitt 6 beschrieben.

<sup>2</sup> Siehe Abschnitt 6.2.2 (b)) für die Erklärung der Interaktivität einer Präsentationstechnik.

<sup>3</sup> Abschnitt 6.4 beschreibt den Auswahlmechanismus einer Präsentationstechnik anhand der Relevanz.

Meldungen besitzen einen Entstehungskontext Das ist der Container, der das Objekt beim Eintreten des Ereignisses enthält.

### b) Relevanz einer Meldung

Die Relevanz ist eine wesentliche Komponente einer Meldung, da sie die Darstellung der Meldung steuert. Es wurde bereits bei der Einführung des Interesse-Begriffs dargelegt, daß die Wichtigkeit einer Ereignismeldung die Auswahl der Präsentationstechnik bestimmt. Um den Benutzer nicht mit Meldungen zu überfluten, müssen diese gefiltert werden. Dazu bietet sich die Einführung eines Meldekontextes an. Die Darstellung der Meldung erfolgt, wenn der Arbeitskontext des Benutzers dem Meldekontext entspricht. Eine Erinnerungsfunktion und das Bestätigen von Meldungen sollten ebenfalls möglich sein. Die Relevanz einer Meldung drückt somit aus, wie wichtig die Meldung dem Empfänger in welchem Kontext ist und ob er von der Meldung bereits Kenntnis genommen oder sie auf einen späteren Zeitpunkt zurückgestellt hat.

Ein Relevanzmodell, das obige Forderungen erfüllt, sei hier vorgeschlagen:

$R : W \times B \times O_c \times T$ ; mit den Größen:

- R - Menge der möglichen Relevanzen,
- W - Menge der möglichen Wichtigkeiten,
- B - Menge der booleschen Werte,
- $O_c$  - Menge der Containerobjekte,
- T - Zeit.

$r := (w, s, c, t)$

Die Relevanz  $r$  einer Meldung ist demnach ein Tupel, das aus den Komponenten *Wichtigkeit*  $w$ , *Informationsstatus*  $s$ , *Meldekontext*  $c$  und *Zurückstellungszeit*  $t$  besteht.

Anhand des Interesses des Meldungsempfängers am aufgetretenen Ereignis werden die initialen Werte für Wichtigkeit und Kontext ermittelt. Dabei stellt der Meldekontext den Container in der Hierarchie dar, in dem die Meldung relevant ist und daher mit einer der Wichtigkeit entsprechenden Präsentationstechnik dargestellt werden soll. Die Darstellung erfolgt, wenn der Kontext geöffnet ist.

Die Zurückstellungszeit dient einer Erinnerungsfunktion. Im Informationsstatus wird gespeichert, ob der Benutzer von der Meldung bereits Kenntnis genommen hat.

### c) Interaktionsmöglichkeiten einer Meldung

Die Interaktionsmöglichkeiten ermöglichen das Reagieren des Empfängers auf die Meldung. Dabei kann zwischen Operationen auf der Meldung, auf dem Ereignisobjekt und auf dem Ereignisinitiator unterschieden werden.

Folgende Operationen sind sinnvoll:

- Markierung der Meldung als „gesehen“
- Zurückstellen der Meldung auf einen späteren Zeitpunkt
- Quittieren (Löschen) der Meldung
- Detaillieren der Meldung
- Anzeige des Ereignisobjektes
- Aufbauen einer synchronen<sup>1</sup> oder asynchronen<sup>2</sup> Kommunikation zum Ereignisinitiator

Die ersten beiden Operationen beeinflussen die Relevanz der Meldung. Markieren als „gesehen“ ändert den Informationsstatus, das Zurückstellen setzt die Zurückstellungszeit. Das Quittieren löscht die Meldung. Die restlichen Interaktionsmöglichkeiten dienen dazu, die durch die Meldung erhaltene Information in der kooperativen Arbeit umzusetzen. Beispielsweise kann der Meldungsempfänger durch Anzeigen des Ereignisobjektes von gemeldeten Änderungen Kenntnis nehmen oder einen Kommunikationskanal für eine Rückfrage nutzen.

Unsinnige Kombinationen von Operationen auf Meldungen müssen gesperrt werden. Das betrifft insbesondere die folgenden Kombinationen:

- Eine bereits als „gesehen“ markierte Ereignismeldung kann nicht noch einmal als „gesehen“ markiert werden.
- Eine bereits zurückgestellte Ereignismeldung kann bis zum Ablauf des Zurückstellungsintervalls nicht noch einmal zurückgestellt werden.
- Propagierte Meldungen können weder als „gesehen“ markiert noch zurückgestellt werden. Durch einen Doppelklick auf die entsprechende Zeile kann der Propagierungsweg entweder durch Öffnen einer untergeordneten Hierarchiestufe weiterverfolgt oder eine solche geschlossen werden, wenn sie geöffnet ist.

---

<sup>1</sup> z.B. Videokonferenz oder Talk-Programm

<sup>2</sup> z.B. Verschicken einer E-Mail

#### d) Ausdünnen und Annullieren von Meldungen

Zur Vermeidung von Informationsüberflutung und zur Konsistenzsicherung müssen veraltete Meldungen gelöscht werden. Hier unterscheidet der Autor zwei Operationstypen.

##### 1) Ausdünnen von Meldungen

Beim Ausdünnen wird die ältere von zwei Meldungen gelöscht, wenn die gemeldeten Ereignisse denselben Typ haben und dieselbe Objekt-Elternobjekt-Kombination betreffen. Diese Operation dient zur Vermeidung von Informationsüberflutung: Je Objekt-Elternobjekt-Kombination wird nur die Meldung über das aktuellste Ereignis des jeweiligen Ereignistyps beibehalten.

##### 2) Annullieren einer Meldungskombination

Beim Annullieren wird eine von zwei Meldungen gelöscht, wenn das Ereignis der ersten Meldung einen bestimmten Typ aufweist, das Ereignis der zweiten Meldung einen bestimmten anderen Typ aufweist, beide dieselbe Objekt-Elternobjekt-Kombination betreffen und das Ereignis der zweiten Meldung zeitlich auf das Ereignis der ersten Meldung folgt. Mit dieser Operation können Aktivitäten modelliert und Widersprüche zwischen einander ausschließenden Ereignissen beseitigt werden. So wird zum Beispiel beim Einloggen eines Benutzers die Meldung über dessen letztes Logout-Ereignis annulliert, beim Ausloggen analog das korrespondierende Login-Ereignis.

#### e) Lebenszyklus einer Meldung

Meldungen haben einen *Lebenszyklus*. Durch die Interaktion des Benutzers mit der Meldung ändert sich ihre Relevanz. Die Operationen Ausdünnen und Annullieren löschen Meldungen. Der Lebenszyklus kann durch ein Zustandsübergangsdiagramm dargestellt werden. Zum Verständnis der Diagramme soll an dieser Stelle noch aus Abschnitt 6.4 vorweggenommen werden, daß „sehr wichtige“ Ereignismeldungen in einer Meldebox dargestellt werden und alle existierenden Ereignismeldungen, deren Wichtigkeit größer als „unwichtig“ ist, generell im Objekticon codiert werden<sup>1</sup>.

---

<sup>1</sup> Zur Wahrung der Übersichtlichkeit werden nur beiden Präsentationstechniken „Meldebox“ und „Icondarstellung“ betrachtet. Für eine ausführliche Beschreibung von Präsentationstechniken siehe Abschnitt 6.

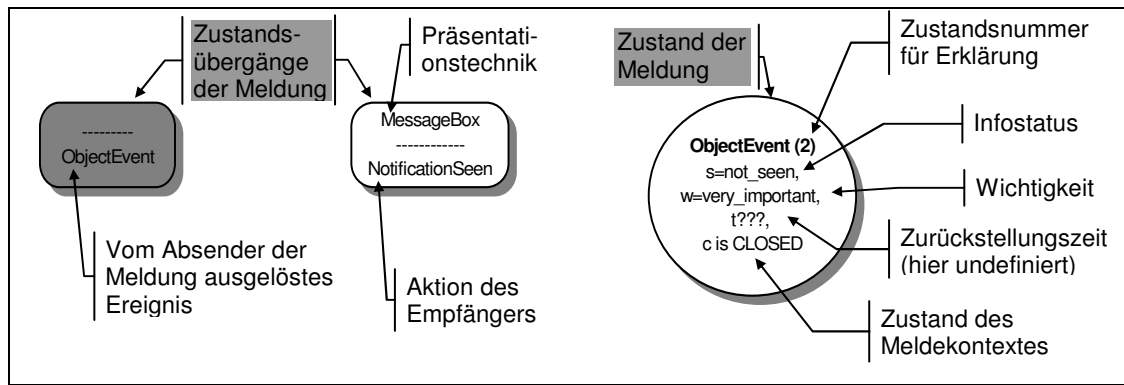
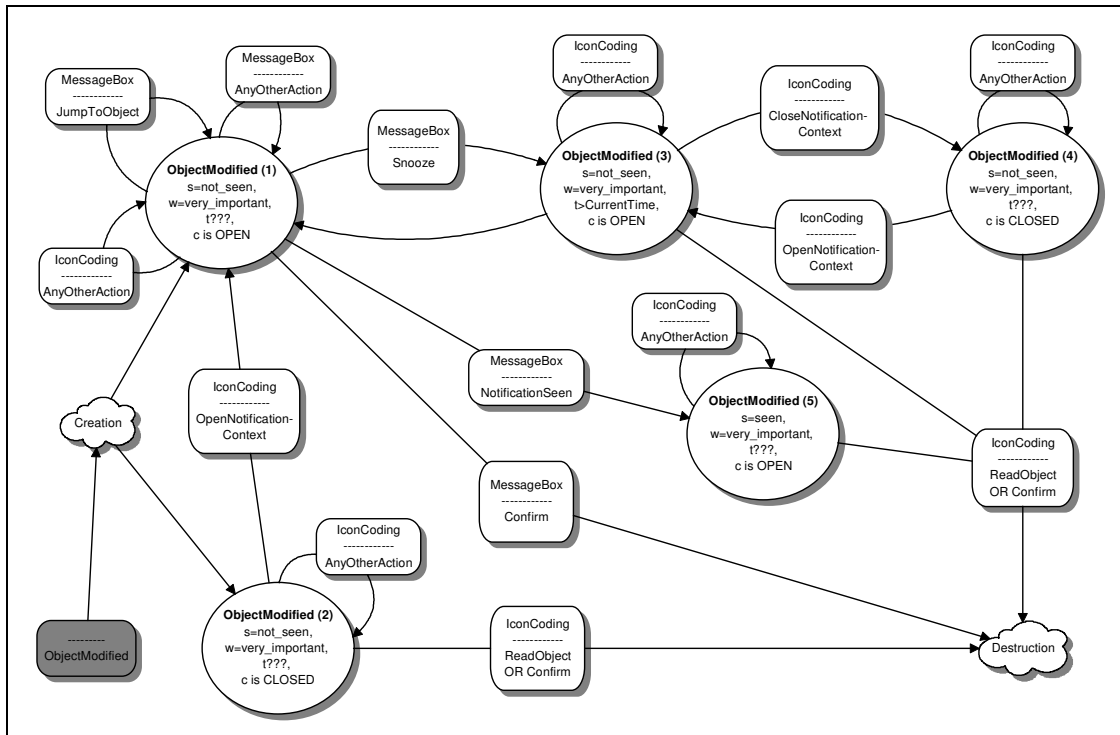


Abbildung 5.6: Legende für die folgenden Abbildungen

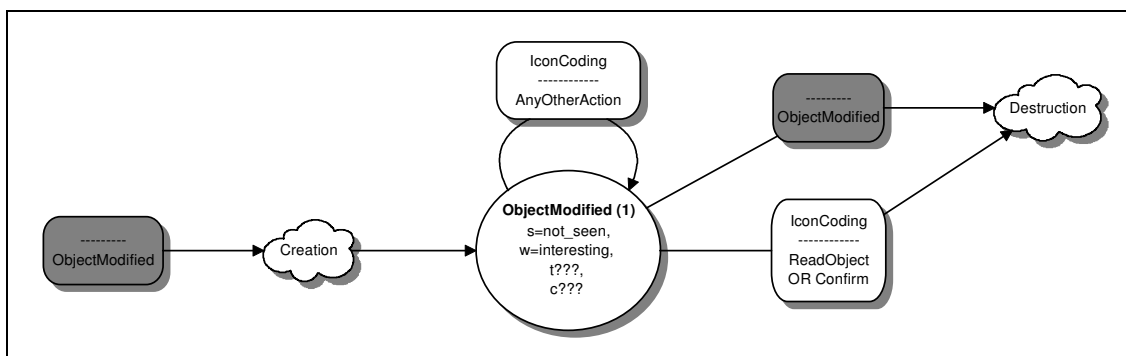
Abbildung 5.7 zeigt als Beispiel den Lebenszyklus der Meldung: "Ein Objekt wurde geändert". Die Meldung entsteht, wenn ein Benutzer das entsprechende Objekt modifiziert. Angenommen, ein Benutzer hat im Interessemuster für das fragliche Objekt als Wichtigkeit „sehr wichtig“ angegeben. Das System erzeugt dann für diesen Benutzer eine Meldung über das Ereignis mit dieser initialen Wichtigkeit. In Abhängigkeit vom Zustand des Meldekontextes (geöffnet oder geschlossen) geht die Meldung sofort in einen der Zustände (1)<sup>1</sup> oder (2) über. Entsprechend wird die Präsentationstechnik ausgewählt. Aus dem Zustand (2) kann die Meldung durch Öffnen des Meldekontextes in den Zustand (1) übergehen. Im Zustand (1) erfolgt die Darstellung in einer Meldebox, in allen anderen Zuständen im Objekticon. Die Meldebox macht u.a. die Aktion *JumpToObject* (Ereignisobjekt anzeigen) verfügbar. Weiterhin gibt es die Möglichkeit, die Meldung zurückzustellen (*Snooze*). Damit geht die Meldung in den Zustand (3) über. Sie wird jetzt nicht mehr in der Meldebox dargestellt. Nach Ablauf der Zurückstellungszeit erfolgt ein Übergang in den Zustand (1) und damit eine erneute Darstellung in der Meldebox. Wird im Zustand (3) der Meldekontext geschlossen, so wird die Zurückstellung der Meldung bis zum erneuten Öffnen des Meldekontextes verzögert. Schließlich kann der Benutzer die Meldung in der Meldebox als „gesehen“ markieren (*NotificationSeen*), worauf sie in den Zustand (5) übergeht. Aus allen Zuständen kann die Meldung entweder durch Lesen des geänderten Objektes oder durch Quittieren der Meldung irrelevant und somit gelöscht werden.

<sup>1</sup> Die Ziffer in Klammern bezeichnet die Zustandsnummer in der entsprechenden Abbildung, also hier in Abbildung 5.7.



**Abbildung 5.7:** Lebenszyklus einer Meldung über das Ereignis „Objekt geändert“ mit initialer Wichtigkeit „Sehr wichtig“

Das Ausdünnen von Meldungen<sup>1</sup> als ein weiterer Grund, die Meldung zu löschen, wurde aus Gründen der Übersichtlichkeit nicht mit ins Diagramm aufgenommen.

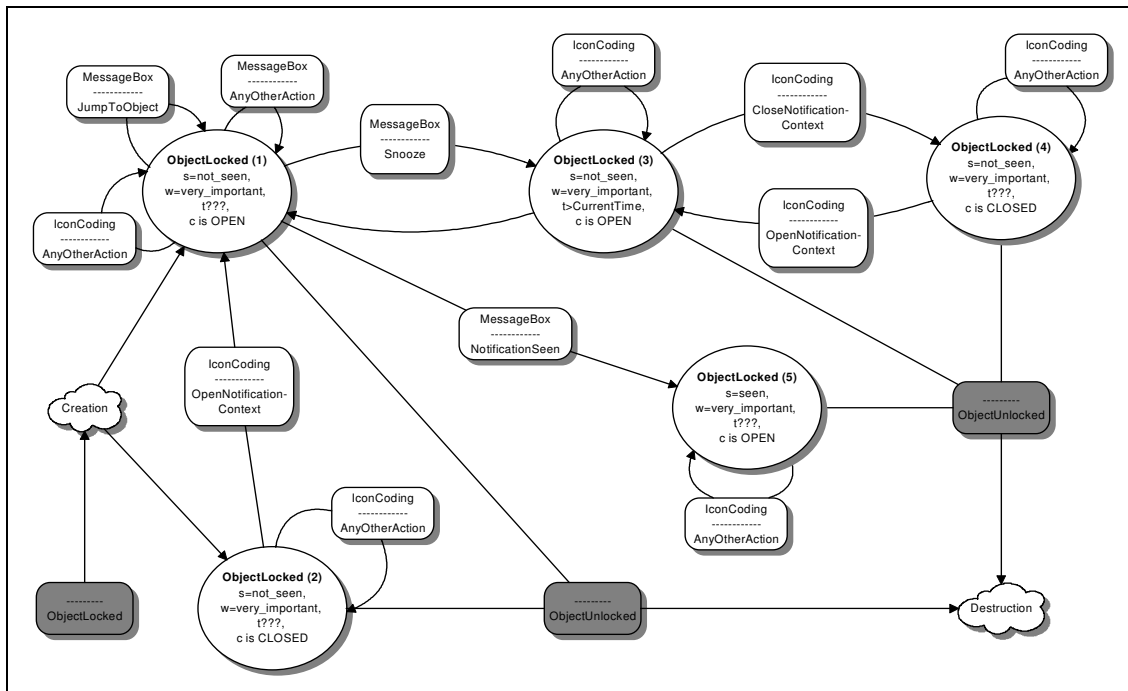


**Abbildung 5.8:** Lebenszyklus einer Meldung über das Ereignis „Objekt geändert“ mit initialer Wichtigkeit „interessant“

In Abbildung 5.8 wird der Lebenszyklus derselben Meldung mit der initialen Wichtigkeit „interessant“ dargestellt. Er ist bedeutend einfacher, denn es gibt lediglich einen Zustand. Das Bestätigen der Meldung (entweder implizit durch Lesen des Objektes oder explizit durch Quittieren der Meldung) ist die einzige verfügbare Operation. Wird während des Lebenszyklus der Meldung das Objekt erneut geändert (tritt also ein Ereignis desselben Typs für dasselbe Objekt während des Lebenszyklus einer Meldung

<sup>1</sup> Abbildung 5.8 enthält den Zustandsübergang für das Ausdünnen von Meldungen.

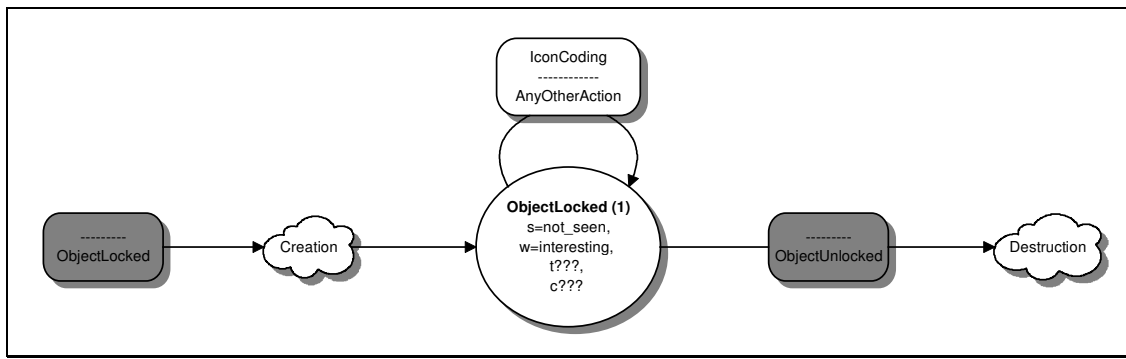
nochmals auf), so wird die Meldung über das alte Ereignis gelöscht und eine Meldung über das neue Ereignis erzeugt. Wie bereits dargelegt, wird diese Technik als *Ausdünnen (Thinning)* bezeichnet.



**Abbildung 5.9:** Lebenszyklus einer Meldung über das Ereignis „Objekt gesperrt“ mit initialer Wichtigkeit „sehr wichtig“

Soll eine Meldung über eine *Aktivität* generiert werden, so wird sie beim Auftreten des Startereignisses erzeugt und beim Auftreten des Endereignisses gelöscht. Abbildung 5.9 veranschaulicht das Prinzip. Auffällig ist hier im Unterschied zu Abbildung 5.7, daß eine Meldung über eine Aktivität nicht quittiert werden kann. Das ist sinnvoll, da Aktivitäten Prozesse mit einem definierten Ende sind, die parallel zu den Aktivitäten des Meldungsempfängers ablaufen. Meldungen über Aktivitäten werden durch das Auftreten des Endereignisses der Aktivität quittiert. Auch ein Ausdünnen ist bei Aktivitätsmeldungen nicht sinnvoll, da dieselbe Aktivität nicht zweimal hintereinander starten bzw. enden kann.

Analog zu Abbildung 5.8 (mit den oben dargestellten Unterschieden) hat eine Aktivitätsmeldung nur einen möglichen Zustand, ihre Zerstörung wird einzig durch das Endereignis der Aktivität initiiert. Abbildung 5.10 stellt den Lebenszyklus einer Aktivitätsmeldung mit initialer Wichtigkeit „interessant“ dar.



**Abbildung 5.10:** Lebenszyklus einer Meldung über das Ereignis „Objekt gesperrt“ mit initialer Wichtigkeit „interessant“

## f) Zwei Möglichkeiten zum Abspeichern von Meldungen

### 1) Benutzerspezifische Objektattribute

Zum Abspeichern der Meldungen bietet sich gemäß dem objektorientierten Ansatz des Systems LinkWorks die Darstellung als Objektattribute an.

Eine Meldung über ein Ereignis eines bestimmten Typs wird in diesem Fall je Benutzer mit Verweis auf das Ereignis als Attribut des Ereignisobjektes gespeichert. Dieses Attribut kann also *je Benutzer einen anderen Wert aufweisen*. Es ist also eigentlich eine Tabelle von Meldungen über dasselbe Ereignis mit dem Empfänger als Schlüssel, von der jeder Benutzer nur eine Zeile sieht, nämlich diejenige mit der an ihn gerichteten Meldung.

Es gibt Ereignistypen, die sich wechselseitig ausschließen. Hier sind zum einen das Start- und Endereignis einer Aktivität zu nennen, zum anderen das Herausnehmen und Hineinlegen eines Objektes in ein Containerobjekt. Solche Ereignistypen können zu einer *Ereignisgruppe* zusammengefaßt werden. Eine Ereignisgruppe kann auch ein einzelner Ereignistyp sein, für den es keinen anderen Ereignistypen gibt, der ihn ausschließt.

Ein Objekt muß je Ereignisgruppe zwei Attribute aufweisen:

- ein Ereignisattribut und
- ein Meldungsattribut, das je Benutzer unterschiedliche Werte enthalten kann.

Bei diesem Vorgehen gibt es zwei Sonderfälle: das Löschen eines Objektes und das Herausnehmen eines Objektes aus einem Containerobjekt. In beiden Fällen hört das Objekt im entsprechenden Elternobjekt auf zu existieren. Da Ereignisse und Meldungen jedoch objektbezogen gespeichert werden, gibt es ein Problem: Wie kann z.B. das Löschereignis als Attribut eines Objektes gespeichert werden, das nicht mehr existiert? Die Lösung ist hier die Erzeugung eines sogenannten *Ghost-Objektes*, das den Platz des nicht mehr existierenden Objektes in der Objektbasis einnimmt und so lange existent

bleibt, bis die Lösch- oder Verschiebemeldung für alle Benutzer irrelevant geworden ist, also ihren Lebenszyklus für alle Benutzer beendet hat.

Die Einführung von Ghost-Objekten führt zu einem weiteren potentiellen Problem: Ein aus einem Containerobjekt herausgenommenes Objekt kann irgendwann auch wieder in dieses hineingelegt werden. Es würde dann neben seinem „Geist“ im Container existieren, was verwirrend wäre. Daher kann das Paar „Objekt herausnehmen - selbes Objekt wieder hineinlegen“ nicht einfach als Ereignisgruppe modelliert werden. Das Containerobjekt muß statt dessen die Konsistenz sicherstellen, indem es beim erneuten Hineinlegen eines herausgenommenen Objektes als Seiteneffekt der Operation Annullieren das zugehörige Ghost-Objekt löscht.

**Vorteile** der Speicherung von Meldungen als Objektattribute sind:

- die Kapselung zusammengehöriger Daten
- die Integration der Meldungen direkt in das betroffene Objekt
- die einfache Konsistenzsicherung zwischen einem Objekt und den dazugehörigen Meldungen
- die Entkopplung der Sichten der Empfänger auf die Meldungen
- die einfache Aktualisierung der Repräsentation eines Objektes beim Eintreffen neuer Meldungen (bei Nutzung eines Message Protocols)

Dem stehen folgende **Nachteile** gegenüber:

- die gesonderte Behandlung der Ghost-Objekte
- die notwendige Konsistenzsicherung beim erneuten Hineinlegen herausgenommener Objekte
- der erhöhte Aufwand beim Erstellen einer Meldungsübersicht über alle Objekte im Container (mehrere Zugriffe auf die Objektbasis)

## 2) Tabellen

Die zweite Möglichkeit ist das Speichern der Ereignisse und Ereignismeldungen in Tabellenform. Je Containerobjekt gibt es eine Ereignistabelle und eine Meldungstabelle. Erstere speichert alle Ereignisse, die im Container auftreten, letztere die existierenden Meldungen. Die Ereigniseinträge in den Meldungen sind Verweise in die Ereignistabelle. Folgende Operationen können auf der Meldungstabelle ausgeführt werden:

- Meldung einfügen
- Meldung quittieren (löschen)
- Meldung zurückstellen

- Meldung als gesehen markieren
- Meldungen ausdünnen
- Meldungskombination annullieren

Diese Speicherungsform hat folgende **Vorteile**:

- konsistente Behandlung der Ereignismeldungen (keine Ghost-Objekte)
- keine Konsistenzsicherung notwendig beim erneuten Hineinlegen herausgenommener Objekte
- einfaches Zusammenstellen der Meldungsübersicht über alle Objekte im Container in nur einem Zugriff auf die Objektbasis

Dem stehen die folgende **Nachteile** gegenüber:

- keine Kapselung zusammengehöriger Daten
- Konsistenzsicherung zwischen einem Objekt und den dazugehörigen Meldungen notwendig
- Aktualisierung der Repräsentation eines Objektes beim Eintreffen neuer Meldungen schwieriger

### g) Probleme beim Ausdünnen von Meldungen

Beim Ausdünnen von Ereignismeldungen kann ein Problem auftreten, wenn man das Angeben von unterschiedlichem Interesse an Ereignissen desselben Typs für unterschiedliche Benutzer bezogen auf dasselbe Objekt zulässt. Angenommen, ein Benutzer spezifiziert für ein bestimmtes Objekt und einen bestimmten Ereignistyp unterschiedliches Interesse in Abhängigkeit vom Ereignisinitiator. Wird das Ereignis jetzt von einem Benutzer ausgelöst, für den eine hohe Wichtigkeit im Interessemuster angegeben wurde, führt das zu einer Meldung mit hoher Wichtigkeit. Wenn danach ein Ereignis desselben Typs für dasselbe Objekt von einem anderen Benutzer ausgelöst wird, generiert das System u.U. eine Meldung mit geringer Wichtigkeit. Es gibt jetzt drei Möglichkeiten, die beiden Ereignismeldungen auszudünnen:

1. Die Meldung über das ältere Ereignis wird gelöscht.  
**Problem:** Dem Empfänger wird die wichtigere Meldung vorenthalten.
2. Die unwichtigere Meldung wird gelöscht.  
**Problem:** Dem Empfänger wird die aktuellere Meldung vorenthalten.
3. Keine der beiden Meldungen wird gelöscht.  
**Problem:** Es besteht die Gefahr von Informationsüberflutung.

Dieser Konflikt kann vermieden werden, wenn das Interesse an Objekteignissen für dieselbe Objekt-Elternobjekt-Kombination und denselben Ereignistyp für alle Ereignisinitiatoren gleich ist. Bei zwei Ereignistypen besteht o.g. Problem nicht: „Objekt erzeugen“ und „Objekt in ein Containerobjekt hineinlegen“. Je Objekt tritt das Ereignis des Erzeugens nur einmal auf und muß daher nicht ausgedünnt werden. Das Hineinlegen desselben Objektes in ein Containerobjekt kann zwar mehrmals hintereinander durch verschiedene Benutzer erfolgen, jedoch bedingt ein erneutes Hineinlegen das vorherige Herausnehmen. Dieses annulliert das vorangegangene Hineinlegen, so daß auch hier der Konflikt vermieden wird. Daher muß gefordert werden, daß für alle Ereignistypen außer den beiden genannten das Interesse für dieselbe Objekt-Elternobjekt-Kombination und denselben Ereignistyp für alle Ereignisinitiatoren gleich ist.

## 5.3 Erzeugen und Propagieren einer Meldung

### 5.3.1 Erzeugen und Verteilen einer Meldung

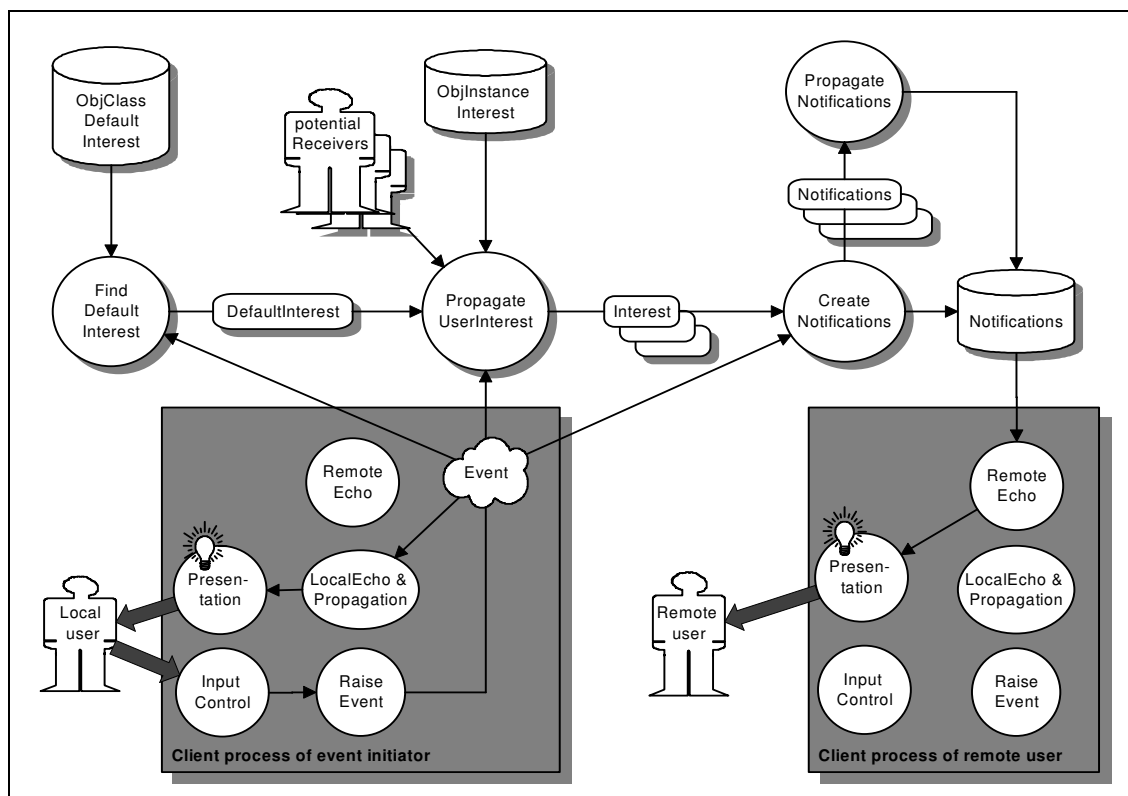


Abbildung 5.11: Erzeugen und Verteilen einer Meldung

Wenn ein Ereignis eingetreten ist, wird für jeden Benutzer im gemeinsamen Arbeitsbereich eine Meldung über dieses Ereignis erzeugt. Abbildung 5.11 veranschaulicht den Datenfluß bei der Erzeugung einer Meldung über ein Ereignis mit einem Ereignisinitiator und einem entfernten Meldungskonsumenten.

Zur Trennung von lokalem und entferntem Echo wird die Meldung für den Ereignis-initiator von der Funktion *LocalEcho&Propagation* sofort lokal propagiert und an die Präsentationskomponente weitergereicht, die sie als lokales Echo präsentiert. Das Wegfallen des Umweges über das Netzwerk in die Objektbasis und wieder zurück zum lokalen Benutzer garantiert eine gemäß den Anforderungen für direktmanipulative Benutzeroberflächen<sup>1</sup> kurze Antwortzeit für den Benutzer, der das Ereignis ausgelöst hat.

Danach erfolgt die Generierung der persistent zu speichernden Meldungen. Zunächst stellt die Funktion *PropagateUserInterest* das Interesse an dem Ereignis für jeden potentiellen Empfänger der Meldung fest. Potentielle Empfänger sind alle Benutzer, die Zugriff auf den Arbeitsbereich haben. Dazu wird entweder auf explizites bzw. implizites Interesse für den Ereignistyp und die Kombination Ereignisobjekt-Elternobjekt zugegriffen oder das Interesse durch Propagieren in der Objekthierarchie ermittelt. Default-Werte werden in der Funktion *FindDefaultInterest* ermittelt und gemäß dem einschränkenden oder dem ergänzenden Ansatz<sup>2</sup> eingesetzt. Die Funktion *Create-Notifications* übernimmt die Werte für Meldekontext und Wichtigkeit aus dem resultierenden Interesseintrag in die Meldung und speichert diese mit Verweis auf das Ereignis in der Objektbasis. Falls notwendig, werden Meldungen ausgedünnt bzw. annulliert. Zusätzlich wird die Meldung durch die Funktion *PropagateNotifications* in der Objekthierarchie in verbundene Kontexte weitergeleitet, die propagierten Meldungen werden ebenfalls in der Objektbasis gespeichert. Um bei der Präsentation zwischen primären und propagierten Meldungen unterscheiden zu können, wird bei propagierten Meldungen im Attribut *Wichtigkeit* der Wert „propagiert“ eingetragen.

Entfernte Benutzer greifen auf die Meldungen in der Objektbasis zu. Das kann entweder durch *Polling* oder durch *Messaging* geschehen. Beim *Polling* fragt jeder Clientprozeß in regelmäßigen Zeitintervallen die Objektbasis ab, ob neue Meldungen vorliegen. Beim *Messaging* informiert die Objektbasis alle betroffenen Clientprozesse beim Vorliegen neuer Meldungen. Letzteres Verfahren ist günstiger, da die Netzlast geringer ist und die Clientprozesse sofort benachrichtigt werden, wenn eine neue Meldung ansteht. Es wird jedoch nicht von allen Systemen unterstützt.

Ermittelte neue Meldungen werden dann von der Funktion *RemoteEcho* an die Präsentationskomponente weitergeleitet.

Wenn sich ein Benutzer neu im System einloggt, werden alle an diesen Benutzer gerichteten Meldungen aus der Objektbasis zum Clientprozeß übertragen und von der Funktion *RemoteEcho* an die Präsentationskomponente weitergereicht.

---

<sup>1</sup> Vgl. [SHN92]

<sup>2</sup> Diese beiden Ansätze wurden im Abschnitt 5.2.2 (c)) eingeführt.

### 5.3.2 Propagieren und Neupropagieren von Meldungen

#### a) Problematik

Propagieren ist das Weiterleiten von Meldungen zum Zeitpunkt ihrer Entstehung aus dem Entstehungskontext in übergeordnete Kontexte. Dort kann eine Zusammenfassung präsentiert werden, um einen Überblick über die Meldungen in untergeordneten Kontexten zu ermöglichen. Das erhält den Zusammenhang zwischen Kontexten für den Benutzer und hilft ihm, aktuelle Ereignismeldungen in der Containerhierarchie schnell aufzufinden.

Meldungen über Aktivitäten werden nicht propagiert, da sie nur für das betroffene Objekt relevant sind<sup>1</sup> und eine Propagierung die Überwachung der Aktivitäten von Benutzern unterstützen würde.

Propagiert man Meldungen nach obigem Ansatz, führt das zu einem großen Datenaufkommen und zu Informationsüberflutung. Eine untere<sup>2</sup> Abschätzung für die Maximalanzahl  $Z_x$  der zu einem Container  $X$  propagierten Meldungen läßt sich nach folgender Formel berechnen:

$$Z_x = N_u \cdot (2 \cdot N_{cx} \cdot N_{ox} + (N_{yo} - 2) \cdot N_{ox} + (N_{yc} - 2) \cdot N_{cx}),$$

wobei die Größen folgendes bedeuten:

- $N_u$  - Anzahl der Benutzer im Arbeitsbereich
- $N_{yo}$  - Anzahl der Ereignistypen für Dokumentobjekte, deren Meldungen propagierbar sind (6 Typen)
- $N_{yc}$  - Anzahl der Ereignistypen für Containerobjekte, deren Meldungen propagierbar sind (5 Typen)
- $N_{ox}$  - Anzahl der Dokumentobjekte in der Objekthierarchie unterhalb des Containers  $X$
- $N_{cx}$  - Anzahl der Containerobjekte in der Objekthierarchie unterhalb des Containers  $X$

Der Summand  $2 \cdot N_{cx} \cdot N_{ox}$  tritt auf, weil das Herausnehmen aus einem Container und das Hineinlegen in einen Container je Objekt und Container einmal möglich ist. Der Summand  $(N_{yo} - 2) \cdot N_{ox}$  gibt die Anzahl der Meldungen über Ereignisse wieder, die sich auf Dokumentobjekte beziehen. Dabei werden die beiden Verschiebeereignistypen nicht berücksichtigt. Die analoge Bedeutung hat  $(N_{yc} - 2) \cdot N_{cx}$  für Containerobjekte.

Geht man jetzt von einem gemeinsamen Arbeitsbereich mit 30 verschachtelten Containern, 10 Benutzern und 200 Dokumentobjekten aus, so können zum Wurzel-

<sup>1</sup> Die Information „Objekt wird zur Zeit von Benutzer XYZ bearbeitet“ ist für einen Benutzer nur relevant, wenn er dieses Objekt selbst bearbeiten möchte. Dazu hat er aber den Container geöffnet, der das Objekt enthält, sieht also die Darstellung der nicht propagierten Meldung.

<sup>2</sup> Es handelt sich um eine untere Abschätzung, weil das Verschieben von Containerobjekten nicht betrachtet wurde.

objekt bis zu 132 900 Meldungen propagiert werden! Davon sind 120 000 Meldungen über Verschiebeereignisse, 12 000 Meldungen über sonstige Ereignisse bezogen auf Dokumentobjekte und 900 Meldungen über sonstige Ereignisse bezogen auf Containerobjekte. Der Wurzelknoten der Objekthierarchie (also der Arbeitsbereich) würde demnach alle Meldungen gesendet bekommen, die irgendwo in untergeordneten Containern entstehen. Diese Meldungsflut würde sowohl das System lahmlegen als auch den Benutzer überfordern.

Folglich muß man propagierte Meldungen ausdünnen. Für einen Überblick genügt es, je Container nur die aktuellste Meldung und die Gesamtanzahl der Meldungen pro Wichtigkeitsstufe zu speichern. Das ermöglicht dem Benutzer, interaktiv die ihn interessierenden Meldungen zu verfolgen, und entlastet das System, da je Container nur eine propagierte Meldung (die aktuellste) gespeichert wird.

Dazu benötigt ein Containerobjekt folgende Attribute:

- aktuellste Meldung
- Anzahl propagierter „interessanter“ Meldungen
- Anzahl propagierter „lokal wichtiger“ Meldungen
- Anzahl propagierter „wichtiger“ Meldungen
- Anzahl propagierter „sehr wichtiger“ Meldungen

Da Meldungen einen Lebenszyklus haben, erfordert dieser Ansatz ein Neupropagieren von Meldungen immer dann, wenn eine propagierte Meldung quittiert (gelöscht) wird. Die Vorgehensweise wird in Abschnitt 5.3.2 (d)) vorgestellt.

Meldungen propagieren innerhalb der Struktur, die von den Verschiebeereignissen „Objekt herausnehmen“ und „Objekt hineinlegen“ verändert werden kann. Wenn Meldungen nach dem obigen Mechanismus propagiert werden, so ist jedoch kein Neupropagieren der Meldungen nach dem Verschieben eines Containerobjektes notwendig. Das rührt daher, daß die Meldung über das Verschiebeereignis aktueller ist als alle bisher vom verschobenen Objekt in der Hierarchie aufwärts propagierten Meldungen und sie diese folglich im Zuge des Ausdünnens ersetzt. Es müssen allerdings die Zähler propagierter Meldungen in den Containerobjekten angepaßt werden, die dem verschobenen Container vor und nach dem Verschieben übergeordnet sind.

### b) Zyklenfreiheit im Strukturgraphen als Voraussetzung

LinkWorks gestattet das Erstellen von Verweisen auf Objekte<sup>1</sup>. Neben der Möglichkeit, Verweise auf Objekte per Mail zur gemeinsamen Nutzung an andere Benutzer zu verschicken, kann man auch Verweise auf Containerobjekte erstellen und diese in andere Containerobjekte hineinlegen. Dadurch entsteht aus der Objekthierarchie ein gerichteter, nicht notwendig zyklensfreier

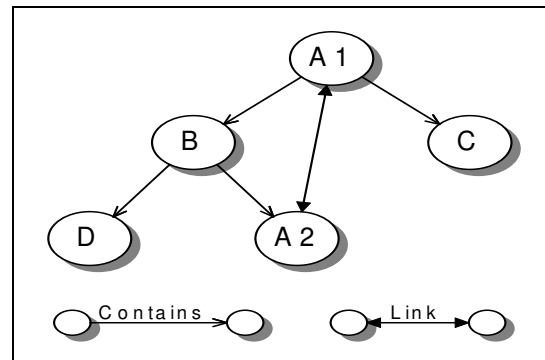


Abbildung 5.12: Nicht zyklensfreier Strukturgraph

Strukturgraph. Abbildung 5.12 zeigt einen solchen Graphen. Der Zyklus entstand durch das Hineinlegen eines Verweises auf den übergeordneten Container A in den in A enthaltenen Container B. B enthält also ein Kindobjekt, das gleichzeitig sein Elternobjekt ist.

Ein solcher Zyklus führt zu Problemen beim Propagieren von Meldungen, die dann möglicherweise in untergeordnete Containerobjekte eines Containers propagiert werden können. Die Durchschaubarkeit von Propagierungswegen wäre in diesem Fall nicht mehr gegeben. Auch für das Propagieren von Interesse ist ein solcher Zyklus problematisch. Das Problem erfordert weitere Untersuchungen, die den Rahmen der Diplomarbeit sprengen würden. Drei Lösungswege bieten sich an:

- Beim Hineinlegen eines Verweises auf ein Containerobjekt in ein anderes Containerobjekt wird der resultierende Strukturgraph auf Zyklensfreiheit geprüft. Da die Strukturänderung an der Benutzeroberfläche über Drag&Drop ausgelöst wird, müssen die entsprechenden Algorithmen zur Zyklenerkennung auch bei komplexen Strukturen schnell genug für ein interaktives Arbeiten sein.
- Beim Propagieren von Meldungen bzw. Interesse werden Algorithmen zur Zyklendetektion ausgeführt. Da die Aktionen der Benutzer ein hohes Meldungsaufkommen erzeugen, müssen auch hier entsprechend schnelle Algorithmen zum Einsatz kommen.
- Das Erzeugen von Verweisen auf Containerobjekte wird generell verboten.

In der vorliegenden Arbeit wird der dritte Ansatz gewählt. Für einen gemeinsamen Arbeitsbereich ist das nicht weiter problematisch, da ohnehin alle zugriffsberechtigten Benutzer die Gesamtheit der Objekte im Arbeitsbereich gemeinsam nutzen können.

<sup>1</sup> Vgl. Abschnitt 4.2

### c) Vorgehen beim Propagieren von Meldungen

Beim Propagieren einer neuen Meldung kann man davon ausgehen, daß die zu propagierende Meldung die aktuellste ist. Daher wird die Meldung zu jedem übergeordneten Container des Ereignisobjektes bis zum Wurzelobjekt, also dem gemeinsamen Arbeitsbereich, propagiert. Weiterhin muß bei jedem dieser Container der Meldungsähler für die Wichtigkeit der aktuellen Meldung um eins erhöht werden, falls diese nicht im Zuge des Ausdünnens eine ältere Meldung überschrieben hat.

### d) Vorgehen beim Neupropagieren von Meldungen

Das Neupropagieren von Meldungen ist etwas komplizierter. Wenn eine Meldung in einem Container quittiert wird, ist bei diesem und bei jedem in der Hierarchie übergeordneten Container der Meldungsähler für die Wichtigkeit der gelöschten Meldung um eins zu verringern.

Weiterhin muß geprüft werden, ob die quittierte Meldung die aktuellste im Container ist. In diesem Fall ist eine Neupropagierung notwendig. Dazu wird zuerst die propagierte Meldung des Containers gelöscht und nachfolgend die aktuellste der verbleibenden Meldungen im Container (einschließlich propagierter Meldungen untergeordneter Container) ermittelt. Zwei Fälle können hierbei auftreten:

- 1) Es gibt keine weitere Meldung.
- 2) Es gibt eine aktuellste Meldung.

Im ersten Fall ist im aktuellen Container nichts weiter zu tun. Die Neupropagierung setzt sich bei dessen Elternobjekt in der Containerhierarchie fort, bis das Wurzelobjekt erreicht ist.

Im zweiten Fall wird die neue aktuellste Meldung zur propagierten Meldung des Containers. Danach wird die Propagierung wie im ersten Fall fortgesetzt. In Listing 5.1 ist der oben dargestellte Algorithmus noch einmal in PASCAL-ähnlichem Pseudocode abgebildet.

```
METHOD ContainerObjClass :: RePropagateNotification
    (timeOfRemovedNotification : Time)
VAR newPropagatedNotification : Notification;
BEGIN
    IF (timeOfRemovedNotification >= m_myPropagatedNotification.m_event.time)
        THEN
            newPropagatedNotification := FindMostCurrentNotification();
            Delete(m_myPropagatedNotification());
            AddPropagatedNotification(newPropagatedNotification);
            Parent.RePropagateNotification(timeOfRemovedNotification);
        ENDIF
    END

METHOD ShrdWorkspaceClass :: RePropagateNotification
    (timeOfRemovedNotification : Time)
BEGIN
    (* Empty method to stop recursion *)
END
```

**Listing 5.1:** Pseudocode für das Neupropagieren einer Meldung

---

# 6 Darstellung von Meldungen

---

## 6.1 Allgemeines

Im vorigen Abschnitt wurde ein Modell für Meldungen entwickelt, bei dem der Benutzer durch Angeben von Interesse an Ereignissen das Erzeugen von Meldungen mit unterschiedlicher Relevanz steuern kann. Die Relevanz einer Meldung bestimmt deren Darstellung (Präsentation). Der Zweck der Präsentation einer Meldung ist es, den Benutzer über den Eintritt des der Meldung zugrundeliegenden Ereignisses zu informieren.

In den folgenden Abschnitten führt der Autor Kriterien zur Charakterisierung von Präsentationstechniken ein, entwirft verschiedene Präsentationstechniken für Ereignismeldungen und trifft eine Zuordnung zwischen der Relevanz einer Meldung und einer Technik zu ihrer Darstellung. Die Präsentationstechniken beruhen auf der fensterbezogenen zweidimensionalen grafischen Darstellung von Information. Alternative Kanäle der Maschine-Mensch-Kommunikation (wie z.B. Sound<sup>1</sup> und dreidimensionale Informationsdarstellung<sup>2</sup>) sollen hier nicht betrachtet werden.

## 6.2 Charakterisierung von Präsentationstechniken

Eine Präsentationstechnik kann man durch ihre Eigenschaften und ihre Fähigkeiten charakterisieren. Die Eigenschaften einer Präsentationstechnik bestimmen einerseits deren Fähigkeiten, müssen aber andererseits selbst bestimmten Ansprüchen des Benutzers genügen. Eigenschaften einer Präsentationstechnik sind *Aufdringlichkeit*, *Kontext*, *Erkennbarkeit* und *Detailliertheit*. Zu den Fähigkeiten zählen *Signalwirkung* und *Interaktivität*.

---

<sup>1</sup> Siehe hierzu [GAV93].

<sup>2</sup> Siehe hierzu [CAR91], [FAH93], [MAC90], [MAC91] und [ROB91].

## 6.2.1 Eigenschaften einer Präsentationstechnik

### a) Aufdringlichkeit

Die Aufdringlichkeit einer Darstellungstechnik drückt aus, wie stark die Technik den Benutzer zwingt, seine Arbeit zu unterbrechen.

- Eine Meldung wird *nicht* dargestellt.
- Eine Meldung wird *nicht-modal* dargestellt, d.h., der Benutzer wird nicht gezwungen, von der Meldung Kenntnis zu nehmen.
- Eine Meldung wird *modal* dargestellt, d.h., der Benutzer wird gezwungen, vor der Weiterarbeit von der Meldung Kenntnis zu nehmen.

Das Nichtdarstellen einer Meldung hat die niedrigste, die modale Darstellung die höchste Aufdringlichkeit.

### b) Kontext

Der Darstellungskontext kennzeichnet den *Ort* der Darstellung an der Benutzeroberfläche. Für den Darstellungskontext einer Meldung gibt es prinzipiell drei Möglichkeiten:

- Eine Meldung wird gebunden an das Ereignisobjekt, also *objektgebunden* dargestellt.
- Eine Meldung wird gebunden an ihren Entstehungskontext, also *kontextgebunden* dargestellt.
- Eine Meldung wird losgelöst vom Ereignisobjekt und vom Entstehungskontext, also *kontextübergreifend* dargestellt. Die Präsentation erfolgt hier zentral im System.

### c) Erkennbarkeit

Die Erkennbarkeit einer Präsentationstechnik bestimmt, wie gut der Mensch mit seinen Augen die Darstellung erkennen kann, wenn sie nicht auf dem Bildschirm verdeckt ist (beispielsweise durch Fenster oder Dialogboxen). Eine hohe Erkennbarkeit wird erreicht durch:

- hohe Helligkeitskontraste,
- hohe Farbkontraste,
- Dynamik.

Unter Dynamik ist das Ändern grafischer Attribute der Darstellung in Abhängigkeit von der Zeit zu verstehen. Dynamische Darstellungstechniken müssen sehr sparsam eingesetzt werden. Vor allem ihr gleichzeitiger Einsatz an mehreren Stellen auf dem Bildschirm wird als extrem irritierend empfunden, da das menschliche Auge Bewegung in einer ansonsten ruhigen Umgebung bevorzugt wahrnimmt und diese Stellen unbewußt abwechselnd fokussiert<sup>1</sup>.

#### d) Detailliertheit

Die Detailliertheit einer Präsentationstechnik sagt aus, wie viele Einzelheiten in der Darstellung erkennbar sind. Für die Darstellung von Meldungen ist hier vor allem wichtig, ob einzelne Meldungen erkenn- und damit selektierbar sind.

### 6.2.2 Fähigkeiten einer Präsentationstechnik

#### a) Signalwirkung

Die Signalwirkung einer Darstellungstechnik ist ihr Potential, die Aufmerksamkeit eines Benutzers für eine Ereignismeldung zum Zeitpunkt ihres Eintreffens zu erregen. Sie wird durch die drei Eigenschaften *Kontext*, *Aufdringlichkeit* und *Erkennbarkeit* bestimmt. Je höher Aufdringlichkeit und Erkennbarkeit, desto höher ist die Signalwirkung. Bezogen auf den Kontext ist die Signalwirkung der kontextübergreifenden Darstellung am höchsten, gefolgt von der kontextgebundenen Darstellung. Die objektgebundene Darstellung hat die geringste Signalwirkung.

#### b) Interaktivität

Die Interaktivität einer Präsentationstechnik gibt die Möglichkeiten für den Benutzer an, mit einer Meldung zu interagieren. Sie macht einige oder alle Interaktionsmöglichkeiten der Meldung für den Benutzer verfügbar. Das betrifft zum einen das *Detaillieren* der Meldung auf Anforderung. Zum anderen beinhaltet Interaktivität das *Ausführen von Operationen* auf der Meldung, die sie in einen anderen Zustand innerhalb ihres Lebenszyklus<sup>2</sup> versetzen. Eine Präsentationstechnik mit hoher Interaktivität ermöglicht damit eine sofortige Reaktion des Empfängers auf die Meldung.

Im wesentlichen wird die Interaktivität einer Präsentationstechnik durch ihre *Aufdringlichkeit* und ihre *Detailliertheit* determiniert. Weil Meldungen zur Interaktion einzeln selektierbar sein müssen, setzt hohe Interaktivität eine hohe Detailliertheit voraus. Eine detaillierte Darstellung hat einen hohen Platzbedarf. Dieser konkurriert mit dem

---

<sup>1</sup> Deshalb wird beispielsweise die Position des blinkenden Cursors auf dem Bildschirm schnell wiedergefunden.

<sup>2</sup> Vgl. Abschnitt 5.2.3 (d).

Platzbedarf für die Anzeige anderer Information. Durch hohe Aufdringlichkeit der Präsentation kann der Konflikt entschärft werden, da eine aufdringliche Darstellungstechnik die Arbeit des Benutzers zur Anzeige einer Meldung unterbricht. Die zur Erledigung der unterbrochenen Aufgabe notwendige Information wird während der Zeit der Unterbrechung nicht benötigt, so daß sie weggeblendet werden kann. Der freiwerdende Platz ist jetzt für eine detaillierte Darstellung nutzbar. Somit wird die für eine hohe Interaktivität notwendige Detailliertheit durch hohe Aufdringlichkeit erkaufte.

### 6.2.3 Die Dringlichkeit der Präsentation

Die ideale Präsentationstechnik bewahrt den Bezug zum Ereignisobjekt. Sie hat eine hohe Signalwirkung bei niedriger Aufdringlichkeit sowie eine hohe Interaktivität. Leider gibt es diese ideale Technik nicht. Eine hohe Signalwirkung und Interaktivität kann - wie oben dargestellt - nur durch hohe Aufdringlichkeit erkaufte werden. Die kontextübergreifende Präsentation hat eine höhere Signalwirkung als die kontext- oder objektbezogene Darstellung, da sie die Information zentral darstellt, während letztere an möglicherweise verdeckte Objekte bzw. Kontexte gebunden ist. Zentrale Informationsdarstellung stellt die Meldungen selbst als Objekte dar und erlaubt somit aufgrund der höheren Detailliertheit eine höhere Interaktivität.

Das Ziel des Benutzers bei der Nutzung des Benachrichtigungs- und Informationsdienstes ist das Erfassen von Ereignismeldungen zur Unterstützung der eigenen Arbeit. Der Grad des Interesses eines Benutzers an einer Meldung bestimmt dabei seine Erwartungen bezüglich Signalwirkung und Interaktivität, seine Toleranzgrenze für die Aufdringlichkeit der Technik sowie seine Anforderungen an die Detailliertheit und die Darstellung der Meldung in anderen Kontexten als ihrem Entstehungskontext. Bei geringem Interesse wird Wert auf eine unaufdringliche Präsentation im Entstehungskontext gelegt, auch auf Kosten von Detailliertheit, Signalwirkung und Interaktivität. Umgekehrt wird bei hohem Interesse eine detaillierte, kontextübergreifende Darstellung gefordert und die Aufdringlichkeit der Technik eher toleriert, wenn sie eine hohe Signalwirkung und Interaktivität aufweist. Die Erkennbarkeit der Darstellung muß stets sichergestellt sein.

Eine Präsentationstechnik ist also ein Kompromiß zwischen Signalwirkung und Interaktivität einerseits sowie Aufdringlichkeit andererseits. Zur Charakterisierung einer Präsentationstechnik bezüglich ihres Potentials zur Darstellung von Meldungen entsprechend deren Wichtigkeit soll der Begriff ihrer *Dringlichkeit* eingeführt werden.

- Eine Präsentationstechnik mit hoher Dringlichkeit ist kontextübergreifend und gekennzeichnet durch hohe Detailliertheit, Signalwirkung und Interaktivität bei hoher Aufdringlichkeit.

- Eine Präsentationstechnik mit geringer Dringlichkeit ist objektgebunden und hat eine niedrige Aufdringlichkeit, auch auf Kosten geringerer Detailliertheit, Signalwirkung und Interaktivität.

## 6.3 Präsentationstechniken für Meldungen

Im vorangegangenen Abschnitt wurde die Dringlichkeit der Präsentation als Kriterium zur Charakterisierung von Präsentationstechniken eingeführt. Nun stellt der Autor fünf Präsentationstechniken vor, die er unter Berücksichtigung abgestufter Dringlichkeit entworfen hat. Die Techniken stellen eine Erweiterung der konventionellen Desktop-Metapher um die Darstellung von Awarenessinformation dar.

### 6.3.1 Technik 1: Keine Präsentation

Hierbei handelt es sich um die unaufdringlichste Darstellungstechnik ohne Signalwirkung, Kontext und Interaktivität. Eine Meldung wird zwar erzeugt, das System stellt sie jedoch nicht an der Benutzeroberfläche dar. Die Anzeige der Meldung erfolgt erst auf Nachfrage des Benutzers.

### 6.3.2 Technik 2: Objektgebundene Präsentation

#### a) Prinzip

Die objektgebundene Präsentation von Ereignismeldungen basiert auf der Darstellung der Attribute des gemeldeten Ereignisses<sup>1</sup> in Form von grafischen Attributen des Icons, welches das Ereignisobjekt bzw. das Propagierungsziel an der Benutzeroberfläche repräsentiert. Durch *Codierung* werden die in der Meldung enthaltenen nicht-grafischen Daten in grafische Daten umgewandelt.

#### b) Charakterisierung

Bei der objektgebundenen Präsentation handelt es sich um eine sehr unaufdringliche Technik zur Ereignisdarstellung. Da die Information in den Icons der bearbeiteten Objekte dargestellt wird, befindet sie sich im Blickfeld des Benutzers und kann von ihm „nebenbei“ aufgenommen werden, also ohne die Arbeit zu unterbrechen und den Blick auf ein Informationsfenster o.ä. zu richten. Das Wahrnehmen einer Ereignismeldung bedingt jedoch, daß sich das Icon im sichtbaren Bereich eines geöffneten, unverdeckten Fensters befindet. Daher ist die Signalwirkung der Technik nicht besonders hoch.

---

<sup>1</sup> Ereignisattribute sind z.B. Ereignistyp oder Ereigniszeitpunkt.

Die Technik eignet sich sehr gut dazu, bisher aufgetretene Ereignisse und parallel zur eigenen Tätigkeit verlaufende Aktivitäten anderer Benutzer den entsprechenden Objekten zuzuordnen. Allerdings ist die Interaktivität der Darstellungstechnik sehr begrenzt - lediglich das Quittieren aller sich direkt auf das Objekt beziehenden Meldungen ist mit vertretbarem Interaktionsaufwand des Benutzers möglich. Daher muß eine Möglichkeit bestehen, auf Benutzerinitiative eine Technik mit höherer Interaktivität aufzurufen.

### c) Darzustellende Information

Meldungen enthalten strukturierte Information über Ereignisse, die dem Benutzer dargestellt werden soll. Ein Objekticon verfügt jedoch nur über limitierten Platz zur Darstellung. Daher können nicht alle Attribute eines gemeldeten Ereignisses präsentiert werden. Tabelle 6.1 priorisiert die Er-

Attribut	Fragen des Benutzers <sup>1</sup>	Priorität
Typ	WAS ist passiert?	hoch
Zeitpunkt	WANN ist es passiert?	hoch
Objekt	WO ist es passiert?	hoch
Benutzer	WER hat es getan?	niedrig bis hoch
Parameter	Welche weitere Information gibt es?	niedrig

reignisattribute bezüglich ihres Potentials zur Unterstützung kooperativer Arbeit im gemeinsamen Arbeitsbereich. Ereignisattribute mit hoher Priorität sollten möglichst sofort aus der Icondarstellung erfaßbar sein, Attribute mit niedriger Priorität auf Nachfrage angezeigt werden. Die drei Attribute Ereignistyp, Ereigniszeitpunkt und Ereignisobjekt sind notwendig für das Erfassen der Aktionen von Kooperationspartnern und müssen daher auf jeden Fall dargestellt werden. Der Ereignisinitiator muß nur bei Aktivitätsereignissen unmittelbar erfaßbar sein, anderenfalls genügt es, diese Information auf Nachfrage anzuzeigen. Die Parameter des Ereignisses schließlich brauchen nicht unbedingt direkt aus der Icondarstellung ersichtlich zu sein, da sie zusätzliche Information darstellen. Sie können auf Nachfrage angezeigt werden.

**Tabelle 6.1:** Priorität der Ereignisattribute

Zusätzlich zu den Ereignistypen müssen propagierte Meldungen dargestellt werden, um ihre Verfolgung in der Containerhierarchie zu ermöglichen. Eine gesonderte Darstellung des Ereignisobjektes entfällt bei der objektgebundenen Präsentation, da es durch das Objekticon repräsentiert wird.

Verschiedene Ereignistypen haben eine ähnliche Semantik und können gleich dargestellt werden. Das betrifft die Ereignisse „Objekt erzeugen“ und „Objekt in ein Containerobjekt hineinlegen“. Beide begründen die Existenz eines neuen Objektes im entsprechenden Container.

<sup>1</sup> Vgl. Tabelle 5.1

#### d) Grafikalphabet der LinkWorks-Icons

In Farbtafel 6.1<sup>1</sup> wird das Grafikalphabet der Icons im existierenden System LinkWorks dargestellt. Ein Icon besteht aus dem *Rahmen*, dem *Hintergrund* und dem *Sinnbild*. Die Form des Rahmens zeigt an, von welcher der Basisklassen *Werkzeug*, *Container* bzw. *Dokument* die Klasse des Objektes abstammt. Das Sinnbild repräsentiert die Klasse des Objektes. Die Farbe des Iconhintergrundes zeigt den lokalen Bearbeitungszustand an. Das kann einer der drei Zustände „normal“, „selektiert“ und „geöffnet“ sein. Bei Objekten im Zustand „geöffnet“ werden neben der dunkelgelben Darstellung des Iconhintergrundes auch die im Normalzustand hellen farbigen Bereiche des Sinnbildes in einem dunkleren Ton derselben Farbe angezeigt. Weiterhin können grafische *Subicons* an das Icon angefügt werden, die in LinkWorks das Vorhandensein eines Objektlaufes (verzweigte Linien), Interesse an Änderungen (Ausrufezeichen) bzw. die Fixierung der Iconposition (Stecknadelsymbol) anzeigen. Unter dem Icon befindet sich der *Icontext*, dessen Kombination von Vordergrund- und Hintergrundfarbe angibt, ob sich der Cursor auf dem Objekt befindet und ob das Objekt ein Verweis ist.

Dieses Grafikalphabet soll jetzt derart erweitert werden, daß Ereignismeldungen im Icon dargestellt werden können.

#### e) Erweiterung des Grafikalphabetes

Bei der Erweiterung des Grafikalphabetes kommt es darauf an, daß gemäß dem Prinzip der Erwartungskonformität für den Benutzer das gewohnte Look & Feel der LinkWorks-Icons erhalten bleibt. Welche Möglichkeiten zur Erweiterung bestehen nach dieser Einschränkung?

Zunächst kann man Information in die Farbe des Sinnbildes codieren, da für die Erkennung der Objektklasse die Form des Sinnbildes ausreichend ist. Damit bei der Darstellung farbcodierter Icons deren ursprüngliches Erscheinungsbild erhalten bleibt, werden vier Farben für das Sinnbild benötigt:

- weiß zur Darstellung weißer Bereiche (z.B. Lichter), die nicht farbcodiert werden
- schwarz zur Darstellung schwarzer Bereiche (z.B. Linien und Schatten), die nicht farbcodiert werden
- ein heller Ton der Codierungsfarbe für die Darstellung heller farbiger Bereiche
- ein dunkler Ton der Codierungsfarbe für die Darstellung dunkler farbiger Bereiche

---

<sup>1</sup> Die Farbtafeln befinden sich aus drucktechnischen Gründen am Abschnittsende ab Seite **Fehler!**  
**Textmarke nicht definiert.**

Eine Modifikation der Form des Sinnbildes kann ebenfalls Information kommunizieren. Die Objektklasse muß trotz der Modifikation weiterhin aus der Form des Sinnbildes erkennbar sein. Beispielsweise könnte man das Sinnbild „Posteingangskorb“ leer bei fehlender oder mit einem Stapel von Briefumschlägen bei neu eingetretener Mail darstellen.

Da im gemeinsamen Arbeitsbereich Verweise auf Objekte nicht zugelassen sind, wird die Farbe des Icontextes für die Codierung anderer Information nutzbar. Zur Darstellung des Cursors werden wie bisher Vordergrund- und Hintergrundfarbe des Icontextes invertiert.

Weitere Möglichkeiten der Informationsdarstellung sind die Überlagerung eines weiteren Sinnbildes über das Icon und eine farbige Umrahmung des Icons (siehe Farbtabelle 6.2). Um bei Überlagerungen die Übersichtlichkeit zu wahren, darf ein Icon maximal eine Überlagerung aufweisen. Auch die Einführung weiterer Subicons und eines zweiten Icontextes oberhalb des Icons sind möglich.

Diese statischen Darstellungsmöglichkeiten können durch dynamische ergänzt werden. Dabei wird eines der o.g. Attribute des Icons ohne Änderung der dargestellten Information in Abhängigkeit von der Zeit modifiziert. Das kann von einfachem Blinken bis zu kompletten Animationen in den Icons<sup>1</sup> reichen. Eine Anwendung der dynamischen Darstellung ist das Erregen von Aufmerksamkeit für neue Ereignismeldungen. Statt lediglich das entsprechende grafische Attribut des Icons zu ändern, kann für ein kurzes Zeitintervall die Änderung blinken, bevor sie statisch dargestellt wird. Diese Technik ist sowohl für synchron zur eigenen Arbeit eintreffende Meldungen als auch für die Darstellung asynchron eingetretener Meldungen beim erneuten Öffnen eines Containers möglich.

Zusammengefaßt kann das LinkWorks-Grafikalphabet zur Darstellung von Ereignismeldungen um die folgenden Attribute erweitert werden:

- Farbe des Sinnbildes
- Formveränderung des Sinnbildes bei Beibehaltung der Grundform
- Umrahmung von Icons
- Überlagerung eines weiteren Sinnbildes
- Weitere Subicons

---

<sup>1</sup> Vgl. [BAE91].

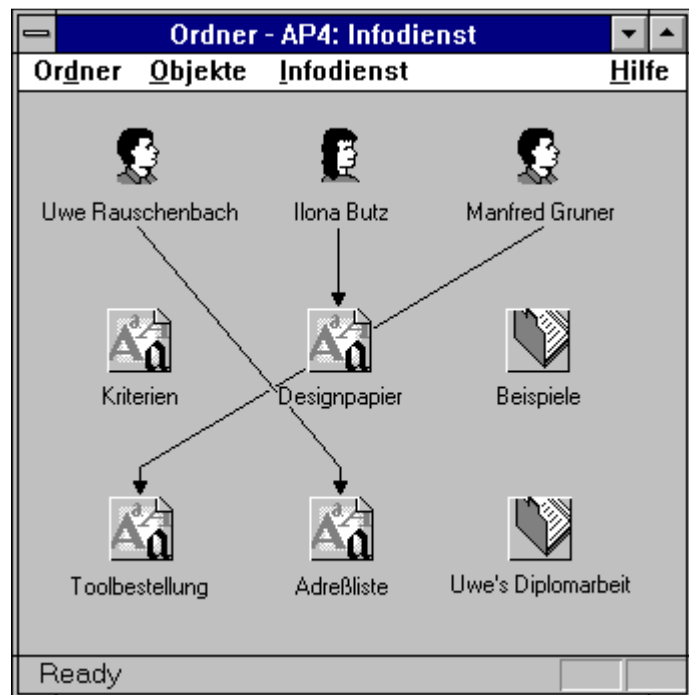
- Zweiter Iconcontext oberhalb des Icons
- Textfarbe
- Dynamische Veränderung eines oder mehrerer o.g. Attribute

### f) Objektgebundene Darstellung der Objekt-Benutzer-Beziehung

Ein Objekt ereignis ist immer eine Beziehung zwischen einem Benutzer (dem Ereignisinitiator) und einem Objekt (dessen Zustand das Ereignis ändert). Der Autor untersuchte die im System DIVA<sup>1</sup> verwendete Graphdarstellung auf ihre Eignung zur Anzeige von Beziehungen im Kontext eines gemeinsamen Arbeitsbereiches und entwickelte eine Alternative. Diese beiden Möglichkeiten der Darstellung werden im folgenden vorgestellt und gewertet.

#### 1) Darstellung als Graph

Die Darstellung als Graph setzt voraus, daß Objekte und Benutzer als Icons an der Benutzeroberfläche dargestellt werden. Eine Beziehung wird durch Verbindungslinien zwischen den Objekten und den Benutzern dargestellt. In die Linienattribute können Attribute der Beziehung codiert werden. Mit dieser Technik wird im System DIVA dargestellt, welcher Benutzer gerade welches Objekt bearbeitet. Der Vorteil dieser Darstellungstechnik besteht darin, daß nicht nur 1:1-



**Abbildung 6.1:** Graphdarstellung von „Objekt bearbeiten“  
Beziehungen, sondern auch 1:n- bzw. m:n-Beziehungen ausgehend vom mehrmals auftretenden Objekt durch Verfolgen der Kanten vollständig erfaßt werden können.

Ein Beispiel zur Erklärung: Das Ereignis „Objekt bearbeiten“ ist eine 1:n-Beziehung zwischen den Objekten „Benutzer“ und „Objekt“. Ein Benutzer kann in dieser Beziehung mehrmals auftauchen, da er gleichzeitig mehrere Objekte bearbeiten kann. Je Objekt gibt es nur maximal einen Benutzer, der es zur Zeit bearbeitet. Von jedem Objekt geht demzufolge maximal eine Kante aus. Von einem Benutzer jedoch können

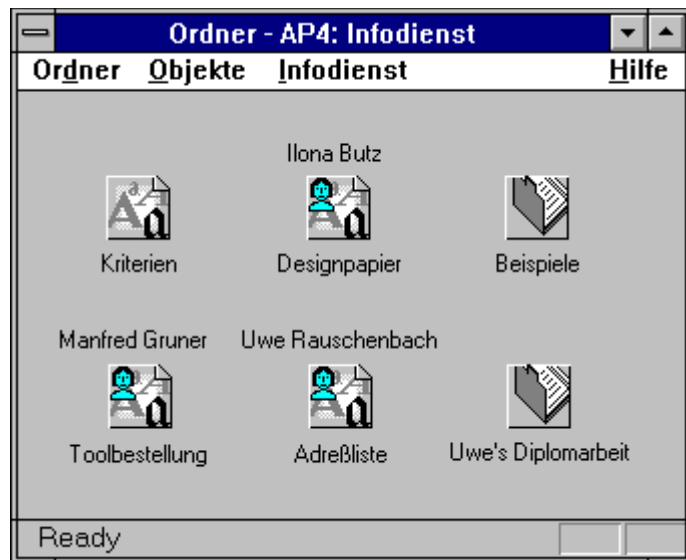
<sup>1</sup> Vgl. [SOH94] und [SOH95].

mehrere Kanten ausgehen, die bei der Darstellung als Graph auch alle verfolgt werden können.

Dem o.g. Vorteil steht der Nachteil gegenüber, daß die Darstellung schnell unübersichtlich wird und daß das Verfolgen von aus dem sichtbaren Bereich herauslaufenden Kanten problematisch ist. Daher ist diese Technik eher für das aktive Erkunden von Beziehungen in einem Netz mit interaktiven Navigationsmöglichkeiten geeignet als für das intuitive Erfassen dessen, was gerade im aktuellen Arbeitsbereich passiert. Ein weiterer Nachteil ist die Notwendigkeit, in jedem Container Icons aller Benutzer darzustellen, was zu uneffektiver Nutzung des knappen Platzes auf dem Bildschirm führt.

## 2) Darstellung als grafisches Attribut eines Icons

Bei dieser Darstellungsform wird der Ereignisinitiator als grafisches Attribut des Icons codiert. Die Vorteile der Technik liegen in ihrer Übersichtlichkeit auch bei einer hohen Anzahl von Objekten und Benutzern. Nachteilig ist, daß 1:n- und m:n-Beziehungen ausgehend vom mehrfach auftretenden Objekt nicht erfaßbar sind. Beispielsweise wäre die Frage



„Welche Objekte bearbeitet Uwe Rauschenbach gerade?“ nur durch Absuchen aller Objekticons zu beantworten.

Eine Farbcodierung des Benutzers ist nicht möglich, weil das die Benutzeranzahl zu stark einschränken würde. Daher sollte entweder der Name des Benutzers als Text dargestellt, der Benutzer in die Form eines überlagerten Sinnbildes codiert oder idealerweise ein Bild des Benutzers dem Icon des Ereignisobjektes überlagert werden. Das schränkt die Anzahl der gleichzeitig darstellbaren Beziehungen auf maximal eine bis zwei ein. In Abbildung 6.2 wird eine Möglichkeit zur Darstellung der Beziehung „Objekt bearbeiten“ als grafisches Attribut veranschaulicht.

## 3) Wertung der beiden Techniken

Für die Präsentation von Ereignismeldungen in einem gemeinsamen Arbeitsbereich zur Kooperationsunterstützung ist die Darstellung der Objekt-Benutzer-Beziehung als grafisches Attribut eines Icons besser geeignet als die Graphdarstellung. Beim gemein-

samen Zugriff auf Objekte sind diese und nicht die Benutzer Arbeitsgegenstand, d.h., der Informationsbedarf eines Benutzers bezieht sich primär auf die Objekte.

Wenn beispielsweise ein Benutzer ein gesperrtes Objekt bearbeiten möchte, ist er daran interessiert herauszufinden, wer das Objekt gerade bearbeitet. Er kann dann mit diesem Benutzer über einen Kommunikationskanal Kontakt aufnehmen und mit ihm über die Freigabe des Objektes verhandeln. Ein Herangehen aus der anderen Richtung (Frage: „Welche Objekte bearbeitet Uwe Rauschenbach gerade?“) unterstützt nicht die Kooperation, sondern die Überwachung von Personen.

Wie bereits im Abschnitt 6.3.2 (c)) festgestellt, ist die Darstellung des Ereignisinitiators nur bei Aktivitätsereignissen essentiell für die Unterstützung des Benutzers. Bei allen anderen Ereignistypen genügt es, wenn die Information auf Nachfrage bereitgestellt wird. Es gibt nur zwei Aktivitätsereignisse, die dargestellt werden müssen:

- das Bearbeiten eines Objektes und
- die Anwesenheit im gemeinsamen Arbeitsbereich.

Da letztere als Farbe des Benutzer-Icons im gemeinsamen Arbeitsbereich angezeigt werden kann, ist nur die Darstellung des Ereignisses „Objekt bearbeiten“ im Objekticon notwendig. Dazu ist die in Abbildung 6.2 vorgeschlagene Präsentation ausreichend.

### g) Beschreibung der Präsentationstechnik

Nach den Vorüberlegungen kann nun die Codierung von Objekt ereignissen im Objekticon entworfen werden.

Information	Darstellung	Bemerkungen
Aktivität „Benutzer eingeloggt“	Farbe des Benutzericons	ohne Ereigniszeit
Aktivität „Objekt gesperrt“	Überlagerung „Benutzer“ und Benutzername über Icon	ohne Ereigniszeit
Ereigniszeit	Intervall als Farbe des dem Ereignistyp entsprechenden Iconattributes	nicht bei Aktivitäten
Ereignis „Objekt gelöscht“	Überlagerung: „durchgestrichen“	andere Ereignistypen nicht mehr dargestellt
Ereignis „Objekt herausgenommen“	Überlagerung: „Pfeil“	andere Ereignistypen nicht mehr dargestellt
Ereignis „Objekt hineingelegt“	Umrahmung	
Ereignis „Objekt erzeugt“	Umrahmung	
Ereignis „Objekt geändert“	Sinnbild des Dokumentobjektes	
Ereignis „Objekt umbenannt“	Hinterlegung des Objektnamens	
Propagierte Meldungen	Sinnbild des Containerobjektes	

**Tabelle 6.2:** Im Objekticon dargestellte Information und die Art ihrer Darstellung

### 1) Darstellung der Ereigniszeit

Die Ereigniszeit bestimmt die Aktualität eines Ereignisses und damit der entsprechenden Meldung. Da mit Farben nur diskrete Werte unterscheidbar dar-

Ereigniszeitpunkt im Intervall	Farbe
älter als zwei Monate	grau
älter als einen, aber neuer als zwei Monate	blau
älter als eine Woche, aber neuer als einen Monat	grün
neuer als eine Woche	rot

**Tabelle 6.3:** Farbcodierung des Ereigniszeitpunktes

zustellen sind, wird die Ereigniszeit in eines von vier Intervallen abgebildet und entsprechend die Farbe für die Präsentation der Ereignismeldung ermittelt. Tabelle 6.3 und Farbtafel 6.3 veranschaulichen eine mögliche Zuordnung. Zumindest die Intervalllänge sollte dabei vom Benutzer konfigurierbar sein. Die Farbreihenfolge blau-gelb-rot oder grün-gelb-rot wäre natürlicher gewesen als die gewählte Reihenfolge blau-grün-rot, jedoch ist die Farbe gelb im System LinkWorks bereits für selektierte und offene Objekte reserviert (siehe Farbtafel 6.1). Die Ereigniszeit für Aktivitäts-Startereignisse braucht nicht angezeigt zu werden, da sie am Ende der Aktivität gelöscht werden und ihr Alter daher i.d.R. einen Tag nicht überschreitet. Dadurch wird das Attribut Farbe bei Aktivitätsereignissen frei für die Anzeige, ob der lokale oder ein entfernter Benutzer das Ereignis ausgelöst hat.

### 2) Darstellung der Aktivität „Benutzer eingeloggt“

Die Farbe eines Benutzericons im gemeinsamen Arbeitsbereich zeigt an, ob der entsprechende Benutzer eingeloggt ist oder nicht (siehe Farbtafel 6.4). Das Icon eines im Arbeitsbereich eingeloggten Benutzers wird farbig, das eines nicht eingeloggten Benutzers grau dargestellt. Bei der farbigen Darstellung wird zwischen dem lokalen Benutzer<sup>1</sup> (gelb) und den entfernten Benutzern (cyan) unterschieden. Diese Darstellung ist konsistent mit dem lokalen Echo (selektierte bzw. geöffnete Objekte werden in Gelbtönen dargestellt) sowie der Anzeige der Aktivität „Objekt gesperrt“.

### 3) Darstellung der Aktivität „Objekt gesperrt“

Die Aktivität „Objekt gesperrt“ wird durch das Überlagern des Objekticons mit einem Benutzericon und zusätzlich durch die Beschriftung mit dem Namen des bearbeitenden Benutzers über dem Objekticon dargestellt (siehe Farbtafel 6.4). Die Farbe des überlagerten Benutzericons zeigt an, ob das Objekt gerade vom aktuellen Benutzer (gelb) oder von einem anderen Benutzer (cyan) bearbeitet wird. Falls der aktuelle Benutzer der Bearbeiter ist, wird zusätzlich der Iconhintergrund dunkelgelb dargestellt, da das Objekt geöffnet ist. Das sichert die Konsistenz mit dem LinkWorks-Grafikalphabet (siehe Farbtafel 6.1).

<sup>1</sup> Der lokale Benutzer wird häufig auch als aktueller Benutzer bezeichnet.

#### 4) Darstellung der Ereignisse „Objekt hineingelegt“ und „Objekt erzeugt“

Beide Ereignisse schaffen ein neues Objekt im entsprechenden Container. Die Darstellung beider Ereignistypen erfolgt daher identisch durch Zeichnen eines Rahmens um das Ereignisobjekt (siehe Farbtafel 6.4). Die Rahmenfarbe codiert das Intervall, in dem die Ereigniszeit liegt.

#### 5) Darstellung des Ereignisses „Objekt gelöscht“

Das Löschen eines Objektes erzeugt als Platzhalter ein sogenanntes *Ghost-Objekt*. Ein solches Objekt existiert nicht mehr, es dient lediglich zur Darstellung der Löschmeldung. Sie erfolgt durch Überlagern des Icons mit einem Kreuz (dem Symbol für „durchgestrichen“) in der dem Ereigniszeitpunkt entsprechenden Farbe (siehe Farbtafel 6.4). Meldungen über Ereignisse aller anderen Typen werden unterdrückt.

#### 6) Darstellung des Ereignisses „Objekt herausgenommen“

Auch das Herausnehmen eines Objektes aus einem Container erzeugt ein Ghost-Objekt, da das herausgenommene Objekt im Container nicht mehr existiert. Wie beim Löscheignis erfolgt auch hier keine Darstellung der Ereignisse anderer Typen, lediglich das Herausnehmen wird durch Überlagern des Icons mit einem Pfeil in der dem Ereigniszeitpunkt entsprechenden Farbe angezeigt (siehe Farbtafel 6.4).

#### 7) Darstellung des Ereignisses „Objekt geändert“

Der Zeitpunkt des letzten Änderungsereignisses eines Dokumentobjekts wird in die Farbe des Sinnbildes des Objekticons codiert (siehe Farbtafel 6.4). Dabei werden ein heller und ein dunkler Ton der entsprechenden Farbe verwendet.

#### 8) Darstellung des Ereignisses „Objekt umbenannt“

Das Umbenennen eines Objektes wird durch eine dem Ereigniszeitpunkt entsprechende farbige Hinterlegung des schwarz dargestellten Icontextes angezeigt (siehe Farbtafel 6.4). Falls der Cursor auf dem umbenannten Objekt steht, werden Vordergrund- und Hintergrundfarbe des Textes invertiert.

#### 9) Darstellung propagierter Meldungen

Propagierter Meldungen können sich nur in Containerobjekten befinden. Weiterhin gibt es für Containerobjekte keine Änderungsereignisse. Daher kann das Vorhandensein propagierter Meldungen in die Farbe des Sinnbildes des Containericons codiert werden (siehe Farbtafel 6.4). Die Farbe stellt hierbei den Ereigniszeitpunkt der propagierten Meldung dar, die - sichergestellt durch den Propagierungsalgorithmus<sup>1</sup> - die aktuellste in der Containerhierarchie unterhalb des betreffenden Containers ist.

---

<sup>1</sup> Vgl. Abschnitt 5.3.2.

### 10) Interaktivität

Die interaktiven Möglichkeiten der Präsentationstechnik beschränken sich auf das Quittieren (Löschen) aller Meldungen der selektierten Objekte im Container und den Aufruf der Darstellungstechnik mit der höchsten Interaktivität. Das kann z.B. über Befehle eines Menüs geschehen.

## 6.3.3 Technik 3: Kontextgebundene Präsentation

### a) Prinzip

Die kontextgebundene Präsentation von Ereignismeldungen basiert auf der Darstellung einer Zusammenfassung der Ereignismeldungen in deren Entstehungskontext. Diese Zusammenfassung wird an einem festen Ort im Container (Meldungsbereich) angezeigt.

Eine Darstellung in übergeordneten Containern des Entstehungskontextes ist nicht vorgesehen. Dafür wäre es notwendig, in jeder Meldung nicht nur einen Meldekontext, sondern den kompletten Propagierungsweg zu speichern. Das würde zu einem erhöhten Speicherplatzbedarf für die dafür notwendigen Datenstrukturen führen. Auch das Ausdünnen propagierter Meldungen wäre nicht möglich.

### b) Charakterisierung

Bei der kontextgebundenen Präsentation handelt es sich um eine relativ unaufdringliche Technik zur Meldungsdarstellung. Weil die Präsentation an einem festen Ort im Container erfolgt und bei geöffnetem Container immer sichtbar ist, kann die Aufmerksamkeit eines Benutzers besser auf ein eintretendes Ereignis gelenkt werden als bei der objektgebundenen Darstellung. Das gilt jedoch nur, wenn das Containerfenster nicht durch andere Fenster verdeckt ist.

Der Grad der Interaktivität der Technik ist ebenfalls höher als bei der objektgebundenen Präsentation, weil Meldungen (bzw. Meldungsgruppen) selbst als Objekte und nicht als Objektattribute dargestellt werden. Auch bei dieser Art der Präsentation sollte es aber möglich sein, auf Benutzerinitiative eine Technik mit noch höherer Interaktivität aufzurufen.

### c) Darzustellende Information

Die Darstellung aller anstehenden Meldungen im Container ist i.d.R. zu detailliert und daher zu platzraubend. Deshalb müssen die Meldungen sinnvoll nach Kriterien zusammengefaßt werden, die eine Einteilung in etwa sieben Kategorien<sup>1</sup> ermöglichen. Diese Bedingung wird von folgenden Auswahlkriterien erfüllt:

- Ereignistyp und
- Zeitintervall zur Einordnung des Ereigniszeitpunktes.









Aus der Darstellung sollte zumindest ein Überblick über Anzahl und Aktualität der anstehenden Ereignismeldungen hervorgehen. Als günstig erweist sich auch die Möglichkeit, die zugrundeliegenden Ereignistypen zu erkennen. Eine detailliertere Darstellung der aktuellsten Meldung ist ebenfalls in Betracht zu ziehen.

### d) Beschreibung der Präsentationstechnik

#### 1) *Darstellung*

Die obigen Betrachtungen sollen jetzt in eine kontextgebundene Darstellungstechnik für Ereignismeldungen umgesetzt werden. Farbtafel 6.5 veranschaulicht die Ideen.

Das Fenster eines jeden Containerobjektes wird um einen Meldungsbe-  
reich erweitert, in dem die Ereignismel-  
dungen nach ihrem Typ zusammenge-  
faßt in einem Band von Icons angezeigt  
werden. Je Ereignistyp gibt es ein Icon,  
dessen Sinnbild den Typ repräsentiert  
und das die Anzahl der anstehenden  
Meldungen über Ereignisse dieses Typs

Sinnbild	Semantik
	Objekt von Benutzer gesperrt
	Objekt erzeugt
	Objekt aus Container herausge- nommen
	Objekt in Container hineingelegt
	Objekt gelöscht
	Objekt umbenannt
	Objekt geändert
	Untergeordnete Container enthalten propagierte Meldungen

**Tabelle 6.4:** Semantik der Ereignistypsymbole

im Container als Zahl wiedergibt. Die Farbe des Sinnbildes stellt den Ereigniszeitpunkt der aktuellsten anstehenden Meldung je Ereignistyp dar und wird nach demselben Schema ermittelt wie in Abschnitt 6.3.2 (g)) beschrieben. In zwei weiteren Icons wird die Gesamtanzahl der anstehenden Meldungen sowie die Anzahl von Meldungen in untergeordneten Containern angezeigt. Zwei Scrollbuttons ermöglichen das Scrollen des Icon-Bandes bei zu geringer Breite des Fensters.

<sup>1</sup> Nach den Erkenntnissen der Wahrnehmungspsychologie liegt die Anzahl der gleichzeitig wahrnehmbaren Informationseinheiten bei etwa sieben.

Jedes Icon kann drei Zustände annehmen:

- *inaktiv*, wenn keine Meldungen über Ereignisse des entsprechenden Typs anstehen.
- *normal*, wenn mindestens eine Meldung über ein Ereignis des entsprechenden Typs ansteht. Ein „normales“ Icon zeigt das Ereignistypsymbol in der dem Zeitpunkt des aktuellsten Ereignisses entsprechenden Farbe und die Anzahl der Meldungen als Zahl an.
- *selektiert*, wenn mindestens eine Meldung über ein Ereignis des entsprechenden Typs ansteht und diese Meldungen vom Benutzer ausgewählt wurden. Ein selektiertes Icon kann zur Unterscheidung von einem nicht selektierten Icon beispielsweise durch einen andersfarbigen Hintergrund oder den 3D-Effekt „eingedrückt“ dargestellt werden.

Zur Verbesserung der Erkennbarkeit und damit der Signalwirkung kann auf neue Meldungen durch kurzzeitiges Blinken des dem Ereignistyp entsprechenden Icons aufmerksam gemacht werden, bevor die Änderungen statisch dargestellt werden.

## 2) Interaktivität

Zwischen den Zuständen „normal“ und „selektiert“ eines Icons kann vom Nutzer per Mausklick umgeschaltet werden.

Auf selektierte Meldungsgruppen beziehen sich die interaktiven Möglichkeiten der Technik, die über ein Menü (siehe Farbtafel 6.6) aktiviert werden können. Das Menü bietet die folgenden Operationen auf den selektierten Meldungsgruppen an:

- *Gesehen* markiert alle Meldungen der selektierten Gruppen als „gesehen“.
- *Quittieren* löscht alle Meldungen der selektierten Gruppen.
- *Zurückstellen* stellt alle Meldungen der selektierten Gruppen auf einen späteren Zeitpunkt zurück.
- *Meldungsübersicht* ruft die Darstellungstechnik mit der höchsten Interaktivität auf.
- *Objekte* selektiert alle Ereignisobjekte der Meldungen der selektierten Gruppen.

Durch Klicken auf die Icons, die die Gesamtanzahl der Meldungen bzw. die Anzahl propagierter Meldungen darstellen, kann die Meldungsübersicht (siehe Farbtafel 6.10) für alle oder für die propagierten Meldungen aufgerufen werden.

### 6.3.4 Technik 4: Nicht-modale kontextübergreifende Präsentation

#### a) Prinzip

Bei der kontextübergreifenden Präsentation wird die Meldung räumlich getrennt von ihrem Entstehungskontext und dessen übergeordneten Kontexten angezeigt. Die Darstellung erfolgt in einer *nicht-modalen Dialogbox*. Diese wird am Bildschirmrand angeordnet und ist immer sichtbar. Aus diesem Grund beansprucht sie ständig Bildschirmplatz.

#### b) Charakterisierung

Die Signalwirkung der nicht-modalen kontextübergreifenden Präsentation ist hoch, da die Dialogbox zum Zeitpunkt des Eintreffens der Meldung sichtbar ist.

Der Objektbezug ist nicht aus dem Ort der Präsentation ablesbar. Daher muß das Herstellen des Ortsbezuges durch textliche Information bzw. Interaktivität ermöglicht werden. Die Aufdringlichkeit der nicht-modalen Meldebox ist relativ gering, da beim Eintreffen einer Meldung kein neues Fenster auftaucht, sondern nur die Informationsdarstellung in einem existierenden Fenster geändert wird.

Die nicht-modale Meldebox ermöglicht nur einen eingeschränkten Detaillierungsgrad, da sie ständig Bildschirmplatz beansprucht. Aufgrund des Übersichtscharakters der nicht-modalen Darstellung ist ihre Interaktivität nicht besonders hoch und kann mit der der kontextgebundenen Technik verglichen werden.

#### c) Darzustellende Information

Die darzustellende Information ist eine Übersicht über Meldungen, die relativ wichtig sind.

#### d) Beschreibung der Präsentationstechnik

Hier wird eine Zusammenfassung der Ereignismeldungen in einer nicht-modalen Dialogbox am Bildschirmrand präsentiert. Die grafische Umsetzung entspricht der kontextgebundenen Darstellung<sup>1</sup>. Der einzige Unterschied ist das Fehlen des Icons für propagierte Meldungen, da die Technik keinen direkten Kontextbezug ermöglicht.

Die grafischen Elemente können entweder horizontal oder vertikal angeordnet werden, so daß die Box sowohl am oberen oder unteren als auch am rechten oder linken Bildschirmrand plaziert werden kann. Zusätzliche Interaktivität gegenüber der

---

<sup>1</sup> Vgl. Abschnitt 6.3.3 (d).

kontextgebundenen Darstellung bietet die Funktion *Schließen*, mit der der Benutzer die Box so lange geschlossen halten kann, bis eine neue Meldung eintrifft. Durch das Weglassen des Titelbalkens wird der Platzbedarf auf dem Bildschirm minimiert. Farbtafel 6.7 und Farbtafel 6.8 zeigen eine mögliche Realisierung dieser Technik.

Der Menüeintrag <Meldungsübersicht> öffnet die modale Meldebox (siehe Farbtafel 6.10), die alle in der nicht-modalen Box übersichtsartig angezeigten Meldungen detaillierter darstellt und eine umfangreichere Interaktivität anbietet.

### **6.3.5 Technik 5: Modale kontextübergreifende Präsentation**

#### **a) Prinzip**

Auch bei der modalen kontextübergreifenden Präsentation wird die Meldung räumlich getrennt von ihrem Entstehungskontext und dessen übergeordneten Kontexten dargestellt. Die Darstellung erfolgt in einer *modalen Dialogbox*. Diese taucht beim Eintreffen einer neuen Meldung auf, unterbricht den Benutzer bei seiner Arbeit und muß vor der Weiterarbeit geschlossen werden. Der Meldekontext einer Meldung dient als Informationsfilter: Kontextübergreifend werden nur Meldungen dargestellt, deren Meldekontext geöffnet ist.

#### **b) Charakterisierung**

Die Signalwirkung der modalen kontextübergreifenden Präsentation ist von allen vorgestellten Techniken am höchsten, weil die modale Dialogbox bei ihrem Auftauchen nicht durch andere Fenster verdeckt wird und sie eine Eingabe des Benutzers erzwingt. Der Objektbezug ist nicht aus dem Ort der Präsentation ablesbar. Daher muß er durch textliche Information bzw. Interaktivität herstellbar sein. Die Aufdringlichkeit der modalen Meldebox ist maximal. Die modale Meldebox beansprucht nicht ständig Bildschirmplatz. Daher gestattet sie eine beliebige, interaktiv weiter verfeinerbare Detaillierung, was die höchste Interaktivität aller Techniken ermöglicht.

#### **c) Darzustellende Information**

Die darzustellende Information ist die Gesamtheit der Attribute sehr wichtiger Meldungen.

## d) Beschreibung der Präsentationstechnik

### 1) Darstellung

Die modale Dialogbox unterbricht den Benutzer bei der Arbeit und zwingt ihn, von den angezeigten Ereignissen Kenntnis zu nehmen. Eine platzsparende Anzeige ist nicht gefordert, da der Eingabefokus auf die modale Meldebox beschränkt ist<sup>1</sup>. Das ermöglicht eine detaillierte Darstellung und damit eine sehr hohe Interaktivität. Farbtafel 6.9 zeigt die modale Dialogbox des Meldedienstes. Jede Ereignismeldung wird als Eintrag in einer Listbox dargestellt, der neben dem bereits bekannten Ereignistypsymbol zur visuellen Darstellung von Ereignistyp und -zeit außerdem das Ereignisdatum, Namen und Klasse des Ereignisobjektes und den Ereignistyp im Klartext anzeigt. Ereignismeldungen mit dem gleichen Ereignis-Elternobjekt werden zusammengefaßt unter einer Kopfzeile dargestellt, die aus Namen und Klasse des Elternobjektes besteht. Zurückgestellte Meldungen erhalten nach Ablauf der Zurückstellungsfrist den Vorsatz „ERINNERUNG“.

### 2) Interaktivität

Die modale Meldebox bietet vielfältige Möglichkeiten der Interaktion mit den Meldungen. Die Operationen werden durch Selektieren der Meldung in der Listbox und Mausklick auf einen entsprechenden Pushbutton ausgelöst. Folgende Operationen sind möglich:

- *OK* markiert alle nicht quittierten und nicht gesehenen Meldungen als „gesehen“ und schließt die Meldebox.
- *Gesehen* markiert die selektierten Meldungen in der Meldebox als „gesehen“.
- *Quittieren* löscht die selektierten Meldungen in der Meldebox.
- *Zurückstellen* stellt die selektierten Meldungen in der Meldebox auf einen späteren Zeitpunkt zur Erinnerung zurück.
- *Details* zeigt das der selektierten Meldung zugrundeliegende Ereignis in einer weiteren Dialogbox detailliert an (siehe Farbtafel 6.11). Das betrifft Name und Klasse von Ereignisobjekt und dessen Elternobjekt, Name des Ereignisinitiators, Ereignisdatum und -zeit sowie Parameter des Ereignistyps (wie das neue Elternobjekt eines herausgenommenen Objektes oder der alte Name eines umbenannten Objekts).
- *Objekt* selektiert das Ereignisobjekt in dessen Elternobjekt. Dazu wird das Elternobjekt geöffnet, falls es noch nicht geöffnet ist.

---

<sup>1</sup> In andere, parallel zu einer modalen Meldebox sichtbare Fenster können keine Eingaben erfolgen.

Zur Bereitstellung von Interaktivität für kontext- oder objektgebunden bzw. nicht dargestellte Meldungen gibt es eine weitere Form der modalen Meldebox, die in Farbtafel 6.10 abgebildet ist. Sie unterscheidet sich von der oben vorgestellten, auf Systeminitiative geöffneten Meldebox dadurch, daß sie auf *Benutzerinitiative* aus einem *Kontext* heraus geöffnet wird. Daher hat sie Zugriff auf Kontextinformation und ermöglicht im Gegensatz zur systeminitiierten Meldebox zusätzlich ein einfaches Verfolgen propagierter Ereignismeldungen innerhalb der Containerhierarchie. Sie bietet weiterhin die Möglichkeit, die minimale Wichtigkeit der anzuzeigenden Ereignismeldungen zu spezifizieren. Statt des *OK*-Buttons enthält sie einen *Schließen*-Button, der die Dialogbox ohne weitere Änderungen schließt.

Die Verfügbarkeit der Interaktionsmöglichkeiten hängt auch von der Relevanz der gerade selektierten Ereignismeldungen ab. Die in Abschnitt 5.2.3 (c)) genannten unzulässigen Kombinationen sind gesperrt. Insbesondere ist es nicht möglich, eine propagierte Meldung als „gesehen“ zu markieren oder zurückzustellen. Durch einen Doppelklick auf die entsprechende Zeile kann statt dessen der Propagierungsweg entweder durch Öffnen einer untergeordneten Hierarchiestufe weiterverfolgt oder eine solche geschlossen werden, wenn sie geöffnet ist.

## 6.4 Relevanzabhängige Auswahl der Präsentationstechnik

### 6.4.1 Prinzip

In Abschnitt 6.2.3 wurde die Dringlichkeit einer Präsentationstechnik eingeführt. Weiterhin wurden fünf Präsentationstechniken entworfen, die eine abgestufte Dringlichkeit aufweisen. Abhängig von ihrer Relevanz wird einer Meldung eine *Maximalpräsentationstechnik* zugewiesen. Die Meldung wird mit allen Präsentationstechniken dargestellt, deren Dringlichkeit kleiner oder gleich der Dringlichkeit dieser Maximalpräsentationstechnik ist. Das stellt die Konsistenz zwischen den einzelnen Präsentationstechniken sicher. Beispielsweise würde eine Meldung, die kontextübergreifend nicht-modal dargestellt wird, auch im Meldebereich ihres Entstehungskontextes und im Icon des Ereignisobjektes zu finden sein.

## 6.4.2 Der Auswahlmechanismus

Die Auswahl der initialen Maximalpräsentationstechnik für eine Meldung wird vom Wichtigkeits-Attribut ihrer Relevanz bestimmt. Dabei gelten die in Tabelle 6.5 dargestellten Zuordnungen. Die zusätzlichen Attribute der Relevanz einer Meldung werden wie folgt berücksichtigt:

Wichtigkeit	Präsentation
unwichtig	Keine
interessant	+ Objektgebunden
lokal wichtig	+ Kontextgebunden
wichtig	+ Nicht-modal kontextübergreifend
sehr wichtig	+ Modal kontextübergreifend

**Tabelle 6.5:** Präsentation entsprechend der Wichtigkeit

- 1) Die kontextübergreifende Präsentation erfolgt nur, wenn der Meldekontext der Meldung geöffnet ist. Dieser dient damit als Informationsfilter.
- 2) Gelesene Meldungen werden maximal objektgebunden präsentiert.
- 3) Zurückgestellte Meldungen werden vom Zeitpunkt der Zurückstellung bis zum Ablauf des Zurückstellungsintervalls maximal objektgebunden präsentiert.
- 4) Propagierte Meldungen werden nicht kontextübergreifend dargestellt<sup>1</sup>. Ihre Darstellung erfolgt bei einer Wichtigkeit größer als „interessant“ immer kontext- und objektgebunden, ansonsten wie in Tabelle 6.5 festgelegt. Dabei werden 2) und 3) beachtet.
- 5) Abweichend von 4) werden propagierte Meldungen in der modalen Meldebox dargestellt, wenn diese auf Benutzerinitiative aus einem Kontext heraus aufgerufen wurde.

Meldungen über vom aktuellen Benutzer ausgelöste Ereignisse werden diesem maximal objektgebunden präsentiert. Dies dient als lokales Echo seiner Aktionen.

<sup>1</sup> Das gilt nur für die propagierten Kopien der Meldung, nicht für die an das Ereignisobjekt gebundene primäre Meldung.

---

# 7 Vorstellung eines Prototypen

---

## 7.1 Allgemeines

Zur Veranschaulichung des vorgestellten Konzeptes erstellte der Autor einen Prototypen, der zwei Präsentationstechniken (die Darstellung im Objekticon und die Darstellung in der modalen Meldebox) implementiert. In diesem Abschnitt werden einige ausgewählte Aspekte der technischen Realisierung des Prototypen beschrieben. Grundkenntnisse des Systems LinkWorks<sup>1</sup> und der Klassenbibliothek MFC<sup>2</sup> müssen zum vollständigen Verständnis dieses Abschnitts vorausgesetzt werden, da deren Vermittlung den Rahmen vorliegender Arbeit sprengen würde. Die Funktionsbeschreibung des Prototypen aus Benutzersicht kann in Anlage B nachgelesen werden, Anlage C stellt ein Nutzungsszenario vor.

## 7.2 Ziele der Erstellung des Prototypen

Mit der Erstellung des Prototypen wurden folgende Ziele verfolgt:

- Aufzeigen der Realisierbarkeit des vorgestellten Konzeptes
- Gewinnung von Erkenntnissen zur Vervollständigung des Konzeptes
- Erkunden der Möglichkeiten, die LinkWorks zur Erweiterung des Funktionsumfangs bietet
- Schaffung einer Diskussionsgrundlage für den weiteren Projektverlauf in den Arbeitspaketen 3 und 4<sup>3</sup>
- Schaffung einer vorführbaren Lösung als Diskussionsgrundlage in den Anwenderworkshops

---

<sup>1</sup> Es wurde das System LinkWorks 3.0 benutzt, das in [DEC94] beschrieben ist.

<sup>2</sup> MFC - Microsoft Foundation Classes. Es wurde die Version 2.0 verwendet. Siehe [KRU93] für eine Beschreibung der MFC.

<sup>3</sup> AP 3 befaßt sich mit dem Gemeinsamen Arbeitsbereich, AP 4 mit dem Benachrichtigungs- und Informationsdienst

Der Prototyp wurde als „Throw-Away-Prototype“<sup>1</sup> konzipiert, um schnell ein lauffähiges System implementieren zu können. An den Funktionsumfang wurden folgende Forderungen gestellt:

- Spezifizieren von Interesse für Objekteignisse
- Implementation zweier Präsentationstechniken (Codierung im Icon und Meldebox) für Farbgrafikkarten mit mindestens 16 Farben in Symboldarstellung
- Implementation einiger Lebenszyklusoperationen auf Meldungen als Beispiel
- Einbindung in LinkWorks
- Unterstützung der LinkWorks-Grundfunktionalität für Objektzugriffe
- Implementierung eines einfachen gemeinsamen Arbeitsbereiches mit Benutzern, einer Containerobjektklasse und einer Dokumentobjektklasse als Grundlage

## 7.3 Machbarkeitsanalyse auf der Basis von LinkWorks

### 7.3.1 Allgemeines

LinkWorks gestattet eine Erweiterung seiner Funktionalität über die in Abschnitt 4.2 erwähnten Programmierschnittstellen. Zur Zeit der Durchführung der beschriebenen Arbeiten war die Betaversion des Systems LinkWorks 3.0 bei der VW-GEDAS verfügbar, in der die Programmierschnittstellen neu eingeführt worden waren. Es lagen also noch keine Erfahrungen bei deren Nutzung vor.

Ein Ziel des Prototypings war es, Erfahrungen mit der LinkWorks-Schnittstelle zu sammeln. Dabei zeigte sich, daß die Flexibilität der aktuellen LinkWorks-Version für die Realisierung des in Abschnitt 5 entworfenen Konzeptes *nicht* ausreicht. Im folgenden Abschnitt werden die Anforderungen der geplanten Architektur an die Erweiterbarkeit von LinkWorks aufgezeigt.

### 7.3.2 Anforderungen der geplanten Architektur

Der Prototyp soll einen gemeinsamen Arbeitsbereich für den Nutzer verfügbar machen, in dem Meldungen über Benutzeraktivitäten relevanzabhängig dargestellt werden. Er soll Teilprozeß des LinkWorks-Clientprozesses auf dem lokalen PC des Benutzers sein.

In Abschnitt 5.2.3 (f)) wurden zwei Speicherungsmöglichkeiten für Meldungen aufgezeigt: die Speicherung als Objektattribut und die Speicherung als Meldungstabelle.

---

<sup>1</sup> Vgl. [SOM92], Seite 107

Beide Speicherungsformen benötigen LinkWorks-Objektattribute dynamischer Länge, in denen Daten in Tabellenform abgelegt werden können. Daraus ergibt sich die erste Anforderung:

**Anforderung 1:** Es müssen strukturierte, dynamische Attribute einer LinkWorks-Objektklasse in der Workbench erzeugbar sein.

Beim Eintreffen einer neuen Meldung muß eine Routine des Client-Prozesses aufgerufen werden, die die notwendigen Änderungen an der Benutzeroberfläche vornimmt. Der existierende LinkWorks-Benachrichtigungsdienst stellt alle Meldungen in einer modalen Dialogbox dar. Meldungen werden als Strings im *Ereignisbuch* auf dem LinkWorks-Server gespeichert. Es wäre günstig, dieses existierende Protokoll zu nutzen und zu erweitern.

**Anforderung 2:** Das existierende Protokoll der Meldungsverteilung muß genutzt und erweitert werden können.

Die Präsentationstechniken mit niedriger Aufdringlichkeit erfordern eine Änderung der Darstellung der Objekticons bzw. das Hinzufügen eines Meldebereiches zum Containerfenster. Daraus ergibt sich die dritte Anforderung.

**Anforderung 3:** Es muß möglich sein, Methoden zur Darstellung des Userinterfaces zu überschreiben bzw. zu erweitern.

### 7.3.3 Aufgetretene Probleme

LinkWorks ermöglicht die Nutzung formatierter Strings als dynamische Attribute. Damit werden beispielsweise Objektläufe abgespeichert. Leider werden diese dynamischen Strings nicht als möglicher Typ für benutzerdefinierte Attribute angeboten. Statische formatierte Strings werden zwar unterstützt, sind aber in der Länge auf 132 Zeichen begrenzt. Damit wird Anforderung 1 nicht erfüllt.

Der Zugriff auf das Ereignisbuch aus benutzerdefinierten Methoden wird nicht unterstützt. Das existierende Protokoll der Meldungsverteilung kann nicht erweitert werden, da ein großer Teil der Funktionalität in privaten Methoden gekapselt ist. Private Methoden sind - wie in C++ - aus abgeleiteten Klassen nicht mehr aufrufbar. Damit wird Anforderung 2 nicht erfüllt.

Das teilweise Ersetzen bzw. Erweitern von Userinterface-Funktionalität ist ebenfalls nicht möglich, da die Zugriffe in privaten Methoden gekapselt sind und kein Zugang zu Fensterhandles u.ä. ermöglicht wird. Damit wird Anforderung 3 nicht erfüllt.

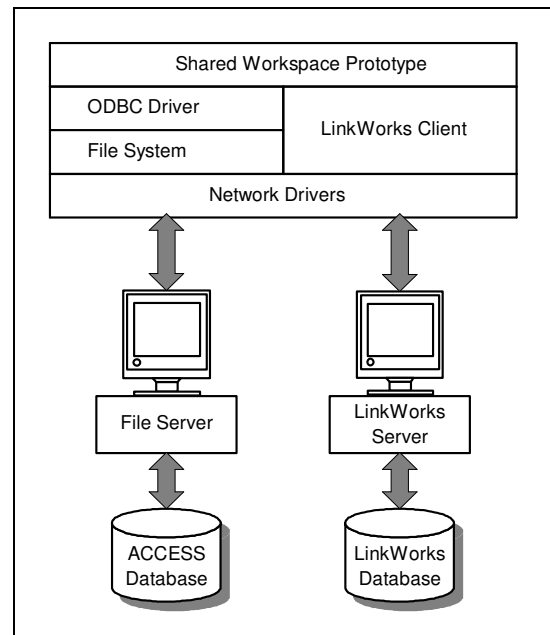
Folglich erfüllen die Erweiterungsmöglichkeiten von LinkWorks nicht die Anforderungen, die die Integration eines Benachrichtigungs- und Informationsdienstes stellt.

Daher suchte der Autor eine andere Lösung, um den Prototypen zu implementieren. Diese wird im folgenden Abschnitt beschrieben.

## 7.4 Beschreibung der realisierten Lösung

### 7.4.1 Systemarchitektur

Da die Speicherung der Ereignisse und Meldungen als LinkWorks-Objektattribute ausscheidet, mußten diese in einer externen Datenbank abgelegt werden. Weiterhin war ein Kommunikationskonzept zu entwickeln und die Grundfunktionalität des LinkWorks-Userinterfaces nachzuimplementieren. Diese Vorarbeiten waren notwendig, bevor überhaupt mit der Implementierung des eigentlichen Benachrichtigungsdienstes begonnen werden konnte. Abbildung 7.1 veranschaulicht die Systemarchitektur. Als Datenbank zur Speicherung von Interesse, Ereignissen und Meldungen wurde eine MS-ACCESS-Daten-



**Abbildung 7.1:** Systemarchitektur des Prototypen  
 bank gewählt, die als Datei auf einem Netzlaufwerk liegt. Der Zugriff auf die Datenbank erfolgt über ODBC-Treiber<sup>1</sup>. Der Gemeinsame Arbeitsbereich wurde als externes Werkzeug<sup>2</sup> in LinkWorks eingebunden. Das System LinkWorks wird benutzt,

- um die Objekte zu speichern, die sich im Gemeinsamen Arbeitsbereich befinden,
- zur Benutzerverwaltung,
- zur Manipulation der Objekte,
- zum Starten des externen Werkzeugs „Gemeinsamer Arbeitsbereich“.

<sup>1</sup> ODBC: Open Database Connectivity, Industriestandard von Microsoft zum Zugriff auf relationale Datenbanken unterschiedlichen Formats

<sup>2</sup> Ein externes Werkzeug ist eine ausführbare Datei, die auf den lokalen Festplatten der Client-Rechner vorliegt.

## 7.4.2 Einbindung in die Klassenstruktur von LinkWorks

LinkWorks wurde um fünf Objektklassen erweitert<sup>2</sup>: den *Gemeinsamen Arbeitsbereich*, eine Ordnerklasse, eine Dokumentklasse sowie die Klassen *Benutzer* und *Benutzerin*<sup>3</sup>.

Der Gemeinsame Arbeitsbereich wurde von der Klasse *Schrank* abgeleitet. Sie stellt einen Container zur Verfügung, in dem sich die Objektrepräsentation der Benutzer und die gemeinsam

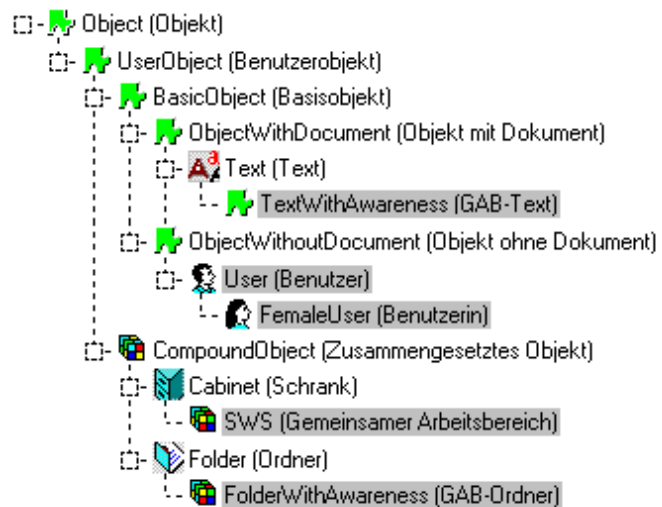


Abbildung 7.2: Neue LinkWorks-Objektklassen<sup>1</sup>

bearbeiteten Objekte befinden. In der LinkWorks-Systemkonfiguration wurde eine neue Werkzeugklasse mit der Aufrufsyntax „{Object.ToolName} {Object.Reference}“ erzeugt und diese der Objektklasse *Gemeinsamer Arbeitsbereich* zugewiesen. Für die Werkzeugklasse wurde das Werkzeug „SWS.EXE“ definiert. Die Methode *EditObject* der Klasse wurde überschrieben, um das externe Werkzeug mit der Objektreferenz des aufrufenden Objektes als Parameter zu starten. Die Ordnerklasse *GAB-Ordner*<sup>4</sup> wurde abgeleitet, um eine Strukturierung der Objektklassen im Gemeinsamen Arbeitsbereich zu ermöglichen. Die Dokumentklasse *GAB-Text* schließlich dient zum Speichern der Textdokumente, in deren Bearbeitung die kooperative Tätigkeit besteht. Mit Hilfe der Workbench-Funktion *Objektlage* kann für jede Containerobjektklasse festgelegt werden, welchen Klassen die darin befindlichen Objekte angehören dürfen. Tabelle 7.1 zeigt die gewählte Zuordnung. Da eine direkte Erweiterung der LinkWorks-Funktionalität nicht möglich ist, wird damit sichergestellt, daß sich keine Objekte nicht unterstützter Klassen im Gemeinsamen Arbeitsbereich befinden. Als Bearbeitungswerkzeug für den GAB-Text wurde WRITE eingestellt.

	Arbeitsbereich	GAB-Ordner	GAB-Text	Benutzerin	Benutzer
Arbeitsbereich		X	X	X	X
GAB-Ordner		X	X		

Tabelle 7.1: Zulässige Schachtelung von Objekten in Containerobjekten

<sup>1</sup> Neu eingeführte Objektklassen sind grau hinterlegt.

<sup>2</sup> Siehe Abbildung 7.2.

<sup>3</sup> Die Objektklasse *Benutzerin* wurde von der Klasse *Benutzer* abgeleitet, um eine unterschiedliche Icons für Benutzer und Benutzerin zu ermöglichen.

<sup>4</sup> GAB steht für Gemeinsamer Arbeitsbereich

Der Gemeinsame Arbeitsbereich benötigt Information darüber, welche Benutzer zugriffsberechtigt sind. Benutzerinformationen liegen im System als Management-Daten identifiziert durch einen User-ID vor. Der Zugriff eines Nutzers auf ein Objekt der Klasse „Gemeinsamer Arbeitsbereich“ wird über die Erteilung eines Verweises auf das Objekt an diesen ermöglicht. LinkWorks unterstützt jedoch nicht das Zurückverfolgen eines solchen Verweises vom Objekt zum Benutzer. Da dynamische Attribute eines Objektes nicht unterstützt werden, scheidet auch eine Liste der User-IDs der berechtigten Benutzer als Attribut aus. Daher wurden die Objektklassen Benutzer und Benutzerin erzeugt, um die Benutzer und Benutzerinnen des Systems aus den Managementdaten in die Objektstruktur zu integrieren. Über das Attribut *UserID* wird die Verbindung hergestellt. Die berechtigten Benutzer werden als Objekte im Gemeinsamen Arbeitsbereich erzeugt und als solche visualisiert.

### 7.4.3 Programm-Architektur

Das Programm wurde in Visual C++ V 1.51 implementiert und läuft unter MS-WINDOWS 3.1. Es benutzt die Klassenbibliothek *Microsoft Foundation Classes (MFC)*. Diese stellt ein Application Framework basierend auf der Dokument-Ansicht-Architektur zur Verfügung. Die Architektur ermöglicht die Trennung von Daten (*Document*) und Benutzeroberfläche (*View*). Für ein Dokument können mehrere Views existieren. Eine Applikation kann mehrere Dokumente enthalten. Für jedes Dokument gibt es mindestens ein Rahmenfenster (*Frame*), das einen oder mehrere Views und eine Menüleiste enthält.

Erzeugen, Verbinden und Verwalten der Kombination Dokument-Rahmen-Ansicht wird von *Document Templates* gesteuert. Die MFC kennen zwei Formen von Document Templates: Single Document Interface (SDI) und Multiple Document Interface (MDI). SDI unterstützt nur ein Dokument je Applikation und zeigt die Views des Dokuments im Hauptrahmenfenster des Programms an. Ein Beispiel für dieses Herangehen ist das Programm WRITE. MDI dagegen ist komplexer. Es unterstützt mehrere Dokumente je Applikation und stellt die Views eines Dokuments in Child-Fenstern des Hauptrahmenfensters dar. Das Hauptrahmenfenster klippt die Child-Fenster und nimmt die Icons ikonisierter Child-Fenster auf. Beispielsweise basiert Word for Windows auf dem Multiple Document Interface.

Für die Nachbildung der LinkWorks-Benutzerschnittstelle ist weder SDI noch MDI geeignet. Vielmehr ist eine Kombination beider Paradigmen gefordert: Das Hauptfenster der Applikation stellt die Objekte und Benutzer im Gemeinsamen Arbeitsbereich dar, die Child-Fenster zeigen Objekte in untergeordneten Ordnern an und werden nicht am Hauptfenster geklippt. Um das zu implementieren, wurde für das Wurzelobjekt eine

neue Document-Template-Klasse vom SDI-Template abgeleitet und in diese eine Instanz des MDI-Templates eingebettet. Ein Überladen der Methode zum Hinzufügen eines neuen Dokuments sorgt dafür, daß für das erste Dokument der Aufruf an die Superklasse SDI-Template weitergereicht und für jedes folgende Dokument die Hinzufügen-Methode des eingebetteten MDI-Templates aufgerufen wird. Das Klippen wird verhindert, indem beim Erzeugen des eingebetteten MDI-Templates nicht die geklippte Rahmenklasse, sondern die Klasse *CChildFrame* (s.u.) übergeben wird.

Im Prototypen wurden C++-Objektklassen eingeführt, die die fünf o.g. LinkWorks-Klassen ein kapseln. Die C++-Klassen mit einigen wichtigen Membervariablen sind in Abbildung 7.3, ihre Zuordnung zu den LinkWorks-Klassen in Tabelle 7.2 dargestellt.

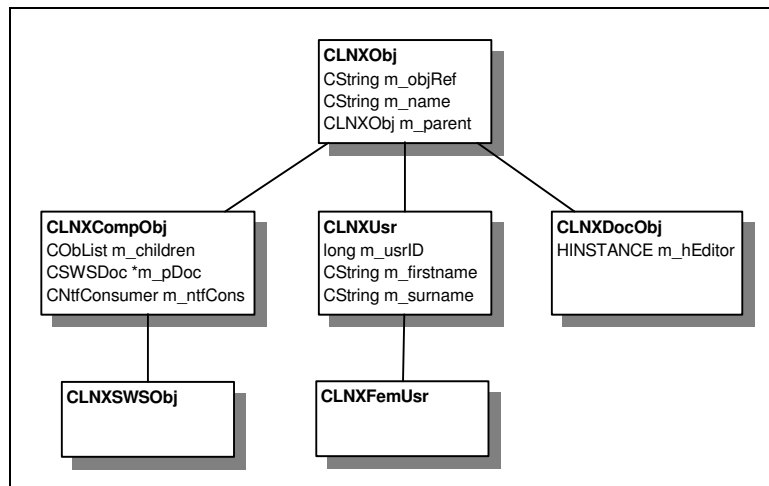


Abbildung 7.3: C++-Klassen zur Einkapselung der LinkWorks-Klassen

Weiterhin wurde die MFC-Dokument-Klasse<sup>2</sup> *CSWSDoc* implementiert, die ein Objekt der Klasse *CLNXCompObj* und ein Objekt zum Datenbank-

C++-Objektklasse	eingekapselte LinkWorks-Objektklasse <sup>1</sup>
CLNXObj	UserObject
CLNXCompObj	FolderWithAwareness
CLNXSWSObj	SWS
CLNXDocObj	TextWithAwareness
CLNXUsr	User
CLNXFemUsr	FemaleUser

zugriff auf die Meldungen in diesem Containerobjekt (Klasse *CNtfConsumer*<sup>3</sup>) enthält. Der *NtfConsumer* übernimmt das Aktualisieren der Ereignismeldungen durch *Polling* der ACCESS-Datenbank. Zunächst war auch ein Objekt der Klasse *CEventRaiser* Teil von *CSWSDoc*. Dieses dient zum Erzeugen und Speichern von Ereignissen und Ereignismeldungen in der Datenbank. Der Datenbankzugriff war jedoch unzumutbar langsam. Weil dadurch die Benutzeroberfläche nach dem Abschluß einer LinkWorks-Aktion auf einer Gruppe von Objekten teilweise 30 Sekunden oder länger durch Meldungserzeugung blockiert

<sup>1</sup> Vgl. Abbildung 7.2.

<sup>2</sup> Diese Klasse darf nicht mit der LinkWorks-Dokumentklasse verwechselt werden. Sie repräsentiert ein Dokument aus der Sicht der Document-View-Architektur, also einen Speicher für die Daten, die an der Benutzeroberfläche durch den View dargestellt werden.

<sup>3</sup> Ntf steht als Abkürzung für Notification, also Meldung.

war, wurde der Eventraiser in die Applikationsklasse verlagert und mit einer Meldungswarteschlange versehen. Die wartenden Meldungen werden als Teil der Hintergrundverarbeitung der Applikation asynchron in die Datenbank geschrieben. Damit ist das Programm unmittelbar nach dem Ausführen der LinkWorks-Operation wieder für Eingaben bereit. In einem praxisreifen System sollte die Ereigniserzeugung und Meldungsverteilung auf dem Server ausgeführt werden. Der Eventraiser wird in Abschnitt 7.4.8 genauer beschrieben.

Durch die Mechanismen der MFC wird mit einem Dokumentobjekt der Klasse *CSWSDoc* ein View-Objekt der Klasse *CSWSView* verbunden. Diese zeigt die Icons aller Objekte im Container an, die keine Instanzen der Klasse *CLNXUtr* und abgeleiteter Klassen sind. Außerdem ermöglicht sie eine Interaktion mit den angezeigten Objekten. Weiterhin existiert eine Klasse *CUtrView*, die nur Instanzen von *CLNXUtr* und abgeleiteten Klassen anzeigt. Damit werden die Benutzer des Gemeinsamen Arbeitsbereiches dargestellt.

Zwei Rahmenklassen vervollständigen die Architektur: *CMainFrame* und *CChildFrame*. *CMainFrame* enthält das Menü für die Klasse *CLNXSWSObj* sowie je eine Instanz von *CSWSView* und *CUtrView*. *CChildFrame* stammt von *CMainFrame* ab und enthält das Menü für die Klasse *CLNXCompObj* sowie eine Instanz von *CSWSView*.

Das im Dokument der Klasse *CSWSDoc* eingebettete Containerobjekt der Klasse *CLNXCompObj* speichert einen Zeiger auf das Dokument. Dadurch kann es über das Dokument mit seinen Views und mit der Verbindung zur Meldungsdatenbank *CNtfConsumer* kommunizieren.

Die Applikation enthält weiterhin eine Instanz der Klasse *CLNX*, die die Verbindung zum System LinkWorks und den Aufruf der benötigten LinkWorks-APO-Funktionen einkapselt. In LinkWorks wird jedes Objekt durch eine 64 Zeichen lange Objektreferenz identifiziert, die in jeder Instanz der Klasse *CLNXObj* gespeichert und bei jedem Aufruf einer LinkWorks-APO-Funktion als Parameter benötigt wird.

### 7.4.4 Datenbankeinbindung

Ereignisse, Interesse und Meldungen werden in einer ACCESS-Datenbank abgespeichert. Für jedes dieser Entities gibt es eine Datenbanktabelle. Die Tabellenstruktur wird im folgenden beschrieben. Attribute, die den Primärschlüssel bilden, sind unterstrichen. Attribute, die als Kombination einen alternativen Schlüsselkandidaten darstellen, sind kursiv gedruckt. Ein „X“ in der Spalte „NULL“ bedeutet, daß das entsprechende Attribut einen Nullwert (Leerwert) enthalten kann.

#### a) Tabelle „Events“

Die Datenbanktabelle „Events“ (siehe Tabelle 7.3) speichert die aufgetretenen Ereignisse. Der kleinste Schlüsselkandidat für die Datenbanktabelle besteht aus fünf Attributen. Da der Primärschlüssel als Fremdschlüssel in der Tabelle „Notifications“ auftaucht, wurde zur Vereinfachung der EventID als synthetischer Primärschlüssel eingeführt.

Attribut	Datentyp	NULL	Bemerkung
<u>EvtID</u>	Long Integer		Synthetischer Primärschlüssel (Zähler)
<i>Type</i>	Integer		Ereignistyp
<i>Usr</i>	Long Integer		LinkWorks-UserID des Ereignisinitiators
<i>EvTime</i>	DateTime		Ereigniszeitpunkt
<i>Obj</i>	Text[65]	X <sup>1</sup>	LinkWorks-Objektreferenz des Ereignisobjektes
<i>Parent</i>	Text[65]	X <sup>1</sup>	LinkWorks-Objektreferenz des Elternobjektes des Ereignisobjektes
ClassParam	Text[103]	X	freier Parameter: LinkWorks-Objektklasse
NameParam	Text[33]	X	freier Parameter: Objektname
BOOLParam	Boolean	X	freier Parameter: BOOLEscher Wert

**Tabelle 7.3:** Struktur der Datenbanktabelle „Events“

Das Speichern der Kombination *Obj* und *Parent* in der Datenbanktabelle „Events“ dient der Beschleunigung der Datenbankabfrage, da ein Containerobjekt mit einer Datenbankabfrage alle primären Meldungen über Ereignisse ermitteln kann, die in ihm enthaltene Objekte betreffen.

<sup>1</sup> Attribut ist NULL für Systemereignisse.

### b) Tabelle „InterestPatterns“

In der Datenbanktabelle „InterestPatterns“ werden Interessemuster gespeichert. Auch hier wurde ein synthetischer Schlüssel eingeführt, da der kleinste Schlüsselkandidat aus vier Attributen besteht. Das Einmaligkeitsflag wird im Prototypen nicht unterstützt. Tabelle 7.4 enthält die Struktur der Datenbanktabelle „InterestPatterns“.

Attribut	Datentyp	NULL	Bemerkung
<u>IntID</u>	Long Integer		Synthetischer Primärschlüssel (Zähler)
<u>IntUsr</u>	Long Integer		LinkWorks-UserID des Benutzers, der Interesse angibt
<u>EvType</u>	Integer		Ereignistyp, an dem Interesse besteht
<u>EvtUsr</u>	Long Integer	X <sup>1</sup>	LinkWorks-UserID des Ereignisinitiators
<u>EvtObj</u>	Text[65]	X <sup>2</sup>	LinkWorks-Objektreferenz des Objektes, für das Interesse angegeben wird
Importance	Integer		Wichtigkeit
NtfCtx	Text[65]		LinkWorks -Objektreferenz des Meldekontextes
Inherit	Boolean		Vererbbarkeit des Interesses
Overwrite	Boolean		Überschreiben des Interesses in untergeordneten Containern bei der Vererbung <sup>3</sup>

**Tabelle 7.4:** Struktur der Datenbanktabelle „InterestPatterns“

### c) Tabelle „Notifications“

Die Speicherung der Meldungen erfolgt in der Datenbanktabelle „Notifications“ (siehe Tabelle 7.5). Der Primärschlüssel der Tabelle „Events“ tritt hier als Fremdschlüssel auf. Da auch die Tabelle „Notifications“ einen zusammengesetzten Schlüssel hat, wurde ein synthetischer Primärschlüssel eingeführt.

Attribut	Datentyp	NULL	Bemerkung
<u>NtfID</u>	Long Integer		Synthetischer Primärschlüssel (Zähler)
<u>EvtID</u>	Long Integer		Fremdschlüssel: ID des gemeldeten Events
<u>NtfReceiver</u>	Long Integer		LinkWorks-UserID des Empfängers der Meldung
<u>Importance</u>	Integer		Wichtigkeit der Meldung
InfoStatus	Boolean		Informationsstatus <sup>4</sup>
SnoozeTime	DataTime		Zurückstellungszeit <sup>5</sup>
Context	Text[65]		LinkWorks-Objektreferenz des Meldekontextes
ContextParent	Text[65]		LinkWorks-Objektreferenz des Elternobjektes des Meldekontextes
LocalEcho	Boolean		Lokales oder entferntes Echo <sup>6</sup>

**Tabelle 7.5:** Struktur der Datenbanktabelle „Notifications“

<sup>1</sup> Wenn dieses Feld NULL ist, wird angenommen, daß das Interesse für alle Benutzer gelten soll.

<sup>2</sup> Ist NULL für Interesse an Systemereignissen.

<sup>3</sup> Wird nur berücksichtigt, wenn Inherit=TRUE.

<sup>4</sup> TRUE, wenn Meldung als „gesehen“ markiert wurden.

<sup>5</sup> Enthält den Zeitpunkt, nach dem eine Meldung mit einer ihrer Wichtigkeit entsprechenden dringlicheren Technik dargestellt werden soll. Wird mit frühestmöglicher Systemzeit initialisiert.

<sup>6</sup> Zeigt an, ob der Eintrag eine lokale (TRUE) oder entfernte Meldung (FALSE) repräsentiert. Lokale Meldungen werden nur beim Initialisieren eines Containers nach dem Öffnen geladen.

Das Feld *Context* hat eine Doppelfunktion in Abhängigkeit vom Wert im Feld *Importance*. Enthält letzteres den besonderen Wert *propagate*, so handelt es sich um eine propagierte Meldung, und *Context* enthält das Ziel der Propagierung. Anderenfalls handelt es sich um eine primäre Meldung, und *Context* enthält den Meldekontext der Meldung. Im ersten Fall spielt das Feld *ContextParent* eine Rolle: Es enthält das Elternobjekt des Propagierungszieles. Dieses wird als Selektionskriterium genutzt, wenn ein Container propagierte Meldungen darstellen muß. Dadurch ist es nicht notwendig, für jedes einzelne im Container enthaltene Containerobjekt eine Anfrage nach propagierten Meldungen zu stellen.

#### d) Datenbankzugriff

Wenn eine Ereignismeldung präsentiert werden soll, so wird sie zunächst als INNER JOIN<sup>1</sup> aus den Tabellen *Events* und *Notifications* zusammengesetzt. Listing 7.1 zeigt die SQL-Abfrage, mit der alle primären und propagierten Meldungen für einen Container ermittelt und nach dem Ereigniszeitpunkt sortiert werden.

```
SELECT *
FROM Events INNER JOIN Notifications ON Events.EvtID=Notifications.EvtID
WHERE ((Parent="myContainer" AND Importance<>propagate) OR
       (ContextParent="myContainer" AND Importance=propagate))
       AND NtfReceiver=myUserID
ORDER BY EvTime DESC;
```

**Listing 7.1:** SQL-Abfrage zur Ermittlung aller primären und propagierten Meldungen eines Containers<sup>2</sup>

Der Zugriff auf die Datenbank erfolgt mit Hilfe von *Recordsets* und *direkten SQL-Abfragen*.

Ein Recordset ist eine Klasse der Klassenbibliothek, die den Zugriff auf ODBC-fähige Datenbanken erleichtert. Es repräsentiert eine Liste von Datensätzen als Ergebnis einer SQL-SELECT-Abfrage, auf der Scrolling-, Update-, Insert- und Delete-Operationen möglich sind. Scrolling-Operationen positionieren einen Zeiger auf einen Datensatz, auf den sich dann die anderen Operationstypen beziehen. Operationen auf einem Recordset erfolgen also *datensatzweise*. Es gibt zwei Typen von Recordsets: *Dynasets* und *Snapshots*. Während erstere Änderungen in der Datenbank automatisch reflektieren, stellen letztere eine Momentaufnahme des Datenbankinhaltes dar und müssen durch Aufruf der Funktion *Requery* in gewissen Zeitabständen aktualisiert werden. Die

<sup>1</sup> Ein INNER JOIN ist eine Exclusionsverknüpfung, bei der Datensätze zweier Datenbanktabellen immer dann verknüpft und dem Ergebnis hinzugefügt werden, wenn die Werte der verknüpften Felder gleich sind. Im Ergebnis der INNER-JOIN-Abfrage über die Tabellen *Events* und *Notifications* sind dann sowohl die Felder der Ereignisse als auch die der Meldungen enthalten.

<sup>2</sup> Diese SQL-Abfrage wurde im Interesse der Lesbarkeit syntaktisch nicht korrekt wiedergegeben. Unterstrichener Text muß durch die entsprechenden numerischen Werte bzw. kryptischen LinkWorks-Objektreferenzstrings ersetzt werden.

vorliegende Version des ODBC-Treibers unterstützt leider keine Dynasets, an die neu eintreffende Meldungen automatisch angehängt würden. Daher wird die Datenbank von einer Methode der Klasse *CNtfConsumer* in regelmäßigen Zeitabständen nach neuen Meldungen abgefragt (Polling).

Direkte SQL-Abfragen werden als String an den ODBC-Treiber übergeben und von diesem auf der Datenbank ausgeführt. Zurückgeliefert wird lediglich ein Boolescher Wert, ob die Ausführung erfolgreich war oder nicht. Sie können also nicht eingesetzt werden, wenn ein Abfrageergebnis erwartet wird. Günstig ist ihr Einsatz zum Löschen mehrerer Datensätze, was in einem Recordset satzweise nacheinander erfolgen muß und daher viel Zeit kostet.

Die SQL-Anfrage eines Recordsets kann parametrisiert werden. Eine parametrisierbare Recordset-Instanz kann man für verschiedene Abfragen benutzen. Parametrisierung erfolgt, indem im SQL-String je Parameter ein Platzhalter eingefügt wird, z.B. „UserID=?“. Die Parameterwerte werden in Membervariablen der Recordset-Klasse gespeichert. Der Mechanismus des dynamischen Datenaustausches der Klassenbibliothek ersetzt die Platzhalter zur Laufzeit durch die aktuellen Parameterwerte.

#### **7.4.5 Asynchrone Ausführung von Aktionen auf LinkWorks-Objekten**

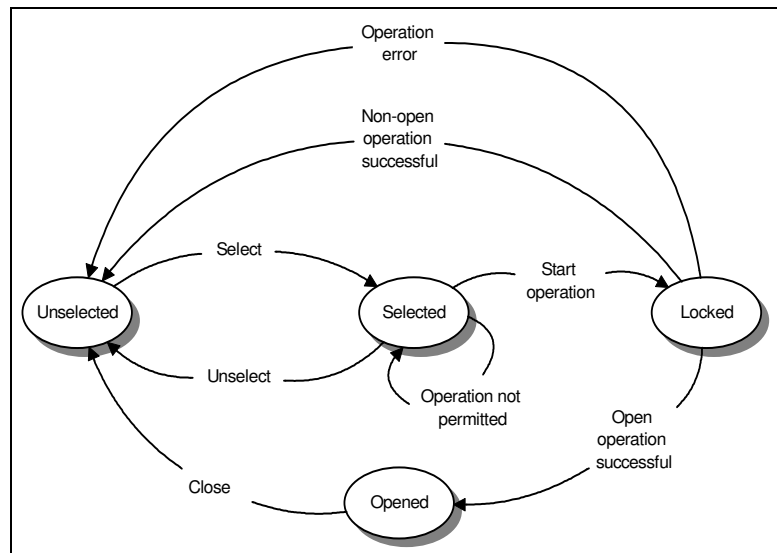
Nach [LUC94] sollte die Benutzerschnittstelle eines Groupwaresystems asynchron sein, damit langandauernde Netzzugriffe nur die betroffenen Objekte sperren, jedoch nicht die gesamte Applikation für Eingaben blockieren.

Die LinkWorks-APO führt alle Aufrufe asynchron aus. Das heißt, beim Aufruf einer APO-Funktion tritt das aufrufende Programm sofort wieder in den Message-Loop ein, während der LinkWorks-Client die Anfrage bearbeitet. Das stellte eine Herausforderung bei der Entwicklung der Benutzerschnittstelle dar. Es mußte verhindert werden, daß eine weitere Operation auf einem Objekt ausgeführt wurde, für das gerade eine Operation lief.

Dazu wurde die Klasse *CLNXObj* um zwei Attribute erweitert:

- den Objektstatus *m\_objState* und
- das Lockticket *m\_lockTicket*.

Der *Objektstatus* stellt dar, in welchem lokalen Interaktionszustand<sup>1</sup> sich ein Objekt gerade befindet. Abbildung 7.4 zeigt das Zustandsübergangdiagramm, Farbtabelle 7.1<sup>2</sup> die entsprechende grafische Repräsentation. Wenn für ein Objekt eine Aktion ausgelöst werden soll (z.B. das Kopieren), so muß



**Abbildung 7.4:** Lokale Zustandsübergänge eines Objektes

es zunächst selektiert werden (Operation *Select*). Über das Menü, über Shortcuts bzw. Mausereignisse können LinkWorks-Aktionen ausgelöst werden, die sich auf alle selektierten Objekte beziehen. Da die Aktionen *asynchron zum Userinterface* ausgeführt werden, könnte der Benutzer während deren Ausführung weitere Objekte selektieren oder bereits selektierte Objekte deselektieren. Um das zu verhindern, müssen die selektierten Objekte zunächst gesperrt werden - sie gehen in den Zustand *Locked* über. Für alle gesperrten Objekte wird dann die angeforderte Aktion ausgeführt. Das erfolgt *sequentiell* - die Aktion auf dem zweiten Objekt beginnt nach dem Ende der Aktion auf dem ersten. Es sind aber aufgrund des asynchronen Userinterfaces sofort nach Beginn der Aktion auf dem ersten Objekt wieder Eingaben im Programm möglich. Folglich könnten andere Objekte selektiert, gesperrt und eine Aktion auf diesen gestartet werden. Das führt zu einem weiteren Problem: Die verschiedenen Gruppen gesperrter Objekte müssen eindeutig unterscheidbar sein. Dazu wird an jedes gesperrte Objekt ein *Lockticket* „angeheftet“, das für alle Objekte der Gruppe gleich ist. Das Lockticket ist eine ganze Zahl, die während der gesamten Ausführungszeit der Applikation eindeutig ist und vom Applikationsobjekt erteilt wird. Die auszuführende Aktion wird dann nur auf solchen gesperrten Objekten ausgeführt, deren Lockticket sie als zur richtigen Gruppe gehörig ausweist.

Tritt bei der Ausführung einer Aktion auf einem gesperrten Objekt ein Fehler auf, so geht es wieder in den Zustand *Unselected* über. Ist die Operation erfolgreich, so gibt es zwei Möglichkeiten. Nach einer erfolgreichen Operation zum Öffnen des Objektes geht es in den Zustand *Open* über, bis es geschlossen wird und somit in den Zustand *Unse-*

<sup>1</sup> Der lokale Interaktionszustand ist der Zustand des Objektes bezogen auf den aktuellen Benutzer. Die möglichen Zustände sind in Abbildung 7.4 als Knoten des Graphen dargestellt.

<sup>2</sup> Die Farbtafeln befinden sich aus drucktechnischen Gründen am Abschnittsende ab Seite **Fehler!**  
**Textmarke nicht definiert..**

lected übergeht. Der Erfolg jeder anderen Operation führt das Objekt in den Zustand *Unselected* zurück.

Durch Inaktivschalten von Menüeinträgen in Abhängigkeit von gewissen Merkmalen selektierter Objekte wird verhindert, daß Aktionen unter nicht erfüllten Vorbedingungen ausgeführt werden. Beim Öffnen eines Menüs ruft die Klassenbibliothek für jeden Eintrag eine Methode auf, die prüft, ob der Eintrag aktivierbar sein soll oder nicht. In dieser Methode wird für alle selektierten Objekte im Container geprüft, ob die mit dem Menüeintrag verbundene Aktion ausführbar ist. Mit Menüeinträgen verknüpfte Shortcuts erhalten den gleichen Aktivierbarkeitsstatus wie diese. Lediglich bei Mausereignissen zur direkten Manipulation (wie Verschieben von Objekten) muß die Gültigkeit in den entsprechenden Ereignisroutinen geprüft werden. Ungültige Aktionen werden so nicht ausgelöst, das Objekt bleibt im Zustand *Selected*.

#### 7.4.6 Trennung von lokalem und entferntem Echo

Eine Anforderung der Direktmanipulation ist es, daß der Benutzer ein unmittelbares Echo auf seine Eingaben erhält. Die Meldungen an entfernte Benutzer laufen über die Datenbank. Der Datenbankzugriff ist jedoch, wie in Abschnitt 7.4.3 beschrieben, so langsam, daß er asynchron ausgeführt werden muß. Daher kann das Echo für den lokalen Benutzer nicht über die Datenbank laufen, sondern muß lokal erzeugt werden. Verzögerungen beim Eintreffen der Meldungen an entfernte Benutzer sind unproblematisch.

Die Einkapselung der LinkWorks-Objektklassen in C++-Objektklassen wurde ebenfalls im Abschnitt 7.4.3 beschrieben. Die Instanzen solcher C++-Klassen sollen zur Unterscheidung von LinkWorks-Objekten als *Speicherobjekte* bezeichnet werden, da sie im Hauptspeicher des lokalen Rechners existieren. *Ghost-Objekte* sind Speicherobjekte, für die es in deren übergeordnetem Container kein entsprechendes LinkWorks-Objekt gibt, da es entweder gelöscht oder verschoben wurde. Speicherobjekte enthalten über die in Abbildung 7.3 dargestellten Membervariablen hinaus noch weitere, die je Ereignistyp oder Gruppe einander ausschließender Ereignistypen die aktuellste Meldung speichern:

- die Lebenszyklusmeldung *m\_pEvLifeCycle*, die die Meldung über das letzte den Lebenszyklus des Objektes beeinflussende Ereignis (Erzeugen, Verschieben, Löschen) enthält,
- die Modifikationsmeldung *m\_pEvModified*, die bei Dokumentobjekten die Meldung über das letzte Modifikationsereignis und bei Containerobjekten die letzte propagierte Meldung enthält,

- die Umbenennungsmeldung *m\_pEvRenamed*, die die Meldung über das letzte Umbenennungsereignis enthält, und
- die Sperrmeldung *m\_pEvCheckedOut*, die die Meldung über das letzte Sperrereignis enthält.

Um schnell ein lokales Echo zu erzeugen, besitzt ein Speicherobjekt Methoden zur Speicherung und Propagierung von Ereignismeldungen. Methoden, die eine ereignis-erzeugende LinkWorks-Aktion einkapseln, rufen im Anschluß an die Aktion zunächst die dem Ereignistyp entsprechende Methode des Eventraisers<sup>1</sup> auf. Diese ermittelt das Interesse an dem Ereignis, schreibt das Ereignis in die Datenbank, plaziert einen Job-Eintrag in der Meldungsqueue des Eventraisers und gibt die an den lokalen Benutzer gerichtete Meldung zurück. Diese wird den Methoden zur Meldungsspeicherung und -propagierung übergeben, die die vier Meldungs-Membervariablen des vom Ereignis beeinflussten Speicherobjektes sowie übergeordneter Speicherobjekte modifizieren. Im Anschluß an diese Modifikation wird die Methode *ColorCoding*<sup>2</sup> aufgerufen, die die neuen Meldungen im Objekticon darstellt. Der Eventraiser schreibt die Meldungen im Hintergrund asynchron in die Datenbank.

Neue Meldungen an den lokalen Benutzer über von entfernten Benutzern ausgelöste Ereignisse werden je geöffnetem Containerobjekt in regelmäßigen Zeitintervallen aus der Datenbank abgefragt. Das erfolgt durch den NtfConsumer des Dokumentobjektes, das mit entsprechenden Containerobjekt assoziiert ist. Er klammert dabei Meldungen über lokale Ereignisse aus und beschränkt das Abfrageergebnis auf Meldungen, die sich auf Objekte im Container beziehen. Die Meldungen werden einzeln der Methode *DispatchNtf* des Speicher-Containerobjektes übergeben, die die Verteilung an die in ihm enthaltenen Speicherobjekte übernimmt. Dazu prüft sie zunächst, ob bereits ein Speicherobjekt mit einer Objektreferenz gleich der des Ereignisobjektes der Meldung existiert. Ist das nicht der Fall, so wird entweder für gelöschte bzw. verschobene Objekte ein neues Ghost-Objekt oder für ein neu erzeugtes Objekt ein normales Speicherobjekt kreiert. Danach wird die Meldung der Methode *Notify* des gefundenen oder erzeugten Speicherobjektes übergeben.

Die Methode *Notify* eines Speicherobjektes prüft, ob die Meldung die aktuellste über den gemeldeten Ereignistyp ist. Ist das der Fall, so wird sie in dem entsprechenden Meldungsattribut des Speicherobjektes abgelegt und die Darstellung des Objekticons durch Aufruf der Methode *ColorCoding* aktualisiert.

---

<sup>1</sup> Der Eventraiser wird in Abschnitt 7.4.8 näher beschrieben.

<sup>2</sup> Siehe Abschnitt 7.4.10.

### 7.4.7 Spezifizieren von Interesse

Interesse für ein Objekt wird spezifiziert, indem in einer Methode des Speicherobjektes eine Dialogbox geöffnet wird, die direkt auf die ACCESS-Datenbank zugreift. Sie ermöglicht ein Editieren der Tabelle „InterestPatterns“. Anhang B beschreibt die Bedienung und enthält Screendumps.

Soll Interesse für auf ein Objekt bezogene Ereignisse angegeben werden, wird zunächst ein Recordset mit einer Abfrage nach allen Interessemustern für das Objekt und den aktuellen Benutzer geöffnet. Das Abfrageergebnis wird in einer Listbox (siehe Abbildung 7.5) dargestellt, in der jede Zeile einem Interesseeintrag entspricht. Die Listbox ermöglicht das *Einfügen* neuer und das *Bearbeiten* oder *Löschen* bestehender Interesseinträge. Das Ergebnis jeder Operation wird sofort in die Datenbank geschrieben.

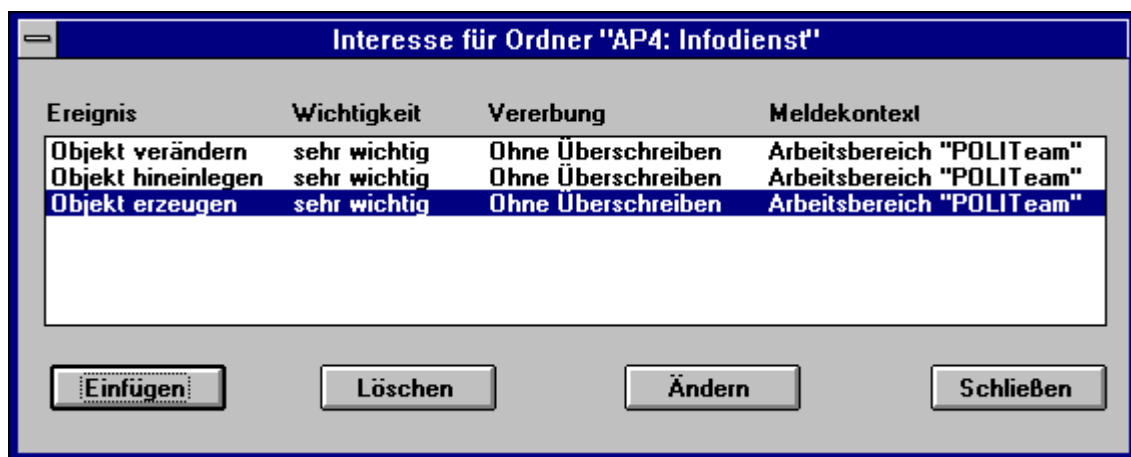


Abbildung 7.5: Liste der Interesse-Einträge für ein Objekt

Beim Einfügen wird dem Benutzer zunächst eine Liste mit Ereignistypen angeboten, für die er Interesse angeben kann. Darin sind nur die Ereignistypen enthalten, für die noch kein Eintrag in der Liste existiert. Für das ausgewählte Ereignis wird ein neuer Eintrag erzeugt, initialisiert und die Funktion *Bearbeiten* für diesen gestartet.

Die Funktion *Bearbeiten* ermöglicht ein Editieren des aktuellen Interesseintrags in einer weiteren Dialogbox. Für ihre Beschreibung wird ebenfalls auf Anlage B verwiesen. Zur Auswahl des Meldekontextes bietet die Editbox dem Benutzer

alle Containerobjekte an, die dem Objekt übergeordnet sind, für das Interesse spezifiziert werden soll. Das Objekt selbst wird ebenfalls in der Liste der potentiellen Meldekontexte angeboten, wenn es ein Containerobjekt ist.

Die für das Spezifizieren von Default-Interesse vorgeschlagenen Ansätze<sup>1</sup> wurden nicht implementiert. Statt dessen wird Default-Interesse als Interesse-Eintrag mit gesetztem Vererbungsflag für das Wurzelobjekt, also den Gemeinsamen Arbeitsbereich, angegeben. Das Default wird so über den normalen Propagierungsweg von Interesse wirksam. Eine weitere Einschränkung des Prototypen besteht darin, daß es nicht möglich ist, für Ereignisse der Typen „Objekt erzeugen“ und „Objekt in ein Containerobjekt hineinlegen“ bei gleichem Elternobjekt unterschiedliches Interesse für unterschiedliche Benutzer zu spezifizieren.

## 7.4.8 Erzeugen von Meldungen

Wie in Abschnitt 7.4.6 beschrieben, erzeugt im Prototypen der Clientprozeß des Ereignisinitiators die Meldungen über ein Ereignis. Der in diesem Abschnitt beschriebene Prozeß ist in Abbildung 5.11 als Transformation *RaiseEvent* dargestellt. Besser als die lokale Ausführung wäre die Auslagerung in einen Serverprozeß, was sich aber aufgrund der losen Integration zwischen Prototyp und LinkWorks nicht realisieren läßt. Die Datenbankzugriffe zur Meldungserzeugung belasten die Performance des Clientprozesses, wenn sie im Vordergrund von der Methode ausgeführt werden, die Ursprung

<sup>1</sup> Vgl. Abschnitt 5.2.2(c).

des Ereignisses ist. Weil ein Ereignis immer durch eine Benutzeraktion initiiert wird, erfolgt der Aufruf dieser Methode als Teil des Message-Loops der Applikation. Daher sind keine Benutzereingaben möglich, während die Meldungen an andere Benutzer erzeugt werden.

Das Konzept der Trennung von lokalem und entferntem Echo ermöglicht es, nur die Meldung für den lokalen Ereignisinitiator synchron zu erzeugen und die Benachrichtigung der entfernten Benutzer einem Hintergrundprozeß zu übertragen.

Alle Funktionen zum Erzeugen von Ereignismeldungen und deren Ablegen in der Datenbank werden von der Klasse *CEventRaiser* eingekapselt. Sie enthält mehrere Recordsets zum Zugriff auf die Datenbank und eine Meldungswarteschlange. Die Einträge der Meldungswarteschlange bestehen aus dem Ereignis und einer Liste des Interesses aller Benutzer an diesem Ereignis. Von den Methoden der Klasse sollen nur die für das Verständnis ihrer Funktion notwendigen erwähnt werden. Die folgenden öffentlichen Methoden sind implementiert:

- *RaiseXXXXEvent*: ist je Ereignistyp XXXX einmal vorhanden, erzeugt ein Ereignis, speichert es in der Datenbank, ermittelt das Interesse, stellt den entsprechenden Eintrag in die Warteschlange und gibt die an den lokalen Benutzer gerichtete Meldung zur Erzeugung des lokalen Echos zurück
- *AddNtf*: schreibt eine Meldung direkt in die Datenbank
- *RemovePropagatedNtfs*: löscht zu einem zusammengesetzten Objekt propagierte Meldungen für einen bestimmten oder für alle Benutzer
- *DoOneStepAsync*: stellt eine Zeitscheibe dar, die einen Schritt beim asynchronen Abspeichern der Meldungen in der Datenbank ausführt
- *DoAllStepsAsync*: speichert alle Meldungen in der Warteschlange in der Datenbank ab

Geschützte Methoden implementieren die von außen nicht ansprechbare Funktionalität:

- *FindInterest*: sucht in der Datenbank für jeden zugriffsberechtigten Benutzer nach dem Interesse an einem eingetretenen Ereignis
- *RaiseEvent*: schreibt ein Ereignis in die Datenbank und gibt dessen ID zurück
- *CreateNtf*: ruft für ein übergebenes Ereignis *FindInterest* auf, erzeugt aus dem gefundenen Interesse und dem Ereignis einen Eintrag in der Meldungswarteschlange und gibt das Interesse des lokalen Benutzers zur Erzeugung des lokalen Echos zurück

- *KillNtf*: entfernt alle Meldungen über das jeweils komplementäre Verschiebe- bzw. aktivitätsbegrenzende Ereignis<sup>1</sup>
- *Thinning*: implementiert das Ausdünnen von Meldungen

Die Methode *FindInterest* wird synchron ausgeführt. Sie wäre ebenfalls ein Kandidat für eine asynchrone Ausführung, aber Zeitmessungen haben ergeben, daß für die im Prototypen anfallenden Datenmengen eine synchrone Ausführung ausreichend ist. Lesende Zugriffe auf die Datenbank erwiesen sich als relativ schnell. *FindInterest* implementiert den in Abschnitt 5.2.2 (c)) entwickelten Algorithmus zum Propagieren von Interesse ohne den ergänzenden bzw. einschränkenden Ansatz zum Einsetzen von Default-Werten. Als Default-Werte werden statt dessen die Interesse-Einträge des Gemeinsamen Arbeitsbereiches genutzt.

Die Methoden *AddNtf* und *RemovePropagatedNtfs* werden von *DoOneStepAsync* aufgerufen, jedoch auch zum Repropagieren von Meldungen benötigt. Die Methode *DoOneStepAsync* ist der eigentliche Kern des Eventraisers. In Listing 7.2 ist der synchrone Grundalgorithmus dargestellt. Das Erzeugen aller Meldungen für einen Warteschlangeneintrag ist zu zeitaufwendig, um in einer Zeitscheibe erledigt zu werden. *DoOneStepAsync* implementiert daher den Algorithmus derart, daß jeweils nach Ausführung eines Schrittes die Steuerung an die aufrufende Funktion zurückgegeben wird.

```

METHOD CEventRaiser :: StoreNotifications()
VAR   entry          : NtfQueueEntry;
      i              : INTEGER;
      propagationTarget : PointerToMemObject;
BEGIN
  WHILE NOT m_ntfQueue.IsEmpty()
  DO
    entry:=m_ntfQueue.GetAndRemoveHead();
    FOR i:=1 TO entry.CountInterestedUsers()
    DO
      AddNtf(entry.m_event, entry.m_importance[i], entry.m_ntfContext[i]);
      IF RequiresPropagation(entry.m_event.m_type)
      THEN
        propagationTarget:=entry.m_event.m_eventObject->GetParent();
        WHILE (propagationTarget<>NULL)
        DO
          AddNtf(entry.m_event, propagate, propagationTarget);
          IF i=entry.CountInterestedUsers()
          THEN
            RemovePropagatedNtfs(propagationTarget, ALLUSERS);
          ENDIF
          propagationTarget:=propagationTarget->GetParent();
        ENDWHILE
      ENDIF
    ENDFOR
    Thinning(entry.m_event.m_type, entry.m_event.m_eventObject);
  ENDWHILE
END

```

**Listing 7.2:** Synchroner Algorithmus des Abspeicherns von Meldungen in der Datenbank

<sup>1</sup> Wenn beispielsweise ein Login-Ereignis aufgetreten ist, so werden vor dem Erzeugen der Meldung über dieses alle Meldungen über das letzte Logout-Ereignis des entsprechenden Benutzers entfernt.

Solange sich Einträge in der Warteschlange befinden, wird der erste Eintrag entnommen. Für jeden interessierten Benutzer in diesem Eintrag wird von der Methode *AddNtf* nun eine primäre Meldung über das Ereignis mit den im Interesse angegebenen Attributen Wichtigkeit und Meldekotext erzeugt. Falls das Ereignis - abhängig vom Typ - in übergeordnete Kontexte propagieren soll, werden je Benutzer und übergeordnetem Objekt außerdem propagierte Meldungen erzeugt. Dazu wird die Wichtigkeit der Meldung auf *propagate* gesetzt und als Meldekotext das Propagierungsziel *propagateTarget* angegeben. Nach dem Speichern der propagierten Meldung an den letzten Benutzer in der Liste wird die Funktion *RemovePropagatedNtfs* für das Propagierungsziel mit dem Parameter *ALLUSERS* aufgerufen. Sie ermittelt die aktuellste propagierte Meldung und löscht dann alle älteren propagierten Meldungen des Propagierungszieles. Dieselbe Aufgabe erfüllt die Funktion *Thinning* für Meldungen über ältere Ereignisse, die vom selben Typ sind wie das aktuell gemeldete. Sie löscht alle Meldungen über solche Ereignisse, die sich auf das Ereignisobjekt des gemeldeten Ereignisses beziehen.

*DoAllStepsAsync* ruft *DoOneStepAsync* in einer WHILE-Schleife auf, bis die Meldungswarteschlange leer ist. Das ist notwendig, wenn der Zustand des lokalen Prozesses mit der Datenbank synchronisiert werden muß. Eine Anwendung dafür ist das Schließen eines Containerobjektes. Dabei werden die darin befindlichen Speicherobjekte gelöscht. In der Meldungswarteschlange sind die Ereignisobjekte als Zeiger auf Speicherobjekte abgelegt. Daher muß diese vor dem Löschen der Speicherobjekte geleert werden, um die Entstehung ungültiger Zeiger zu verhindern.

### 7.4.9 Konsumieren von Meldungen

Meldungen, die sich in der Datenbank befinden, müssen dem lokalen Benutzer dargestellt werden. Das betrifft sowohl alle anstehenden Meldungen an diesen Benutzer beim Öffnen des Gemeinsamen Arbeitsbereiches (asynchronous awareness) als auch neue Meldungen über parallel zu dessen Arbeit von anderen Benutzern ausgelöste Ereignisse (synchronous awareness)<sup>1</sup>. Weiterhin muß der Benutzer die Möglichkeit haben, Meldungen zu quittieren, als „gesehen“ zu markieren oder zurückzustellen.

Diese Funktionalität wird von der Klasse *CNtfConsumer* bereitgestellt. Jede Instanz der Klasse *CSWSDoc* enthält eine Instanz obiger Klasse, die sich um das Laden und Verteilen der Meldungen für die im Containerobjekt des Dokuments enthaltenen Speicherobjekte kümmert. Ein geöffnetes Containerobjekt dient somit als Filter für die Meldungen, die sich auf in ihm enthaltene Objekte beziehen. Damit wird sichergestellt, daß nur wirklich für die Darstellung im Containerobjekt benötigte Meldungen aus der Daten-

---

<sup>1</sup> Vgl. Abschnitt 3.3

bank geladen werden. Außerdem ist die Verteilung der Meldungen an die Speicherobjekte einfacher. Um das von einer Meldung betroffene Objekt zu finden, muß die darin gespeicherte Objektreferenz mit der aller Speicherobjekte verglichen werden. Die Einschränkung des Suchraumes auf einen Container verringert die Anzahl erfolgreicher Vergleiche.

Die Klasse *CNtfConsumer* enthält zwei parametrisierte Recordsets:

- ein Recordset *m\_qry*, das alle Meldungen über Ereignisse selektiert, die sich auf Objekte im Container beziehen
- ein Recordset *m\_ctxQry*, das alle Meldungen über von entfernten Benutzern ausgelöste Ereignisse selektiert, für die der Container den Meldekontext darstellt<sup>1</sup>.

```
SELECT *
FROM Events INNER JOIN Notifications ON Events.EvtID=Notifications.EvtID
WHERE Context="myContainer" AND Importance=very_important
      AND NtfReceiver=myUserID AND LocalEcho=FALSE
      AND InfoStatus=FALSE AND SnoozeTime<=currentTime
ORDER BY Parent, EvTime DESC;
```

**Listing 7.3:** SQL-Abfrage zur Ermittlung aller „sehr wichtigen“ Meldungen mit dem Meldekontext „myContainer“

Die Darstellung der Meldungen in einem Container geschieht in zwei Phasen, die in Anlehnung an die Awarenessmodi als *asynchrone* und *synchrone* Phase bezeichnet werden sollen. Für die asynchrone Phase wird das Recordset *m\_qry* beim Öffnen des Containers zunächst so parametrisiert<sup>2</sup>, daß alle Meldungen für Objekte im Container erfaßt werden. In der Membervariablen *m\_lastNtfID* wird nach erfolgter Abfrage der höchste Wert des Meldungs-IDs, also des Datenbankfeldes *NtfID*, abgespeichert. Die Methode *ProcessAllNtfs* wird aufgerufen, um die Meldungen zu verarbeiten. Weiterhin wird die Methode *ShowNtfsInMsgBox* aufgerufen, um die sehr wichtigen Meldungen in der Meldebox<sup>3</sup> darzustellen. Damit ist die asynchrone Phase beendet.

```
SELECT *
FROM Events INNER JOIN Notifications ON Events.EvtID=Notifications.EvtID
WHERE ((Parent="myContainer" AND Importance<>propagate)
      OR (ContextParent="myContainer" AND Importance=?))
      AND NtfReceiver=myUserID
ORDER BY NtfID DESC;
```

**Listing 7.4:** SQL-Abfrage zur Ermittlung aller Meldungen für Objekte in einem Container

In der Methode *ProcessOneNtfAsync* wird jeweils eine Meldung dem Recordset entnommen und an die Methode *DispatchNtfs* des Containerobjektes weitergereicht.

<sup>1</sup> Siehe Listing 7.3 für die unterliegende SQL-Abfrage.

<sup>2</sup> Siehe Listing 7.4 für die unterliegende SQL-Abfrage.

<sup>3</sup> Siehe Abschnitt 7.4.11.

Diese übernimmt die Verteilung an die enthaltenen Speicherobjekte. Die Methode *ProcessAllNtfs* ruft *ProcessOneNtfAsync* in einer Schleife auf, bis alle Meldungen verarbeitet sind.

```
SELECT *
FROM Events INNER JOIN Notifications ON Events.EvtID=Notifications.EvtID
WHERE ((Parent="MyContainer" AND Importance<>propagate AND Importance<>uninteresting)
      OR (ContextParent="MyContainer" AND Importance=propagate))
      AND NtfReceiver=myUserID AND NtfID>m_lastNtfID AND LocalEcho=FALSE
ORDER BY NtfID DESC;
```

**Listing 7.5:** SQL-Abfrage für neu eintreffende Meldungen

Zur Vorbereitung der synchronen Phase wird das Recordset *m\_qry* geschlossen und mit neuen Parametern<sup>1</sup> erneut geöffnet. Der Parameter *m\_lastNtfID* dient beim nachfolgenden *Polling* dazu, daß nur noch neu eintreffende Meldungen geladen werden. In regelmäßigen Zeitabständen wird von der Applikation für jedes offene Dokument die Methode *DoTimeSlice* aufgerufen, die ihrerseits die Methode *TimeSlice* des entsprechenden *NtfConsumer*s aktiviert. Diese ruft die Methode *ProcessOneNtfAsync* auf, falls sich im Recordset unverarbeitete Einträge befinden. Ansonsten wird, wenn ein bestimmtes Zeitintervall seit dem letzten *Polling* verstrichen ist, die Datenbank wiederum abgefragt. Falls neue Meldungen mit der Wichtigkeit „sehr wichtig“ eingetroffen sind, wird erneut die Meldebox zu deren Darstellung geöffnet. Im Prototypen verfügt jeder *NtfConsumer* über eine eigene Meldebox<sup>2</sup>.

Die Meldebox<sup>3</sup> greift über das Recordset *m\_ctxQry* direkt auf die Datenbank zu. Sie ermöglicht das Zurückstellen von Meldungen und ihre Markierung als „gesehen“. Die Funktionalität dafür stellt die Klasse *CNtfConsumer* in Form der Methoden *SnoozeNtf* und *SeenNtf* zur Verfügung. Als „gesehen“ markierte Meldungen enthalten im Datenbankfeld *InfoStatus* den Wert TRUE und werden daher bei der nächsten Abfrage der Datenbank durch *m\_ctxQry* nicht mehr geladen. Anders verhält sich das mit zurückgestellten Meldungen. In deren Datenbankfeld *SnoozeTime* wird die aktuelle Systemzeit plus drei Minuten<sup>4</sup> eingetragen, das Feld *InfoStatus* bleibt unverändert. Dadurch werden sie bei einer erneuten Datenbankabfrage nach drei Minuten als Erinnerung angezeigt.

<sup>1</sup> Siehe Listing 7.5 für die unterliegende SQL-Abfrage.

<sup>2</sup> Günstiger wäre das Weiterreichen der Meldungen an eine applikationsweite Präsentationsinstanz. Zur Demonstration des Grundprinzips reicht jedoch die vorgestellte Implementation aus.

<sup>3</sup> Siehe Abschnitt 7.4.11.

<sup>4</sup> Im Prototypen wurde das Zeitintervall zu Demonstrationszwecken auf zwei Minuten festgelegt. In einem praktisch nutzbaren System müßte es benutzerdefinierbar sein.

### 7.4.10 Präsentationstechnik „Farbcodierung“

Die Präsentationstechnik „Farbcodierung im Objekticon“ wurde vollständig gemäß dem Entwurf in Abschnitt 6.3.2 als Methode *ColorCoding* der Speicherobjekte implementiert.

Die Methode greift auf die Meldungsmembervariablen<sup>1</sup> und auf den lokalen Status<sup>2</sup> des Speicherobjektes zu und codiert diese in die Farbe der verschiedenen Teile des Objekticons<sup>3</sup>. Die Icon-Bitmaps werden aus den Ressourcen der ausführbaren Datei geladen. Verschiedene Teile der Icon-Bitmap sind zur Unterscheidung verschieden eingefärbt. Tabelle 7.6 enthält die Zuordnung. Die Abbildung der resultierenden Farben auf die Farben der Icon-Bitmap erfolgt in der Methode *SetBitmapColors* durch pixelweises Ersetzen<sup>4</sup>. Auch die Bitmaps zur Überlagerung des Objekticons werden aus den Ressourcen geladen und durch pixelweises Ersetzen in die Icon-Bitmap eingemischt.

Teil der Icon-Bitmap	Farbe
Hintergrund, opak	hellgrau
Hintergrund, transparent	magenta
Rahmen, schwarz	rot
Rahmen, weiß	gelb
Sinnbild, schwarz	schwarz
Sinnbild, weiß	weiß
Sinnbild, dunkel	dunkelgrau
Sinnbild, hell	cyan

**Tabelle 7.6:** Farbliche Unterscheidung der Teile einer Icon-Bitmap

Die virtuelle Methode *ColorCoding* für die grundlegende Objektklasse *CLNXObj* ist als Listing 7.6 abgedruckt. Sie kann für abgeleitete Objektklassen überschrieben werden. Die Klasse *CLNXUsr* beispielsweise definiert eine eigene Methode, da die Farbcodierung hier anders gehandhabt wird als bei einem Container oder Dokument<sup>5</sup>.

```
// Farbcodierung anstehender Ereignisse
// chgBitmap==FALSE: Bitmapänderungen werden nicht ausgeführt
// update steuert die Aktualisierung der Views
// => auf FALSE setzen bei Initialisierung, da Views noch nicht existieren

void CLNXObj :: ColorCoding(BOOL chgBitmap, BOOL update)
{
    UINT overlay=OVR_NONE;           // ID der Overlay-Bitmap
                                     // OVR_NONE: spezielle Konstante für „keine“
    COLORREF ovrFg=DEF_OVR_FG;       // Vordergrundfarbe Overlay-Bitmap
    COLORREF bkg=DEF_BKG;            // Hintergrundfarbe
    COLORREF lightFg=DEF_SHP_LIGHT;  // helle Vordergrundfarbe Sinnbild
    COLORREF darkFg=DEF_SHP_DARK;    // dunkle Vordergrundfarbe Sinnbild

    COLORREF tbkg=DEF_TEXT_BKG;      // Texthintergrund
    COLORREF tcol=DEF_TEXT_COL;      // Textvordergrund
    COLORREF frmcol=DEF_BKG;         // Umrahmungsfarbe
    COLORREF dummy;                  // Hilfsvariable
}
```

<sup>1</sup> Siehe Abschnitt 7.4.6.

<sup>2</sup> Siehe Abschnitt 7.4.5.

<sup>3</sup> Siehe Abschnitt 6.3.2 (g).

<sup>4</sup> Durch die Verwendung von Masken, die beim Login-Vorgang aus den Ressourcen der Icon-Bitmaps generiert werden, ließen sich hier Performancegewinne erzielen, da Rasteroperationen häufig in Hardware unterstützt werden.

<sup>5</sup> Siehe Abschnitt 6.3.2 (g)

```

BOOL showOverlayOnly=FALSE;        // TRUE, wenn Overlay-Bitmap alle anderen
                                   // Meldungsdarstellungen unterdrücken soll

CTime time(0,0);                   // Hilfsvariable

// Name des Benutzers, der Objekt gesperrt hat
m_Icon.SetTopName(m_editUserName);

// Existiert Interesse? => Ausrufezeichen dargestellt
m_Icon.SetInterestFlag(m_bInterest);

// Codieren der Zeit des letzten Umbenennungsereignisses in den Texthintergrund
if(m_pEvRenamed) time=m_pEvRenamed->m_time;
// Time2Color ordnet Zeit in Intervall ein und liefert entsprechende Farbcodes
if(time!=CTime(0,0)) ::Time2Color(time, tbkg, dummy);

// Textfarben setzen
// Wenn Objekt Cursor hat, Textfarben vertauschen
if (m_bPointed) m_Icon.SetTextColors(tbkg, tcol);
else m_Icon.SetTextColors(tcol, tbkg);

// Codieren des letzten Lebenszyklusereignisses
if(m_pEvLifeCycle)
{
    TEvent evType=m_pEvLifeCycle->GetType();
    time=m_pEvLifeCycle->m_time;
    switch(evType)
    {
        case ev_del:                // Objekt wurde gelöscht
        {
            overlay=IDB_DELETED;    // ID der Overlay-Bitmap „durchgestrichen“
            showOverlayOnly=TRUE;   // Ghost-Objekt => unterdrücke andere Meld.
            ::Time2Color(time, ovrFg, dummy); // Codiere Zeit in Overlay-Farbe
            break;
        };
        case ev_movefrom:          // Objekt wurde herausgenommen
        {
            overlay=IDB_MOVED_FROM; // ID der Overlay-Bitmap „Pfeil“
            showOverlayOnly=TRUE;   // Ghost-Objekt => unterdrücke andere Meld.
            ::Time2Color(time, ovrFg, dummy); // Codiere Zeit in Overlay-Farbe
            break;
        };
        case ev_moveto:            // Objekt wurde hineingelegt
        case ev_create:            // Objekt wurde erzeugt
        {
            ::Time2Color(time, frmcol, dummy); // Codiere Zeit in Umrahmungsfarbe
            break;
        }
    };
}

// Unterdrücke Textfarbe wenn nur OverlayBitmap angezeigt werden soll
// Die bitmapbezogene „Unterdrückung“ übernimmt SetBitmapColors
if(showOverlayOnly)
{
    if (m_bPointed) m_Icon.SetTextColors(DEF_TEXT_BKG, DEF_TEXT_COL);
    else m_Icon.SetTextColors(DEF_TEXT_COL, DEF_TEXT_BKG);
}

if(chgBitmap) // Modifiziere Bitmap
{
    if(m_pEvModified) // Objekt wurde modifiziert
    {
        time=m_pEvModified->m_time;
        ::Time2Color(time, lightFg, darkFg); // Codiere Zeit in Sinnbild-Farbe
    }

    if(!m_editUserName.IsEmpty()) // Objekt wird editiert
    {
        overlay=IDB_ACCUSER;        // ID der Overlay-Bitmap „Benutzer“
        ovrFg=OVR_FG_OTHERUSER;    // Default: Entfernter Benutzer editiert
    }
}

```

```

// Codiere lokalen Objektstatus
switch(m_objState)
{
    case ost_open:                // Objekt geöffnet
    {
        bkg=COL_BKG_OPEN;        // Codierung in Hintergrundfarbe
        ovrFg=OVR_FG_THISUSER;   // Farbe des Overlays für lokalen Benutzer
        break;
    }
    case ost_locked:              // Objekt vom Rechner gesperrt
    {
        overlay=IDB_ACCOMPUTER;   // ID der Overlay-Bitmap „Computer“
        bkg=COL_BKG_OPEN;        // Codierung in Hintergrundfarbe
        break;
    }
    case ost_selected:           // Objekt selektiert
    {
        bkg=COL_BKG_SELECTED;    // Codierung in Hintergrundfarbe
        break;
    }
    case ost_unselected:
    {
        break;
    }
    default: ASSERT(FALSE); // Zum Testen: Zweig darf nicht erreicht werden
}

// Bitmap einfärben
m_Icon.SetBitmapColors(overlay, bkg, lightFg, darkFg, frmcol,
                       showOverlayOnly, ovrFg);
}

// Aktualisieren des Views des Elternobjektes
if(m_Parent && update)
{
    // Dokumentobjekt holen
    if(((CLNXCompObj *)m_Parent)->GetDocument()!=NULL);
    // Alle View-Objekte des Dokumentes aktualisieren
    ((CLNXCompObj *)m_Parent)->GetDocument()->UpdateAllViews(NULL, 0, NULL);
}
}

```

**Listing 7.6:** Die Methode *ColorCoding* für die Objektklasse *CLNXObj*

### 7.4.11 Präsentationstechnik „Meldebox“

Die Präsentationstechnik „Meldebox“ dient zum einen der dringlichen Präsentation von Meldungen, zum anderen der Bereitstellung von Interaktionsmöglichkeiten, um den Zustand einer Meldung zu verändern und damit ihren Lebenszyklus zu beeinflussen. Entsprechend dieser beiden Zielstellungen existieren auch zwei verschiedene Ausprägungen von Meldeboxen: der systeminitiiert aufgerufene *Meldedienst* und der benutzerinitiiert aufgerufene Informationsdienst, der sich in die *Meldungsübersicht* und die *Objektmeldungen* gliedert. Für die Beschreibung der Benutzeroberfläche wird auf Anlage B verwiesen.

Der Meldedienst wird vom *NtfConsumer* des entsprechenden Meldekontextes aufgerufen und greift über ein Recordset direkt auf die Datenbank zu. Er kann nicht die maximale Interaktivität zur Verfügung stellen, da das der Meldung unterliegende Speicherobjekt zur Zeit der Meldungsdarstellung nicht in jedem Fall existiert. Das liegt daran, daß eine Meldung in der Meldebox dargestellt wird, sobald ihr Meldekontext geöffnet ist. Ein Speicherobjekt zu einem *LinkWorks*-Objekt existiert jedoch erst, wenn

der Container geöffnet ist, in dem sich das Objekt befindet. Ein geöffneter Meldekontext bedeutet aber nicht unbedingt, daß auch das Elternobjekt des Ereignisobjektes der entsprechenden Meldung geöffnet ist<sup>1</sup>.

Insbesondere das *Quittieren* einer Meldung ist nicht möglich, da für die nachfolgende Neupropagierung der Teilbaum der Containerhierarchie oberhalb des Ereignisobjektes der quittierten Meldung benötigt würde. Es ist sicher machbar, diesen für den Zweck der Repropagierung im Hauptspeicher aufzubauen, aber der Aufwand wurde für den Prototypen gescheut. Aus demselben Grunde wurde auch die Funktion „Objekt anzeigen“ nicht implementiert, die das Elternobjekt des Ereignisobjektes öffnet und das Ereignisobjekt selektiert. Die anderen Operationen (*Alle Gesehen*, *Zurückstellen* und *Details*) kommen mit den in der Datenbank abgelegten Informationen aus und sind daher beim Meldedienst verfügbar.

Der Informationsdienst wird vom Benutzer entweder für ein selektiertes Objekt (Objektmeldungen) oder für alle Objekte in einem Container (Meldungsübersicht) aufgerufen. Obwohl die Dialogbox auch in diesem Fall direkt auf die Meldungen in der Datenbank zugreift, wird sie hier von einer Methode eines Speicherobjektes aufgerufen. Damit ist der Bezug zu diesem Speicherobjekt hergestellt und das Quittieren von Meldungen möglich. Die aufrufende Methode kann nach dem Schließen der Dialogbox anhand von Attributwerten des Dialogbox-Objektes feststellen, ob eine Meldung quittiert wurde. In diesem Fall initiiert sie die Repropagierung der Ereignismeldungen. Im Gegensatz zum Meldedienst gestattet der Informationsdienst nicht das Markieren von Meldungen als „gelesen“ und deren Zurückstellung, da diese Funktionen den Meldedienst steuern und somit auch nur aus ihm erreichbar sind.

## 7.5 Wertung

Nach einer Analyse der Erweiterungsmöglichkeiten des bestehenden Systems Link-Works und der Erkenntnis, daß sie den Anforderungen nicht genügen, wurde eine unvollständige prototypische Lösung gefunden. Der Prototyp veranschaulicht die Ideen, ermöglicht ihre Wertung und beweist die Umsetzbarkeit des Konzeptes. Er stellt eine neuartige Lösung für das Problem der relevanzabhängigen Darstellung von Meldungen über Benutzeraktivitäten in einem gemeinsamen Arbeitsbereich dar. Bisher existiert nach Kenntnis des Autors kein System, in dem eine solche Darstellung abhängig von einer benutzerdefinierten Relevanz möglich ist. Der Prototyp kann als Diskussionsgrundlage für Anforderungsspezifikation und User-Interface-Design dienen, wenn auf

---

<sup>1</sup> Wie bei der Einführung des Interesse-Begriffs dargelegt, ist der Meldekontext ein dem Ereignisobjekt in der Containerhierarchie übergeordneter Container.

der Basis einer zukünftigen LinkWorks-Version eine praktisch anwendbare Lösung entwickelt wird. Somit wurden die in Abschnitt 7.2 gesteckten Ziele erreicht.

Es darf jedoch nicht übersehen werden, daß die Pflege einer zweiten Datenbank neben LinkWorks eine Insellösung darstellt und zu Kompatibilitätsproblemen sowie einem hohen Aufwand für die Konsistenzsicherung führt. Die notwendige Nachimplementierung des LinkWorks-Userinterfaces und die Kapselung der LinkWorks-Objektklassen sowie -Methoden wurden nur teilweise und als „Sparversion“ durchgeführt, wodurch die Offenheit des Systems LinkWorks verlorengelht. Durch das Stützen auf Microsoft-Windows-Standards (MFC, ODBC) ist die Lösung nicht portabel.

---

## 8 Schlußbetrachtungen

---

Die vorliegende Arbeit untersuchte Möglichkeiten zur relevanzabhängigen Darstellung von Meldungen über Benutzeraktivitäten in einem gemeinsamen Arbeitsbereich. Diese fand bisher kaum Beachtung in der Literatur und in existierenden Systemen. Es wurde festgestellt, daß o.g. Meldungen die Effektivität kooperativen Arbeitens in virtuellen gemeinsamen Arbeitsbereichen erhöhen können. Der Benachrichtigungsdienst des existierenden Groupwaresystems LinkWorks Version 3.0 wurde analysiert und sein Funktionsumfang als nicht ausreichend erkannt.

Ausgehend von der Analyse entwickelte der Autor ein neuartiges Modell zur Erzeugung von Meldungen. Aktivitäten eines Benutzers bestehen im Zugriff auf die gemeinsam genutzten Objekte, wodurch Ereignisse ausgelöst werden. Meldungen über diese Ereignisse werden an die Benutzer des gemeinsamen Arbeitsbereiches gesandt. Das Modell trennt konsequent zwischen Ereignissen und den Meldungen über diese Ereignisse. Die Relevanz einer Meldung wird von Interessensmustern bestimmt, die entweder vom Meldungsempfänger definiert oder als Default vorgegeben sein können. Der Empfänger der Meldung kann mit dieser interagieren, um zusätzliche Information abzufragen oder die Relevanz der Meldung zu verändern. Meldungen werden durch Kontexte gefiltert und vom Kontext ihrer Entstehung in übergeordnete Kontexte propagiert. Ein Kontext ist dabei ein Containerobjekt in der LinkWorks-Objekthierarchie.

In Abhängigkeit von der Relevanz und dem Zustand der Meldung erfolgt ihre Präsentation. Fünf Präsentationstechniken mit abgestufter Dringlichkeit wurden vorgeschlagen. Der Schwerpunkt lag hier auf einer Erweiterung des Grafikalphabetes der Icons des bestehenden Systems LinkWorks, die die Darstellung von Meldungen als Teil der Iconrepräsentation der Objekte im System ermöglicht. Die Erkennbarkeit einer Darstellung hängt auch von den Wahrnehmungsfähigkeiten des jeweiligen Benutzers und von den Darstellungsfähigkeiten der verwendeten Hard- und Software ab. Die vorliegende Arbeit könnte durch die Entwicklung alternativer Darstellungstechniken für nicht farbtüchtige Menschen oder für Rechnersysteme mit eingeschränkten Darstellungsfähigkeiten weitergeführt werden. Auch eine formale Beschreibung der Eigenschaften der Präsentationstechniken wäre ein lohnenswertes Forschungsgebiet.

Schließlich wurden die technischen Möglichkeiten zur Erweiterung des Systems LinkWorks um Mechanismen zur Meldungserzeugung und -präsentation untersucht. Dabei stellte sich heraus, daß die Programmierschnittstellen des Systems nicht ausreichen, um die Erweiterung zu implementieren. Zur Umsetzung und Veranschaulichung der Konzepte wurde ein Prototyp unter Nutzung einer externen Datenbank erstellt. Er dient als Diskussionsgrundlage für Anforderungsspezifikation und Design des Benachrichtigungs- und Informationsdienstes im Projektkonsortium und mit den Anwendungspartnern. Die Architektur des Prototypen bringt jedoch einen hohen Aufwand zur Konsistenzsicherung der Daten und Verluste an Offenheit mit sich und kommt daher für ein praxisreifes System nicht in Frage.

Als nächster Schritt nach der Implementation des Prototypen sollte sich eine Validierung des Konzeptes mit Systemanwendern und seine eventuelle Anpassung anschließen. Danach würde die Neuimplementierung unter Beachtung der Richtlinien der Software-Qualität folgen. Aufgrund der Probleme mit der Erweiterbarkeit von LinkWorks sollte diese entweder dem Hersteller von LinkWorks vorgeschlagen oder bis zur Verfügbarkeit einer verbesserten, offeneren Version des Systems verschoben werden.

---

# Literaturverzeichnis

---

- [BAE91] Baecker, R. et al.: „*Bringing Icons to Life*“, Proceedings of the CHI '91, ACM, New York, 1991, Seite 1-6
- [BEA92] Beaudouin-Lafon, M.; Karsenty, A.: „*Transparency and Awareness in a Real-Time Groupware System*“, in: Proceedings of UIST '92, ACM, (1992), Seite 171-180
- [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)
- [BOR91] Borning, A.; Travers, M.: „*Two Approaches to Casual Interaction Over Computer and Video Networks*“, in: Proceedings of the CHI '91, ACM, New York, 1991, Seite 13-19
- [BUC92] Bucheit, M.: „*Windows Programmierhandbuch*“, SYBEX-Verlag, Düsseldorf, 1992
- [CAR91] Card, K. S. et al.: „*The Information Visualizer, an Information Workspace*“, in: Proceedings of the CHI '91, ACM, New York, 1991, Seite 181-188
- [DEC94] Digital Equipment Corporation: Dokumentation zum Groupware-System LinkWorks™ Version 3.0.1, 1994
- [DEW93] Dewan, P.; Choudhary, R.: „*Primitives for Programming Multi User Interfaces*“, in: Baecker, R.M. (ed.): „*Readings in Groupware and CSCW*“, Morgan Kaufman Publishers, San Mateo, 1993, Seite 681-690
- [DOU92a] Dourish, P.; Bellotti, V.: „*Awareness and Coordination in Shared Workspaces*“, in: Proceedings of ACM CSCW '92 Conference on Computer-Supported Cooperative Work, (1992), Seite 107-114

- [DOU92b] Dourish, P.; Bly, S.: „*Portholes: Supporting User Awareness in a Distributed Work Group*“, in: Proceedings of CHI '92, ACM, (1992), Seite 541-547
- [FAH93] Fahlèn, L. E. et al.: „*A Space Based Model for User Interaction in Shared Synthetic Environments*“, in: Proceedings of INTERCHI '93, ACM, (1993), Seite 43-48
- [FUC94] Fuchs, L; Prinz, W.: „*Supporting User Awareness with Local Event Mechanisms in the GroupDesk System*“, noch nicht veröffentlicht, Quelle: GMD, Schloß Birlinghoven, 53754 St Augustin, email: prinz@gmd.de
- [GAV93] Gaver, W. W.: „*Sound Support for Collaboration*“, in: Baecker, R.M. (ed.): „*Readings in Groupware and CSCW*“, Morgan Kaufman Publishers, San Mateo, 1993, Seite 355-362
- [GOL92] Goldberg, Y. et al.: „*Active Mail -- A Framework for Implementing Groupware*“, in: Proceedings of ACM CSCW '92 Conference on Computer-Supported Cooperative Work, (1992), Seite 75-83
- [GRA94] O'Grady, T.; Greenberg, S.: „*A Groupware Environment for Complete Meetings*“, in: Conference Companion CHI '94, ACM, (1994), Seite 307-308
- [HAA92] Haake, A; Haake, J.: „*Take CoVer: exploiting version support in cooperative systems.*“, in: Arbeitspapiere der GMD (Gesellschaft für Mathematik und Daten-verarbeitung), Report-Nr. 686 (1992) Seite 1-16
- [HEN86] Henderson, D. A. jr.; Card, S. K.: „*Rooms: The Use of Multiple Virtual Workspaces to Reduce Space Contention in a Window-Based Graphical User Interface*“, in: ACM Transactions on Graphics, Vol. 5, No. 3, July 1986, Seite 2111-243.
- [ISH92] Ishii, H.: „*Translucent multiuser interface for realtime collaboration.*“, in: IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences, Band E75-A (1992) Heft 2, Seite 122-131
- [KIS93] Kishimoto, T; Suzuki, G.: „*Virtual offices.*“, in: IEEE Communications Magazine, Band 31 (1993) Heft 10, Seite 36-38
- [KNI90] Knister, M. J.; Prakash, A.: „*DistEdit: A Distributed Toolkit for Supporting Multiple Group Editors*“, in: Proceedings of ACM CSCW '90 Conference on Computer-Supported Cooperative Work, (1992), Seite 343-355

- [KOB93] Kobayashi, M; Ishii, H.: „*ClearBoard: a novel shared drawing medium that supports gaze awareness in remote collaboration.*“, in: IEICE Transactions on Communications, Band E76-B (1993) Heft 6, Seite 609-617
- [KRE93] Kreifelts, T. et al.: „*Experiences with the DOMINO Office Procedure System*“, in: Proceedings of the Second European Conference on Computer-Supported Cooperative Work (ECSCW '91), Kluwer, Dordrecht, 1993, Seite 117-130
- [KRU93] Kruglinski, D.J.: „*Inside Visual C++*“, Microsoft Press Deutschland, Unterschleißheim, 1993.
- [LAI93] Lai, K. Y. et al.: „*Object Lens: A „Spreadsheet“ for Cooperative Work*“, in: Baecker, R.M. (ed.): „*Readings in Groupware and CSCW*“, Morgan Kaufman Publishers, San Mateo, 1993, Seite 474-484
- [LI92] Li, J.; Mantei, M.: „*Working Together, Virtually*“, in: Graphics Interface '92, (1992), Seite 115-122
- [LUC94] Lucas, P.; Schneider, L.: „*Workscape: A Scriptable Document Management Environment*“, in: Conference Companion CHI '94, ACM, (1994), Seite 9-10
- [MAC90] Mackinlay, J. D. et al.: „*A Semantic Analysis of the Design Space of Input Devices*“, in: Human-Computer Interaction, Vol. 5, No. 2-3, (1990), Seite 145-190
- [MAC91] Mackinlay, J. D. et al.: „*The Perspective Wall: Detail and Context Smoothly Integrated*“, in: Proceedings of the CHI '91, ACM, New York, 1991, Seite 173-179
- [MAL93] Malone, T. W. et al.: „*The Information Lens: An Intelligent System for Information Sharing and Coordination*“, in: Baecker, R.M. (ed.): „*Readings in Groupware and CSCW*“, Morgan Kaufman Publishers, San Mateo, 1993, Seite 461-473
- [MAR90] Markus, M. L.; Connolly, T.: „*Why CSCW applications fail: problems in the adoption of interdependent work tools.*“, in: CSCW '90, Proceedings of the Conference on Computer-Supported Cooperative Work, Los Angeles, USA, October 7-10, 1990, (1990) New York: Association for Computing Machinery, Seite 371-380, ISBN 0-89791-402-3.

- [MAR91] Marcus, A; van Dam, A.: „*User-interface developments for the nineties.*“, in: Computer, Long Beach, Band 24 (1991) Heft 9, Seite 49-57
- [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), Seite 265-272
- [OLS92] Olson, G.M.; Olson, J. S. „*Interface-defining a metaphor for group work.*“, in: IEEE Software, Band 9 (1992) Heft 3, Seite 93-95
- [ORL92] Orlikowski, W. J.: „*Learning from Notes: Organizational Issues in Groupware Implementation*“, in: Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work, (1992), Seite 362-369
- [PET92] Petrovic, O.: „*Groupware - Systemkategorien, Anwendungsbeispiele, Problemfelder und Entwicklungsstand.*“, in: Information Management, München, Band 7 (1992) Heft 1, Seite 16-22.
- [PRA92] Prakash, A.; Knister, M. J.: „*Undoing Actions in Collaborative Work*“, in: Proceedings of ACM CSCW'92 Conference on Computer-Supported Cooperative Work, (1992), Seite 273-280
- [PUC93] Pucko, M et al.: „*Developing multi-user interfaces for CSCW environment.*“, in: Microprocessing and Microprogramming, Band 37 (1993) Heft 1-5, Seite 123-126
- [ROB91] Robertson, G. G. et al.: „*Cone Trees: Animated 3D Visualizations of Hierarchical Information*“, in: Proceedings of the CHI '91, ACM, New York, 1991, Seite 189-194
- [ROD92] Rodden, T; Blair, G. S.: „*Distributed systems support for computer supported cooperative work*“, in: Computer Communications, Band 15 (1992) Heft 8, Seite 527-538
- [ROH95] Rohde, M.; Pfeiffer, A und Wulf, V.: „*Konfliktmanagement bei Vorgangsbearbeitungssystemen*“, Institut für Informatik III, Universität Bonn, 1995, eingereicht für: Zeitschrift für Wirtschaftsinformatik
- [SAN92] Santos, A.: „*A Cooperative Architecture for Hypermedia Editing - CoMEdiA*“, in: Computer Graphics Forum, Volume 11 (1992), Number 5, Seite 309-322

- [SCH93] Schallenmueller, S.: „*Bueroablaeufer werden wesentlich beschleunigt.*“, in: PC Magazin, (1993) Heft 33, Seite 37-38, 40
- [SHN92] Shneiderman, B.: „*Designing the User Interface*“, Addison-Wesley, Reading, MA, 1992
- [SOH94] Sohlenkamp, M.; Chwelos, G.: „*Integrating Communication, Cooperation and Awareness: The DIVA Virtual Office Environment*“, in: Proceedings of CSCW '94, ACM, (1994), Seite 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)
- [SOM92] Sommerville, I.: „*Software Engineering*“, Fourth Edition, Addison-Wesley, Reading, MA, 1992
- [STE88] Stefik, M. et al.: „*Beyond the Chalkboard: Computer Support for Collaboration and Problem Solving in Meetings*“, in: Proceedings of CSCW '88, ACM, (1988)
- [TAN94] Tang, J. C.; Rua, M.: „*Montage: Providing Teleproximity for Distributed Groups*“, in: Proceedings of CHI '94, Seite 37-43
- [TOL94] Tollmar, K. et al.: „*The Collaborative Desktop. An Environment for Computer Supported Cooperative Work*“, in: Conference Companion CHI '94, ACM, (1994), Seite 23-24
- [WIN87] Winograd, T.: „*A Language/Action Perspective on the Design of Cooperative Work*“, in: Human-Computer Interaction, Vol. 3 (1987/88) No. 1, Seite 3-30
- [WIN88] Winograd, T.: „*Where the Action Is*“, in: BYTE Vol. 13 (1988) No. 12, Seite 256A-258
- [WOI90] Woitass, M.: „*Koordination in strukturierten Konversationen. Ein Koordinationsmodell fuer kooperierende Agenten und seine Anwendungen im Bereich Computer-Supported Cooperative Work (CSCW).*“, Dissertation: GMD-Bericht, Gesellschaft fuer Mathematik und Datenverarbeitung, Band 190 (1990) Muenchen, Wien: R. Oldenbourg Verlag, Seite 1-222, ISBN 3-486-21843-3.

---

# Abbildungsverzeichnis

---

Abbildung 5.1: Objektklassen im gemeinsamen Arbeitsbereich.....	23
Abbildung 5.2: Überblick über das Modell .....	25
Abbildung 5.3: Erweiterung der Meldungserzeugung um Filter .....	25
Abbildung 5.4: Ereignis- bzw. Aktivitätstypen und ihr Bezug auf Benutzer und Objektklassen .....	28
Abbildung 5.5: Ereignisklassen .....	29
Abbildung 5.6: Legende für die folgenden Abbildungen .....	39
Abbildung 5.7: Lebenszyklus einer Meldung über das Ereignis „Objekt geändert“ mit initialer Wichtigkeit „Sehr wichtig“ .....	40
Abbildung 5.8: Lebenszyklus einer Meldung über das Ereignis „Objekt geändert“ mit initialer Wichtigkeit „interessant“ .....	40
Abbildung 5.9: Lebenszyklus einer Meldung über das Ereignis „Objekt gesperrt“ mit initialer Wichtigkeit „sehr wichtig“ .....	41
Abbildung 5.10: Lebenszyklus einer Meldung über das Ereignis „Objekt gesperrt“ mit initialer Wichtigkeit „interessant“ .....	42
Abbildung 5.11: Erzeugen und Verteilen einer Meldung .....	45
Abbildung 5.12: Nicht zyklischer Strukturgraph .....	49
Abbildung 6.1: Graphdarstellung von „Objekt bearbeiten“ .....	60
Abbildung 6.2: Attributdarstellung von „Objekt bearbeiten“ .....	61
Abbildung 7.1: Systemarchitektur des Prototypen .....	76
Abbildung 7.2: Neue LinkWorks-Objektklassen .....	77
Abbildung 7.3: C++-Klassen zur Einkapselung der LinkWorks-Klassen .....	79
Abbildung 7.4: Lokale Zustandsübergänge eines Objektes .....	85
Abbildung 7.5: Liste der Interesse-Einträge für ein Objekt .....	88
Abbildung 7.6: Bearbeiten eines Interesse-Eintrags .....	89

*Die Farbtafeln befinden sich in Anhang D.*

Farbtafel 4.1:	Der LinkWorks-Schreibtisch.....	137
Farbtafel 4.2:	Der existierende Benachrichtigungsdienst .....	138
Farbtafel 6.1:	Das LinkWorks-Grafikalphabet .....	139
Farbtafel 6.2:	Erweiterung des LinkWorks-Grafikalphabetes .....	139
Farbtafel 6.3:	Darstellung des Ereigniszeitpunktes.....	140
Farbtafel 6.4:	Darstellung von Ereignistypen und propagierten Meldungen .....	140
Farbtafel 6.5:	Kontextgebundene Darstellung von Ereignismeldungen .....	141
Farbtafel 6.6:	Interaktivität der kontextgebundenen Darstellungstechnik .....	141
Farbtafel 6.7:	Kontextübergreifende nicht-modale Darstellung von Ereignismeldungen .....	142
Farbtafel 6.8:	Interaktivität der kontextübergreifenden nicht-modalen Darstellungstechnik .....	142
Farbtafel 6.9:	Kontextübergreifende modale Darstellung von Ereignismeldungen..	142
Farbtafel 6.10:	Benutzerinitiiert aufrufbare Meldungsliste mit maximaler Interaktivität .....	143
Farbtafel 6.11:	Detaillierte textliche Darstellung des gemeldeten Ereignisses.....	143
Farbtafel 7.1:	Codierung der lokalen Objektzustände im Objekticon .....	144

---

# Tabellenverzeichnis

---

Tabelle 3.1:	Systemkategorien, ihre Merkmale und der Informationsbedarf eines Benutzers .....	13
Tabelle 3.2:	Aktivitäten und Informationsbedarf eines Benutzers in einem gemeinsamen Arbeitsbereich.....	14
Tabelle 3.3:	Benutzeraktivitäten, die dem Gebiet der Awareness zugeordnet werden können .....	15
Tabelle 3.4:	Klassifizierung von Awarenessinformation .....	15
Tabelle 5.1:	Fragen des Benutzers und zu erfassende Daten.....	26
Tabelle 5.2:	Semantik der Ereignistypen.....	27
Tabelle 5.3:	Wichtigkeit .....	31
Tabelle 6.1:	Priorität der Ereignisattribute .....	57
Tabelle 6.2:	Im Objekticon dargestellte Information und die Art ihrer Darstellung .....	62
Tabelle 6.3:	Farbcodierung des Ereigniszeitpunktes .....	63
Tabelle 6.4:	Semantik der Ereignistypsymbbole .....	66
Tabelle 6.5:	Präsentation entsprechend der Wichtigkeit .....	72
Tabelle 7.1:	Zulässige Schachtelung von Objekten in Containerobjekten.....	77
Tabelle 7.2:	Zuordnung der C++-Klassen zu den LinkWorks-Klassen .....	79
Tabelle 7.3:	Struktur der Datenbanktabelle „Events“ .....	81
Tabelle 7.4:	Struktur der Datenbanktabelle „InterestPatterns“ .....	82
Tabelle 7.5:	Struktur der Datenbanktabelle „Notifications“ .....	82
Tabelle 7.6:	Farbliche Unterscheidung der Teile einer Icon-Bitmap .....	95

---

## Anlage A: Glossar

---

- Aktivität:* Prozeß mit einer Zeitdauer größer als Null, der den Zustand von Objekten ändern, also → Ereignisse erzeugen kann. A. wird von Anfangs- und Endereignis begrenzt und ist somit durch Ereignisse beschreibbar.
- Annullieren:* (von → Meldungen): Operation zur Konsistenzsicherung, bei der eine von zwei Meldungen abhängig von den Typen der unterliegenden → Ereignisse gelöscht wird.
- Asynchronous Awareness:* Erfassen des aktuellen Standes der Arbeiten und inzwischen eingetretener Änderungen nach einer Arbeitsunterbrechung.
- Aufdringlichkeit:* Eigenschaft einer → Präsentationstechnik, die ausdrückt, wie stark diese den Benutzer zwingt, seine Arbeit zu unterbrechen.
- Ausdünnen:* (von → Meldungen): Operation zur Vermeidung von Informationsüberflutung, die die ältere von zwei Meldungen über → Ereignisse löscht, die sich nur im Zeitpunkt ihres Eintretens unterscheiden.
- Awareness:* Nutzung implizit vorhandener Informationskanäle mit dem Ziel der bewußten Erfassung vergangener und gegenwärtiger → Aktivitäten anderer Kooperationspartner im aktuellen Arbeitsumfeld (von englisch *to be aware of sth.* - sich einer Sache bewußt sein). Es gibt zwei Modi von A.: → Synchronous A. und → Asynchronous A.
- Container:* Siehe → Containerobjekt.
- Containerhierarchie:* Baum, dessen Knoten → Containerobjekte sind.
- Containerobjekt:* Objekt, das andere Objekte enthalten kann. Auch als Container bezeichnet. Dient zur Bildung der → Containerhierarchie und zur Strukturierung aller Objekte in einer → Objekthierarchie. Ein C. ist *geöffnet*, wenn ein Bildschirmfenster zur Anzeige der enthaltenen Objekte existiert, ansonsten *geschlossen*.
- Detailliertheit:* Eigenschaft einer → Präsentationstechnik, die ausdrückt, wieviele Einzelheiten in der Darstellung erkenn- und selektierbar sind.

- Dringlichkeit:* (einer  $\rightarrow$  Präsentationstechnik): Charakterisiert eine Präsentationstechnik bezüglich ihres Potentials zur Darstellung von  $\rightarrow$  Meldungen entsprechend deren Wichtigkeit.
- Entstehungskontext:* Siehe  $\rightarrow$  Kontext.
- Ereignis:* Zustandsänderung eines Objektes zu einem bestimmten Zeitpunkt mit einer Zeitdauer von Null. E. ist damit primärer Träger von Awarenessinformation und wird als Tupel von  $\rightarrow$  Ereigniszeitpunkt,  $\rightarrow$  Ereignistyp, optionalen  $\rightarrow$  Ereignisparametern,  $\rightarrow$  Ereignisinitiator,  $\rightarrow$  Ereignisobjekt und  $\rightarrow$  Ereignis-Elternobjekt modelliert. Softwaretechnisch betrachtet kann E. als Objekt modelliert werden: Ereignistyp bestimmt die Objektklasse, die anderen Felder des Ereignis-Tupels werden die Attribute der Klasse realisiert.
- Ereignis-Elternobjekt:* Optionales Feld eines  $\rightarrow$  Ereignisses, das das Elternobjekt des  $\rightarrow$  Ereignisobjektes zum  $\rightarrow$  Ereigniszeitpunkt bezeichnet.
- Ereignisinitiator:* Feld eines  $\rightarrow$  Ereignisses, das den Benutzer bezeichnet, der das Ereignis auslöst.
- Ereignismeldung:* Siehe  $\rightarrow$  Meldung.
- Ereignisobjekt:* Feld eines  $\rightarrow$  Ereignisses, das das Objekt bezeichnet, dessen Zustand durch das Ereignis geändert wird.
- Ereignisparameter:* Optionales Feld eines  $\rightarrow$  Ereignisses, das abhängig vom  $\rightarrow$  Ereignistyp zusätzliche Information über die durch dieses Ereignis repräsentierte Zustandsänderung enthält.
- Ereignistyp:* Feld eines  $\rightarrow$  Ereignisses, das die Art der durch dieses Ereignis repräsentierten Zustandsänderung ausdrückt.
- Ereigniszeitpunkt:* Feld eines  $\rightarrow$  Ereignisses, das den Zeitpunkt des Eintretens des Ereignisses repräsentiert.
- Erkennbarkeit:* Eigenschaft einer  $\rightarrow$  Präsentationstechnik, die bestimmt, wie gut der Mensch mit seinen Augen die Darstellung erkennen kann.
- Ghost-Objekt:* Platzhalter für ein gelöscht oder verschobenes Objekt. Dient zum  $\rightarrow$  objektgebundenen Anzeigen der Lösch- bzw. Verschiebemeldung.
- Groupwaresystem:* Softwaresystem, das die kooperative Arbeit unterstützt.

- Interaktionsmöglichkeiten:* (einer → Meldung): Operationen, die das Reagieren des Empfängers auf die Meldung gestatten. Sie umfassen das Löschen der Meldung, das Ändern ihrer → Relevanz, den Aufbau einer Kommunikationsverbindung mit dem Initiator des gemeldeten → Ereignisses und die Interaktion mit dem → Ereignisobjekt.
- Interaktivität:* Fähigkeit einer → Präsentationstechnik, die die Möglichkeiten des Benutzers angibt, mit einer angezeigten → Meldung zu interagieren. Die von einer Präsentationstechnik bereitgestellte I. macht einige oder alle → Interaktionsmöglichkeiten der Meldung für den Benutzer verfügbar.
- Interesse:* Muster, das angibt, wie interessant ein → Ereignis für einen Benutzer ist. Bestimmt die → Relevanz der erzeugten → Meldung über das Ereignis für diesen Benutzer. Zwei Möglichkeiten zum Angeben von I.: *Explizites I.* (Interessemuster wird direkt vom Benutzer angegeben) und *Implizites I.* (Interessemuster wird automatisch beim Start einer → Aktivität erzeugt und bei deren Ende gelöscht).
- Kontext:* 1) Beschreibung einer Arbeitssituation. Wurde in vorliegender Arbeit vereinfachend mit einem → Containerobjekt gleichgesetzt. K. erbt die Eigenschaften *geöffnet* und *geschlossen* und die hierarchische Anordnung vom Containerobjekt. Bezogen auf → Meldungen gibt es zwei Arten des K.: *Entstehungskontext* (Container, in dem sich das vom gemeldeten → Ereignis betroffene Objekt beim Eintreten des Ereignisses befand) und *Meldekontext* (Container, in dem die Meldung relevant ist).  
2) Der Darstellungskontext ist eine Eigenschaft einer → Präsentationstechnik und kennzeichnet den Ort der Darstellung an der Benutzeroberfläche.
- Kontextgebundene Präsentation:* → Präsentationstechnik, die eine Zusammenfassung von → Ereignismeldungen in deren → Entstehungskontext darstellt.
- Kontextübergreifende Präsentation:* → Präsentationstechnik, die → Ereignismeldungen räumlich getrennt vom → Entstehungskontext und dessen übergeordneten → Kontexten in einer Dialogbox darstellt.
- Maximalpräsentationstechnik:* → Präsentationstechnik mit der höchsten → Dringlichkeit bezogen auf eine → Meldung. Die M. wird der Meldung in Abhängigkeit von ihrer → Relevanz zugeordnet. Die Meldung wird mit allen Techniken dargestellt, deren Dringlichkeit kleiner oder gleich der Dringlichkeit der M. ist.

<i>Meldekontext:</i>	Siehe → Kontext.
<i>Meldung:</i>	Information über den Eintritt eines → Ereignisses mit einer gewissen → Relevanz und bestimmten → Interaktionsmöglichkeiten, die an einen Benutzer gerichtet ist. Auch als Ereignismeldung bezeichnet.
<i>Objektereignis:</i>	→ Ereignis, das sich auf ein Objekt im gemeinsamen Arbeitsbereich bezieht.
<i>Objektgebundene Präsentation:</i>	→ Präsentationstechnik, die eine → Ereignismeldung im Icon des vom → Ereignis betroffenen Objektes darstellt. Die Attribute des gemeldeten Ereignisses werden auf grafische Attribute des Icons abgebildet.
<i>Objekthierarchie:</i>	Baum, dessen Nicht-Blattknoten → Containerobjekte und dessen Blattknoten Objekte einer beliebigen Klasse sind.
<i>Präsentation:</i>	Darstellung einer → Meldung abhängig von ihrer → Relevanz mit → Präsentationstechniken entsprechender → Dringlichkeit.
<i>Präsentationstechnik:</i>	Methode zur grafischen Darstellung von → Meldungen, die durch die <i>Eigenschaften</i> → Aufdringlichkeit, → Kontext, → Erkennbarkeit und → Detailliertheit sowie die <i>Fähigkeiten</i> → Signalwirkung und → Interaktivität gekennzeichnet ist.
<i>Propagieren:</i>	<p>1) von → Meldungen: Weiterleiten von Meldungen zum Zeitpunkt ihrer Entstehung aus dem → Entstehungskontext in übergeordnete → Kontexte.</p> <p>2) von → Interesse: Weiterleiten von Interesse in untergeordnete → Container in der → Containerhierarchie mit oder ohne Überschreiben der dort vorhandenen Interessemuster.</p>
<i>Relevanz:</i>	Komponente einer → Meldung, die ausdrückt, wie wichtig die Meldung dem Empfänger in welchem → Kontext ist und ob er von der Meldung bereits Kenntnis genommen oder sie auf einen späteren Zeitpunkt zurückgestellt hat. Die R. steuert die → Präsentation der Meldung.
<i>Signalwirkung:</i>	Fähigkeit einer → Präsentationstechnik, die das Potential der Technik zum Erregen der Aufmerksamkeit des Benutzers ausdrückt.
<i>Synchronous Awareness:</i>	Ständige detaillierte Information über zur eigenen Arbeit parallele Tätigkeiten von Kooperationspartnern.

---

*Systemereignis:* → Ereignis, das eine Zustandsänderung des → Groupwaresystems ausdrückt.

---

# Anlage B: Funktionsbeschreibung des Prototypen

---

## B.1 Allgemeines

Der Prototyp „Gemeinsamer Arbeitsbereich“ ist eine LinkWorks-Applikation. Die Kenntnis der Bedienphilosophie und des Objektklassenkonzeptes von LinkWorks in der Version 3.0 wird für das Verständnis dieser Funktionsbeschreibung vorausgesetzt. Wenn von Objekten bzw. Objektklassen die Rede ist, so sind stets LinkWorks-Objekte bzw. -Objektklassen gemeint.

## B.2 Objektklassen

Die Objektklasse „Gemeinsamer Arbeitsbereich“ ist eine Containerobjektklasse, die die gemeinsame Benutzung der darin befindlichen Objekte ermöglicht. Zur Zeit werden vier Objektklassen unterstützt, die sich im Gemeinsamen Arbeitsbereich befinden können:

Objektklasse	Erklärung
Dokument	Repräsentiert ein mit dem Werkzeug WRITE bearbeitbares Textdokument
Ordner	Enthält Objekte der Klassen Dokument und Ordner
Benutzer	Repräsentiert einen Benutzer
Benutzerin	Repräsentiert eine Benutzerin

Nach dem Erzeugen eines neuen Objektes der Klasse „Gemeinsamer Arbeitsbereich“ muß der erzeugende Benutzer folgende Schritte<sup>1</sup> ausführen, um den Arbeitsbereich gemeinsam mit anderen Kollegen zu nutzen:

- Öffnen des Arbeitsbereiches
- Einfügen aller Benutzer, die den Arbeitsbereich gemeinsam nutzen sollen, durch wiederholten Aufruf der Funktion <Benutzer hinzufügen> aus dem Menü <Arbeitsbereich><sup>2</sup>
- Schließen des Arbeitsbereiches

---

<sup>1</sup> Diese umständliche Schrittfolge im Prototypen muß in einem praxisreifen System so umgestellt werden, daß der erzeugende Benutzer automatisch der Benutzerliste des Arbeitsbereiches hinzugefügt und das Versenden eines Verweises in die Funktion <Benutzer hinzufügen> integriert wird.

<sup>2</sup> Siehe Abschnitt B-B.5.1 (b)).

- Selektieren des Arbeitsbereiches
- Versenden des Arbeitsbereiches per LinkWorks-Mail an alle eingefügten Benutzer als Verweis
- Falls einer der neuen Benutzer den Arbeitsbereich nicht öffnen kann, muß der Systemadministrator die Client-Komponenten auf dessen Arbeitsplatzrechner installieren

### **B.3 Iconbasierte Benutzeroberfläche**

Für eine grundsätzliche Beschreibung der Icons und ihrer Semantik wird auf die Benutzerdokumentation des Systems LinkWorks verwiesen<sup>1</sup>.

### **B.4 Darstellungstechniken für Objektmeldungen**

Es gibt zwei verschiedene Darstellungsformen für Objektmeldungen - die Darstellung im Objekticon und die Darstellung in einer modalen Meldebox. Während erstere die Meldungen auf eine unaufdringliche, nebenbei wahrnehmbare Art und Weise darstellt, unterbricht das System bei letzterer die Arbeit des Benutzers, um eine sehr wichtige Meldung darzustellen und den Benutzer unmittelbar zu einer Reaktion aufzufordern. Der Aufruf der Meldebox ist auch auf Benutzerinitiative möglich, um auf alle Meldungen - auch auf ausschließlich im Icon oder gar nicht präsentierte - reagieren zu können.

#### **B.4.1 Codierung im Icon**

Aus der Codierung im Objekticon sind verschiedene Informationen zu entnehmen:

- das Alter des gemeldeten Ereignisses
- der Typ des gemeldeten Ereignisses
- ggf. der Benutzer, der das gemeldete Ereignis ausgelöst hat
- der Selektionsstatus eines Objektes

In den folgenden Abschnitten wird auf die codierte Information im einzelnen eingegangen.

#### **a) Alter des gemeldeten Ereignisses**

---

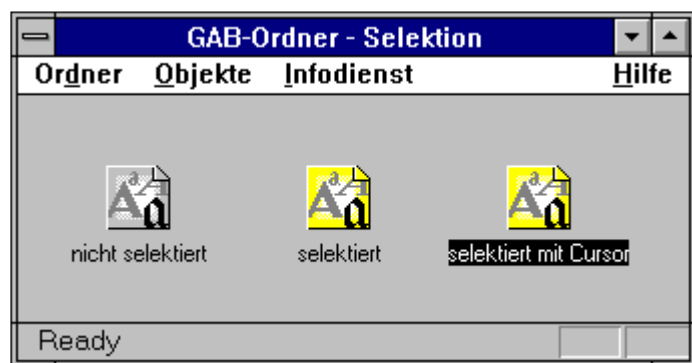
<sup>1</sup> Der Prototyp implementiert nur eine Untermenge der LinkWorks-Objektklassen. Zeilen- und Strukturdarstellung stehen ebenfalls nicht zur Verfügung.

Das Alter des gemeldeten Ereignisses wird durch die Farbe des entsprechenden Icon-Teiles dargestellt, wie aus nebenstehender Abbildung ersichtlich ist.



### b) Selektion

Ein selektiertes Objekt wird durch einen hellgelben Iconhintergrund dargestellt. Bei dem Objekt, auf dem sich der Cursor befindet, ist zusätzlich der Iconcontext invertiert. Je Container können null oder mehr Objekte selektiert sein, jedoch besitzt nur ein Objekt den Cursor. Auf



selektierte Objekte können Funktionen aus den Menüs <Objekte> und <Infodienst> angewendet werden. Die Selektion erfolgt mit der Maus durch Ziehen bzw. Anklicken oder mit der Tastatur<sup>1</sup>. Bei Nutzung der Tastatur kann der Cursor mit Hilfe der Cursortasten verschoben und das entsprechende Objekt mit der Leertaste selektiert bzw. deselektiert werden. Bei Nutzung der Maus wird das angeklickte Objekt selektiert. Wird beim Anklicken die Umschalttaste betätigt, so wird das angeklickte Objekt zusätzlich zu den bereits ausgewählten Objekten selektiert, ansonsten wird es zum einzigen selektierten Objekt. Anklicken des Fensterhintergrundes deselektiert alle Objekte.

<sup>1</sup> Im Prototypen wurde bisher nur die Selektion durch Anklicken mit der Maus implementiert.

### c) Geöffnete Objekte

Geöffnete Objekte werden durch einen dunkelgelben Iconhintergrund repräsentiert. Der Zustand „geöffnet“ bezieht sich immer auf den aktuellen Benutzer. Durch andere Benutzer geöffnete Objekte werden nicht als „geöffnet“

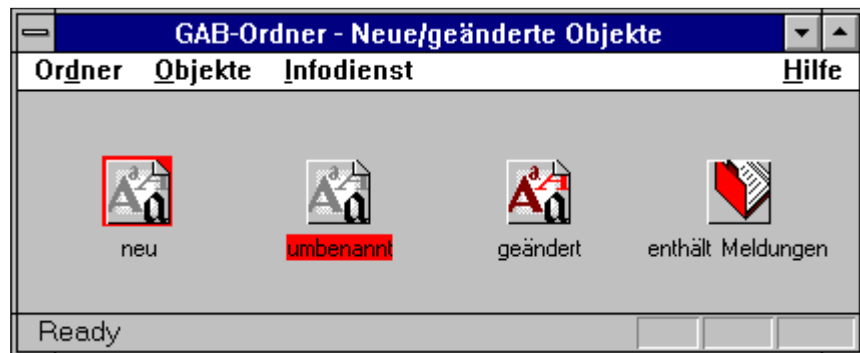


dargestellt. Ein geöffnetes Objekt kann nicht selektiert werden. Wurde ein Objekt zum Bearbeiten durch den aktuellen Benutzer geöffnet, hat es für diesen den Status „gesperrt“ und „geöffnet“. Der Status „gesperrt“ wird durch ein kleines Benutzer-Icon und zusätzlich durch den Namen des Bearbeiters über dem Objekticon codiert. Für alle anderen Benutzer hat das Objekt lediglich den Status „gesperrt“. Sie können das Objekt in diesem Fall immer noch selektieren und zum Lesen öffnen oder umbenennen.

Das als geöffnet dargestellte Objekt mit dem überlagerten Computer-Icon wird zur Zeit nach einer Benutzeranforderung vom Groupware-System bearbeitet (z.B. verschoben, kopiert, ...), es ist also quasi „vom Rechner gesperrt“. Diese Darstellung unterstützt die asynchrone Bearbeitung von Objekten. Während der Rechner ein Objekt bearbeitet (was bei langsamen Netzen u.U. lange Zeit in Anspruch nimmt), kann der Benutzer mit anderen Objekten weiterhin arbeiten.

#### d) Neue oder modifizierte Objekte

Ereignismeldungen über Objekte, die momentan im aktuellen Container existieren, werden in Teilen des Objekticons dargestellt. Die Farbe des entsprechenden



Icon-Teils codiert das Alter des Ereignisses. Neue Objekte (die durch Erzeugen oder Hineinlegen entstehen) werden durch einen Rahmen um das Icon dargestellt. Das Umbenennen eines Objektes wird durch farbige Hinterlegung des Objektamens codiert. Änderungen werden in die Farbe des Icon-Sinnbildes abgebildet. Von der Semantik her muß dabei zwischen Dokumentobjekten und Containerobjekten unterschieden werden. Während bei Dokumentobjekten der Zeitpunkt der letzten Änderung des Dokuments angezeigt wird, stellt die Farbe des Sinnbildes eines Containerobjekts den Zeitpunkt der letzten unquittierten<sup>1</sup> Ereignismeldung im Container (oder in diesem untergeordneten Containern) dar.

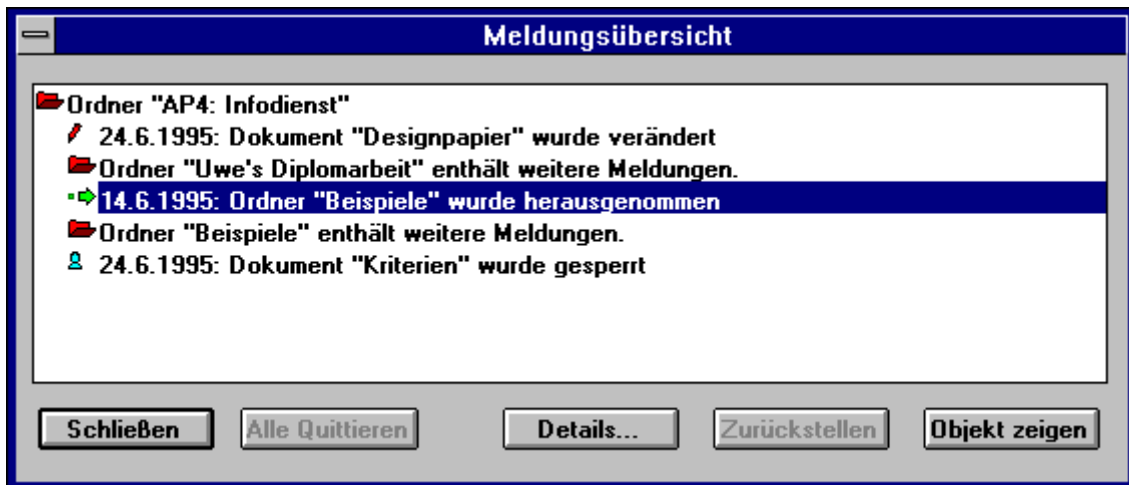
#### e) Nicht mehr existierende Objekte

Nicht mehr existierende Objekte werden als „Geister“ (Ghost-Objekte) dargestellt. In einem Overlay-Icon wird das Ereignis dargestellt, das den Lebenszyklus des Objektes im aktuellen Container beendete. Das Kreuz symbolisiert ein Löscheignis, der Pfeil das Ereignis „Herausnehmen“. Für Ghost-Objekte sind die Funktionen im Menü <Objekte> nicht mehr verfügbar.



<sup>1</sup> Siehe Abschnitt B-B.5.3 (e))

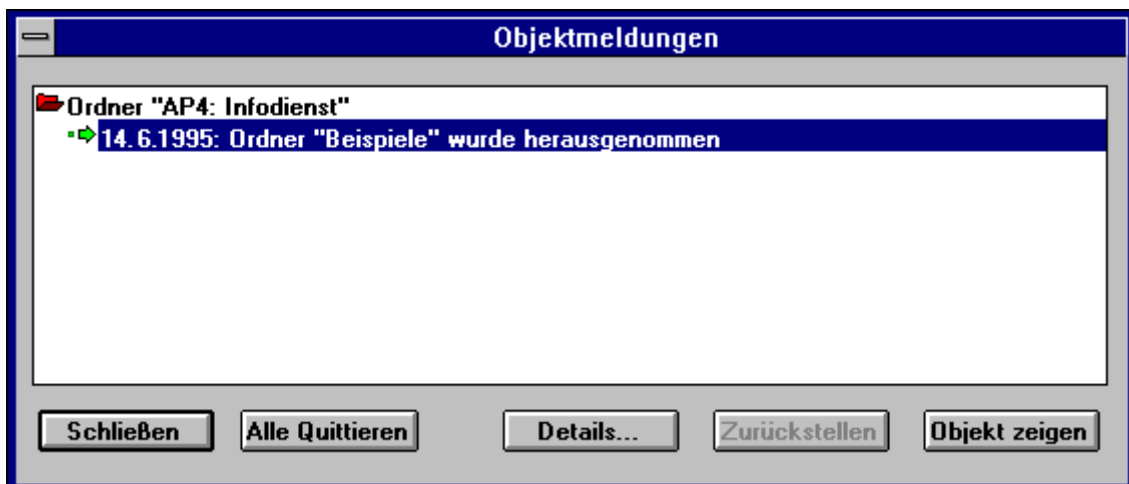
## B.4.2 Darstellung in der Meldebox



Die Darstellung der Ereignismeldungen in der Meldebox erfolgt in textueller Form. Hier werden für alle Ereignismeldungen das Datum des Ereignisses, das Ereignisobjekt, der Ereignistyp und das Elternobjekt des Ereignisobjektes (also der Container, in dem sich dieses befindet) dargestellt.

In der ersten Spalte wird zur besseren Übersichtlichkeit ein Miniicon angezeigt, dessen Form den Ereignistyp und dessen Farbe den Ereigniszeitpunkt darstellt.

Es gibt drei verschiedene Arten von Meldeboxen: die *Meldungsübersicht*, die *Objektmeldungen* und den *Meldedienst*. Die *Meldungsübersicht* (siehe obige Abbildung) stellt alle Meldungen dar, die sich auf Objekte im aktuellen Container beziehen. Sie wird auf Nachfrage des Benutzers (Menü <Infodienst> <Meldungsübersicht>) geöffnet. Die zweite auf Benutzerinitiative geöffnete Meldebox enthält *Objektmeldungen*, also alle Meldungen der aktuell selektierten Objekte<sup>1</sup>.

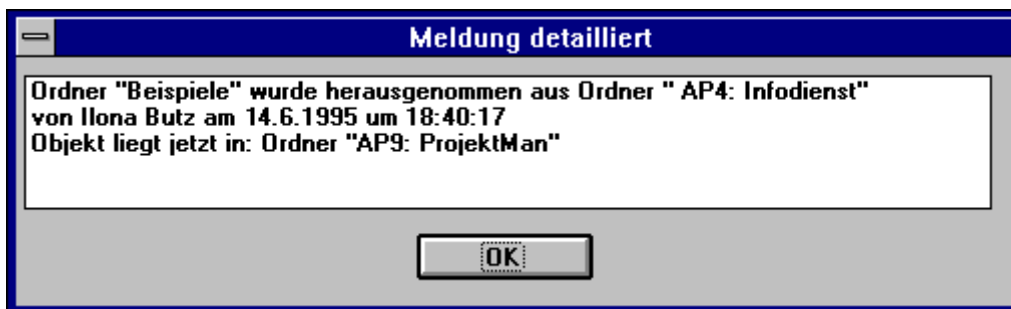


<sup>1</sup> Im Prototypen wurde die Funktion *Objektmeldungen* bisher nur für genau ein selektiertes Objekt implementiert. Sind mehrere Objekte selektiert, ist die Funktion nicht verfügbar.

Auf Initiative des Systems wird der *Meldedienst* aktiv. Dieser stellt Meldungen über sehr wichtige Ereignisse unmittelbar dar und zur Interaktion bereit.



Jede Meldebox ermöglicht eine Interaktion mit den Meldungen. Durch Selektieren einer Meldung mit der Maus und Anklicken des Aktionsknopfes <Details> können weitere in der Meldung enthaltene Informationen angezeigt werden, wie die Zeit des Ereignisses, der auslösende Benutzer sowie vom Ereignistyp abhängige Parameter des Ereignisses<sup>1</sup>.



Anklicken des Aktionsknopfes <Objekt anzeigen> öffnet den Container, der das Ereignisobjekt enthält, und selektiert letzteres<sup>2</sup>. Betätigen des Aktionsknopfes <Alle Quittieren> quittiert (löscht) alle Meldungen in der Meldebox. Damit können für den Benutzer nicht mehr relevante Meldungen entfernt werden<sup>3</sup>.

Die Funktionen <Alle Gesehen> und <Zurückstellen> sind nur beim Meldedienst verfügbar. <Zurückstellen> stellt einzelne Meldungen auf einen späteren Zeitpunkt

<sup>1</sup> Ein Ereignisparameter kann zum Beispiel das neue Elternobjekt eines herausgenommenen Objektes oder der alte Name eines umbenannten Objektes sein.

<sup>2</sup> Diese Funktion ist im Prototypen noch nicht implementiert.

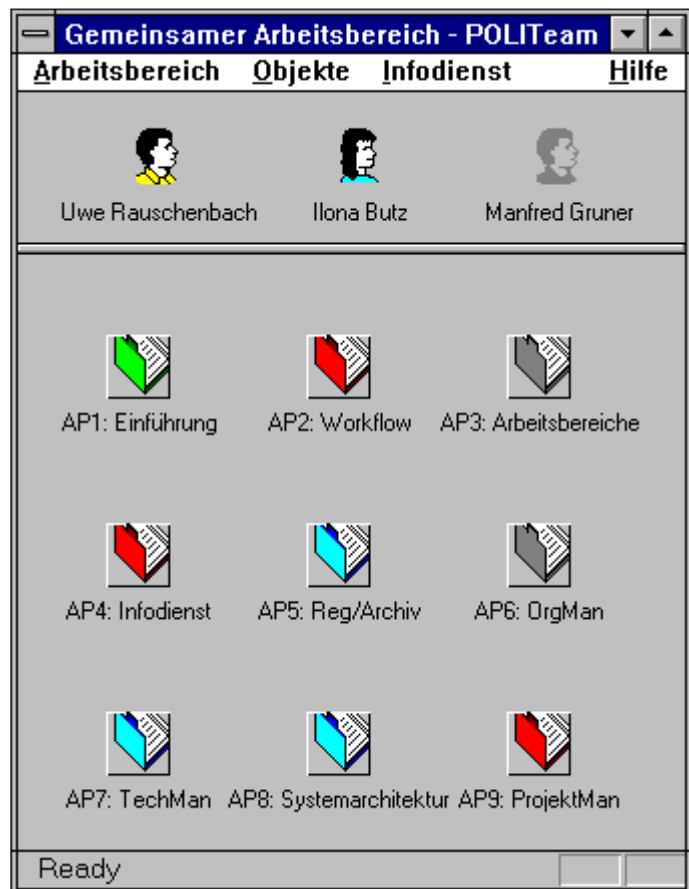
<sup>3</sup> Diese Funktion wurde im Prototypen bisher nur für Objektmeldungen implementiert, um das Konzept zu veranschaulichen. In einem anwendungsreifen System muß es möglich sein, die Quittierungsfunktion ebenfalls in der Meldungsübersicht zu verwenden und auch nur ausgewählte Meldungen zu quittieren.

zurück, realisiert also eine Erinnerungsfunktion. Nach Ablauf eines Zeitintervalls<sup>1</sup> zeigt der Meldedienst die Meldung erneut an. Zurückgestellte Meldungen werden in der Liste durch Voranstellen der Zeichenkette „>>>“ gekennzeichnet, bei der erneuten Anzeige wird ihnen der Text „ERINNERUNG“ vorangestellt. <Alle Gesehen> schließt die Dialogbox des Meldedienstes und markiert alle nicht zurückgestellten Meldungen in der Liste als *gelesen*. Sie werden daraufhin nur noch im Objekticon dargestellt und können auf Benutzeranfrage in der Dialogbox *Objektmeldungen* oder *Meldungsübersicht* angezeigt werden<sup>2</sup>.

## B.5 Der Gemeinsame Arbeitsbereich

Der Gemeinsame Arbeitsbereich ist ein Containerobjekt, das Ordner, Dokumente und Benutzer enthalten kann und Ereignismeldungen in den Objekticons anzeigt.

Der untere Teil des Fensters enthält Dokumente und Ordner und ermöglicht eine Interaktion mit diesen. Im oberen Bereich befinden sich die Icons aller Benutzer, die berechtigt sind, auf den Arbeitsbereich zuzugreifen. Das ermöglicht die Implementation weitergehender Konzepte wie Initiierung synchroner Kommunikation bzw. Versenden von Objekten per Mail an ausgewählte Adressaten per Drag&Drop<sup>3</sup>. Gerade „im Arbeitsbereich anwesende“ Benutzer sind farbig, „abwesende“ Benutzer dunkelgrau dargestellt.



Bei der Farbdarstellung wird zwischen dem lokalen Benutzer (gelb) und den entfernten Kooperationspartnern (cyan) unterschieden.

<sup>1</sup> Im Prototypen sind drei Minuten fest codiert. Das Zeitintervall sollte in einem fertigen System benutzerdefinierbar sein.

<sup>2</sup> Siehe Abschnitt B-B.5.3.

<sup>3</sup> Diese Konzepte sind im Prototypen noch nicht realisiert.

## B.5.1 Funktionen im Menü <Arbeitsbereich>

Das Menü <Arbeitsbereich> enthält Funktionen, die sich auf den Arbeitsbereich als ganzes beziehen.

### a) Objekte ausrichten

Der Befehl <Objekte ausrichten> richtet die Objekte so aus, daß die ganze Breite des Fensters ausgenutzt wird.

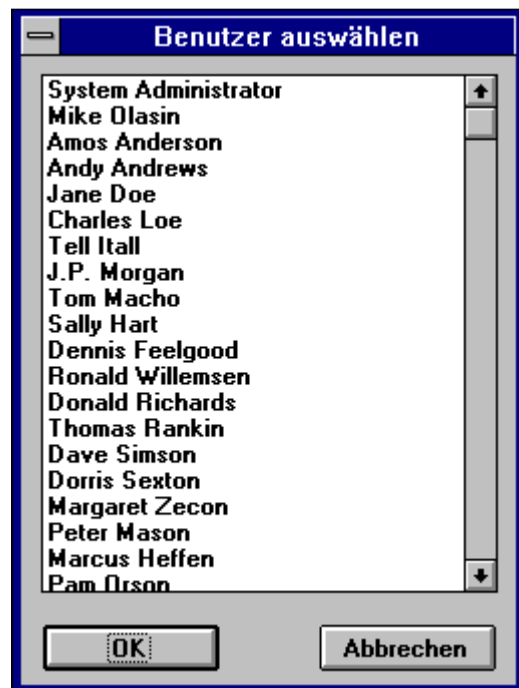
### b) Benutzer hinzufügen

Der Befehl <Benutzer hinzufügen> präsentiert eine Liste aller im LinkWorks-System bekannten Benutzer. Hier kann ein neuer Benutzer ausgewählt werden, der zum Arbeitsbereich Zutritt erhalten soll<sup>1</sup>.

### c) Arbeitsbereich verlassen

Die Funktion <Arbeitsbereich verlassen> schließt den Arbeitsbereich.

Falls noch Objekte zum Lesen bzw. Bearbeiten geöffnet oder vom System gesperrt sind, wird eine Fehlermeldung angezeigt. Anderenfalls werden alle anderen gerade im Arbeitsbereich aktiven Benutzer davon benachrichtigt, daß der aktuelle Benutzer den



Arbeitsbereich verläßt, was sich bei diesen in der Änderung der Farbe des Icons des entsprechenden Benutzers äußert. Vor dem Schließen des Arbeitsbereiches werden Aktualisierungen der Meldungsdatenbank vorgenommen. Während dieser Zeit wird eine Meldebox angezeigt, die die geschätzte Maximalanzahl der Meldungen und den Fortschritt darstellt.

<sup>1</sup> Im Prototypen wird nicht geprüft, ob der Benutzer bereits früher zum Arbeitsbereich hinzugefügt wurde. Es ist also möglich, denselben Benutzer mehrmals einzufügen, was aus Konsistenzgründen vermieden werden sollte. Auch das Löschen eines Benutzers ist bisher nicht implementiert.

## B.5.2 Funktionen im Menü <Objekte>

Das Menü <Objekte> ermöglicht den Zugriff auf die grundlegenden LinkWorks-Objektfunktionen. Diese werden vom LinkWorks-Client direkt ausgeführt, der auch die notwendigen Dialogboxen präsentiert. Da in diesem Fall das Schreibtischfenster in den Vordergrund geholt wird, sollte es nach dem Öffnen des Arbeitsbereiches ikonisiert werden.

Für eine Beschreibung der Objektfunktionen wird auf die LinkWorks-Benutzerdokumentation verwiesen. Nach dem Aktivieren einer Objektfunktion für die selektierten Objekte werden diese an den LinkWorks-Client weitergegeben, der die Funktion asynchron ausführt. Während der Ausführung der Funktion sind die entsprechenden Objekte als „vom Rechner gesperrt“ markiert<sup>1</sup>, mit anderen Objekten kann wie gewohnt weitergearbeitet werden.

Das Verschieben von Objekten erfordert das Öffnen des Zielcontainers. Danach können die zu verschiebenden Objekte selektiert und per Drag&Drop im Zielcontainer abgelegt werden.

## B.5.3 Funktionen im Menü <Infodienst>

Das Menü <Infodienst> enthält Funktionen, zur Steuerung der Darstellung von Ereignismeldungen.

### a) Default-Objektinteresse

Diese Funktion dient dazu, für den aktuellen Benutzer einen Standardwert für das Interesse an zukünftigen Ereignissen festzulegen. Interesse wird je Ereignistyp spezifiziert. Für die Eingabe des Default-Objektinteresses werden dieselben Dialoge benutzt wie für die Eingabe des Objektinteresses<sup>2</sup>.

### b) Objektinteresse

Mit Hilfe der Funktion <Objektinteresse> kann der aktuelle Benutzer Interesse an zukünftigen Ereignissen für die selektierten Objekte<sup>3</sup> bzw. für in diesen enthaltene Objekte anmelden. Wenn für ein Objekt Interesse angemeldet wurde, so wird das durch ein kleines rotes Ausrufungszeichen rechts vom Objekticon angezeigt. Das

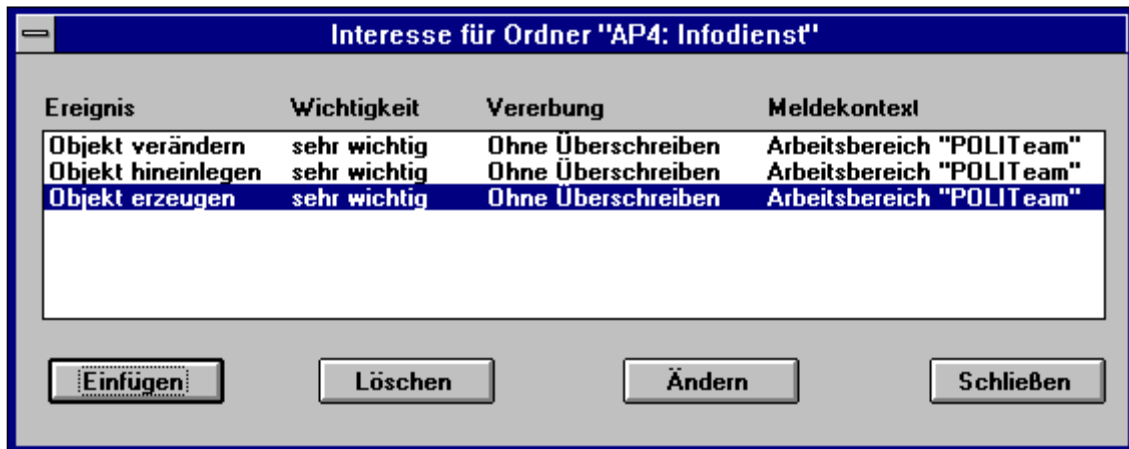


<sup>1</sup> Vgl. Abschnitt B-B.4.1 (c)).

<sup>2</sup> Siehe Abschnitt B-B.5.3 (b)).

<sup>3</sup> Im Prototypen ist bisher nur das Anmelden von Interesse für genau ein selektiertes Objekt implementiert. Sind mehrere Objekte selektiert, ist die Funktion nicht verfügbar.

angemeldete Interesse beeinflusst die Darstellung (Präsentationstechnik) der Meldungen, wenn das Ereignis von einem anderen Benutzer als dem Interesse anmeldenden ausgelöst wird. Interesse wird je Objekt und Ereignistyp angegeben. Nachfolgende Abbildung zeigt die Interesse-Liste für das Ordner-Objekt „AP4: Infodienst“.



In der Spalte *Ereignis* steht der Typ des Ereignisses, an dem Interesse angemeldet wurde. Die Spalte *Wichtigkeit* enthält die Wichtigkeit, die die Auswahl der Darstellungstechnik für die Meldung über das entsprechende Ereignis steuert.

Die fünf möglichen Werte für die Wichtigkeit und die entsprechenden Darstellungstechniken<sup>1</sup> zeigt folgende Tabelle:

Wichtigkeit	Darstellungstechnik
uninteressant	nur auf Anfrage dargestellt (Menü <Infodienst> <Objektmeldungen> bzw. <Infodienst> <Meldungsübersicht>)
interessant	im Objekticon dargestellt
lokal wichtig	im Objekticon und im Ereignismeldebereich des Containers dargestellt
wichtig	im Objekticon und in globaler nicht-modaler Meldebox dargestellt
sehr wichtig	im Objekticon und in modaler Meldebox <sup>2</sup> dargestellt

Die dritte Spalte enthält die Art der *Vererbung*. Vererbung heißt, daß sich das angegebene Interesse nicht nur auf das aktuelle Objekt, sondern auch auf in beliebiger Schachtelungstiefe in diesem enthaltene (also in der Containerhierarchie untergeordnete) Objekte bezieht. Vererbung ist nur bei Objekten der Klasse „Ordner“ möglich.

<sup>1</sup> Im Prototypen sind bisher nur die Darstellungstechniken für *uninteressante*, *interessante* und *sehr wichtige* Meldungen implementiert. Die Wichtigkeitsstufen *lokal wichtig* und *wichtig* werden wie *interessant* behandelt.

<sup>2</sup> Vgl. Abschnitt B-B.4.2 (Meldedienst).

Es gibt die folgenden drei Vererbungsmöglichkeiten:

Vererbungsform	Auswirkungen auf in der Hierarchie untergeordnete Objekte
Keine	Das Interesse wird nicht vererbt. Für Dokumente die einzige mögliche Vererbungsform.
Ohne Überschreiben	Das Interesse wird auf solche untergeordneten Objekte vererbt, bei denen für den entsprechenden Ereignistyp kein Interesse angegeben ist.
Mit Überschreiben	Das Interesse wird auf alle untergeordneten Objekte vererbt, egal ob bei ihnen für den entsprechenden Ereignistyp Interesse angegeben ist oder nicht.

In der vierten Spalte wird der Meldekontext ausgewiesen. Der Meldekontext ist ein dem Objekt übergeordneter Container. Wenn dieser geöffnet ist, wird die Ereignismeldung in der ihrer Wichtigkeit entsprechenden Technik angezeigt.

Die Operationen zum Anmelden und Ändern von Interesse werden durch die Push-buttons <Ändern>, <Löschen> und <Einfügen> ausgelöst.

<Ändern> ermöglicht das Editieren des aktuell ausgewählten Interesse-Eintrages in einer Dialogbox, die in nebenstehender Abbildung dargestellt ist. Das umfaßt das Auswählen der Wichtigkeit und des Meldekontextes sowie das Spezifizieren

der Vererbungsform. Die Dialogbox erzwingt korrekte Eingaben durch das Sperren falscher Kombinationen. Das betrifft insbesondere:

- das Angeben einer Vererbung bei Dokumentobjekten<sup>1</sup>
- das Zulassen von Überschreiben ohne Vererbung<sup>2</sup>

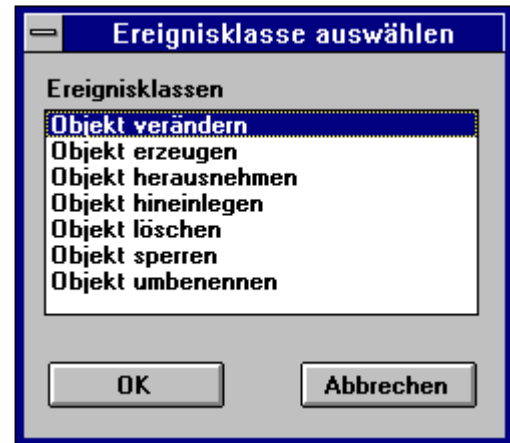
<sup>1</sup> Da Dokumentobjekte keine weiteren Objekte enthalten, können sie auch kein Interesse an enthaltene Objekte vererben.

<sup>2</sup> Überschreiben ist eine Form der Vererbung (siehe oben).

- das Angeben eines Meldekontextes bei einer Wichtigkeit kleiner als „wichtig“<sup>1</sup>.

<Löschen> entfernt den aktuell ausgewählten Eintrag aus der Interesse-Liste.

<Einfügen> fügt einen neuen Eintrag für einen Ereignistyp ein, für den bisher kein Interesse existierte. Nebstehende Dialogbox ermöglicht die Auswahl des Ereignistyps. Danach wird für den neuen Eintrag die Funktion <Ändern> (s.u.) aufgerufen. Die in der Auswahlbox aufgelisteten Ereignistypen hängen von der Klasse der selektierten Objekte ab.



### c) Objektmeldungen

Der Befehl <Objektmeldungen> öffnet die modale Dialogbox *Objektmeldungen*<sup>2</sup>.

Vor dem Öffnen der Dialogbox erfolgt eine Synchronisation mit der Awarenessdatenbank. Während dieser Zeit wird eine Meldebox angezeigt, die die geschätzte Maximalanzahl der Meldungen und den Fortschritt darstellt.

### d) Meldungsübersicht

Der Befehl <Meldungsübersicht> öffnet die modale Dialogbox *Meldungsübersicht*<sup>2</sup>.

Vor dem Öffnen der Dialogbox erfolgt eine Synchronisation mit der Awarenessdatenbank. Während dieser Zeit wird eine Meldebox angezeigt, die die geschätzte Maximalanzahl der Meldungen und den Fortschritt darstellt.

### e) Meldungen quittieren

Die Funktion <Meldungen quittieren> quittiert (löscht) alle Meldungen, die sich auf die selektierten Objekte beziehen. Das ermöglicht es dem Benutzer, für ihn nicht mehr relevante Meldungen zu entfernen.

<sup>1</sup> Meldungen mit einer Wichtigkeit kleiner als „wichtig“ werden nie kontextübergreifend präsentiert.

<sup>2</sup> Siehe Abschnitt B-B.4.2.

## B.6 Der Ordner

Der Ordner ist ein zusammengesetztes Objekt, das andere Objekte enthalten kann. Die Darstellung der Objekte und Meldungen sind identisch mit denen im Gemeinsamen Arbeitsbereich. Das gilt bis auf wenige Ausnahmen auch für die Menüs.



In den Menüs fehlen die Funktionen <Benutzer hinzufügen> und <Default-Interesse>.

Das Menü <Ordner> enthält die Funktionen <Objekte ausrichten><sup>1</sup> und <Ordner schließen>. Letzere schließt das Fenster des Ordners. Falls noch Objekte zum Lesen bzw. Bearbeiten geöffnet oder vom System gesperrt sind, wird eine Fehlermeldung angezeigt. Anderenfalls werden alle noch in der Meldungswarteschlange befindlichen Meldungen in die Datenbank geschrieben. Während dieser Zeit wird eine Meldebox angezeigt, die die geschätzte Maximalanzahl der Meldungen und den Fortschritt darstellt.

---

<sup>1</sup> Siehe Abschnitt B-B.5.1 (a).

---

# Anlage C: Szenario

---

## C.1 Überblick

Das folgende Szenario stellt einige Funktionen des Prototypen anhand eines ausführlichen Beispiels vor und zeigt auf, wie durch die relevanzabhängige Darstellung von Ereignismeldungen der Benutzer in seiner Arbeit unterstützt werden kann.

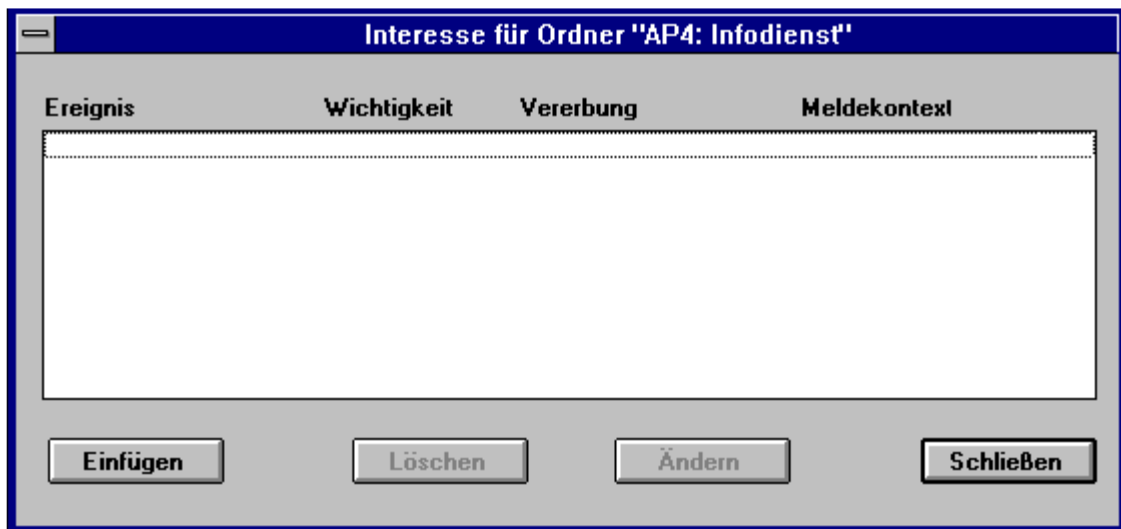
## C.2 Ein mögliches Szenario für die kooperative Arbeit im Gemeinsamen Arbeitsbereich

Der Benutzer Uwe Rauschenbach hat den Gemeinsamen Arbeitsbereich geöffnet. Er erfaßt sofort anhand der Icon-Darstellung, daß außer ihm noch die Kollegin Ilona Butz im Arbeitsbereich aktiv ist und daß die Ordner der Arbeitspakete 2 und 4 Änderungen enthalten, die nicht älter als eine Woche sind.

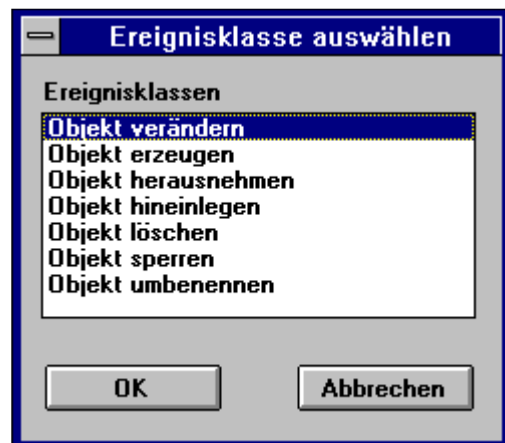
Da Uwe am Arbeitspaket 4 des Projektes mitarbeitet, interessieren ihn Änderungen in diesem Bereich besonders. Wenn im Ordner „AP4: Infodienst“ ein Dokument erzeugt, modifiziert oder hineingelegt wird, möchte er sofort informiert werden. Um sein Interesse anzumelden, selektiert er den Ordner und wählt die Funktion <Objektinteresse> aus dem Menü <Infodienst>.



In der erscheinenden leeren Dialogbox klickt er auf den Button <Einfügen>.

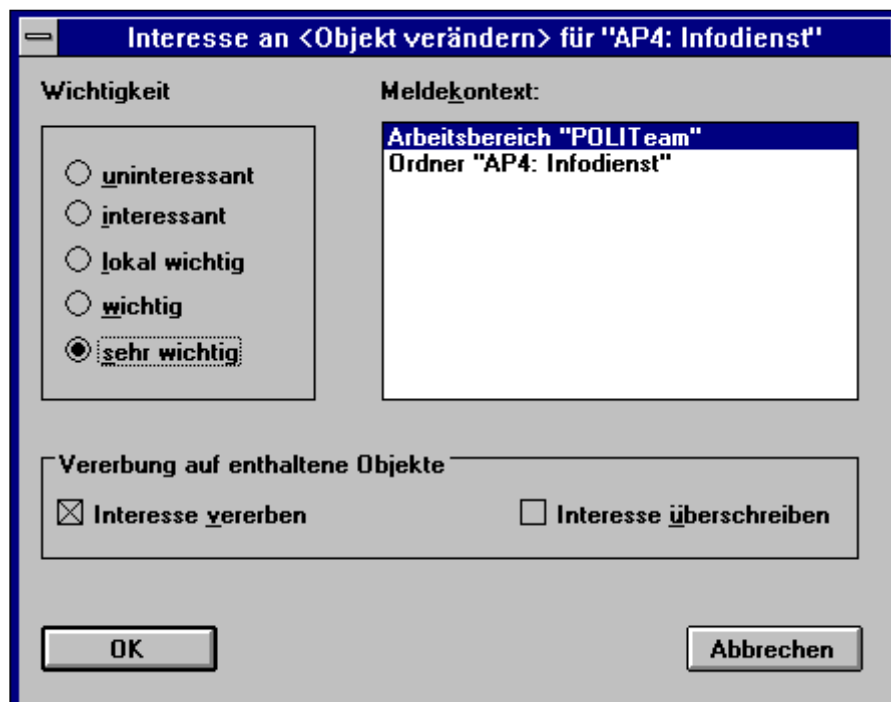


Es erscheint eine Auswahlbox, die ihm alle Klassen von Ereignissen anbietet, an denen er Interesse anmelden kann. Er wählt zunächst die Klasse „Objekt verändern“ aus und klickt auf den Button <OK>.



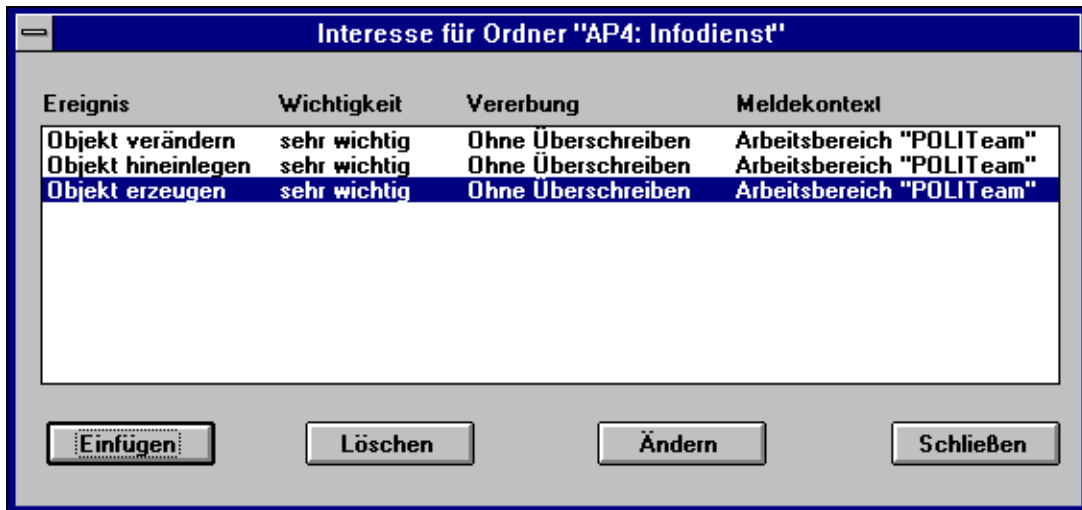
Eine Dialogbox erscheint, in der unser Benutzer sein Interesse genauer spezifizieren kann. Zunächst setzt er die Wichtigkeit auf „sehr wichtig“, um bei einer Objektänderung mit

einer modalen Meldebox informiert zu werden. Die Meldung soll immer angezeigt werden, wenn der Arbeitsbereich geöffnet ist. Das wird durch die Auswahl des Arbeitsbereiches als Meldekontext erreicht. Da das Interesse für alle

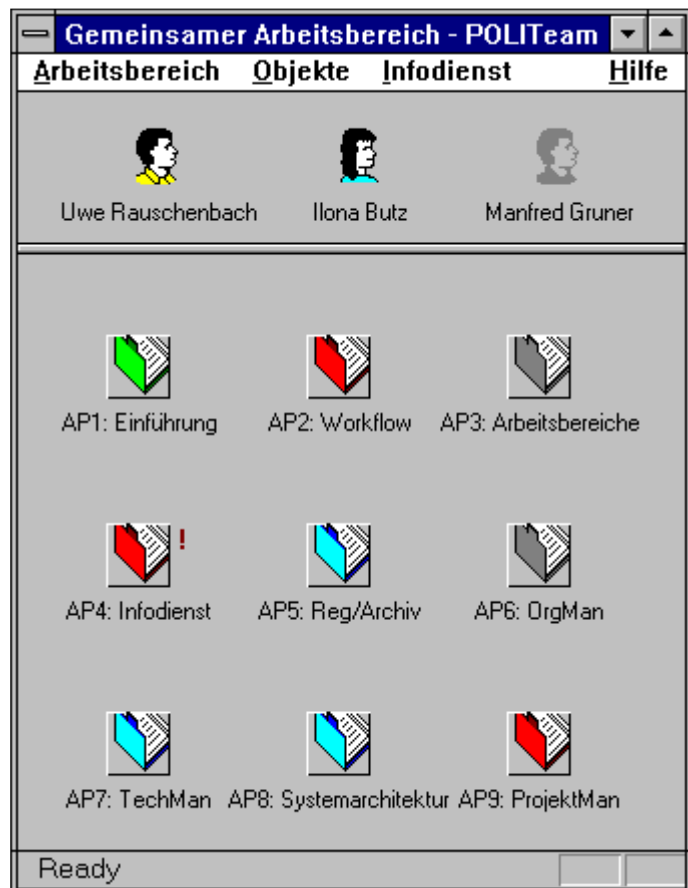


Objekte gelten soll, die im Verlauf der Bearbeitung des Arbeitspaketes entstehen, aktiviert er zusätzlich das Kontrollkästchen <Interesse vererben>. Das Kontrollkästchen <Interesse überschreiben> bleibt deaktiviert, damit evtl. später angegebenes Interesse für im Ordner enthaltene Objekte nicht überschrieben wird.

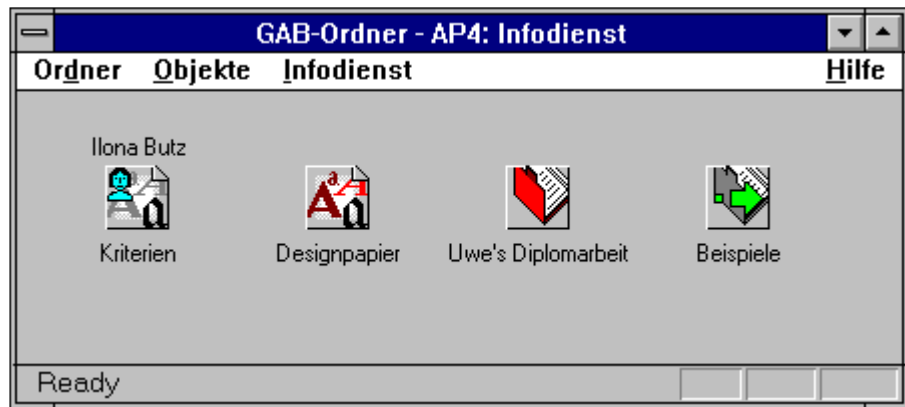
Diese Schrittfolge wird für die Ereignisklassen „Objekt hineinlegen“ und „Objekt erzeugen“ wiederholt, bis schließlich die Interesse-Liste mit den folgenden drei Einträgen gefüllt ist:



Durch Klicken auf den Button <Schließen> wird die Dialogbox geschlossen. Das angegebene Interesse ist jetzt in der Datenbank gespeichert und wird ab sofort bei der Erzeugung neuer Meldungen berücksichtigt. Durch ein kleines rotes Ausrufungszeichen rechts vom Icon des AP4-Ordners zeigt das System an, daß für dieses Objekt benutzerdefiniertes Interesse vorliegt.

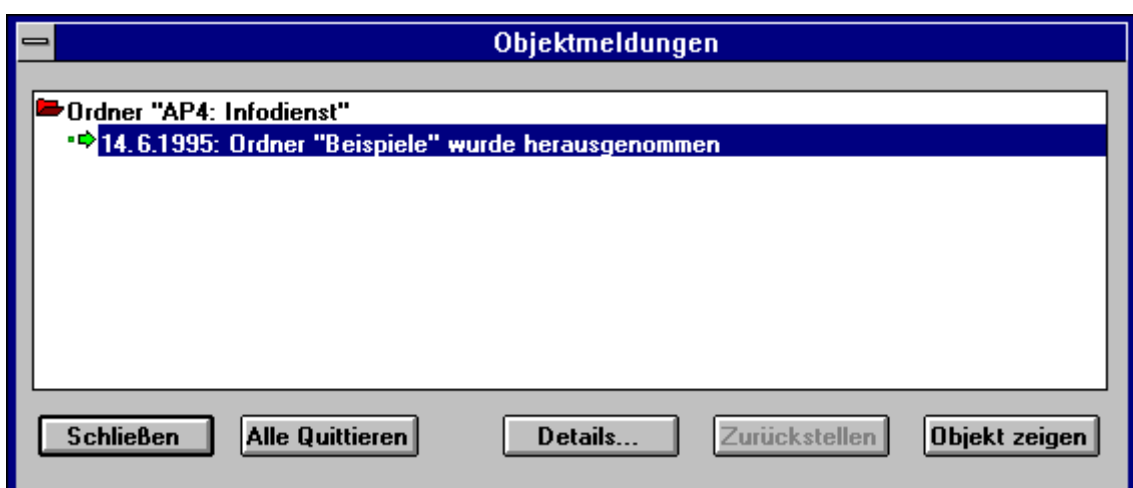
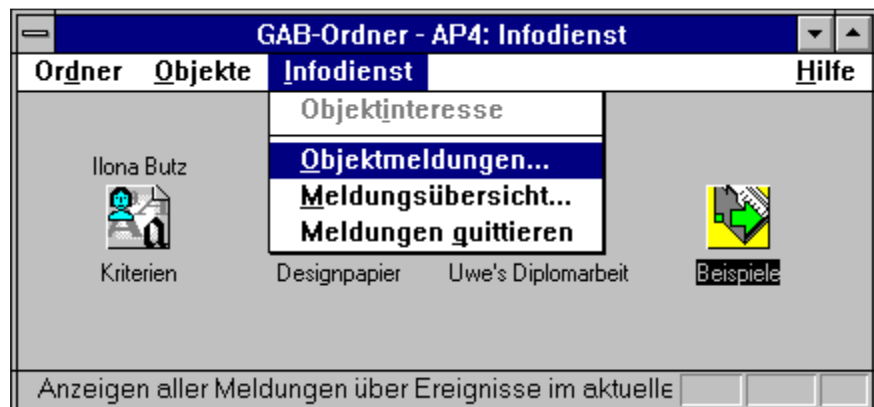


Nachdem der Benutzer Uwe sein Interesse an den oben genannten Ereignissen angemeldet hat, öffnet er den Ordner „AP4: Infodienst“.



Er erkennt, daß seine Kollegin Ilona Butz gerade das Kriterienpapier bearbeitet, daß es in der letzten Woche Änderungen in seiner Diplomarbeit und am Designpapier gegeben hat und daß der Beispiel-Ordner verschoben wurde. Da Uwe evtl. noch Dokumente aus diesem Ordner benötigt, möchte er gern dessen neue Lage feststellen.

Er selektiert das Icon des Ordners und aktiviert die Funktion <Objektmeldungen> des Infodienstes. Die Meldung wird in einer Meldebox dargestellt.

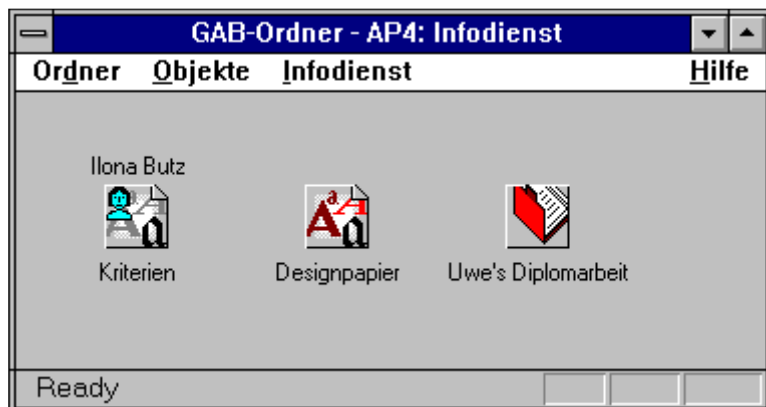


Nach dem Selektieren der Ereignismeldung und dem Mausklick auf den Button <Details> zeigt das System die

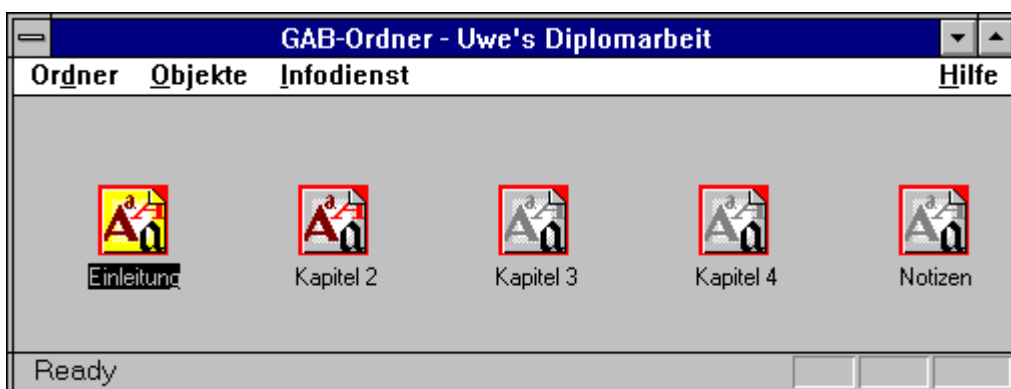


Meldung noch einmal detaillierter in einer Dialogbox an. Hier kann abgelesen werden, daß das Objekt sich jetzt im Ordner „AP9: ProjektMan“ befindet.

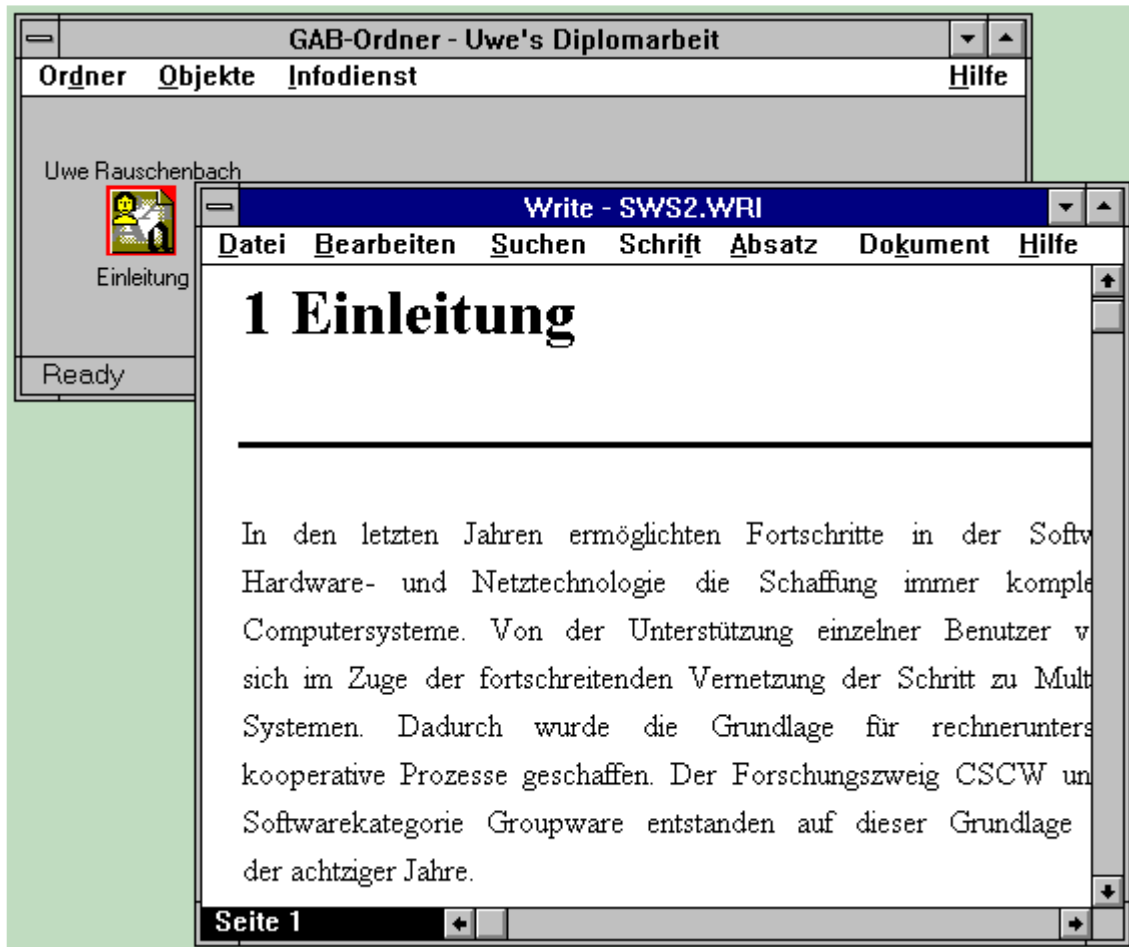
Der Benutzer schließt die Detailbox durch Klicken auf den Aktionsknopf <OK>. Da ihm die Meldung über das Verschieben des Objektes nun bekannt ist, quittiert er sie durch einen Mausklick auf den Aktionsknopf <Alle Quittieren>. Damit wird die Meldung gelöscht, und das Ghost-Objekt „Beispiele“ verschwindet.



Um an seiner Diplomarbeit weiterzuarbeiten, öffnet Uwe nun den Ordner „Uwe's Diplomarbeit“ durch einen Doppelklick auf das korrespondierende Icon. Der Text „Einleitung“ soll bearbeitet werden.



Durch einen weiteren Doppelklick wird das Werkzeug WRITE gestartet, und gleichzeitig wird das Icon für Uwe als „geöffnet und gesperrt“ dargestellt.

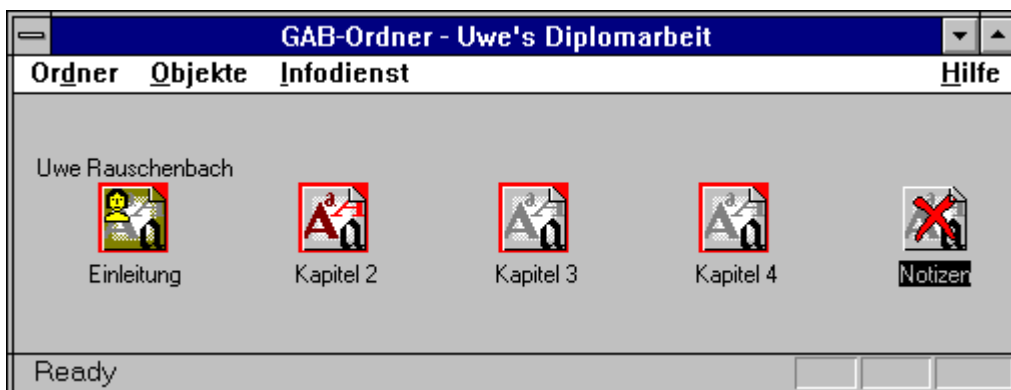


Für alle anderen Kooperationspartner wird die Meldung erzeugt, daß dieses Dokument jetzt für eine Bearbeitung gesperrt ist. Das folgende Bild zeigt, wie sie an der Benutzeroberfläche von Ilona Butz dargestellt wird.

Die asynchrone Benutzerschnittstelle gestattet es, parallel zur laufenden Bearbeitung eines Objektes in einem externen Werkzeug oder im Groupware-System LinkWorks andere Objekte weiterhin zu bearbeiten. Beispielsweise könnte der Benutzer Uwe Rauschenbach das Dokument „Notizen“ löschen, da es nicht mehr benötigt wird. Dazu selektiert er es per Mausklick und wählt den Befehl <löschen> aus dem Menü <Objekte>. Der Löschbefehl wird nun vom Groupwaresystem ausgeführt. Dazu wird das Dokument für Uwe Rauschenbach als „vom Rechner gesperrt“ dargestellt. Andere Benutzer erhalten diese Information nicht.



Nach der Ausführung des Löschbefehls wird das gelöschte Objekt als „gelöscht“ markiert und die Meldung über das Löschereignis an alle interessierten Benutzer abgesandt, die Zutritt zum Arbeitsbereich haben.



Inzwischen hat Ilona Butz die Bearbeitung des Kriterienkataloges abgeschlossen und die Änderungen abgespeichert. Da Uwe Rauschenbach Interesse an Änderungsereignissen mit der Wichtigkeit „sehr wichtig“ angemeldet hat, stellt das System jetzt die Meldung über die Änderung in der Dialogbox des Meldedienstes dar.



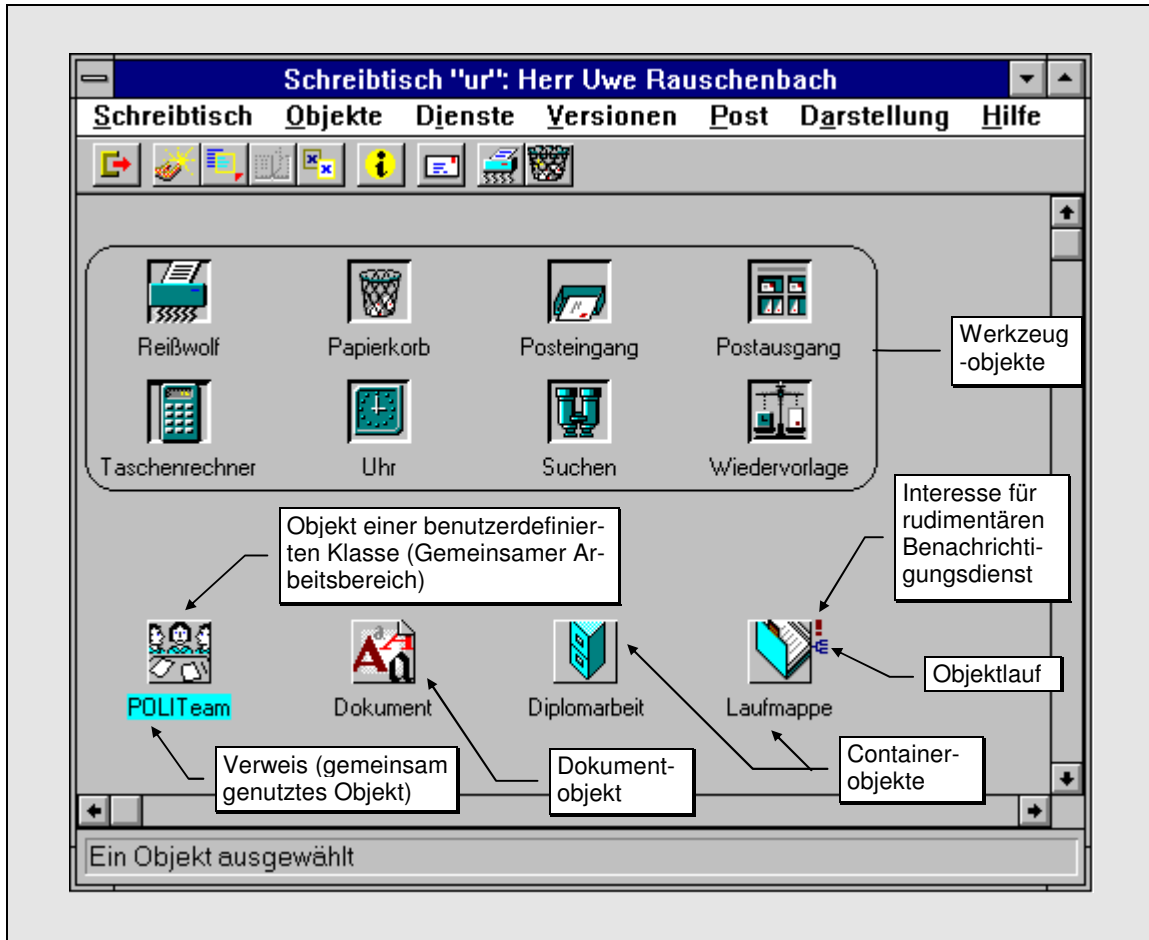
Mit einem Mausklick auf den Button <Alle Gesehen> wird die Meldung als „gelesen“ markiert und die Meldebox geschlossen. Die Meldung über die Änderung wird demzufolge nur noch in der Farbe des Icons „Kriterien“ dargestellt.

Dieses Szenario sollte eine kurze handlungsorientierte Einführung in die Möglichkeiten geben, die die relevanzgesteuerte Darstellung von Meldungen über Benutzeraktivitäten für die Unterstützung kooperativer Prozesse bereitstellt.

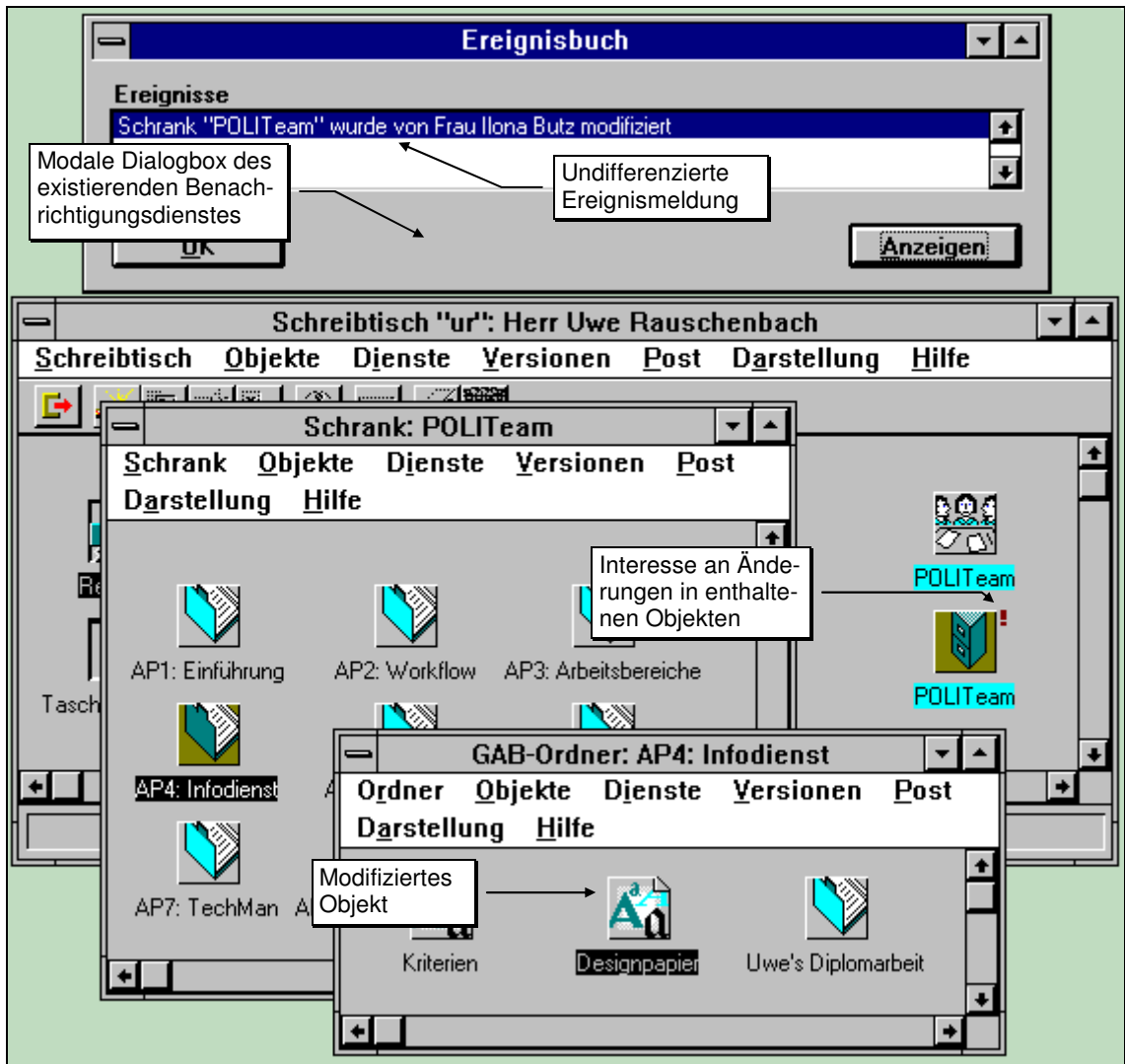
---

# Anlage D: Farbtafeln

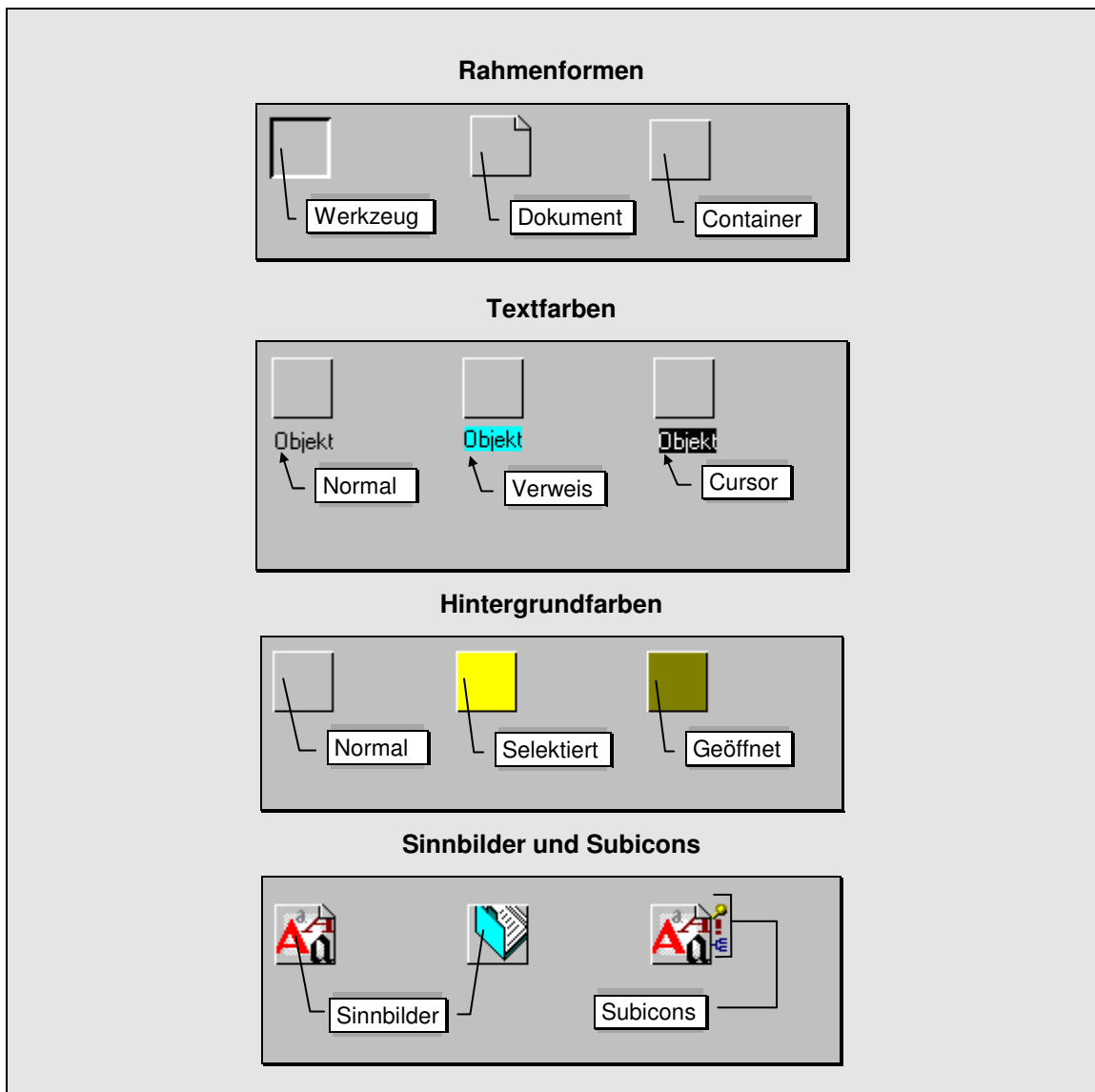
---



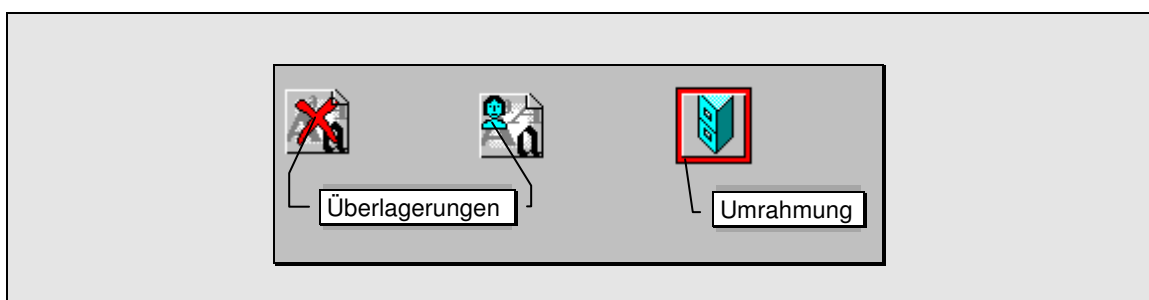
Farbtafel 4.1: Der LinkWorks-Schreibtisch



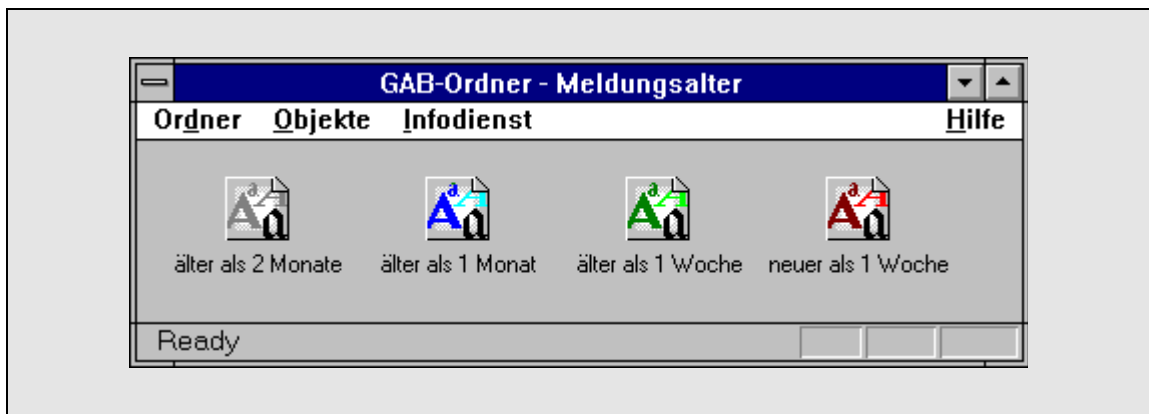
Farbtafel 4.2: Der existierende Benachrichtigungsdienst



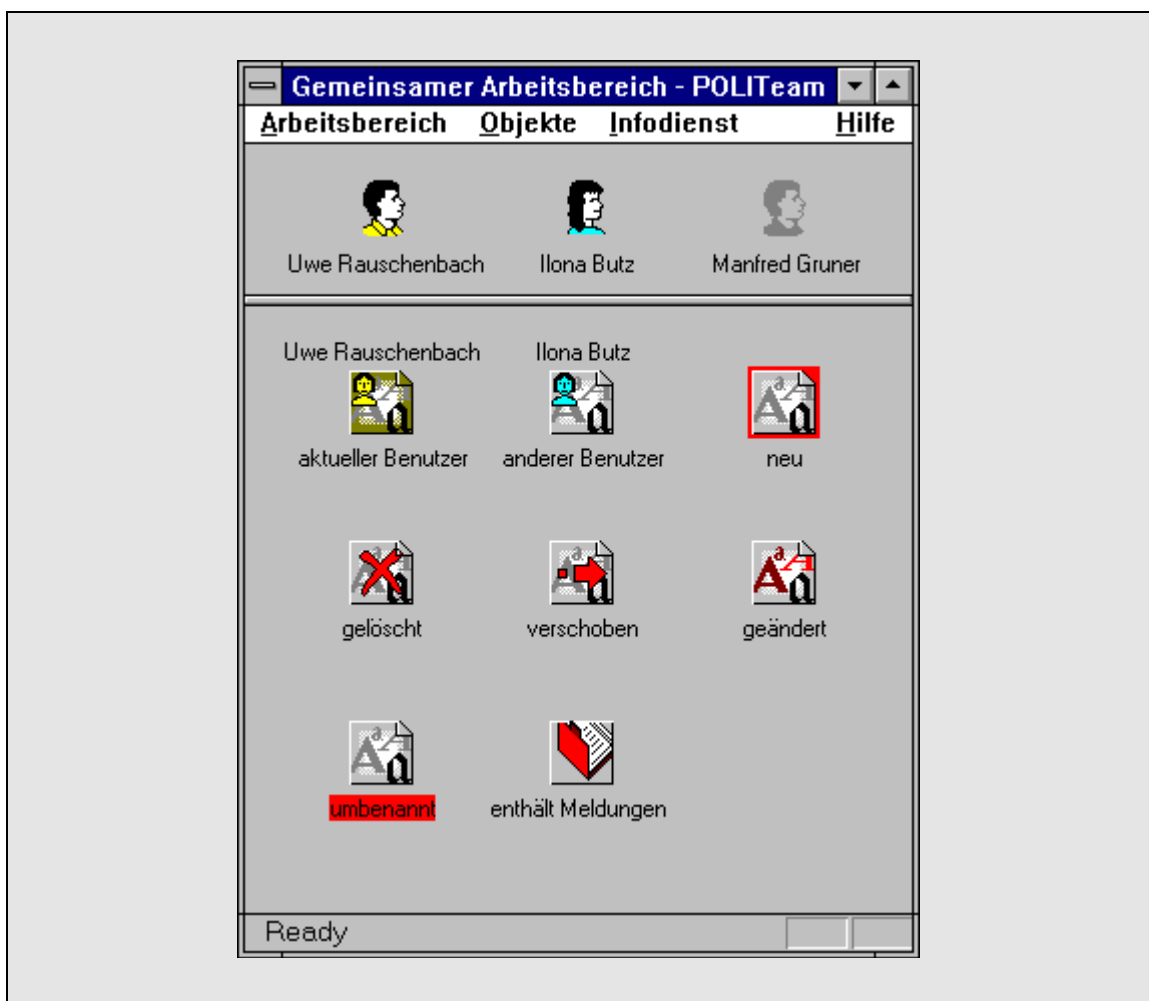
**Farbtafel 6.1:** Das LinkWorks-Grafikalphabet



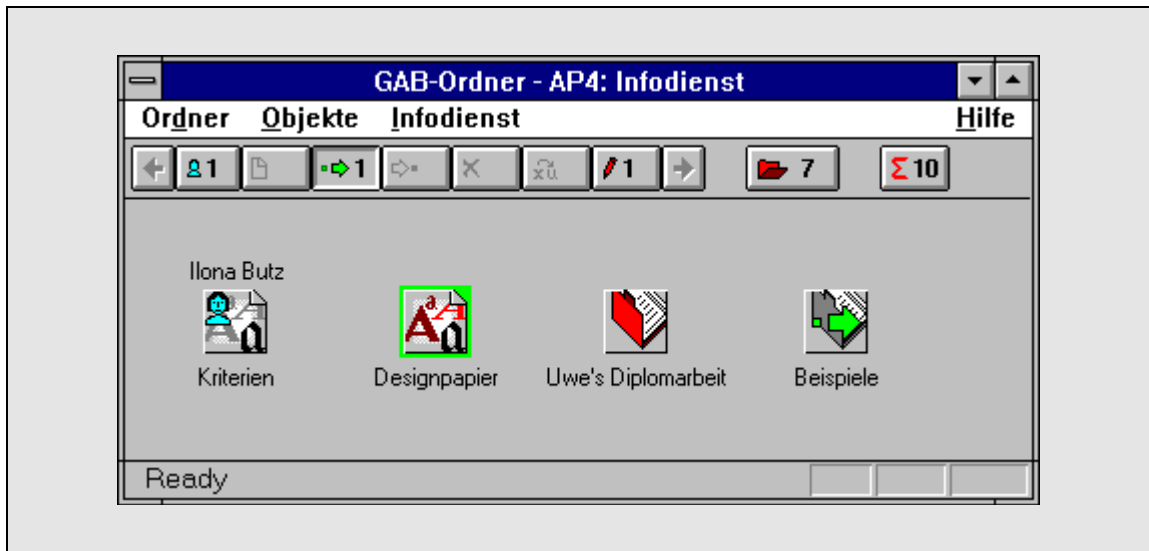
**Farbtafel 6.2:** Erweiterung des LinkWorks-Grafikalphabetes



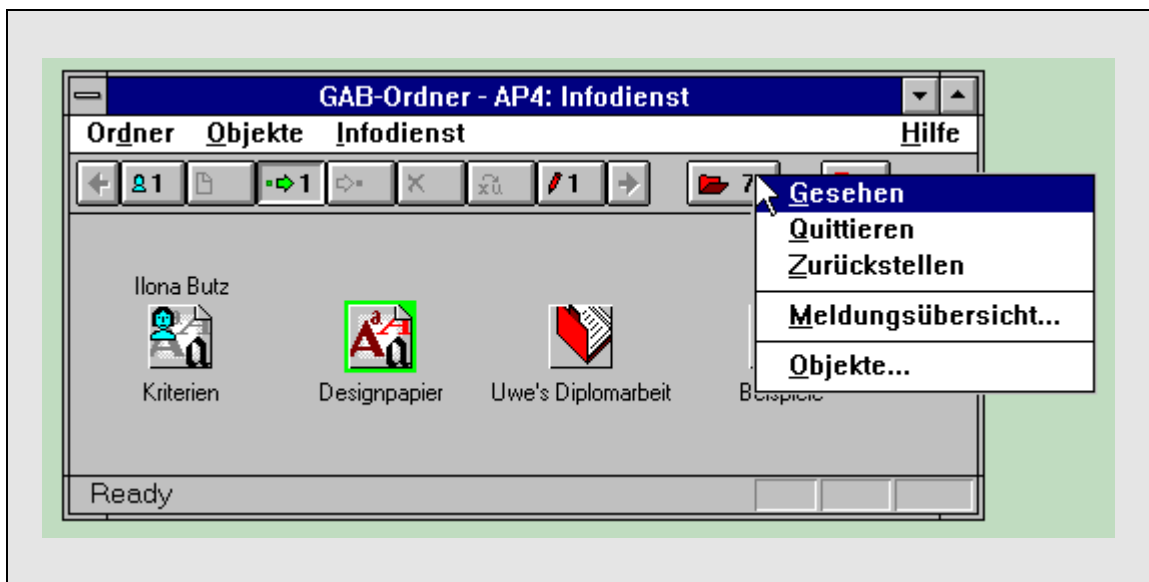
Farbtafel 6.3: Darstellung des Ereigniszeitpunktes



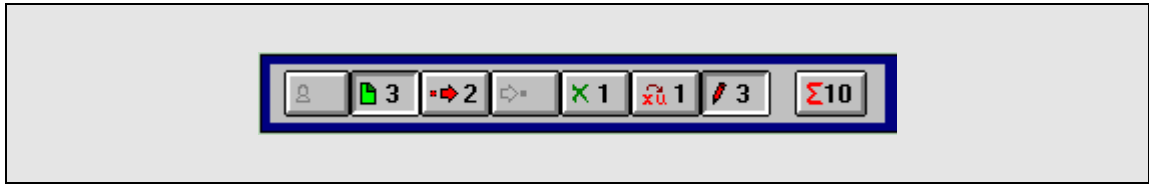
Farbtafel 6.4: Darstellung von Ereignistypen und propagierten Meldungen



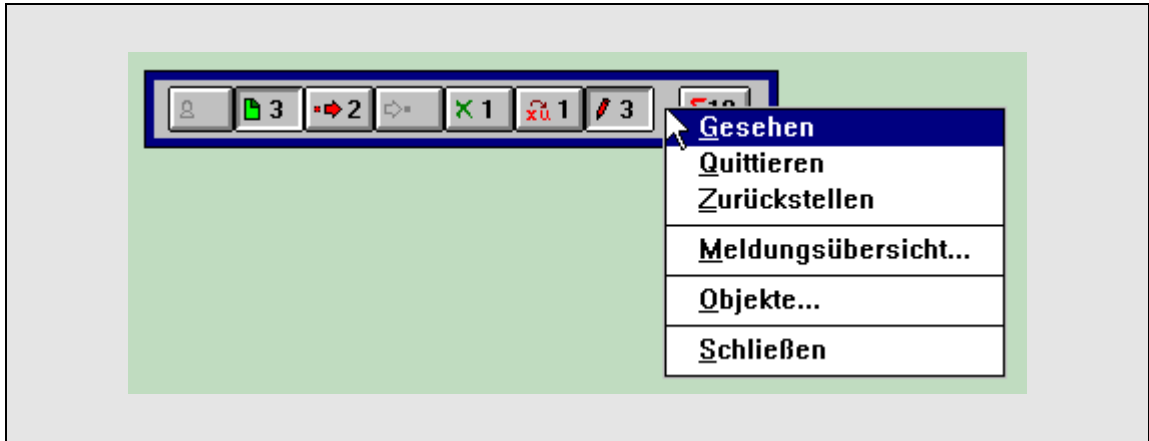
Farbtafel 6.5: Kontextgebundene Darstellung von Ereignismeldungen



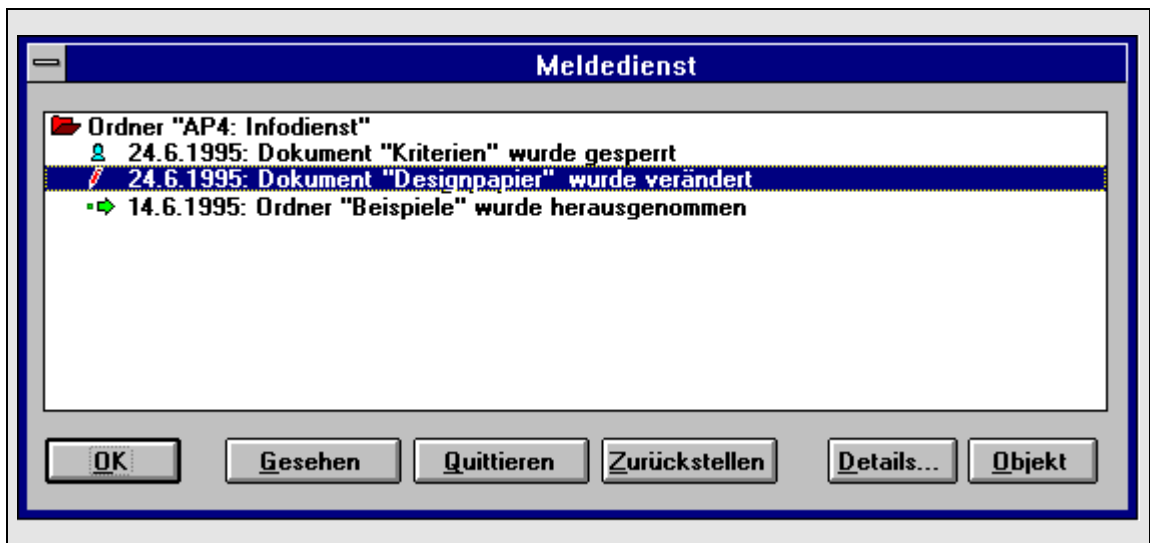
Farbtafel 6.6: Interaktivität der kontextgebundenen Darstellungstechnik



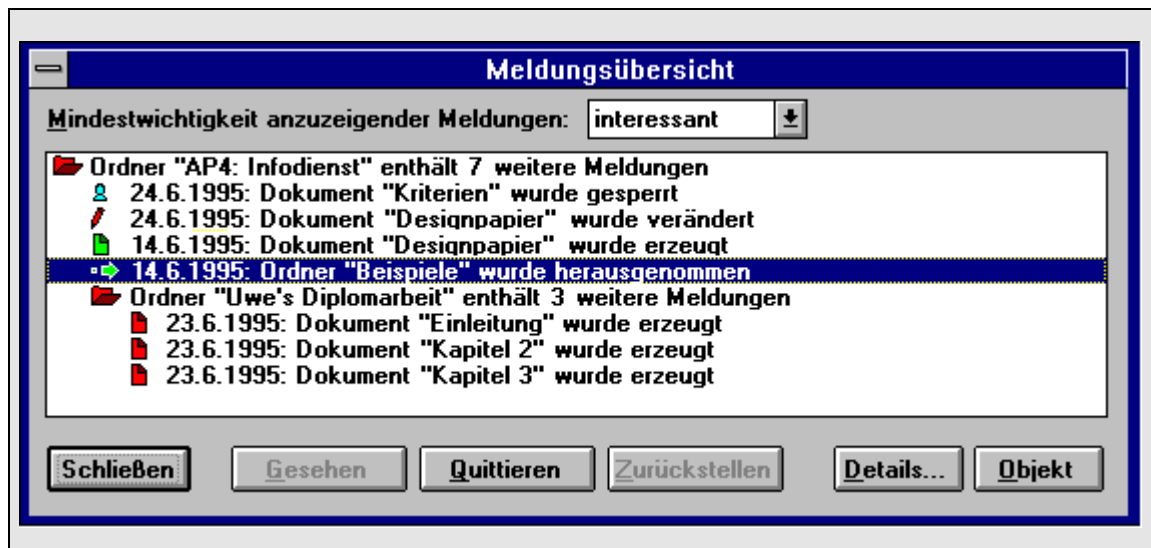
Farbtafel 6.7: Kontextübergreifende nicht-modale Darstellung von Ereignismeldungen



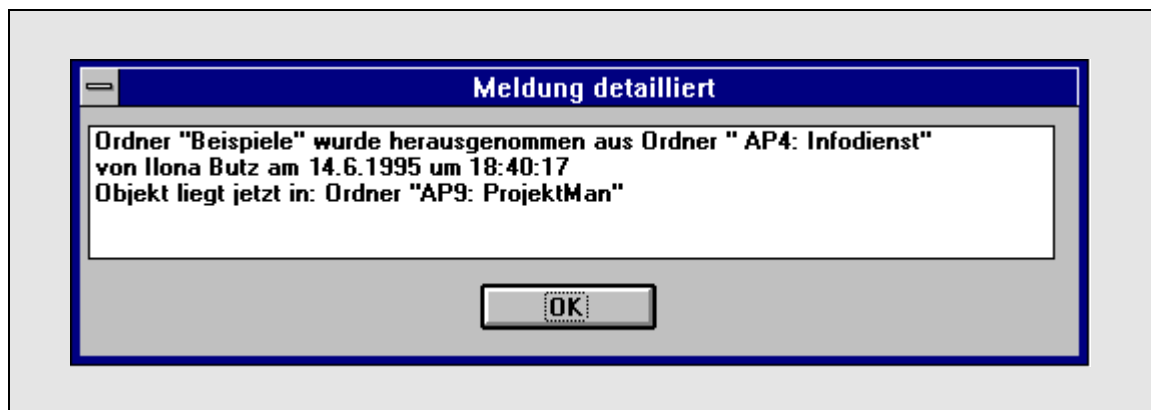
Farbtafel 6.8: Interaktivität der kontextübergreifenden nicht-modalen Darstellungstechnik



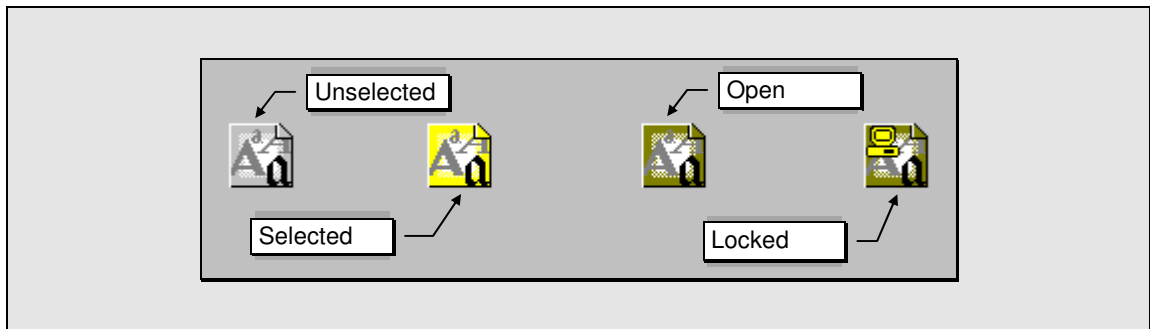
Farbtafel 6.9: Kontextübergreifende modale Darstellung von Ereignismeldungen



Farbtafel 6.10: Benutzerinitiiert aufrufbare Meldungsliste mit maximaler Interaktivität



Farbtafel 6.11: Detaillierte textliche Darstellung des gemeldeten Ereignisses



**Farbtafel 7.1:** Codierung der lokalen Objektzustände im Objekticon

---

# Selbständigkeitserklärung

---

Hiermit erkläre ich, daß ich die vorliegende Arbeit selbständig und nur unter Vorlage der angegebenen Literatur und Hilfsmittel angefertigt habe.

Rostock, 15. September 1995

.....  
Uwe Rauschenbach

---

# Thesen

---

- I. Die Darstellung von Information über die Aktivitäten anderer Benutzer bei der Bearbeitung gemeinsam genutzter Objekte ist ein Kriterium, das ein Groupware-System von einem Multi-User-System unterscheidet.
- II. Information über die Aktivitäten von Kooperationspartnern schafft und erhält einen gemeinsamen Kontext für die kooperative Arbeit und kann so deren Effektivität auch beim Arbeiten über große Entfernungen in virtuellen gemeinsamen Arbeitsbereichen erhöhen.
- III. Der Informationsbedarf eines Benutzers bezieht sich sowohl auf parallel zur eigenen Arbeit verlaufende Aktivitäten als auch auf die Ergebnisse vergangener.
- IV. Meldungen über die Aktivitäten anderer Benutzer müssen benutzerdefiniert gewichtet und gefiltert werden können, um eine Informationsüberflutung zu vermeiden.
- V. Die Aktivitäten eines Benutzers in einem gemeinsamen Arbeitsbereich lassen sich durch Ereignisse beschreiben. Es wurde ein dreistufiges Modell entwickelt, das für jeden Benutzer ausgehend von dessen *Interesse* an einem eintretenden *Ereignis* eine *Meldung* mit entsprechender Relevanz erzeugt. Besonderer Wert wurde auf die Trennung eines Ereignisses von den benutzerspezifischen Meldungen über dieses Ereignis gelegt.
- VI. Eine Meldung hat einen Lebenszyklus, der durch Zustandsübergänge gekennzeichnet ist. Sie wird innerhalb der Objekthierarchie in übergeordnete Kontexte weitergeleitet. Durch Interaktion mit der Meldung kann der Benutzer Zustandsübergänge auslösen oder weitere Information erhalten.

- VII. Präsentationstechniken dienen zur Darstellung von Meldungen. Fünf Techniken mit abgestufter Dringlichkeit wurden entwickelt. Eine Meldung wird mit einer Präsentationstechnik dargestellt, deren Dringlichkeit der Relevanz und dem Zustand der Meldung entspricht. Das Filtern der Meldungen erfolgt durch Meldekontexte.
- VIII. Bei der systemtechnischen Umsetzung des Meldedienstes muß zwischen lokalem und entferntem Echo unterschieden werden. Während die Meldung über ein Ereignis sofort dem Benutzer präsentiert werden muß, der es ausgelöst hat, sind für entfernte Benutzer längere Verzögerungen zulässig.
- IX. Der Benachrichtigungsdienst des existierenden Systems LinkWorks Version 3.0 wurde untersucht. Seine Funktionalität erfüllt nicht die in dieser Arbeit gesteckten Anforderungen.
- X. Die von der Programmierschnittstelle des Systems LinkWorks zur Verfügung gestellten Möglichkeiten reichen für eine Integration des entwickelten Modells nicht aus.
- XI. Auf der Basis der eingeschränkten Erweiterungsmöglichkeiten von LinkWorks und einer externen Datenbank wurde ein Prototyp implementiert, der einige Aspekte des entwickelten Modells umsetzt.
- XII. Die vorliegende prototypische Lösung erfordert einen hohen Aufwand für die Konsistenzsicherung und geht auf Kosten der Offenheit und Erweiterbarkeit des Systems. Sie kommt daher für ein praxisreifes System nicht in Frage.