



Forschungszentrum Karlsruhe
Technik und Umwelt

Wissenschaftliche Berichte
FZKA 5921

**Adaption eines elektronischen
Laborbuchs (Elab) an ein
objektorientiertes Datenbank-
Management-System**

**K.-U. Stucky, H. Eggert, A. Schmidt,
R. Reiger**

Institut für Angewandte Informatik

Juli 1997

FORSCHUNGSZENTRUM KARLSRUHE

Technik und Umwelt

Wissenschaftliche Berichte

FZKA 5921

Adaption eines elektronischen Laborbuchs (Elab) an ein objektorientiertes Datenbank-Management-System

K.-U. Stucky, H. Eggert, A. Schmidt, R. Reiger*

Institut für Angewandte Informatik

*) Daimler Benz AG, München

Forschungszentrum Karlsruhe GmbH, Karlsruhe

1997

Als Manuskript gedruckt
Für diesen Bericht behalten wir uns alle Rechte vor

Forschungszentrum Karlsruhe GmbH
Postfach 3640, 76021 Karlsruhe

Mitglied der Hermann von Helmholtz-Gemeinschaft
Deutscher Forschungszentren (HGF)

ISSN 0947-8620

Adaption of an Electronic Laboratory Book (Elab) to an Object Oriented Database Management System

Abstract

The Electronic Laboratory Book Elab is a tool which supports the development of manufacturing processes in microsystem engineering. Processes and process sequences are presented and can be modified by the user. Development of variants is supported, too.

Elab is an object oriented system. Multi user operation, access control and versioning require support of an object oriented database management system. Therefore the database system ObjectStore containing a Smalltalk interface (OSST) has been selected.

Parallel access of different applications on persistent data requires special locking mechanisms based on the low level locking mechanisms of ObjectStore database. A high level mechanism has been developed which allows semantic locking on an object based level.

During multi user operation users are members of different project groups and only have access to data belonging to their own groups. A Superuser is responsible for managing user data.

Adaption eines elektronischen Laborbuchs (Elab) an ein objektorientiertes Datenbank-Management-System

Zusammenfassung

Das Elektronische Laborbuch Elab ist ein Softwarewerkzeug zur Unterstützung der Entwicklung von Herstellungsprozessen von Mikrosystemen. Es dient der Darstellung und dem Modifizieren von Prozessen und Prozeßreihen. Dabei wird auch die Variantenbildung unterstützt.

Elab ist ein objektorientiert aufgebautes System. Mehrbenutzerbetrieb, Zugriffsberechtigungen und Versionsverwaltung machen die Anbindung an ein objektorientiertes Datenbank-Management-System erforderlich. Aus diesem Grund wurde das Datenbanksystem ObjectStore mit Sprachanbindung zu Smalltalk (OSST) ausgewählt.

Der parallele Zugriff von verschiedenen Applikationen auf den persistenten Datenbestand erfordert spezielle Sperrmechanismen auf der Grundlage einfacherer Mechanismen der ObjectStore Datenbanken. Durch die Implementierung von semantischen Sperrern kann objektweises Sperrern erreicht werden.

Im Mehrbenutzerbetrieb gehören die Benutzer bestimmten Projektgruppen an, die nur Zugriff auf die Daten ihres Projekts haben. Ein Superuser übernimmt die Aufgaben der Benutzerverwaltung.

Inhalt

1	Einleitung	7
2	Beschreibung des Elektronischen Laborbuchs	9
2.1	Installation des Systems	9
2.1.1	Dateien	9
2.1.2	Die Installation	10
2.2	Funktionalität des Systems	12
2.2.1	Dialogfenster	12
2.2.2	Anmelden beim System	15
2.2.3	Das Hauptfenster	17
2.2.4	Die Fragmentenverwaltung	27
2.2.5	Erzeugung und Verwaltung von Protokollen	39
2.3	Produktverwaltung & Statistik	51
3	Datenbankanbindung	53
3.1	Das Datenbank-Management-System	53
3.2	Wichtige Klassen von Elab	55
3.3	Datenbanken des Systems Elab	58
4	Semantische Sperren	59
4.1	Die Notwendigkeit zusätzlicher Sperrmechanismen	59
4.2	Klassen für Sperrverwaltung und Datenbankanbindung	61
5	Mehrbenutzerbetrieb	65
6	Zusammenfassung	69
	Anhang: Quellcodedokumentation	71
	Klasse DBObject	71
	Klasse ElabApplicationModel	78
	Klasse ElabCalculator	85
	Klasse ElabLock	87
	Klasse ElabLockDaemon	93
	Klasse ElabLogout	95
	Klasse ElabMain	96
	Klasse ElabProcessEditor	101

Klasse ElabWorkspace	108
Klasse Login	109
Klasse ObjectWithHistory	117
Klasse Password	121
Klasse PasswordDialog	123
Klasse User	125
Klasse UserModel	132
Literatur	141

1 Einleitung

Im Rahmen des BMBF-Verbundprojektes METEOR wurden Softwarewerkzeuge für die Unterstützung der fertigungs-, qualitäts- und montagegerechten Konstruktion von Mikrosystemen entwickelt. Das **Elektronische Laborbuch Elab** ist eines dieser Werkzeuge. Es dient dem Prozeßingenieur bei der Entwicklung von Herstellungsverfahren für Mikrosysteme [1][2].

Die Ziele dieser Entwicklung sind die Optimierung von Prozessen und Prozeßsequenzen und die Verifikation der Fertigbarkeit eines Systems hinsichtlich der Prozeßsequenzen.

Diese Ziele zu erreichen, ist keine triviale Aufgabenstellung. Obwohl für einzelne Prozesse in der Mikrosystemtechnik die Technologie gut beherrscht wird, bereitet die Integration dieser Prozesse im Rahmen des gesamten Herstellungsverfahrens Probleme, etwa durch Inkompatibilitäten. Zu deren Lösung werden i. allg. viele verschiedene Versuchsreihen notwendig sein. Daneben wird in bestimmten Teilen eines Gesamtprozesses auch mit dem Einsatz neuer Technologien zu rechnen sein. Bestimmte Vorgehensweisen wie z.B. Chargensplitting werden bei der Lösung der geschilderten Probleme eingesetzt.

Als unterstützendes Werkzeug besitzt das Elektronische Laborbuch verschiedene Eigenschaften, die mit der Speicherung, Darstellung, Modifikation und Verbreitung von Daten und komplexen Wissensinhalten über den Herstellungsprozeß eines Mikrosystems zusammenhängen. Durch konsequentes Arbeiten mit diesem System können die Entwickler von den Erkenntnissen und Erfahrungen ihrer Kollegen profitieren und ihr eigenes Wissen anderen zur Verfügung stellen. Prozeßabläufe werden nach einem für alle gleichermaßen verständlichen Schema geplant und editiert. Die administrative Verwaltung von Labormustern und deren Eigenschaften ist Aufgabe von Elab und entlastet damit die Entwickler [3]. Auch die Bedeutung des Systems für die Dokumentation von Prozeßwissen ist hier hervorzuheben.

Dem System Elab liegt ein objektorientiertes Design zugrunde. Die wichtigsten Klassen sind **Prozess** mit allgemeinen Attributen und prozeßbezogenen Parametern, **Fragment** als Folge von Prozessen und **Protokoll** als Dokumentation der Vorgehensweise der Entwickler bei der Bildung von Varianten.

Die Programmierung erfolgte in der Sprache Smalltalk, die Implementierung unter Windows und Solaris. Ferner basiert die Benutzeroberfläche auf der Model-View-Controller Architektur, wie sie innerhalb der Smalltalk-Umgebung vorgesehen ist.

Die hier genannten Klassen speichern die wesentlichen Informationen über Prozesse und Prozeßfolgen sowie über die Historie der Entwicklung eines Herstellungsverfahrens für ein Mikrosystem. Diese Daten müssen persistent gehalten werden. Verschiedene Entwickler arbeiten zusammen in einem Projekt und sollen jederzeit Zugang zu den vorhandenen Daten ihres Projekts erhalten. Dieser Zugang muß in bestimmten Fällen eingeschränkt werden, etwa damit dieselben Daten nicht gleichzeitig von mehreren Benutzern editiert werden können. Außerdem sollen sich die Benutzer nur innerhalb der Grenzen ihres Projekts bewegen können.

Die genannten Eigenschaften führen konsequenterweise zur Einrichtung eines Mehrbenutzerbetriebs mit den erforderlichen Zugriffsberechtigungen bzw. -beschränkungen und zur Adap-

tion des Elektronischen Laborbuchs Elab an ein Datenbank-Management-System. Unter dem Gesichtspunkt des objektorientierten Designs von Elab wurde das objektorientierte Datenbanksystem ObjectStore (Version 3.1) verwendet. Das System Elab selbst wurde mit VisualWorks Version 2.0 implementiert; ObjectStore für Smalltalk lag in Version 1.0 vor.

ObjectStore ist ein mehrbenutzerfähiges, objektorientiertes Client/Server-System. Dabei ist die Grundidee, Datenstrukturen der jeweiligen Programmiersprache persistent zu machen. Für ein objektorientiertes Softwaresystem bedeutet das die Verwaltung von Instanzen der Klassen des Systems.

Schwierigkeiten, die es bei der Datenbankadaption des Elektronischen Laborbuchs zu überwinden galt, lagen zum einen darin begründet, daß ObjectStore bei Transaktionen nicht auf einzelne Objekte getrennt sondern auf größere Einheiten von Daten zugreift und diese auch für die Dauer einer Transaktion sperrt. Hierzu war die Konzeption und Implementierung von semantischen Sperren notwendig. Zum anderen mußten je nach Zugriffsberechtigung eines Benutzers bereits in der Oberfläche bestimmte Teile der Elab-Funktionalität gesperrt werden [4].

Das zweite Kapitel dieses Handbuchs beschreibt die Funktionalität des Elektronischen Laborbuchs. Es gibt damit auch einen Überblick über die Begriffswelt des Systems.

Die weiteren Kapitel befassen sich mit der Anbindung von Elab an das Datenbank-Management-System ObjectStore. Sie beschreiben die dabei auftretenden Probleme sowie deren Lösung durch Konzeption und Implementierung von semantischen Sperren und eines Mehrbenutzerbetriebs.

2 Beschreibung des Elektronischen Laborbuchs

2.1 Installation des Systems

2.1.1 Dateien

Zum Elektronischen Laborbuch Elab gehören folgende Dateien:

- elab.cfg

Diese Konfigurationsdatei enthält den Pfadnamen des Datenbank-Verzeichnisses. Der Name muß am Anfang die Bezeichnung des Server-Rechners enthalten. Die Datei elab.cfg muß in Smalltalk durch das Versenden von Nachrichten erzeugt worden sein. Beim Installationsprozeß geschieht das automatisch. I. allg. wird ein Editieren von Hand nicht erfolgreich sein.

- elab.cha

Diese Datei enthält alle Änderungen, die im Smalltalk-System seit Beginn der Arbeiten durchgeführt wurden. Sie ist sehr groß. Ohne diese Datei wird der Code aller von den Elab-Entwicklern hinzugefügten Klassen und Methoden bei einer Weiterentwicklung decompiliert. In diesem Fall könnten Dokumentationen, Kommentare und Bezeichner nicht mehr zur Verfügung gestellt werden.

Es ist lediglich noch möglich, aus elab.cha alle Einträge bis auf die jeweils letzten bzgl. einer Klasse oder Methode zu entfernen (s. Handbuch *User's Guide, rev. 1.1* von VisualWorks, Chapter 11, p. 164: "Condensing the changes file")

- elab.im

Auch diese Datei ist sehr groß. Sie enthält das sogen. Image des Systems. Darin enthalten sind sämtliche Klassen und Methoden von Smalltalk, sowie alle Erweiterungen von ObjectStore, Elab und anderer bei der Implementierung verwendeter Software.

elab.im kann auch verkleinert werden, indem alle Klassen, die nicht benötigt werden, entfernt werden. Dabei handelt es sich im wesentlichen um die Entwicklungswerkzeuge von VisualWorks. Für die Verkleinerung des Image s. VisualWorks *User's Guide, rev. 1.1, Chapter 12, p. 165*).

- elab.sou

Hier ist der Quellcode des Smalltalk-Systems abgelegt. Fehlt diese Datei, tritt derselbe Effekt auf, wie für hinzugefügte Klassen und Methoden ohne elab.cha. Kommentare und Bezeichner sind im decompilierten Code nicht vorhanden.

- login

Hier handelt es sich um eine Datenbank mit Verwaltungsdaten für die Erstanmeldung eines Benutzers bei der Installation.

- users

Diese Datenbank enthält den Benutzer **Install**, der eine Installierung durchführen kann.

- workspace.txt

Diese Datei enthält den Inhalt einer Arbeitsfläche von VisualWorks mit einigen nützlichen Botschaften und Programmen. Diese Arbeitsfläche wird beim Aktivieren der Entwicklungsumgebung mit aufgeblendet und kann auch nach Änderungen neu gespeichert werden.

2.1.2 Die Installation

Die Installation muß auf dem Server-Rechner von ObjectStore durchgeführt werden. Auf diesem Rechner müssen sich auch das Home-Verzeichnis mit den oben beschriebenen Dateien, sowie das neu anzulegende Datenbank-Verzeichnis befinden. Benutzt werden kann Elab dann von verschiedenen Client-Rechnern aus.

Nachfolgend werden die einzelnen Schritte der Installationsprozedur für Elab beschrieben:

1. Zunächst muß auf dem Server-Rechner ein Home-Verzeichnis angelegt werden. Der Name des Verzeichnisses ist beliebig.
2. Sämtliche Dateien des Elab-Systems sind in dieses Home-Verzeichnis zu kopieren.
3. Elab wird aufgerufen, indem die von ObjectStore modifizierte Object Engine des VisualWorks-Smalltalk-Systems gestartet wird. Dem entsprechenden Kommando ist der Name der Image-Datei (elab.im) zu übergeben.

Dieses Kommando ist auch für die Installation aufzurufen.

4. Nach dem Starten der Object Engine - dies kann einige Zeit dauern - erscheint auf dem Bildschirm das in Abb. 6, Seite 15 dargestellte Fenster zur Anmeldung. In dessen Texteingabefeld ist als Benutzername **Install** einzutragen. Betätigen des Buttons **OK** oder der Eingabetaste lösen den Anmeldevorgang aus.
5. Beim Installieren ist zunächst in einem Texteingabefenster (s. Abb. 1, Seite 12) der absolute Pfadname des Home-Verzeichnisses anzugeben.
6. Anschließend wird der Benutzer mittels eines Fensters zur Bestätigung (s. Abb. 2, Seite 13) aufgefordert, das Paßwort zum Installieren zu ändern.

Die Änderung des Paßworts ist unbedingt erforderlich, um die Installation durchführen zu können. Ohne Paßwortänderung wird der Installationsvorgang mit der Warnmeldung (s. Abb. 3, Seite 13) **Sorry, permission denied** abgebrochen.

7. Stimmt der Benutzer einer Paßwortänderung zu, erscheint ein Paßworteingabefenster (vgl. Abb. 7, Seite 16). Bei Erstinstallation lautet das Paßwort **No Password**. Die Paßworteingabe ist nur für ca. 15 s möglich. Ist sie dann noch nicht abgeschlossen oder wird ein falsches Paßwort eingegeben, wird der Installationsvorgang mit der Warnmeldung **Permission denied** abgebrochen.
8. Das neue Passwort muß eingegeben und anschließend noch einmal bestätigt werden. Auch hier wird der Installationsvorgang nach ca. 15 s mit der Warnmeldung **Permission denied** abgebrochen.

9. In einem Texteingabefenster muß der absolute Pfadname des gewünschten Datenbank-Verzeichnisses angegeben werden. Außerdem muß am Beginn dieses Namens unbedingt der Name des Server-Rechners stehen. D.h. z.B.

```
miserv2:/projects/elab/db/
```

Wird der Name des Rechners nicht angegeben, führt das zu Fehlern bei Datenbankzugriffen, die über ObjectStore-Fehlermeldungen mitgeteilt werden. Der Benutzer muß den Prozeß dann ggf. abbrechen.

Existiert das Verzeichnis bereits, erfolgt die Meldung und Abfrage **Directory exists. Overwrite databases?** Bestätigt der Benutzer, werden die vorhandenen Datenbanken einfach überschrieben. Bei Verneinen wird gefragt, ob der neue Pfadnamen akzeptiert wird (**Accept existing database?**). Entsprechend wird der Pfad tatsächlich auf das vorhandene neue Verzeichnis umgesetzt, oder der alte Pfad bleibt erhalten. Beim Umsetzen auf ein bereits existierendes Verzeichnis muß der installierende Benutzer allerdings sicher sein, daß dieses Verzeichnis auch Datenbanken mit konsistentem Inhalt enthält. Ohne Umsetzen wird das Programm beendet. Die Installation muss dann neu durchgeführt werden.

Falls das neue Verzeichnis noch nicht existiert und das Anlegen nicht möglich ist, erfolgt die Meldung **Database directory does not exist** und der Installationsvorgang wird abgebrochen.

10. Anschließend erfolgt die Mitteilung an den installierenden Benutzer, daß ein Benutzer namens **Super** erzeugt und beim System angemeldet wird. Dessen Paßwort ist ebenfalls **Super** und er gehört zu einer fiktiven Projektgruppe **Elab**. Der genaue Wortlaut der Meldung ist:

**Creating and logging in user Super
with password >Super<, belonging
to group Elab.**

Remember to change password.

11. Entsprechend der Ankündigung wird das Paßwort für den Benutzer **Super** verlangt und nach korrekter Eingabe erscheint das Hauptfenster des Systems Elab (s. Abb. 8, Seite 17) auf dem Bildschirm. Der Superuser **Super** ist nun beim System angemeldet. Falls die Paßworteingabe wegen Überschreitens der Zeitvorgabe oder Fehleingaben abgebrochen und der Superuser damit wieder abgemeldet worden sein sollte, muß die Installation ab Schritt 3. wiederholt werden. Zu beachten: Das Paßwort wurde inzwischen geändert.
12. Zunächst empfiehlt es sich, einen neuen Superuser für den tatsächlichen Systemadministrator mit dessen Namen zu erzeugen. Zur Erzeugung von Benutzern s. Kapitel 2.2.3 auf Seite 21.
13. Die Entwicklungsumgebung ist zu starten (vgl. Kapitel 2.2.3 auf Seite 22). Dabei müssen, wie dort beschrieben, die Pfadnamen von Source- und Changes-Datei geändert werden (elab.sou und elab.cha). Es sind absolute Namen anzugeben.
14. In der Arbeitsfläche gibt es im Originalzustand einen Methodenaufruf **Login open**. Falls dies in einem späteren Zustand nicht mehr der Fall sein sollte, muß dieser Aufruf wieder an beliebiger Stelle eingefügt werden. Dieser Aufruf ist durchzuführen.

15. Danach sind alle Fenster bis auf das Fenster zur Anmeldung, das durch den vorangegangenen Aufruf erzeugt wurde, zu schließen. Beim Hauptfenster von VisualWorks wird der Benutzer gefragt, ob er es tatsächlich schließen möchte. Dies ist zu bestätigen. Bei der Arbeitsfläche wird der Benutzer gefragt, ob diese gesichert werden soll. Dies ist zu verneinen.
16. Durch Betätigen des kleinen roten Buttons im Fenster zur Anmeldung, der nur aktiv ist, wenn die Entwicklungsumgebung gestartet wurde, kann das System im vorliegenden Zustand, also nur mit geöffnetem Fenster zur Anmeldung, als Image gespeichert werden.

In Texteingabefenstern wird noch einmal nach dem Pfad des Home-Verzeichnisses und nach dem Pfad der Image-Datei gefragt. Die angegebenen Namen sind zu kontrollieren und ggf. zu korrigieren.

Nach erfolgter Speicherung, was üblicherweise einige Zeit in Anspruch nimmt, verschwindet das Fenster zur Anmeldung.

17. Es wird empfohlen, das System noch einmal zu starten. Der Systemadministrator, sofern vom Benutzer **Super** erzeugt, sollte sich anmelden und den Benutzer Super löschen (s. Kapitel 2.2.3 auf Seite 22). Tut er das nicht, ist zumindest dringend anzuraten, das Paßwort von **Super** zu ändern.

2.2 Funktionalität des Systems

2.2.1 Dialogfenster

Bei der Implementierung von Elab wurden an vielen Stellen Dialogfenster verwendet, wie sie bereits in VisualWorks zur Verfügung gestellt werden. Beispiele für diese Fenster zeigen die folgenden Abbildungen. Die Fenster sehen im wesentlichen immer gleich aus. Sie unterscheiden sich nur in Beschriftung und Größe.

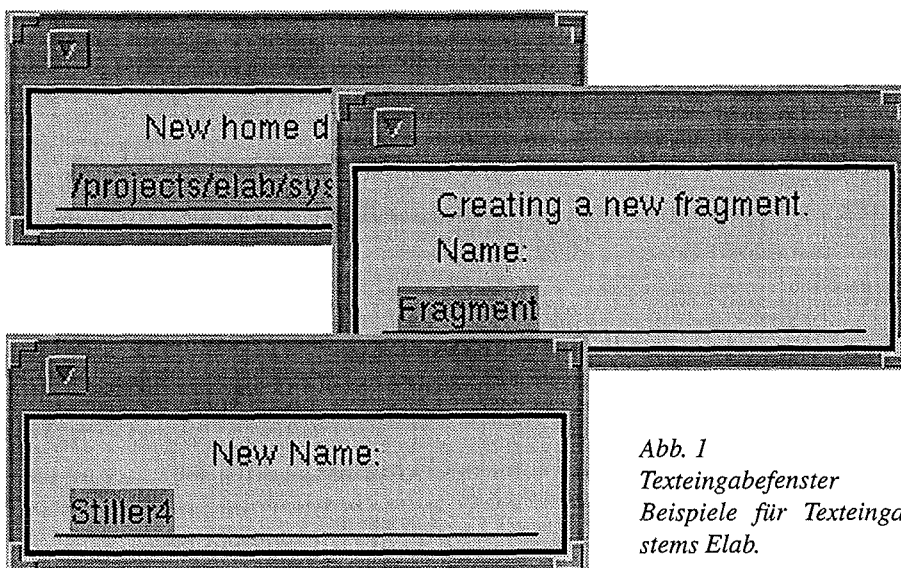


Abb. 1
Texteingabefenster
Beispiele für Texteingabefenster des Systems Elab.

1. Ein Texteingabefenster hat i. allg. eine kurze Überschrift, die dem Benutzer mitteilt, zu welchem Zweck er Text eingeben soll. Der eingegebene Text erscheint auf einer durchgezogenen Linie. Abb. 1 zeigt Beispiele für Texteingabefenster, die im System Elab vorkommen..
2. Abb. 2 zeigt Beispiele für Fenster zur Bestätigung des Systems Elab. Der Benutzer muß jeweils einen Sachverhalt bestätigen oder verneinen, der durch den angegebenen Text spezifiziert wird.

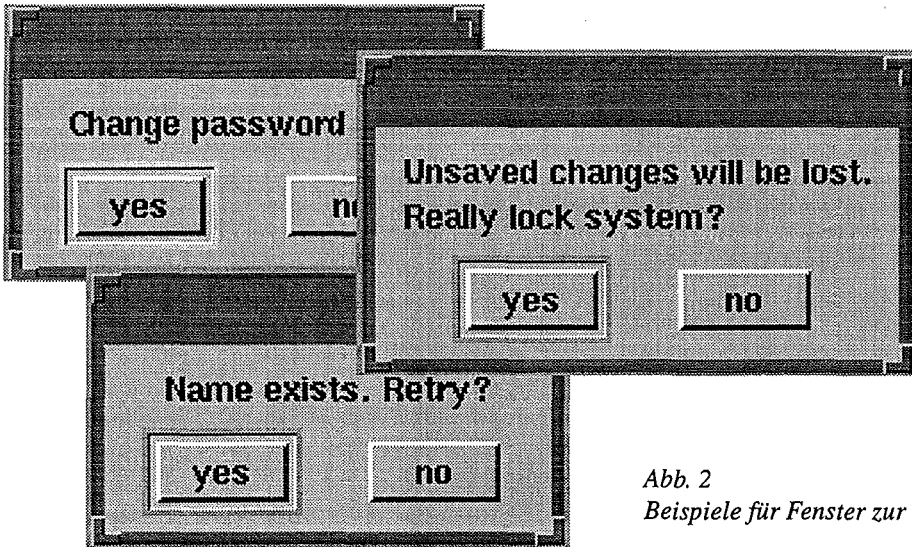


Abb. 2
Beispiele für Fenster zur Bestätigung

3. Eine Warnmeldung, (s. Abb. 3) erfolgt immer dann, wenn sich das Programm in einer Sackgasse befindet. Der Benutzer wird über den vorliegenden Fehlerfall und/oder das weitere Vorgehen informiert. Wenn er die Nachricht zur Kenntnis genommen hat, kann er durch Betätigen des Buttons **OK** weitermachen.

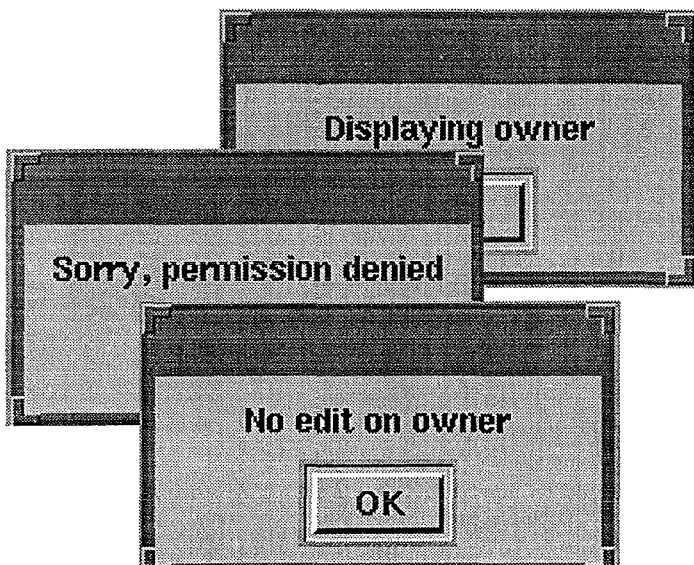


Abb. 3
Verschiedenen Warnmel-
dungen des Systems Elab

4. Eine Auswahlliste (Beispiele in Abb. 4) zeigt eine beliebige Anzahl von Textzeilen. Die Liste ist mit einem Rollbalken und zwei Buttons versehen (hier: **OK** und **Cancel**). Die Beschriftung der Buttons kann geändert sein. Durch Doppelklick auf einen Eintrag oder Einfachklick zur Selektion und anschließender Betätigung des linken Buttons wird der betreffende Eintrag ausgewählt und dem Programm übergeben. Das Fenster verschwindet vom Bildschirm. Betätigen des rechten Buttons bricht den Auswahlvorgang ab.

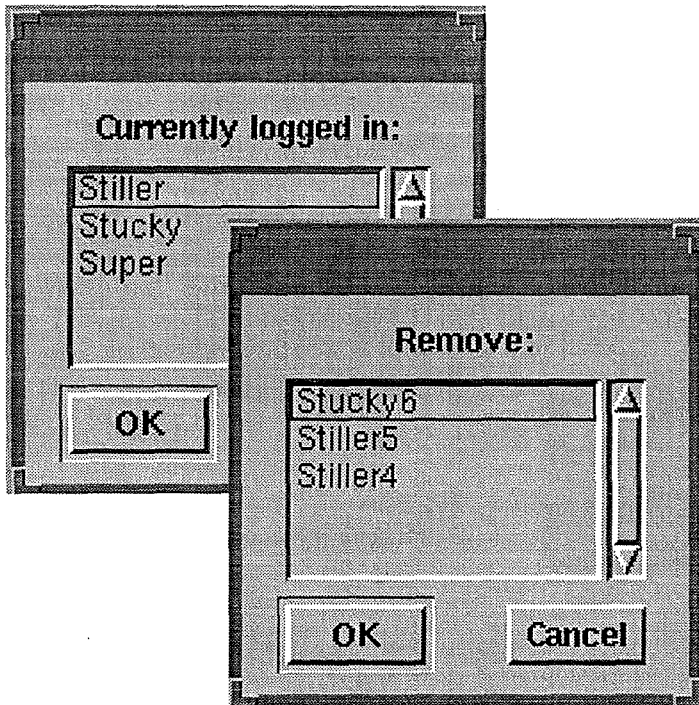


Abb. 4
Beispiele für Auswahllisten de
Systems Elab

5. Ein Fenster zur Auswahl aus einer festen Anzahl von Möglichkeiten (Abb. 5) zeigt entsprechend der gegebenen Alternativen eine bestimmte Anzahl von Buttons mit kontextabhängiger Beschriftung. Der Benutzer kann genau eine Alternative durch Betätigen des zugehörigen Buttons auswählen.

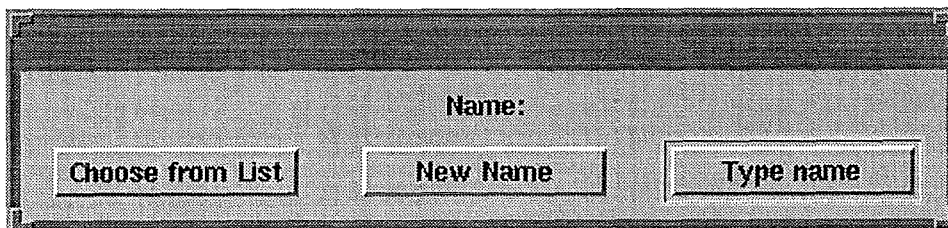


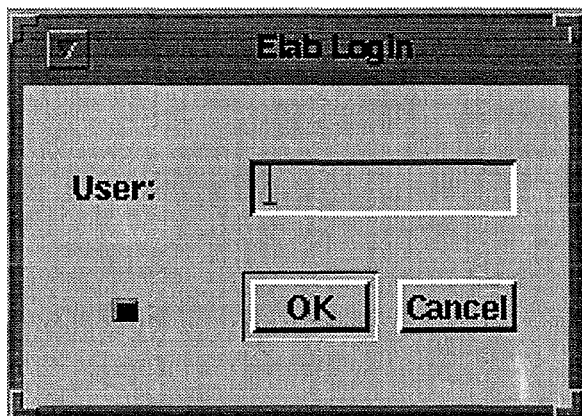
Abb. 5
Fenster zur Auswahl aus einer festen Auswahl von Möglichkeiten
Der Benutzer kann genau eine Alternative auswählen.

Alle anderen Fenster sind speziell für das System Elab eingeführte Fenster, die in den weiteren Unterkapiteln beschrieben werden.

2.2.2 Anmelden beim System

Elab wird aufgerufen, indem die von ObjectStore modifizierte Object Engine des Visual-Works-Smalltalk-Systems gestartet wird. Dem entsprechenden Kommando ist der Name der Image-Datei (elab.im) zu übergeben.

Nach dem Starten der Object Engine erscheint nach einer kurzen Wartezeit das in Abb. 6 dargestellte Fenster zur Anmeldung. Seine Überschrift lautet **Elab Login**. Es besteht aus einem Texteingabefeld, einem kleinen roten Button und zwei weiteren Buttons.



*Abb. 6
Fenster zur Anmeldung
Eingetragen wird der Name des Anmel-
denden. Der Vorgang kann mit **Cancel**
abgebrochen werden.*

Der Anmeldevorgang kann jederzeit durch Betätigen des Buttons **Cancel** abgebrochen werden. Der kleine rote Button ist nur aktiv, wenn ein Superuser die Smalltalk-Entwicklungsumgebung gestartet hat. Dieser Vorgang wird später beschrieben (Kapitel 2.2.3 auf Seite 22).

Im Texteingabefeld wird der Benutzername eingetragen. Die Anmeldung wird durch Betätigen der Eingabetaste oder des Buttons **OK** ausgeführt. Dabei werden die Buttons deaktiviert. Wenn kein Benutzer mit dem angegebenen Namen existiert, erfolgt eine Warnmeldung **User [Name] unknown**. Nach Bestätigung dieser Meldung werden die Buttons wieder aktiviert und somit besteht erneut die Chance, einen gültigen Namen einzugeben. Der Vorgang kann nun natürlich auch abgebrochen werden.

Nachdem ein gültiger Benutzername eingegeben wurde, erfolgt nach kurzer Wartezeit die Abfrage nach dem Paßwort dieses Benutzers. Einzige Ausnahme: falls gerade ein Superuser die Daten des betreffenden Benutzers editiert, sind diese gesperrt. Eine Anmeldung ist dann nicht möglich und es erfolgt die Warnmeldung **Permission denied**. Das Fenster zur Anmeldung bleibt weiterhin aktiv.

Die Eingabe des Paßworts geschieht im Paßworteingabefenster (dargestellt in Abb. 7). Im Textfeld dieses Fensters kann das Paßwort des sich anmeldenden Benutzers eingegeben werden. Es ist dabei nicht sichtbar. Im Feld erscheinen nur die Zeichen '*'. Abgeschlossen wird die Eingabe durch Betätigen der Eingabetaste oder des Buttons **OK**. Die Paßworteingabe kann auch mit **Cancel** abgebrochen werden. Dann wird der gesamte Anmeldevorgang abgebrochen

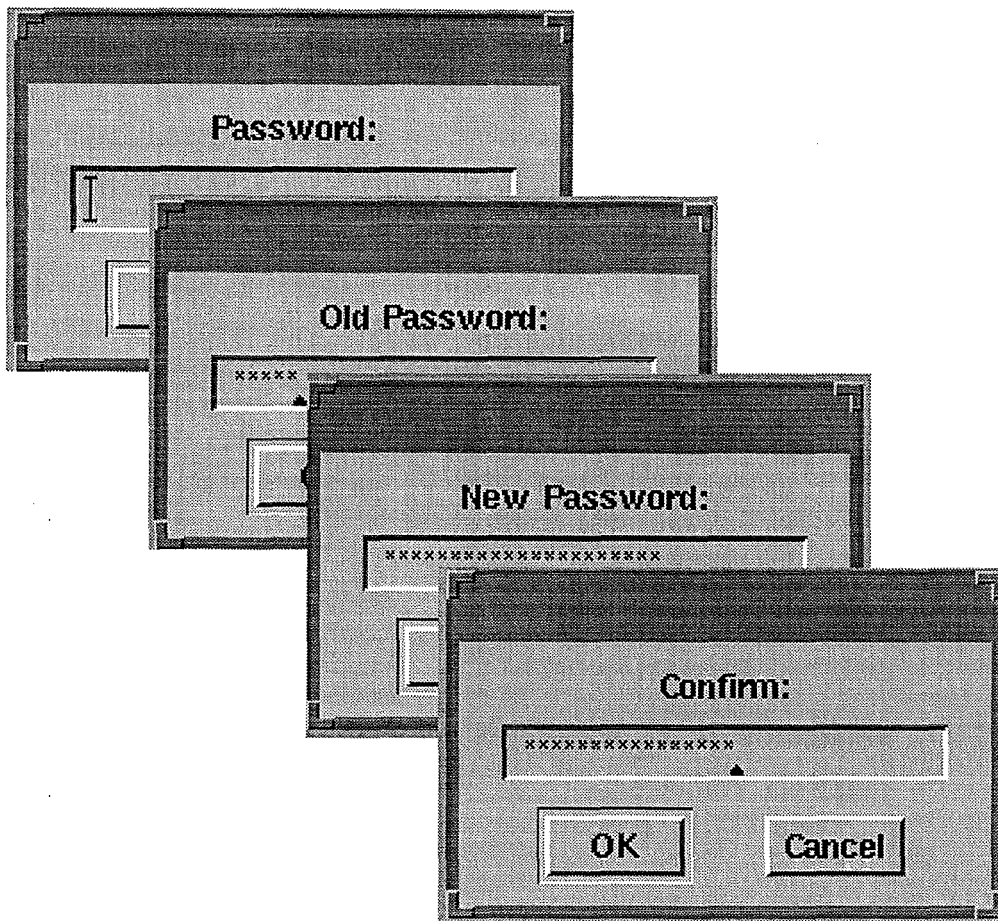


Abb. 7
Paßworteingabefenster
Dieses Fenster kommt mit den Beschriftungen *Password*
und z.B. beim Ändern des Paßworts mit *Old Password*,
New Password und *Confirm* vor.

Bei falscher Eingabe erfolgt die Warnmeldung **Wrong Input!** Für die Paßworteingabe sind nur drei Versuche möglich.

Das Paßworteingabefenster kommt auch noch mit anderen Beschriftungen vor. Der Eingabevorgang bei einem Paßworteingabefenster muß innerhalb von ca. 15 s abgeschlossen sein. Ansonsten verschwindet das Fenster und der Vorgang wird abgebrochen.

Nach der Eingabe des richtigen Paßworts ist der Benutzer beim System Elab angemeldet. Auf dem Bildschirm erscheint das im nächsten Unterkapitel beschriebene Hauptfenster. Ein Benutzer kann sich auch mehrfach anmelden.

Ein Paßwort ist jeweils nur dem jeweiligen Benutzer bekannt. Bei der Erzeugung eines Benutzers (s. Kapitel 2.2.3) vergibt der Superuser ein vorläufiges Paßwort, das vom Benutzer möglichst bald zu ändern ist. Danach hat auch ein Superuser keine Möglichkeit mehr, auf das Paßwort zuzugreifen.

2.2.3 Das Hauptfenster

Nach dem Anmelden eines Benutzers erscheint das Hauptfenster von Elab (Abb. 8). Als Überschrift ist der Benutzername sowie Datum und Uhrzeit der Anmeldung verzeichnet. Das Fenster ist einerseits der Einstieg in die einzelnen Systemteile von Elab (linkes Drittel), andererseits wird es zur Benutzerverwaltung genutzt (für den normalen Systembenutzer in der Mitte, für Aufgaben eines Superusers im rechten Drittel). Ferner kann eine Druckereinstellung vorgenommen und die Smalltalk-Entwicklungsumgebung gestartet werden. Die Funktionalität für den Superuser ist bei einem normalen Benutzer deaktiviert.

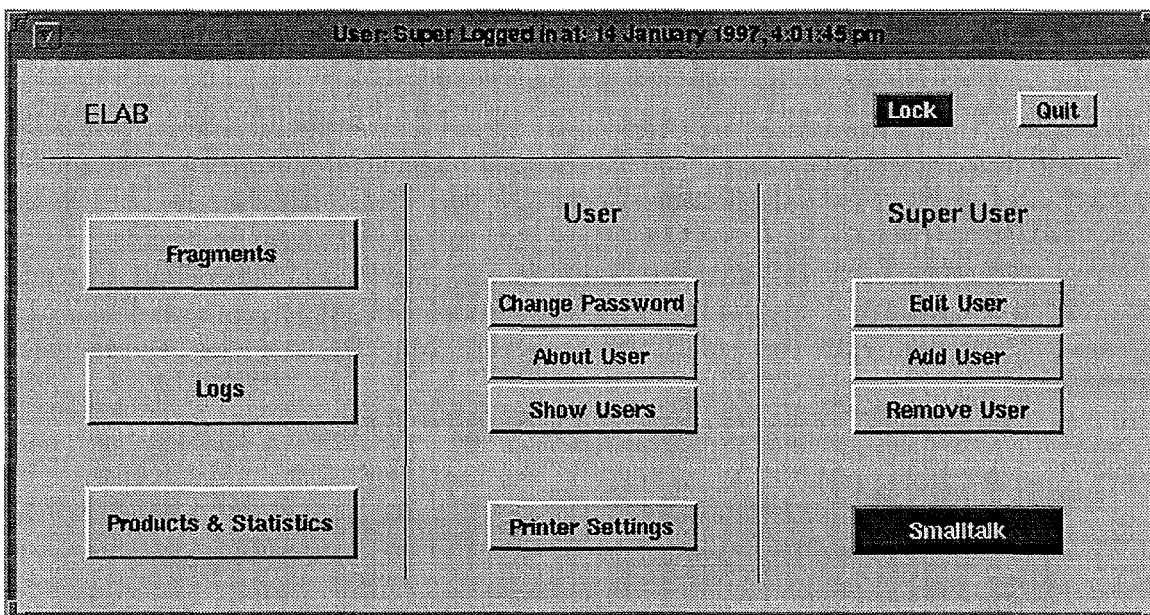


Abb. 8

Hauptfenster des Systems Elab

Einzelne Teilsysteme können aufgerufen werden. Ferner kann eine Benutzerverwaltung vom normalen Benutzer oder auch vom Superuser durchgeführt werden.

Die Buttons **Fragments**, **Logs** und **Products & Statistics** verzweigen zu den einzelnen Teilsystemen Fragmentenverwaltung, Protokollgenerator und -verwaltung sowie Produktverwaltung & Statistik des Elektronischen Laborbuchs (Kapitel 2.2.4 bis Kapitel 2.3).

Im Rahmen der Benutzerverwaltung kann der normale Systembenutzer folgende Funktionen ausführen:

- Ändern des Paßworts
- Anzeige von eigenen Benutzerdaten
- Auswahl eines anderen, gleichzeitig angemeldeten Benutzers und Anzeige von dessen Benutzerdaten

Zum Ändern des Paßworts ist der Button **Change Password** zu betätigen. Falls der Vorgang nicht abgebrochen wird oder es zu einer Fehleingabe kommt, erscheint nacheinander dreimal ein Paßworteingabefenster (s. Abb. 7, Seite 16) mit den Beschriftungen **Old Password**, **New Password** und **Confirm**. Entsprechend muß zuerst das alte Paßwort zur Legitimation des Benutzers angegeben werden. Dann kann das neue Paßwort eingegeben werden und anschließend ist dieses zur Bestätigung noch einmal einzugeben.

Jedes der Paßworteingabefenster bleibt nur für ca. 15 s auf dem Bildschirm. Ist bis dahin die Eingabe noch nicht abgeschlossen, wird der gesamte Vorgang abgebrochen. Wenn das alte Paßwort falsch eingegeben wird, erscheint die Warnmeldung **Wrong Input!** Anschließend wird der Vorgang abgebrochen. Wird bei der Bestätigung das neue Paßwort falsch eingegeben, erhält der Benutzer lediglich eine weitere Chance zur Neueingabe in einem zusätzlichen Paßworteingabefenster mit der Beschriftung **Wrong Input, Second Chance**. Ist auch diese Neueingabe falsch, erscheint noch einmal die Warnmeldung **Wrong Input!** und der gesamte Vorgang wird abgebrochen ohne daß das Paßwort geändert wird.

Mit dem Button **About User** kann der Benutzer die eigenen Benutzerdaten abrufen. Es erscheint das in Abb. 9 gezeigte Benutzerfenster mit der Überschrift **User**, wobei hier alle Felder und Buttons bis auf den Button **OK** deaktiviert sind. Ist bereits auf dem Bildschirm ein Benutzerfenster vorhanden, so wird dieses ggf. in den Vordergrund geholt und die Daten darin werden entsprechend den Daten des Benutzers aktualisiert. Außerdem werden die Aktivierungszustände der Textfelder und Buttons angepaßt. Mit **OK** wird das Fenster wieder geschlossen.

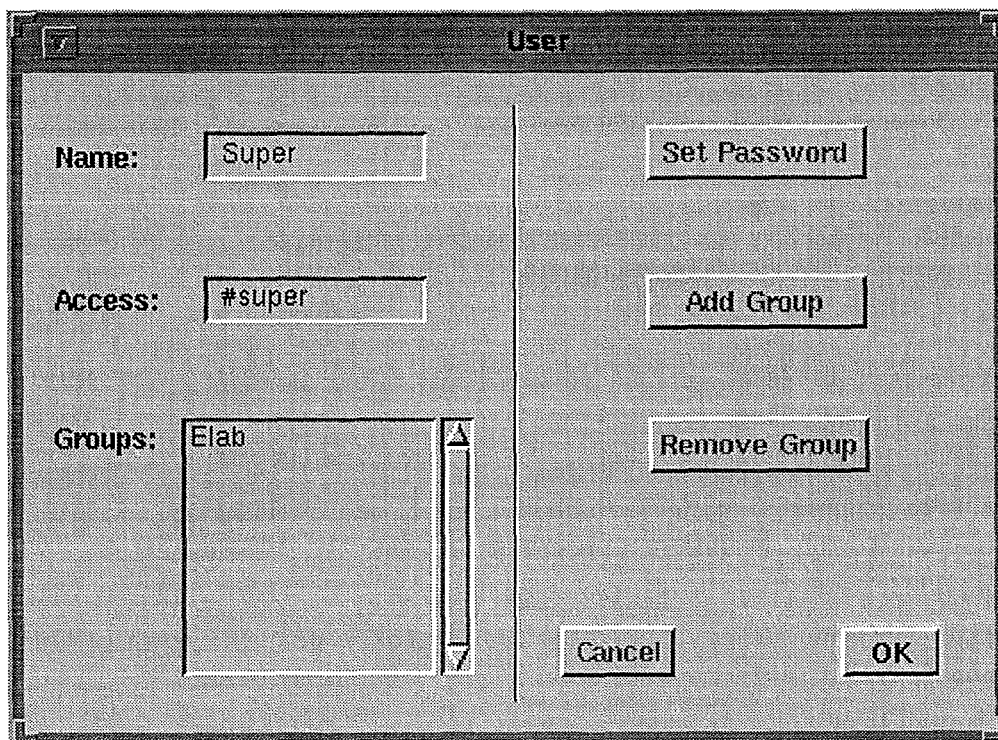


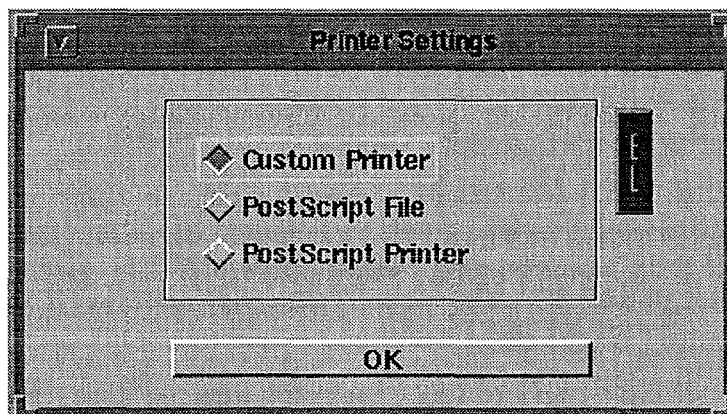
Abb. 9
Benutzerfenster zur Darstellung und Modifikation von Benutzerdaten im Rahmen der Benutzer-
verwaltung

Ein Elab-Benutzer hat einen Namen und ein Zugriffsrecht. Das Zugriffsrecht ist entweder **super** (#super im betreffenden Textfeld des Benutzerfensters) oder **fragment** (#fragment im Textfeld) entsprechend einem Superuser oder einem normalen Benutzer.

Ferner kann ein Benutzer einer beliebigen Anzahl von Projektgruppen zugeordnet werden. Diese sind im Benutzerfenster in einer Liste dargestellt. In Abb. 9 gehört der Superuser **Super** zu der Projektgruppe **Elab**.

Der Button **Show Users** im Hauptfenster zeigt eine Auswahlliste mit den Namen aller momentan angemeldeten Benutzer. Bei Auswahl eines Namens werden die Daten des betreffenden Benutzers dargestellt. Dazu wird entweder ein neues Benutzerfenster geöffnet oder auch, wie oben beschrieben, ein bereits vorhandenes Fenster benutzt. Auch hier sind alle Textfelder und Buttons bis auf **OK** deaktiviert. **OK** schließt das Fenster.

Außer den beschriebenen Funktionen der Benutzerverwaltung kann ein normaler Benutzer auch eine Druckereinstellung vornehmen. Dazu betätigt er den Button **Printer Settings** im Hauptfenster. Damit wird das in Abb. 10 gezeigte Fenster zur Druckereinstellung mit der Überschrift **Printer Settings** dargestellt.



*Abb. 10
Fenster zur Druckereinstellung
Ausgaben von Informationen aus Elab sind auf dem jeweiligen Standarddrucker, in eine PostScript-Datei oder auf einen PostScript-Drucker möglich.*

In diesem Fenster kann für alle Druckbefehle des Laborbuchs ausgewählt werden, ob die Ausgabe auf dem Standarddrucker, in eine PostScript-Datei oder auf einen speziellen PostScript-Drucker (nicht allgemein anwendbar) erfolgen soll.

Eine PostScript-Datei, in die geschrieben wird, befindet sich standardmäßig in dem Verzeichnis, in dem das System gestartet wurde. Dort wird auch eine Ausgabe auf dem Standarddrucker in einer Datei **temp.prt** mitprotokolliert. Die Ausgabe auf dem Standarddrucker ist die Standardeinstellung beim ersten Öffnen dieses Fensters.

Nach Betätigen des Buttons **OK** wird der Anwender in einem Fenster zur Bestätigung gefragt, ob das Fenster zur Druckereinstellung nach der Übernahme der aktuellen Einstellung auf dem Bildschirm bleiben soll oder nicht. Die Beschriftung des Fensters lautet **Closing Window?** Falls **Quit** im Menü des Fensters selektiert wird, wird die aktuelle Einstellung auch übernommen, das Fenster wird aber auf jeden Fall geschlossen.

Der rote Balken rechts neben der Auswahlliste dient dazu, das Hauptfenster wiederaufzufinden. Es wird bei Betätigung des Balkens in den Vordergrund geholt oder auch geöffnet, falls es als Icon vorlag. Auch andere Fenster des Systems Elab enthalten einen solchen Balken, der unterschiedlich groß sein kann.

Die Buttons auf der rechten Seite des Hauptfensters sind nur dann aktiv, wenn der Benutzer Superuser ist. Ein Superuser kann folgende Funktionen der Benutzerverwaltung nutzen:

- Editieren der Daten von Benutzern
- Hinzufügen eines neuen Benutzers
- Löschen eines Benutzers

Bei Betätigen des Buttons **Edit User** erscheint eine Auswahlliste mit den Namen aller dem System bekannten Benutzer. Bei Auswahl eines Namens werden die Daten des betreffenden Benutzers dargestellt. Dazu wird auch hier wieder entweder ein neues Benutzerfenster geöffnet oder ein bereits vorhandenes Fenster benutzt. In diesem Fall sind nur das Feld **Name** und der Button **Set Password** deaktiviert. Name und Paßwort eines existierenden Benutzers können auch vom Superuser niemals geändert werden.

Falls die Daten des ausgewählten Benutzers gesperrt sind, d.h. dieser Benutzer ist entweder gerade beim System angemeldet oder er wird von diesem oder einem anderen Superuser editiert, erscheint zuerst die Warnmeldung **Permission denied** und anschließend die Meldung **Displaying owner**. Anstelle der angeforderten Benutzerdaten werden dann die Daten des betreffenden Superusers dargestellt.

Falls der Benutzer den eigenen Namen auswählt erscheinen nacheinander die Meldungen **No edit on owner** und **Displaying owner**, was bedeutet, daß die eigenen Benutzerdaten mit gesperrten Eingabefeldern und deaktivierten Buttons angezeigt werden.

Im Textfeld **Access** kann das Zugriffsrecht des Benutzers (**fragment** oder **super**) geändert werden. Das führende Zeichen '#' muß nicht mit eingegeben werden. Das neue Zugriffsrecht wird nach Betätigen der Eingabetaste übernommen. Eine ungültige Eingabe wird mit

User>>access: [Eingabe] not allowed as argument.

Setting access to #fragment.

kommentiert. Das Zugriffsrecht wird automatisch auf **fragment** gesetzt (Abb. 11).

Mit den Buttons **Add Group** und **Remove Group** im Benutzerfenster (Abb. 9, Seite 18) können Projektgruppen hinzugefügt oder gelöscht werden.

Jeder neue Benutzer gehört zu mindestens einer Projektgruppe. Neu erzeugte Benutzer gehören der fiktiven Projektgruppe **Undefined Project** an. Falls bei der Erzeugung (s. unten) keine gültige Projektgruppe angegeben wird, kann es vorkommen, daß ein Benutzer auch beim späteren Editieren lediglich dieser fiktiven Gruppe angehört.

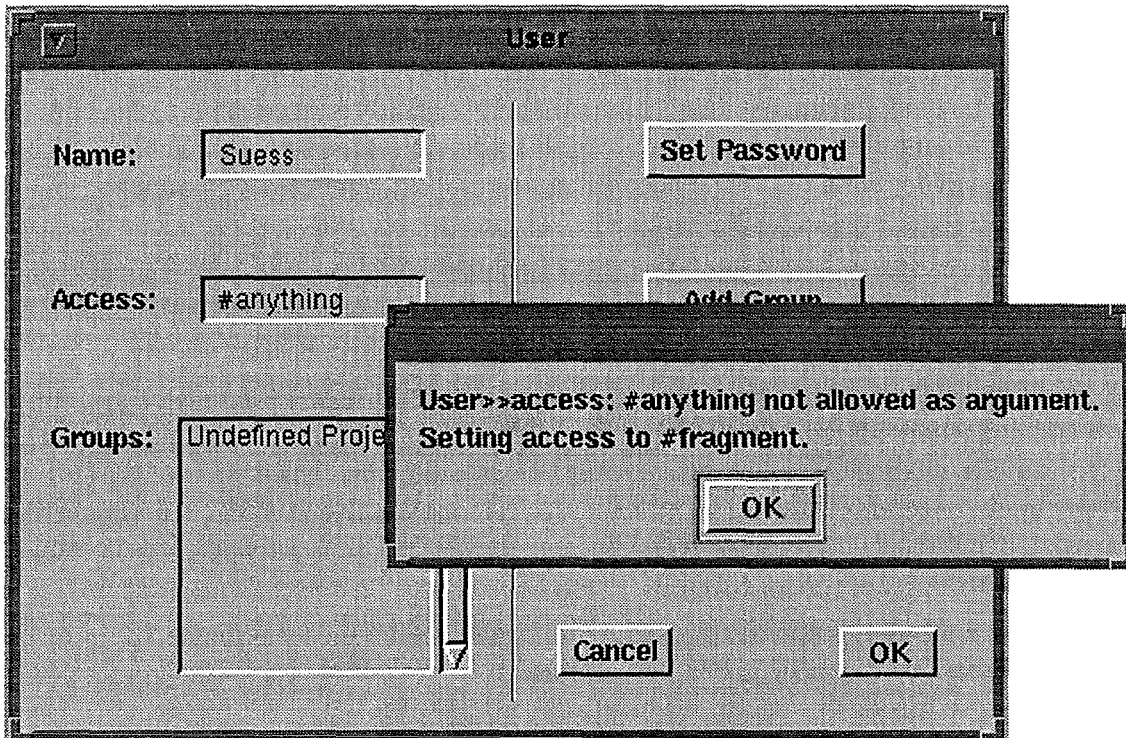


Abb. 11

Falsche Eingabe beim Zugriffsrecht:

Der Benutzer darf nur #super oder #fragment angeben. Falsche Eingaben werden abgefangen, das Zugriffsrecht wird automatisch auf #fragment gesetzt und der Benutzer wird entsprechend informiert.

Beim Hinzufügen einer Projektgruppe erscheint ein Texteingabefenster, das bereits den Text **Undefined Project** enthält. Im Normalfall wird hier ein gültiger Projektgruppenname angegeben, der einfach hinzugefügt wird und sofort in der Liste links im Benutzerfenster erscheint. Wenn hier lediglich **Undefined Project** existierte, wird diese Gruppe automatisch gelöscht. Der im Texteingabefenster vorhandene Vorschlag **Undefined Project** sollte in jedem Fall geändert werden, da der Benutzer sonst wieder zu dieser Gruppe gehört und sie dann explizit gelöscht werden muß, wenn auch andere Gruppen vorhanden sind. Wird ein bereits existierender Gruppenname angegeben, geschieht nichts. Gruppennamen werden also nicht mehrfach vergeben.

Wenn eine Projektgruppe gelöscht werden soll, muß in der Projektgruppenliste ein Name selektiert sein. Ansonsten erscheint beim Betätigen von Remove Group die Warnmeldung **No group selected**. Bei selektiertem Namen muß der Benutzer das Löschen in einem Fenster mit der Beschriftung **Really delete?** bestätigen. Der Gruppenname verschwindet aus der Liste im Benutzerfenster. Die letzte Projektgruppe kann nicht gelöscht werden. Beim Versuch, dies zu tun, erscheint die Warnmeldung **Last group cannot be removed**.

Um einen neuen Benutzer zu erzeugen muß der Superuser den Button **Add User** im Hauptfenster betätigen. Es erscheint ein Benutzerfenster bzw. ein bereits vorhandenes wird in den Vordergrund geholt und mit den Standard-Daten für einen neuen Benutzer belegt. Diese Stan-

dard-Daten sind **ElabUser** für den Namen, **fragment** für das Zugriffsrecht und die Projektgruppe **Undefined Project**. Das Paßwort ist unbelegt. Alle Felder und Buttons sind aktiviert.

Im Textfeld **Name** wird der Name des neuen Benutzers eingetragen, im Feld **Access** das Zugriffsrecht (**fragment** oder **super**). Das führende Zeichen '#' muß nicht mit eingegeben werden. Es ist zu empfehlen, daß nicht zu viele Superuser im System existieren, da ein Superuser sehr umfassende Rechte besitzt. Im Idealfall sollte ein Superuser ausreichen. Die neuen Werte der Textfelder werden jeweils nach Betätigen der Eingabetaste übernommen.

Mit Hilfe des Buttons **Set Password** kann der Superuser das Paßwort des neuen Benutzers vorbelegen. Es ist dringend zu empfehlen, daß sich der neuen Benutzer sofort nach Erzeugen des neuen Benutzerobjekts durch den Superuser beim System anmeldet und sein Paßwort ändert.

Das Paßwort wird in einem Paßworteingabefenster (s. Abb. 7, Seite 16) eingegeben. Wird bei der Bestätigung ein falscher Text eingegeben, erhält der Superuser lediglich eine weitere Chance zur Neueingabe in einem zusätzlichen Paßworteingabefenster mit der Beschriftung **Wrong Input, Second Chance**. Ist auch diese Neueingabe falsch, wird der Vorgang kommentarlos abgebrochen. Das Paßwort bleibt ungesetzt. Die Paßworteingabefenster sind nur für ca. 15 s aktiv.

Es können neue Projektgruppen hinzugefügt und auch wieder gelöscht werden (Buttons **Add Group** und **Remove Group** wie oben beschrieben).

Das Ändern von Benutzerdaten oder die Erzeugung eines neuen Benutzers durch den Superuser können jeweils durch Betätigen des Buttons **Cancel** im Benutzerfenster (Abb. 9, Seite 18) abgebrochen werden. Mit dem Button **OK** werden hier die neuen Daten bzw. das neue Benutzerobjekt vom System übernommen.

Wenn bei geöffnetem Benutzerfenster, aber ohne daß die darin enthaltenen Daten mit **OK** gesichert wären, erneut irgendeine Funktion der Benutzerverwaltung, die das Benutzerfenster benötigt, aufgerufen wird, erscheint ein Fenster zur Bestätigung mit der Beschriftung **Save to database before changing?** Dieses Fenster hat auf dem Button zur Bestätigung die Beschriftung **Save**. Bei **Cancel** (Button zur Verneinung) werden die Änderungen nicht übernommen.

Quit im Menü des Benutzerfensters schließt das Fenster kommentarlos ohne etwaige Änderungen zu sichern.

Ein Benutzer kann mit dem Button **Remove User** im Hauptfenster vom Superuser gelöscht werden. Es erscheint zunächst eine Auswahlliste mit den Namen aller dem System bekannten Benutzer. Nach der Auswahl eines Namens wird der Superuser gefragt, ob er das betreffende Benutzerobjekt tatsächlich löschen möchte (**Really remove [Name]**). Falls das Objekt gesperrt ist, weil es entweder editiert wird oder der Benutzer beim System angemeldet ist, erscheint die Warnmeldung **Permission denied** und der Vorgang wird abgebrochen. Ansonsten wird das betreffende Benutzerobjekt aus dem System entfernt.

Der rote Button **Smalltalk** kann ebenfalls nur von einem Superuser betätigt werden. Er startet die Smalltalk-Entwicklungsumgebung VisualWorks (Version 2.0) zur Weiterentwicklung des Systems Elab.

Zunächst muß der Superuser noch bestätigen, daß er tatsächlich Smalltalk starten möchte. Die Abfrage lautet: **Are you really sure you want to start VisualWorks?** Nach der Bestätigung

erfolgt die Frage, ob die Pfadnamen von Source- und Changes-Datei gesetzt werden sollen (**Edit location of Source File and Changes File**) Die Source-Datei beinhaltet die Quellen der Smalltalk-Umgebung und der von ihr zur Verfügung gestellten Software (**elab.sou** im Home-Verzeichnis). Die Changes-Datei enthält alle zusätzlichen Änderungen und u.a. auch die Quellen der Elab-Software (**elab.cha** im Home-Verzeichnis). Ein Setzen der Pfadnamen ist jeweils beim Installieren des Systems Elab nötig (Kapitel 2.1.2).

Nacheinander werden folgende Fenster aufgeblendet:

- Das in Abb. 12 dargestellte Fenster zum Setzen von Source- und Changes-Datei mit der Überschrift **Settings**, falls die entsprechende Frage mit **Yes** beantwortet wurde.

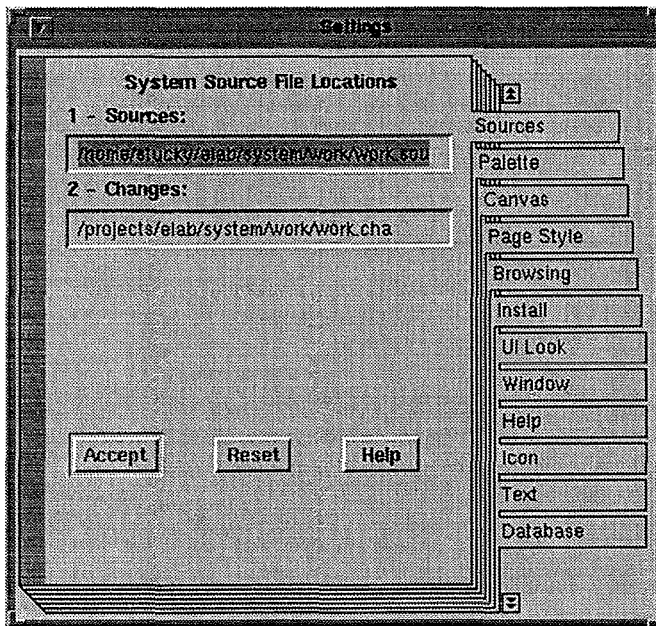


Abb. 12
Fenster zum Setzen der Pfade von Source- und Changes-Datei
Es müssen absolute Pfadnamen angegeben werden.

- Das Hauptfenster von VisualWorks mit der Überschrift **VisualWorks** (Abb. 13).

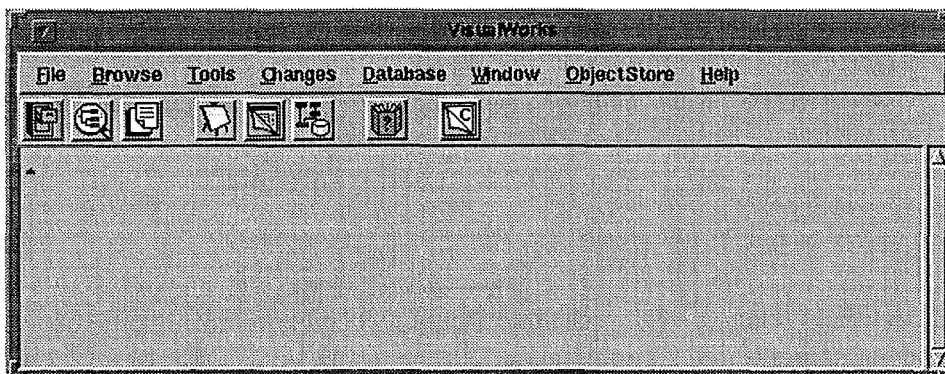
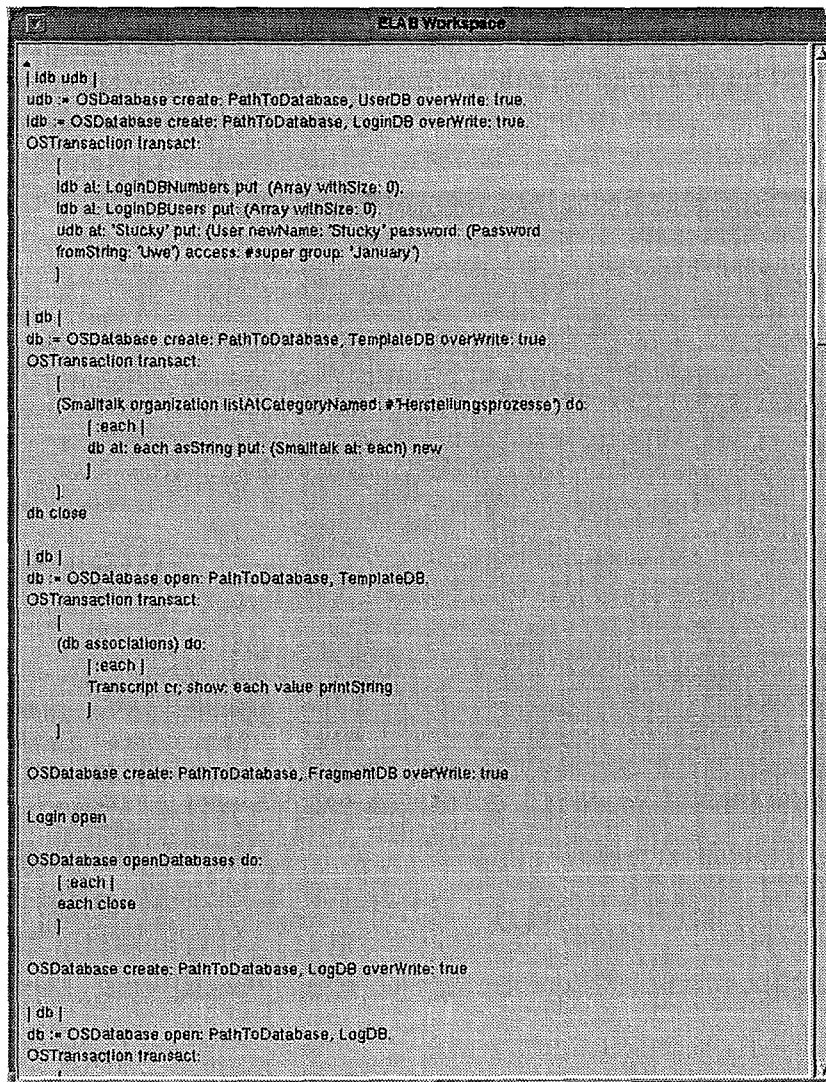


Abb. 13
Hauptfenster von VisualWorks

- Eine spezielle Elab-Arbeitsfläche mit der Überschrift **ELAB Workspace** (Abb. 14), die Methodenaufrufe und Programme enthält, die beim Entwickeln benutzt werden können. Diese Arbeitsfläche unterscheidet sich von einer gewöhnlichen VisualWorks-Arbeitsfläche nur durch ihr Speicherverhalten.



```
ELAB Workspace
| ldb udb |
udb := OSDatabase create: PathToDatabase, UserDB overwrite: true.
ldb := OSDatabase create: PathToDatabase, LoginDB overwrite: true.
OSTransaction transact:
|
  ldb at: LoginDBNumbers put: (Array withSize: 0).
  ldb at: LoginDBUsers put: (Array withSize: 0).
  udb at: 'Stucky' put: (User newName: 'Stucky' password: (Password
fromString: 'Uwe') access: #super group: 'January').
|

| db |
db := OSDatabase create: PathToDatabase, TemplateDB overwrite: true.
OSTransaction transact:
|
  (Smalltalk organization listAtCategoryNamed: #'Herstellungsprozesse') do:
  [each]
  db at: each asString put: (Smalltalk at: each) new
|
)
db close

| db |
db := OSDatabase open: PathToDatabase, TemplateDB.
OSTransaction transact:
|
  (db associations) do:
  [each]
  Transcript cr; show: each value printString
|
)

OSDatabase create: PathToDatabase, FragmentDB overwrite: true

Login open

OSDatabase openDatabases do:
| each |
  each close
|

OSDatabase create: PathToDatabase, LogDB overwrite: true

| db |
db := OSDatabase open: PathToDatabase, LogDB.
OSTransaction transact:
```

Abb. 14
Arbeitsfläche der Smalltalk-Entwicklungsumgebung mit Methodenaufrufen und
Programmen, die bei der Systementwicklung hilfreich sind

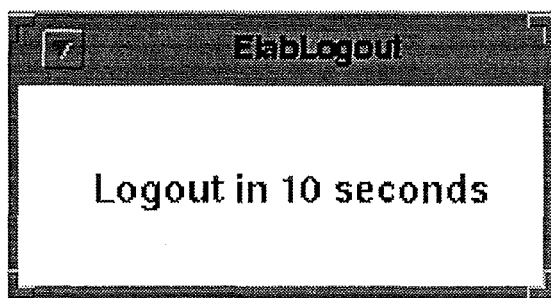
Zum Setzen der Pfadnamen von Source- und Changes-Datei sind jeweils die absoluten Pfadnamen in den entsprechenden Texteingabefeldern des Fensters aus Abb. 12 einzutragen. Diese Namen werden mit **Accept** übernommen. Das Fenster kann dann mit **Quit** im Menü des Fensters geschlossen werden.

Wenn das System anschließend wieder im Normalzustand, d.h. nur mit geöffnetem Fenster zur Anmeldung gespeichert werden soll, muß in der Arbeitsfläche die Nachricht **Login open** gesendet werden. Dazu wird die entsprechende Textzeile selektiert und im Menü der Arbeitsfläche der Eintrag **do it** gewählt. Damit erscheint das Fenster zur Anmeldung. Alle übrigen Fenster sind zu schließen. Beim Hauptfenster von VisualWorks muß das Schließen in einem Fenster mit der Beschriftung **Are you sure you want to close this Launcher?** bestätigt werden. Bei der Arbeitsfläche wird der Superuser in einem Fenster mit **Save this workspace?** gefragt, ob der Inhalt der Arbeitsfläche gespeichert werden soll. Das ist immer dann zu empfehlen, wenn sinnvolle Programme hinzugefügt wurden. Gespeichert wird dann in der Datei `workspace.txt` im Home-Verzeichnis. Wenn der Inhalt geändert wurde erscheint noch die Abfrage

The text showing has been altered.

Do you wish to discard those changes?

Diese Frage ist mit **Yes** zu beantworten, damit das Fenster tatsächlich geschlossen wird.



*Abb. 15
Meldung beim Beenden des Systems Elab
Sie erscheint für etwa 10 s, bevor das Programm
endgültig verlassen wird.*

Das Speichern des Systems als neue Image-Datei **elab.im** im Home-Verzeichnis kann dann durch Betätigen des kleinen roten Buttons im Fenster zur Anmeldung bewerkstelligt werden. Dieser Button ist nur aktiv, wenn die Smalltalk-Entwicklungsumgebung gestartet wurde. Das Speichern nimmt eine gewisse Zeit in Anspruch. Anschließend verschwindet das Fenster.

Der Button **Lock** im Hauptfenster (Abb. 8, Seite 17) sperrt das gesamte System ohne den Benutzer abzumelden. Vor dem Sperren erscheint ein Fenster zur Bestätigung mit folgendem Text:

Unsaved changes will be lost.

Really lock system?

Wenn nicht bestätigt wird, geschieht nichts, ansonsten werden alle geöffneten Fenster außer dem Hauptfensters geschlossen ohne daß irgendwelche modifizierten, nicht gesicherten

Daten gespeichert würden, alle Buttons bis auf **Lock** werden deaktiviert und **Lock** ändert seine Beschriftung zu **Unlock**.

Um das System wieder freizugeben, ist eben dieser Button **Unlock** zu betätigen. Es erscheint ein Paßworteingabefenster (s. Abb. 7, Seite 16). Der Benutzer hat drei Versuche, sein Paßwort einzugeben. Bei Fehleingabe wird er mit der Meldung **Wrong Input!** gewarnt. Das Paßworteingabefenster bleibt jeweils nur für ca. 15 s aktiv. Danach wird der Vorgang abgebrochen. Mißlingt auch der dritte Versuch zur Paßworteingabe, wird das System verlassen.

Nach Freigabe des Systems befindet es sich im selben Zustand wie nach einer Anmeldung.

Elab wird durch Betätigen des Buttons **Quit** im Hauptfenster verlassen. Alternativ kann auch **Quit** im Menü des Fensters gewählt werden. Alle Fenster werden geschlossen, wobei ungesicherte Änderungen verlorengehen. Für etwa 10 s erscheint das in Abb. 15 dargestellte Fenster. Danach ist das Programm beendet.

2.2.4 Die Fragmentenverwaltung

Wird im Hauptfenster (Abb. 8, Seite 17) der Button **Fragments** selektiert, erhält der Anwender das in Abb. 16 dargestellte Fenster, die Fragmentenverwaltung, mit der Überschrift **Fragment Manager**.

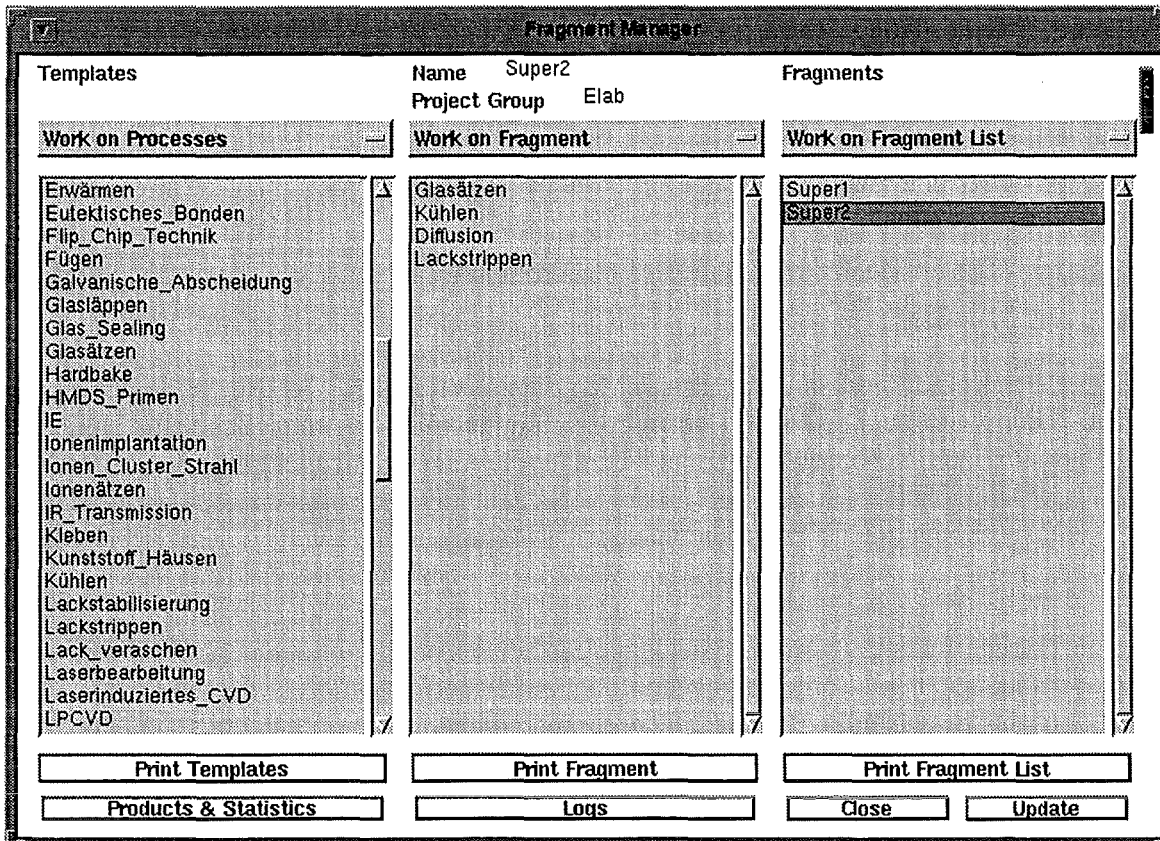


Abb. 16

Fenster zur Fragmentenverwaltung:

Links befindet sich die Liste aller implementierten Prozessschritte; auf der rechten Seite ist die Liste aller erzeugten Fragmente dargestellt. In der Mitte kann ein neues Fragment unter Benutzung von Prozessen aus der Prozessliste sowie aufbauend auf einem bereits existierenden Fragment zusammengestellt werden.

Der Zweck der Fragmentenverwaltung ist es, den Anwender bei der Zusammenstellung von neuen Fragmenten (d.h. Prozeßsequenzen oder Teilsequenzen) zu unterstützen, indem ihm einzelne Prozeßschritte, die im System Elab als Klassen implementiert sind und als Templates bezeichnet werden, zur Verfügung gestellt werden (links: **Templates**), sowie bereits erzeugte Fragmente (rechts: **Fragments**), die als Ausgangspunkt neuer Fragmente dienen können. Die Zusammenstellung des neuen Fragments als Folge von Prozessen erfolgt in der mittleren Liste.

Die Implementierung von Fragmenten ist in Kapitel 3.2 beschrieben.

Die Bearbeitung der drei Listen erfolgt über die Menüs **Work on Processes**, **Work on Fragment** und **Work on Fragment List**, die sich jeweils über den Listen befinden und in diesem Kapitel detailliert beschrieben werden.

Ferner können zu jeder Liste Informationen ausgedruckt werden, die auf Seite 38 näher erläutert werden.

Es ist möglich durch Selektion der Buttons **Products & Statistics** bzw. **Logs** die beiden anderen Teilsysteme von Elab zu aktivieren, ohne ins Hauptfenster wechseln zu müssen.

Der Button **Close** schließt - nach Bestätigung durch den Anwender (**Really close Fragment Manager?**) - die Fragmentenverwaltung. Dabei werden alle Prozeßeditoren, die von hier aus geöffnet wurden, geschlossen. Die Fenster der beiden anderen Teilsysteme Produktverwaltung & Statistik und Protokolle (Chargensplitting-Workfloweditoren) bleiben offen. Ein Speichern von ungesicherten Änderungen bei den Fragmenten und den darin enthaltenen Prozessen erfolgt nicht.

Mit **Quit** im Fenster-Menü wird das Fenster geschlossen ohne daß der Benutzer bestätigen muß.

Mit dem Button **Update** kann die Liste der Fragmente aktualisiert werden, wenn der Benutzer weiß oder vermutet, daß seit Aufruf der Fragmentenverwaltung oder seit der letzten Aktualisierung durch andere Benutzer neu erzeugte Fragmente hinzugekommen sind, die sich zwar in der Datenbank befinden, aber in der vorliegenden Fragmentliste noch nicht enthalten sind. Genauso können auch Fragmente gelöscht worden sein, die in der Liste noch enthalten sind.

Der rote Balken rechts dient dazu, das Hauptfenster wiederaufzufinden. Es wird bei Betätigung des Balkens in den Vordergrund geholt oder auch geöffnet, falls es als Icon vorlag. Auch andere Fenster des Systems Elab enthalten einen solchen Balken, der unterschiedlich groß sein kann.

Es folgen die Beschreibungen der jeweils zu einer der drei Listen der Fragmentenverwaltung gehörigen Menüs. Einträge in diesen Menüs sind entweder aktiviert (schwarz oder auch farbig) oder deaktiviert (grau).

In der mittleren und in der rechten Liste wird das Menü auch aufgeblendet, wenn innerhalb der Liste die mittlere Maustaste betätigt wird.

Bei Betätigung der mittleren Maustaste in der linken Liste wird der Hierarchiebaum der Prozeßschrittclassen aufgeblendet.

Menü Work On Processes

Dieses Menü, die Prozeßbearbeitung enthält zwei Einträge:

- **Add Process**

Ein in der Prozeßliste ausgewählter Prozeßschritt kann in die Liste der Fragmentbearbeitung (Mitte) eingefügt werden.

- **Edit Template**

Hier können die Attributwerte der Klasse eines ausgewählten Prozeßschritts, eines Templates, vorbelegt werden. Jedes neu erzeugte Objekt wird dann mit diesen Werten initialisiert.

Diese Funktion kann nur von einem Superuser ausgeführt werden und ist für normale Benutzer deaktiviert.

Solange in der Prozeßliste kein Prozeßschritt selektiert wurde, ist auch bei einem Superuser nur der erste Eintrag aktiviert. Wegen der fehlenden Selektion eines Prozeßschritts hat die Auswahl dieses Eintrags aber nur den Effekt, daß eine Warnmeldung **No template selected** aufgeblendet wird. Wenn in der rechten Liste der Fragmentenverwaltung kein Fragment selektiert wurde, ist nicht bekannt, zu welchem Fragment ein evtl. selektierter Prozeßschritt hinzugefügt werden soll. Dann erfolgt in analoger Weise eine Warnmeldung **No fragment selected**.

Bei selektiertem Prozeßschritt und selektiertem Fragment wird bei Auswahl des Menüeintrags **Add Process** ein Auswahlfenster mit den Buttons **Choose from List**, **New Name** und **Type name** aufgeblendet.

- **Choose from List**

Es erscheint eine Auswahlliste mit den Namen aller bereits in der Prozeßliste des selektierten Fragments vorhandenen Prozesse. Daraus kann ein Name für den neu hinzuzufügenden Prozeß ausgewählt werden.

Der Auswahlvorgang kann auch abgebrochen werden. Dann erscheint die Warnmeldung

No name selected.

Process cannot be added.

- **New Name**

Es erscheint ein Texteingabefeld, in dem ein beliebiger Name für den neu hinzuzufügenden Prozeß angegeben werden kann. Im Feld steht **ELABProcess** als Standardname. Der Vorgang kann abgebrochen werden.

- **Type name**

Hier wird der Klassenname, d.h. der Name des selektierten Templates wie er in der Prozeßliste angezeigt wird, verwendet.

Der neue Prozeß für das selektierte Fragment erscheint sofort unter dem ausgewählten Namen in der Prozeßliste des Fragments in der Mitte der Fragmentenverwaltung und zwar direkt unterhalb des in dieser Liste selektierten Prozesses. Falls in der Prozeßliste des Fragments kein Prozeß selektiert ist, wird der neue Prozeß am Anfang der Liste eingefügt.

Bei einem Superuser ist **Edit Template** aktiviert, sobald ein Template-Eintrag selektiert ist. Bei Auswahl von **Edit Template** erscheint die Warnmeldung **Permission denied**, falls gerade ein anderer Superuser das betreffende Template editiert. Wenn das Template-Objekt nicht gesperrt ist, erscheint ein Prozeßeditor auf dem Bildschirm. Prozeßeditoren werden bei den Erläuterungen zur Prozeßliste eines Fragments (Seite 33) ausführlicher beschrieben. Prozeß-

editoren für Templates unterscheiden sich von gewöhnlichen Editoren außer durch die Hintergrundfarbe dadurch, daß der Name nicht geändert werden kann und durch die immer gleichlautende Überschrift **Init Template (Name cannot be changed)**. Abb. 17 zeigt den Editor für das Template des Prozeßtyps **Photolithographie**. Änderungen, die hier vorgenommen werden, werden für alle nachfolgend erzeugten Prozesse des gleichen Typs als Standardwerte eingesetzt.

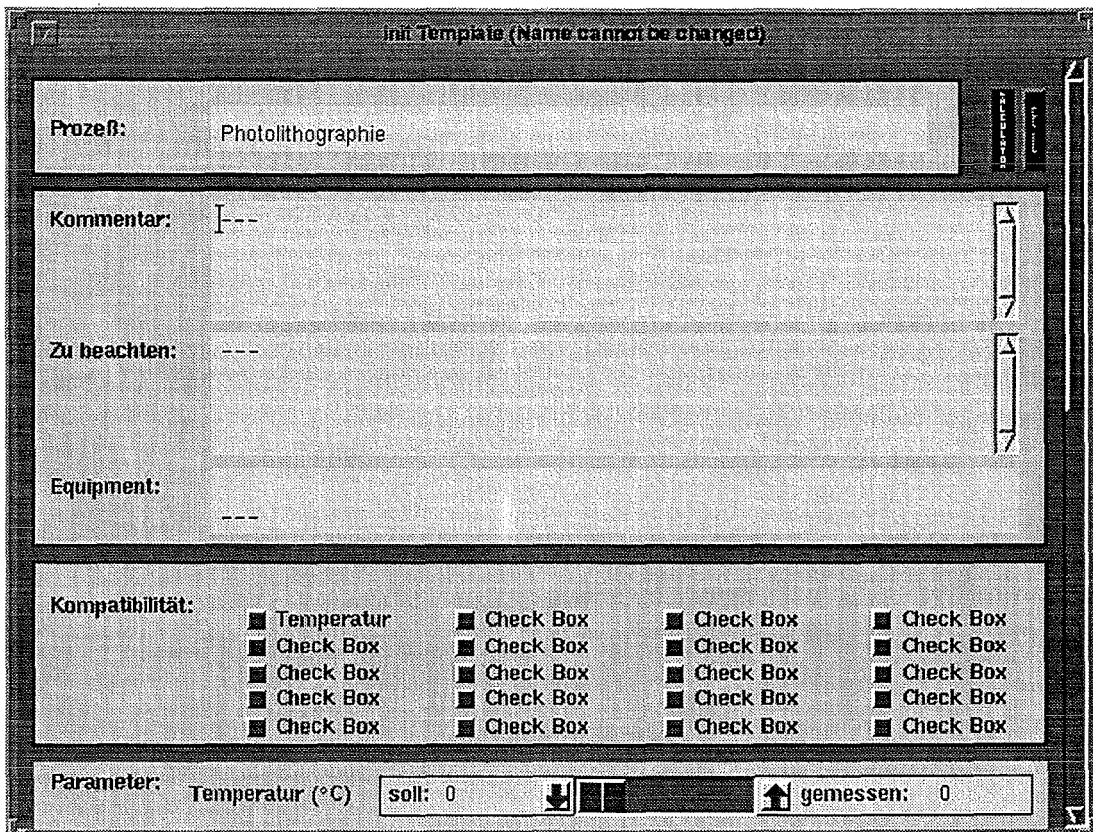


Abb. 17
Prozesseditor für Templates
Das Fenster ist mit einem Rollbalken versehen. Im nicht sichtbaren Teil befinden sich Eingabefelder für weitere prozeßspezifische Parameter.

Menü Work on Fragment

Beim Aufruf der Fragmentenverwaltung ist die mittlere Liste, die Prozeßliste des Fragments, leer. Wenn in der rechten Liste, der Fragmentliste, ein Fragment selektiert ist, werden dessen Prozesse in der mittleren Liste dargestellt. Oberhalb der Menüleiste ist der Name und die Projektgruppe des selektierten Fragments eingetragen. Ein Fragment ist immer genau einer Projektgruppe zugeordnet. Das Menü **Work on Fragment**, die Fragmentbearbeitung, enthält drei Einträge:

- **Edit Process**

Die Werte der Attribute einer im Fragment enthaltenen Prozeßinstanz können geändert werden.

- **Delete Process**

Eine Prozeßinstanz kann aus dem Fragment gelöscht und vernichtet werden.

- **New Fragment**

Die mittlere Liste wird geleert und erhält einen neuen Namen. Unter diesem Namen kann ein neues Fragment aufgebaut werden.

Solange der mittleren Liste kein Fragment zugeordnet ist, können die beiden ersten Einträge zwar ausgewählt werden. Der Anwender erhält aber die Warnmeldung **No process selected**. Dasselbe geschieht, wenn die Liste ein Fragment darstellt, aber kein Prozeß selektiert wurde.

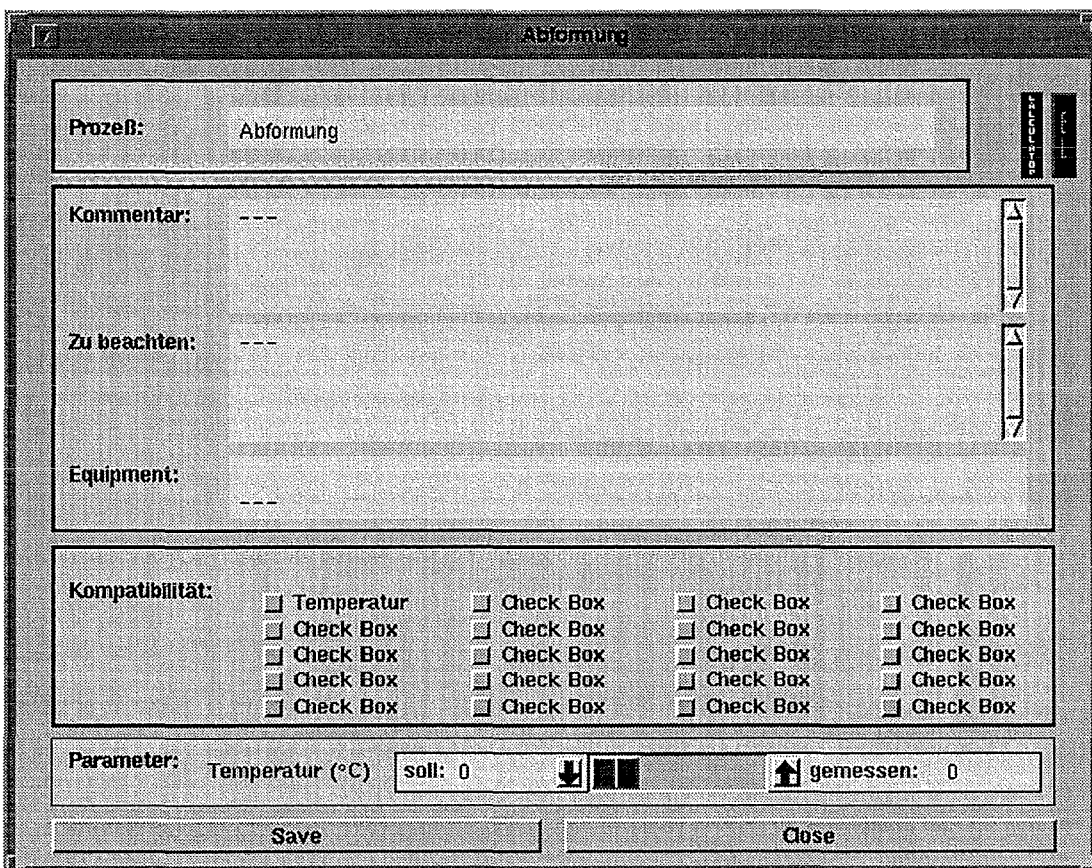


Abb. 18

Prozeßeditor in der Standardform

Für Prozeßtypen, die noch weitere Parameter enthalten, sind zusätzliche Eingabefelder vorhanden und das Fenster ist mit einem Rollbalken versehen.

Edit Process kann also aktiviert werden, sobald ein Fragment und darin ein Prozeß selektiert sind. Es kann beliebig viele gleichzeitige Lesezugriffe auf ein Fragment geben, jedoch nur einen Schreibzugriff ohne gleichzeitige Lesezugriffe. Das Editieren eines Prozesses ist mit einem Schreibzugriff auf das dargestellte Fragment verbunden.

Falls der Benutzer bisher nur lesend zugegriffen hat, gleichzeitig aber auch andere Benutzer auf das Fragment zugreifen, kann keine Schreibsperre gesetzt werden. In diesem Fall erscheint die Warnmeldung **Permission denied** und anschließend die Abfrage **Open read only?** Wird die Frage verneint, ist der Vorgang beendet, ansonsten wird der zum Prozeßtyp gehörige Prozesseditor geöffnet, wobei die Werte der Parameter allerdings nicht geändert werden können.

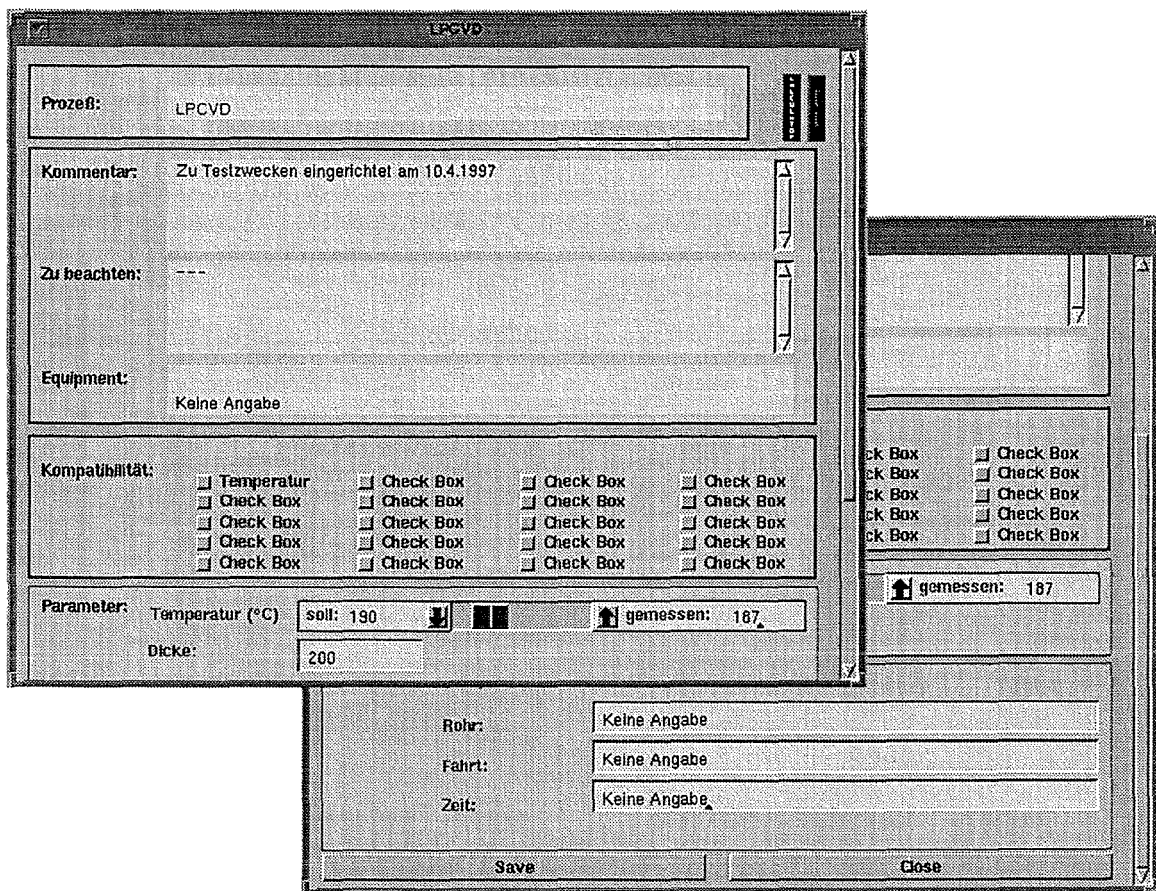


Abb. 19

Prozeßeditor für den Prozeß LPCVD

Das Fenster ist mit einem Rollbalken versehen. Im oberen Teil werden die allen Prozessen gemeinsamen Eigenschaften dargestellt. Der untere Teil (z.T. verdeckt) enthält spezielle Parameter des jeweiligen Prozesses.

Falls eine Schreibsperre gesetzt werden darf, wird gleich nach Auswahl des Menüeintrags ein Prozeßeditor mit Schreibrechten geöffnet. Abb. 18 zeigt den Editor für den Prozeßtyp **Abformung**. Für Prozeßtypen, die noch weitere Parameter enthalten, sind zusätzliche Eingabefelder vorhanden und das Fenster ist mit einem Rollbalken versehen. In Abb. 19 ist als Beispiel

der Editor für den Typ **LPCVD** dargestellt. Die Abbildung besteht aus zwei Teilen, weil das Fenster mit einem Rollbalken versehen ist und nicht alle Eigenschaften dieses Prozesses gleichzeitig dargestellt sind. Wenn die Eingabefelder gesperrt sind, ist auch der Button **Save** deaktiviert. Die Implementierung von Prozessen und den Editoren ist in Kapitel 3.2 beschrieben.

Das Feld **Prozeß** enthält hier den Namen des Prozesses. Wird der Name geändert, dann wird beim Verlassen des Eingabefeldes die Änderung sofort beim Eintrag in der Prozeßliste des Fragments übernommen. Falls der Editor ohne Speichern geschlossen wird, wird auch die Änderung in der Liste wieder rückgängig gemacht.

Die Felder **Kommentar**, **Zu beachten** und **Equipment** fassen jeweils beliebig viel Text. Die ersten beiden sind in der vertikalen Richtung mit Rollbalken versehen. Beim Feld **Equipment** fließt der Text nach links hinaus. Es kann ein beliebiger Ausschnitt eines langen Textes in den sichtbaren Feldbereich geholt werden.

Das Feld **Kompatibilität** hat derzeit noch keine Bedeutung. Insbesondere werden Einstellungen, die dort vorgenommen werden, nicht übernommen.

Der Sollwert des Parameters Temperatur kann mittels Tastatur (Mauszeiger mit linker Maustaste setzen), mit den beiden Pfeilschaltern oder mit dem Schieberegler zwischen den Feldern **soll** und **gemessen** variiert werden. Der gemessene Temperaturwert läßt sich nur über die Tastatur eingeben.

Der Button **Save** speichert den Prozeß mit den geänderten Werten wieder beim Fragment. Damit sind die Änderungen aber noch nicht persistent. Dazu muß erst das gesamte Fragment gespeichert werden (Seite 35).

Bei Betätigen von **Close** wird der Benutzer gefragt, ob er Änderungen speichern möchte oder nicht. Das Speichern entspricht hier dem Vorgang beim Betätigen von **Save**. Unabhängig von einer Speicherung wird der Editor nach Beantwortung der Frage geschlossen. **Quit** im Menü des Fensters entspricht **Close** ohne Speichern.

Das Fenster zur Prozeßinitialisierung enthält neben dem Feld **Prozeß** einen dunklen vertikalen Balken. Dieser Button ruft einen einfachen Taschenrechner auf, wie er in Abb. 20 zu sehen ist.

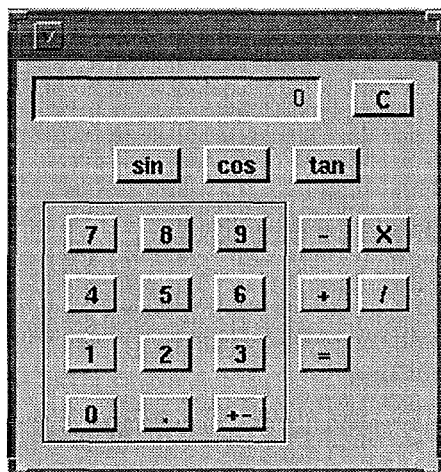


Abb. 20
Einfacher Taschenrechner wie er in die
Prozeßeditoren integriert ist

Der rote Balken rechts dient dazu, das Hauptfenster wiederaufzufinden. Es wird bei Betätigung des Balkens in den Vordergrund geholt oder auch geöffnet, falls es als Icon vorlag. Auch andere Fenster des Systems Elab enthalten einen solchen Balken, der unterschiedlich groß sein kann.

Der Eintrag **Delete Process** im Menü **Work on Fragment** löscht einen Prozeß aus der Prozeßliste des Fragments. Ein zu diesem Prozeßschritt evtl. geöffneter Prozeßeditor wird geschlossen. Auch das Löschen eines Prozesses ist selbstverständlich nicht möglich, wenn der Benutzer keine Schreibrechte für das betreffende Fragment erhält. In diesem Fall erscheint bei Auswahl des Menüeintrags die Warnmeldung **Permission denied**. Das Löschen eines Prozesses muß vom Benutzer bestätigt werden (**Really delete?**).

Mit dem dritten Eintrag **New Fragment** kann ein neues Fragment angelegt werden. Der Anwender kann über ein Textfeld den Namen des neuen Fragments wählen. Es sind alle alphanumerischen Zeichen, aber auch Sonderzeichen und das Leerzeichen erlaubt. Standardmäßig ist der Name **Fragment** angegeben.

Falls der Benutzer mehreren Projektgruppen angehört, kann er über eine Auswahlliste (s. Abb. 4, Seite 14) diejenige Gruppe angeben, zu der das Fragment gehören soll. Falls er hier abbricht, wird der erste Name in der Liste der Benutzer-Projektgruppen verwendet. Der Vorgang kann hier abgebrochen werden.

Falls der Name bereits existiert, wird der Benutzer darüber informiert und gleichzeitig gefragt, ob er den Vorgang wiederholen möchte (**Name exists. Retry?**). Diese Abfrage wird solange fortgesetzt, bis ein gültiger Name angegeben oder abgebrochen wird.

Auch wenn ein Fragment mit dem abgelehnten Namen sogar nach einer Aktualisierung der Fragmentliste nicht in dieser aufgeführt ist, kann es trotzdem zur Namenskollision kommen. In der Fragmentliste sind nämlich immer nur die Fragmente aufgeführt, zu deren Projektgruppe der betreffende Benutzer gehört.

Falls ein neues Fragment erzeugt wird, erscheinen der Name und die Projektgruppe über der mittleren Liste. Das Fragment wird in der Fragmentliste rechts aufgeführt und ist automatisch selektiert. Außerdem ist es sofort persistent und bleibt damit auch erhalten, wenn die Fragmentenverwaltung ohne weitere Sicherungsmaßnahmen geschlossen wird. Dies gilt nicht für weitere Änderungen am Fragment.

Menü **Work on Fragment List**

Dieses Menü zur Bearbeitung der Fragmentliste enthält vier Einträge:

- **Add Fragment**

In das aktuell selektierte und in der mittleren Liste dargestellte Fragment wird die Prozeßsequenz des selektierten Fragments eingefügt.

- **Rename Fragment**

Das selektierte Fragment erhält einen neuen Namen.

- **Copy Fragment**

Eine Kopie des selektierten Fragments wird unter einem neuen Namen angelegt. Dabei wird der Name des Originals in der Versionsliste der Kopie am Beginn eingefügt.

- **Delete Fragment**

Das selektierte Fragment wird aus der Fragmentliste entfernt und vernichtet.

- **Show History**

Die Versionsliste des selektierten Fragments wird angezeigt. Einträge können gelöscht werden.

Wird einer dieser fünf Einträge ausgewählt ohne daß ein Fragment in der Fragmentliste selektiert ist, erscheint eine entsprechende Meldung (**No fragment selected**).

Falls ein Fragment selektiert wird, das gesperrt ist, erscheint die Meldung **Fragment is write locked**. Anschließend ist auch ein vorher selektiertes Fragment nicht mehr selektiert. Ein selektiertes Fragment wird immer deselektiert, wenn irgendein Eintrag der Fragmentliste selektiert wird oder wenn auf eben dieses selektierte Fragment geklickt wird.

Falls ein Fragment selektiert wird, das zwischenzeitlich gelöscht wurde und mangels Aktualisierung immer noch in der Fragmentliste angezeigt wird, erscheint die Meldung

Fragment not found.

Updating fragment list.

Eine Aktualisierung erfolgt automatisch. Danach ist kein Fragment selektiert.

Fragmente werden nicht explizit gespeichert. Beim Deselektieren eines Fragments verschwinden dessen Prozeßschritte aus der mittleren Liste und evtl. Änderungen werden persistent. Geöffnete Prozeßeditoren werden geschlossen, wobei die Änderungen innerhalb der Prozesse nur dann persistent werden, wenn sie innerhalb der Editoren gesichert wurden (s. Seite 33). Fragmente, die explizit für den Zugriff durch andere Benutzer gesperrt sind werden demzufolge auch erst beim Deselektieren wieder freigegeben.

Die Funktionen der fünf Menüeinträge bei selektiertem Fragment sind im folgenden beschrieben:

Bei Auswahl des ersten Eintrags **Add Fragment** wird auf dem Bildschirm eine zusätzliche Fragmentliste - vom Inhalt her identisch mit der Fragmentliste rechts im Fenster der Fragmentenverwaltung - als Auswahlliste (s. Abb. 4, Seite 14) dargestellt. Daraus kann der Anwender ein Fragment wählen oder diesen Vorgang abbrechen. Wenn auf das ausgewählte Fragment nicht zugegriffen werden darf, erhält der Benutzer die Meldung **Fragment locked** und der Vorgang wird abgebrochen.

Die Prozeßsequenz eines ausgewählten Fragments wird in der mittleren Liste nach dem dort selektierten Prozeßschritt eingefügt. Falls dort kein Prozeßschritt selektiert ist, wird die Sequenz am Beginn dieser Liste eingefügt.

Falls das Fragment bereits von einem anderen Benutzer gelesen wird, kann es nicht zum Schreiben gesperrt werden. Bei Auswahl von **Add Fragment** erscheint die Meldung **Permission denied** und der Vorgang wird abgebrochen. Andernfalls wird das Fragment für Zugriffe durch andere Benutzer gesperrt.

Mit dem Menüeintrag **Rename Fragment** kann der Name des selektierten Fragments geändert werden. In einem Textfeld können alphanumerische Zeichen, Sonderzeichen und das Leerzeichen verwendet werden.

Falls der neue Name mit einem anderen identisch ist, kann der Benutzer entscheiden, ob er einen anderen Namen wählen oder den Vorgang abbrechen will (**Name exists. Retry?**). Auch wenn ein Fragment mit dem abgelehnten Namen sogar nach einer Aktualisierung der Fragmentliste nicht in dieser aufgeführt ist, kann es trotzdem zur Namenskollision kommen. In der Fragmentliste sind nämlich immer nur die Fragmente aufgeführt, zu deren Projektgruppe der betreffende Benutzer gehört.

Nach erfolgreicher Namensänderung wird das Fragment unter seinem neuen Namen in der Liste angezeigt und bleibt weiterhin selektiert. Auch über der mittleren Liste der Fragmentverwaltung wird der Name geändert.

Falls das Fragment bereits von einem anderen Benutzer gelesen wird, kann es nicht zum Schreiben gesperrt werden. Bei Auswahl von **Rename Fragment** erscheint die Meldung **Permission denied** und der Vorgang wird abgebrochen. Andernfalls wird das Fragment für Zugriffe durch andere Benutzer gesperrt.

Mit **Copy Fragment** kann in der Fragmentliste eine Kopie des selektierten Fragments unter einem neuen Namen angelegt werden. Der Name der neuen Kopie wird in einem Textfeld angegeben. Es sind alphanumerische Zeichen, Sonderzeichen und das Leerzeichen erlaubt.

Falls der Benutzer mehreren Projektgruppen angehört, kann er über eine Auswahlliste diejenige Gruppe angeben, zu der das Fragment gehören soll. Falls er hier abbricht, wird der erste Name in der Liste der Benutzer-Projektgruppen verwendet.

Falls der neue Name mit einem anderen identisch ist, kann der Benutzer entscheiden, ob er einen anderen Namen wählen oder den Vorgang abbrechen will (**Name exists. Retry?**). Auch wenn ein Fragment mit dem abgelehnten Namen sogar nach einer Aktualisierung der Fragmentliste nicht in dieser aufgeführt ist, kann es trotzdem zur Namenskollision kommen. In der Fragmentliste sind nämlich immer nur die Fragmente aufgeführt, zu deren Projektgruppe der betreffende Benutzer gehört.

Nach erfolgreicher Namensänderung wird das Fragment unter seinem neuen Namen in der Liste angezeigt. Das Original bleibt weiterhin selektiert.

Die Kopie ist eine neue Version des ursprünglichen Objekts. Jedes Fragment speichert Informationen über seine Herkunft, indem die Namen derjenigen Objekte, aus denen es durch fortgesetztes Kopieren erzeugt wurde, in einer Liste festgehalten werden. Diese Versionsliste wird mitkopiert. Dabei werden beim Kopieren die Namen des Originals jeweils am Beginn der Versionsliste der Kopie eingefügt. Einzelne Namen können aus dieser Liste gelöscht werden (s. unten). Der Inhalt der Versionslisten ist unabhängig davon, ob die Fragmente, deren Namen darin angegeben sind, noch existieren oder evtl. gelöscht wurden. In der Versionsliste können auch Namen von Fragmenten enthalten sein, die der betreffende Benutzer nie zu sehen bekommt, weil er nicht der Projektgruppe dieser Fragmente angehört. Sie können deshalb enthalten sein, weil evtl. ein anderer Benutzer, der von diesen Fragmenten eine Kopie erzeugt hat, sowohl deren Projektgruppe als auch der Gruppe der Kopie angehörte.

Versionslisten besitzen außer Fragmenten auch die Protokolle (s. Kapitel 2.2.5).

Kopieren ist auch dann erlaubt, wenn das Fragment auch von anderen Benutzern gelesen wird, weil beim Kopieren das Original nicht geändert wird. Aus diesem Grund wird das Fragment während des Vorgangs auch nicht für andere Benutzer gesperrt.

Mit **Delete Fragment** kann das selektierte Fragment nach Bestätigung in einem Fenster mit der Beschriftung **Really delete?** aus der Fragmente Liste entfernt und vernichtet werden.

Falls das Fragment bereits von einem anderen Benutzer gelesen wird, kann es nicht zum Schreiben gesperrt werden. Bei Auswahl von **Delete Fragment** erscheint die Meldung **Permission denied** und der Vorgang wird abgebrochen. Andernfalls wird das Fragment für Zugriffe durch andere Benutzer gesperrt, weil es nicht sofort oder, falls der Benutzer die Bestätigung verweigert, überhaupt nicht gelöscht wird.

Nach dem Löschen ist kein Fragment selektiert. Die mittlere Liste der Fragmentenverwaltung ist leer und über dieser Liste ist kein Name und keine Projektgruppe mehr eingetragen.

Der Eintrag **Show History** bewirkt die Darstellung einer Auswahlliste mit dem Inhalt der Versionsliste, also den Namen derjenigen Fragmente, aus denen das vorliegende schließlich durch fortgesetztes Kopieren erzeugt wurde.

Es ist möglich, einen Namen aus dieser Liste zu löschen, unabhängig davon, ob das betreffende Fragment noch existiert oder nicht. Der Button **OK** löscht einen selektierten Namen. Das Löschen eines Namens aus der Versionsliste stellt eine Änderung am Fragment dar. Wenn das selektierte Fragment also gleichzeitig von einem anderen Benutzer gelesen wird, kann es nicht gesperrt werden und es erscheint die Warnmeldung **Permission denied**. Andernfalls wird das selektierte Fragment für andere Benutzer gesperrt. In jedem Fall erscheint nach dem Löschen oder nach der Warnmeldung die Auswahlliste erneut. Die Funktion **Show History** kann nur durch **Cancel** in der Auswahlliste abgebrochen werden.

Drucken

Im Fenster Fragmentenverwaltung befindet sich unter jeder Liste ein **Print**-Button. Während **Print Templates** und **Print Fragment List** den Inhalt der jeweiligen Liste ausdrucken, besteht der Ausdruck bei **Print Fragment** aus einer Auflistung aller im Fragment enthaltenen Prozessschritte mit allen Attributen. Als Beispiel ist hier der Ausdruck eines Fragments namens **Drehspiegel_komplett** dargestellt:

Elektronisches Laborbuch Fragment: Drehspiegel_komplett	Datum: 4.1.1996
--	-----------------

Häusen_Blechabdeckung Kommentar : keine Angabe Zu beachten: keine Angabe Equipment : keine Angabe Temperatur : Soll: 0 °C Ist: 0 °C

Kleben_UVKleber Kommentar : keine Angabe Zu beachten: keine Angabe Equipment : keine Angabe Temperatur : Soll: 0 °C Ist: 0 °C

Kleben_selbstklebend Kommentar : keine Angabe Zu beachten: keine Angabe Equipment : keine Angabe Temperatur : Soll: 0 °C Ist: 0 °C
--

Reflow_Löten Kommentar : keine Angabe Zu beachten: keine Angabe Equipment : keine Angabe Temperatur : Soll: 0 °C Ist: 0 °C
--

Fügen_Basisplatte Kommentar : keine Angabe Zu beachten: keine Angabe Equipment : keine Angabe Temperatur : Soll: 0 °C Ist: 0 °C

Brechen Kommentar : keine Angabe Zu beachten: keine Angabe Equipment : keine Angabe Temperatur : Soll: 0 °C Ist: 0 °C

Bei Ausgabe in eine PostScript-Datei heißen die entsprechenden Dateien **Prozesse.ps**, **Frag-List.ps** und **Fragment.ps** (s. auch die Ausführungen zur Druckereinstellung auf Seite 19).

2.2.5 Erzeugung und Verwaltung von Protokollen

Beim Chargensplitting werden, ausgehend von einem bestimmten Prozeßschritt, verschiedene Varianten für die weitere Vorgehensweise getestet. Die Prozesse sind dann nicht mehr linear angeordnet wie in den Fragmenten. Vielmehr kann sich die Prozeßfolge von jedem beliebigen Prozeß aus verzweigen und bildet dann eine Baumstruktur.

Ein solcher Baum dokumentiert die Vorgehensweise der Prozeßentwickler bei ihrer Suche nach einer optimalen Prozeßfolge für den Herstellungsprozeß eines Mikrosystems. Im Elektronischen Laborbuch wurde hierfür die Bezeichnung Protokoll eingeführt. Informationen über die Implementierung von Protokollen enthält Kapitel 3.2. Ein Protokoll kann mit Hilfe des Protokollgenerators auf zwei verschiedene Arten erzeugt werden. Entweder wird ein Fragment, also eine noch unverzweigte Prozeßfolge, kopiert und die Kopie in ein Protokoll umgewandelt oder es wird eine Kopie eines bereits vorhandenen Protokolls als Basis für ein neues Protokoll ausgewählt.

Wenn der Anwender im Hauptfenster (Abb. 8, Seite 17) den Button **Logs** selektiert, erhält er ein Fenster zur Auswahl aus einer festen Anzahl von Möglichkeiten (s. Abb. 5, Seite 14). Der Protokollgenerator kann auch aus der Fragmentenverwaltung (s. Abb. 16, Seite 27) sowie aus dem Fenster zur Produktverwaltung & Statistik (s. Abb. 24, Seite 52) heraus aktiviert werden.

Das Auswahlfenster trägt die Beschriftung **choose** und stellt neben den erwähnten Möglichkeiten zur Erzeugung von Protokollen auch die Möglichkeit zum Aufruf bereits vorhandener Protokolle zur Verfügung:

- **Copy Logs**

Ein Protokoll wird neu erzeugt, indem eine Kopie eines bereits vorhandenen Protokolls erzeugt wird.

- **Logs**

Der Benutzer kann ein bereits vorhandenes Protokoll auswählen.

- **Fragments**

Der Benutzer kann ein Fragment auswählen, aus dem ein Protokoll erzeugt werden soll.

Beim Erzeugen eines Protokolls als Kopie eines bereits vorhandenen (**Copy Logs**) kann der Benutzer das zu kopierende Protokoll aus einer Auswahlliste (s. Abb. 4, Seite 14) mit der Beschriftung **Choose Log** auswählen. Diese Liste zeigt alle Protokolle, auf die der Benutzer gemäß seiner Projektgruppenzugehörigkeit zugreifen darf. In einem Texteingabefenster muß ein neuer Name für das zu erzeugende Protokoll angegeben werden. Falls dieser Name bereits existiert, erscheint eine Warnung mit Abfrage, ob der Benutzer einen neuen Versuch wünscht (**Name exists. Retry?**). Der Vorgang wird dann entweder abgebrochen oder es erscheint erneut eine Auswahlliste. Auch wenn ein Protokoll mit dem abgelehnten Namen nicht in der Auswahlliste aufgeführt war, kann es trotzdem zur Namenskollision kommen. In der Liste sind nämlich immer nur diejenigen Protokolle aufgeführt, zu deren Projektgruppe der betreffende Benutzer gehört.

Falls der Benutzer mehreren Projektgruppen angehört, kann er über eine Auswahlliste diejenige Gruppe angeben, zu der das Fragment gehören soll. Falls er hier abbricht, wird der erste Name in der Liste der Benutzer-Projektgruppen verwendet.

Falls das zum Kopieren ausgewählte Protokoll gesperrt ist, erscheint die Warnmeldung **Permission denied** und der Vorgang wird abgebrochen.

Die Kopie eines Protokolls ist eine neue Version. Der Name des Originals wird bei der Kopie am Beginn einer Versionsliste gespeichert. Über Versionslisten s. Seite 36.

Beim Betätigen von **Logs** erhält der Benutzer eine Auswahlliste mit allen Protokollen, auf die er zugreifen darf (**Choose Log**). Falls ein ausgewähltes Protokoll gesperrt ist, erscheint die Warnmeldung **Permission denied** und der Vorgang wird abgebrochen. Andernfalls wird das Protokoll dargestellt.

Mit dem Button **Fragments** erhält der Benutzer eine Auswahlliste mit allen Fragmenten, auf die er aufgrund seiner Projektgruppenzugehörigkeit zugreifen darf (**Choose Fragment**). Er kann eines dieser Fragmente auswählen und ein neues, noch unverzweigtes Protokoll generieren. In einem Texteingabefenster muß ein neuer Name für das zu erzeugende Protokoll angegeben werden. Falls dieser Name bereits existiert, erscheint eine Warnung mit Abfrage, ob der Benutzer einen neuen Versuch wünscht (**Name exists. Retry?**). Der Vorgang wird dann entweder abgebrochen oder es erscheint erneut eine Auswahlliste.

Falls der Benutzer mehreren Projektgruppen angehört, kann er über eine Auswahlliste diejenige Gruppe angeben, zu der das Fragment gehören soll. Falls er hier abbricht, wird der erste Name in der Liste der Benutzer-Projektgruppen verwendet.

Falls das zum Kopieren ausgewählte Fragment gesperrt ist, erscheint die Warnmeldung **Permission denied** und der Vorgang wird abgebrochen.

Sobald ein Protokoll neu generiert oder ausgewählt wurde, wird es in einem Chargensplitting-Workfloweditor dargestellt. Es ist dann für jeglichen weiteren Zugriff gesperrt. Ein neu generiertes Objekt ist sofort persistent. Das gilt aber nicht für evtl. folgende Änderungen im Editor.

Der Chargensplitting-Workfloweditor (im folgenden kurz Workfloweditor genannt) ist ein graphischer Editor, der vielfältige Manipulationen an einem Protokoll erlaubt. Abb. 21 zeigt die Darstellung eines Protokolls in einem solchen Editor.

Die Überschrift des Editors lautet: **Chargensplitting Workflow Editor for: [Name]; Project Group: [Projektgruppe]**.

Der Editor zeigt auf einer zentralen Zeichenfläche einen Graphen des Herstellungsprozesses. Dabei sind die einzelnen Prozeßschritte als Knoten durch Pfeile miteinander verbunden, die deren zeitliche Abfolge symbolisieren. Die Prozeßschritte können selektiert werden, wobei sich deren Darstellung je nach Darstellungsmodus (s. Menü **Extra**, Seite 43) ändert.

Zu Beginn sind - wenn vorhanden - die ersten beiden Prozeßschritte abgebildet. Der Anfangsknoten befindet sich am linken Rand der Zeichenfläche. Das Diagramm kann nach Bedarf expandiert werden (s. Seite 48 und Seite 50). Falls das betreffende Protokoll keinen Prozeßschritt beinhaltet, ist nur ein einzelner Knoten mit der Aufschrift **dummy** abgebildet.

Unterhalb der Zeichenfläche befindet sich ein Textfeld und eine Fußzeile enthält Hinweise zur Mausbenutzung.

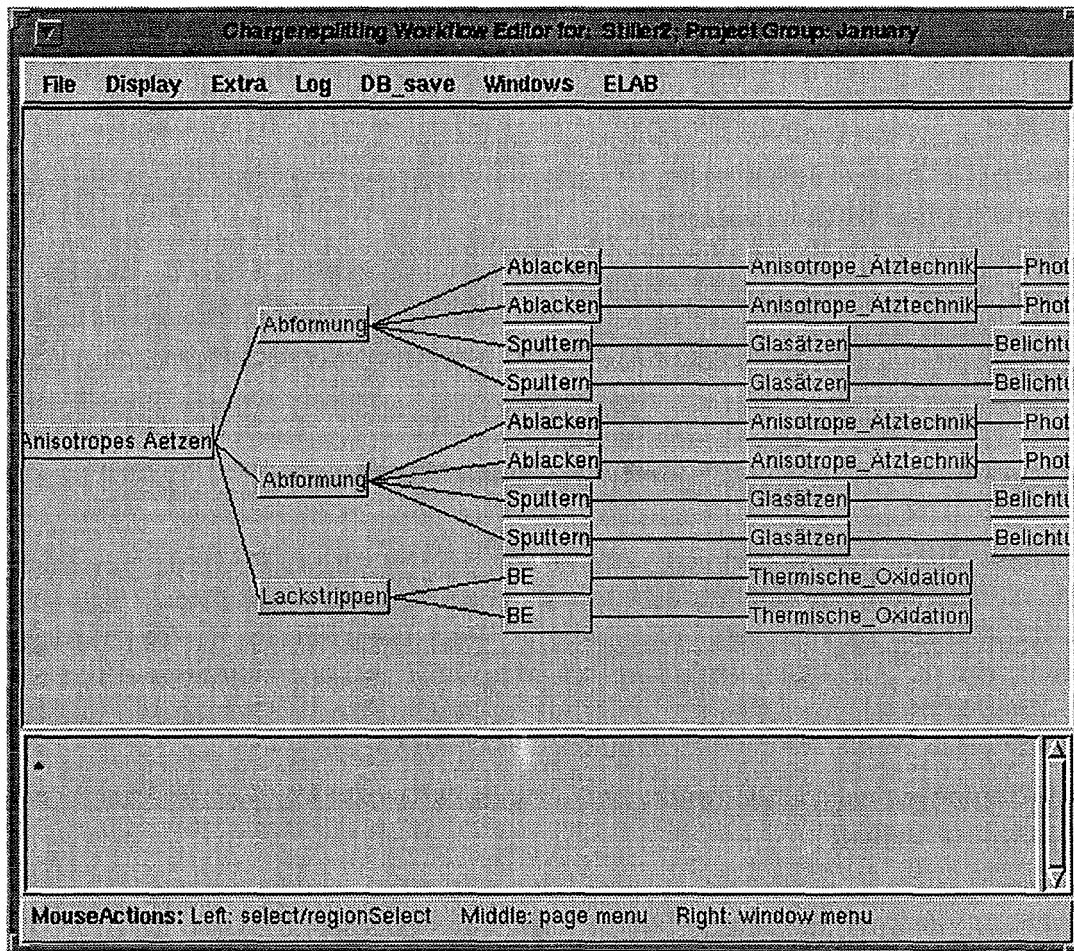


Abb. 21

Workfloweditor:

Die graphische Darstellung zeigt Prozessschritte und ihre Vernetzung (zeitliche Abfolge). Nach dem Aufruf des Editors sind zwei Prozessschritte dargestellt. Weitere Prozessschritte können nach Bedarf zugeschaltet werden. Die Navigation durch den dargestellten Graphen sowie das Editieren werden im Text ausführlich erläutert.

Geschlossen wird ein Workfloweditor nur durch Betätigen von **Quit** im Menü des Fensters. Der Benutzer kann in einem Fenster zur Bestätigung mit der Beschriftung **Saving Log?** wählen, ob die Änderungen, die seit dem letzten Speichern oder seit Aufruf des Editors erfolgt sind, persistent gespeichert werden sollen oder nicht.

Oberhalb der Zeichenfläche befindet sich die Hauptmenüleiste, deren Einträge im folgenden besprochen werden.

- **File**

Dieses Menü enthält die Einträge **load document ...** und **save ...**.

Die Funktion Laden ist derzeit nicht verfügbar.

Beim Sichern wird im Startverzeichnis eine Datei geschrieben, deren Namen über ein Textfeld angegeben werden kann. Ist der Name bereits vorhanden, wird der Anwender gewarnt.

- **Display**

Dieses Menü hat zwei verschiedene Listen mit Einträgen. Zunächst werden die Einträge im Normalzustand beschrieben:

In einem Textfeld kann mit **display aspect** der Name einer Methode der Klasse `RrProtokollNode` angegeben werden. Diese Methode muß ein Objekt zurückliefern, das als Beschriftung der Knoten des Graphen ausgegeben werden kann. Diese Ausgabe wird veranlaßt.

Durch **redraw** wird das Neuzeichnen des Graphen veranlaßt.

Wenn der Eintrag **zoom** selektiert wird nimmt der Mauszeiger die Form eines Kreuzes an. Innerhalb der Zeichenfläche kann - bei gedrückter linker Maustaste - ein Rechteck aufgezogen werden. Solange dieses Rechteck innerhalb der Zeichenfläche bleibt, wird die Abbildung vergrößert und zwar um so mehr, je kleiner das Rechteck ist. Das Rechteck kann auch über den Rand der Zeichenfläche hinausgezogen werden, wodurch eine Verkleinerung der Abbildung bewirkt wird.

Sobald **zoom** einmal betätigt wurde, ist die zweite Liste mit Einträgen des Menüs **Display** aktiv (s. unten).

Durch die Auswahl des Eintrags **adapt page** wird der gesamte Graph innerhalb der sichtbaren Zeichenfläche abgebildet. Normalerweise ist die Größe der Knoten des Graphes auf einen Standardwert eingestellt, so daß bei einer größeren Anzahl von Knoten immer nur ein Teil des Graphen sichtbar ist.

Die Größe des Graphen wird mit **normal size** auf den Standardwert zurückgesetzt. Er wird so positioniert, daß sich der Anfangsknoten am linken Rand der Zeichenfläche befindet.

Die zweite Liste mit Einträgen des Menüs **Display** wird aktiviert, wenn die Größe der Abbildung mittels **zoom** verändert wird. Es sind hier folgende Einträge vorhanden:

Durch **redraw** wird das Neuzeichnen des Graphen veranlaßt.

Mit **grid on/off** kann der Graph mit einem Gitter (Standardgröße 75 x 75 Pixel ohne Zoom) hinterlegt werden. Das Gitter wird abwechselnd ein- bzw. ausgeschaltet. Es bleibt auch dann sichtbar, wenn mit **entire page** zur Normalgröße zurückgekehrt wird (s. unten) und damit die erste Liste mit Einträgen des Menüs **Display** aktiviert wird.

In einem Textfeld kann bei Auswahl von **size** die Größe des Gitters in Pixel angegeben werden. Das Gitter muß nicht quadratisch sein; es können unterschiedliche Werte gewählt werden. Die Werte müssen durch das Zeichen '@' getrennt werden.

Wenn der Eintrag **zoom** selektiert wird nimmt der Mauszeiger die Form eines Kreuzes an. Innerhalb der Zeichenfläche kann - bei gedrückter linker Maustaste - ein Rechteck aufgezogen werden. Solange dieses Rechteck innerhalb der Zeichenfläche bleibt, wird die Abbildung vergrößert und zwar um so mehr, je kleiner das Rechteck ist. Das Rechteck kann auch über den Rand der Zeichenfläche hinausgezogen werden, wodurch eine Verkleinerung der Abbildung bewirkt wird.

Die zweite Liste mit Einträgen des Menüs **Display** bleibt erhalten.

Der Graph wird mit **entire page** auf die Normalgröße zurückgesetzt. Der Anfangsknoten befindet sich in der Ursprungsposition am linken Rand der Zeichenfläche. Ein evtl. unterlegtes Gitter bleibt erhalten.

Das Menü **Display** enthält nun wieder die erste Liste mit Einträgen.

Der Ausschnitt aus dem Graphen, der gerade in der Zeichenfläche zu sehen ist, wird durch **page shift left (shift right; shift up; shift down)** um ein kleines Stück (etwa um die Größe eines Knotens) nach links (rechts; oben; unten) verschoben, d.h. der Graph wandert in die entgegengesetzte Richtung.

Der Effekt stellt sich erst ein, wenn nach der Auswahl des entsprechenden Eintrags die linke Maustaste in der Zeichenfläche betätigt und anschließend **redraw** selektiert wird.

- **Extra**

confirmation ist ein Schalter, mit dem ausgewählt werden kann, ob bei bestimmten Operationen, etwa dem Löschen eines Knotens (s. Seite 49) nach einer zusätzlichen Bestätigung gefragt wird oder nicht.

Mit **node display** kann ausgewählt werden, ob ein Knoten mit 3D-Berandung (**3-D border**), 2D-Berandung (**2-D border**) oder ohne Berandung (**no border**) dargestellt wird.

- **Log**

Dieses Menü enthält die Einträge **create fragment**, **show history** und **delete log**.

Um ein neues Fragment aus einem dargestellten Protokoll zu erzeugen (mit **create fragment**), muß genau ein Prozeßschritt im Graphen selektiert werden (Selektion eines Prozeßschritts s. Seite 50). Die Selektion von mehr als einem Prozeßschritt erzeugt eine Warnmeldung **No Process selected**, ebenso, wenn kein Prozeßschritt selektiert ist.

Das Fragment beginnt mit dem Anfangsknoten des Protokolls. Der selektierte Prozeßschritt ist das Ende des neu zu erzeugenden Fragments. Befindet sich dieser Knoten in einem Zweig des Protokolls (s. Chargensplitting, Seite 48), dann besteht das neue Fragment aus der Prozeßschrittfolge vom Anfangsknoten bis zum selektierten Knoten. Das neue Fragment wird unter einem neuen Namen, den der Anwender in einem Texteingabefeld angeben muß, persistent generiert.

Falls der Name bereits existiert, wird der Benutzer gewarnt (**Name exists. Retry?**) und kann entscheiden, ob er einen neuen Versuch wünscht oder den Vorgang abbrechen möchte.

Der Eintrag **show history** bewirkt die Darstellung einer Auswahlliste mit dem Inhalt der Versionsliste, also den Namen derjenigen Protokolle, aus denen das vorliegende schließlich durch fortgesetztes Kopieren erzeugt wurde.

Es ist möglich, einen Namen aus dieser Liste zu löschen, unabhängig davon, ob das betreffende Protokoll noch existiert oder nicht. Der Button **OK** löscht einen selektierten Namen. Nach dem Löschen erscheint die Auswahlliste erneut. Die Funktion **show history** kann nur durch **Cancel** in der Auswahlliste abgebrochen werden.

Mit dem Eintrag **delete log** kann ein bestehendes Protokoll - auch eines, das gerade bearbeitet wird - gelöscht werden. Dazu wird dem Benutzer eine Protokoll-Auswahlliste präsentiert, die alle Protokolle enthält, auf die er zugreifen darf.

Falls das zu löschende Protokoll nicht gerade bearbeitet wird, muß der Anwender das Löschen bestätigen (**Really delete?**). Ein Protokoll, das gerade bearbeitet wird - im vorliegenden Workfloweditor; für andere Zugriffe ist das Protokoll ja gesperrt - kann auch gelöscht werden. Dabei muß der Workfloweditor geschlossen werden. Die Beschriftung des Fensters zur Bestätigung lautet in diesem Fall:

Deleting current log.

Window will be closed.

Really delete?

Ein Protokoll, das in einem anderen Editor bearbeitet wird, ist gesperrt und der Vorgang wird mit der Warnmeldung **Permission denied** abgebrochen.

- **DB_save**

Der einzige Eintrag **DB save** sorgt dafür, daß alle seit dem Aufruf des Workfloweditors oder seit dem letzten Auswählen dieses Eintrags erfolgten Änderungen persistent werden.

- **Windows**

Der einzige Eintrag **Logs** öffnet einen Protokollgenerator (s. Seite 39) zur Erzeugung bzw. Darstellung eines weiteren Protokolls in einem weiteren Workfloweditor.

- **ELAB**

Dieses Menü enthält als einzigen Eintrag einen roten Button. Dieser entspricht dem in einigen Fenstern vorhandenen roten Balken und dient dem Wiederauffinden des Hauptfensters. Es wird bei Betätigung des Buttons in den Vordergrund geholt oder auch geöffnet, falls es als Icon vorlag. Auch andere Fenster des Systems Elab enthalten einen solchen Balken, der unterschiedlich groß sein kann.

Wenn sich der Mauszeiger innerhalb der Zeichenfläche des Workfloweditors befindet, können mit der linken Maustaste Knoten des Graphen selektiert und auf verschiedene Weise bearbeitet werden.

Betätigen der mittleren Maustaste liefert bei selektiertem Knoten ein Menü zur Bearbeitung des Graphen.

Falls kein Knoten selektiert ist, erhält der Anwender ein anderes Menü, das hauptsächlich Einträge enthält, die auch in der Menüleiste des Workfloweditors zu finden sind. Dieses Menü wird im folgenden beschrieben. Anschließend erfolgt die Beschreibung der Bearbeitung des Graphen.

- **file In/Out**

Dieses Menü entspricht dem Menü **File** in der Menüleiste des Workfloweditors (s. Seite 41). Die Einträge heißen hier **save document** und **load document**.

- **new root node**

Dieser Eintrag ist für Elab derzeit ohne Bedeutung.

- **tree from model**

Bei Betätigen dieses Eintrags erscheint das in Abb. 22 dargestellte Fenster auf dem Bildschirm. Dieses Fenster enthält mehrere Textfelder.

In den weiteren Textfeldern können Methoden angegeben werden, die bei der Bearbeitung des Graphen verwendet werden oder auch Relationen zwischen den Knoten des Graphen implementieren. Dabei handelt es sich um Methoden der Klasse **RrProzessNode**. Im einzelnen können folgende Punkte geändert werden:

Subnode Relation

Die Methode beschreibt die Beziehung zu den Nachfolgeknoten eines Knotens.

Supernode Relation

Diese Methode beschreibt die Beziehung zum Vorgängerknoten eines Knotens.

Definition Message

Unbekannt.

Rename Message

Diese Methode dient zum Ändern des Namens eines Prozeßschritts.

Change Supernode to Message

Mit dieser Methode kann der Vorgängerknoten geändert werden.

Create Subnode Message

Mit dieser Methode kann ein neuer Nachfolgeknoten erzeugt werden.

Remove Message

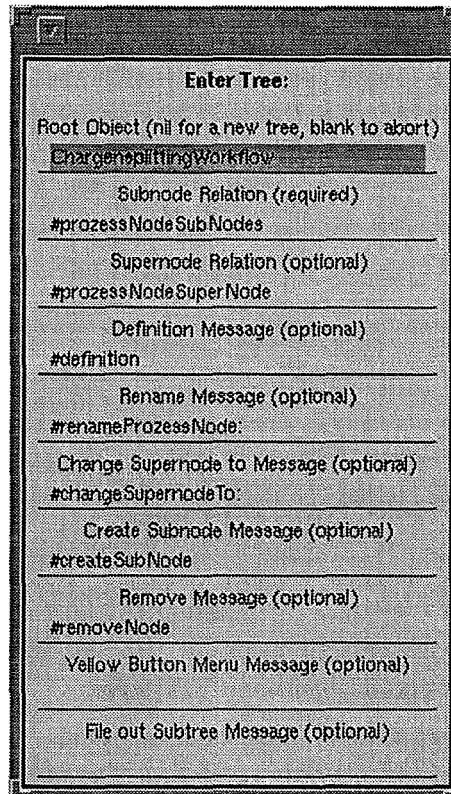
Die Methode entfernt einen Knoten aus dem Graph.

Yellow Button Menu Message

Unbekannt.

File out Subtree Message

Unbekannt.



The image shows a dialog box titled "Enter Tree:" with a list of input fields for graph manipulation. The fields are as follows:

- Root Object (nil for a new tree, blank to abort):
- Subnode Relation (required):
- Supernode Relation (optional):
- Definition Message (optional):
- Rename Message (optional):
- Change Supernode to Message (optional):
- Create Subnode Message (optional):
- Remove Message (optional):
- Yellow Button Menu Message (optional):
- File out Subtree Message (optional):

Abb. 22
Textfelder zur Manipulation des Graphen
und von Beziehungen zwischen den Knoten
des Graphen

- **find**

In einem Texteingabefeld mit der Beschriftung **Find #(#printString)** kann der Name eines Prozessschritts eingegeben werden. Für jeden Knoten im Graphen, der diesen Namen trägt erscheint auf der Zeichenfläche ein Button. Wird einer dieser Buttons selektiert, so wird ein Ausschnitt des Graphen in der Zeichenfläche abgebildet, der den dazugehörigen Knoten enthält. Der betreffende Knoten ist selektiert. Falls der betreffende Bereich des Graphen nicht dargestellt ist, wird er expandiert.

- **redraw**

Das Neuzeichnen des Graphen wird veranlaßt.

- **layout and redraw**

Hier handelt es sich um eine Erweiterung der **redraw**-Funktionalität, die auch Seitenzweige erfaßt, die unter Umständen nicht neu gezeichnet wurden.

- **view**

Die Einträge **zoom**, **adapt page** und **normal size** entsprechen den gleichnamigen Einträgen im Hauptmenü **Display** (Graph in Normalgröße) (s. Seite 42).

- **spawn**

Das Protokoll wird in einem weiteren Workfloweditor ein weiteres Mal dargestellt. Alle Änderungen, die in einem Editor vorgenommen werden, sind sofort auch in jedem anderen Editorfenster desselben Protokolls sichtbar.

Die einzelnen Zweige bei Chargensplitting (s. Seite 48) können in verschiedenen Editoren unterschiedlich angeordnet sein.

- **inspect**

Der Anwender kann sich hier die aktuellen Werte der Attribute der Zeichenfläche (Objekt der Klasse **TreeDocumentPage**) anschauen. Auf dem Bildschirm wird das Fenster aus Abb. 23 mit der Überschrift **TreeDocumentPage** dargestellt. Es enthält auf der linken Seite eine Liste der Attribute mit Angabe des Klassennamens; rechts wird der Wert eines selektierten Attributs angezeigt.

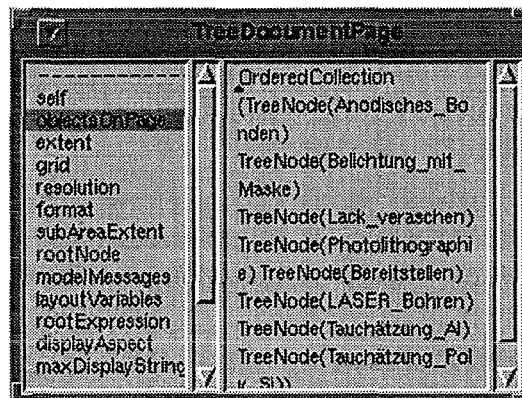


Abb. 23

*Darstellung der Attribute der Zeichenfläche
Der aktuelle Wert des Attributs **objectsOnPage**
wird rechts angezeigt. Hier handelt es sich um
ein Objekt der Klasse **OrderedCollection**, das
wiederum Objekte der Klasse **TreeNode** enthält.
Von diesen Objekten ist jeweils der Name ange-
geben.*

Die Darstellung kann durch Betätigen von **Quit** im Menü der Kopfzeile des Fensters beendet werden.

Das Menü, das bei Betätigen der mittleren Maustaste in der Zeichenfläche erscheint, falls genau ein Knoten des Graphen selektiert ist, sieht folgendermaßen aus:

- **toggle expansion**

Falls der selektierte Knoten nicht expandiert ist (Name des Knotens in Fettschrift), wird er expandiert und zwar genau so weit, wie er bisher per Hand expandiert worden ist (s. Seite 50), mindestens jedoch bis zu den direkten Nachfolgeknoten. Ist der Knoten expandiert, werden sämtliche Nachfolgeknoten in allen Zweigen vom Bildschirm gelöscht. Die Schriftart des Namens wechselt von Normalschrift zu Fettschrift, es sei denn es handelt sich um einen Endknoten.

- **expand all**

Alle Nachfolgeknoten des selektierten Knotens werden auf dem Bildschirm dargestellt.

- **center node**

Der Ausschnitt der Zeichenfläche wird so verschoben, daß der selektierte Knoten in der Mitte der Zeichenfläche dargestellt ist. Der Anfangsknoten bleibt jedoch immer am linken Rand der Zeichenfläche, d.h. die ersten zwei oder drei Knoten werden nur in der vertikalen Richtung zentriert.

- **add subnode**

Der selektierte Knoten erhält einen neuen Nachfolgeknoten. Durch die Möglichkeit, mehr als einen Nachfolger für einen Prozeßschritt anzugeben, kann ein Chargensplitting modelliert werden, wobei die einzelnen Zweige, die auf einen bestimmten Prozeßschritt folgen, unterschiedlich sein können.

Nach der Auswahl von **add subnode** erscheint auf dem Bildschirm ein Fenster zur Auswahl aus einer festen Anzahl von Möglichkeiten (s. Abb. 5, Seite 14). Die Auswahlmöglichkeiten sind durch die Buttons **Last process in list**, **Choose from List** und **New** vorgegeben.

Im ersten Fall wird der letzte Prozeß in der Liste der Nachfolger des selektierten Prozesses kopiert und als weiterer Nachfolger angefügt.

Im zweiten Fall kann der Benutzer aus einer Auswahlliste mit den Namen aller Nachfolger des selektierten Prozesses (**Choose successor**) einen auswählen, der dann kopiert und hinzugefügt wird.

Schließlich wird bei **New** eine Auswahlliste mit allen Templates vorgegeben Beschriftung (**Choose process**). Der neue Prozeßschritt erhält dann den Klassennamen.

- **insert supernode**

Der Anwender hat die gleichen Möglichkeiten, einen Prozeßschritt als neuen Vorgängerknoten zum selektierten Knoten auszuwählen, wie bei **add subnode** (s. oben). Die Liste, aus der bereits vorhandene Knoten kopiert werden können, ist hier die Liste der Nachfolgeknoten des alten Vorgängers.

Wird ein Knoten kopiert, so wird auch hier der gesamte Zweig, der bei diesem Knoten beginnt, mitkopiert.

- **rename node**

In einem Texteingabefeld kann ein neuer Name für den selektierten Prozeßschritt angegeben werden. Die Abbildung des Prozeßschritts wird entsprechend geändert. Falls der Prozeß allerdings gerade editiert wird, sollte die Namesänderung im Prozeßeditor geschehen (s. Seite 49). Der Benutzer wird in diesem Fall durch folgende Warnmeldung darauf hingewiesen:

Process is being edited.

Please use the editor to rename it.

Der Vorgang wird dann abgebrochen.

- **remove node**

Der selektierte Knoten kann gelöscht werden. Der Anwender muß das Löschen vorher bestätigen. Die Beschriftung des Fensters zur Bestätigung lautet:

Do you really want

to remove the node [Name]?

Die Nachfolgeknoten des gelöschten Knotens werden zu Nachfolgern des Vorgängers.

- **remove subtree**

Der gesamte Zweig, der am selektierten Knoten beginnt, kann gelöscht werden. Auch hier muß der Anwender bestätigen.

Do you really want

to remove the node [Name]

with all subnodes?

- **edit process**

Der Benutzer kann die Attribute des Prozesses, der durch den selektierten Knoten symbolisiert wird, editieren. Dazu erscheint auf dem Bildschirm das in Kapitel 2.2.4 beschriebene Prozeßeditorfenster (s. Abb. 12, Seite 23).

Wird der Name geändert, dann wird beim Verlassen des Eingabefeldes die Änderung sofort beim Knoten im Workfloweditor übernommen. Falls der Prozeßeditor ohne Speichern geschlossen wird, wird auch die Änderung im Workfloweditor wieder rückgängig gemacht.

- **inspect**

Der Anwender kann sich hier die aktuellen Werte der Attribute des selektierten Knotens (Objekt der Klasse `TreeNode`) anschauen. Auf dem Bildschirm wird das Fenster aus Abb. 23, Seite 47 dargestellt. Die Überschrift ist hier **TreeNode**.

Die Darstellung kann durch Betätigen von **Quit** im Menü der Kopfzeile des Fensters beendet werden.

Falls mehr als ein Knoten selektiert wurde, besteht das Menü, das durch Betätigen der rechten Maustaste in der Zeichenfläche erhalten wird, nur noch aus zwei Einträgen (Selektion von mehreren Knoten s. unten).

- **remove nodes**

Mit **remove nodes** können alle selektierten Knoten gelöscht werden. Der Benutzer hat das Löschen jedes einzelnen Knotens zu bestätigen. Dementsprechend wird jeder Knoten einzeln gelöscht oder nicht gelöscht. Die Beschriftung des Fensters zur Bestätigung entspricht derjenigen des Fensters beim Löschen eines einzelnen Knotens (s. Seite 49).

- **inspect**

Mit **inspect** kann sich der Anwender hier die aktuellen Werte der Attribute der selektierten Gruppe von Knoten (Objekt der Klasse `TreeNodeGroup`) anschauen. Auf dem Bildschirm wird das Fenster aus Abb. 23, Seite 47 dargestellt. Die Überschrift ist hier `TreeNodeGroup`.

Mit Hilfe der linken Maustaste können folgende Bearbeitungen am Graphen und der Zeichenfläche vorgenommen werden:

- **Selektion eines Knotens oder einer Gruppe von Knoten**

Wenn sich der Mauszeiger über dem Symbol eines Knotens befindet, kann dieser durch Tastendruck selektiert werden. Zusätzlich können weitere Knoten selektiert werden, wenn die Taste `<Shift>` gedrückt wird (Knotengruppe). Beim Selektieren ändert sich die Ausprägung (3D-Darstellung) oder die Farbe (2D-Darstellung) der Umrandung des Knotensymbols. Falls die Symbole ohne Umrandung dargestellt sind (nur Text), erhalten sie beim Selektieren eine Umrandung.

Die Selektion eines einzelnen Knotens bewirkt abhängig von der Position des Mauszeigers relativ zum Knotensymbol und abhängig vom Zustand des Knotens weitere Effekte:

Befindet sich der Mauszeiger im linken Bereich des Symbols und ist der Knoten expandiert, so verschwindet die Darstellung aller Nachfolgeknoten und der Name des Knotens wird fett dargestellt. Ist der Knoten nicht expandiert, erfolgt eine reine Selektion.

Ist der Mauszeiger im rechten Bereich eines nicht expandierten Knotens, so wird dieser expandiert, aber nur soweit, wie er zuvor schon einmal expandiert war (zur vollständigen Expansion s. Seite 48: Menüeintrag **expand all**). Wenn der Knoten bereits entsprechend weit expandiert ist, erfolgt eine reine Selektion.

Wenn sich der Mauszeiger in der Mitte des Knotensymbols befindet erfolgt immer nur eine Selektion.

Mehrere Knoten gleichzeitig können auch durch Aufziehen eines Rechtecks um die betreffenden Symbole herum selektiert werden. Bei diesem Vorgang ist die linke Maustaste gedrückt zu halten. Falls der Anwender zuvor in einem der Menüs den Eintrag **zoom** (s. Seite 43) gewählt hat, erfolgt hier allerdings die Bestimmung des Zoomfaktors über die Größe des aufgezogenen Rechtecks und eine Selektion von Knoten findet nicht statt.

- **Verschieben von Knoten**

Selektierte Knoten (einer und mehrere) können mit Hilfe der Maus verschoben werden. Dazu ist der Mauszeiger über eines der selektierten Knotensymbole zu verschieben und die linke Maustaste zu drücken. Von allen selektierten Symbolen wird eine Art Geisterbild, d.h. eine zweite, andersfarbige Abbildung dargestellt. Indem die linke Maustaste weiterhin gedrückt wird, kann diese Abbildung über das Symbol eines beliebigen Knotens geschoben werden. Beim Loslassen der Taste erfolgt - nach Bestätigung durch den Anwender - die Verschiebung aller selektierten Knoten als Nachfolgeknoten zum ausgewählten Knoten. Die Beschriftung im Fenster zur Bestätigung lautet:

**Do you really want
to move the subtree [Name]?**

Dabei werden die gesamten Zweige, die an diesen Knoten beginnen, verschoben. Sie werden bei ihren alten Vorgängern gelöscht.

Bei jedem Vorgang, bei dem die linke Maustaste gedrückt und gleichzeitig der Mauszeiger verschoben wird, ändert sich dessen Darstellung.

2.3 Produktverwaltung & Statistik

Das Teilprogramm Produktverwaltung & Statistik von Elab befindet sich zur Zeit in der Entwicklung und kann noch nicht sinnvoll genutzt werden. Das Fenster, mit dem sich die Produktverwaltung durchführen läßt, ist in Abb. 24 dargestellt.

Es enthält ein Textfeld mit Rollbalken und mehrere Buttons mit allgemeiner Bedeutung wie die Fenster der anderen Programmteile auch.

Durch Auswahl des Buttons **Fragment Manager** erhält der Anwender ein Fenster zur Fragmentenverwaltung (Abb. 16, Seite 27).

Bei Betätigen des Buttons **Logs** erscheint das Auswahlfenster des Protokollgenerators (s. Kapitel 2.2.5).

Das Teilprogramm Produktverwaltung & Statistik wird bei Auswahl von **Close** verlassen und das Fenster geschlossen, nachdem der Benutzer diese Aktion bestätigt hat (**Really close Products & Statistics?**).

Der Button **DB save** hat derzeit keine Funktion.

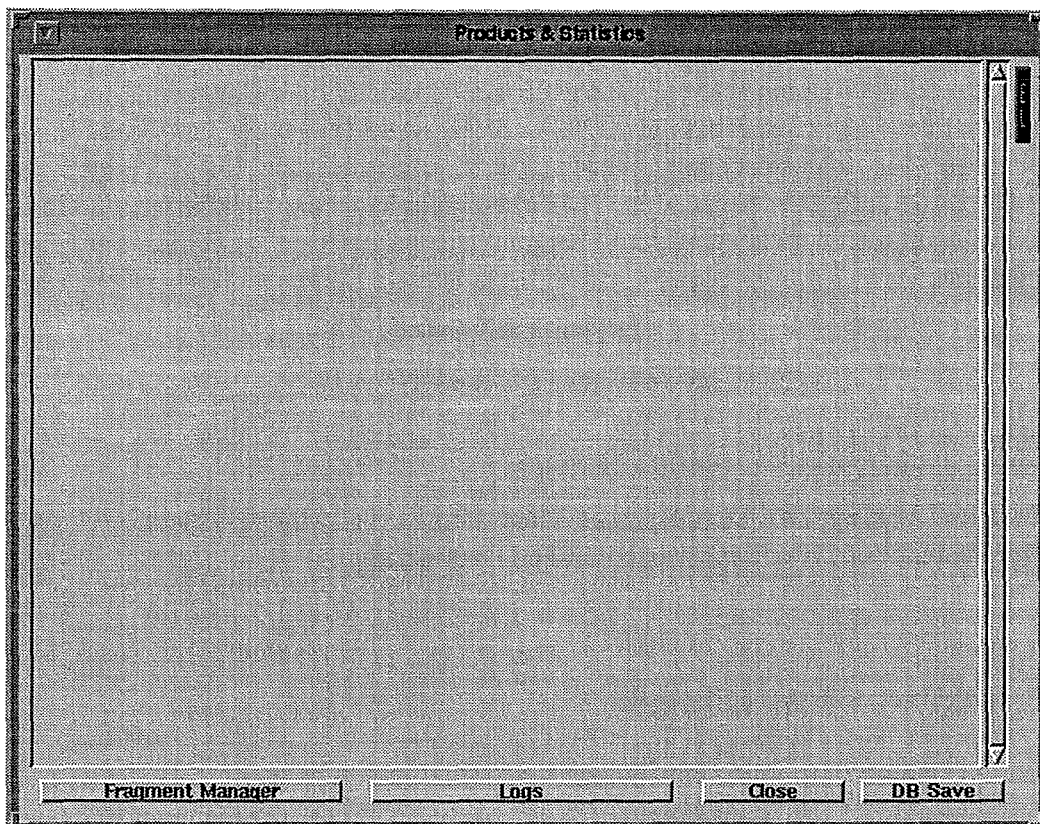


Abb. 24
Fenster für das Teilprogramm Produktverwaltung & Statistik
Das Programm ist zur Zeit noch nicht sinnvoll nutzbar.

3 Datenbankbindung

3.1 Das Datenbank-Management-System

Die Ausführungen im vorangegangenen Kapitel haben gezeigt, daß im Verlauf der Entwicklung eines Herstellungsverfahrens für Mikrosysteme Daten als Attribute und Parameter von Prozessen anfallen, deren Verwaltung und Darstellung Hauptaufgabe des Elektronischen Laborbuchs sind. Prozesse wiederum sind Bestandteile der komplexeren Datenstrukturen Fragment, Protokoll und Produkt.

Verschiedene Teams von Entwicklern benutzen das System, wobei i. allg. mehrere Benutzer gleichzeitig auf die Daten zugreifen wollen. Dabei muß der Zugriff nach bestimmten Zugriffsrechten und Gruppen- bzw. Projektzugehörigkeiten geregelt sein.

Mehrbenutzerbetrieb, Zugriffsberechtigungen aber auch Versionsverwaltung, die im Elektronischen Laborbuch bei der Erzeugung neuer Datenstrukturen als Kopien von bereits existierenden auftritt, lassen sich nur im Rahmen einer zentralen persistenten Datenhaltung realisieren.

Zu diesem Zweck war die Anbindung an ein Datenbank-Management-System erforderlich. Bei der Auswahl des geeigneten Systems stand die Objektorientierung des Elab-Systems und die Möglichkeit der Anbindung an Smalltalk im Vordergrund.

ObjectStore (Version 3.1) von der Firma ObjectDesign ist ein mehrbenutzerfähiges, objektorientiertes Client/Server-Datenbanksystem. Es unterstützt unter anderem die Sprachanbindung an Smalltalk (Version 1.0) [5].

Die ObjectStore-Datenbanken, die die persistenten Daten aufnehmen, werden jeweils von genau einem Server verwaltet. Die Übertragungseinheit zwischen persistentem Speicher und Programmspeicher ist eine *Seite*, die in der Regel mehrere Objekte enthält und 4 bis 8 kByte groß ist. Es können jedoch auch mehrere Seiten zusammengefasst werden, so daß diese größeren logischen Einheiten als Übertragungseinheit dienen (z.B. *Segment*).

Zum Betrieb einer ObjectStore-Datenbank sind neben der eigentlichen Applikation noch zwei Prozesse notwendig, die sich auf den einzelnen Rechnern befinden müssen. Auf der Serverseite ist dies der eigentliche Serverprozeß, der den Zugriff auf alle persistenten Daten übernimmt. Die Synchronisation der Zugriffe erfolgt mittels eines 2-Phasen-Sperrprotokolls [6]. Die Granularität der Sperren ist die jeweilige Seite innerhalb der sich ein Objekt befindet. Der Inhalt einer Datenbankseite ist für deren Bereitstellung durch den Serverprozeß nicht von Bedeutung.

Der zweite Prozeß, der zum Betrieb von ObjectStore notwendig ist, ist der Cache-Manager. Er muß sich auf dem Client-Rechner befinden und wird automatisch von der Applikation gestartet. Dieser Prozeß ist u.a. zuständig für die Hauptspeicherverwaltung des Clients. Laufen mehrere Applikationen auf einem Client, so teilen sie sich einen Cache-Manager. Dieser Sachverhalt ist in Abb. 25 verdeutlicht..

Für den Anwender werden die Daten transparent zwischen der Datenbank und dem Client-Cache übertragen. Diese Daten werden im virtuellen Speicher der Applikation abgebildet.

Nachdem eine Seite zum erstenmal angefordert und über das Netz übertragen wurde, kann der Zugriff auf die persistenten Daten dadurch ebenso schnell wie auf die transienten Daten des Prozesses erfolgen.

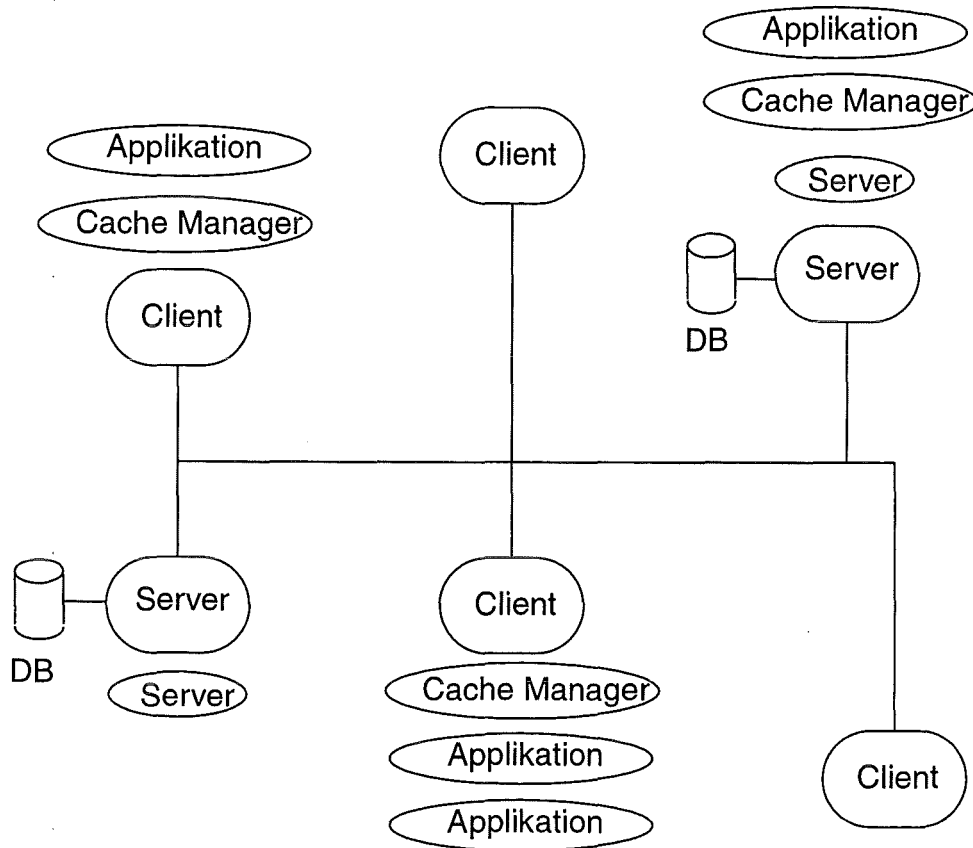


Abb. 25

ObjectStore als Client/Server-System [5]

Auf der Serverseite übernimmt ein Serverprozeß den Zugriff auf die persistenten Daten in den Datenbanken. Auf dem Client-Rechner befindet sich der Cache-Manager, der automatisch von der Applikation gestartet wird. Mehrere Applikationen teilen sich einen Cache-Manager. Dessen Aufgabe ist die Hauptspeicherverwaltung des Clients.

Bei der C++-Sprachanbindung [7] von ObjectStore muß für jedes Objekt explizit zum Erzeugungszeitpunkt angegeben werden, ob es in die Datenbank eingetragen werden soll oder ob es sich um ein transientes Objekt handelt. Die Smalltalk-Sprachanbindung von ObjectStore geht hier allerdings einen anderen Weg. Ausgehend von einem persistenten Objekt werden alle von diesem Objekt aus referenzierten Objekte in die Datenbank geschrieben. Dieser Mechanismus wird als Objekt-Migration bezeichnet. Wie diese *Objekt-Migration* realisiert ist wird in der Klassendefinition unter der *migrationPolicy* spezifiziert. Hierbei kann angegeben werden, wie am Ende einer Transaktion Referenzen von persistenten Objekten auf Objekte im transienten Speicher behandelt werden sollen. Standardmäßig wird ein referenziertes Objekt in den persistenten Speicher kopiert und die transiente Version wird gelöscht. Es ist jedoch auch möglich, daß eine Kopie des transienten Objekts erstellt und in die Datenbank geschrieben wird und das Objekt weiterhin erhalten bleibt oder daß das transiente Objekt durch eine Referenz (*OSReference*) auf das entsprechende Datenbankobjekt ersetzt wird. Dabei werden Nachrichten, die an eine transiente Instanz vom Typ *OSReference* geschickt werden, an das

persistente Objekt in der Datenbank weitergeleitet. Eine andere Migrationsart ist die, eine Referenz durch ein nicht definiertes Objekt (Klasse *UndefinedObject*/Objekt *nil* in Smalltalk) zu ersetzen. Es existieren noch weitere Migrationsarten, auf die hier nicht weiter eingegangen werden soll. Neben der Festlegung der *migrationPolicy* für eine komplette Klasse ist es auch möglich, für einzelne Variablen eine eigene Migrationsart zu definieren oder die Migrationsart für eine bestimmte Instanz einer Klasse zu setzen.

Informationen über die einzelnen Klassen müssen nicht explizit aus dem Metadatenprotokoll der Datenbank ausgelesen werden, da diese direkt in das Smalltalk Metaklassenprotokoll eingebunden sind und in gewohnter Weise auf die Metainformationen zugegriffen werden kann.

In der Version 1.0 von ObjectStore für Smalltalk werden im Gegensatz zur C++-Sprachanbindung keine geschachtelten Transaktionen und keine Versionshaltung unterstützt.

In den Datenbanken existieren spezielle Verzeichnisse vom Typ *OSDatabase*, die zum Eintrag von *Ankerobjekten* eingesetzt werden. Diese Ankerobjekte sind über einen eindeutigen Namen zugreifbar (*named objects*).

3.2 Wichtige Klassen von Elab

Dieser Abschnitt gibt einen Überblick über wichtige Klassen des Systems Elab, indem die dazugehörigen Konzepte kurz erläutert werden und über wesentliche Methoden der Klassen informiert wird.

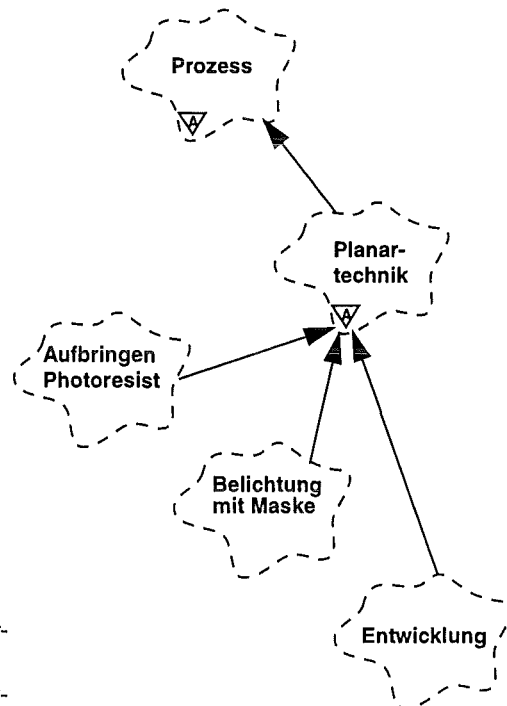


Abb. 26
Ausschnitt aus der Prozessklassenhierarchie
Prozess und *Planartechnik* sind abstrakte Klassen. Die Pfeile symbolisieren die Vererbung.

Prozess

Diese Klasse dient der Beschreibung von Einzelprozessen der Mikrosystemfertigung, die mit Hilfe des Systems kombiniert werden können. Sie enthält allgemeine Attribute, die unabhängig von speziellen Eigenschaften der Prozesse sind. Für die Implementierung spezieller Eigenschaften wurden Klassen von der Klasse *Prozess* als Oberklasse abgeleitet. Abb. 26 zeigt einen kleinen Ausschnitt aus der Vererbungshierarchie der Prozesse. Die Vererbung ist dabei durch Pfeile symbolisiert. Die Klassen *Prozess* und *Planartechnik* sind abstrakt, während es sich bei den übrigen dargestellten Klassen um konkrete Klassen handelt, von denen Instanzen erzeugt werden können. *Prozess* hat noch eine Reihe weiterer abstrakter Unterklassen; *Planartechnik* hat außer den dargestellten noch weitere abstrakte und konkrete Unterklassen.

Die Methoden der Klasse *Prozess* dienen in erster Linie dem Zugriff auf die allgemeinen Prozessparameter. Weitere wichtige Methoden sind für das Kopieren von Prozeßobjekten vorgesehen sowie für das Einleiten eines Editiervorgangs für Prozesse und Templates.

Prozeßeditoren

Zu jedem Prozeß eines Fragments kann der Benutzer einen Editor aufrufen (s. Abb. 18, Seite 31 und Abb. 19, Seite 32). Für alle Editoren existiert eine gemeinsame Oberklasse *ElabProcessEditor*, in der wesentliche Eigenschaften zusammengefaßt sind. Diese Klasse wurde im Rahmen der Arbeiten zur Datenbankankündigung des Systems eingeführt und ist im Anhang *Quellcodedokumentation* ausführlich beschrieben. Sie ist von einer ebenfalls dort beschriebenen Klasse *ElabApplicationModel* abgeleitet, die als Ergänzung zum MVC-Oberflächenmodell von Smalltalk eine speziell auf die Anforderungen von Elab zugeschnittene Möglichkeit schafft, Fenster als Abkömmlinge von anderen Fenstern zu erzeugen und zu verwalten.

Spezielle Eigenschaften von Prozeßeditoren sind in zusätzlichen Klassen untergebracht, die zum größten Teil von *ElabProcessEditor* abgeleitet wurden.

Fragmente

Ein *Fragment* ist eine Kette von aufeinanderfolgenden Prozeßschritten. Bei der Entwicklung eines Herstellungsprozesses sind die Fragmente gewissermaßen Testsequenzen, die solange modifiziert werden, bis der Prozeßentwickler eine brauchbare Lösung gefunden hat.

Objekte der Klasse *Fragment* enthalten die Prozesse in einer Liste. Ihre Reihenfolge entspricht dem zeitlichen Ablauf der einzelnen Prozeßschritte im Gesamtprozeß. Die wichtigsten Methoden dieser Klasse dienen zum Kopieren und zur Manipulation der Prozeßliste.

Protokolle

Ein Prozeßentwickler, der mit dem System Elab arbeitet, kann dabei durch sogen. Chargensplitting vom System unterstützt werden. Dabei können für Teile des Fragments Alternativen durchgespielt werden. D.h. ab einem bestimmten Prozeßschritt verzweigt sich die Prozeßfolge. In den einzelnen Zweigen können unterschiedliche Prozeßschritte oder die gleichen Prozeßtypen, jedoch mit unterschiedlichen Werten für deren Parameter eingefügt werden.

Die dadurch entstehenden Baumstrukturen werden *Protokolle* genannt. Abb. 27 zeigt eine dieser Baumstrukturen. Stärker umrandet ist das ursprüngliche Fragment dargestellt. Dieses Protokoll ist durch mehrfaches Verzweigen entstanden.

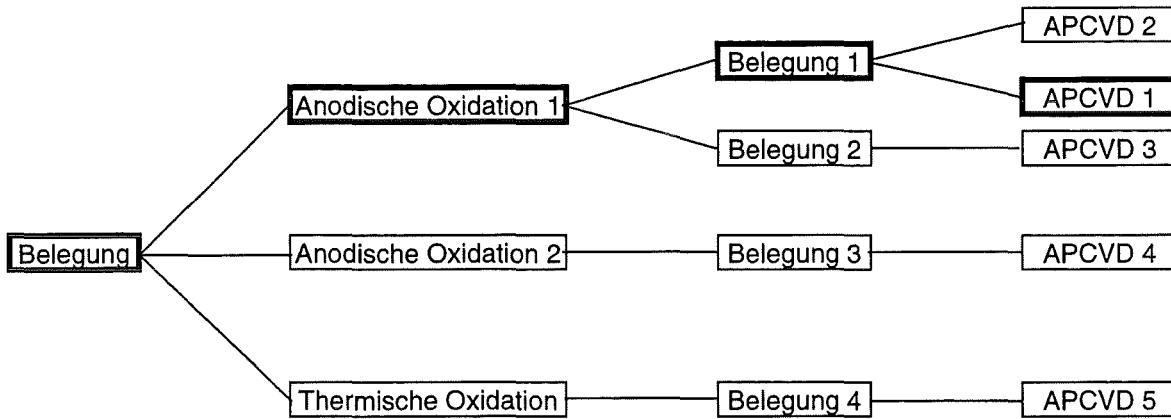


Abb. 27

Darstellung eines Protokolls

Die aus den fett umrandeten Knoten bestehende Prozeßfolge ist das ursprüngliche Fragment. Die Baumstruktur ist durch mehrfaches Chargensplitting zustande gekommen.

Protokolle wurden mit Hilfe der beiden Klassen *RrProtokoll* und *RrProtokollNode* implementiert. *RrProtokollNode* entspricht einem Prozeß und enthält selbst ein Objekt einer Prozeßklasse. Es dient zu Darstellungszwecken innerhalb einer Baumstruktur im Chargensplitting-Workfloweditor (s. Abb. 21, Seite 41).

Produkte

Die Einführung von Produkten ist erst im Laufe der weiteren Entwicklung des Systems Elab vorgesehen.

Ein *Produkt* ist eine spezielle Art von Fragment. Wenn ein Fragment soweit ausgetestet ist, daß seine Prozeßfolge und die Parameter der darin enthaltenen Prozesse feststehen, kann es als Produkt abgelegt werden. Dieses Produkt unterscheidet sich vom Fragment nur durch eine Reihe von Verwaltungsdaten. Eine Änderung der Prozesse oder der Prozeßfolge eines Produkts wird nicht mehr möglich sein. Es kann nur als Ganzes gelöscht werden.

Im Elektronischen Laborbuch ist ein Viewer vorgesehen, der ein Produkt für den Anwender darstellt (vgl. Abb. 24, Seite 52). Dieser Viewer soll die Prozeßliste und die Verwaltungsdaten, die noch editiert werden können, zeigen.

3.3 Datenbanken des Systems Elab

Für die Speicherung der Daten des Elektronischen Laborbuchs existieren drei Datenbanken:

- Templates (*templates*)

Für jede konkrete Prozeßklasse ist in dieser Datenbank eine Instanz als Vorlage unter dem jeweiligen Klassennamen abgelegt. Bei der Erzeugung von Instanzen einer bestimmten Prozeßklasse werden Kopien dieser Vorlagen verwendet.

Die Templates können editiert werden, um die Initialisierung von neuen Prozeßobjekten zu ändern (vgl. Kapitel 2.2.4, Seite 29).

In der Template-Datenbank werden die Prozeßinstanzen als Ankerobjekte verwendet.

- Fragmente (*fragments*)

Fragmente werden unter ihrem Namen als Ankerobjekte in einer eigenen Datenbank abgelegt. Die Prozesse eines Fragments können sowohl direkt durch Zugriff auf die Prozeßliste als auch über die Objekt-Migration beim Speichern des Fragments in die Datenbank geschrieben werden.

Ein Fragment enthält auch eine Liste mit den Namen von älteren Versionen, das sind Fragmente, aus denen das betrachtete Fragment durch Kopieren hervorgegangen ist.

- Protokolle (*logs*)

Für Protokolle gilt dasselbe wie für Fragmente. Protokolle und Fragmente sind in ihrem Aufbau ähnlich, außer daß Fragmente die Prozesse als lineare Liste, Protokolle dagegen als Baumstruktur enthalten.

- Eine weitere Datenbank ist für Produkte vorgesehen. Da Produkte im wesentlichen Fragmente mit einigen zusätzlichen Verwaltungsdaten sind, ist hier derselbe Aufbau vorgesehen wie für Fragmente oder Protokolle.

Somit haben Objekte der Klassen *Fragment*, *Prozess* und *Protokoll* gemeinsam, dass sie als persistente Objekte in Datenbanken vorkommen. Alle Eigenschaften, die diese Objekte wegen der Persistenz besitzen, wurden in einer gemeinsamen Oberklasse zusammengefaßt. Die Darstellung dieser Oberklasse ist u.a. Gegenstand des nächsten Kapitels.

4 Semantische Sperren

4.1 Die Notwendigkeit zusätzlicher Sperrmechanismen

In verteilten Applikation treten Probleme mit der Parallelität von Zugriffen auf den gemeinsamen Datenbestand auf. So ist es z.B. nicht tragbar, daß ein Programm ein Objekt ausliest und Modifikationen an diesem durchführt, da in der Zwischenzeit ein anderer Prozeß ebenfalls Modifikationen an diesem Objekt durchführen könnte. Zwischenergebnisse dürfen für einen anderen Prozeß nicht sichtbar sein, da sie einen inkonsistenten Objektzustand darstellen können.

Die Zugriffe von verschiedenen Prozessen auf einen Datenbestand müssen daher synchronisiert werden. Für diese Synchronisation wurden Mechanismen entwickelt, die unter dem Begriff *Transaktion* bekannt sind [6][8].

Eine Transaktion ist ein Programmabschnitt, der aus der Sicht der Anwendung unteilbar und isoliert abläuft. Sie wird entweder vollständig oder gar nicht ausgeführt. Zwischenergebnisse sind für andere Transaktionen nicht sichtbar. Ferner ist garantiert, daß die Ergebnisse nach erfolgreicher Beendigung der Transaktion persistent sind.

Während einer Transaktion werden die Daten, auf die zugegriffen wird, gesperrt. Kein anderer Prozeß kann für die Dauer der Transaktion auf diese Daten zugreifen.

ObjectStore führt hier jedoch keine objektbasierte Sperrung durch, sondern sperrt die komplette physikalische Seite innerhalb der sich das angeforderte Objekt befindet [5]. Es kann also vorkommen, daß eine Transaktion, die Änderungen an einem bestimmten Objekt vornimmt, einer anderen Transaktion verbietet, auf ein ganz anderes Objekt zuzugreifen, das sich zufällig auf dieser Seite befindet. In diesem Fall muß die zweite Transaktion solange warten, bis die erste Transaktion beendet ist. Insbesondere wenn viele kleinere Objekte gleichzeitig von mehreren Anwendern bearbeitet werden müssen, kann es zu unnötigen Verzögerungen kommen, aufgrund von Sperrkollisionen, die eigentlich keine sind. Dieser Zustand ist speziell bei Applikationen, die eine Benutzerinteraktion erfordern, nicht tragbar.

Im Elektronischen Laborbuch sind viele Benutzerinteraktionen erforderlich, bei denen Objekte über einen längeren Zeitraum hinweg gesperrt werden müssen (z.B. das Editieren von Prozessen). Problematisch sind hier nicht nur die o.g. unnötigen Sperrkollisionen sondern auch die Tatsache, daß eine Anwendung einfach anhält, bis eine Sperre auf einem gewünschten Objekt wieder aufgehoben wird.

Um diese Schwierigkeiten zu umgehen, wurden im System Elab *semantische Sperren* [9] als eigene Sperrverwaltung eingeführt. Eine semantische Sperre ist ein zusätzliches Datenfeld eines persistenten Objekts, das einen eigenen Sperrmechanismus realisiert. Beim Elektronischen Laborbuch wird hier ein Objekt einer Klasse *ElabLock* verwendet. Über dieses Objekt wird der Zugriff auf das persistente Objekt gesteuert. Dabei kann der Systementwickler die Granularität des Sperrrens selbst festlegen. Wünscht ein Prozeß Zugriff auf ein bestimmtes Datenbankobjekt, so öffnet er eine Transaktion und bestimmt zunächst den Zustand der semantischen Sperre des Objekts. Besteht eine unverträgliche Sperre so beendet er die Transaktion und ein Zugriff auf das Objekt ist nicht möglich. Kann das Objekt jedoch gesperrt wer-

den, so wird dies vom Prozeß veranlaßt und die Transaktion wird beendet. Von diesem Moment an hat der Prozeß das Objekt gesperrt und kein anderer Prozeß darf eine unverträgliche Sperre auf das Objekt setzen. Die Sperre behält der Prozeß so lange, bis er sie explizit wieder freigibt. Der Unterschied zur Synchronisation mittels ObjectStore-Transaktionen besteht darin, daß immer nur sehr kurze Transaktionen benutzt werden um auf die semantische Sperre zuzugreifen. Dies geschieht in extrem kurzer Zeit, so daß für den Anwender keine bemerkbaren Verzögerungen aufgrund von wartenden Transaktionen entstehen. Insbesondere existieren keine langen Transaktionen über Benutzeraktionen hinweg, z.B. während der kompletten Editierzeit eines Objekts, sondern das Objekt ist mittels seiner semantischen Sperre vor konkurrierendem Zugriff geschützt. Die Transaktion, die zum Zugriff auf die semantische Sperre benutzt wird, erfüllt zudem den Zweck, daß das Lesen und eventuelle Ändern der Sperre als atomare Aktion geschieht und so sichergestellt wird, daß nicht mehrere Prozesse gleichzeitig auf die Sperre zugreifen. Üblicherweise trägt man in das Feld nicht nur die Minimalinformation ein, die zum Sperren nötig ist, sondern fügt zusätzlich noch Informationen über den Benutzer bzw. den Prozeß mit hinzu. Da der Zugriff auf persistente Daten nur innerhalb von Transaktionen möglich ist, kopiert sich der Prozeß das gesperrte Objekt in seinen transienten Speicher und führt die Operationen an diesem Objekt aus. Er kann ja sicher sein, daß die Kopie stets dem Originalobjekt entspricht, da kein anderes Objekt eine unverträgliche Sperranforderung an das Objekt stellen kann. Hat der Prozeß seine Änderungen an dem Objekt abgeschlossen, öffnet er wieder eine Transaktion, kopiert das modifizierte Objekt in den persistenten Speicher und gibt die semantische Sperre wieder frei.

Ein Problem, das hierbei auftaucht, ist der Zustand, daß ein Prozeß ein oder mehrere Objekte solange sperrt, bis er diese explizit wieder freigibt. Tritt nun der Fall ein, daß ein Prozeß eine Anzahl von Objekten gesperrt hat und anschließend nicht mehr in der Lage ist, diese Sperren wieder freizugeben, so sind diese Objekte möglicherweise für immer gesperrt. Gründe für die Verhinderung der Freigabe wären etwa ein Netzausfall oder ein Systemabsturz. Eine Möglichkeit wäre, die gesperrten Objekte durch einen speziellen Prozeß wieder freizugeben. Dies hat jedoch den Nachteil, daß dieser Aufräumprozeß explizit gestartet werden muß und dies durch einen Anwender mit besonderen Privilegien (Superuser, Administrator) zu geschehen hat, weil dieser Prozeß Sperren von fremden Benutzern oder Prozessen entfernen muß. Auch kann dieser Prozeß je nach Größe der Datenbank sehr zeitaufwendig sein, weil, sofern nicht explizit auf persistentem Speicher mitprotokolliert, nicht gesagt werden kann, welche Objekte von dem gescheiterten Prozeß gesetzt wurden und so die komplette Datenbank durchsucht werden muß.

Eine elegantere Möglichkeit bietet hier der Einsatz von *Zeitstempeln* für die jeweiligen Sperren. Bei dieser Variante wird neben der eigentlichen Sperre auch noch ein Datum abgespeichert, das die Gültigkeitsdauer der Sperre angibt. Ist die Zeitspanne abgelaufen so wird die Sperre für ungültig erklärt und darf von anderen Prozessen überschrieben werden. Da aber i. allg. im Voraus nicht gesagt werden kann, wie lange ein Objekt gesperrt werden soll, gibt es für den sperrenden Prozeß die Möglichkeit, eine bestehende Sperre zu verlängern. Dies garantiert auf der einen Seite, daß ein Prozeß sichergehen kann, daß seine Sperre beim Einbringen der Änderungen noch besteht und andererseits bei einem unvorhergesehenen Ende eines Prozesses auf Dauer keine Sperren zurückbleiben. Der Vorteil dieses Verfahrens liegt darin, daß keine zusätzlichen Programme benötigt werden sondern der ganze Aufräummechanismus in der Sperrverwaltung bereits integriert ist. Dies ist besonders bei räumlich verteilten Anwendungen, die über ein möglicherweise unzuverlässiges Netzwerk, kommunizieren, von großem Vorteil. Der Nachteil liegt in der aufwendigeren Realisierung der Sperrverwaltung.

Das Elektronische Laborbuch bietet nach Einführung dieses Konzeptes für die Sperrverwaltung nun die Möglichkeit, über die Funktionalität von ObjectStore hinaus, einzelne Datenobjekte zu sperren. Wenn einem Benutzer dann der Zugriff auf ein Objekt verwehrt wird, dann geschieht das genau deshalb, weil es tatsächlich gerade von einem anderen bearbeitet wird.

4.2 Klassen für Sperrverwaltung und Datenbankbindung

Das zusätzliche Objekt eines persistenten Objekts, das die Sperre repräsentiert, enthält nicht nur die zum Sperren notwendigen Minimalinformationen sondern auch Informationen über den Benutzer und den Prozeß (das sperrende Programm). Es ist eine Instanz der Klasse *ElabLock*.

Die Klasse besitzt als Attribute einen Zeitstempel, eine Gültigkeitsdauer, einen eindeutigen Bezeichner, um herauszufinden zu können, wer die Sperre gesetzt hat, sowie ein Symbol, das den Zustand der Sperre beschreibt. Einen Auszug aus dem Instanzenprotokoll zeigt Abb. 28.

Instanzenprotokoll Klasse <i>ElabLock</i>	
Folgende Methoden modifizieren die betreffende Sperre	
<hr/>	
getUpdatePeriod	Liefert die Dauer einer Aktualisierungsperiode.
markInvalid	Markiert die Sperre explizit als ungültig.
markReleased	Markiert die Sperre als freigegeben.
markValid	Markiert die Sperre explizit als gültig.
setInitialTimeStamp	Setzt den allerersten Zeitstempel.
updateTimeStamp	Verlängert die Gültigkeitsdauer
Diese Methoden informieren über den Zustand der betreffenden Sperre	
<hr/>	
isExpired	Überprüft, ob der Zeitstempel abgelaufen ist.
isNilLock	Ein NilLock ist sozusagen das Nullelement unter den Sperren. Die Methode gibt an, ob ein solches Nullelement vorliegt.
isValid	Überprüft, ob der Zeitstempel noch gültig ist.

Abb. 28

Auszug aus dem Instanzenprotokoll der Klasse *ElabLock*

Wesentliche Funktionen sind die Modifikation und die Überprüfung von Sperren.

Beim Setzen einer Sperre wird von der Anwendung zusätzlich ein Prozeß (Thread) [10] gestartet, der für die Aktualisierung des Zeitstempels in bestimmten Intervallabständen zuständig ist. Dieser Prozeß ist von der Anwendung abhängig, so daß eine Aktualisierung des Zeitstempels unterbleibt, wenn die Anwendung nicht mehr aktiv ist.

Realisiert wird ein Aktualisierungsprozess als einzige Instanzvariable einer Klasse *ElabLockDaemon*. Der Prozeß wird bei der Installation der Sperre von einer Klassenmethode von *ElabLockDaemon* erzeugt. Er terminiert in der Regel von selbst, wenn die Gültigkeitsdauer der Sperre nicht mehr verlängert wird. Für das explizite Terminieren existiert eine Instanzenmethode. Außerdem terminiert der Prozeß bei einem Systemabsturz oder ähnlich einschneidenden Ereignissen. Dadurch kommt der erwünschte Effekt zustande, daß die beaufsichtigte Sperre nicht mehr aktualisiert und beim nächsten Zugriff auf das gesperrte persistente Objekt entfernt wird.

Die überwachte Sperre selbst ist nur als lokale Variable dem Aktualisierungsprozess bekannt. Im System Elab müssen folgende Klassen mit semantischen Sperren versehen werden können:

- *Fragment*
- *Protokoll*
- *Prozess*
- *User* (s. Kapitel 5)

Instanzenprotokoll Klasse <i>DBObject</i>	
Folgende Methoden liefern entweder true oder false zurück	
<hr/> isLocked	Gibt an ob das Objekt mit einer Sperre belegt ist.
isReadLocked	Gibt an ob das Objekt mit einer Lesesperre belegt ist.
isWriteLocked	Gibt an ob das Objekt mit einer Schreibsperre belegt ist.
Diese Methoden sind für das Setzen und Löschen von Sperren zuständig	
<hr/> setWriteLock: aLock	Setzt, falls keine Sperrenunverträglichkeit besteht, eine Schreibsperre auf das Objekt. Rückgabewerte: (#write nil)
setReadLock: aLock	Setzt, falls keine Sperrenunverträglichkeit besteht, eine Lesesperre auf das Objekt. Rückgabewerte: (#read nil)
setBestLock: aLock	Versucht, die bestmögliche verträgliche Sperre auf ein Objekt zu setzen. Rückgabewerte: (#write #read nil)
releaseLock: aLock	Gibt eine gesetzte Sperre wieder frei. Rückgabewerte: (Instanz nil)

Abb. 29

Auszug aus dem Instanzenprotokoll der Klasse *DBObject*

DBObject ist die Oberklasse für alle Elab-Klassen, auf die semantische Sperren gesetzt werden müssen.

Weiter wurde zur Realisierung der semantischen Sperren - und auch zur Zusammenfassung anderer Eigenschaften persistenter Objekte - eine Klasse *DBObject* eingeführt, von der die hier genannten Klassen erben. Diese Klasse besitzt eine Liste mit Sperren für lesenden Zugriff (*Read Locks*), von denen beliebig viele gesetzt werden können sowie ein Feld für einen Schreibschutz (*Write Lock*), der nur einmal gesetzt werden darf und auch keine gleichzeitigen Lesesperren erlaubt. Abb. 29 zeigt einen Auszug aus dem Instanzenprotokoll der Klasse *DBObject*.

Erweitertes Instanzenprotokoll Klasse <i>DBObject</i>	
Folgende Methoden realisieren den Checkin/Checkout-Mechanismus	
<hr/>	
transientCopy	Liefert eine transiente Kopie des Empfängers der Nachricht zurück. Die Methode 'merkt' sich das Ursprungsobjekt, so daß mittels der commit/abort -Methoden, die dieses Objekt versteht, die Änderungen zurückgeschrieben und die Sperren zurückgesetzt werden können.
commit	Schreibt den Empfänger, bei dem es sich um ein mittels der Methode transientCopy erzeugtes Objekt handeln muß, zurück in die Datenbank und gibt die Sperre wieder frei.
abort	Wie bei commit, nur wird das transiente Objekt nicht zurückgeschrieben.

Abb. 30

Erweiterung des Instanzenprotokolls von *DBObject* zur Realisierung des im Text beschriebenen Checkin/Checkout-Mechanismus

Basierend auf dem Instanzenprotokoll aus Abb. 29 wurde eine weitere, höhere Ebene von Methoden entwickelt, die es dem Anwendungsprogrammierer mit einem minimalen Aufwand erlauben, persistente Objekte zu bearbeiten. Diese Methoden realisieren einen einfachen *Checkin/Checkout*-Mechanismus [11]. Der Checkout wird durch die Methode **transientCopy** realisiert. Diese Methode liefert eine komplette Kopie des zu bearbeitenden persistenten Objekts. Alle weiteren Operationen werden auf der transienten Kopie ausgeführt. Da die Kopie ihr Ursprungsobjekt in der Datenbank kennt, muß der Anwendungsprogrammierer keine Referenzen auf die Datenbank mitführen. Soll ein geändertes transientes Objekt in die Datenbank zurückgeschrieben werden, so wird diesem Objekt die Nachricht **commit** gesendet. Die transiente Kopie überprüft daraufhin, ob die von ihm vor dem Checkout gesetzte Sperre noch Bestand hat und schreibt im positiven Falle das geänderte Objekt zurück in die Datenbank und gibt die gesetzte Sperre wieder frei. Handelte es sich bei der Sperre nur um eine Lesesperre, so wird diese überprüft und der Anwender ggf. über eine verfallene Sperre informiert. Zurückgeschrieben werden kann in diesem Fall nicht. Wird der transienten Kopie die Nachricht **abort** geschickt, so bedeutet dies ebenfalls die Zurücknahme der Sperre, ohne

daß Ergebnisse zurückgeschrieben werden. Das Instanzenprotokoll für den Checkin/Checkout-Mechanismus ist in Abb. 30 zu sehen.

Die Methode *transientCopy* setzt keine Sperre auf das betreffende Objekt. Diese muß zuvor mittels den Methoden *setReadLock:*, *setWritelock:* bzw. *setBestLock:* explizit durchgeführt werden.

Neben der oben beschriebenen Zugriffskontrolle zur Synchronisation existiert noch eine weitere, benutzerbezogene Zugriffskontrolle. Diese spielt Hand in Hand mit der in Kapitel 5 beschriebenen Klasse *User*.

Jeder Benutzer gehört einem oder mehreren Projekten (Gruppen) an, innerhalb denen er mitarbeitet. Er darf nur auf Objekte zugreifen, die zu diesen Projekten gehören. Die Gruppenzugehörigkeit eines Objekts wie etwa ein Fragment oder ein Protokoll ist in einer Klasse *ObjectWithHistory* geregelt. Diese Klasse ist Oberklasse für *Fragment* und *Protokoll* und selbst von *DBObject* abgeleitet, weil es sich um in der Regel persistente Objekte handelt, die daraus instanziiert werden. Der Name der Klasse resultiert daher, daß für Fragmente und Protokolle auch eine einfache Versionsverwaltung im Rahmen dieser Klasse implementiert wurde. Eine Versionsverwaltung wird allgemein für alle Objekte, die auch Projektgruppen zugeordnet werden, notwendig sein.

Eine ausführliche Darstellung der Klassen *ElabLock*, *ElabLockDaemon*, *DBObject* und *ObjectWithHistory* ist im Anhang *Quellcodedokumentation* zu finden.

5 Mehrbenutzerbetrieb

In ObjectStore (Version 3.1) gibt es keine benutzer- und datenbezogenen Sicherheitsmechanismen. Diese müssen daher vom Anwender selbst realisiert werden. Außerdem existieren keine Mechanismen zur administrativen Verwaltung von Benutzern auf der Datenbankseite.

Es war somit notwendig für den Mehrbenutzerbetrieb von Elab ein Konzept für die Benutzerverwaltung zu erstellen und zu implementieren. Eine Klasse *User* steht im Mittelpunkt dieser Konzeption. Über ein Paßwort kann sich der Benutzer beim System anmelden. Er erhält dann über die Benutzeroberfläche von Elab einen Zugang zu den Datenbanken für Templates, Fragmente und Protokolle.

In der Regel wird jeder Benutzer, sobald er Zugang zum System erhält, sowohl auf Fragmente und Protokolle als auch auf die darin enthaltenen Prozesse voll zugreifen können, d.h. er darf lesen und schreiben. Einschränkungen gibt es in zwei Bereichen:

- Die Verwaltung von Benutzerdaten liegt in den Händen eines Superusers. Dieser kann neue Benutzer erzeugen, bestehende Benutzer löschen und Änderungen an den Daten bestehender Benutzer vornehmen. Der normale Benutzer kann sich hier nur einen Überblick über die momentan angemeldeten Benutzer verschaffen.
- Templates dürfen im Rahmen der Prozeßinitialisierung nur von Superusern manipuliert werden. Alle anderen Benutzer erhalten lesenden Zugriff, wenn sie ein Template bei der Erzeugung eines neuen Prozeßobjekts kopieren wollen.

Das Elektronische Laborbuch Elab wird für die Verwaltung von Prozeßdaten aus verschiedenen *Projekten* benutzt. Dabei ist es sehr wichtig, daß immer nur bestimmte Personen, die einer *Projektgruppe* angehören, auf Fragmente und Protokolle dieses Projekts zugreifen können.

Im Folgenden werden die Klasse *User*, der Mechanismus des Anmeldens, die Einrichtung verschiedener Projektgruppen und die Rolle des Benutzers beim Setzen und Entfernen semantischer Sperren näher beschrieben:

Die Klasse *User* besitzt folgende Attribute:

- Name

Der Name wird genau einmal gesetzt, nämlich bei der Erzeugung des *User*-Objekts durch einen Superuser. Der Name kann später nicht mehr geändert werden. Falls eine Namensänderung unumgänglich sein sollte, muß der Superuser das alte *User*-Objekt löschen und ein neues Objekt mit dem neuen Namen erzeugen.

- Paßwort

Jeder Benutzer besitzt ein Paßwort, das vom Superuser bei der Erzeugung des *User*-Objekts zum ersten Mal gesetzt wird und danach nur vom Benutzer selbst geändert werden darf. Das Paßwort wird nur in codierter Form gespeichert, so daß es in keiner Weise zugänglich ist.

- Projektgruppen

Jeder Benutzer kann einer oder mehreren Projektgruppen angehören. Die Namen der Projektgruppen sind beim Benutzer gespeichert. Eine spezielle Gruppe ist für Superuser eingerichtet. Ein Superuser ist für die Benutzerverwaltung zuständig und darf Templates ändern.

- Zugriffsrecht

Mit diesem Attribut wird der Superuser vom normalen Benutzer unterschieden.

- Benutzernummer (Lognummer)

Bei der Anmeldung eines Benutzers erhält dieser vom System eine Nummer zugeteilt, mit der seine Anmeldung eindeutig von anderen Anmeldungen unterschieden wird.

Abb. 31 zeigt Auszüge aus dem Instanzen- und aus dem Klassenprotokoll der Klasse *User*. Eine ausführliche Darstellung dieser Klasse ist im Anhang *Quellcodedokumentation* zu finden.

Sämtliche *User*-Objekte sind in einer eigenen Datenbank (*users*) gespeichert. Beim Anmelden und im Rahmen der Benutzerverwaltung wird auf diese Daten zugegriffen und ggf. eine Kopie des gewünschten Objekts angelegt, so daß die Benutzerdaten innerhalb der Applikation jederzeit zur Verfügung stehen ohne daß ständig auf die Benutzerdatenbank zugegriffen werden muß.

Der Zugang zu einem *User*-Objekt innerhalb der Nutzeroberfläche von Elab ist in Abb. 9, Seite 18 dargestellt.

Ein Benutzer hat vom Hauptfenster der Applikation (Abb. 8, Seite 17) aus verschiedene Zugriffsmöglichkeiten auf Benutzerdaten. Hier kann der normale Benutzer sein Paßwort ändern und sich die eigenen Daten bzw. die Daten von anderen, gleichzeitig angemeldeten Benutzern anzeigen lassen. Ein Superuser kann hier Benutzerdaten editieren, *User*-Objekte erzeugen oder löschen.

Sobald sich ein Benutzer beim System anmeldet und nach der Eingabe des Paßworts Zugang erhält, wird dieser *Login* vom System verwaltet und gegenüber allen anderen Logins, auch vom selben Benutzer, abgegrenzt. Daten über diese Logins stehen für jede aktive Elab-Anwendung in einer eigenen Datenbank (*login*) zur Verfügung. Eine eindeutige Kennzeichnung eines bestimmten Logins ist wegen der Verwaltung von semantischen Sperren von Datenbankobjekten nötig.

Fragmente und Protokolle gehören immer zu einem bestimmten *Projekt*. Beim Zugriff auf Fragmente oder Protokolle sind für einen bestimmten Benutzer nur diejenigen Objekte zugänglich, die zu einem Projekt gehören, dem auch der betreffende Benutzer angehört.

Für semantische Sperren von Datenbankobjekten werden außer einer eindeutigen Kennung des betreffenden Logins auch Daten des betreffenden Benutzers herangezogen. In der Regel ist das *User*-Objekt auch beim Entfernen der Sperre in ähnlicher Weise beteiligt. Ausnahme: Das Entfernen von Sperren wegen abgelaufener Zeitstempel.

Instanzenprotokoll Klasse <i>User</i>	
Die folgenden Methoden sind beim An- und Abmelden des Benutzers beteiligt	
getLoginFrom: aDB	Anmelden des Benutzers beim System.
removeLoginFrom: aDB	Abmelden des Benutzers.
Diese Methoden betreffen Veränderungen am Paßwort	
changePassword	Ändern des Paßworts.
setPassword	Setzen des Paßworts.
Folgende Methoden verwalten die Projektgruppenzugehörigkeit	
addGroup: aName	Hinzufügen einer neuen Projektgruppe.
firstGroup	Rückgabe der ersten Projektgruppe in der Liste.
removeGroup: aName	Entfernen der angegebenen Projektgruppe..
showGroups	Anzeigen der Liste der Projektgruppen.
singleGroup	Falls der benutzer nur zu einer einzigen Projektgruppe gehört, wird diese zurückgegeben.
Diese Methoden liefern Informationen über den betreffenden Benutzer.	
areYou: aName	Vergleicht den Namen des betreffenden Benutzers mit dem vorgegebenen Namen.
checkPassword	Die Abfrage des Paßworts wird veranlaßt.
includesGroup: aName	Überprüft, ob der betreffende Benutzer zu der angegebenen Projektgruppe gehört.
isLogged in	Testet, ob der betreffende Benutzer beim System angemeldet ist.
Klassenprotokoll Klasse <i>User</i>	
currentlyLoggedIn	Die Namen der Benutzer, die gerade beim System angemeldet sind, werden ermittelt.

Abb. 31

Auszug aus dem Instanzenprotokoll der Klasse *ElabLock*

Wesentliche Funktionen sind die Modifikation und die Überprüfung von Sperren.

6 Zusammenfassung

Das Elektronische Laborbuch Elab ist ein Softwarewerkzeug zur Unterstützung der Arbeit der Entwickler von Herstellungsprozessen von Mikrosystemen. Es dient der Darstellung und dem Modifizieren von Prozessen und Prozeßreihen. Dabei wird auch die Variantenbildung unterstützt.

Mehrbenutzerbetrieb, Zugriffsberechtigungen und Versionsverwaltung machen die Anbindung an ein Datenbank-Management-System erforderlich. Da Elab objektorientiert aufgebaut und in der Sprache Smalltalk implementiert ist, wurde das Datenbanksystem ObjectStore mit Sprachanbindung zu Smalltalk ausgewählt.

Das Elektronische Laborbuch kennt verschiedene Typen von Objekten, aus denen komplette Herstellungsprozesse für Mikrosysteme zusammengesetzt werden. Neben Einzelprozessen existieren Fragmente, das sind Prozeßreihen, die zu Testzwecken zusammengestellt werden können. Durch Chargensplitting entstehen aus Fragmenten die sogenannten Protokolle, Baumstrukturen, bei denen die einzelnen Zweige alternative Prozeßreihen darstellen. Dabei können außer den Prozeßparametern auch Typ und Anzahl der Prozeßschritte geändert werden.

Im Teilsystem Fragmentenverwaltung können Fragmente erzeugt und bearbeitet werden. Neue Prozesse werden als Kopien von Templates angelegt. Templates sind Prozeßobjekte, die als Vorlagen für neue Prozeßschritte dienen. Sie können innerhalb der Fragmentenverwaltung editiert werden, um die Werte für die Initialisierung der neuen Prozeßobjekte zu ändern. Jeder einzelne Prozeß innerhalb eines Fragments kann individuell editiert werden.

Zum Editieren von Prozessen und Templates enthält Elab verschiedene Prozeßeditoren, die sich durch Eingabefelder für verschiedene Parameter unterscheiden. Dabei ist ein Teil der Parameter für alle Prozeßtypen gleich.

Protokolle können in speziellen Chargensplitting-Workfloweditoren modifiziert werden. Auch hier kann zu jedem Prozeßschritt ein Prozeßeditor aufgerufen werden. Aus Teilen der Protokolle können wieder neue Fragmente erzeugt werden.

Auf Fragmente, Protokolle und Templates können verschiedene Systembenutzer zugreifen. Darum müssen diese Objekte persistent in Datenbanken gespeichert werden. Die entsprechenden Klassen *Fragment*, *RrProtokoll* und *Prozess* sind Unterklassen einer Klasse *DBObject* für die Datenbankfunktionalität. Fragmente und Protokolle sind dabei zunächst von der Klasse *ObjectWithHistory* abgeleitet, die für Versionshaltung und die Zugehörigkeit zu einer bestimmten Projektgruppe zuständig ist.

Für unterschiedliche Prozeßtypen existiert eine einfache Klassenhierarchie, in der Typen mit besonderen Parametern jeweils von einer Oberklasse abgeleitet werden, die alle gemeinsamen Parameter beinhaltet.

Der parallele Zugriff von verschiedenen Applikationen aus auf den zentralen Datenbestand erfordert spezielle Sperrmechanismen. Durch die Implementierung von semantischen Sperren kann objektweises Sperren erreicht werden. Es gibt Objekte, auf die die Benutzer über einen längeren Zeitraum hinweg zugreifen, z.B. auf Prozeßobjekte zum Editieren. Hier wird eine Kopie aus der Datenbank geholt, wofür nur eine kurze Transaktion erforderlich ist. Das Originalobjekt wird für die Zeitdauer der Benutzung mit einer semantischen Sperre belegt. Nach-

dem der Benutzer seine Aktion beendet hat, wird die Kopie in die Datenbank zurückgeschrieben und die Sperre freigegeben.

Die Funktionalität zum Einrichten und Verwalten von semantischen Sperren ist Bestandteil der Klasse *DBObject*. Semantische Sperren selbst sind als eigenständige Klasse des Elab-Systems implementiert. Hinzu kommt eine Klasse, die einen Rechnerprozess für das Aktualisieren von Sperren beinhaltet.

Im Mehrbenutzerbetrieb gehören die Benutzer bestimmten Projektgruppen an, die nur Zugriff auf die Daten ihres Projekts haben. Ein Superuser übernimmt die Aufgaben der Benutzerverwaltung. Es existieren Benutzerobjekte, die in einer eigenen Datenbank gespeichert sind. Dabei handelt es sich um Objekte einer Klasse *User*.

Die Benutzer können im Hauptfenster des Elektronischen Laborbuchs Daten von sich oder anderen Benutzern abrufen und ihr eigenes Paßwort ändern. Superusern stehen im Rahmen der Benutzerverwaltung noch weitere Funktionen zur Verfügung. Sie können Benutzerobjekte editieren und Benutzer hinzufügen bzw. entfernen.

Im folgenden Anhang *Quellcodedokumentation* sind alle Klassen, die im Rahmen der Datenbankankbindung implementiert wurden ausführlich beschrieben.

Anhang: Quellcodedokumentation

Dieser Anhang beschreibt die Klassen des Systems Elab und ihre Methoden, soweit sie im Rahmen der Datenbankanbindung des Systems implementiert wurden. Die Klassen sind eingeteilt nach den Klassenkategorien, so wie sie von VisualWorks verwaltet werden.

Zu jeder Klasse gibt es eine Kurzbeschreibung, die Oberklasse wird genannt und jede Instanz- und Klassenvariable wird erläutert. Anschließend erfolgt die Kurzbeschreibung jeder einzelnen Methode, ebenfalls unterteilt nach den in VisualWorks angelegten Methoden. Dabei werden zuerst die Instanzenmethoden und dann die Klassenmethoden (**class**) beschrieben. Bei Methoden mit Schlüsselwörtern werden die anzugebenden Argumente erläutert.

Die Programmierung von Elab erfolgte mit VisualWorks V 2.0. Die Standard-Smalltalkklassen sind von ObjectStore für Smalltalk V 1.0 modifiziert. Ferner wurde NEDT-Software für die Darstellung und Verwaltung von Graphen verwendet (NEDT V 2.1 der Ars Nova Software GmbH).

Außer diesen neu implementierten Klassen wurden auch bestehende Klassen des Elektronischen Laborbuchs für die Anbindung an die Datenbank geändert, insbesondere Klassen wie *Fragment*, *Prozess*, *RrProtokoll*. Für eine Beschreibung dieser Klassen sowie aller übrigen Klassen des Systems kann der Quellcode-Dokumentation *Elektronisches Laborbuch Elab - Software-Dokumentation* entnommen werden.

Kategorie Elab-Additions

Klasse DBObject

Diese Klasse ist Oberklasse für alle Klassen, die Datenbankobjekte beschreiben. Sie stellt Methoden für die Verwaltung von Sperren und für den Checkin/Checkout-Mechanismus zur Verfügung.

Oberklasse:
Object

Instanzvariablen:

dbReference:
Referenz auf ein Datenbankobjekt. Sie wird bei transienten Objekten gesetzt, wenn sie als Kopie eines Datenbankobjekts erzeugt werden.

readLocks:
Eine Datenstruktur der Klasse Bag zur Aufnahme von Lesesperren.

writeLock:
Eine Schreibsperre.

Kategorie accessing

dbReference

Zugriff auf Datenbankreferenz

Die Datenbank ist bei einer transienten Kopie gesetzt und referenziert das entsprechende Objekt in der ObjectStore-Datenbank. Diese Referenz wird zurückgegeben.

dbReference: aDBObject

Setzen der Datenbankreferenz

Die Datenbank ist bei einer transienten Kopie gesetzt und referenziert das entsprechende Objekt in der ObjectStore-Datenbank. Diese Referenz wird auf aDBObject gesetzt und zurückgegeben.

aDBObject:

Zu referenzierendes Datenbank-Objekt

getValidReadLocks

Zugriff auf gültige Lesesperren

Aus allen Lesesperren werden diejenigen herausgesucht, die noch gültig sind und deren Zeitstempel nicht abgelaufen ist. Die übrigen werden als ungültig markiert und aus der Datenstruktur für Lesesperren entfernt. Die Datenstruktur für Lesesperren wird zurückgegeben.

getValidWriteLock

Zugriff auf gültige Schreibsperre

Falls es sich bei der Schreibsperre nicht bereits um ein Nullelement handelt, wird geprüft, ob der Zeitstempel abgelaufen oder ungültig ist. Ist dies der Fall, wird die Sperre als ungültig markiert und auf eine neu erzeugte Sperre gesetzt.

Die Schreibsperre wird zurückgegeben.

readLocks

Zugriff auf die Read-Locks

Die Instanzvariable readLocks (Datenstruktur Bag) wird zurückgegeben.

readLocks: aBag

Neue Datenstruktur für Read-Locks

Die Instanzvariable readLocks wird auf aBag gesetzt und zurückgegeben.

aBag: neue Datenstruktur

writeLock

Zugriff auf den Write-Lock

Die Instanzvariable writeLock (String) wird zurückgegeben.

writeLock: aString

Ändern des Write-Locks

Die Instanzvariable writeLock wird auf aString gesetzt und zurückgegeben.

aString:
neuer Write-Lock

Kategorie checkIn-checkOut

abort: aString

Beenden einer Verbindung zwischen transientem und Datenbank-Objekt

Innerhalb einer Transaktion wird zunächst geprüft, ob eine Datenbankreferenz vorhanden ist. Ist dies nicht der Fall, wird die Transaktion wieder abgebrochen.

Ansonsten wird die Sperre beim Datenbank-Objekt freigegeben.

Bei Abbruch der Transaktion erfolgt eine Fehlermeldung, die nur für den Entwickler gedacht ist, da abort: nur benutzt werden sollte, wenn eine Verbindung zu einem Datenbank-Objekt besteht. Bei erfolgreicher Transaktion wird die Verbindung zum Datenbank-Objekt entfernt.

aString:
Bezeichner der Sperre beim Datenbank-Objekt

commit: aString

Zurückschreiben eines Objekts in die Datenbank

Innerhalb einer Transaktion wird zunächst geprüft, ob eine Datenbankreferenz vorhanden ist. Ist dies nicht der Fall, wird die Transaktion wieder abgebrochen.

Falls das Datenbank-Objekt nicht schreibgeschützt ist, wird eine evtl. vorhandene Lesesperre freigegeben. Falls auch keine Lesesperre mit dem vorgegebenen Bezeichner vorlag, wird die Transaktion ebenfalls abgebrochen.

Bei Vorliegen einer Schreibsperre - nur dann ist ein commit: eigentlich sinnvoll - wird eine Kopie des vorliegenden Objekts angelegt und dessen Datenbankreferenz gelöscht. Die Sperre des Datenbankobjekts wird freigegeben und die Kopie als neues Datenbank-Objekt übernommen.

Die Fälle, in denen die Transaktion abgebrochen wird, sind Fälle, in denen commit: nicht verwendet werden sollte. Daher sind die Fehlermeldungen nur als Hinweise für den Entwickler gedacht. Bei Abbruch der Transaktion erfolgt die Rückgabe von nil.

Nach einer erfolgreichen Transaktion wird auch die Datenbankreferenz beim vorliegenden Objekt gelöscht.

aString:
Bezeichner der Sperre des Datenbank-Objekts

releaseDBReference

Entfernen der Verbindung zum Datenbank-Objekt

Die Datenbankreferenz wird auf nil gesetzt.

transientCopy

Erzeugen einer transienten Kopie eines Datenbank-Objekts

Das Objekt wird ohne Sperren kopiert. Die Datenbankreferenz der Kopie wird auf das betreffende Datenbank-Objekt gesetzt. Die Kopie wird zurückgegeben.

Kategorie copying

copy

Kopie eines DBObjects ohne Locks

Es wird ein neues Objekt mit self copy erzeugt. Die Referenz auf das Datenbank-Objekt, die Read-Locks und der Write-Lock werden auf ihre Initialisierungswerte gesetzt. Das neue Objekt wird zurückgegeben.

copyAcceptLocks

Kopie eines DBObjects unter Beibehaltung der alten Sperren

Es wird ein neues Objekt mit super copy erzeugt. Die Read-Locks und der Write-Lock werden unverändert übernommen. Das neue Objekt wird zurückgegeben.

copyWithReference

Kopie eines DBObjects ohne Locks

Es wird ein neues Objekt mit self copy erzeugt. Die Referenz auf das Datenbank-Objekt wird übernommen. Die Read-Locks und der Write-Lock werden auf ihre Initialisierungswerte gesetzt. Das neue Objekt wird zurückgegeben.

Kategorie initialize

initialize

Initialisierung

Für readLocks wird eine neue Datenstruktur Bag erzeugt, writeLocks wird auf eine neue Sperre gesetzt.

Kategorie locking

releaseLock: aString

Löschen eines Locks

Es ist entweder ein Write-Lock oder eine beliebige Anzahl von Read-Locks oder kein Lock vorhanden. Falls ein Write-Lock mit dem gegebenen Bezeichner vorhanden ist, wird writeLock auf eine neue Sperre gesetzt. Falls gültige Read-Locks mit dem gegebenen Bezeichner vorhanden sind, wird einer gelöscht. Wenn eine Löschung erfolgt, wird das Objekt selbst zurückgegeben. Ist das Objekt nicht gelockt, wird nil zurückgegeben.

aString:

Bezeichner der zu löschenden Sperre (Benutzername\Loginnummer)

setBestLock: aString

Setzen des restriktivsten Locks

Falls ein Write-Lock nicht gesetzt werden kann, wird versucht einen Read-Lock zu setzen und das Ergebnis dieses Versuchs zurückgegeben. Bei erfolgreichem gesetztem Write-Lock wird das Symbol #write zurückgegeben.

aString:

Der zu setzende Lock als String (Benutzername\Loginnummer)

setReadLock: aString

Setzen eines Read-Locks

Falls das Objekt nicht schreibgeschützt ist, kann ein Read-Lock gesetzt werden und es wird das Symbol #read zurückgegeben. Falls ein Write-Lock vorhanden ist, wird #write zurückgegeben, wenn es derselbe Lock wie der gewünschte ist, ansonsten nil.

aString:

Bezeichner der zu setzenden Sperre (Benutzername\Loginnummer)

setWriteLock: aString

Setzen eines Write-Locks

Bei gesetztem Write-Lock wird das Symbol #write zurückgegeben, falls der Lock mit aString übereinstimmt oder ansonsten durch Rückgabe von nil abgelehnt. Ist kein Write-Lock gesetzt, wird zunächst ermittelt, ob eine Lesesperre mit demselben Bezeichner wie für die angeforderte Schreibsperre vorhanden ist. Falls es dann nur die eine Lesesperre gibt, wird sie freigegeben und eine entsprechende Schreibsperre gesetzt. Sind noch andere Sperren vorhanden, wird mit nil abgelehnt.

Ist keine Lesesperre mit dem vorgegebenen Bezeichner vorhanden, wird mit nil abgelehnt, falls es andere Lesesperren gibt oder eine Schreibsperre gesetzt, falls es keine Lesesperren gibt.

Bei erfolgreichem Setzen der Schreibsperre wird das Symbol #write zurückgegeben.

aString:

Bezeichner der zu setzenden Sperre (Benutzername\Loginnummer)

Kategorie printing

printString

Ausgabe als String

Die Methode der Superklasse wird erweitert um die Ausgabe der Read- und des Write-Locks. Dabei werden nur gültige Sperren berücksichtigt.

Kategorie testing

hasWriteLock: aString

Anfrage, ob ein bestimmter Write-Lock gesetzt ist.

Das Ergebnis des Vergleichs mit writeLock wird zurückgegeben. Dabei wird aber nur eine gültige Sperre berücksichtigt.

aString:

Zu überprüfender Lock als String (Benutzername\Loginnummer)

includesReadLock: aString

Anfrage, ob ein bestimmter Read-Lock gesetzt ist

Das Ergebnis der Anfrage bei der Datenstruktur readLocks wird zurückgegeben. Dabei werden aber nur gültige Sperren berücksichtigt.

aString:

Zu überprüfender Lock als String (Benutzername>Loginname)

isLocked

Abfrage, ob das Objekt gelockt ist

Es wird danach gefragt, ob das Objekt mit einem Write- oder einem Read-Lock versehen ist und das Ergebnis davon zurückgegeben.

isReadLocked

Anfrage, ob ein Read-Lock gesetzt ist.

Bei readLocks wird nachgefragt, ob die Datenstruktur leer ist und das Ergebnis zurückgegeben. Dabei werden nur gültige Sperren berücksichtigt.

isReadLockedOnlyBy: aString

Anfrage, ob ein Objekt ausschließlich einen bestimmten Read-Lock hat

Es wird geprüft, ob die Liste mit Read-Locks genau ein Element enthält und ob es sich dabei um den vorgegebenen Lock handelt. Das Ergebnis wird zurückgegeben. Es werden nur gültige Sperren berücksichtigt.

aString:

Zu überprüfender Lock

isWriteLocked

Anfrage, ob ein Write-Lock gesetzt ist

Es wird nach einer gültigen Schreibsperre gefragt. Das Ergebnis einer Überprüfung, ob es sich dabei um das Nullelement handelt wird negiert zurückgegeben.

Kategorie instance creation (class)

new

Default-Erzeugung

Senden an die Superklasse und Initialisierung.

Klasse ElabApplicationModel

Diese Klasse ist Oberklasse für alle ApplicationModels des Systems Elab. Ausgenommen sind die Chargensplitting-Workfloweditoren. Die Klasse implementiert einen Mechanismus zum Aufbau und zur Verwaltung einer Hierarchie zwischen den Fenstern.

Oberklasse:

ApplicationModel

Instanzvariablen:

openChild:

Ein Set mit Models von Fenstern, die von einer Funktion des betreffenden Models aus geöffnet wurden.

parent:

Das Model des Fensters, von dem aus das Fenster des betreffenden Models geöffnet wurde.

user:

Besitzer des Fensters.

Kategorie accessing

openChilds

Zugriff auf die aus dem vorliegenden ApplicationModel heraus geöffneten ApplicationModels

Die Instanzvariable openChilds (Set) wird zurückgegeben.

openChilds: aSet

Neue Datenstruktur für openChilds

Die Instanzvariable openChilds wird auf das neue Set gesetzt und zurückgegeben.

aSet:

Neue Datenstruktur (Set)

parent

Zugriff auf das ApplicationModel, von dem das vorliegende geöffnet wurde

Die Instanzvariable parent wird zurückgegeben.

parent: aModel

Ändern des ApplicationModels, von dem das vorliegende Model geöffnet wurde.

Die Instanzvariable parent wird auf das neue Model gesetzt und zurückgegeben.

aModel:

Das neue ApplicationModel

user

Zugriff auf den Benutzer, dem das Fenster gehört

Die Instanzvariable user wird zurückgegeben.

user: aUser

Ändern des Benutzers, dem das Fenster gehört

Die Instanzvariable user übernimmt den neuen Benutzer und wird zurückgegeben.

aUser:

Neuer Benutzer

Kategorie actions

openLogs

Öffnen eines Protokollgenerators (Button 'Logs')

Hier wurde die ursprünglich bei der Klasse ElektronischesLaborbuch implementierte Methode openProtokollgenerator verwendet. Ein Öffnen als Kind des vorliegenden Modells ist nicht möglich, weil nicht von ApplicationModel geerbt wird.

Zunächst wird geprüft, ob die Protokoll-Datenbank vorhanden ist. Wenn nicht, wird eine Fehlermeldung ausgegeben und nil zurückgeliefert.

Der Benutzer des Chargensplitting-Workfloweditors wird gesetzt. Außerdem wird der Editor selbst dem Model bekanntgemacht. Der Name des geladenen Protokolls wird beim Editor gesetzt. Das Label des Editors wird gesetzt.

Die Klassenvariablen RootExpressionForOpenProtokollgenerator und AktuellProtokollName werden auf nil gesetzt.

restartEleLabWin

Hauptfenster in den Vordergrund holen

Zunächst wird das Hauptfenster (das Model) ermittelt. Falls das Ergebnis nil ist, also kein Hauptfenster ermittelt werden konnte, wird der Benutzer informiert. Sonst wird das Hauptfenster geöffnet, falls es ikonifiziert ist und es wird in den Vordergrund geholt.

wahlVonFragmenten

Auswahl eines Fragments als Eingabe für den Protokollgenerator

Die Namen der Fragmente, die zu den Projektgruppen des Benutzers gehören, werden in eine Liste eingelesen. In einem Dialog kann der Benutzer aus dieser Liste das Fragment auswählen, aus dem ein Protokoll erzeugt werden soll. Bei Abbruch des Dialogs wird nil zurückgegeben.

Ansonsten wird die Fragment-Datenbank geöffnet und in einer Transaktion eine Kopie des ausgewählten Fragments aus der Datenbank geholt. Falls das Fragment

nicht mehr vorhanden oder schreibgeschützt ist, wird die Transaktion abgebrochen, der Benutzer informiert und nil zurückgegeben.

Der Benutzer wird nach einem Namen für das neue Protokoll gefragt. Falls ein leerer String angegeben wird, wird mit der Rückgabe von nil abgebrochen.

Ansonsten wird die Protokoll-Datenbank geöffnet. In einer Transaktion wird überprüft, ob der gewählte Name bereits existiert und falls nicht, ein neues, schreibgeschütztes Protokoll in die Datenbank geschrieben, um den Namen zu belegen.

Falls der Name bereits existiert, wird der Benutzer außerhalb der Transaktion danach gefragt, ob er einen neuen Versuch wünscht. Bei einem neuen Versuch wird die Nachricht rekursiv gesendet und deren Rückgabewert zurückgegeben. Ansonsten wird nil zurückgegeben.

Falls der Name noch nicht existiert, wird, falls der Benutzer nur einer Projektgruppe angehört, diese als Projektgruppe für das neue Protokoll übernommen. Ansonsten kann der Benutzer aus seinen Projektgruppen eine Gruppe auswählen. Verzichtet er auf die Auswahl, wird die erste Gruppe aus der Liste des Benutzers übernommen. Ein Protokoll mit dem neuen Namen sowie der Prozessliste des ausgewählten Fragments und der festgelegten Projektgruppe wird neu erzeugt. Die entsprechende Klassenvariable (`aktuellProtokollName`) von `ChargensplittingWorkflow` wird mit dem neuen Protokoll belegt. Außerdem wird eine Kopie dieses Protokolls angelegt. In einer Transaktion wird diese Kopie schreibgeschützt, die Sperre beim Platzhalterobjekt in der Datenbank freigegeben und die Kopie in die Datenbank geschrieben. Anschließend wird der Root-Prozess des neuen Protokolls zurückgegeben.

wahlVonProtokollen

Auswahl eines Protokolls als Eingabe für den Protokollgenerator

Die Namen der Protokolle, die zu den Projektgruppen des Benutzers gehören, werden in eine Liste eingelesen. In einem Dialog kann der Benutzer aus dieser Liste das Protokoll auswählen, das im Editor dargestellt werden soll. Bei Abbruch des Dialogs wird nil zurückgegeben.

Ansonsten wird die Protokoll-Datenbank geöffnet. In einer weiteren Transaktion wird eine transiente Kopie des gewünschten Protokolls aus der Datenbank geholt. Das Objekt wird dabei schreibgeschützt. Wenn das Objekt nicht mehr vorhanden oder gesperrt ist, wird die Transaktion abgebrochen, der Benutzer entsprechend informiert und nil zurückgegeben.

Bei erfolgreichem Anlegen der Kopie wird die Klassenvariable `aktuellProtokollName` von `ChargensplittingWorkflow` mit dem Protokoll belegt und der Root-Prozess des Protokolls wird zurückgegeben.

wahlVonProtokollenUndKopie

Auswahl eines Protokolls als Eingabe für den Protokollgenerator (als Kopie)

Die Namen der Protokolle, die zu den Projektgruppen des Benutzers gehören, werden in eine Liste eingelesen. In einem Dialog kann der Benutzer aus dieser

Liste das Fragment auswählen, aus dem ein Protokoll erzeugt werden soll. Bei Abbruch des Dialogs wird nil zurückgegeben.

Ansonsten wird die Protokoll-Datenbank geöffnet und in einer Transaktion eine Kopie des ausgewählten Protokolls aus der Datenbank geholt. Falls das Protokoll nicht mehr vorhanden oder schreibgeschützt ist, wird die Transaktion abgebrochen, der Benutzer informiert und nil zurückgegeben.

Der Benutzer wird nach einem Namen für das neue Protokoll gefragt. Falls ein leerer String angegeben wird, wird mit der Rückgabe von nil abgebrochen.

In einer Transaktion wird überprüft, ob der gewählte Name bereits existiert und falls nicht, ein neues, schreibgeschütztes Protokoll in die Datenbank geschrieben, um den Namen zu belegen.

Falls der Name bereits existiert, wird der Benutzer außerhalb der Transaktion danach gefragt, ob er einen neuen Versuch wünscht. Bei einem neuen Versuch wird die Nachricht rekursiv gesendet und deren Rückgabewert zurückgegeben. Ansonsten wird nil zurückgegeben.

Falls der Name noch nicht existiert, wird, falls der Benutzer nur einer Projektgruppe angehört, diese als Projektgruppe für das neue Protokoll übernommen. Ansonsten kann der Benutzer aus seinen Projektgruppen eine Gruppe auswählen. Verzichtet er auf die Auswahl, wird die erste Gruppe aus der Liste des Benutzers übernommen. Ein Protokoll mit dem neuen Namen und der festgelegten Projektgruppe wird als neue Version des zu kopierenden Protokolls erzeugt. Die entsprechende Klassenvariable (aktuellProtokollName) von `ChargensplittingWorkflow` wird mit dem neuen Protokoll belegt. Außerdem wird eine Kopie dieses Protokolls angelegt. In einer Transaktion wird diese Kopie schreibgeschützt, die Sperre beim Platzhalterobjekt in der Datenbank freigegeben, die Kopie in die Datenbank geschrieben und beim Original die referenz auf das Datenbank-Objekt gesetzt. Anschließend wird der Root-Prozess des neuen Protokolls zurückgegeben.

Kategorie changing

updateProcessNames

Dummy-Methode

Diese Dummy-Methode wird benötigt, weil Objekte der Klasse `ElabApplicationModel` beim `Chargensplitting-Workfloweditor` als Dummy-Models verwendet werden. Dabei wird diese Nachricht zwar gesendet; ein Aktualisieren von Prozessnamen findet aber nicht statt.

Kategorie children

addChild: aBuilder

Ein neu geöffnetes Fenster wird der Liste mit geöffneten Fenstern hinzugefügt.

`openChilds` wird zur Aufnahme des Builders aufgefordert. Das Model des Builders (`source`) erhält das vorliegende Objekt als erzeugendes Model.

aBuilder:
Builder des neu geöffneten Fensters

Kategorie events

noticeOfWindowClose: aWindow

Aufräumarbeiten, die beim Schließen des Fensters automatisch durchgeführt werden sollen

Falls ein ApplicationModel vorhanden ist, von dem aus das vorliegende geöffnet wurde, wird dieses informiert, damit das vorliegende Model aus dessen Liste von geöffneten Models gelöscht wird. Anschließend werden diejenigen Models geschlossen, die vom vorliegenden Model aus geöffnet wurden.

aWindow:
Das Fenster, das geschlossen wird.

Kategorie initialize

initialize

Initialisierung

Für die Instanzvariable openChilds wird ein neues Set erzeugt, für user ein neuer Benutzer.

Kategorie parents

mainWindow

Ermitteln des Hauptfensters von Elab

Falls das betreffende Fensters kein darüberliegendes Fenster besitzt, wird nil zurückgegeben. Ansonsten wird der Rückgabewert des darüberliegenden Fensters beim Senden derselben Nachricht zurückgegeben.

Kategorie private

closeChilds

Schließen aller von diesem ApplicationModel aus geöffneten Models

Die Instanzvariable openChilds enthält in einem Set alle Builder der geöffneten Fenster. Jeweils vom Model dieser Builder (Methode source) wird das Schließen des Fensters verlangt. Für openChilds wird ein neues Set erzeugt.

closeEditors

Schließen von Prozesseditoren

Mitunter müssen eventuelle Editiervorgänge an Prozessen von außen abgeschlossen werden. Dabei soll keine Speicherung der Änderungen im Editor erfolgen, die nicht explizit vom Benutzer innerhalb des Editors angefordert waren. Aus diesem Grund wird jeder Editor dazu aufgefordert, seine Prozesse so umzusetzen, dass der Zustand beim Öffnen bzw. beim letzten expliziten Speichern wiederhergestellt wird. Danach wird der Editor zum Schließen aufgefordert.

Bei Editoren, die für Templates geöffnet waren, ist dieses explizite Schließen nicht erwünscht.

Kategorie removing

removeChild: aModel

Entfernen eines von diesem ApplicationModel aus geöffneten Models aus der eigenen Liste

Aus openChilds wird das übergebene Model entfernt, falls es nicht vorhanden ist, wird nichts getan.

aModel:

das zu entfernende Model

Kategorie instance creation (class)

newParent: parentModel

Erzeugen eines Application Models mit Übernahme des erzeugenden Models

Es wird ein neues Model erzeugt, die Instanzvariable parent gesetzt und das neue Model zurückgegeben.

parentModel:

Das erzeugende Model

newUser: aUser

Erzeugen eines ApplicationModels mit Übernahme des Besitzers

Ein neues Model wird erzeugt, der Besitzer gesetzt und das Model wird zurückgegeben.

aUser:

Besitzer

Kategorie interface opening (class)

openOnUser: aUser

Öffnen eines neuen Fensters für einen bestimmten Besitzer

Ein neues Model mit Besitzer wird erzeugt und geöffnet. Der Builder wird zurückgegeben.

aUser: Besitzer des neuen Fensters

Klasse ElabCalculator

Diese Klasse beschreibt den im System Elab integrierten Taschenrechner.

Oberklasse:

ElabApplicationModel

Instanzvariablen:

calculator:

Das Objekt, das die eigentliche Arbeit verrichtet und die eigentlichen Modelle der Oberflächenelemente zur Verfügung stellt.

Kategorie binding

actionFor: aKey

Nicht kommentiert.

aKey:

Nicht kommentiert

aspectFor: aKey

Nicht kommentiert.

aKey:

Nicht kommentiert

Kategorie initialize-release

calculator: aCalc

Nicht kommentiert.

aCalc:

Nicht kommentiert

initialize

Initialisierung

Die Instanzvariablen der Superklasse werden initialisiert. Für die Instanzvariable calculator wird ein neues Objekt der Klasse calculator erzeugt.

Kategorie interface opening

postBuildWith: aBuilder

Nicht kommentiert.

aBuilder:

Nicht kommentiert

Kategorie private

keyPress: ev

Nicht kommentiert.

ev:

nicht kommentiert

pressButton: componentName

Nicht kommentiert.

componentName:

nicht kommentiert

Kategorie filtering (class)

isVisualExample

Used by UIFinder to divine that the receiver is an example class.

Kategorie interface specs (class)

windowSpec

Klasse ElabLock

Diese Klasse implementiert eine semantische Sperre für Datenbankobjekte des Systems Elab. Eine solche Sperre wird i. allg. von einem Objekt der Klasse ElabLockDaemon überwacht. Es besitzt eine Gültigkeitsdauer, die von Zeit zu Zeit aktualisiert werden muss.

Oberklasse:

Object

Instanzvariablen:

lockString:

Zeichenkette als Bezeichner für die Sperre. Dieser Bezeichner muss eindeutig sein.

offset:

Die Gültigkeitsdauer der Sperre.

timeStamp:

Zeitstempel, der angibt, wie lange die Sperre gültig ist. Beim Setzen eines Zeitstempels wird offset zur aktuellen Uhrzeit addiert.

validity:

Symbol, das den Zustand der Sperre angibt:

#undefined *Nullelement*

#valid *gültig*

#invalid *ungültig; das betreffende Objekt kann wieder benutzt werden*

#released *Sperre wurde explizit freigegeben; das betreffende Objekt kann wieder benutzt werden*

Kategorie accessing

lockString

Zugriff auf den Bezeichner der Sperre

Der Bezeichner besteht aus dem Namen des Benutzers, der die Sperre gesetzt hat, aus dem Zeichen 'l' und der Login-Nummer. Der Bezeichner wird zurückgegeben.

lockString: aString

Setzen des Bezeichners der Sperre

Der Bezeichner besteht aus dem Namen des Benutzers, der die Sperre gesetzt hat, aus dem Zeichen 'l' und der Login-Nummer. Der Bezeichner wird auf aString gesetzt und zurückgegeben.

aString:

String, der den zu setzenden Bezeichner enthält.

offset

Zugriff auf die Gültigkeitsdauer der Sperre

Die Gültigkeitsdauer einer Sperre wird in s angegeben. Sie kann um denselben Betrag verlängert werden. Die Gültigkeitsdauer wird zurückgegeben.

offset: aNumber

Setzen der Gültigkeitsdauer

Die Gültigkeitsdauer einer Sperre wird in *s* angegeben. Sie kann um denselben Betrag verlängert werden. Die Gültigkeitsdauer wird auf *aNumber* gesetzt und zurückgegeben.

aNumber:
Zeit in s

timeStamp

Zugriff auf den Zeitstempel

Der Zeitstempel ist die Zeit des Ablaufs der Gültigkeit der Sperre in *s* seit dem 1.1.1901. Er wird zurückgegeben.

timeStamp: aNumber

Setzen des Zeitstempels

Der Zeitstempel ist die Zeit des Ablaufs der Gültigkeit der Sperre in *s* seit dem 1.1.1901. Er wird auf *aNumber* gesetzt und zurückgegeben.

aNumber:
Zeit in s

validity

Zugriff auf den aktuellen Zustands der Sperre

Der Zustand kann mit den Symbolen *#valid*, *#invalid*, *#released* und dem Nullelement *#undefined* beschrieben werden. Der aktuelle Zustand wird zurückgegeben.

validity: aSymbol

Setzen des Zustands der Sperre

Der Zustand kann mit den Symbolen *#valid*, *#invalid*, *#released* und dem Nullelement *#undefined* beschrieben werden. Der aktuelle Zustand wird nur dann auf *aSymbol* gesetzt, wenn es sich um ein zulässiges Symbol handelt. Andernfalls wird eine Fehlermeldung ausgegeben und der Zustand auf *#undefined* gesetzt. Der Zustand wird zurückgegeben.

aSymbol:
Zulässiges Symbol für die Beschreibung des Zustands einer Sperre

Kategorie comparing

= anObject

Objektvergleich

Falls das zu vergleichende Objekt nicht der Klasse *ElabLock* angehört, wird *false* zureckgegeben. Wenn die Werte aller Instanzvariablen gleich sind, wird *true*, andernfalls *false* zurückgegeben.

anObject

Zu vergleichendes Objekt

compareString: aString

Vergleich des Bezeichners mit angegebenem String

Das Ergebnis eines Vergleichs zwischen dem Bezeichner der Sperre und angegebenem String wird zurückgegeben.

aString:

Zu vergleichender String

Kategorie initialize

initialize

Initialisierung

Die Initialisierung der Superklasse wird durchgeführt. Die Gültigkeitsdauer wird auf den Standardwert der Klasse gesetzt. Die übrigen Instanzvariablen werden auf ihre Nullwerte gesetzt.

Kategorie modification

getUpdatePeriod

Liefert die Zeitdauer, nach der ein Aktualisieren der Gültigkeitsdauer erfolgen soll

Die Zeit in s für die Aktualisierungsperiode wird zurückgegeben.

markInvalid

Markiert einen Lock als ungültig

Die Zustandsbeschreibung der Sperre wird auf #invalid gesetzt und zurückgegeben. Wenn der Update-Prozess das nächste mal den Zeitstempel erhöhen will, erkennt er, dass der Lock nicht mehr gültig ist und beendet sich.

markReleased

Markiert einen Lock als freigegeben

Die Zustandsbeschreibung der Sperre wird auf #released gesetzt und zurückgegeben. Wenn der Update-Prozess das nächste mal den Zeitstempel erhöhen will, erkennt er, dass der Lock freigegeben ist und beendet sich.

markValid

Markiert einen Lock als gültig

Die Zustandsbeschreibung der Sperre wird auf #valid gesetzt und zurückgegeben.

setInitialTimeStamp

Setzt den ersten Zeitstempel, bis zu dem das Objekt gültig ist.

Die Sperre wird als gültig gekennzeichnet. Der Zeitstempel wird durch Aktualisieren auf einen gültigen Wert gesetzt.

updateTimeStamp

Aktualisieren des Zeitstempels der Sperre

Bei gültigem Zeitstempel wird dieser inkrementiert und true zurückgegeben. Andernfalls wird false zurückgegeben.

Kategorie printing

printString

Standardausgabe

Es erfolgt die Standardausgabe der Oberklasse. Die Werte der Instanzvariablen werden mit Kommentaren ausgegeben.

Kategorie testing

isExpired

Überprüft, ob der gesetzte Zeitstempel noch aktuell ist

Der Zeitstempel ist abgelaufen, wenn die aktuelle Zeit (Systemzeit in s seit dem 1.1.1901) größer als der Zeitstempel ist.

isNilLock

Vergleich mit Nullelement

Das Objekt wird mit einem neu erzeugten Nullelement verglichen und das Ergebnis zurückgegeben.

isValid

Abfrage auf Gültigkeit

Falls der Zustand #valid ist, d.h. die Sperre ist gültig, wird true, andernfalls false zurückgegeben.

Kategorie defaults (class)

defaultOffset

Standardwert für die Gültigkeitsdauer

Ein konstanter Wert wird zurückgegeben.

Kategorie installing (class)

installLockString: aString offset: aNumber validity: aSymbol

Installieren einer neuen Sperre mit Übergabe eines Bezeichners und einer Zustandsbeschreibung

Eine neue Sperre wird mit dem angegebenen Bezeichner, der angegebenen Gültigkeitsdauer und der angegebenen Zustandsbeschreibung erzeugt. Der Anfangszeitstempel wird gesetzt. Ein Objekt zur Überwachung und Aktualisierung der Sperre wird installiert. Die neue Sperre wird zurückgegeben.

aString:
Bezeichner

aNumber:
Zeit in s

aSymbol:
Zustand

installLockString: aString validity: aSymbol

Installieren einer neuen Sperre mit Übergabe eines Bezeichners und einer Zustandsbeschreibung

Eine neue Sperre wird mit dem angegebenen Bezeichner und der angegebenen Zustandsbeschreibung erzeugt. Der Anfangszeitstempel wird gesetzt. Ein Objekt zur Überwachung und Aktualisierung der Sperre wird installiert. Die neue Sperre wird zurückgegeben.

aString:
Bezeichner

aSymbol:
Zustand

Kategorie instance creation (class)

new

default creator.

newLockString: aString offset: aNumber validity: aSymbol

Erzeugung einer neuen Sperre mit Übergabe eines Bezeichners, einer Gültigkeitsdauer und einer Zustandsbeschreibung

Eine neue Sperre wird mit dem angegebenen Bezeichner und dem angegebenen Zustand erzeugt. Die Gültigkeitsdauer wird auf aNumber gesetzt. Die neue Sperre wird zurückgegeben.

aString:
Bezeichner

aNumber:
Zeit in s

aSymbol:
Zustand

newLockString: aString validity: aSymbol

Erzeugung einer neuen Sperre mit Übergabe eines Bezeichners und einer Zustandsbeschreibung

Eine neue Sperre wird erzeugt. Der Bezeichner wird auf aString gesetzt und das Objekt in den angegebenen Zustand versetzt. Die neue Sperre wird zurückgegeben.

aString:
Bezeichner

aSymbol:
Zustand

Klasse ElabLockDaemon

Diese Klasse implementiert einen Aktualisierungsprozess der eine semantische Sperre im Elab-System überwacht.

Oberklasse:

Object

Instanzvariablen:

updateProcess:

Ein vom Hauptprozess abstammender unabhängiger Prozess, der eine Sperre überwacht und ggf. aktualisiert.

Kategorie accessing

updateProcess

Zugriff auf den Prozess zur Aktualisierung

Der Aktualisierungsprozess wird zurückgegeben.

updateProcess: aProcess

Setzen des Aktualisierungsprozesses

Der angegebene Aktualisierungsprozess wird übernommen und die Instanzvariable *updateProcess* zurückgegeben.

aProcess:

Aktualisierungsprozess

Kategorie release

terminate

Beenden des Aktualisierungsprozesses

Falls ein Aktualisierungsprozess gestartet ist, wird dieser beendet.

Kategorie installing (class)

openOnLock: aLock

Installieren eines Überwachungsobjekt für eine Sperre

Ein neues Überwachungsobjekt wird erzeugt. In einer Transaktion wird die Aktualisierungsperiode der Sperre angefordert.

Ein Aktualisierungsprozess wird eingerichtet. Dieser Prozess führt in einer Transaktion die Aktualisierung der Sperre durch. Diese Aktualisierung wird nach Ablauf der Periode solange wiederholt, bis sie von der Sperre abgelehnt wird. Wenn keine Aktualisierung mehr erfolgt, wird der Prozess beendet.

Die Priorität des Aktualisierungsprozesses wird geändert und der Prozess gestartet.

aLock:
Sperre

Kategorie instance creation (class)

new

default creator.

newUpdateProcess: aProcess

Erzeugung eines neuen Überwachungsobjekts mit Übernahme AktualisierungsprozessElabLockDaemon

Ein neues Überwachungsobjekt wird erzeugt und der Aktualisierungsprozess auf aProcess gesetzt. Das neue Objekt wird zurückgegeben.

aProcess:
Aktualisierungsprozess

Klasse ElabLogout

Diese Klasse implementiert ein Fenster, das dem Benutzer das Beenden des Programms anzeigt. Außerdem wird das Beenden mit Hilfe dieser Klasse tatsächlich durchgeführt. Ein zusätzliches Fenster ist hauptsächlich deshalb notwendig, damit alle wesentlichen Fenster des Systems Elab geschlossen werden können und dabei auch alle Aufräumarbeiten wie etwa die Freigabe von Sperren durchgeführt werden können, ohne daß ein VisualWorks-Hauptfenster auf dem Bildschirm erscheint.

Oberklasse:
ApplicationModel

Kategorie interface opening (class)

Öffnen eines Fensters beim Beenden von Elab

Die Nachricht wird zunächst an die Oberklasse weitergeleitet. Diese Klasse startet dann noch einen Prozess, der 10 s wartet und dann das Programm beendet.

Kategorie interface specs (class)

windowSpec

Klasse ElabMain

Diese Klasse beschreibt das Model des Elab-Hauptfensters. Sie implementiert die Methoden, die bei Betätigung von Hauptfenster-Buttons ausgeführt werden.

Oberklasse:

ElabApplicationModel

Kategorie actions

addUser

Erzeugen eines neuen Benutzers für das System Elab (Button 'Add User' im Elab-Hauptfenster)

Falls der Benutzer nicht Superuser ist, wird der Zugriff verweigert und nil zurückgegeben. Innerhalb des Systems Elab kann dieser fall nicht eintreten, da der betreffende Button nur für Superuser aktiv ist.

In openChilds wird nachgesehen, ob ein Builder mit einem Model (Methode source) der Klasse UserModel vorhanden ist. Ein Elab-Hauptfenster soll immer nur ein UserModel geöffnet haben. Falls ein solches Model gefunden wird, wird es aufgefordert, einen neuen User mit dem Dummy-Namen ElabUser und dem Zugriffsrecht #write darzustellen. Das Fenster, das möglicherweise unter anderen Fenstern versteckt liegt, wird in den Vordergrund geholt.

Falls kein UserModel gefunden wurde, wird ein neues als Kind des Elab-Hauptfensters und mit demselben User geöffnet.

In beiden Fällen wird das Objekt selbst zurückgegeben.

changePassword

Ändern des Passworts (Button 'Change Password' im Elab-Hauptfenster)

Die Benutzer-Datenbank wird geöffnet. Auf das Datenbankobjekt wird ein Write-Lock gesetzt. Falls dies nicht möglich ist, wird die Transaktion abgebrochen, der Benutzer informiert und nil zurückgegeben. Andernfalls wird das Passwort bei der im Elab-Hauptfenster vorhandenen Kopie des Benutzers geändert. In einer zweiten Transaktion wird der aktuelle Benutzer in die Benutzer-Datenbank zurückgeschrieben. Das Datenbank-Objekt erhält eine neue Lesesperre und für den aktuellen benutzer wird eine neue transiente Kopie angelegt.

editUser

Editieren der Daten eines Benutzers (Button 'Edit User' im Elab-Hauptfenster)

Falls der Benutzer nicht Superuser ist, wird der Zugriff verweigert und nil zurückgegeben. Innerhalb des Systems Elab kann dieser fall nicht eintreten, da der betreffende Button nur für Superuser aktiv ist.

Es wird eine neue Liste (SortedCollection) erzeugt und die Benutzer-Datenbank geöffnet. In einer Transaktion wird diese Liste mit den Namen der in der Datenbank vorhandenen Benutzern gefüllt.

In einem Dialogfenster wird der Benutzer aufgefordert, einen Namen aus der Liste zu wählen. Falls er abbricht, wird nil zurückgegeben.

In openChilds wird nachgesehen, ob ein Builder mit einem Model (Methode source) der Klasse UserModel vorhanden ist. Ein Elab-Hauptfenster soll immer nur ein UserModel geöffnet haben. Falls ein solches Model gefunden wird, wird es aufgefordert, den Benutzer mit dem ausgewählten Namen und dem Zugriffsrecht #edit darzustellen. Das Fenster, das möglicherweise unter anderen Fenstern versteckt liegt, wird in den Vordergrund geholt.

Falls kein UserModel gefunden wurde, wird ein neues als Kind des Elab-Hauptfensters und mit demselben User geöffnet.

In beiden Fällen wird das Objekt selbst zurückgegeben.

lock

Aktionen beim Betätigen des Buttons 'Lock'

Der Button 'Lock' dient abwechselnd dem Sperren des Systems Elab und dem Aufheben der Sperre (dann mit dem Label 'Unlock'). Je nach Label sind damit zwei unterschiedliche Abfolgen von Aktionen durchzuführen.

'Lock': Der Benutzer wird darüber informiert, dass evtl. nicht gesicherte Änderungen verlorengehen. Falls der Benutzer damit nicht einverstanden ist und den Vorgang abbricht, wird nil zurückgegeben. Ansonsten werden alle Fenster bis auf das Hauptfenster geschlossen. Im Hauptfenster werden alle Buttons für den normalen Benutzer und, falls es sich um einen Superuser handelt, auch die Superuser-Buttons deaktiviert. Das Label des Lock-Buttons wird geändert.

'Unlock': Der Benutzer hat drei Versuche, das Passwort einzugeben. Falls er bei einem dieser drei Versuche abbricht wird der gesamte Vorgang abgebrochen und die Sperre nicht entfernt. Wird das korrekte Passwort eingegeben, werden die Buttons für den normalen Benutzer und, falls es sich um einen Superuser handelt, die Superuser-Buttons aktiviert und das Label des Lock-Buttons geändert. Ist auch das dritte Passwort falsch, wird die Applikation beendet.

openFragments

Öffnen einer Fragmentenverwaltung (Button 'Fragments')

Die Methode öffnet die entsprechende Teilapplikation. Der eigene Besitzer wird als neuer Besitzer übergeben. Falls die fragmentenverwaltung erfolgreich geöffnet wurde, wird die Fragmentliste der Fragmentenverwaltung explizit initialisiert. Die neue Fragmentenverwaltung wird Kind des Hauptfensters.

openProducts

Öffnen eines Fensters 'Produktverwaltung & Statistik' (Button 'Products')

Ein Fenster Produktverwaltung wird als Kind des Elab-Hauptfensters geöffnet. Der eigene Besitzer wird als Besitzer der Fragmentenverwaltung übergeben.

printerSettings

Öffnen des Fensters für Drucker-Einstellungen

Zunächst wird überprüft, ob bereits ein Fenster für die Druckereinstellung geöffnet ist. Wenn das der Fall ist, wird es, falls es als Icon vorliegt, geöffnet. Danach wird es in den Vordergrund geholt.

Falls kein Fenster vorhanden ist, wird eins als Kind des Hauptfensters geöffnet.

quit

Beenden des Elab-Systems (Button 'Quit' im Elab-Hauptfenster)

Das Model selbst wird zum Schließen aufgefordert.

removeUser

Entfernen eines Benutzers aus der Datenbank (Button 'Remove User' im Elab-Hauptfenster)

Falls der Benutzer nicht Superuser ist, wird der Zugriff verweigert und nil zurückgegeben. Innerhalb des Systems Elab kann dieser fall nicht eintreten, da der betreffende Button nur für Superuser aktiv ist.

Es wird eine neue Liste (SortedCollection) erzeugt und die Benutzer-Datenbank geöffnet. In einer Transaktion wird diese Liste mit den Namen der in der Datenbank vorhandenen Benutzern gefüllt.

In einem Dialogfenster wird der Benutzer aufgefordert, einen Namen aus der Liste zu wählen. Falls er abbricht, wird nil zurückgegeben.

Der Benutzer darf sich nicht selbst löschen. Falls er das versucht, wird er informiert und nil zurückgegeben. Außerdem muss der Benutzer das Löschen bestätigen. Tut er das nicht, wird nil zurückgegeben.

In einer Transaktion wird der Benutzer aus der Datenbank gelöscht. Das kann nicht geschehen, wenn der Benutzer nicht mehr vorhanden ist oder eine Sperre gesetzt ist. In diesen Fällen wird die Transaktion abgebrochen, der Benutzer informiert und nil zurückgegeben.

Wenn das Löschen möglich ist, wird das Objekt selbst zurückgegeben.

showUser

Anzeigen der Daten des Benutzers

In openChilds wird nachgesehen, ob ein Builder mit einem Model (Methode source) der Klasse UserModel vorhanden ist. Ein Elab-Hauptfenster soll immer nur ein UserModel geöffnet haben. Falls ein solches Model gefunden wird, wird es aufgefordert, den aktuellen Benutzer darzustellen. Das Fenster, das möglicherweise unter anderen Fenstern versteckt liegt, wird in den Vordergrund geholt.

Falls kein UserModel gefunden wurde, wird ein neues als Kind des Elab-Hauptfensters und mit demselben User geöffnet.

In beiden Fällen wird das Objekt selbst zurückgegeben.

showUsers

Anzeigen der Daten eines beliebigen Benutzers, der gerade das System Elab benutzt

Über einen Dialog kann der Benutzer den Namen eines der gerade eingeloggten Benutzer auswählen.

In openChilds wird nachgesehen, ob ein Builder mit einem Model (Methode source) der Klasse UserModel vorhanden ist. Ein Elab-Hauptfenster soll immer nur ein UserModel geöffnet haben. Falls ein solches Model gefunden wird, wird es aufgefordert, den Benutzer mit dem ausgewählten Namen und dem Zugriffsrecht #read darzustellen. Das Fenster, das möglicherweise unter anderen Fenstern versteckt liegt, wird in den Vordergrund geholt.

Falls kein UserModel gefunden wurde, wird ein neues als Kind des Elab-Hauptfensters und mit demselben User geöffnet.

In beiden Fällen wird das Objekt selbst zurückgegeben.

startSmalltalk

Starten der Entwicklungsumgebung (Button 'Smalltalk')

Nach einer Bestätigung des Benutzers wird der Benutzer gefragt, ob er den Pfad von Source- und Changes-Datei ändern will und ggf. das Fenster 'Settings' geöffnet. Anschließend wird das Hauptfenster von VisualWorks und ein Workspace mit dem Inhalt der Datei 'workspace.txt' geöffnet. Die Klassenvariable saveEnabled der Klasse Login wird auf true gesetzt. Dies ermöglicht nach Beendigung der Entwicklungsarbeiten das Sichern des Images, wobei lediglich ein Login-Fenster geöffnet ist.

Fehlt die Bestätigung, wird nil zurückgegeben.

Kategorie events

noticeOfWindowClose: aWindow

Aufräumarbeiten, die beim Schließen des Fensters automatisch durchgeführt werden sollen

Falls der Aktuelle Benutzer ordnungsgemäß beim system angemeldet ist, werden folgende Aktionen durchgeführt:

Die Login-Datenbank wird geöffnet. Die Verbindung zwischen aktuellem Benutzer und dem Datenbank-Objekt wird beendet. Der aktuelle Benutzer wird auch aufgefordert, seine Login-Daten aus der Login-Datenbank zu entfernen. Nach Beenden der Transaktion wird die Methode der Oberklasse aufgerufen.

Falls der Aktivierungszustand des Buttons zum Sichern im Login-Fenster false ist, d.h. die Entwicklungsumgebung ist nicht aktiviert, werden alle Objekte der Klasse ElabLockDaemon aufgefordert, ihre Aktualisierungsprozesse zu beenden und das gesamte System durch Öffnen eines Fensters zum Beenden terminiert. Falls die Entwicklungsumgebung aktiviert ist, bleibt es beim Schließen der Elab-Fenster.

aWindow:

Das Fenster, das geschlossen wird.

Kategorie interface opening

postBuildWith: aBuilder

Durchführung von Aktionen nach dem Öffnen des Fensters

Falls der Benutzer nicht Superuser ist, werden die entsprechenden Buttons für den Superuser deaktiviert. Das Label des Hauptfensters wird gesetzt (Name des Benutzers und Datum und Uhrzeit des Anmeldens). Die Druckereinstellung wird auf ihren Standardwert gesetzt.

aBuilder:

Builder des Fensters

Kategorie parents

mainWindow

Ermitteln des Hauptfensters von Elab

Das Hauptfenster gibt sich selbst zurück.

Kategorie private

closeChilds

Schließen der Kinder des Elab-Hauptfensters

Die Methode der Superklasse wird aufgerufen. Wegen der besonderen Implementierung der Protokollgeneratoren (keine Unterklassen von ApplicationModel) muss nach Controllern gesucht werden, deren Models der Klasse RrPixmapHolder angehören. Diese müssen extra geschlossen werden. Hierbei darf aber kein Speichern von Protokollen erfolgen (Nachricht noSave: true) und die Models müssen explizit zur Durchführung von bestimmten Aktionen aufgefordert werden.

Kategorie interface specs (class)

windowSpec

Klasse ElabProcessEditor

Diese Klasse ist Oberklasse für alle existierenden Models für Prozesseditoren. Ursprünglich wurden die Prozesseditoren unabhängig voneinander implementiert. Der überwiegende Teil ihrer Funktionalität ist jedoch identisch. In dieser Klasse wurde einiges an identischer Funktionalität zusammengefasst. Die Klasse kann bei der Weiterentwicklung von Elab kontinuierlich zu einer Oberklasse ausgebaut werden, die die gesamte parallel implementierte Funktionalität der Prozesseditoren enthält.

Oberklasse:

ElabApplicationModel

Instanzvariablen:

aLocalProzess:

Ein Objekt der Klasse KleinerStandardProzesseditor, in dem alle Oberflächenelemente zusammengefasst sind, die alle Prozesseditoren gemeinsam haben. Hier besteht auch Zugriff zum aktuell dargestellten Prozess.

readOnly:

Kennzeichnet Editoren, die nur zum Lesen geöffnet sind (true).

template:

Wird ein Prozesseditor zur Darstellung von Prozesstemplates verwendet, ist sein Erscheinungsbild etwas anders als üblich und er muss etwas anders als sonst verwaltet werden. Mit diesem Flag (true) wird die Benutzung als Editor für ein Template gekennzeichnet.

Kategorie accessing

aLocalProzess: anEditor

Ändern des Standardteils des Editors

Der neue Editor wird der Instanzvariablen aLocalProzess zugewiesen.

anEditor:

Neuer Kleiner Standard-Prozess-Editor

readOnly

Zugriff auf die Instanzvariable readOnly

Die Instanzvariable readOnly zeigt an, ob der Editor Änderungen des Prozesses zulässt (false) oder nicht (true). readOnly wird zurückgegeben.

readOnly: aBoolean

Ändern der Instanzvariablen readOnly

Die Instanzvariable readOnly zeigt an, ob der Editor Änderungen des Prozesses zulässt (false) oder nicht (true). readOnly wird auf aBoolean gesetzt und zurückgegeben.

aBoolean:

Boolscher Wert

template

Zugriff auf die Instanzvariable *template*

Die Instanzvariable *template* zeigt an, ob der Editor für die Prozessinitialisierung geöffnet wurde (*true*). *template* wird zurückgegeben.

template: aBoolean

Ändern der Instanzvariablen *template*

Die Instanzvariable *template* zeigt an, ob der Editor für die Prozessinitialisierung geöffnet wurde (*true*). *template* wird auf *aBoolean* gesetzt und zurückgegeben.

aBoolean:

Boolscher Wert

Kategorie actions

closeRr

Schließen des Fensters (Close-Button)

Die Methode war zuvor bei den einzelnen Editor-Klassen implementiert.

Der Close-Button dient hier nicht nur dem Schließen des Fensters, sondern auch der Speicherung der Prozessdaten in der Prozessliste des jeweiligen Fragments. Daher wird der Benutzer, falls der Editor nicht nur zum Lesen geöffnet wurde, gefragt, ob er eine Speicherung wünscht. Wenn das der Fall ist, erfolgt eine Umsetzung von Prozessobjekten (*save*). Die eigentliche Speicherung in der Datenbank geschieht an zentraler Stelle beim Zurückschreiben des Fragments.

Das Fenster wird zum Schließen aufgefordert.

save

Speichern des Prozessobjekts

Die Aufgabe wird an die gleichnamige Methode des kleinen Standard-Prozesseditors delegiert.

startELCalculator

Starten des Taschenrechners

Zunächst wird bei den Controllern nach einem Controller für einen Elab-Taschenrechner gesucht, der demselben Benutzer gehört wie der Editor. Wenn einer gefunden wird, wird das entsprechende Fenster geöffnet, falls es als Icon vorliegt. Das Fenster wird in den Vordergrund geholt.

Falls kein Controller gefunden wird, wird ein neuer Taschenrechner für den betreffenden Benutzer als Kind des vorliegenden Prozesseditors geöffnet.

Kategorie aspects

aLocalProzess

This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

Kategorie changing

changeRequest

Aktionen bei Änderungen, die den Editor betreffen

Der kleine Standard-Prozesseditor und das Objekt selbst erhalten die Nachricht `retractYourInterest`; die abhängigen Objekte erhalten die Nachricht `updateRequest`. Deren Rückgabewert wird zurückgegeben.

Kategorie events

noticeOfWindowClose: aWindow

Aufräumarbeiten, die beim Schließen des Fensters automatisch durchgeführt werden sollen

Falls ein Editor zum Schreiben geöffnet war, werden dessen Prozessobjekte umgesetzt. Dadurch wird der Zustand beim Öffnen bzw. beim letzten expliziten Speichern hergestellt. Falls der Editor für die Prozessinitialisierung benutzt worden ist, wird der Prozess in die Template-Datenbank zurückgeschrieben. Falls beim kleinen Standard-Prozesseditor ein Knoten gespeichert ist, der Prozess also aus einem Protokoll stammt, wird dieser Knoten über das Schließen des Editors informiert.

Die Methode der Superklasse wird aufgerufen.

aWindow:

Das Fenster, das geschlossen wird.

Kategorie initialize

initialize

Initialisierung

Die Instanzvariablen der Superklasse werden initialisiert. Die Instanzvariable `template` wird auf `false` gesetzt.

Kategorie interface opening

postBuildWith: aBuilder

Durchführung von Aktionen nach dem Öffnen des Fensters

Zunächst erfolgt eine Sonderbehandlung für den Fall, dass es sich beim aufrufen-
den Model um einen NEDT TreeNode handelt.

Falls der neu geöffnete Editor der Prozessinitialisierung dient, werden das Label
und die Farben geändert, ansonsten wird nur das Label geändert.

Falls der Editor keine Änderungen des Prozesses zulassen soll, werden die Kom-
ponenten des Fensters deaktiviert.

aBuilder:

Builder des Fensters

Kategorie parents

mainWindow

Ermitteln des Hauptfensters von Elab

Prozesseditoren werden u.a. auch zum Editieren von Prozessen in Protokollen
benutzt. Wegen der unterschiedlichen Programmierung der Chargensplitting-
Workfloweditoren, in denen Protokolle dargestellt werden, kann dort für die
Ermittlung des Hauptfensters vom Prozesseditor aus nicht über eine Hierarchie
von Fenstern gegangen werden (Methode mainWindow der Klasse ElabApplica-
tionModel).

Für alle Prozesseditoren wird hier das Hauptfenster durch Suchen des Controllers
mit passendem Bezeichner für Sperren des Benutzers des Models ermittelt. Falls
kein Controller gefunden wurde, wird nil zurückgegeben, ansonsten das Model
des Controllers.

Kategorie instance creation (class)

newUser: aUser

Erzeugung eines neuen Editors mit einem bestimmten Benutzer

Der neue Editor wird mit dem angegebenen Benutzer erzeugt. Für seinen Stan-
dardteil wird ein kleiner Standard-Prozess-Editor mit dem angegebenen Benutzer
erzeugt. Der neue Editor wird zurückgegeben.

aUser:

Benutzer

newUser: aUser forTemplate: aBoolean

Erzeugung eines neuen Editors mit einem bestimmten Benutzer und Verwendungszweck

Der neue Editor wird mit dem angegebenen Benutzer erzeugt. Die Instanzvariable *template*, die angibt, ob der Editor für die Prozessinitialisierung geöffnet wurde, wird auf *aBoolean* gesetzt. Der neue Editor wird zurückgegeben.

aUser:

Benutzer

aBoolean:

Boolscher Wert

newUser: aUser readOnly: aBoolean

Erzeugung eines neuen Editors mit einem bestimmten Benutzer und Verwendungszweck

Der neue Editor wird mit dem angegebenen Benutzer erzeugt. Die Instanzvariable *readOnly*, die angibt, ob der Editor Änderungen des Prozesses zulässt (*false*) oder nicht (*true*), wird auf *aBoolean* gesetzt. Der neue Editor wird zurückgegeben.

aUser:

Benutzer

aBoolean:

Boolscher Wert

Kategorie interface opening (class)

forTemplateWith: aProcess caller: aModel

Öffnen eines Prozesseditors speziell für die Prozessinitialisierung

Es wird ein neuer Editor mit dem Benutzer des aufrufenden Models und für den bestimmten Verwendungszweck erzeugt und initialisiert. Der neue Editor wird geöffnet und sein Builder als Kind beim aufrufenden Model eingetragen.

Normalerweise wird der Builder bei einer 'open'-Methode zurückgegeben, hier jedoch nicht. Diese Nachricht wird nicht vom aufrufenden Model sondern vom Prozess gesendet. Daher wird die Eintragung als Kind auch von hier aus und nicht wie üblich vom Sender vorgenommen.

aProcess:

Prozess, der dargestellt werden soll

aModel:

Model, von dem aus geöffnet wird

with: aProcess caller: aModel

Öffnen eines Prozesseditors

Es wird ein neuer Editor mit dem Benutzer des aufrufenden Models erzeugt und initialisiert. Der neue Editor wird geöffnet und sein Builder als Kind beim aufrufenden Model eingetragen.

Normalerweise wird der Builder bei einer 'open'-Methode zurückgegeben, hier jedoch nicht. Diese Nachricht wird nicht vom aufrufenden Model sondern vom Prozess gesendet. Daher wird die Eintragung als Kind auch von hier aus und nicht wie üblich vom Sender vorgenommen.

aProcess:

Prozess, der dargestellt werden soll

aModel:

Model, von dem aus geöffnet wird

with: aProcess caller: aModel node: aNode readOnly: aBoolean

Öffnen eines Prozesseditors

Es wird ein neuer Editor mit dem Benutzer des aufrufenden Models erzeugt und initialisiert. Mit readOnly wird angegeben, ob der Editor Änderungen des Prozesses zulässt (false) oder nicht (true). Beim neuen Editor wird der zum Prozess gehörige Knoten auf das angegebene Argument gesetzt. Der neue Editor wird geöffnet und sein Builder als Kind beim aufrufenden Model eingetragen.

Normalerweise wird der Builder bei einer 'open'-Methode zurückgegeben, hier jedoch nicht. Diese Nachricht wird nicht vom aufrufenden Model sondern vom Prozess gesendet. Daher wird die Eintragung als Kind auch von hier aus und nicht wie üblich vom Sender vorgenommen.

aProcess:

Prozess, der dargestellt werden soll

aModel:

Model, von dem aus geöffnet wird

aNode:

Bei Protokollen der zum Prozess gehörige Knoten

aBoolean:

Boolscher Wert

with: aProcess caller: aModel readOnly: aBoolean

Öffnen eines Prozesseditors

Es wird ein neuer Editor mit dem Benutzer des aufrufenden Models erzeugt und initialisiert. Mit readOnly wird angegeben, ob der Editor Änderungen des Prozesses zulässt (false) oder nicht (true). Der neue Editor wird geöffnet und sein Builder als Kind beim aufrufenden Model eingetragen.

Normalerweise wird der Builder bei einer 'open'-Methode zurückgegeben, hier jedoch nicht. Diese Nachricht wird nicht vom aufrufenden Model sondern vom

Prozess gesendet. Daher wird die Eintragung als Kind auch von hier aus und nicht wie üblich vom Sender vorgenommen.

aProcess:

Prozess, der dargestellt werden soll

aModel:

Model, von dem aus geöffnet wird

aBoolean:

Boolscher Wert

Kategorie resources (class)

Calculator

restartEleLabWin

Klasse ElabWorkspace

Diese Klasse implementiert den Workspace, der von den Systementwicklern benutzt wird. Sie unterscheidet sich nur geringfügig von ihrer Oberklasse.

Oberklasse:

ComposedTextView

Kategorie updating

updateRequest

Aktionen beim Schließen des Workspace

Falls der Benutzer eine Sicherung des Workspaces wünscht, wird dessen Inhalt in der Datei 'workspace.txt' abgelegt. Die Nachricht wird an die Oberklasse gesendet.

Klasse Login

Diese Klasse ist das Model für das Fenster zum Anmelden beim System Elab. Es erbt direkt von der Klasse ApplicationModel und nicht von ElabApplicationModel.

*Oberklasse:
ApplicationModel*

Instanzvariablen:

*entry:
Ein Symbol, dass beim Schließen des Fensters ausgewertet wird und angibt, ob die Anmeldung erfolgreich war, also Zugang gewährt wird oder ob abgebrochen wird. Wird Zugang gewährt, ist das Symbol #ok, ansonsten #undefined.*

*user:
Benutzer.*

*userName:
Angegebener Benutzername.*

Klassenvariablen:

*SaveEnabled:
Das Login-Fenster enthält einen Button zum Sichern, der von Superusern benutzt werden kann. Diese Variable gibt an, ob ein Superuser gerade Entwicklungsarbeiten durchführt und evtl. den Sicherheits-Button benutzen möchte (true) oder nicht (false).*

Kategorie accessing

entry

Zugriff auf die Instanzvariable entry
entry wird zurückgegeben.

entry: aSymbol

Setzen der Instanzvariablen entry
entry wird auf aSymbol gesetzt und zurückgegeben.

*aSymbol:
Wert, auf den entry gesetzt werden soll*

user

Zugriff auf den Benutzer
Der Benutzer wird zurückgegeben.

user: aUser

Setzen des Benutzers
Die Instanzvariable user wird auf den übergebenen Benutzer gesetzt und zurückgegeben.

aUser:
Ein Benutzer

userName: aName

Ändern des angezeigten Benutzernamens

Der Valueholder für den Benutzernamen übernimmt den neuen Wert und wird zurückgeliefert.

aName:
Neuer Name

Kategorie actions

cancel

Abbruch des Logins (Button 'Cancel')

Das Objekt selbst wird zum Schließen aufgefordert.

login

Einloggen eines Benutzers in das System Elab

Zunächst werden alle Komponenten des Fensters gesperrt, damit bis zum Erscheinen des Passwort-Eingabefensters keine weiteren Eingaben vom Benutzer mehr angenommen werden.

Anschließend wird der derzeit gültige Pfadname für das Verzeichnis der Datenbanken ermittelt. Falls dies nicht möglich ist, wird das Login-Fenster geschlossen und damit die Anmeldung abgebrochen. Ferner wird das Vorhandensein dieses Verzeichnisses sowie die darin enthaltene Benutzer- und die Login-Datenbank geprüft. Falls hier ein Fehler festgestellt wurde, wird die Anmeldung ebenfalls abgebrochen. Bei Abbruch wird der Status entry auf das Symbol #databaseError gesetzt. Die Ermittlung des Datenbank-Verzeichnisses und die Überprüfung der Datenbanken erfolgt nur, falls als Benutzername nicht 'Install' eingegeben wurde.

Die Instanzvariable entry wird mit #undefined initialisiert. Falls als Benutzername die Zeichenkette 'Install' eingegeben wurde, erfolgt zunächst die Installation der Datenbanken. Falls von der zuständigen Methode nil zurückgeliefert wird, wurde der Vorgang abgebrochen. Die Anmeldung wird beendet und nil wird zurückgegeben. Im anderen Fall liefert die Installations-Methode einen Benutzernamen. Dieser Name ist dann Benutzername für den Anmeldevorgang bei den neu installierten Datenbanken.

Die Benutzer- und die Logindatenbank werden geöffnet.

In einer Transaktion wird nach dem Benutzer mit dem eingegebenen Namen gesucht. Falls er nicht gefunden wird oder schreibgeschützt ist, wird die Transaktion abgebrochen und der Benutzer informiert. Die Komponenten des Fensters werden wieder freigegeben.

Falls ein Benutzer gefunden wird, wird für die eigene Instanzvariable eine transiente Kopie angelegt. Das Benutzer-Objekt wird aufgefordert die Login-Daten in

der Login-Datenbank abzulegen und mit einem Read-Lock versehen. Nach Abschluss der Transaktion wird der Benutzer zur Eingabe des Passworts aufgefordert. Er hat drei Versuche. Falls das Passwort richtig ist, wird entry auf #ok gesetzt. Wird bei einem der drei Versuche vom Benutzer abgebrochen, wird entry nicht auf #ok gesetzt und die Schleife beendet. Nach der Passworteingabe wird das Login-Fenster geschlossen.

savelmage

Sichern des Images (Kleiner roter Button)

Der kleine rote Button im Login-Fenster dient zum Sichern des Images nach Beendigung von Entwicklungsarbeiten in VisualWorks durch den Benutzer. Es wird davon ausgegangen, dass beim Sichern nur ein einziges Login-Fenster geöffnet ist. In diesem Fall wird vor dem Sichern die Klassenvariable saveEnabled auf true gesetzt und der Button deaktiviert. Damit ist diese Funktionalität für den normalen Benutzer nicht erreichbar. Der Tastatur-Fokus wird auf das Eingabefeld gesetzt. Der Pfad des Home-Verzeichnisses wird angefordert, ggf. um einen Separator ergänzt und in einer globalen Variablen gespeichert.

In jedem Fall werden alle Objekte der Klasse ElabLockDaemon zur Beendigung ihrer Aktualisierungsprozesse aufgefordert. Damit sind sie beim Sichern für den Garbage-Collector erreichbar.

Anschließend erfolgt das Speichern und das Verlassen von VisualWorks.

Der Button sollte nur zum endgültigen Sichern nach Schließen aller übrigen Fenster benutzt werden. Ein ordnungsgemäßes Sichern ist aber auch bei geöffneter Entwicklungsumgebung möglich. Sinnvoller ist hier aber die explizite Sicherung im Hauptfenster von VisualWorks, zumal damit nicht das Verlassen des gesamten Systems verbunden ist.

Kategorie aspects

userName

This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

Kategorie database

checkDB

Überprüfen Login-Datenbank

Das Verzeichnis der Datenbanken wird in eine neue globale Variable übernommen, die beim Prüfen auf Existenz von Datenbanken verwendet wird.

Dann wird geprüft, ob das Verzeichnis der Datenbanken vorhanden ist. Existiert es nicht, erfolgt eine Fehlermeldung an den Benutzer. Dieser kann dann entscheiden, ob er abrechen oder eine Installation der Datenbanken durchführen möchte. Bei Fortfahren wird der Benutzername auf 'Install' gesetzt. Dadurch wird unter

diesem Namen eine Anmeldung zur Installation der Datenbanken durchgeführt. Andernfalls wird nil zurückgegeben.

Weiter wird überprüft, ob eine Login- bzw. eine Benutzer-Datenbank unter dem gültigen Pfadnamen vorhanden ist. Wenn nicht, wird eine Fehlermeldung ausgegeben und nil zurückgegeben.

Alle geöffneten Datenbanken werden vorsorglich geschlossen.

Die Login-Datenbank ist zu überprüfen, ob ein Benutzer angemeldet ist, dessen Applikation beendet ist.

Die Login- und die Benutzer-Datenbank werden geöffnet. In einer Transaktion wird für jede Login-Nummer, zu der ein Benutzername eingetragen ist beim entsprechenden Benutzer in der Benutzer-Datenbank ermittelt, ob eine Lesesperre für diese Anmeldung existiert. Falls dies der Fall ist, ist die entsprechende Applikation noch nicht beendet und es liegt kein Fehler in der Login-Datenbank vor. Ist dies jedoch nicht der Fall, wird die entsprechende Nummer in der Login-Datenbank aktualisiert, d.h. die Nummer wird eingetragen und der zugehörige Benutzername entfernt.

getPathToDatabase

Ermittlung des gültigen Pfadnamens für das Verzeichnis der Datenbanken

Der Pfadname für das Verzeichnis der Datenbanken ist in einer Datei mit dem Namen 'elab.cfg' im Elab-Verzeichnis enthalten. Falls diese Datei nicht gefunden wird, wird zunächst angenommen, dass der Pfad für das Home-Verzeichnis nicht richtig gesetzt ist. Der Pfadname wird vom Benutzer angefordert und evtl. um einen Separator ergänzt.

Falls die Datei dann auch nicht gefunden wird, erfolgt eine Fehlermeldung und die Rückgabe von nil. Ansonsten wird der Pfadname für das Verzeichnis der Datenbanken aus 'elab.cfg' ausgelesen. Die entsprechende globale Variable wird gesetzt.

installDatabase

Installieren der Datenbanken des Elab-Systems

Die Nachricht wird gesendet, wenn sich ein Benutzer mit dem Namen 'Install' anmeldet. Im Elab-Verzeichnis müssen sich die Datenbanken 'users' und 'login' sowie die Konfigurationsdatei 'elab.cfg' befinden.

Zuerst wird der Pfadname des Home-Verzeichnisses angefordert und ggf. um einen Separator ergänzt.

Das Passwort von 'Install' muss bei jeder Installation geändert werden. Der Benutzer wird darüber informiert. Er hat die Möglichkeit, abzulehnen, wodurch der Vorgang unter Ausgabe einer Fehlermeldung und Zurückgeben von nil abgebrochen wird.

Ansonsten wird die Datenbank 'users' geöffnet. In einer Transaktion wird eine transiente Kopie des Benutzers 'Install' angelegt. Das Datenbank-Objekt wird mit einer Schreibsperre versehen. Es wird eine Kopie des alten Passworts gespeichert.

Der Benutzer muss das Passwort ändern. Falls das nicht ordnungsgemäß erfolgt, wird die Sperre beim Datenbank-Objekt wieder entfernt, der Zugang wird verweigert und nil zurückgegeben.

Ansonsten wird jeweils eine Kopie der alten Namen des Datenbank-Verzeichnisses gespeichert. Der Benutzer muss einen neuen Namen für dieses Verzeichnis angeben. Falls er den Separator im Namen nicht mit angibt, wird dieser explizit hinzugefügt. Das Verzeichnis der Datenbanken wird in eine weitere globale Variable übernommen, die beim Prüfen auf Existenz von Datenbanken verwendet wird. Falls das Verzeichnis bereits existiert, wird der Benutzer informiert. Wünscht er kein Überschreiben des bestehenden Verzeichnisses, wird er noch gefragt, ob er das vorhandene Verzeichnis akzeptiert. Der Pfadname wird entweder zurückgesetzt und nil zurückgegeben oder der neue Name wird übernommen und in der Konfigurationsdatei gespeichert.

Falls das Verzeichnis noch nicht existiert, wird ein neues Verzeichnis mit dem angegebenen Namen erzeugt. Falls auf die alten Namen zurückgesetzt wurde, besteht noch die Möglichkeit, dass hierzu kein Verzeichnis existiert, was geprüft werden muss. Ist dies der Fall, wird der Benutzer informiert und nil zurückgegeben.

Anschließend erfolgt die Erzeugung von dDaenbanken, es sei denn, es werden bereits bestehende Datenbanken übernommen.

Eine Login-Datenbank wird erzeugt. In einer Transaktion werden die beiden Felder in dieser Datenbank eingerichtet und anschließend die Datenbank geschlossen.

Eine Template-Datenbank wird erzeugt. Sie wird in einer Transaktion mit Objekten aller in der Kategorie 'Herstellungsprozesse' vorhandenen Klassen gefüllt. Anschließend wird die Datenbank geschlossen.

Eine Fragment- und eine Protokoll-Datenbank werden erzeugt.

Eine Benutzer-Datenbank wird erzeugt. In einer Transaktion wird ein Superuser mit Namen 'Super' und Projektgruppe 'Elab' erzeugt. Dies ist der erste Dummy-Benutzer in der Benutzer-Datenbank. Die Datenbank wird wieder geschlossen. Die Erzeugung des Superusers wird dem installierenden Benutzer mitgeteilt.

Der Pfadname des Datenbank-Verzeichnisses wird in der Konfigurationsdatei eingetragen.

Die Zeichenkette 'Super' als Name des Dummy-Benutzers wird zurückgegeben.

Bei Rückgabe von nil wird jeweils der Status entry auf das Symbol #installInterrupted gesetzt, damit beim Verlassen des Fensters keine Zugriffe auf Datenbanken stattfinden.

Kategorie events

noticeOfWindowClose: aWindow

Aufräumarbeiten, die beim Schließen des Fensters automatisch durchgeführt werden sollen

Bei nicht erfolgreichem Einloggen (entry hat nicht den Wert #ok) werden die Login- und die Benutzer-Datenbank geöffnet. Der Benutzer mit dem angegebenen Namen wird gesucht, falls nicht vorhanden, wird ein neu erzeugter Benutzer verwendet. Falls der Benutzer einen Namen hat, dh. er wurde gefunden, wird er aufgefordert, Locks zu löschen und die Login-Daten aus der Login-Datenbank zu entfernen. Die Zugriffe auf die Datenbanken erfolgen nicht bei einer abgebrochenen Installationsprozedur und bei nicht vorhandenen Datenbank-Verzeichnissen.

Geöffnete Datenbanken werden vorsorglich geschlossen.

Falls der Aktivierungszustand des Buttons zum Sichern im Login-Fenster false ist, d.h. die Entwicklungsumgebung ist nicht aktiviert, werden alle Objekte der Klasse ElabLockDaemon aufgefordert, ihre Aktualisierungsprozesse zu beenden und das gesamte System terminiert. Falls die Entwicklungsumgebung aktiviert ist, bleibt es beim Schließen der Elab-Fenster.

Bei erfolgreichem Einloggen wird das Elab-Hauptfenster geöffnet und der Benutzer dabei übergeben.

aWindow:

Das Fenster, das geschlossen wird.

Kategorie initialize

initialize

Initialisierung

Die Instanzvariable entry wird auf #undefined gesetzt und für user ein neuer Benutzer erzeugt.

Kategorie interface opening

postBuildWith: aBuilder

Aktionen nach dem Aufbau des Fensters

Falls der Aktivierungszustand für den Button zum Sichern nicht gesetzt ist, wird er auf false gesetzt. Falls er auf true gesetzt ist, wird der Button zum Sichern des Images aktiviert.

aBuilder:

Builder des Fensters

Kategorie private

disableAll

Sperren aller Komponenten des Fensters

Das Eingabefeld wird zum Schreiben gesperrt; die buttons werden deaktiviert.

enableAll

Freigabe aller Komponenten des Fensters

Das Eingabefeld wird zum Schreiben freigegeben; die Buttons werden aktiviert.

setTempPath

Setzen einer neuen globalen Variable für das Verzeichnis der Datenbanken

Das Verzeichnis der Datenbanken wird in eine neue globale Variable übernommen, die beim Prüfen auf Existenz von Datenbanken verwendet wird. Existiert dieses Verzeichnis nicht, wird alles bis zum ersten Auftreten eines Dateinamen-Trennzeichens aus dem String entfernt, falls nicht bereits das erste Zeichen ein solches Trennzeichen oder das Zeichen für den relativen Pfad des aktuellen Verzeichnisses ist. Damit wird der Fall behandelt, dass der Pfad mit zusätzlichem Rechnernamen angegeben ist.

Kategorie accessing (class)

saveEnabled

Zugriff auf den Aktivierungszustand des Buttons zum Sichern

Der Aktivierungszustand wird zurückgegeben. Der Wert der Variablen ist true, solange ein Superuser mit VisualWorks Entwicklungsarbeiten durchführt. Er ist false, wenn ein Image gesichert wird, bei dem lediglich ein Login-Fenster geöffnet ist. Dieses Image wird dann von einem normalen Benutzer verwendet.

saveEnabled: aBoolean

Setzen des Aktivierungszustands des Buttons zum Sichern

Der Aktivierungszustand wird auf aBoolean gesetzt zurückgegeben. Der Wert der Variablen ist true, solange ein Superuser mit VisualWorks Entwicklungsarbeiten durchführt. Er ist false, wenn ein Image gesichert wird, bei dem lediglich ein Login-Fenster geöffnet ist. Dieses Image wird dann von einem normalen Benutzer verwendet.

aBoolean:

Logischer Wert

Kategorie interface specs (class)

windowSpec

Klasse ObjectWithHistory

Es gibt Datenbankobjekte, die sich die Namen von denjenigen Objekten merken, aus denen sie durch Kopieren erzeugt wurden. Die Verwaltung dieser Namen wird von dieser Klasse implementiert. Außerdem ist hier die Projektgruppe gespeichert, zu der ein solches Objekt gehören kann.

Oberklasse:

DBObject

Instanzvariablen:

group:

Projektgruppe

history:

Eine Liste vom Typ OrderedCollection, die die Namen der Vorgängerobjekte enthält.

name:

Name des Objekts

Kategorie accessing

group

Zugriff auf die Projektgruppe

Die Projektgruppe wird zurückgegeben.

group: aName

Setzen der Projektgruppe

Die Projektgruppe wird auf das übergebene Argument gesetzt und zurückgegeben.

aName:

Neuer Projektgruppenname

history

Zugriff auf die Liste mit Namen älterer Versionen

Die Liste mit Versionsnamen wird zurückgegeben.

history: aCollection

Setzen der Liste mit Versionsnamen

Die Liste mit Versionsnamen wird auf das übergebene Argument gesetzt und zurückgegeben.

aCollection:

Neue Liste

name

Zugriff auf den Objektnamen

Der Name wird zurückgegeben.

name: aName

Setzen des Objektnamens

Der Name wird auf das übergebene Argument gesetzt und zurückgegeben.

aName:

Neuer Name

Kategorie copying

copy

Kopieren eines Objekts mit Versionen

Ein neues Objekt wird durch super copy erzeugt. Die Projektgruppe und der Name werden als Kopie der eigenen Instanzvariablen gesetzt. Die Liste der Versionen wird neu erzeugt und mit Kopien aus der eigenen Liste gefüllt. Das neue Objekt wird zurückgegeben.

createNewVersion: aName

Erzeugen einer neuen Version mit vorgegebenem Namen

Ein neues Objekt wird mittels copy erzeugt. Die Sperren werden dabei nicht mitkopiert. Der vorgegebene Name wird gesetzt. Der eigene Name wird in dessen Liste der Versionsnamen eingefügt. Das neue Objekt wird zurückgegeben.

aName:

Name für die neue Version

createNewVersion: aName group: aString

Erzeugen einer neuen Version mit vorgegebenem Namen und Projektgruppe

Ein neues Objekt wird mittels copy erzeugt. Die Sperren werden dabei nicht mitkopiert. Der vorgegebene Name und die Projektgruppe werden gesetzt. Der eigene Name wird in dessen Liste der Versionsnamen eingefügt. Das neue Objekt wird zurückgegeben.

aName:

Name für die neue Version

Kategorie history

addHistory: aName

Hinzufügen eines neuen Namens in die Liste mit Versionen

Der übergebene Name wird am Anfang in die Liste eingefügt. Zurückgegeben wird der Rückgabewert der Methode addFirst:.

aName:

Hinzuzufügender Name

removeHistory: aName

Entfernen eines Versionsnamens aus der Liste

Die Aufgabe wird an die Liste weiterdelegiert.

aName:

Name des zu entfernenden Objekts

Kategorie initialize

initialize

Initialisierung

Die Initialisierung der Superklasse wird durchgeführt. Die Projektgruppe wird auf den Default-Wert 'Undefined Project' gesetzt. Der Name wird mit einem leeren String initialisiert; für die Liste mit Versionsnamen wird eine neue OrderedCollection erzeugt.

Kategorie printing

printString

Ausgabe von Daten eines Objekts mit Versionen in einen String

Die Nachricht wird zunächst an die Oberklasse gesendet. Danach wird die Nachricht an die Liste mit den Versionsnamen gesendet. Außerdem wird die Projektgruppe ausgegeben. Alle Objekte schreiben auf einen lokalen String, der zurückgegeben wird.

Kategorie database (class)

listOfNamedObjects: aUser

Erzeugen einer Liste mit den Namen der Datenbank-Objekte, deren Projektgruppe der Benutzer angehört

Eine neue, leere Liste wird erzeugt. Die entsprechende Datenbank wird geöffnet. In einer Transaktion werden diejenigen Objekte, deren Projektgruppe der Benut-

zer angehört, in die Liste eingefügt. Nach der Transaktion wird die Liste zurückgegeben.

aUser:
Benutzer

openDB

Öffnen der Datenbank

Nur als Methode der Unterklassen.

Kategorie instance creation (class)

newName: aName

Erzeugung einer neuen Instanz mit Namen

Eine neue Instanz wird erzeugt. Der Name wird mit dem Wert des angegebenen Arguments initialisiert. Die neue Instanz wird zurückgegeben.

aName:
Name

newName: aName group: aString

Erzeugung einer neuen Instanz mit Namen und Projektgruppe

Das neue Objekt wird erzeugt. Name und Projektgruppe werden auf die übergebenen Werte gesetzt. Das neue Objekt wird zurückgegeben.

aName:
Name

aString:
Projektgruppe

Klasse Password

Die Klasse implementiert das Passwort eines Benutzers von Elab. Passwörter sind immer nur codiert gespeichert. Es gibt keine Decodierungsmöglichkeit.

Oberklasse:

ByteString

Kategorie changing

change

Ändern des Passworts

Der Benutzer wird aufgefordert sein Passwort einzugeben. Dieses wird dabei codiert und überprüft. Wenn es richtig ist, wird der Benutzer zur Eingabe des neuen Passworts aufgefordert, das ebenfalls codiert ist. Anschließend muss der Benutzer noch bestätigen, wobei die erneute Eingabe ebenfalls codiert ist und überprüft wird. Falls die Bestätigung falsch war, gibt es eine zweite Chance. Wenn die Bestätigung dann richtig ist, wird das neue Passwort als aktuelles Passwort übernommen, ansonsten wird der Benutzer über die Fehleingabe informiert und nil zurückgegeben.

Wenn bereits die Eingabe des alten Passworts falsch ist, wird der Benutzer auch informiert und nil zurückgegeben. Bei erfolgreichem Abschluss wird das Objekt selbst zurückgegeben.

Der Änderungsvorgang wird abgebrochen und nil zurückgegeben, falls bei einer der Passwort-Eingaben ein Abbruch durch den Benutzer erfolgt.

set

Setzen des Passworts

Der Benutzer wird zur Eingabe eines Passworts aufgefordert. Das eingegebene Passwort muss bestätigt werden, wofür zwei Versuche möglich sind. Falls eine der Eingaben abgebrochen wird, wird nil zurückgegeben. Ansonsten wird das eingegebene Passwort übernommen.

Kategorie copying

copy

Kopieren eines Passworts

Es wird ein neues Passwort mit der Größe des vorliegenden erzeugt. Die einzelnen Zeichen des vorliegenden Passworts werden in das neue kopiert, das dann zureuckgegeben wird.

Kategorie testing

check

Abfrage des Passworts

Der Benutzer wird zur Eingabe seines Passworts aufgefordert. Bei einem Abbruch durch den Benutzer wird nil zurückgegeben. Das Passwort wird dabei sofort codiert und überprüft. Falls es falsch ist wird der Benutzer informiert und false zureuckgegeben, ansonsten wird true zurückgegeben.

Kategorie user

requestForPassword: aString

Eingabe eines Passworts

Der Benutzer wird in einem Dialogfenster zur Eingabe eines Passworts aufgefordert. Falls eine Eingabe erfolgt, wird mit dem angegebenen String ein Passwort erzeugt, das dabei sofort codiert wird. Das Passwort wird zurückgegeben.

Falls die Eingabe abgebrochen wird, wird nil zurückgegeben.

wrongInput

Information des Benutzers über Fehleingabe

Darstellung eines Dialogfensters.

Kategorie instance creation (class)

fromString: aString

Erzeugen eines Passworts aus einem String

Es wird ein neues Passwort mit derselben Größe wie aString erzeugt. Der String wird Zeichen für Zeichen in das neue Passwort kopiert. Das neue Passwort wird codiert.

aString:

Passwort als normaler String

Klasse PasswordDialog

Die Klasse implementiert ein Dialogenster zur Passwordeingabe mit Timeout.

Oberklasse:
SimpleDialog

Instanzvariablen:

delayProcess:
Ein Prozess, der das Zeitlimit für die Passwordeingabe kontrolliert.

Kategorie accessing

delayProcess

Zugriff auf den Prozess zur Kontrolle des Zeitlimits für die Passwort-Eingabe
Der Prozess wird zurückgegeben.

delayProcess: aProcess

Setzen des Prozesses zur Kontrolle des Zeitlimits bei der Passwort-Eingabe
Der angegebene Prozess wird übernommen und die Instanzvariable *delayProcess* zurückgegeben.

aProcess:
Prozess

Kategorie events

noticeOfWindowClose: aWindow

Aktionen beim Schließen des Fensters
Falls ein Prozess zur Kontrolle bei der Passwort-Eingabe existiert, wird er beendet. Die Nachricht wird an die Oberklasse gesendet.

aWindow:
Fenster zur Passwort-Eingabe

Kategorie interface construction

addTextLine: model

Hinzufügen eines Text-Eingabefeldes
Die Methode ist eine Erweiterung der entsprechenden Methode der Klasse *SimpleDialog*, die den Typ des Eingabefeldes auf *#password* setzt, so dass nicht die eingegebenen Zeichen, sondern jeweils das Zeichen '*' angegeben wird.

model:
ValueHolder für eine Zeichenkette

Kategorie utility

request: messageString initialAnswer: aString onCancel: aBlockOrNil

Öffnen des Dialogs

Ein Prozess zur Kontrolle des Zeitlimits bei der Passwort-Eingabe wird erzeugt. Dieser Prozess wartet 15 s und schließt dann das Eingabefenster wieder. Die Priorität des Prozesses wird gesetzt und der Prozess gestartet. Die Nachricht wird an die Oberklasse gesendet.

messageString:

Überschrift

aString:

Standardeintrag im Eingabefeld

aBlockOrNil:

Aktionen beim Abbruch

Klasse User

Diese Klasse implementiert einen Benutzer des Elab-Systems. Sie erbt von DBObject, weil auch Benutzerobjekte in einer Datenbank gespeichert werden.

Oberklasse:

DBObject

Instanzvariablen:

access:

Zugriffsrecht des Benutzer innerhalb Elab:

#super Superuser

#fragment Normaler Benutzer

groups:

Ein Set mit den Namen der Projektgruppen, denen ein Benutzer angehört.

Initialisiert wird dieses Set mit einem Element 'Undefined Project'. Bei Hinzufügen eines richtigen Gruppennamens wird dieses Element automatisch gelöscht.

login:

Die Nummer, mit der der Benutzer beim System angemeldet ist.

name:

Name des Benutzers.

password:

Passwort des Benutzers.

Kategorie accessing

access

Zugriff auf das Zugriffsrecht

Das Zugriffsrecht wird zureckgegeben.

access: aSymbol

Ändern des Zugriffsrechts

Das Zugriffsrecht wird auf aSymbol gesetzt und zurückgegeben. Falls das Argument keines der beiden erlaubten Symbole ist, wird eine Fehlermeldung ausgegeben und das Zugriffsrecht auf #fragment gesetzt.

aSymbol:

Das neue Zugriffsrecht (#super oder #fragment)

groups

Zugriff auf die Projektgruppennamen

Das Set mit den Projektgruppennamen wird zureckgegeben.

groups: aSet

Ändern des Sets mit den Projektgruppennamen

Die Instanzvariable groups wird auf aSet gesetzt und zurückgegeben.

aSet:

Die neuen Namen

lockString

Rückgabe eines Strings der als Lock verwendet wird.

Der Lockstring wird aus dem Benutzernamen, dem Zeichen | und der Loginnummer zusammengestellt.

login

Zugriff auf die Loginnummer

Die Loginnummer wird zurückgegeben.

login: aNumber

Ändern der Loginnummer

Die Loginnummer wird auf die neue Nummer gesetzt und zurückgegeben.

aNumber:

Neue Loginnummer

name

Zugriff auf den Benutzernamen

Der Benutzername wird zurückgegeben.

name: aName

Ändern des Benutzernamens

Der Benutzername wird auf den neuen Namen gesetzt und zurückgegeben. Evtl. im Namen vorhandene Zeichen '|' werden entfernt.

aName:

Neuer Name

password

Zugriff auf das Passwort

Das Passwort - es ist immer codiert - wird zurückgegeben.

password: aPassword

Ändern des Passworts

Die Instanzvariable password wird auf das neue Passwort gesetzt. Wenn aPassword kein codiertes Passwort ist, funktioniert die Sache nicht. Irgendein normaler String ist hier nutzlos.

aPasswort:

Neues Passwort; dabei soll es sich um ein Objekt der Klasse Passwort, d,h, einen codierten String handeln.

Kategorie changing

changePassword

Ändern des Passworts

Die Aufgabe wird an die Instanzvariable password weitergereicht.

setPassword

Setzen des Passworts

Die aufgabe wird an die Instanzvariable password weitergereicht.

Kategorie copying

copy

Kopieren eines Benutzers

Ein neuer Benutzer wird durch super copy erzeugt. Für die Projektgruppennamen wird ein neues Set erzeugt, die Namen des alten Objekts werden hineinkopiert. Name und Passwort des neuen Benutzers werden als Kopie der eigenen Instanzvariablen gesetzt. Der neue Benutzer wird zurückgegeben.

Kategorie groups

addGroup: aName

Hinzufügen eines neuen Projektgruppennamens

Zunächst wird der Dummy-Name 'Undefined Project', falls vorhanden, entfernt. Anschließend wird aName dem Set groups hinzugefügt.

Solange Projektgruppennamen nur über diese Methode eingefügt werden, kann 'Undefined Project' nur alleine vorkommen, d.h. es sorgt dafür, dass groups nicht leer ist. 'Undefined Project' kann aber auch explizit zu anderen Namen hinzugefügt werden und bleibt dann solange im Set enthalten, bis ein weiterer Name hinzugefügt wird.

aName:

Neuer Name

firstGroup

Erstes Element des Sets mit Projektgruppen

Das erste Element des Sets mit Projektgruppen wird zurückgegeben.

removeGroup: aName

Entfernen eines Projektgruppennamens

Der angegebene Name wird aus dem Set groups entfernt.

aName:

Zu entfernender Name

showGroups

Anzeigen der Projektgruppen mit Auswahlmöglichkeit

Die Projektgruppen werden angezeigt. Die Auswahl des Benutzers oder nil, falls er abbricht, wird zurückgegeben.

singleGroup

Anfrage nach einzelner Projektgruppe

Falls das Objekt nur zu einer Projektgruppe gehört, wird deren Name zurückgegeben, ansonsten nil.

Kategorie initialize

initialize

Initialisierung

Die Initialisierung der Superklasse wird durchgeführt. Das Zugriffsrecht wird auf #fragment gesetzt. Für die Projektgruppennamen wird ein Set angelegt, das den Namen 'Undefined Project' enthält. Die Loginnummer ist als Default 0, der Name ein leerer String. Das Passwort wird aus dem leeren String codiert erzeugt.

Kategorie login

getLoginFrom: aDB

Übertragen der Login-Daten in die Login-Datenbank

Die Nachricht kann nur aus einer Transaktion heraus gesendet werden.

Eine Login-Datenbank enthält ein Array mit Nummern, die beim Login vergeben werden können und ein Array für Strings, in das die Namen von eingeloggten Benutzern geschrieben werden.

Falls die Nummern oder die Namen nicht vorhanden sind, also beispielsweise eine falsche Datenbank übergeben wurde, wird nil zurückgegeben.

Es wird im Nummern-Array nach einer freien (vorhandenen) Nummer gesucht. Wenn eine gefunden wird, wird sie durch nil ersetzt. Am selben Index im Namen-Array wird der Benutzername eingetragen. Der Benutzer erhält die gefundene Loginnummer.

Falls im Nummern-Array keine freien Nummern mehr aufzufinden sind, werden die beiden Arrays um 1 vergrößert. Die neue Nummer wird auf dieselbe Art wie oben beschrieben an den Benutzer vergeben.

Bei erfolgreicher Durchführung wird das Objekt selbst zurückgegeben.

aDB:

Hierbei sollte es sich um eine Login-Datenbank handeln.

removeLoginFrom: aDB

Entfernen von Login-Daten aus der Login-Datenbank

Die Nachricht kann nur aus einer Transaktion heraus gesendet werden.

Eine Login-Datenbank enthält ein Array mit Nummern, die beim Login vergeben werden können und ein Array für Strings, in das die Namen von eingeloggten Benutzern geschrieben werden.

Falls das Nummern- oder das Namen-Array nicht gefunden werden, also beispielsweise eine falsche Datenbank angegeben wurde, wird nil zurückgegeben. Ansonsten wird die Loginnummer an den entsprechenden Index im Nummern-Array und nil an denselben Index im Benutzerarray geschrieben.

Bei erfolgreicher Durchführung wird das Objekt selbst zurückgegeben.

aDB:

Hierbei sollte es sich um eine Login-Datenbank handeln.

Kategorie printing

printString

Ausgabe als String

Die Methode der Superklasse wird erweitert um die Ausgabe des Benutzernamens und der Projektgruppen.

Kategorie testing

areYou: aName

Frage nach dem Namen des Benutzers

Vergleicht den angegebenen Namen mit dem des Objekts und gibt true oder false zurück.

aName:

Name

checkPassword

Passwort-Abfrage

Die Aufgabe wird an das Passwort delegiert.

includesGroup: aName

Anfrage nach der Zugehörigkeit zu einer bestimmten Projektgruppe

Das Ergebnis der entsprechenden Anfrage beim Set groups wird zurückgegeben.

aName:

Name der Projektgruppe

isLoggedIn

Test, ob Benutzer angemeldet

Ein Benutzer, der beim System angemeldet ist, hat eine Login-Nummer.

Kategorie instance creation (class)

newName: aName

Erzeugen eines neuen Benutzers mit Übergabe des Namens

Ein neuer Benutzer wird erzeugt, der Name gesetzt und der Benutzer zurückgegeben.

aName:

Name des neuen Benutzers

newName: aName password: aPassword access: aSymbol

Erzeugen eines neuen Benutzers mit Übergabe des Namens, des Passworts und des Zugriffsrechts

Ein neuer Benutzer wird erzeugt, die drei Instanzvariablen gesetzt und der Benutzer zurückgegeben.

aName:

Name des neuen Benutzers

aPassword:

Passwort des neuen Benutzers

aSymbol:

Zugriffsrecht des neuen Benutzers (#fragment oder #super)

newName: aName password: aPassword access: aSymbol group: aString

Erzeugen eines neuen Benutzers mit Übergabe des Namens, des Passworts, des Zugriffsrechts und einer Projektgruppe

Ein neuer Benutzer wird erzeugt, der angegebene Projektgruppenname wird durch Hinzufügen zum Set groups übernommen, die drei Instanzvariablen werden gesetzt und der Benutzer zurückgegeben.

aName:

Name des neuen Benutzers

aPassword:

Passwort des neuen Benutzers

aSymbol:

Zugriffsrecht des neuen Benutzers (#fragment oder #super)

aString:

Name der Projektgruppe

Kategorie login (class)

currentlyLoggedIn

Ermitteln der aktuell eingeloggten Benutzer

Die Login-Datenbank wird geöffnet. Diese Datenbank enthält ein Array mit nil-Objekten oder Namen von eingeloggten Benutzer. Aus diesem Array werden Kopien aller vorhandenen Namen in einem Set gesammelt. Das Set wird zurückgegeben.

Klasse UserModel

Diese Klasse beschreibt das Model eines Fensters zur Darstellung von Benutzerdaten.

Oberklasse:

ElabApplicationModel

Instanzvariablen:

access:

Objekt der Klasse ValueHolder zur Aufnahme des Zugriffsrechts des dargestellten Benutzers.

accessOnUser:

Ein Symbol, das die gewünschte Zugriffsart auf die Benutzerdaten angibt:

#readOwner *Der angemeldete Benutzer lässt seine eigenen Daten anzeigen*

#read *Lesender Zugriff*

#edit *Zugriff zum Editieren*

#write *Editieren eines neu zu erzeugenden Benutzers*

groups:

Objekt der Klasse SelectionInList zur Aufnahme der Projektgruppennamen des dargestellten Benutzers.

name:

Name des Benutzers.

userToDisplay:

Der dargestellte Benutzer.

Kategorie accessing

access: aSymbol

Ändern des angezeigten Werts des Zugriffsrechts

Der Valueholder access übernimmt den neuen Wert und wird zurückgeliefert.

aSymbol:

Neuer Wert des Zugriffsrechts

accessOnUser

Zugriff auf die Zugriffsart der Benutzerdaten

Die Instanzvariable accessOnUser wird zurückgegeben.

accessOnUser: aSymbol

Ändern der Zugriffsart auf die Benutzerdaten.

accessOnUser übernimmt den neuen Wert und wird zurückgegeben. Falls das Argument keines der erlaubten Symbole ist, wird eine Fehlermeldung ausgegeben und die Zugriffsart auf #read gesetzt.

aSymbol:

Bezeichnet die gewünschte Zugriffsart (#readOwner, #read, #edit oder #write)

groups: anOrderedCollection

Ändern der darzustellenden Liste der Projektgruppennamen

Die Liste von groups (Klasse SelectionInList) wird auf anOrderedCollection gesetzt. groups wird zurückgegeben.

anOrderedCollection:

Neue Liste mit Namen

name: aName

Ändern des angezeigten Benutzernamens

Der Valueholder name übernimmt den neuen Namen und wird zurückgegeben.

aName:

Neuer Name

userToDisplay

Zugriff auf den Benutzer, der dargestellt wird

Die Instanzvariable userToDisplay wird zurückgegeben. userToDisplay nicht mit dem Benutzer, dem das Fenster gehört - user aus der Superklasse - verwechseln!

userToDisplay: aUser

Ändern des Benutzers, der dargestellt wird

Die Instanzvariable userToDisplay übernimmt den neuen Benutzer und wird zurückgegeben. userToDisplay nicht mit dem Benutzer, dem das Fenster gehört - user aus der Superklasse - verwechseln!

aUser:

Neuer Benutzer

Kategorie actions

addGroup

Hinzufügen einer neuen Projektgruppe beim dargestellten Benutzer (Button 'Add Group')

Der Benutzer wird nach dem Namen der neuen Gruppe gefragt. Falls es diese Gruppe bereits gibt oder kein Name angegeben wurde (etwa durch Abbruch der Namenseingabe), wird nil zurückgegeben. Ansonsten wird der neue Name in die Liste der Projektgruppennamen des dargestellten Benutzers eingefügt und die Liste im Fenster aktualisiert.

cancel

Abbruch der Benutzerdarstellung (Button 'Cancel')

Das Objekt selbst wird zum Schließen aufgefordert. Der Button ist nur bei änderndem Zugriff aktiv.

quit

Beenden der Benutzerdarstellung (Button 'OK')

Die dargestellten Benutzerdaten werden in der Datenbank gespeichert. `saveUser` macht das nur, falls ändernder Zugriff vorliegt. Falls beim Speichern ein Fehler auftritt, wird das Objekt selbst zurückgegeben und es passiert nichts, ansonsten wird das Model selbst zum Schließen aufgefordert.

removeGroup

Entfernen einer Projektgruppe beim dargestellten Benutzer (Button 'Remove Group')

Die letzte Gruppe kann nicht entfernt werden. Es erfolgt eine Warnmeldung an den Benutzer und `nil` wird zurückgegeben. Falls kein Gruppenname selektiert ist, geschieht das Gleiche. Der Benutzer muss das Löschen bestätigen. Erfolgt diese Bestätigung, wird der Gruppenname aus dem Set des dargestellten Benutzers gelöscht. Die im Fenster dargestellte Liste wird aktualisiert.

setAccess

Übernahme eines angezeigten Zugriffsrechts in die Benutzerdaten.

Das Zugriffsrecht des dargestellten Benutzers wird auf den Wert des Valueholders `access` gesetzt. Falls das Zugriffsrecht nicht erlaubt ist, wird es vom Benutzerobjekt korrigiert und korrigiert auch zurückgeliefert. Deshalb wird der Rückgabewert auch als Wert des Valueholders übernommen. An den ValueHolder wird eine Änderungsnachricht gesendet. Das Zugriffsrecht wird zurückgegeben.

setName

Übernahme eines angezeigten Namens in die Benutzerdaten.

Der Name des dargestellten Benutzers wird auf den Wert des Valueholders `name` gesetzt. Falls der Name nicht erlaubt ist, wird er vom Benutzerobjekt korrigiert und korrigiert auch zurückgeliefert. Deshalb wird der Rückgabewert auch als Wert des Valueholders übernommen. An den ValueHolder wird eine Änderungsnachricht gesendet. Der Name wird zurückgegeben.

setPassword

Setzen des Passworts (Button 'Set Password')

Die Aufgabe wird an den dargestellten Benutzer delegiert. Der Button ist nur bei der Zugriffsart `#write` aktiv. Damit kann ein Passwort nur beim Erzeugen eines Benutzers gesetzt werden.

Kategorie aspects

access

This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

groups

This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

name

This method was generated by UIDefiner. Any edits made here may be lost whenever methods are automatically defined. The initialization provided below may have been preempted by an initialize method.

Kategorie changing

switchDefault

Umschalten auf Darstellung des Besitzers des Fensters

Die Darstellung des alten Benutzers wird beendet, die Zugriffsart auf #readOwner gesetzt und der Besitzer des Fensters auch als darzustellender Benutzer eingetragen. Die Darstellung an der Oberfläche wird veranlasst. Der Builder wird zurückgegeben.

switchDisplay

Ändern der Oberfläche bei neuem Benutzer

Einzelne Oberflächenelemente werden mit postBuildWith: gesperrt bzw. freigegeben, wie auch beim Öffnen des Fensters. Die Darstellung von Zugriffsrecht, Projektgruppennamen und Namen des aktuellen Benutzers wird veranlasst.

switchToAccess: aSymbol onName: aName

Umschalten der Darstellung auf neuen Benutzer

Für die Zugriffsart #readOwner wird selfDefault gesendet. Für alle anderen Zugriffsarten wird zunächst die Darstellung des alten Benutzers geändert. Durch Senden von refresh:name: werden die Daten des gewünschten Benutzers bereitgestellt. Wenn dabei ein Fehler auftritt, wird der Besitzer dargestellt und der Benutzer durch ein Dialogfenster darüber informiert.

Die Darstellung an der Oberfläche wird veranlasst. Der Builder wird zurückgegeben.

aSymbol:

Zugriffsart auf den neuen Benutzer

aName:
Name des neuen Benutzers

Kategorie events

noticeOfWindowClose: aWindow

Aufräumarbeiten, die beim Schließen des Fensters automatisch durchgeführt werden sollen

Locks auf dem aktuellen Benutzer werden entfernt. Ansonsten wird dieselbe Nachricht an die Superklasse gesendet.

aWindow:
Das Fenster, das geschlossen wird.

Kategorie initialize

initialize

Initialisierung

Dieselbe Nachricht wird zunächst an die Superklasse gesendet.

access, userToDisplay und name werden gesetzt.

Kategorie interface opening

postBuildWith: aBuilder

Durchführung von Aktionen nach dem Öffnen des Fensters

Wenn die Zugriffsart #write oder #edit ist, werden der Cancel-Button, der Button 'Add Group' und der Button 'Remove Group' aktiviert und das Feld für das Zugriffsrecht kann beschrieben werden. Der Button 'Set Password' wird deaktiviert, das Feld für den Benutzernamen wird gesperrt. Falls die Zugriffsart #write ist, wird zusätzlich der Button 'Set Password' aktiviert und das Feld für den Namen beschreibbar gemacht.

Falls die Zugriffsart #read oder #readOwner ist, werden die Buttons deaktiviert und die Felder sind nur lesbar.

aBuilder:
Builder des Fensters

Kategorie private

getUser: aName

Auffinden eines Benutzers

Falls schreibender Zugriff besteht, d.h. ein neuer Benutzer soll erzeugt werden, wird ein neuer Benutzer mit dem vorgegebenen Namen erzeugt und als darzustellender Benutzer übernommen.

Ansonsten wird die Benutzer-Datenbank geöffnet. Der Benutzer mit dem vorgegebenen Namen wird in einer Transaktion aus der Datenbank geholt. Falls er darin nicht enthalten ist, wird die Transaktion abgebrochen. Falls nur lesender Zugriff auf den Benutzer besteht, wird das Datenbankobjekt mit einem Read-Lock versehen, falls das nicht möglich ist, wird die Transaktion abgebrochen.

Bei nicht-lesendem Zugriff wird die Transaktion abgebrochen, wenn der vorgegebene Name und der Name des Besitzers des vorliegenden UserModels identisch ist (Ein Benutzer darf sich nicht selbst editieren) oder wenn keine Schreibsperre gesetzt werden kann.

Bei erfolgreichem Verlauf der Transaktion wird eine transiente Kopie des Datenbankobjekts als darzustellender Benutzer übernommen.

Bei Abbruch der Transaktion wird der Benutzer entsprechend informiert und nil zurückgegeben.

aName:

Name des Benutzers

refresh: aSymbol name: aName

Aktionen bei der Umstellung des Models auf einen neuen Benutzer

Das Zugriffsrecht des Models wird neu gesetzt. Der Benutzer mit dem vorgegebenen Namen wird in das Model geladen und der Rückgabewert der Methode `getUser`: zurückgegeben.

aSymbol:

Neues Zugriffsrecht

aName:

Name des neuen Benutzers

releaseLock

Entfernen einer Sperre beim Datenbankobjekt des dargestellten Benutzers

Aktionen werden nur durchgeführt, wenn es sich beim dargestellten Benutzer nicht um den Besitzer des Models handelt, der lediglich Informationen über sich selbst erhalten wollte. In diesem Fall besitzt das Datenbankobjekt nur eine Lesesperre, die beibehalten werden muss, weil der Benutzer selbst noch angemeldet ist und daher ein transientes Objekt geladen ist.

Zunächst wird die Benutzer-Datenbank geöffnet. In einer Transaktion erfolgt das Beenden der Verbindung zum Datenbank-Objekt. Falls der Besitzer des UserMo-

dels selbst dargestellt war (ohne `accessOnUser = #readOwner`) wird hier noch einmal eine neue Lesesperre gesetzt.

saveUser

Speichern des dargestellten Benutzers

Die Benutzer-Datenbank wird geöffnet.

Falls schreibender Zugriff auf den dargestellten Benutzer besteht (Benutzer ist neu erzeugt worden), wird in einer Transaktion zunächst überprüft, ob ein Benutzer gleichen Namens bereits in der Datenbank existiert. In diesem Fall wird die Transaktion abgebrochen. Andernfalls wird eine Kopie in der Datenbank abgelegt.

Falls die Transaktion abgebrochen wurde, wird der Benutzer informiert, die Datenbank geschlossen und nil zurückgegeben.

Bei Zugriff zum Editieren, wird das transiente Benutzer-Objekt in die Datenbank zurückgeschrieben.

stop

Aktionen beim Beenden einer Darstellung zwecks Umschaltens auf einen anderen darzustellenden Benutzer

Zunächst wird der Eingabefokus auf das vorliegende Fenster gesetzt, damit neu eingetragene Werte übernommen werden. Falls der Zugriff auf den dargestellten Benutzer nicht-lesend ist, muss der Benutzer entscheiden, ob das Objekt in der Datenbank abgespeichert werden soll. Wird das gewünscht und ist ein Speichern nicht möglich, erfolgt eine Meldung an den Benutzer.

Falls `accessOnOwner` nicht auf `#readOwner` gesetzt ist, wird die Sperre beim Datenbankobjekt entfernt.

Kategorie instance creation (class)

newParent: parentModel user: owner access: aSymbol userToDisplay: aUser

Erzeugen eines neuen Models mit Übernahme von erzeugendem Model, Besitzer, Zugriffsart und darzustellendem Benutzer

Ein neues `ApplicationModel` mit erzeugendem Model wird erzeugt. Besitzer, Zugriffsart und darzustellender Benutzer werden gesetzt und das Model zurückgegeben.

parentModel:

Das erzeugende Model

user:

Besitzer des Models

access:

Zugriffsart (#readOwner, #read, #edit oder #write)

userToDisplay:
Darzustellender Benutzer

Kategorie interface opening (class)

openOnUser: aUser

Erzeugen und Öffnen eines neuen Models mit Besitzer und erzeugendem Model
Ein neues Model wird dem erzeugenden Model erzeugt, der Besitzer wird gesetzt.
Die Zugriffsart wird auf #readOwner gesetzt und der Besitzer auch als darzustellender Benutzer übernommen. Der Name im Model wird auf den Namen des Benutzers gesetzt. Das Model wird geöffnet und sein Builder zurückgegeben.

aUser:
Besitzer

openOnUser: aUser withAccess: aSymbol onName: aName

Erzeugen und Öffnen eines neuen Models mit Besitzer, Zugriffsart, Name des darzustellenden Benutzers und erzeugendem Model

Falls die Zugriffsart #readOwner ist, wird ein Model nur mit Besitzer erzeugt und geöffnet und der Builder zurückgegeben.

Bei anderen Zugriffsarten wird ein neues Model, Besitzer und Zugriffsart werden gesetzt. Zum angegebenen Namen wird ein Benutzer ermittelt. Falls es dabei zu einem Fehler kommt, wird der Besitzer über ein Dialogfenster darüber informiert, dass der Besitzer dargestellt wird. Es wird ein Model mit Besitzer und geöffnet und dessen Builder zureuckgegeben. Ohne Fehler wird der Name im Model auf den Namen des dargestellten Benutzers gesetzt, das Model geöffnet und sein Builder zurückgegeben.

aUser:
Besitzer

aSymbol:
Zugriffsart

aName:
Name des darzustellenden Benutzers

openOnUser: aUser withAccess: aSymbol onName: aName withParent: parentModel

Erzeugen und Öffnen eines neuen Models mit Besitzer, Zugriffsart, Name des darzustellenden Benutzers und erzeugendem Model

Falls die Zugriffsart #readOwner ist, wird ein Model nur mit Besitzer und erzeugendem Model erzeugt und geöffnet und der builder zurückgegeben.

Bei anderen Zugriffsarten wird ein Model mit erzeugendem Model erzeugt, Besitzer und Zugriffsart werden gesetzt. Zum angegebenen Namen wird ein Benutzer ermittelt. Falls es dabei zu einem Fehler kommt, wird der Besitzer über ein Dialogfenster darüber informiert, dass der Besitzer dargestellt wird. Es wird ein

Model mit Besitzer und erzeugendem Model erzeugt und geöffnet und dessen Builder zureuckgegeben. Ohne Fehler wird der Name im Model auf den Namen des dargestellten Benutzers gesetzt, das Model geöffnet und sein Builder zurückgegeben.

aUser:

Besitzer

aSymbol:

Zugriffsart

aName:

Name des darzustellenden Benutzers

parentModel:

Erzeugendes Model

openOnUser: aUser withParent: parentModel

Erzeugen und Öffnen eines neuen Models mit Besitzer und erzeugendem Model

Ein neues Model wird dem erzeugenden Model erzeugt, der besitzer wird gesetzt. Die Zugriffsart wird auf #readOwner gesetzt und der Besitzer auch als darzustellender Benutzer übernommen. Der Name im Model wird auf den Namen des Benutzers gesetzt. Das Model wird geöffnet und sein Builder zurückgegeben.

aUser:

Besitzer

parentModel:

Erzeugendes Model

Kategorie interface specs

windowSpec

Literatur

- [1] R. Reiger, R. Kiehnscherf, S. Görlitz: *Zur Wissensbasis für die fertigungs- und montagegerechte Konstruktion von Silizium-Mikrostrukturen*; W. John, H. Eggert (Hrsg.), Tagungsband 1. Workshop "Methoden- und Werkzeugentwicklung für den Mikrosystementwurf", Karlsruhe, 15. November 1994, S. 102-108.
- [2] R. Reiger: *Wissensbasiertes System für die fertigungs- und qualitätsgerechte Konstruktion von Mikrosystemen*; W. John, H. Eggert, U. Hamm, K.-D. Müller-Glaser, P. Schwarz (Hrsg.), Tagungsband 4. Workshop "Methoden- und Werkzeugentwicklung für den Mikrosystementwurf", Karlsruhe, 18./19. November 1996, S. 433-469.
- [3] R. Reiger, R. Kiehnscherf, S. Görlitz: *Wissensbasiertes System für die fertigungs-, qualitätsgerechte und montagegerechte Konstruktion von Mikrosystemen*; W. John, H. Eggert, K.-D. Müller-Glaser (Hrsg.), Tagungsband 2. Workshop "Methoden- und Werkzeugentwicklung für den Mikrosystementwurf", Karlsruhe, 14. November 1995, S. 72-79.
- [4] K.-U. Stucky, H. Eggert, A. Schmidt, R. Reiger: *Werkzeugadaption an ein objektorientiertes Datenbank-Management-System am Beispiel "Elektronisches Laborbuch"*; W. John, H. Eggert, U. Hamm, K.-D. Müller-Glaser, P. Schwarz (Hrsg.), Tagungsband 4. Workshop "Methoden- und Werkzeugentwicklung für den Mikrosystementwurf", Karlsruhe, 18./19. November 1996, S. 375-388.
- [5] ObjectStore: *ObjectStore for Smalltalk*; User Guide Release 1.0, Burlington, MA, März 1995.
- [6] S.M. Lang, P.C. Lockemann: *Datenbankeinsatz*; Springer-Verlag, Berlin, Heidelberg, 1995.
- [7] ObjectStore: *UserGuide for C++*; ObjectStore Release 3 for Unix Systems, Object Design, Inc., 25 Burlington Mall Road, Burlington, MA, 1993.
- [8] H.F. Korth, A. Silberschatz: *Database system concepts*; 2. ed., McGraw-Hill, New York, 1991.
- [9] R.F. Resende, D. Agrawal, A. El Abbadi: *Semantic Locking in Object-Oriented Database Systems*; Technical Report - Department of Computer Science, University of California, Santa Barbara, CA, 1994.
- [10] A. Goldberg, D. Robson: *Smalltalk-80: The Language and its Implementation*; Addison-Wesley, Reading, Mass., 1989, S. 249 ff.
- [11] P.C. Lockemann (Hrsg.): *Datenbank-Handbuch*; Springer-Verlag, Berlin, Heidelberg, 1987.