Universität Hamburg

Fachbereich Informatik

STUDIENARBEIT

Verwendung und Erweiterung des JWAM Rahmenwerkes

Vorgelegt von:

Olaf Thiel
14. Semester Informatik
Matrikelnummer: 4341039

Basselweg 105
22527 Hamburg

Telefon: 040 54768703

Michael Skutta
14. Semester Informatik
Matrikelnummer: 4341113

Richard-Linde-Weg 15b
21033 Hamburg

Telefon: 040 7382722

Betreut durch:

Prof. Heinz Züllighoven

Hamburg, den 16.03.1999

Inhaltsverzeichnis

Abbildungen

0 Einleitung

Die nachfolgende Arbeit entstand aus der Notwendigkeit heraus, eine umfangreiche Beispiel-Anwendung für das am Arbeitsbereich Softwaretechnik des Fachbereiches Informatik der Universität Hamburg entwickelte Rahmenwerk „JWAM“ zu bauen. Es existierten zwar bereits einige kleine Beispiel-Anwendungen, diese bezogen sich jedoch immer nur auf einen sehr kleinen Ausschnitt der von diesem Rahmenwerk zur Verfügung gestellten Funktionalität. Unser Ziel war es, eine in ihrer Komplexität überschaubare, aber dennoch hinreichend große Anwendung mit Hilfe des JWAM-Rahmenwerkes zu erstellen.

Das von uns dabei erstellte Pausenplaner-System basiert hauptsächlich auf den Diplomarbeiten von Andreas Hartmann[1] und Tim Krauß[2], die uns diese freundlicherweise zur Verfügung stellten.

Eine der ersten Hürden, auf die wir zu Beginn unserer Arbeit stießen, war das Fehlen jeglicher Art von Materialverwaltung innerhalb der damaligen JWAM-Version. Das Schließen dieser Lücke wurde zu einem zweiten Schwerpunkt unserer Arbeit. Dieser zweite Schwerpunkt der Arbeit wurde unter anderem durch die Teilnahme am Seminar »Corba, WWW, Java«[3] motiviert, in dessen Rahmen ebenfalls eine Materialverwaltung benötigt wurde. Parallel zu der Entwicklung einer umfangreicheren Beispiel-Anwendung für das JWAM-Framework, einem verteilten Pausenplaner-System für Schulen, haben wir daher zusätzlich ein Archiv entwickelt, welches die vom Pausenplaner benutzen Materialien verwalten und persistent halten soll. Das Archiv ist dabei nicht explizit auf den Pausenplaner zugeschnitten, sondern soll auch für ein möglichst breites Spektrum von anderen Anwendungen zur Verfügung stehen. Eine Folge dieser beiden Schwerpunkte unserer Arbeit war eine positive Wechselwirkung zwischen der Entwicklung des Pausenplaners und des Archives, da zum einen die Anforderungen des Pausenplaners in den Entwurf des Archives eingingen und zum anderen der Pausenplaner das entstehende Archiv zur Materialverwaltung nutzen konnte, wobei wiederum die hierbei gemachten Erfah-

1 Andreas Hartmann: „Softwarekonstruktion nach WAM zur Unterstützung von Kooperation an einem Ort am Beispiel eines Pausenplaner-Systems“ Diplomarbeit, Uni Hamburg 1998
2 Tim Krauß: „Softwarekonstruktion nach WAM unter Nutzung des MFC Rahmenwerks“ Diplomarbeit, Uni Hamburg 1998
3 Projektseminar „Corba, WWW, Java: Verteilte Anwendungssysteme am Beispiel von KIS“

rungen erneut in die Entwicklung des Archives eingingen.

Der Aufbau unserer Arbeit gliedert sich zunächst in die Vorstellung der einzelnen Komponenten, die für die Erstellung des Pausenplanersystems benötigt werden, bzw. mit deren Hilfe später die Implementierung erfolgt. Als erste dieser Komponenten wird in Kapitel 1 die fachliche Aufgabe des Pausenplaners erläutert, sowie die zur Erfüllung dieser Aufgaben benötigten Werkzeuge und deren Handhabung beschrieben. Im zweiten Kapitel folgt eine Vorstellung des JWAM-Rahmenwerkes, dessen Benutzung anhand des Pausenplaners gezeigt werden soll. Die Eignung des Pausenplaners als Beispiel-Anwendung wird im darauffolgenden Kapitel diskutiert. Bevor im Kapitel 5 ausführlich auf das Archiv eingegangen wird, erfolgt in Kapitel 4 noch ein kurzer Abriß über kooperative Arbeit, der für das Verständnis der nachfolgenden Kapitel von Vorteil sein kann. Im Kapitel 6 werden schließlich die in den vorangegangenen Kapiteln erläuterten Einzelteile zusammengefügt. Hier gehen wir ausführlich auf die Implementation der Werkzeuge unter Verwendung des JWAM-Frameworks ein und demonstrieren dessen Erweiterung am Beispiel eigener Präsentationsformen. Des Weiteren schildern wir in diesem Kapitel die Konstruktion der Materialien und des zur Materialversorgung der Werkzeuge verwendeten Automatens. Den Abschluß dieses Kapitels bildet ein technisches Szenario, in dem das Zusammenspiel der Werkzeuge mit der Materialversorgung und dem Archiv als Kooperationsmedium veranschaulicht werden soll. Im 7. Kapitel werden wir die im Verlauf dieser Arbeit gewonnenen Erkenntnisse noch einmal zusammenfassen und ein Fazit ziehen.

Wenn wir im Verlauf dieser Arbeit von *dem Anwender*, *dem Leser* oder *dem Benutzer* sprechen, meinen wir damit Rollennamen im Sinne funktioneller Rollen[4], die menschliche Aktivitäten und Aufgaben bezeichnen. Wir verzichten ausschließlich wegen der besseren Lesbarkeit im weiteren auf Schreibweisen wie *EntwicklerInnen* oder *Benutzer und Benutzerinnen*.

4 siehe Christiane Floyd: Einführung indie Softwaretechnik.Scriptum zur gleichnamigen Vorlesung, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softtwaretechnik, Hamburg 1997.

1 Was ist der Pausenplaner

In diesem Kapitel möchten wir die von uns erstellte Beispiel-Anwendung vorstellen. Dazu werden wir im ersten Teil einige Grundbegriffe erklären (Abschnitt 1.1). Danach möchten wir an Hand von zwei Szenarien in den Anwendungsfall einführen um zum Abschluß dieses Kapitels einige Werkzeuge und deren Handhabung mit Hilfe von Visionen vorzustellen.

1.1 Begriffsbildung

Wir werden in den nächsten Kapiteln viel über Begriffe wie Lehrer, Pause, Aufsichtsort usw. reden. Deshalb möchten wir hier die Gelegenheit nutzen die elementarsten Grundbegriffe kurz zu erläutern. Der interessierte Leser findet die vollständigen Materialvisionen in der Diplomarbeit von Andreas Hartmann[5].

- **Zeitraum**

  Mit Zeitraum meinen wir ein Intervall innerhalb einer Woche. Er beginnt und endet an einem Wochentag zu einer bestimmten Uhrzeit. Dabei kann sich der Zeitraum auch über mehrere Tage erstrecken, so das der Wochentag für den Beginn und das Ende nicht identisch sein müssen.

- **Ausschlußzeitraum**

  Ein Ausschlußzeitraum bezeichnet einen Zeitraum in dem ein Lehrer nicht in der Schule anwesend sein kann.

- **Lehrer**

  Für uns ist es nicht so entscheidend, daß ein Lehrer Schüler unterrichtet. Viel mehr steht der Gesichtspunkt der Pausenbeaufsichtigung im Vordergrund dieser Arbeit. Daher ist es wichtig zu wissen, daß sich die Lehrer in einer Schule nicht nur durch ihren Namen unterscheiden, sondern auch in der Art ihrer Stelle und in ihrer zeitlichen Verfügbarkeit. So ist die Anzahl der Pausen, welche ein Lehrer beaufsichtigen muß, abhängig von der Art seiner Stelle, wobei es ganze, halbe und viertel Stellen gibt. Und natürlich ist ein Lehrer nicht den ganzen Tag anwesend. Deshalb muß man für die Pausenplanung wissen, in welchen Zeiträu-

5 Siehe Diplomarbeit von Andreas Hartmann

men der Lehrer nicht in der Schule ist (Ausschlußzeitraum).

- **Lehrkörper**

  Im Lehrkörper sind alle Lehrer der Schule mit ihren Daten zusammengefaßt.

- **Aufsichtsort**

  Ein Aufsichtsort ist ein Platz innerhalb der Schule, an welchem sich die Schüler in der Pause aufhalten und welcher daher beaufsichtigt werden muß. Zu jedem Aufsichtsort wird bestimmt, wie viele Lehrer ihn mindestens beaufsichtigen sollen. Damit diese Zahl nicht unterschritten wird, werden ebenso viele Vertreter eingeteilt.

- **Pause**

  Eine Pause wird sowohl durch ihren Anfangs- und End-Zeitpunkt bestimmt, als auch durch eine Menge von Aufsichtsorten, die in der Pause beaufsichtigt werden müssen.

- **Pausenliste**

  Alle vorhanden Pausen stehen in einer Pausenliste.

- **Pausenplan**

  Der Pausenplan enthält die Einteilung der Lehrer eines Lehrkörpers zu den Pausen einer Pausenliste.

1.2 Szenarien

Nachdem wir im vorigen Abschnitt die grundlegenden Begriffe erläutert haben, möchten wir im folgenden Abschnitt in den eigentlichen Anwendungsfall einführen. Dies soll mittels eines Überblicksszenarios und eines Aufgabenszenarios geschehen. Während erstes den grundlegenden Sachverhalt verdeutlichen soll, möchten wir mit dem zweiten Szenario eine alltägliche Arbeitssituation veranschaulichen.

1.2.1 Überblicksszenario Pausenplanung

Die Planung der Pausen an einer Schule gliedert sich in zwei Teilaufgaben. Dies sind die Lehrkörperpflege und die Belegung der Pausen mit Aufsichtspersonal. Die erste Aufgabe wird von einem oder mehreren Sekretariatsmitarbeitern über-

nommen. Sie beinhaltet die Aktualisierung des Lehrkörpers bei Veränderungen am Personalstamm, sowie Veränderungen bei der zeitlichen Verfügbarkeit der einzelnen Lehrer. Für die zweite Aufgabe ist ein fest bestimmter Sekretariatsmitarbeiter zuständig. Er ist für die konfliktfreie Einteilung des Lehrpersonals verantwortlich. Mit konfliktfrei ist gemeint, daß er beim Einteilen der Lehrer auf deren Verfügbarkeit und ihre Zahl von Aufsichtspflichten achtet. Im Verwaltungsraum der Schule stehen zwei Ordner mit der Aufschrift „Lehrpersonal“ und „Pausenplanung“. Der erste Ordner enthält alle Informationen zum Lehrkörper der Schule. In dem zweiten sind die Einteilungen der Aufsichten enthalten.

1.2.2 Aufgabenszenario Lehrkörperpflege

Der Pausenplaner möchte die Pausenplanung durchführen. Zu diesem Zweck begibt er sich in den Verwaltungsraum der Schule und nimmt die Ordner „Lehrpersonal“ sowie „Pausenplanung“ aus dem Regal. Er begibt sich mit beiden Ordnern in sein Zimmer, um dort die Planung der Pausen durchzuführen.

Zur gleichen Zeit erlangt ein Sekretariatsmitarbeiter Kenntnis von einer Änderung der zeitlichen Verfügbarkeit eines Lehrers. Er begibt sich sofort in den Verwaltungsraum der Schule, um die notwendigen Änderungen im Ordner „Lehrkörper“ vorzunehmen. Im Verwaltungsraum stellt er fest, daß beide Ordner fehlen. Aus diesem Sachverhalt heraus schließt der Sekretariatsmitarbeiter, daß sich die beiden Ordner beim Pausenplaner befinden müssen, denn dieser ist die einzige Person in der Schule, welche beide Ordner verwendet. Da der Sekretariatsmitarbeiter weiß, wo sich das Zimmer des Pausenplaners befindet, begibt er sich dorthin. Nach einer kurzen Begrüßung verständigen sich die beiden über das Anliegen des Sekretariatsmitarbeiters und der Pausenplaner läßt den Sekretariatsmitarbeiter die Änderungen am Lehrkörper durchführen. Der Pausenplaner hat dabei weiterhin Einsicht auf den Ordner „Lehrkörper“und kann dadurch ungehindert mit der weiteren Einteilung der Pausenaufsichten fortfahren. Nachdem der Sekretariatsmitarbeiter die Änderungen durchgeführt hat, bedankt sich der Pausenplaner für die prompte Mitteilung, da er andernfalls größere Teile seiner Pläne hätte neu aufstellen müssen. Der Sekretariatsmitarbeiter geht daraufhin wieder zurück in sein Zimmer.

1.3 Visionen

Zum Abschluß des 1 . Kapitels möchten wir die von uns unter Verwendung des JWAM-Rahmenwerkes gebauten Werkzeuge zur Pausenplanung und deren Handhabung mittels entsprechender Werkzeug- und Handhabungs-Visionen vorstellen.

1.3.1 Werkzeugvision: Lehrkörper-Bearbeiter

Das Lehrkörper-Bearbeiter-Werkzeug (siehe Abbildung 1)dient dazu, einen vorhandenen Lehrkörper zu pflegen. Hier können Lehrer erstellt und in den Lehrkörper aufgenommen oder aus dem Lehrkörper entfernt werden. Des Weiteren können die Daten eines Lehrers geändert werden. Dazu zählen Vor- und Nachname genauso wie die Stelle des Lehrers und seine Ausschlußzeiträume.

1.3.1.1 Materialpräsentation

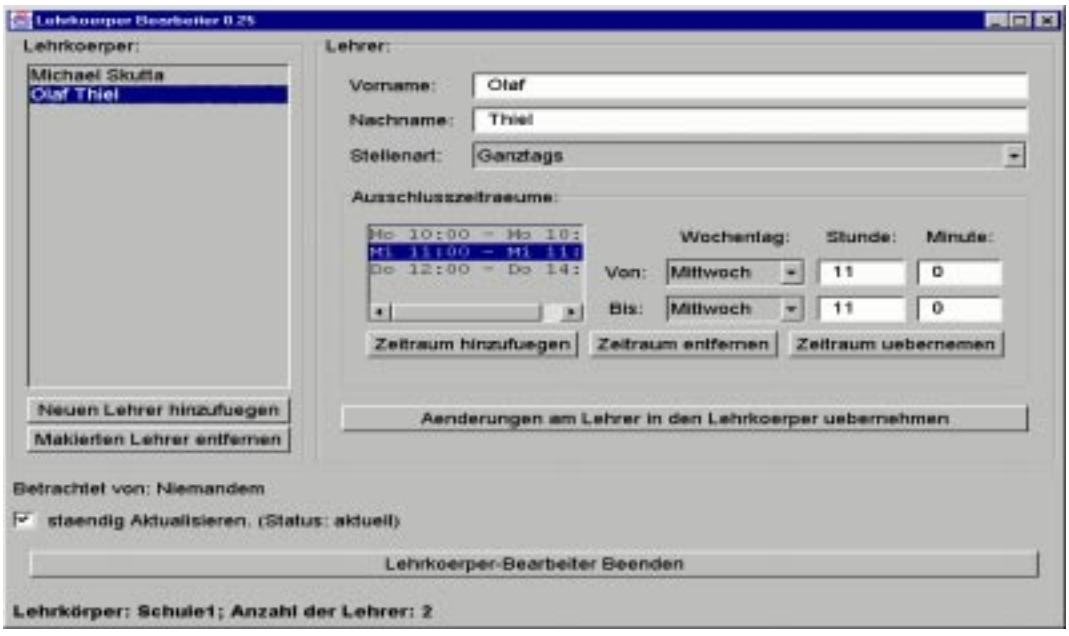


Abbildung 1: Werkzeugvision Lehrkörper-Bearbeiter

Das Werkzeug zeigt auf seiner linken Seite die Liste der im Lehrkörper befindli - chen Lehrer in alphabetischer Reihenfolge. Wird in dieser Liste ein Lehrer mar - kiert, so werden auf der rechten Seite die Daten des Lehrers dargestellt. Oben ste -

hen der Vor- und Nachname sowie die Stelle des Lehrers. Darunter wird eine Liste seiner Ausschlußzeiträume dargestellt. Dort erscheinen die Ausschlußzeit - räume zeitlich geordnet. Wird aus dieser Liste ein Eintrag markiert, so erscheinen die Zeitraum-Daten rechts neben der Liste und können geändert werden.

1.3.1.2 Aktionen des Benutzers

- Neuen Lehrer in den Lehrkörper einfügen
  Der Benutzer klickt auf den Button „Neuen Lehrer hinzufügen“. Daraufhin erscheint in der Lehrer-Liste ein neuer Eintrag mit dem Namen „neuer Lehrer“. Dieser ist sofort selektiert und der Benutzer kann auf der rechten Seite des Werkzeuges den wirklichen Namen und die korrekte Stelle des Lehrers eintra - gen. Nachdem der Benutzer diese Änderungen mit einem Klick auf den But ton „Änderungen am Lehrer in den Lehrkörper übernehmen“ bestätigt hat, werden die Änderungen in den Lehrkörper übernommen und die Anzeige aktualisiert.
- Lehrer aus dem Lehrkörper entfernen
  Der Benutzer wählt den Lehrer, welchen er aus dem Lehrkörper entfernen möchte, in der Liste der Lehrer aus. Daraufhin klickt er auf den Button „Mar - kierten Lehrer entfernen“. Der Lehrer wird jetzt aus dem Lehrkörper entfernt und die Anzeige aktualisiert.
- Ausschlußzeitraum eines Lehrers hinzufügen
  Der Benutzer hat den Lehrer, welchem er den Ausschlußzeitraum hinzufügen möchte, ausgewählt. Er klickt auf den Button „Zeitraum hinzufügen“. Es wird ein neuer Zeitraum in die Liste der Zeiträume aufgenommen, welcher schon selektiert ist. Der Benutzer ändert daraufhin rechts neben der Zeitraumliste die Daten des Zeitraumes nach seinen Wünschen ab und bestätigt diese Änderun - gen mit einem Klick auf den Button „Zeitraum übernehmen“. Hierdurch wer - den die vom Benutzer am Zeitraum vorgenommenen Änderungen in den Leh - rer somit auch in den Lehrkörper übernommen.
- Ausschlußzeitraum eines Lehrers entfernen
  Der Benutzer wählt zunächst den entsprechenden Lehrer aus der Liste aus. Daraufhin werden rechts von der Liste die Daten des Lehrers angezeigt. Aus der

Liste der Ausschlußzeiträume wählt der Benutzer denjenigen aus, welchen er gelöscht haben möchte. Danach klickt er auf den Button „Zeitraum entfernen“. Diese Änderung wird sofort in den Lehrer und in den Lehrkörper übernommen.

- Daten eines Lehrers verändern

  Der Benutzer wählt den zu ändernden Lehrer aus der Lehrer-Liste aus, worauf rechts neben der Liste die Daten des Lehrers erscheinen und zur Änderung frei - gegeben werden. Nun macht der Benutzer seine Änderungen an diesen Daten. Wenn er fertig ist, bestätigt er die gemachten Änderungen mit einem Klick auf den Button „Änderungen am Lehrer in den Lehrkörper übernehmen“.

1.3.2 Werkzeugvision: Pausenlisten-Bearbeiter

Das Pausenlisten-Bearbeiter Werkzeug (siehe Abbildung 2) dient zur Pflege der Pausenlisten. Hier können Pausen in die Liste aufgenommen, aus der Liste ent - fernt sowie die Daten einer Pause verändert werden.

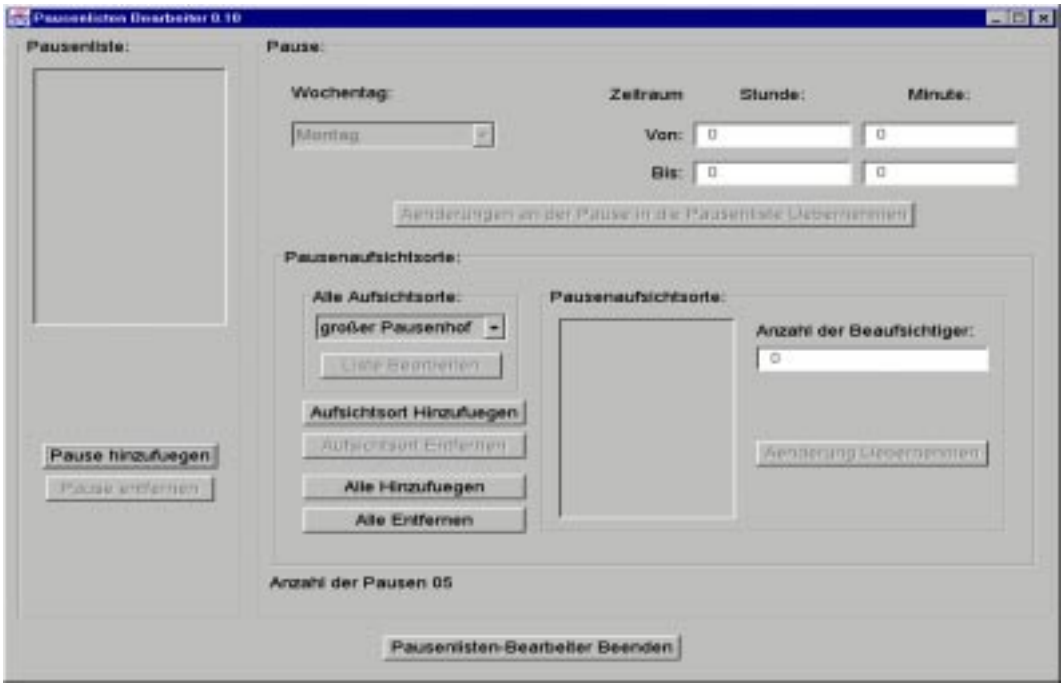


Abbildung 2: Werkzeugvision Pausenlisten-Bearbeiter

1.3.2.1 Materialpräsentation

Das Werkzeug stellt auf seiner linken Seite die Menge der in der Pausenliste enthaltenen Pausen zeitlich geordnet dar. Auf seiner rechten Seite werden, sofern in der Liste eine Pause selektiert wurde, die Daten der Pause angezeigt. In der oberen Hälfte der rechten Seite werden die Eckdaten der Pause dargestellt und darunter eine Liste mit den zur Pause gehörenden Aufsichtsorten.

1.3.2.2 Aktionen des Benutzers

- Eine neue Pause in die Pausenliste aufnehmen

  Der Benutzer klickt auf den Button „Pause hinzufügen“, woraufhin eine neue Pause mit einem Zeitraum von „Montag 00:00 bis Montag 00:00“ erzeugt wird. Diese Pause, welche in der Liste auch gleich selektiert ist, ändert der Benutzer auf der rechten Seite des Werkzeugs nach seinen Wünschen ab. Wenn er alle Änderungen durchgeführt hat, bestätigt er diese durch Klicken des Buttons „Änderungen an der Pause in die Pausenliste übernehmen“.
- Eine Pause aus der Pausenliste entfernen

Will der Benutzer eine bestehende Pause aus der Pausenliste entfernen, so markiert er diese in der Liste der enthaltenen Pausen. Danach klickt er auf den Button „Pause entfernen“. Die betreffende Pause wird somit aus der Pausenliste entfernt und die Anzeige entsprechend aktualisiert.

- Einer Pause einen Aufsichtsort zuweisen
  Bei der Zuweisung von Aufsichtsorten zu Pausen hat der Benutzer grundsätzlich zwei Möglichkeiten. Wenn er alle verfügbaren Aufsichtsorte zuweisen möchte, dann wählt er die Pause in der Liste aller Pausen aus. Daraufhin werden die angezeigten Daten in der rechten Hälfte des Werkzeuges aktualisiert. Der Benutzer klickt jetzt auf den Button „Alle hinzufügen“. Nun werden alle bis jetzt angelegten Aufsichtsorte der Pause zugewiesen und die Liste mit den zugewiesenen Aufsichtsorten aktualisiert. Möchte er der Pause nur einen bestimmten Aufsichtsort zuweisen, so wählt er auch hier wie weiter oben schon beschrieben die betreffende Pause aus. Nun wählt er den gewünschten Aufsichtsort in dem Auswahlfeld „Alle Aufsichtsorte“ aus und klickt auf den Button „Aufsichtsort hinzufügen“.
- Einen Aufsichtsort von der Pause entfernen
  Um einen Aufsichtsort von einer Pause zu entfernen, wählt der Benutzer diesen in der Liste der zugewiesenen Aufsichtsorte aus und klickt auf den Button „Aufsichtsort entfernen“. Diese Änderung bestätigt er sodann mit einem Klick auf den Button „Änderungen an der Pause in die Pausenliste übernehmen“. Alternativ hat der Benutzer die Möglichkeit, alle bisher der Pause zugewiesenen Aufsichtsorte auf einmal zu entfernen. Dazu klickt er nur auf den Button „Alle entfernen“ und bestätigt dies wie weiter oben beschrieben.
- Die Liste der Aufsichtsorte bearbeiten
  Will der Benutzer die Liste aller verfügbaren Aufsichtsorte bearbeiten, so kann er durch Klick auf den Button „Liste bearbeiten“ das Aufsichtsorte-Bearbeiter Werkzeug (Abschnitt 1.3.3) starten.
- Die Daten der Pause verändern
  Der Benutzer möchte die Daten einer Pause verändern. Hierzu wählt er die gewünschte Pause aus der Liste aus. Rechts neben der Liste erscheinen die

Daten und werden zur Bearbeitung freigegeben. Er ändert nun die Daten der Pause wie gewünscht ab. Zum Schluß bestätigt er die gemachten Änderungen indem er auf den Button „Änderungen an der Pause in die Pausenliste übernehmen“ klickt.

1.3.3 Werkzeugvision: Aufsichtsorte-Bearbeiter

Das Aufsichtsorte-Bearbeiter Werkzeug (Abbildung 3)dient zur Pflege der Liste aller Aufsichtsorte einer Schule. Es wird im Pausenplaner-System aus dem Pau - senlisten-Bearbeiter Werkzeug (Abschnitt 1.3.2) heraus gestartet. Es können neue Aufsichtsorte erstellt, alte Aufsichtsorte entfernt und bestehende Aufsichtsorte geändert werden.

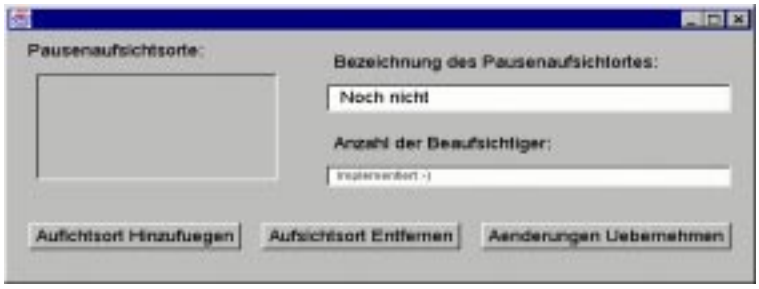


Abbildung 3: Werkzeugvision Aufsichtsorte-Bearbeiter

1.3.3.1 Materialpräsentation

Auf der linken Seite des Werkzeuges wird die Liste aller bisher angelegten Auf - sichtsorte alphabetisch sortiert angezeigt. Wird ein Aufsichtsort markiert, so wird sein Name rechts neben der Liste angezeigt.

1.3.3.2 Aktionen des Benutzers

- einen neuen Aufsichtsort aufnehmen

  Will der Benutzer einen neuen Aufsichtsort in die Liste aller Aufsichtsorte einfügen, so klickt er auf den Button „Aufsichtsort hinzufügen“. Es wird ein Aufsichtsort mit der Bezeichnung „neuer Aufsichtsort“ und einer Anzahl von 0 Beaufsichtigern in die Liste eingefügt. Dieser ist sofort selektiert, so daß der Benutzer die Daten des Aufsichtsortes rechts von der Liste abändern kann. Zum

Abschluß bestätigt er diese Änderungen mit Klick auf den Button „Änderungen übernehmen“.

- einen Aufsichtsort entfernen
  Möchte der Benutzer einen existierenden Aufsichtsort aus der Liste aller Auf - sichtsorte entfernen, so wählt er diesen aus der Liste aus und klickt auf den Button „Aufsichtsort entfernen“.
- den Aufsichtsort verändern
  Zum Verändern der Bezeichnung oder der Anzahl der Beaufsichtigter eines Aufsichtsortes wählt der Benutzer zunächst den betreffenden Aufsichtsort aus der Liste aller Aufsichtsorte aus. Die zum gewählten Aufsichtsort gehörenden Daten werden nun rechts neben der Liste zur Veränderung freigegeben. Hat der Benutzer seine Änderungen durchgeführt, so bestätigt er dies mit Klick auf den Button „Änderungen übernehmen“.

1.3.4 Werkzeugvision: Pausenplan-Bearbeiter

Das Pausenplan-Bearbeiter Werkzeug (siehe Abbildung 4) dient der Belegung von Pausen mit Lehrern. Es kann eine Pause mit einem Lehrer belegt werden und es können existierende Belegungen entfernt werden. Zusätzlich werden Bele - gungskonflikte angezeigt.

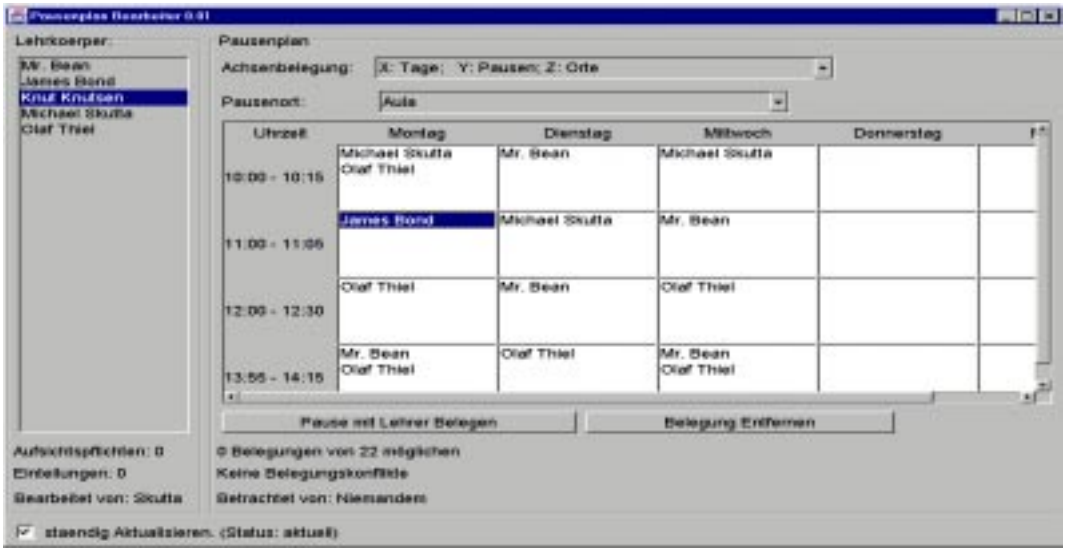


Abbildung 4: Werkzeugvision Pausenplan-Bearbeiter

1.3.4.1 Materialpräsentation

Im Werkzeug werden auf der linken Seite alle Lehrer des Lehrkörpers alphabetisch sortiert in einer Liste angezeigt. Rechts neben dieser Liste wird der Pausenplan angezeigt. Da der Pausenplan drei Dimensionen hat (Uhrzeit, Tag, Aufsichtsort), aber immer nur zwei Dimensionen angezeigt werden können, kann der Benutzer zwischen unterschiedlichen Darstellungen des Pausenplanes wählen (siehe hierzu auch den betreffenden Abschnitt in 1.3.4.2). Grundsätzlich werden immer zwei der drei Dimensionen in einer Tabelle angezeigt, und die dritte Dimension kann durch ein Auswahlfeld gewählt werden.

1.3.4.2 Aktionen des Benutzers

- Darstellung des Pausenplanes ändern

  Wie schon im vorangegangenen Abschnitt dargelegt wurde, ist es nicht möglich, alle Informationen des Pausenplanes gleichzeitig zur Anzeige zu bringen. Direkt nach dem Starten des Pausenplan-Bearbeiter Werkzeuges wird der Pausenplan als Tabelle dargestellt, wobei in Y-Richtung die Uhrzeiten und in X-Richtung die Wochentage abgebildet sind. Die dritte Dimension, in diesem Falle die unterschiedlichen Aufsichtsorte, können über das Auswahlfeld „Pausenort“ gewählt werden. Eine andere Aufteilung der Dimensionen erreicht der Benutzer, indem

er die von ihm gewünschte Aufteilung in dem Auswahlfeld „Achsenbelegung“ selektiert. Danach wird die Anzeige der drei Dimensionen nach dem gewählten Schema abgeändert.

- Eine Pause mit einem Lehrer belegen

  Der Benutzer kann eine Pause mit einem Lehrer aus dem Lehrkörper belegen, indem er zuerst den gewünschten Lehrer in der Lehrer-Liste markiert. Danach markiert er die entsprechende Pause in der Tabelle und klickt auf den Button „Pause mit Lehrer belegen“. Danach wird die gewählte Pause mit dem markierten Lehrer belegt und die Anzeige aktualisiert.

- Eine Belegung entfernen

  Um eine Belegung zu entfernen, markiert der Benutzer die gewünschte Belegung in der Tabelle und klickt auf den Button „Belegung entfernen“. Danach wird die Belegung entfernt und die Anzeige aktualisiert.

1.3.5 Werkzeugvision: Pausenplaner

Aus dem Pausenplaner-Werkzeug (Abbildung 5) heraus können alle Aktionen begonnen werden, welche nötig sind, um die Aufgabe der Pausenplanung durchführen zu können. Von hier aus werden neue Lehrkörper, Pausenlisten und Pausenpläne generiert. Des Weiteren können von hier alle anderen Werkzeuge gestartet werden, um die Materialien zu bearbeiten.

Abbildung 5: Werkzeugvision Pausenplaner

1.3.5.1 Materialpräsentation

Das Werkzeug gliedert sich in drei Bereiche. Links oben liegt der Lehrkörper-Bereich. Dort werden alle angelegten Lehrkörper in einer Liste alphabetisch geordnet angezeigt. Rechts daneben, im Pausenlisten-Bereich, werden alle angelegten Pausenlisten in einer ebenfalls alphabetisch sortierten Liste angezeigt. Unter diesen beiden Listen liegt der Pausenplan-Bereich, in welchem eine weitere Liste mit allen angelegten Pausenplänen angezeigt wird. Auch diese ist ebenfalls alphabetisch sortiert.

1.3.5.2 Aktionen des Benutzers

- Einen neuen Lehrkörper anlegen

  Der Benutzer legt einen neuen Lehrkörper an, indem er den Namen des neuen Lehrkörpers im Textfeld eingibt und auf den Button „Neuen Lehrkörper erstellen“ klickt.

- Einen Lehrkörper entfernen

  Zum Entfernen eines Lehrkörpers markiert der Benutzer den betreffenden Lehrkörper in der Liste und klickt auf den Button „Lehrkörper entfernen“.

- Einen Lehrkörper bearbeiten

  Um einen Lehrkörper zu bearbeiten, markiert der Benutzer den gewünschten

Lehrkörper in der Liste und klickt auf den Button „Lehrkörper bearbeiten“. Nun wird das Lehrkörper-Bearbeiter Werkzeug (1.3.1) mit dem gewählten Lehrkörper gestartet.

- Eine neue Pausenliste anlegen

  Um eine neue Pausenliste anzulegen, gibt der Benutzer den gewünschten Namen der Pausenliste im Textfeld ein und klickt auf den Button „Neue Pausenliste erstellen“.

- Eine Pausenliste entfernen

  Um eine existierende Pausenliste wieder zu entfernen, markiert der Benutzer diese in der Liste und klickt anschließend auf den Button „Pausenliste entfernen“.

- Eine Pausenliste bearbeiten

  Der Benutzer markiert die zu bearbeitende Pausenliste in der Liste aller Pausenlisten und klickt auf den Button „Pausenliste bearbeiten“. Jetzt wird das Pausenlisten-Bearbeiter Werkzeug (1.3.2)mit der gewünschten Pausenliste gestartet.

- Einen neuen Pausenplan anlegen

  Um einen neuen Pausenplan anzulegen, geht der Benutzer analog zu den vorangegangenen Beschreibungen für den Lehrkörper und die Pausenliste vor. Nur klickt er hier auf den Button „Pausenplan neu erstellen“.

- Einen Pausenplan entfernen

  Der Benutzer markiert den gewünschten Pausenplan in der Liste und klickt auf den Button „Pausenplan entfernen“.

- Einen Pausenplan bearbeiten

  Hat der Benutzer einen Pausenplan in der Liste markiert, klickt er auf den Button „Pausenplan bearbeiten“. Danach wird das Pausenplan-Bearbeiter Werkzeug (1.3.4) mit dem markierten Pausenplan gestartet.

1.3.6 Handhabungsvision: „Erstellen eines neuen Pausenplanes“

Der für die Pausenplanung zuständige Sekretariatsmitarbeiter möchte einen neuen

Pausenplan anlegen. Dazu startet er das „Pausenplaner“ Werkzeug (vergl. 1.3.5). Hier wählt er in der Liste mit den verfügbaren Lehrkörpern einen aus. Als nächstes wählt er aus der Liste mit den verfügbaren Pausenlisten eine aus und gibt den Namen des neuen Pausenplanes in das dafür vorgesehene Feld ein. Danach erstellt er den neuen Pausenplan mit einem Klick auf den Button „Pausenplan neu erstellen“.

1.3.7 Handhabungsvision: „Erstellen eines neuen Lehrkörpers“

Ein Sekretariatsmitarbeiter möchte einen neuen Lehrkörper erstellen. Er startet dazu das „Pausenplaner“ Werkzeug. Hier trägt er in dem dafür vorgesehenen Feld den Namen des neuen Lehrkörpers ein und klickt auf den Button „Neuen Lehrkörper erstellen“. Der neue Lehrkörper wird erstellt und in der Liste mit den verfügbaren Lehrkörpern angezeigt.

1.3.8 Handhabungsvision: „Neuen Pausenplan bearbeiten“

Der für die Pausenplanung verantwortliche Sekretariatsmitarbeiter hat wie in Abschnitt 1.3.6 beschrieben einen neuen Pausenplan erstellt. Um diesen gleich zu bearbeiten selektiert er den Namen des Pausenplanes in der Liste aller verfügbaren Pausenpläne und klickt den Button „Pausenplan Bearbeiten“. Hierauf wird das „Pausenplan-Bearbeiter“ Werkzeug (vergl. 1.3.4) gestartet und der Sekretariatsmitarbeiter führt seine Änderungen durch.

2 Das JWAM-Rahmenwerk

In diesem Kapitel möchten wir das JWAM-Rahmenwerk vorstellen. Da der Umfang einer Studienarbeit begrenzt ist, können wir an dieser Stelle keine vollständige Dokumentation liefern. Leider bleibt uns in diesem Zusammenhang nur auf die JWAM-Homepage[6] des Fachbereiches SWT[7] der Uni zu verweisen, da sich die Dokumentation noch recht spartanisch ausnimmt. Zur Zeit existiert nur die vom JDK[8] aus dem Source-Code generierte Dokumentation mit einigen Kurzanleitungen für spezielle Aufgaben wie Werkzeugkonstruktion oder die Erzeugung von grafischen Oberflächen.

Zuerst wollen wir klären, wofür JWAM überhaupt steht und wie das Rahmenwerk aufgebaut ist. Danach gehen wir auf die grundlegenden Komponenten des Rahmenwerks ein. Zum Abschluß dieses Kapitels werden wir diskutieren, was dem Rahmenwerk zur Lösung unserer Aufgabe noch fehlt.

2.1 Struktur des JWAM-Rahmenwerks

JWAM ist ein Rahmenwerk, welches am Arbeitsbereich SWT der Uni Hamburg erstellt wurde, um den Entwickler bei der Konstruktion objektorientierter Software in Java (J) nach der **W**erkzeug- **A**utomaten- und **M**aterial-Metapher (WAM[9]) zu unterstützen. Wir werden hier nicht näher auf das WAM-Konzept eingehen, da dies ebenfalls den Umfang einer Studienarbeit sprengen würde. Zu diesem Thema existiert aber eine große Anzahl an Literatur; einen Teil enthält unsere Literaturliste am Ende dieser Arbeit.

Da das Rahmenwerk selbst in Java geschrieben ist, liegen die einzelnen Komponenten von vornherein in einer hierarchischen Struktur vor (den sog. Packages). Die Packages sind dabei so gewählt, das sich die in Abbildung 6 gezeigten Schichten ergeben, welche der WAM-Layer-Architektur entsprechen. Wir wollen nun jede Schicht kurz vorstellen und die darin enthaltenen Mechanismen beschreiben.

6 http://swt-www.informatik.uni-hamburg.de/Software/JWAM
7 Abk.: Softwaretechnik
8 Java Devellopment Kit
9 Heinz Züllighoven: Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz, Heidelberg 1998.

Abbildung 6: Layer des JWAM-Rahmenwerks

2.1.1 Handhabungs- und Präsentationsschicht

In dieser Schicht befinden sich alle Klassen die der Anwendungsentwickler zur Oberflächengestaltung und zur Werkzeugkonstruktion benötigt, sowie die Klasse ***Thing***, welche die Oberklasse von allen Materialien und Werkzeugen ist.

Letztlich enthält die Schicht auch die sog. WAM-Arbeitsumgebung, durch die das fertige System nach außen hin abgeschlossen wird und in welcher der elektronische Schreibtisch enthalten ist, mit dem die konstruierten Werkzeuge gestartet werden.

2.1.2 Systembasisschicht

Von dieser Schicht werden die systemspezifischen Leistungen gekapselt und dem Anwendungsentwickler durch die Verwendung einer einheitlichen Schnittstelle zur Verfügung gestellt. Unter anderem enthält sie die Klassen der jConlib[10], des Vertragsmodells, der Reaktionsmechanismen und für das Exceptionhandling.

10 von Holger Bohlmann entwickelte Container-Bibliothek

2.1.3 Technologieschicht

Diese Schicht beinhaltet alle Technologien, welche bei der Anwendungsentwicklung zum Einsatz kommen. Zur Zeit befinden sich hier hauptsächlich Klassen für die Realisierung von verteilten Systemen.

2.2 Fehlenden Komponenten des JWAM-Rahmenwerks zur Implementation des Pausenplaner-Systems

Obwohl das JWAM-Rahmenwerk in der uns vorliegenden Version[11] schon über eine gute Grundfunktionalität verfügt, gibt es noch mehrere Bereiche, die kaum oder noch gar nicht abgedeckt werden. Zu allererst sei hier das Fehlen eines Persistenz-Mechanismusses erwähnt. Zur Zeit besteht die einzige Möglichkeit seine Daten zu speichern darin, sie direkt aus Java mittels des sog. Serialisierens[12] auf einen Datenträger zu schreiben. Vor diesem Hintergrund haben wir uns dazu entschlossen unser Pausenplaner-System mit einem kleinen Archiv auszustatten. Des Weiteren sind zur Zeit nur die elementarsten Präsentationsformen integriert. So waren weder ein dynamisch beschriftbares Textlabel noch eine Tabelle vorhanden. Beides haben wir implementiert und gehen später näher darauf ein. Das von Hartmann und Krauß in ihren Arbeiten beschriebene System sah darüber hinaus eine Art „Schwarzes-Brett“[13] und eine „Drag & Drop“-Funktionalität vor. Beides haben wir in unserer Arbeit nicht realisiert, da die hierzu benötigten Funktionen ebenfalls noch nicht vorhanden waren. „Drag & Drop“ ist in der Version die uns vorlag noch nicht ausgereift und eine Art Desktop, der zur Realisierung eines „Schwarzen Brettes“ hilfreich gewesen wäre, fehlt noch völlig.[14]

Alle hier beschriebenen „Mängel“ sollen allerdings in einer der nächsten Versionen behoben werden.

11 JWAM V1.2
12 Siehe SUN JAVA Dokumentation
13 Dipl. Arbeiten von A. Hartmann und T. Krauß
14 Es sei erwähnt, das Martin Lippert zur Zeit an einem Desktop und „Drag & Drop“ arbeitet.

3 Eignung des Pausenplaners als Beispiel-Anwendung

Im Zuge der Entwicklung des JWAM-Rahmenwerkes und seiner Verwendung entstanden ein paar sehr knappe Beispiel-Anwendungen. Diese beleuchten aber leider nur einen jeweils sehr kleinen Teil der Funktionalität, die in den meisten Fällen nur von einer oder wenigen Komponenten erbracht wird. Man wünschte sich eine überschaubare aber doch komplette Beispiel-Anwendung, die zum einen die Zusammenarbeit dieser bisher nur isoliert betrachteten Komponenten demonstrieren und dabei auch den Verteilungsaspekt berücksichtigen sollte. In diesem Kapitel möchten wir aufzeigen, warum sich daher gerade das Pausenplaner-System für diese Aufgabe eignet.

3.1 Vorteile

In diesem Abschnitt möchten wir in knappen Sätzen aufzeigen, wo die besonderen Vorteile des Pausenplaner-Problems sind, die uns bewogen haben dieses für eine Beispiel-Anwendung zu benutzen.

- **Verteilte Anwendung**
  Die Pausenplaner-Problematik ist von Haus aus eine verteilte Anwendung, da hier verschieden Personen an unterschiedlichen Orten zugleich auf das selbe Material zugreifen können.
- **kleiner Anwenderkreis**
  Die Anzahl der an dem Prozeß der Pausenplanung beteiligten Personen ist nicht sehr groß.
- **überschaubare Werkzeuge**
  Die Werkzeuge sind sowohl bei der Materialpräsentation wie auch in der Handhabung recht unproblematisch.
- **komplexe Materialien**
  Die Materialien sind relativ komplex. So enthält beispielsweise der Lehrkörper Lehrer und ein Pausenplan enthält einen Lehrkörper sowie eine Pausenliste, welche wiederum mehrere Pausen beinhaltet.

- **Szenarien und Visionen schon vorhanden**

  Ein nicht zu unterschätzender Vorteil war, daß wie schon öfter erwähnt, Andreas Hartmann und Tim Krauß für ihre Diplomarbeiten auch ein Pausenplaner-System entwickelten und uns ihre Arbeit zugänglich machten. So konnten wir uns relativ schnell an erste Implementierungen wagen.

- **einigermaßen bekanntes Beispiel**

  Der Pausenplaner wird mit dieser Arbeit nicht zum ersten Mal implementiert. Es existiert sogar eine nicht unerheblich Anzahl an Implementierungen und Konzepten, so daß der eine oder andere JWAM-Neuling sich eventuell gut mit dem Pausenplaner auskennt, was ihm beim Studium der Beispiel-Anwendung zu Gute kommt. Zusätzlich bieten die mit anderen Rahmenwerken und Programmiersprachen erfolgten anderen Implementationen einer identischen Anwendung die Möglichkeit, die Stärken und Schwächen der Kombination von Java und JWAM-Rahmenwerk mit den vorherigen Lösungen zu vergleichen.

3.2 Nachteile

Den einzigen Nachteil sehen wir lediglich darin, das wir für die Implementierung des Systems mehr Funktionalität benötigten, als im JWAM-Rahmenwerk zur Verfügung stand. So kann sich der Anwender nicht auf das Erlernen der JWAM-Funktionalität beschränken, sondern muß sich auch mit der von uns zusätzlich implementierten Funktionalität auseinandersetzen, was ihn am Erlernen des JWAM-Rahmenwerkes eher hindert. Deshalb haben wir nur die nach unserer Auffassung absolut unverzichtbaren Funktionen implementiert und keine höheren Konzepte wie ein „Schwarzes Brett“ oder „Drag & Drop“.

4 Kooperation

Da in den nachfolgenden Kapiteln mehrfach von „Kooperation“, „Kooperationsformen“ oder „kooperativer Arbeit“ die Rede sein wird, wollen wir in diesem Abschnitt diese Begriffe kurz erläutern.

4.1 Kooperative Arbeit

Unter kooperativer Arbeit[15] verstehen wir Arbeit, »... bei der verschiedene Personen geplant und koordiniert zusammen[arbeiten], um ein gemeinsames Ergebnis zu erreichen.«[16] Innerhalb der Kooperation kann man wiederum zwei Arten unterscheiden:

- explizite Kooperation

  Als explizite Kooperation bezeichnet man eine Kooperationssituation, in der dem Benutzer einer Arbeitsumgebung deutlich wird, daß andere Benutzer mit ihm in einer Umgebung arbeiten und er geeignete Mittel für die Weitergabe von Arbeitsgegenständen und die Koordination im Anwendungssystem findet.

- implizite Kooperation

  Bei der impliziten Kooperation teilt ein einzelner Benutzer seine Arbeitsumgebung mit anderen Benutzern im Rahmen einer kooperativen Arbeitssituation, ohne daß die Kooperation zunächst deutlich wird.

Der grundsätzliche Unterschied zwischen expliziter und impliziter Kooperation besteht demnach darin, daß bei der expliziten Kooperation gegenständliche Kooperationsmittel bereitgestellt werden, die in den Arbeitsplatz integriert sind. Als Kooperationsmittel können beispielsweise Archive, Postkörbe oder »Schwarze Bretter« dienen. Diese Kooperationsmittel bieten den am Arbeitsprozeß beteiligten Benutzern zum einen die Möglichkeit, mit ihrer Hilfe die zur Koordination der Arbeit benötigten Informationen auszutauschen, und zum anderen dienen sie dem direkten Austausch der bearbeiteten Materialien. Hierbei gibt es drei unterschiedliche Arten des Zugriffes auf diese von mehreren Personen benötigten Materialien:

15 die folgenden Ausführungen basieren im wesentlichen auf Gryczan, Guido in: Züllighoven, Konstruktionshandbuch, S. 427-437
16 ebenda, S. 428

1. Exklusiver Zugriff ohne Kopien

   Es gibt nur ein Original, das jeweils nur von einem Benutzer bearbeitet werden kann und damit solange für die anderen Benutzer gesperrt ist.

2. Exklusiver Zugriff und Kopien

   Es gibt auch hier nur ein Original, an dem nur der jeweilige Benutzer Veränderungen vornehmen kann. Die anderen Benutzer können jedoch zur Ansicht Kopien des Materials anfordern. Es ist ebenfalls möglich, an diesen Kopien Veränderungen vorzunehmen. Inwieweit die Änderungen an einer Kopie das Original beeinflussen können oder welche Auswirkungen Veränderungen des Originals auf die Kopien haben, muß von den Benutzern geregelt werden.

3. Zugriff nur auf Kopien

   Jeder Benutzer erhält nur eine Arbeitskopie des Originals an dem er Veränderungen vornehmen möchte. Der Benutzer kann danach das Original auf Basis seiner Kopie aktualisieren, bzw. überschreibt die alte Version mit seiner neuen Version. Bei dieser Zugriffsart werden alle Materialien mit einem Zeitstempel versehen. Versucht ein Benutzer, ein Original zu ersetzen, das einen jüngeren Zeitstempel aufweist als seine Arbeitskopie, wird ein Zugriffskonflikt angezeigt, da das Original nach der Herausgabe dieser Arbeitskopie schon von einem anderen Benutzer durch dessen Arbeitskopie aktualisiert worden ist.

5 Das Archiv

In den Versionen 1.0 bis 1.2 des JWAM-Frameworks gab es keine Unterstützung für die Konstruktion, Verwaltung oder Speicherung von Materialien. Da dieser Bereich jedoch von großer Bedeutung für ein umfangreicheres Anwendungsbeispiel wie den Pausenplaner ist, haben wir im Rahmen unserer Studienarbeit versucht, eine eigene Lösung zu entwickeln, die zu einem späteren Zeitpunkt auch in das JWAM-Framework integriert werden könnte. Die Entscheidung, anstatt einer speziellen, nur auf die Bedürfnisse des Pausenplanersystems ausgerichteten Lösung einen allgemein verwendbaren Entwurf auszuarbeiten, wurde durch unsere Teilnahme an dem »Java/CORBA« - Projekt aus dem Wintersemester 97/98 maßgeb-

lich beeinflußt, da die im Rahmen dieses Projektes zu entwickelnden verteilten Anwendungen aus dem Banken- und Krankenhausbereich ebenfalls irgendeine Art von Materialverwaltung benötigten.

Den Ausgangspunkt unseres Vorgehens bildete dabei das Konzept des »Archives«, das die von den Werkzeugen bearbeiteten Materialien verwaltet und persistent hält. Wir haben jedoch nicht zuletzt aus zeitlichen Gründen auf den Entwurf eines in das Framework integrierbaren Konzeptes für die Konstruktion dieser Materialien verzichtet. Andererseits werden wir in diesem Kapitel noch näher auf die fachlichen und technischen Gründe eingehen, die für eine von der Art der verwalteten Materialien unabhängige Konstruktion des Archives sprechen.

5.1 Die Anforderungen an das Archiv

Bei der Ermittlung der Anforderungen an das Archiv haben wir uns nicht auf die Bedürfnisse des Pausenplaner-Systems beschränkt. Das vorrangige Ziel bei der Konstruktion des Archives war es, dieses später für ein möglichst breites Spektrum von Anwendungen einsetzen zu können. Anhand des Pausenplaners konnten wir die Benutzbarkeit des Archives in **einem** der vielen möglichen Anwendungsfälle testen. Die in den folgenden Kapiteln aufgeführten Anforderungen sollen den äußeren Rahmen abstecken, der unserer Ansicht nach zur Erfüllung unseres Zieles notwendig ist. Der größte Teil dieser Anforderungen entstammt dabei dem Projekt aus dem Wintersemester 97/98, in dessen Verlauf unterschiedliche Anwendungen aus dem Bankenbereich, der Krankenhausverwaltung oder der Dokumentverwaltung unter Benutzung des JWAM-Frameworks gebaut werden sollten.

5.1.1 Die fachliche Anforderungen

Bevor es um die „fortgeschrittenen“ fachlichen Anforderungen an das Archiv geht, sollten zuerst noch einmal die grundlegenden Funktionen des Archives benannt werden. Sinn und Zweck des Archives ist die Verwaltung von Materialien. Der Benutzer kann Materialien aus dem Archiv anfordern und ebenso Materialien in das Archiv zurückstellen. Um dem Benutzer darüber Aufschluß zu geben, welche

Materialien von dem Archiv verwaltet werden, kann er eine Bestandsliste anfordern, in der die dem Archiv bekannten Materialien aufgeführt sind. Des Weiteren muß es dem Benutzer auch möglich sein, neue Materialien dem Archiv hinzuzufügen.
Da wir das Archiv als Kooperationsmittel der verteilten Arbeit ansehen, ist zudem der Zugriff auf das Archiv durch mehrere Benutzer in einem verteilten System eine weitere Grundvorraussetzung. Die in einem Mehrbenutzersystem anfallenden Informationen über den Ort der entliehenen Materialien sollen jedoch nicht vom Archiv gesammelt oder verwaltet werden. Im Rahmen des »Java/Corba«-Projektes sollte dazu ein eigener Automat, der sogenannte »Vorgangsmonitor«, implementiert werden. Dieser Vorgangsmonitor sollte im Zusammenspiel mit dem Archiv die Anfragen nach dem aktuellen Aufenthaltsort und dem aktuellen Besitzer eines Materials beantworten. Zusätzlich sollte der Vorgangsmonitor eine Art Logbuch über sämtliche Materialbewegungen führen. Inzwischen scheint jedoch das Konzept der Trennung zwischen Vorgangsmonitor und Archiv nicht mehr ganz aktuell zu sein, wodurch der Aufgabenbereich des Vorgangsmonitors derzeit weder von seiner nur in einer einfachen Basisversion verfügbaren Implementierung noch von unserem Archiv hinreichend abgedeckt wird.

5.1.1.1 Unterstützung der verschiedenen Kooperationsformen

Eine der wichtigsten Anforderungen, die an das Archiv gestellt werden, ist die Unterstützung unterschiedlicher Arten des konkurrierenden Zugriffes. Es sollen sowohl

- der exklusive Zugriff auf das Original ohne die Existenz von Kopien (“Nur Original“)
- der exklusive Schreibzugriff auf das Original bei lesendem Zugriff auf Kopien (“Original und Kopie“)
- die völlige Beschränkung des Zugriffes auf Kopien, die das Original überschreiben können (“Kopie-Kopie“)

durch das Archiv ermöglicht werden. Sollte das Archiv diese drei, sich zum Teil

gegenseitig ausschließenden Arten des konkurrierenden Zugriffes anbieten, stellt sich die Frage nach der Trennung der Zugriffsarten, bzw. inwieweit die Nutzung des Archives nach der einen Zugriffsmethode die jeweils anderen Zugriffsmethoden unterbinden darf. Eine Beschränkung auf nur eine Art des Zugriffes kann sich dabei zum einen an den Werkzeugen orientieren, beispielsweise könnte ein Editor nach der „Original und Kopie“- Logik auf die Materialien zugreifen. Andererseits ist es auch denkbar, daß sich in bestimmten Fällen die Zugriffsart nach dem Typ des Materials richten sollte.

5.1.1.2 Transaktionen / atomare Handlungen

In einem verteilten System, in dem mehrere Benutzer voneinander unabhängig mit den gleichen Materialien arbeiten, ist es oft notwendig, mehrere Bearbeitungsschritte in einer bestimmten Reihenfolge durchzuführen. Diese Folge von Bearbeitungsschritten könnte allerdings zu einem inkonsistenten Ergebnis führen, falls zwischenzeitlich abhängige Materialien von einem anderen Benutzer verändert wurden. Daher ist das Konzept der „atomaren Handlung“[17] auch für die Benutzung des Archives von großer Bedeutung. Das Archiv sollte also einen Mechanismus zur Verfügung stellen, der es dem Benutzer erlaubt, einen oder mehrere Zugriffe auf Materialien vorzunehmen, ohne dabei befürchten zu müssen, durch gleichzeitig stattfindende Aktivitäten anderer Benutzer in seiner Arbeit behindert zu werden.

5.1.1.3 Logische Gliederung des Archives

Aus naheliegenden Gründen ist die Möglichkeit einer Gliederung des Archives in unterschiedliche Abteilungen in fast allen Anwendungsfällen wünschenswert. Um einen weitestgehend an den fachlichen Anforderungen des Anwendungsfalles orientierten Aufbau der Gliederung zu ermöglichen, sollte das Archiv wenn möglich keine Beschränkungen hinsichtlich der Anzahl und Komplexität der Strukturen dieser Unterabteilungen aufweisen.

17 Eike Jessen, Rüdiger Valk: Rechensysteme. Grundlagen der Modellbildung. Berlin, Heidelberg, New York, Paris, Tokyo (1987), S. 137 - 140

5.1.1.4 Keine Einschränkungen bezüglich der Art der Materialien und Verwaltung komplexer Materialien

Das Archiv muß in der Lage sein, jede beliebige Art von Material zu verwalten. Dies können sowohl die „üblichen“ Materialien wie Datensätze oder Dokumente sein, aber auch Bilder, Klänge oder Prozeßmuster. Der Benutzer des Archives sollte damit in der Lage sein, sich bei der Gestaltung seiner Materialien soweit wie möglich an deren fachlichen Anforderungen zu orientieren, ohne dabei auf vom Archiv erzwungene Einschränkungen Rücksicht nehmen zu müssen.

Eine weitere Anforderung an das Archiv besteht in der Verwaltung komplexer Materialien. Das Archiv soll es ermöglichen, auf einzelne Komponenten eines zusammengesetzten Materials zuzugreifen, ohne dabei dieses Material komplett aus dem Archiv herauszunehmen. Zusätzlich sollten die Benutzer des Archives von den komplexen Materialien Inhaltsverzeichnisse anfordern können.

5.1.2 Die technischen Anforderungen

5.1.2.1 Das Archiv sollte auch in Einzelplatzsystemen einsetzbar sein

Die Benutzung des Archives in Einzelplatzsystemen darf dabei nicht durch die nur in Mehrbenutzersystemen erforderlichen Bereiche der Schnittstellen umständlicher als nötig gemacht werden. Eine auf Einzelplatzsysteme zugeschnittene Version des Archives sollte jedoch nicht nur den Benutzer, sondern auch die Ressourcen des Rechners von unnötigem Ballast entlasten.

5.1.2.2 Austauschbarkeit der Netzwerkkomponente

Aufgrund der zu diesem Zeitpunkt noch lange nicht abgeschlossenen Entwicklung von Java ist es nicht ratsam, die netzwerkspezifischen Komponenten des Archives zu fest mit dem derzeitigen von Java zur Verfügung gestellten Zugriffsmechanis-

mus auf verteilte Objekte RMI[18] zu verdrahten. Eine möglichst vollständige Abkapselung der Netzwerkkomponente hat daher das Ziel, einen späteren Wechsel vom anfänglich verwendeten RMI zu anderen Architekturen wie CORBA[19] ohne Änderungen der Anwendungsprogramme oder des Archives zu ermöglichen.

5.1.2.3 Flexibilität bei der Erreichung von Persistenz

Das Archiv sollte bei dem Problem der Persistenz den vielfältigen Anwendungsfällen möglichst angemessene Lösungen anbieten. Es wird wohl wenig Sinn machen, einem Studienarbeitsprojekt wie dem Pausenplaner für die Benutzung des Archives die Installation eines Oracle-Datenbanksystems zur Vorraussetzung zu machen. Auf der anderen Seite würde die Kombination aus JConLib[20] und Serialisierungsmechanismus kaum die Anforderungen eines Krankenhaus-Informations-Systems erfüllen können.

5.1.2.4 Einfache Anpassung schon vorhandener Programme

Eine nachträgliche Erweiterung eines schon bestehenden Programmes um die Funktionalität des Archives darf keine umfassenden Änderungen der vorhandenen Material- oder Werkzeugklassen nach sich ziehen. So würde beispielsweise eine Aspektklasse ***Archivierbar***, von der sich alle im Archiv zu speichernden Materialien ableiten den Entwurf des Archives vereinfachen, demgegenüber aber den Anwendungsprogrammierer dazu zwingen, grundlegende Veränderungen seiner Materialklassen durchzuführen.

5.2 Entwurfsziele und Schnittstellengestaltung

Bei dem Entwurf einer Rahmenwerkkomponente gibt es neben der möglichst vollständigen Erfüllung der fachlichen und technischen Anforderungen noch mehrere zusätzliche Ziele und Vorgaben, die sich allerdings zum Teil gegenseitig ausschließen. Eine dieser Zielsetzungen ist die Gewährleistung größtmöglicher Flexi-

18 RMI: **R**emote **M**ethod **I**nvocation
19 CORBA: **C**ommon **O**bject **R**equest **B**roker **A**rchitecture
20 JConLib: von Holger Bohlmann entwickelte Container-Bibliothek für das JWAM-Rahmenwerk

bilität, die die Benutzung des Archives auch in einem breiten Spektrum von Anwendungsfällen erlaubt. Dieses Ziel steht andererseits im Widerspruch zu dem Wunsch nach einer Benutzerschnittstelle, die von vornherein versucht, semantische Fehler in der Benutzung so weit wie möglich auszuschließen und auf diese Weise den Anwender[21] zu entlasten.

Da unser Archiv auch im Kontext des schon mehrfach erwähnten »Java/CORBA«-Projektes entstanden ist und damit den unterschiedlichsten Anwendungen zur Verfügung stehen sollte, haben wir uns in der Gestaltung unseres Archives zugunsten der Flexibilität entschieden. Zum anderen sollte es unserer Ansicht nach auch im Sinne des WAM-Leitbildes liegen, dem Benutzer eine größere Anzahl von Werkzeugen oder Bausteinen zur Verfügung zu stellen, mit deren Hilfe er eigenverantwortlich seine Vorstellungen verwirklichen kann. Das WAM-Leitbild vom Benutzer, der als Experte seines Aufgabengebietes mit Hilfe verschiedener Werkzeuge die ihm vorliegenden Aufgaben bewältigt, ist unserer Meinung nach nicht nur für einen Endbenutzer eines Softwaresystems gültig, sondern ebenfalls auch auf den Entwickler dieser Anwendungen übertragbar. Gerade bei einer so komplexen Aufgabe wie der Entwicklung von Software ist es kaum möglich, sämtliche in Betracht kommenden Probleme sowie deren „richtige“ Lösung vorrauszusehen, beziehungsweise „falsche“ Vorgehensweisen von vornherein zu verhindern.

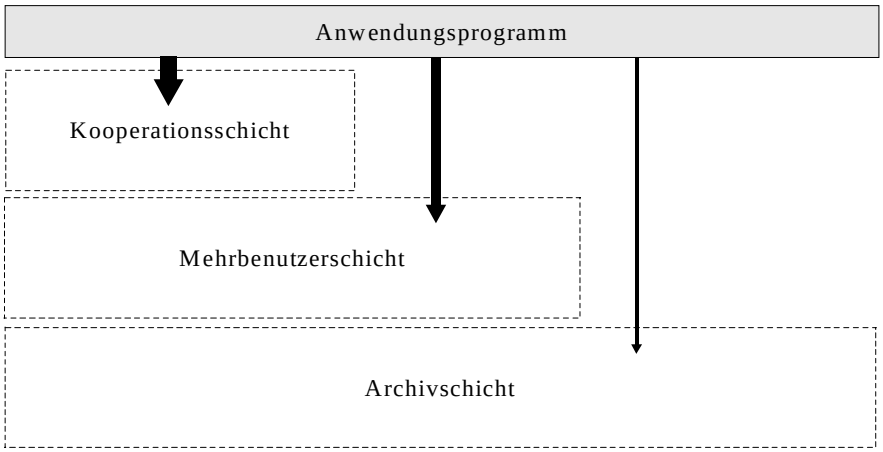
Auf der anderen Seite liegt jedoch der Daseinszweck eines Rahmenwerks beziehungsweise einer Komponente davon gerade darin, dem Benutzer ein Gerüst zur Verfügung zu stellen, das ihm sowohl einen Teil seiner Arbeit abnehmen, als auch ihm helfen sollte, diese Arbeit an einem vom Rahmenwerk vorgegebenen Muster oder Leitbild auszurichten.

Um diese beiden sich widersprechenden Ziele so gut wie möglich miteinander zu vereinbaren, haben wir beim Entwurf der Programmierschnittstellen versucht, eine Art Schichtenmodell ähnlich dem GKS-Modell[22] in der Computergrafik zu realisieren.

---

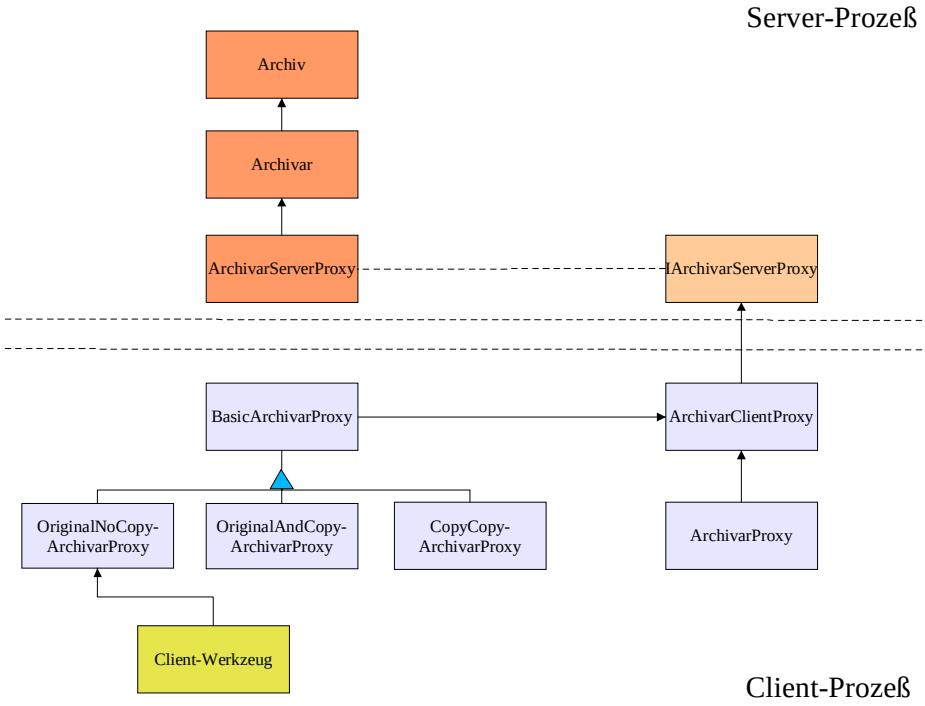
21 Wenn wir im Zusammenhang mit dem Archiv von dem „Anwender“ oder „Benutzer“ sprechen, ist in der Regel nicht der Endbenutzer eines Softwaresystems gemeint, sondern ein Anwendungsprogrammierer

22 GKS: „Graphical Kernel System“ Eine Standardspezifikation für Computer-Grafik



Die unterste Schicht wird von der Archivschicht gebildet. Diese Schicht bietet für ein im Einzelplatzsystem eingesetztes Archiv die benötigten Basisdienste wie Ausgabe eines Inhaltsverzeichnisses, Materialausgabe, Materialneuaufnahme und Materialrücknahme an. Auf der Archivschicht baut die Mehrbenutzerschicht auf, die die bereitgestellten Dienste um die wie von ihrem Namen versprochene Mehrbenutzerfähigkeit erweitert. Dabei handelt es sich um Benutzerverwaltung, Verwaltung von Kopien sowie einer einfachen Transaktionslogik. Die Mehrbenutzerschicht bietet zudem alle für die unterschiedlichen Kooperationsformen benötigten Zugriffsarten an, ohne jedoch deren konsistente Anwendung zu erzwingen.
Die oberste Schicht wird von der Kooperationsschicht gebildet. Diese Schicht stellt dem Benutzer die Dienste des Archives im Sinne der in Abschnitt 5.3.1.1. erläuterten Kooperationsformen zur Verfügung. Hierbei greift die Kooperationsschicht auf die Dienste der darunterliegenden Mehrbenutzerschicht zurück, um mit diesen durch Gruppierung und Erweiterungen konsistente Kooperationsmodelle zu bilden. Bei Gebrauch der Kooperationsschicht legt sich der Benutzer auf eine Kooperationsform fest und schließt dadurch in dieser Kooperationsform nicht vorgesehene Arten des Zugriffes aus.

5.2.1 Die Architektur des Archives



Die technische Grundlage des Archives bildet eine Client/Server-Architektur. Den Kern dieser Architektur bilden auf der Server-Seite das ***Archiv*** und der ***Archivar***. Das ***Archiv*** bietet die Grundfunktionalität (Materialausgabe, Materialannahme, Inhaltsverzeichnisse usw.) an und realisiert damit die in Abschnitt 8.2.1. skizzierte Archivschicht. Der ***Archivar*** benutzt das ***Archiv*** und erweitert dessen Dienste um die in der Mehrbenutzerschicht angebotene Funktionalität. Auf die von dem Server bereitgestellte Funktionalität wird auf der Client-Seite über ein Stellvertreterobjekt (Proxy) zugegriffen. Der ***ArchivarClientProxy*** kapselt auf der Client-Seite die für die Netzwerkkommunikation erforderlichen Zusatzobjekte und Methoden, während der ***ArchivarServerProxy*** auf der Server-Seite die von dem Interface ***IArchivarServerProxy*** definierte Schnittstelle um die benötigte Netzwerkfunktionalität erweitert.

Der Benutzer kann auf die gesamte Funktionalität des ***Archivars*** über den ***ArchivarProxy*** zugreifen, der damit die Mehrbenutzerschicht auf der Client-Seite zur Verfügung stellt. Die Kooperationsschicht wird schließlich durch die drei Klassen ***OriginalNoCopyArchivarProxy***, ***OriginalAndCopyArchivarProxy*** und ***CopyCo-***

pyArchivarProxy implementiert. Diese drei Klassen erben von dem ***BasicArchivarProxy*** die von allen genutzte Funktionalität wie Benutzerverwaltung und Transaktionslogik und erweitern diese um die für das jeweilige Kooperationsmodell benötigten Zugriffsmethoden. Der Benutzer kann innerhalb eines Prozeßraumes nur von einer dieser drei Klassen eine Instanz erzeugen.

5.3 Die Erfüllung der Anforderungen

5.3.1 Erfüllung der fachlichen Anforderungen

5.3.1.1 Unterstützung der unterschiedlichen Kooperationsformen

Das Archiv bietet dem Benutzer zwei Möglichkeiten, den Umgang mit dem Archiv gemäß der von ihm gewählten Kooperationsform zu gestalten. Im Normalfall wird der Anwender mit der von der Kooperationsschicht angebotenen Funktionalität auskommen. Wie schon im vorangegangenen Kapitel erläutert, stehen dem Benutzer hier drei Klassen zur Verfügung, die jede ein einem der drei Kooperationsmodelle entsprechendes Archiv auf der Client-Seite zur Verfügung stellen. Im Gegensatz zu einem dynamischen Ansatz, in dem der Benutzer mittels Parameter das Archiv mit der gewünschten Kooperationsform initialisiert, kann durch die Kapselung eines Kooperationsmodells in einer eigenen Klasse schon zur Kompilierzeit ein großer Teil der möglichen Zugriffsfehler ausgeschlossen werden. Die Instantiierung einer Proxy-Klasse (***OriginalNoCopyArchivarProxy***, ***OriginalAndCopyArchivarProxy*** oder ***CopyCopyArchivarProxy***) sorgt also dafür, daß die Zugriffe auf dieses Proxy-Objekt mit der Kooperationsform im Einklang stehen. Weitaus schwieriger zu beantworten ist jedoch die Frage, inwieweit diese Instantiierung das System oder Teile davon auf eine Kooperationsform festlegen darf. Eine radikale Lösung wäre die Einbeziehung der Server-Seite des Archives, also eine Parametrisierung der Klasse ***Archivar*** mit einer Kooperationsform. Dies würde allerdings bedeuten, daß alle Werkzeuge, die ihre Materialien aus dem

Archiv beziehen, an diese einzige Kooperationsform gebunden wären. Demnach wäre es nicht mehr möglich, in unterschiedlichen Teilen des Systems verschiedene, dem Aufgabenbereich des jeweiligen Teilsystems entsprechende Kooperationsformen zu gestatten oder etwa die Zugriffsarten vom Materialtyp abhängig zu machen. Eine endgültige Beantwortung dieser Fragen gehört wohl zu den Themen, deren Bearbeitung zu weit über den Rahmen dieser Studienarbeit hinausgegangen wäre. Aus diesem Grund haben wir den Gültigkeitsbereich einer dieser Kooperationsformen recht willkürlich auf den jeweiligen Prozeßraum der Anwendung beschränkt. Da wir auch keine dynamische Parametrisierung zur Laufzeit zulassen, ist sichergestellt, daß zwei Instanzen des selben Anwendungsprogrammes die gleiche Kooperationsform anbieten. Um zu verhindern, daß innerhalb eines Prozeßraumes unterschiedliche Werzeuge mit verschiedenen Kooperationsformen benutzt werden, können innerhalb dieses Prozeßraumes nur von genau einer der drei Kooperationsformklassen Objekte erzeugt werden.
Sollten die von der Kooperationsschicht gebotenen Möglichkeiten für einen speziellen Anwendungsfall nicht ausreichen, kann der Benutzer über den ***ArchivarProxy*** direkt auf die darunterliegende Mehrbenutzerschicht zugreifen. Der ***ArchivarProxy*** bietet alle Zugriffsmethoden an, während die Klassen der Kooperationsschicht nur eine für ihre Kooperationsform benötigte Teilmenge bereitstellen. Bei der Benutzung des ***ArchivarProxys*** liegt es jedoch in der Verantwortung des Anwenders, eine konsistente Verwendung der Zugriffsmethoden einzuhalten.

Das Archiv stellt in der Kooperationsschicht und auch in der Mehrbenutzerschicht die für die unterschiedlichen Kooperationsformen benötigten Zugriffsmethoden zur Verfügung und kann dadurch als Kooperationsmittel eingesetzt werden. Das Archiv kann allerdings nur in sehr begrenztem Umfang die kooperative Arbeit an verteilten Materialien direkt unterstützen. Eine wirksame, direkte Unterstützung kooperativer Arbeit kann nur mit dem Wissen über die fachlichen Zusammenhänge erfolgen. Das Archiv kann nicht entscheiden, welche Veränderung eines Materials zu welchen Auswirkungen auf das Anwendungssystem führen muß. Das Archiv kann lediglich die Umgebung über Zugriffe auf die von ihm verwalteten Materialien

informieren. Aufgrund dieser Informationen kann das Anwendungssystem nach den vom Anwendungsprogrammierer festgelegten Regeln entscheiden, welche weiteren Schritte zur Kooperation erfolgen müssen. Als Zugriff gelten hierbei nur die Herausgabe und Rückgabe von Materialien (Original oder Kopie) aus dem Archiv. Der Versand von Materialien von einem Werkzeug oder Arbeitsplatz zum nächsten sowie Veränderungen an den Materialien werden ausdrücklich nicht berücksichtigt.

5.3.1.2 Transaktionen / Atomare Handlungen

Die Durchführung von Transaktionen ist eine der Leistungen der Mehrbenutzerschicht. Eine Transaktion ist ein Auftrag A, der sich im allgemeinen in mehrere unteilbare Teilaufträge a_1, a_2, ... a_n zergliedert, die sequentiell auszuführen sind. Eine Transaktion, die auf einen konsistenten Datenbestand angewandt wurde, muß nach ihrer vollständigen Abarbeitung diesen Datenbestand ebenfalls in einem konsistenten Zustand hinterlassen [23]. Ausgehend von diesem Transaktionsbegriff bietet die Mehrbenutzerschicht dem Anwender zwei Arten von Sperren an, mit deren Hilfe er eine unteilbare Folge von Handlungen vornehmen kann, in deren Verlauf kein anderer Benutzer den Zustand der betroffenen Materialien verändern kann. Zum einen kann ein Anwender das gesamte Archiv sperren, so daß er als einziger Benutzer den uneingeschränkten Zugriff auf im Archiv vorhandene Originale hat. Für alle anderen Benutzer sind alle Zugriffe, die den Zustand der von der Transaktion betroffenen Materialien im Archiv verändern, oder die Verfügbarkeit der im Archiv vorhandenen Materialien einschränken, gesperrt. Konkret gibt es drei Zugriffe, die während einer Transaktion für die anderen Benutzer gesperrt sind: Das Ausleihen eines Originals, das Überschreiben eines Originals durch eine Version einer Kopie dieses Originals sowie das Löschen eines Originals. Demgegenüber sind mehrere andere Arten des Zugriffes gestattet: Das Ausleihen einer Kopie, das Zurückstellen eines Originals sowie das Hinzufügen eines neuen Materials.
Das Ausleihen einer Kopie birgt für den Entleiher zwar die Gefahr in sich, eine veraltete oder inkonsistente Version des Originals zu erhalten, hat aber auf den

23 Jessen/Valk, S. 147

Zustand des Materials für den die Transaktion durchführenden Benutzer keinen Einfluß, da der Entleiher der Kopie diese im Verlauf der Transaktion nicht als neue Version des Originals zurückschreiben kann.
Das Zurückstellen eines Originals ist möglich, da der Benutzer dieses Original nur vor Beginn der Transaktion entliehen haben konnte. Daher kann es für den Durchführenden der Transaktion nur von Vorteil sein, nun auch über das zurückgestellte Original verfügen zu können. Sollte der Benutzer oder Prozeß, der die Transaktion durchführt, von der Verfügbarkeit eines bestimmten Originals abhängig sein, kann das Zurückstellen dieses Originals durch einen anderen Benutzer eine Verklemmung verhindern.
Auch das Hinzufügen eines neuen Materials sollte die Arbeit des Transaktionsdurchführenden nicht behindern, da auch dieses nur ein Zugewinn an Information bedeuten kann.
Der Vorteil einer Transaktionsart, die das gesamte Archiv für bestimmte Zugriffe der anderen Benutzer sperrt, liegt darin, daß der durchführende Benutzer nicht immer vorher weiß, welche Materialien von seiner Handlungsfolge betroffen sein werden. Demgegenüber sind die Nachteile dieser Art von Transaktion in einem System mit einer größeren Anzahl von Benutzern, die eine größere Anzahl von Transaktionen durchführen wollen, offensichtlich. In solch einem System würde der ursprüngliche Daseinszweck des Archives, als Hilfsmittel der kooperativen, und damit auch parallelen Arbeit zu dienen, vollkommen verloren gehen. Aus diesem Grund stellt das Archiv eine zweite Transaktionsart zur Verfügung. Diese Transaktionsart unterscheidet sich von der ersten dadurch, daß sie dem Durchführenden den exklusiven Zugriff nur auf eine zu Beginn der Transaktion festgelegte Teilmenge von Materialien gewährt. Für die anderen Benutzer unterliegt der Zugriff auf diese Materialien den gleichen Einschränkungen wie im Fall der ersten Transaktionsart. Während es bei der ersten Transaktionsart, der *unbegrenzten Transaktion*, immer nur eine solche Transaktion zu einem Zeitpunkt geben kann, können von der zweiten Transaktionsart, der *begrenzten Transaktion*, beliebig viele parallel durchgeführt werden. Hierbei gilt allerdings die Einschränkung, das keine Transaktion begonnen werden kann, die für sich ein Material reserviert, das

von einer anderen schon aktiven Transaktion bereits gesperrt worden ist. Diese Einschränkung soll auf verhältnismäßig einfache Art und Weise mögliche Verklemmungen verhindern, auch wenn dieses zu Lasten des Anteiles an paralleler Arbeit geht.
Um möglichen Konflikten zwischen den Zugriffsrechten von begrenzten und unbegrenzten Transaktion vorzubeugen, ist die gleichzeitige Durchführung beider Transaktionsarten ausgeschlossen worden. Eine begrenzte Transaktion kann also nur dann begonnen werden, wenn gerade keine unbegrenzte Transaktion durchgeführt wird. Ebensowenig ist es möglich, eine unbegrenzte Transaktion vor Ablauf aller begrenzten Transaktionen zu beginnen.
Das derzeitige Archiv stellt keine Commit/Rollback-Mechanismen zur Verfügung, die das Archiv im Fall eines Transaktionsabbruches auf den zu Beginn der Transaktion gültigen, konsistenten Zustand zurücksetzen. Ein solcher Mechanismus ist zwar aus fachlicher Sicht wünschenswert, gehört jedoch zu den Bereichen, auf die angesichts des begrenzten Umfanges einer Studienarbeit verzichtet wurde.

5.3.1.3 Die logische Gliederung des Archives

Eine Gliederung des Archives kann durch die Verwendung von Unterabteilungen oder Kategorien vorgenommen werden. Jedes Material, das dem Archiv neu hinzugefügt wird, kann einer frei wählbaren Kategorie zugeordnet werden. Das Konzept der Kategorie zeichnet sich unter anderem durch seine überschaubare Komplexität aus: Die Kategorie eines Materials ist lediglich eine Zeichenkette, die als zusätzliches Attribut diesem Material zugeordnet wird. Der Vorteil dieser äußerst einfachen Lösung liegt in ihrer großen Flexibilität. Es gibt keine Einschränkungen bezüglich der Anzahl, der Namenslänge oder der Tiefe der Verschachtelung der Kategorien. Das Archiv bietet über diese einfache Grundlage hinaus keine weitere Funktionalität zur Verwaltung dieser Kategorien an. Für das Archiv gibt es keine mehrstufigen Kategorie-Ebenen oder Unterverzeichnisse. Auch bietet das Archiv keine Möglichkeit zur Vergabe von Zugriffsrechten auf einzelne Kategorien an.

Diese höheren Aufgaben können meiner Ansicht nach jedoch relativ einfach und dabei genau dem jeweiligen Anwendungsfall entsprechend von den Anwendungsprogrammen übernommen werden.

5.3.1.4 Keine Einschränkungen bezüglich der Art der Materialien

Eine der wichtigsten Anforderungen an das Archiv ist es, möglichst jede Art von Materialien verwalten zu können. Für den praktischen Gebrauch des Archives ist es unseres Erachtens nach unannehmbar, von den Benutzern zu verlangen, alle ihre Materialien von einer vom Archiv vorgegebenen Oberklasse erben zu lassen, oder ein breites Interface zu implementieren. Um dieses zu verhindern, wurde der ***MaterialWrapper*** eingeführt. Der ***MaterialWrapper*** dient als Umschlag, in den jedes Material, das dem Archiv entnommen oder hinzugefügt wird, verpackt wird. Jeder Umschlag bekommt eine eindeutige technische ID, über die später der Zugriff im Archiv erfolgt. Zusätzlich kann ein Umschlag noch mit einem (fachlichen) Namen versehen werden, der die Identifikation für den Benutzer erleichtert. Als drittes kann dem Umschlag noch der Name der Kategorie zugeordnet werden, unter der das Material im Archiv aufbewahrt werden soll. Das Archiv verwaltet letztendlich nicht die vielfältigen Materialien, sondern nur die Umschläge deren Inhalt ihm nicht bekannt ist. Der Benutzer muß daher dafür sorgen, daß ein Material, welches er in einem solchen Umschlag »verpackt« aus dem Archiv erhält, später in genau diesem Umschlag wieder in das Archiv zurückgestellt wird. Der ***MaterialWrapper*** ist damit das einzige Bindeglied zwischen den Materialien und dem Archiv und ermöglicht somit die völlige Unabhängigkeit der Materialien vom Archiv, stellt aber auch eine Schwachstelle im Umgang mit dem Archiv dar. Unserer Ansicht nach ist der etwas erhöhte Aufwand im Umgang mit dem Archiv ein akzeptabler Preis dafür, daß sich der Benutzer im Entwurf seiner Materialien ganz auf deren fachliche Anforderungen konzentrieren kann, ohne auf technisch begründete Vorgaben des Archives (das es evtl. zu diesem Zeitpunkt noch gar nicht gegeben hat) Rücksicht nehmen zu müssen.

Da das Archiv keinerlei Kenntnis vom Inhalt der Umschläge hat, kann das Archiv

keine Beziehungen zwischen den Materialien überwachen. Falls beispielsweise das Dokument A einmal als einzelnes Material DA in das Archiv gestellt wird, später aber auch mit anderen Dokumenten zu dem Projekt X zusammengefaßt wird und dieses Projekt als Material PX in das Archiv gestellt wird, gibt es von Seiten des Archives keine Mechanismen, die die Konsistenz des Dokument A sicherstellen. Dem Archiv ist nicht bekannt, daß das als Material DA gespeicherte Dokument A auch als Teil des Materials PX im Archiv vorhanden ist. Änderungen, die im Rahmen des Projektes X an Dokument A vorgenommen werden, werden durch das Archiv nicht automatisch im Material DA übernommen.

Um derartige Beziehungen überwachen zu können, bräuchte entweder das Archiv ein zusätzliches Wissen über fachliche Zusammenhänge zwischen den Materialien, oder die Zusammensetzung der Materialien müßte sich an bestimmten Vorgaben des Archives orientieren. Die erste dieser beiden Möglichkeiten wäre nur mit großem Aufwand zu verwirklichen, da das Archiv nicht nur für einen einzigen, schon vorhandenen fachlichen Kontext ausgelegt sein müßte, sondern für eine Vielzahl von möglichen Kontexten mit bisher unbekannten Anforderungen gerüstet sein sollte. Die zweite Alternative würde wiederum zu der zu Beginn dieses Abschnittes verworfenen Ausrichtung der fachlichen Materialien auf die technischen Anforderungen des Archives führen[24].

Der in diesem Archiv gewählte Weg der völligen Abstrahierung von Inhalt und Struktur der Materialien verlagert die Lösung des Problems der komplexen Materialien auf die Anwendungsseite. In vielen Anwendungsfällen sollte es unserer Meinung nach möglich sein, nicht das komplexe Material als ganzes, sondern nur seine Komponenten als einzelne Materialien in das Archiv zu stellen. Die Informationen über die Zusammenhänge zwischen den einzelnen Materialien werden wiederum als eigene Materialien dem Archiv hinzugefügt. Diese Methode wird beispielsweise auch von den meisten Entwicklungsumgebungen bei ihrer Projektverwaltung benutzt. Die Dateien mit dem Quelltext werden dabei einzeln abgespeichert und sind auch außerhalb des Projektkontextes zugreifbar, während die Projektinforma-

24 Eine tiefergehende Beschäftigung mit der Problematik komplexer Materialien ist in der Studienarbeit von Norbert Schüler und Michael Otto zu finden, die ein spezielles Archiv für komplexe Materialien entwickeln

tionen in einzelnen Projektdateien abgespeichert werden. Änderungen an den Quelltextdateien, die außerhalb des Projektkontextes vorgenommen werden, wirken sich unmittelbar auf den Zustand dieser Queltextdateien im Projektkontext aus. Diese Art der Speicherung eignet sich für komplexe Materialien die eine Baumstruktur mit einer klaren Trennung zwischen Knoten und Blättern aufweisen. Der Nachteil der oben skizzierten Möglichkeit, diese Art von komplexen Materialien mit unserem Archiv zu verwalten, liegt in der im Gegensatz zu den Entwurfszielen nun doch notwendigen Anpassung der Materialien an die Erfordernisse des Archives. Dennoch erscheint es uns für den Anwender vorteilhafter, wenn dieser die notwendigen Anpassungen nach seinen eigenen Vorstellungen realisieren kann, ohne dabei ein vom Archiv vorgegebenes Schema zu befolgen. Eine weitere Schwäche dieses Archives liegt darin, daß komplexe Materialien, bei denen keine Trennung zwischen dem »reinen Inhalt« und den Beziehungen zu anderen Materialien vorgenommen werden kann, nur auf sehr unhandliche Art und Weise vom Archiv verwaltet werden können.
Ein Beispiel für den Umgang des Archives mit komplexen Materialien liefert unser Pausenplaner. Bei dem Pausenplaner haben wir die oben skizzierte Lösung gewählt, die komplexen Materialien vor der Übergabe an das Archiv in ihre einzelnen Materialien zu zerlegen, bzw. diese nach nach der Lieferung durch das Archiv wieder zu komplexen Materialien zusammenzusetzen. Auf die Einzelheiten dieser Lösung gehen wir in Kapitel 6.3. noch näher ein.

5.3.2 Erfüllung der technischen Anforderungen

5.3.2.1 Archiv soll auch in Einzelplatzsystemen einsetzbar sein

Aufgrund der Schichtenarchitektur des Archives ist es ohne weiteres möglich, das Archiv in einem Einzelplatzsystem einzusetzen. In diesem Fall beschränkt sich der Anwender auf die von der Archivschicht angebotenen Dienste, indem er direkt auf ein ***Archiv*** Objekt zugreift.
Das Archiv im Einzelplatzsystem bietet allen Werkzeugen dieses Systems eine

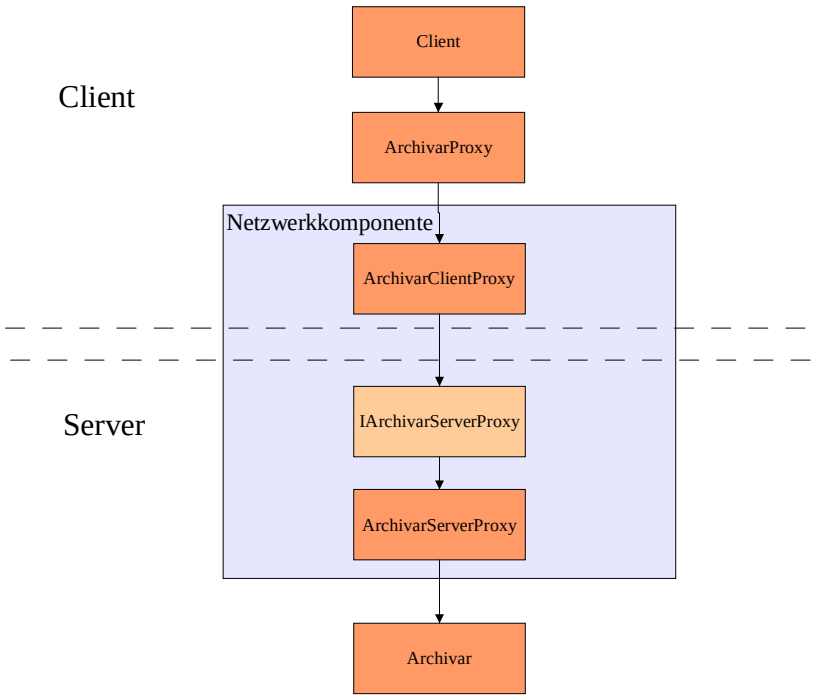
gemeinsame Schnittstelle, über die sie auf unterschiedliche Materialien zugreifen, bzw. ihre Materialien persistent machen können. Im Regelfall erscheint es bei Einzelplatzsystemen nicht sinnvoll, neben den Originalmaterialien auch Kopien zu verwenden. Die Archivschicht bietet jedoch trotzdem die Möglichkeit, Kopien zu erstellen. Aus fachlicher Sicht könnten Kopien allerdings dann sinnvoll sein, wenn mehrere Einzelplatzsysteme über Postkörbe untereinander Materialien versenden, ohne dabei auf ein zentrales Archiv zurückzugreifen. In der »realen« Arbeitswelt kommt es häufig vor, daß ein Dokument zwar nur von einem Mitarbeiter bearbeitet wird, dieser jedoch Kopien des Dokuments »zur Ansicht« an seine Kollegen verteilt.

Der Frage, ob eine Unterstützung dieses Konzeptes besonders zweckmäßig ist, sind wir im Rahmen unserer Studienarbeit nicht weiter nachgegangen. Da diese Funktionalität im Verlauf der Implementation als »Nebenprodukt« angefallen ist, haben wir sie auch dem Anwender zur Verfügung gestellt. Eine weitere Schwierigkeit in der Benutzung von lokalen Archiven und Kopien liegt bei der derzeitigen Implementation darin, daß nur das lokale Archiv, welches die Kopie erstellt hat, überhaupt das Wissen besitzt, daß es sich bei diesem Material um eine Kopie handelt.

5.3.2.2 Austauschbarkeit der Netzwerkkomponenten

Die Unabhängigkeit sowohl der Anwendungen als auch des Archives von der verwandten Netzwerkkomponente wird durch den bei Züllighoven[25] beschriebenen Einsatz von Proxy-Objekten erreicht. Auf der Serverseite wird die Funktionalität des ***Archivars*** durch den ***ArchivarServerProxy*** um die für die Netzwerkkommunikation benötigten Eigenschaften ergänzt. Der ***ArchivarServerProxy*** implementiert die vom Interface ***IarchivarServerProxy*** deklarierte Schnittstelle. Auf diese Schnittstelle bezieht sich das auf Clientseite liegende Proxyobjekt, der ***ArchivarClientProxy.***Der ***ArchivarClientProxy*** wird jedoch nicht direkt vom Werkzeug benutzt, sondern über einen ***ArchivarProxy***. Der Vorteil dieser Konstruktion liegt darin, daß im Falle eines Austausches des Netzwerkmechanismus das Anwen-

25 Züllighoven, Konstruktionshandbuch, S. 527-531

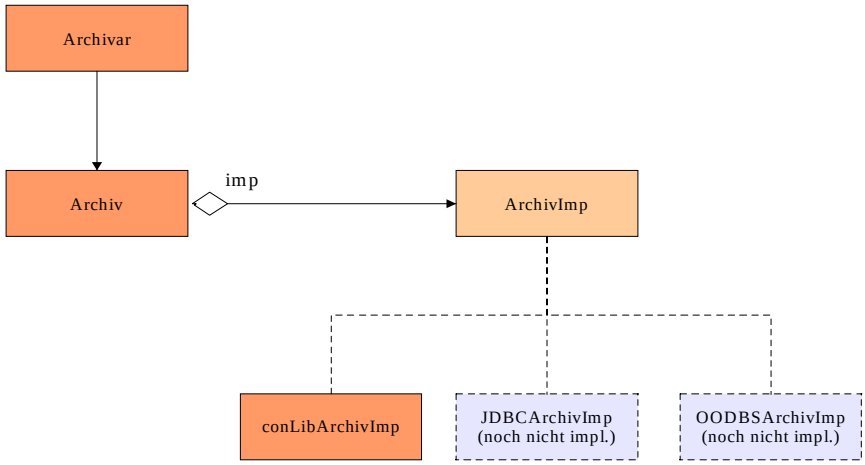


dungsprogramm weiterhin auf das gleiche Proxyobjekt zugreift. Würde die Anwendung direkt auf den ***ArchivarClientProxy*** zugreifen, könnten die notwendigen Änderungen nur innerhalb der Implementierung dieser Klasse vorgenommen werden. Eine Änderung der Schnittstelle des ***ArchivarClientProxy*** hätte andernfalls auch Änderungen innerhalb der Anwendungklassen zur Folge. Durch die Abkapselung des ***ArchivarClientProxys*** durch eine zusätzliche Proxyklasse ist es zudem möglich, verschiedene Clientproxys anzubieten, beispielsweise einen ***RMI-ArchivarClientProxy,*** einen ***CorbaArchivarClientProxy*** und einen ***VoyagerArchivarClientProxy.*** Auf der Serverseite bleibt der ***Archivar*** von jeder Änderung des Netzwerkmechanismus unberührt, da er vom ***ArchivarServerProxy*** nur benutzt wird, und daher von dessen Existenz keine Kenntnis hat.

5.3.2.3 Flexibilität bei dem Erreichen von Persistenz

Bei der Implementierung der physikalischen Abspeicherung des Archivinhalts wird

durch die Verwendung des Brückenmusters[26] die geforderte Flexibilität erreicht.



Die Brückenstruktur ermöglicht es dem Benutzer, eine seinen Bedürfnissen entsprechende Implementation zu wählen. Des weiteren gibt dieses dem Benutzer die Möglichkeit, auch eigene Implementationen mit geringem Aufwand in das Archiv zu integrieren. Diese Ausbaumöglichkeiten sind im Falle Javas von besonderer Bedeutung, da es derzeit noch wenige abgeschlossene Entwicklungen im Bereich Java/Datenbanken gibt, wobei vor allem auf dem Gebiet der objektorientierten Datenbanken erst in Zukunft mit uneingeschränkt einsetzbaren Lösungen zu rechnen ist.

5.3.2.4 Einfache Anpassung schon vorhandener Programme

Um schon vorhandenen Anwendungen die Benutzung des Archives zu ermöglichen, darf die Einbeziehung des Archives keine größeren Veränderungen im Quellcode der Anwendung nach sich ziehen. Diese Änderungen können hauptsächlich in zwei Bereichen auftreten: Zum einen bei der Benutzung der Schnittstelle des Archives, d.h. der Aufruf der entsprechenden Methoden zum Speichern

26 Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software. Deutsche Übersetzung von Dirk Riehle, (Bonn 1996), S. 165

und Anfordern von Materialien; zum anderen bei der Struktur der Materialien. Den von uns gewählten Ansatz zur Vermeidung von Änderungen der Materialien haben wir bereits im Kapitel 5.3.1.4. versucht näher zu erläutern. Die in dem Kapitel erläuterte fachliche Lösung der Materialunabhängigkeit durch den Einsatz von »Umschlägen« bildet gleichzeitig die technische Lösung dieses Problems, da von der fachlichen »Verpackung« des Materials zur technischen Implementation keine größeren Differenzen bestehen. Neben der Frage des Materials ist die Benutzung der Archivschittstelle der zweite Bereich, der den vorhandenen Quellcode des Anwendungsprogrammes betrifft. Die hier notwendigen Veränderungen können durch die Kapselung des Archivzugriffes in einer auf die Anwendung zugeschnittenen Schicht auf ein erträgliches Maß reduziert werden. Dies ist auch der Weg, den wir bei der Entwicklung unseres Pausenplaners beschritten haben. Der Zugriff auf das Archiv wurde erst implementiert, nachdem die meisten Werkzeuge des Pausenplaners fertiggestellt waren. Die Werkzeuge greifen über eine einfache Schnittstelle auf den ***ppMaterialVersorger*** zu, der wiederum den Materialverkehr mit dem Archiv abwickelt. Die dafür im Nachhinein notwendigen Veränderungen des Quelltextes der Werkzeuge beschränkten sich auf ein paar Zeilen Code. Der etwas umständliche Umgang mit den ***MaterialWrappern*** wird vollständig vom ***ppMaterialVersorger*** erledigt. Auf die Einzelheiten dieser Lösung werden wir noch im Kapitel 6.4.1. näher eingehen.

6 Implementation des Pausenplan-Systems

Dieses Kapitel gliedert sich in fünf Abschnitte. Im ersten Abschnitt möchten wir auf die Implementierung der aus den Visionen schon bekannten Werkzeuge eingehen. Bevor wir uns in Abschnitt 6.3 der Implementation des Materials widmen, wollen wir die von uns erstellte Materialverwaltung vorstellen. In den beiden letzten Abschnitten diese Kapitels gehen wir noch näher auf den Automaten ***ppMaterialversorger*** und auf die Unterstützung von Kooperation im Pausenplaner-System ein. Letzteres soll zusätzlich als Erläuterung für das Zusammenspiel der vorher einzeln behandelten Werkzeuge, Automaten und Materialien dienen, die gemeinsam das Pausenplanersystem bilden.

6.1 Implementation der Werkzeuge

Für die Werkzeugkonstruktion nach der WAM-Metapher hält das JWAM-Rahmenwerk im Wesentlichen drei Oberklassen bereit. Die Interaktionskomponente des Werkzeugs beerbt die Klasse ***ipObject***, während die Funktionskomponente von der Klasse ***fpObject*** erbt. Die dritte Klasse ***toolObject*** wird zum Starten der Werkzeuge über den sog. elektronischen Schreibtisch benötigt.

Weil wir durch die Arbeiten von Tim Krauß und Andreas Hartmann schon eine sehr genaue Vorstellungen vom Aussehen der Werkzeuge und ihrem Umgang mit den Materialien gewonnen hatten, konzentrierten wir uns in der ersten Implementationsphase des Pausenplaner-Systems auf die Konstruktion der Werkzeuge. Da sich die grundlegenden Implementationsschritte für jedes Werkzeug kaum von einander unterscheiden, soll der folgende Abschnitt am Beispiel des Pausenlisten-Berarbeiter Werkzeugs veranschaulichen, wie wir bei der Entwicklung der Werkzeuge mit dem JWAM-Rahmenwerk vorgegangen sind. Er gliedert sich in fünf Themenschwerpunkte. Im ersten Teil gehen wir auf die Besonderheiten bei der Gestaltung der grafischen Oberflächen mit dem JWAM-Rahmenwerk ein. Danach möchten wir kurz vorstellen, wie man eigene Präsentationsformen in das Rahmenwerk integrieren kann. In den nächsten drei Abschnitten gehen wir dann auf die eigentliche Werkzeugkonstruktion ein. Hier veranschaulichen wir, wie die konkreten Tool-,

FK-, und IAK-Klassen implementiert werden.

Da der Pausenplaner ein vergleichsweise kleines System mit nur wenigen Werkzeugen ist, haben wir auf die Verwendung von Aspektklassen bei der Koppelung von Werkzeugen und Materialien verzichtet. Die in unserem Pausenplanersystem eingesetzten Werkzeuge sind zudem jeweils speziell auf ein Material zugeschnitten, dessen ohnehin nicht allzu komplexe Schnittstelle kaum über die Schnittstelle einer möglichen Aspektklasse hinausgeht.

Im letzten Teil dieses Kapitels möchten wir schon einmal grob schildern, wie die einzelnen Werkzeuge des Pausenplaner-Systems von den Materialveränderungen anderer Werkzeuge unterrichtet werden. Wir werden diesen Aspekt in Abschnitt 6.5 noch einmal, dann aber ausführlicher, beleuchten.

6.1.1 Erstellen der grafischen Oberfläche

In dem folgenden Abschnitt befassen wir uns mit der Erstellung der grafischen Oberflächen. Dabei werden wir uns in den ersten zwei Unterabschnitten mit den beiden grundlegenden Prinzipen für die GUI-Unterstützung des JWAM-Rahmenwerkes beschäftigen. Anschließend schildern wir, wie man eigene GUI-Elemente zum Rahmenwerk hinzufügen kann. Dies veranschaulichen wir am Beispiel unserer für das Pausenplaner-System benötigten Präsentationsformen pfTextLabel und pfTable.

6.1.1.1 Trennung in Interaktion und Präsentation

Die Anbindung der Oberfläche an die Interaktionskomponente des Werkzeugs geschieht im JWAM-Rahmenwerk über ein vierstufiges Design (siehe Abbildung 7[27]). Dabei wird zwischen den eigentlichen Interaktionsformen wie „Auswahl“, „Aktivierung“, Ausfüllen“, etc. und deren Präsentation an der Oberfläche (etwa Button, Liste, etc.) unterschieden.

Die Interaktionskomponente kümmert sich damit nur um Interaktionsformen und

27 Aus Martin Lippert: Konzeption und Realisierung eines GUI-Frameworks in Java nach der WAM-Metapher. Studienarbeit Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik 1997, S. 16

ist von der tatsächlicher Repräsentation an der Oberfläche weitestgehend entkoppelt. Es ist möglich, daß einer Interaktion mehrere Präsentationen zugeordnet sind. Als Beispiel sei hier das Beenden des Werkzeuges genannt, welches über

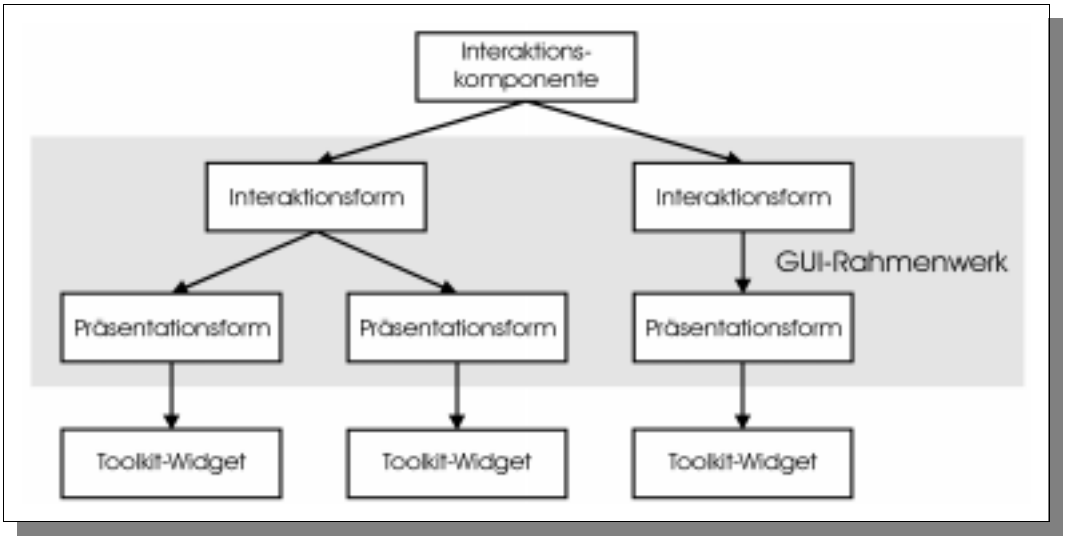


Abbildung 7: Struktur der Oberflächenanbindung an die Interaktionskomponente des Werkzeugs

einen Button oder etwa durch die Auswahl eines Menü-Eintrages geschehen kann. Umgekehrt ist es jedoch durchaus denkbar, daß eine Präsentationform zwei oder mehr Interaktionensformen unterstützt, so kann zum Beispiel eine Liste durch Doppelklick aktiviert oder ein Eintrag durch einfachen Klick ausgewählt werden.

Um dieses Design zu unterstützen ist die Verwendung der speziellen Oberflächenklassen des JWAM-Rahmenwerkes bei der Gestaltung der GUIs[28] erforderlich. Andere Komponenten dürfen zwar verwendet werden, können zur Laufzeit aber nicht angesprochen (bzw. deren Attribute können nicht verändert) werden. Zusätzlich hat die Benutzung der JWAM-Komponenten den großen Vorteil, daß man von dem zugrunde liegenden System und der Entwicklungsumgebung unabhängig wird. Im Zusammenhang mit dem Serializer-Bean (siehe Abschnitt 6.1.1.2), gelingt die Abkapselung von einer einzigen Entwicklungsumgebung fast vollständig.

Die zur Zeit vorhandene Auswahl an implementierten Präsentationsformen läßt allerdings noch keine Entwicklung von ansprechend gestalteten Oberflächen zu, da bisher nur die absolut notwendigsten GUI-Elemente vom JWAM-Framework zur

28 Graphical User Interface

Verfügung gestellt werden. Der für die Konstruktion eigener Präsentationsformen nötige Programmieraufwand hält sich jedoch in Grenzen, sofern die gewünschte Präsentationsform nicht zu komplex ist. In Abschnitt 6.1.1.3 werden wir näher auf die Erstellung eigener Präsentationsformen eingehen.

6.1.1.2 Die Verwendung des „Serializer-Bean“

Einer der großen Vorteile der Programmiersprache JAVA ist zweifelsohne die Unabhängigkeit bezüglich der zu Grunde liegenden Rechner-Plattform. Sofern eine virtuelle JAVA-Maschine (bzw. ein JDK) für die entsprechende Plattform verfügbar ist, gilt dies sowohl für die Programmausführung, als auch für die Programmentwicklung.

Bei der Konstruktion grafischer Oberflächen ist man in der Regel auf eine ganz spezielle Entwicklungsumgebung angewiesen, es sei denn, man macht sich die Mühe, die einzelnen Komponenten der Oberfläche in primitivem Source-Code von Hand einzugeben. Dies resultiert aus der Tatsache, daß der überwiegende Teil der Tools ihre eigene Beschreibungssprache für die Oberflächen verwendet, welche nicht (bzw. nur mit sehr hohem Aufwand) untereinander austauschbar oder konvertierbar sind. Mag diese Bindung an eine bestimmte Entwicklungsumgebung für den einzelnen Programmierer noch vertretbar sein, ist sie bei größeren Software-Projekten, mit einer Vielzahl von Programmierern, nicht mehr akzeptabel. Damit eine heterogene Zusammenstellung der Entwicklungstools möglich und der Source-Code untereinander austauschbar ist, erstellte Martin Lippert sein Konzept des „Serializer-Beans“[29]. Es stützt sich auf die Verwendung der Java-Beans[30] Architektur und dem Mechanismus der Objektserialisation[31] und sorgt im Wesentlichen dafür, daß der Benutzer jeden JAVA-Beans fähigen GUI-Editor zur Konstruktion seiner Oberflächen verwenden kann. Er braucht seiner Oberfläche dann nur noch das sog. „Serializer-Bean“ hinzufügen. Sobald er das Serializer-bean aktiviert, wird die gesamte Oberfläche serialisiert und gespeichert. Das resultierende File wird dann später vom Werkzeug wieder eingelesen und zur Darstellung der

29 Lippert, S. 31-33
30 Javasoft: Java-Beans 1.0 API specification. Sun Microsystems, 1996
31 Javasoft: Java Object Serialization specification. Sun Microsystems, 1996

Oberfläche genutzt.

6.1.1.3 Hinzufügen eigener Präsentationsformen zum Rahmenwerk

Der Aufwand für das Hinzufügen eigener Präsentationsformen zu den im Rahmenwerk schon gegebenen richtet sich in erster Linie nach ihrer Komplexität. Ist die von der neuen Präsentationsform verwendete Interaktionsform schon im Rahmenwerk vorhanden, so beschränkt sich die eigene Arbeit auf das Implementieren des entsprechenden Interfaces. Dies ist bei der von uns erstellten Präsentationsform ***pfTextLabel*** (siehe Abschnitt 6.1.1.4) der Fall. Bei unserer ***pfTable*** (Abschnitt 6.1.1.5) hingegen haben wir zusätzlich zur eigentlichen Präsentationsform die Interaktionsform, deren Interface und eine eigene Kommando-Klasse selbst geschrieben.

6.1.1.4 Präsentationsform „pfTextlabel“

Für die Anzeige von diversen Statusinformationen (wie Anzahl der Lehrer im Lehrkörper, deren Einteilungen, etc.) benötigten wir Text-Labels, deren Wert, in diesem Fall der angezeigte Text, wir zur Laufzeit dynamisch verändern konnten. Da das JWAM-Rahmenwerk[32] hierfür noch keine passende Präsentationsform bereit hält, mußten wir diese selbst erstellen.

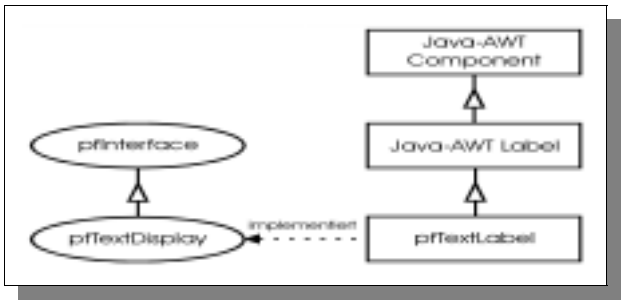


Abbildung 8: Vererbungshierarchie der pfTextLabel

Die Implementation gestaltete sich jedoch recht einfach. Zum einen ist die passende Interaktionsform ***ifTextDisplay*** im Rahmenwerk vorhanden, und zum anderen ent-

32 In der von uns benutzten Version 1.2

hält die Java AWT[33] schon eine Komponente ***Label***, welche die gewünschten Eigenschaften besitzt. Daher muß unsere Präsentationsform ***pfTextLabel*** nur von dem ***Label*** aus der AWT erben und das Interface ***pfTextDisplay*** implementieren (siehe auch Abbildung 8). Abbildung 9 zeigt einen kleinen Ausschnitt aus dem Source-Code, der zeigen soll, daß es sich in diesem Fall größten Teils um eine Adapter-Klasse[34] handelt, welche die Schnittstellen von ***pfTextDisplay*** und der schon vorhandenen AWT-Klasse (***Label***) zusammenführt. Zusätzlich enthält die Klasse nur noch einige Methoden nach der Java-Bean-Spezifikation[35], um spezielle JWAM-Eigenschaften (wie den Namen der Präsentationsform) in der Entwicklungsumgebung zugreifbar zu machen.

33 Abstract Windowing Toolkit
34 Gamma et al, S. 151-163
35 Javasoft: Java-Beans 1.0 API specification. Sun Microsystems 1997

```
public class pfTextLabel extends Label implements pfTextDisplay
{
  //Konstruktor
  public pfTextLabel(String text)
  {
    super(text);
    [...]
  };

  [...]

  //Methode um den Text zu ändern
  public void setText (String text)
  {
    super.setText(text);
  };
```

*Abbildung 9: Source-Code **pfTextLabel***

6.1.1.5 Präsentationsform „pfTable“

Das Erstellen der von uns für das Pausenplan-Bearbeiter Werkzeug benötigten Präsentationsform einer Tabelle war etwas aufwendiger, als das der im Vorangegangenen Abschnitt beschriebenen ***pfTextLabel***. Zum einen existierte keine entsprechende Komponente in der Java-AWT und zum anderen war auch keine passende Interaktionsform im JWAM-Rahmenwerk vorhanden. Es existiert zwar die Interaktionsform ***if1fromNSelection***, diese reicht für eine Tabelle aber nicht aus, da wir hierfür eine Auswahl von eins aus n mal m Möglichkeiten benötigen. Im Einzelnen mußten daher von uns folgende Klassen (bzw. Interfaces) bereitgestellt werden.

- ***pfTable***

  Hierbei handelt es sich um die eigentliche Präsentationsform. Da es nicht möglich war, sich im Rahmen dieser Studienarbeit intensiv mit der Konstruktion von Komplexeren Präsentationsformen[36] im allgemeinen oder mit der einer Tabelle im speziellen auseinander zu setzen, entschieden wir uns für ein möglichst einfaches Design.

  Wie bei der ***pfTextLabel*** benutzten wir auch hier die grundlegenden Oberflä-

36 Sven Lammers: (Titel noch unbekannt) Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik, 1998

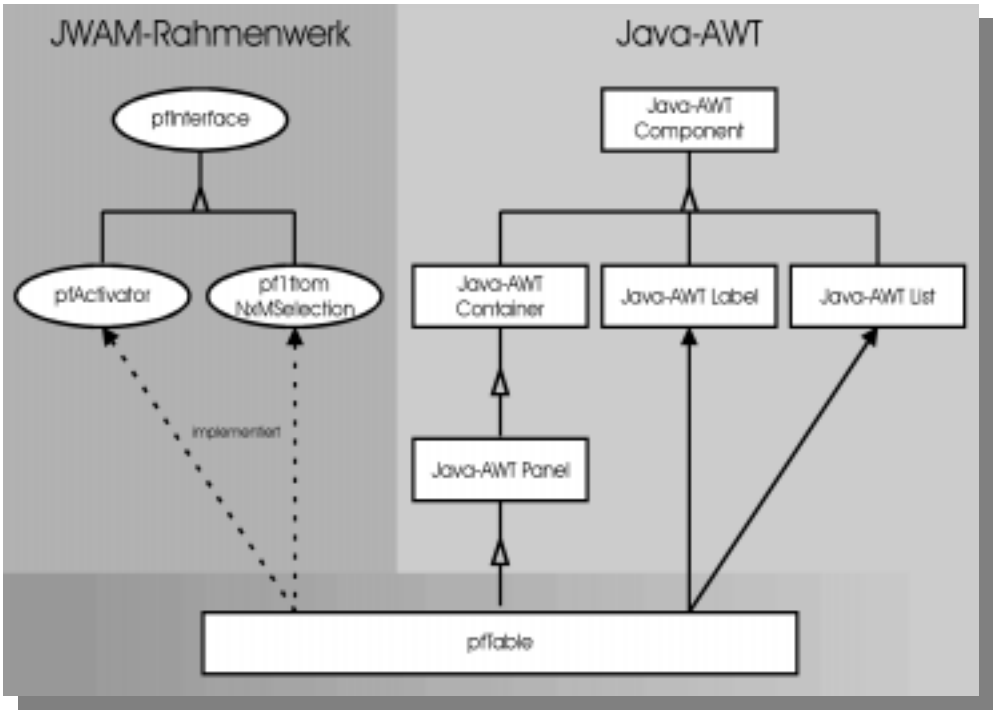


*Abbildung 10: Struktur der Präsentationsform **pfTable***

chenelemente der Java-AWT (siehe auch Abbildung 10). Unsere ***pfTable*** erbt von Java-AWT ***Panel*** und benutzt Java-Awt ***List*** Objekte für die Darstellung der Tabellen-Zellen. Um über die Interaktionsformen ***ifActivator*** und natürlich ***if1fromNxMSelection*** auf die Präsentationsform zugreifen zu können, implementiert sie die entsprechenden Interfaces. Außerdem muß auch die ***pfTable*** ein paar Java-Beans-Methoden implementieren, um in der Entwicklungsumgebung auf die Eigenschaften der Praesentationsform zugreifen zu können (wie Name der Präsentationsform, Anzahl der Spalten und Zeilen u.s.w.).

- ***pf1fromNxMSelection***

  Dies ist das Interface, welches von der ***pfTable*** implementiert wird, damit über die Interaktionsform ***if1fromNxMSelction*** auf die Präsentationsform zugegriffen werden kann.

  Der Umgang mit der Tabelle lehnt sich an dem mit einfachen Listen über die Interaktionsform ***if1fromNSelection*** an. Er unterscheidet sich im Wesentlichen dadurch, daß zu jeder Manipulation angegeben werden muß, auf welche Liste sie sich beziehen soll, was durch die Angabe der Zeile und Spalte erreicht wird.

Eine Gegenüberstellung der Schnittstellen zeigt Abbildung 11.

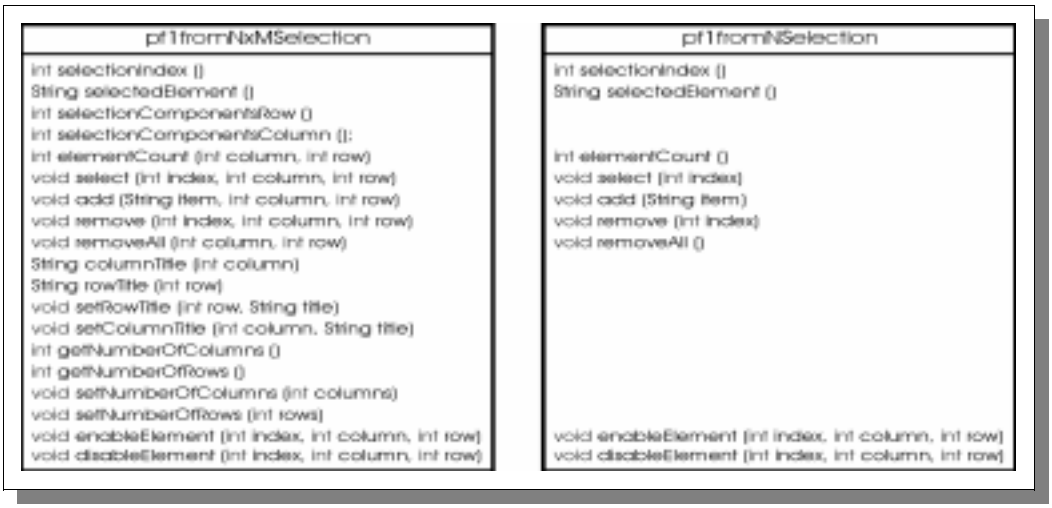


Abbildung 11: Gegenüberstellung der Schnittstellen von ***pf1formNxMSelection*** und **pf1fromNSelection**

- ***if1fromNxMSelection***

  Dies ist die Interaktionsform, welche die Auswahl eines Elements aus einer von n mal m Listen ermöglicht. Um zu veranschaulichen, wie die Kopplung zwischen Interaktionsform und Präsentationsform funktioniert zeigt Abbildung 12 Teile des Konstruktors der ***if1fromNxMSelection***.

  Dem Konstruktor wird der sogenannte ***IAFsContext***[37] und der Name der zu koppelnden Präsentationsform übergeben**.** Die Interaktionsform läßt sich dann von dem ***IAFsContext*** eine Menge mit den Präsentationsformen liefern, welche den übergebenen Namen tragen. Danach erzeugt sie sich ein ***pfSelectCommand*** Objekt, mit dem sie sich bei jeder gefundenen Präsentationsform einträgt.

37 Der ***IAFsContext*** enthält die Struktur, die Namen und die Typen der Oberflächenelemente und wird beim Start des Werkzeugs aus der serealisierten Oberfläche erzeugt.

```
[...]
public class if1fromNxMSelection extends ifObject
{

 public if1fromNxMSelection (IAFsContext context,
                             String      name)
  {
   [...]

   // Diese IAF an die entsprechenden PFs binden

    conSet pfs = context.pfs(new
       PFDescriptor(pf1fromNxMSelection.class,
       "getSelectionName"), name);
    setPFs(pfs);

    // Command zur Kommunikation mit den PFs erzeugen ...
    _pfSelectCommand = new cmdObject(this, "pfSelected");

    // ... und bei allen PFs anmelden
    Cursor cursor = Factory.newCursor(pfs);
    cursor.forth();
    while (cursor.isInside())
    {
      ((pf1fromNxMSelection)cursor.current()).
          attachSelectCommand (_pfSelectCommand);
      cursor.forth();
    }
    cursor.logout();
  };
  [...]

  public void pfSelected (cmdObject cmd)
  {
    [...]
  };
  [...]
```

*Abbildung 12: Source-Code **if1fromNxMSelection***

- ***cmdNxMSelect***

  Dies ist die Kommando-Klasse, zur losen Kopplung der Interaktionskomponente des Werkzeugs an die Interaktionsform. Der im Rahmenwerk vorhandenen Klasse ***cmdSelect*** fehlt für unsere Zwecke lediglich die Möglichkeit, die Spalten und Zeilen-Information zu übermitteln, weshalb wir diese Klasse beerbt und die nötigen Erweiterungen ergänzt haben. Einen Ausschnitt aus dem Source-Code zeigt Abbildung 13.

```
public class cmdNxMSelect extends cmdSelect
{
  [...]
  // Konstruiert Kommando-Objekt ohne Befehlsausfuehrer
  public cmdNxMSelect ()
  {
    super();
  };

  [...]
  // Setzt die Spalte in welcher das Element selektiert ist
  public void setSelectComponentsColumn(int column)
  {
    _column = column;
  };

  // Liefert die Spalte in welcher das Element selektiert ist
  public int selectComponentsColumn()
  {
    return _column;
  };

  // Setzt die Zeile in welcher das Element selektiert ist
  public void setSelectComponentsRow(int row)
  {
    _row = row;
  };

  // Liefert die Zeile in welcher das Element selektiert ist
  public int selectComponentsRow()
  {
    return _row;
```

Abbildung 13: Source-Code zu **cmdNxMSelect**

6.1.2 Erstellen der Tool-Klasse

Die ***tool***-Klasse hat weder etwas mit den Funktionen noch mit dem Aussehen der Werkzeuge zu tun. Sie dient nur dem Erzeugen der Funktionskomponente und der Interaktionskomponente sowie der Bindung der Werkzeuge an die entsprechenden Ressourcen (den serealisierten GUIs). Aus diesem Grund ist die Konstruktion der ***tool***-Klassen auch sehr einfach. Man beerbt die Klasse ***toolObject*** des Rahmenwerks und implementiert die beiden abstrakten Methoden ***createFP*** und ***createIP*** die dafür sorgen, eine Funktionskomponente und eine Interaktionskomponente des Werkzeugs zu erzeugen. Wie dies am Beispiel des Pausenlisten-Bearbeiter Werkzeugs aussieht, zeigt Abbildung 14.

```
public class toolPausenListenBearbeiter extends toolObject
{
  public toolPausenListenBearbeiter ()
  {
    super();
  };

  // Erzeuge die Funktionskomponente
  protected fpObject createFP ()
  {
    _fp = new fpPausenListenBearbeiter(requestController());
    return _fp;
  };

  // Erzeuge die Interaktionskomponente
  protected ipObject createIP (fpObject fp)
  {
    guiManagerInterface guiManager =
        guiManagerBeans.instance();
    IAFsContext iafsContext = guiManager.
        iafsContext(this.getClass(), "PausenListenBearbeiter");
    _ip = new ipPausenListenBearbeiter(iafsContext,
```

Abbildung 14: Source-Code ***toolPausenListenBearbeiter***

Was bei der Erzeugung der Funktionskomponente passiert, sollte unmittelbar klar sein, wird doch lediglich eine Instanz der entsprechenden Klasse generiert. Interessanter ist da schon was in der Methode ***createIP()*** geschieht. Zuerst wird der ***IAFsContext*** geladen. Dieses Objekt wird mit Hilfe des sog. ***guiMangerBeans*** erzeugt. Das Bean kehrt die zuvor in Kapitel 6.1.1.2 beschriebene Funktionsweise des Seraializer-Beans um und generiert aus der serialisierten Oberfläche den ***IAFsContext***. Über diesen ***IAFsContext*** werden später auch die Interaktionsformen an die Interaktionskomponente gekoppelt. Erst danach kann eine Instanz der Interaktionskomponente erzeugt werden, da hierzu wie schon erwähnt der ***IAFsContext*** benötigt wird.

6.1.3 Erstellen der Funktionskomponente

Für die Funktionskomponente des Werkzeugs existiert im JWAM-Rahmenwerk eine spezielle Oberklasse ***fpObject***, von der alle Funktionskomponenten erben. Besitzt das Werkzeug keine Sub-Werkzeuge, so sind keine weiteren Schritte not-

```
public class fpPausenListenBearbeiter extends fpObject
                                       implements EventObserver
{
  // Konstruktor: FK wird initialisiert.
  public fpPausenListenBearbeiter (RequestController successor)
  {
    super(successor);
    [...]
  };
  [...]

  // Sub-FK's erzeugen.
  public void doCreateSubFPs()
  {
    // Behälter mit den Sub-FKs holen
    conList subFPs = (conList)subFPsContainer();
    // Neu Sub-FK erzeugen
    _fpPausenBearbeiter = new
        fpPausenBearbeiter(requestController());
    // Erzeugte Sub-FK dem Behälter hinzufügen
    addSubFP(_fpPausenBearbeiter);
    // Beim Event der Sub-FK anmelden
    _fpPausenBearbeiter.pauseUebernehmenEvent().
        register(this, "reactOnPauseUebernehmen");
  };

  // Callback-Methode für das Event der Sub-FK
  public void reactOnPauseUebernehmen(evtObject event)
  {
```

*Abbildung 15: Source-Code aus **fpPausenListenBearbeiter***

wendig, und es braucht nur die Funktionalität des Werkzeugs implementiert zu werden. Ist dies aber, wie im Beispiel Pausenlisten-Bearbeiter Werkzeug, der Fall, so muß die Methode ***doCreateSubFPs()*** überschrieben werden. Wie dies aussehen kann, zeigt Abbildung 15. Zuerst wird der Container mit allen eventuell schon vorhandenen Sub-Funktionskomponenten geholt. Danach wird die eigentliche Sub-FK erzeugt und dann in den zuvor besorgten Container aufgenommen. Muß die Kontext-FK nicht über Zustandsänderungen ihrer Sub-FK informiert werden, so wären wir schon fertig. Im Beispiel des Pausenlisten-Bearbeiter Werkzeugs muß die Kontext-FK aber wissen, wenn sich die Pause im Pausen-Bearbeiter Werkzeug verändert hat, weshalb sie sich zum Schluß noch bei dem entsprechenden Event der Sub-FK anmeldet (dies ist auch der Grund dafür, weshalb die Klasse ***fpPausenListen-***

Bearbeiter noch das Interface ***EventObserver*** implementiert).

6.1.4 Erstellen der Interaktionskomponente

Das Programmieren der Interaktionskomponente verläuft weitgehend symmetrisch zu dem eben Beschriebenen der Funktionskomponente. Auch für die IAK hält das JWAM-Rahmenwerk eine entsprechende Oberklasse ***ipObject*** bereit, von der alle IAKs erben. Und genau wie bei der Funktionskomponete muß auch in der IAK-Klasse nur eine Methode überschrieben werden, wenn das Werkzeug Sub-Werkzeuge besitzt. Bei der IAK heißt die zu überschreibende Methode allerdings ***doNewSubIP()***. Wie diese Methode aussehen kann, zeigt Abbildung 16.

```
protected ipObject doNewSubIP (fpObject fp)
{
  // Zum Typ der Sub-FK passende Sub-IAK ermitteln
  if (fp instanceof fpPausenBearbeiter)
  {
    // Instanz des guiManagerBeans holen
    guiManagerInterface guiManager =
        guiManagerBeans.instance();
    // IAFsContext des Subwerkzeugs besorgen
    IAFsContext subIafsContext =
        guiManager.iafsContext(this.getClass(),
        "PausenBearbeiter", this.iafsContext());
    // Sub-IAK mit deren FP und ihrem IAFsContext erzeugen
    _ipPausenBearbeiter = new
        ipPausenBearbeiter(subIafsContext,
        (fpPausenBearbeiter) fp);
    // Sub-IAK anzeigen
    _ipPausenBearbeiter.show();
    // Sub-IAK zurückgeben
    return _ipPausenBearbeiter;
  }
  // Keine weiteren Sub-Werkzeugtypen vorgesehen
  else
  {
    return null;
  }
```

Abbildung 16: Source-Code Methode doNewSubIP()

Wenn die FK des Werkzeugs eine Sub-FK erzeugt, sorgt die Oberklasse ***ipObject*** dafür, daß automatisch die Methode ***doNewSubIP()*** der zum Werkzeug gehörenden IAK gerufen wird. Dabei wird die erzeugte Sub-FK mit übergeben, wodurch

die IAK erkennen kann, welches Sub-Werkzeug generiert werden soll. Auf diese Weise können beliebig viele Sub-Werkzeuge gestartet werden. Leider ist es aber nicht möglich, zwei Sub-Werkzeuge vom selben Typ in ein Kontext-Werkzeug zu integrieren, da die Sub-Werkzeuge momentan noch über ihren Typ unterschieden werden und es so unmöglich wäre, die beiden Sub-Werkzeuge korrekt zu differenzieren.

Die IAK ermittelt nun den Typ der Sub-FK und besorgt sich mit Hilfe des ***guiManagerBeans*** den ***IAFsContext*** des entsprechenden Sub-Werkzeugs. Danach kann die Sub-IAK mit ihrer FK und ihrem ***IAFsContext*** generiert werden. Zum Abschluß wird die IAK angezeigt und an die aufrufende Methode von ***ipObject*** zurückgegeben.

Ein interessanter Aspekt beim Programmieren der Interaktionskomponente ist auch, wie man die verschiedenen Oberflächenelemente der GUI in der IAK ansprechen kann. Um dies zu veranschaulichen, zeigt Abbildung 17 den Source-Code um das Oberflächenelement, welches die Liste der Pausen enthält, an die IAK zu binden.

Zuerst wird die gewünschte Interaktionsform generiert. Dem Konstruktor wird der ***IAFsContext*** und der Name der zu bindenden Präsentationsform übergeben. Die Interaktionsform sucht sich dann selbst in dem Kontext die Präsentationsform mit dem passenden Namen heraus und stellt die Verbindung her. Dies reicht aus, um auf die Attribute der entsprechenden Präsentationsform (bzw. des Oberflächenelementes), hier die Liste der Pausen, über die Interaktionsform zugreifen zu können. Möchte man, wie in unserem Beispiel, bestimmte Events der Präsentationsform verarbeiten, so erzeugt man ein entsprechendes Command-Objekt und hängt dies an die Interaktionsform an. Der Command-Mechanismus des JWAM-Rahmenwerks sorgt dann dafür, das die Methode, welche bei der Erzeugung des Commands durch die Übergabe des Methodennamens spezifiziert wurde, in unserem Beispiel die Method ***pausenListeSelectet(),*** mit dem Commando-Objekt gerufen wird. Diese Schritte sind für alle Präsentationsformen, welche verwendet werden sollen, durchzuführen.

```
public class ipPausenListenBearbeiter extends ipObject
{
  public ipPausenListenBearbeiter (IAFsContext iafsContext,
                                   fpPausenListenBearbeiter fp)
  {
    [...]

    // Listenfeld: Pausenliste-Liste
    // Interaktionsform 'if1fromNSelektion' erzeugen ...
    _pausenListeSelection  = new if1fromNSelection
        (context(), "selPausenlistenBearbeiterListPausen");
    // ... Commando 'cmdSelect' erzeugen und ...
    _pausenListeCommand   = new
        cmdSelect(this, "pausenListeSelected");
    // ... an die erzeugte Interaktionsform übergeben
    // und koppeln
    _pausenListeSelection.
        attachSelectCommand(_pausenListeCommand);

    [...]

   // Callback-Methode für das Commando 'cmdSelect'
   public void pausenListeSelected(cmdSelect cmd)
   {
     Identificator id;

     // ID des in der Liste selektierten Eintrages holen
     id = _pausenListeSelection.selectionID();
     // Die FK mit der ID rufen
     _fp.pausenAuswahl(id);
   };
```

Abbildung 17: Source-Code ***ipPausenListenBearbeiter***

6.1.5 Der Informationsfluß zwischen verschiedenen Werkzeugen

Mit Hilfe von Abbildung 18 möchten wir veranschaulichen, wie die Werkzeuge Kenntnis über Veränderungen an den von ihnen bearbeiteten Materialien erlangen. Dabei spielt der in den Abschnitten 6.2 sowie 6.4.1 ausführlich vorgestellte ***ppMaterialVersorger*** eine zentrale Rolle. Dieser Automat existiert als Singleton[38] in jedem Prozeßraum genau ein Mal und bildet das Bindeglied zwischen Werkzeug und Archivar.

Wurde ein Material in das Archiv zurückgestellt, so sendet der ***Archivar*** eine entsprechende Nachricht an den globalen ***MessageBroker***. Dieser leitet die Nachricht

38 Gamma et al Seite S. 139-148

an alle bei ihm registrierten ***ppMaterialVersorger*** in den unterschiedlichen Prozeßräumen weiter. Jeder der verschiedenen ***ppMaterialVersorger*** prüft nun, ob das Material, dessen Eingang der Archivar gemeldet hat, von der Kategorie *Pausenplaner*-Material ist. Ist dies nicht der Fall, so kann er die Nachricht ignorieren. Anderenfalls sendet er eine entsprechende Nachricht an den lokalen ***MessageBroker***, welcher sie an alle registrierten Funktionskomponenten der Werkzeug weiterleitet. Letztlich können die Funktionskomponenten prüfen, ob sie das veränderte Material gerade bearbeiten und geeignete Aktionen auslösen. Im Normalfall würde sich die Funktionskomponente die neue Version des Materials beim ***ppMaterialVersorger*** holen. Danach könnte sie das Material auf Inkonsistenzen prüfen und ihre Interaktionskomponente durch ein Event veranlassen, deren Anzeige zu aktualisieren.

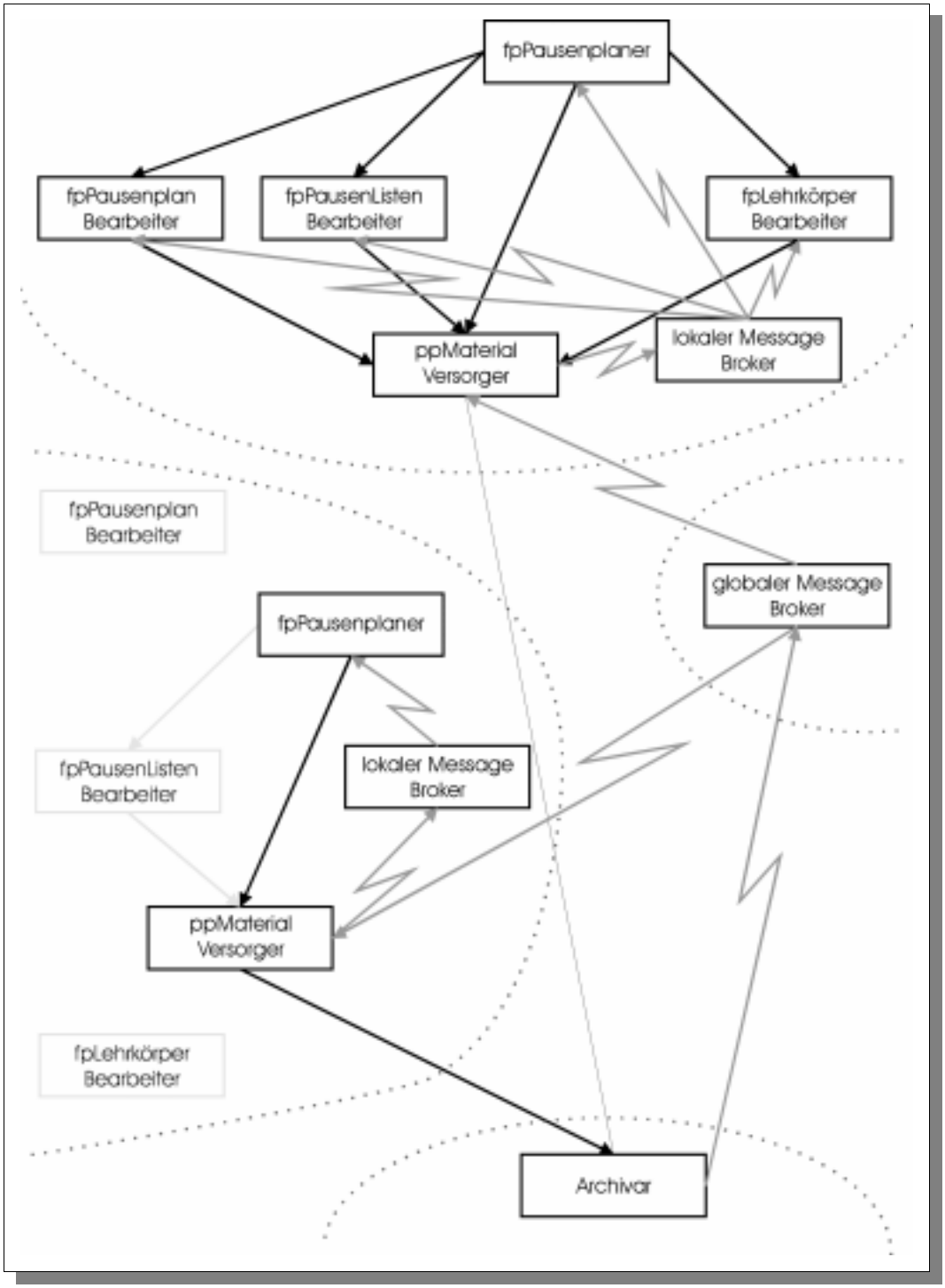
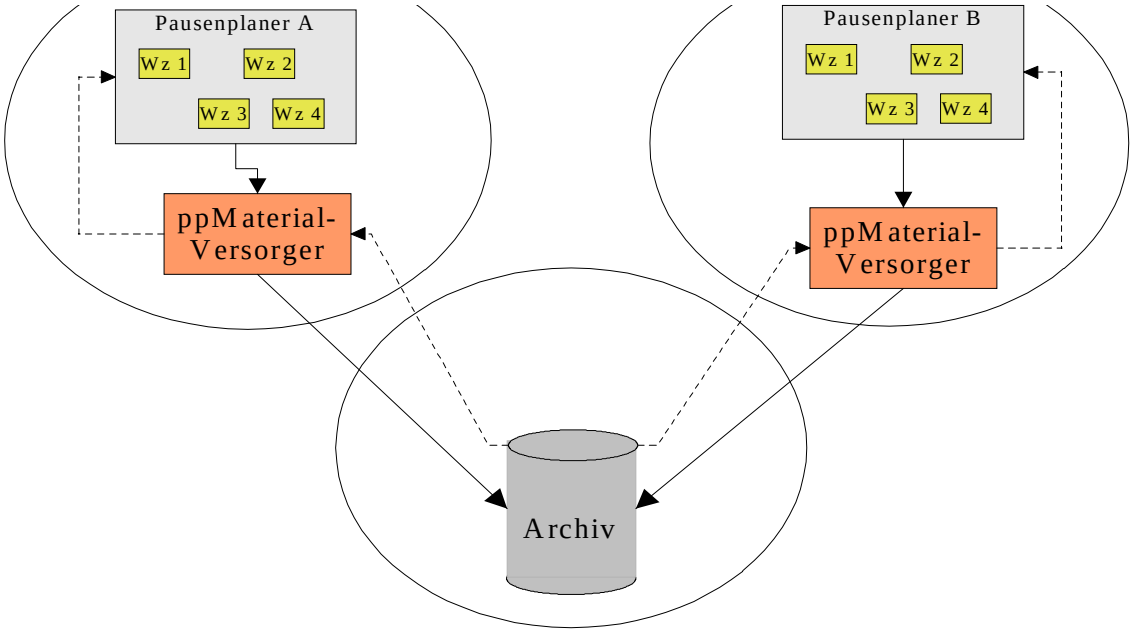


Abbildung 18: Zusammenspiel des Pausenplaner-Systems

6.2 Die Materialverwaltung

Die Werkzeuge des Pausenplaners fordern ihre Materialien beim ***ppMaterialVersorger*** an, bzw. übergeben diesem die bearbeiteten Materialien zur Aufbewahrung.



Der ***ppMaterialVersorger*** bildet damit eine anwendungsspezifische Abkapselung der Archivschnittstelle von den Werzeugen des Pausenplaners. Der ***ppMaterialVersorger*** registriert sich beim ***MessageBroker*** für die vom ***Archivar*** versandten Nachrichten und filtert aus diesen eingehenden Nachrichtigen diejenigen heraus, die für die in seinem Prozeßraum registrierten Werkzeuge von Interesse sein könnten. Diese Nachrichten werden daraufhin von dem ppMaterialVersorger über einen lokalen MessageBroker an diese Werkzeuge versandt.

Die von uns gewählte Materialverwaltung unterscheidet sich demnach in mehreren Punkten von der bei Züllighoven[39] beschriebenen Lösung. Während der Nachrichtenaustausch auf der dort beschriebenen Art und Weise geschieht, haben wir uns für eine etwas andere Aufteilung der Funktionalität entschieden. Unser ***ppMaterialVersorger*** entspricht weitestgehend dem dort eingeführten *Materialverwalter*, übernimmt allerdings auch die Aufgaben des *Materialmagazins.* Zusätzlich sorgt der ***ppMaterialversorger*** für das Ver- und Entpacken der von dem Pausenplanerwerkzeugen benutzten Materialien in die für den Austausch mit dem Archiv benötigten ***MaterialWrapper***. Die übrigen Dienste des *Materialversorgers*

39 Züllighoven, Konstruktionshandbuch, S. 299-305.

werden von den einzelnen Komponenten unseres Archives erbracht.

6.3 Implementation der Materialien

Bei der Implementation der Materialien haben wir uns zunächst nur an den rein fachlichen Vorgaben orientiert. Diese Vorgehensweise wurde auch mit Blick auf unser für jedes beliebige Material verwendbare Archiv beibehalten. In einer ersten Version unseres Pausenplaner-Systems wurden dementsprechend die verwendeten komplexen Materialien jeweils als Ganzes in das Archiv gestellt. Im weiteren Verlauf unserer Arbeit stellten wir jedoch fest, daß dieser einfache Mechanismus an anderer Stelle mehrere Probleme aufwarf. Eines dieser Probleme war die Gewährung der Konsistenz der bearbeiteten Materialien. Die beiden grundlegenden Materialien des Pausenplaners sind wie schon im Kapitel 1.1 beschrieben zum einen der Lehrkörper und zum anderen der Pausenplan. Beide komplexen Materialien enthalten aber mit den Lehrern gleiche »Untermaterialien«. Wenn mit Hilfe des Lehrkörper-Bearbeiters ein Lehrer verändert wird, der auch im Pausenplan enthalten ist, so muß diese Änderung auch in diesem Material übernommen werden. In unserer ersten Lösung wurde allerdings der aus fachlicher Sicht gleiche Lehrer einmal als Teil des Lehrkörper-Materials und einmal als Teil des Pausenplaner-Materials im Archiv abgespeichert. Aus diesem Grund hätten wir irgendeinen Mechanismus implementieren müssen, der die Konsistenz der beiden Versionen des gleichen Lehrers gewährleistet. Wir haben uns jedoch dafür entschieden, nachträglich eine vollkommen andere Lösung des Konsistenzproblems zu implementieren. Unser zweiter Ansatz folgt dem im Kapitel 5.3.2.4. skizzierten Vorschlag, ein komplexes Material vor der Übergabe an das Archiv in seine einzelnen Bestandteile zu zerlegen, und diese Einzelmaterialien jeweils für sich zu speichern. Neben den Erhaltung der Materialkonsistenz sollte dieser Ansatz auch zusätzlich für eine Verringerung des Netzwerkverkehrs sorgen, sowie einen eleganteren technischen Umgang mit komplexen Materialien ermöglichen.

6.3.1 Die Basisklassen der Materialien

Der Konstruktion der vom Pausenplaner verwandten Materialien liegt die Annahme zu Grunde, daß alle diese Materialien eine »Baumstruktur« aufweisen. Die Menge der im Pausenplaner bearbeiteten Materialien ist abgeschlossen bezüglich der Verknüpfung, d.h. daß alle Materialien, die selbst einen »Baum« bilden, als »Unterbaum« einem anderen Material hinzugefuegt werden können, ohne daß das erweiterte Material dadurch seine Baumstruktur verliert. Von dieser Vorausetzung ausgehend, haben wir nach dem Kompositum-Muster[40] eine Struktur von abstrakten Basisklassen aufgebaut, die die für den Umgang mit Baumstrukturen benötigte Funktionalität bereitstellt.

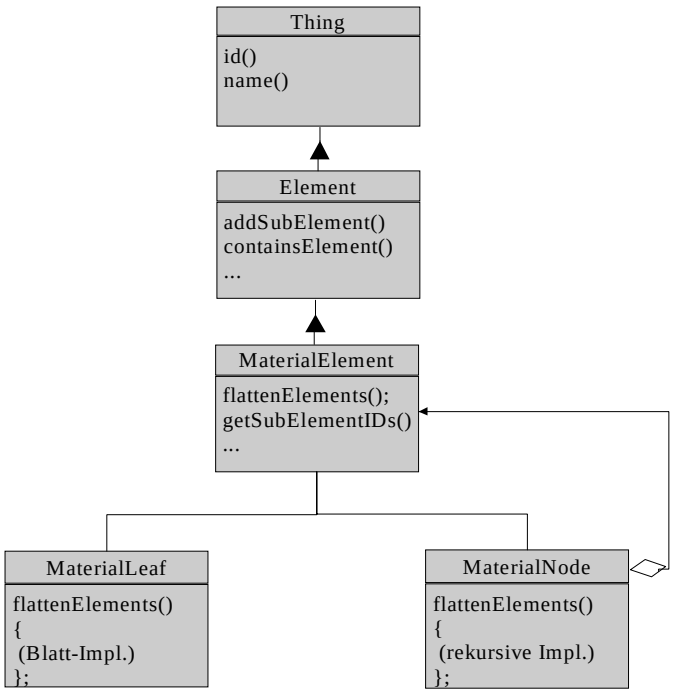


Abbildung 19: Architektur der Basisklassen der Materialien

Die abstrakte Klasse ***Element*** deklariert die für den Umgang mit Baumstrukturen benötigten Methoden und folgt damit der ***Komponente***-Klasse des Kompositum-Musters. ***Element*** erbt dabei von der JWAM-Klasse ***Thing*** die zur netzwerkweit eindeutigen Identifizierung notwendige ID. Diese ID wird beim Aufruf des Kon-

40 Gamma et al, S. 213-225

struktors von ***Thing*** für das erzeugte Objekt generiert und kann über die Methode ***id()*** abgefragt werden. In dieser Implementierung des Kompositum-Musters haben wir dem Ziel der Maximierung der Komponentenschnittstelle den Vorzug vor anderen Prinzipien des Klassenentwurfes gegeben[41], die eine Beschränkung der in der Oberklasse definierten Methoden auf die für alle Unterklassen sinnvollen fordern. So werden in ***Element*** schon alle Methoden deklariert und zum großen Teil auch schon implementiert, die zur Verwaltung der Subelemente (bei Gamma et al als »Kindobjekte« bezeichnet) dienen.

Die Klasse ***MaterialElement*** erbt von ***Element*** und erweitert deren Schnittstelle um die materialspezifischen Methoden. Dies sind vor allem die für das Zerlegen und wieder Zusammensetzen komplexer Materialien benötigten Methoden sowie die Ergänzung des Konzeptes der Subelemente um die Verwaltung der IDs dieser Subelemente.

Von ***MaterialElement*** erben schließlich die Klassen ***MaterialLeaf*** und ***MaterialNode***, die den Klassen ***Blatt*** bzw. ***Kompositum*** bei Gamma et al entsprechen. In der Klasse ***MaterialNode*** werden die meisten der in ***Element*** und ***MaterialElement*** vorgenommenen Default-Implementierungen überschrieben. Dazu gehören in erster Linie die Methoden, die von einem ***MaterialNode***-Objekt rekursiv auf seine Subelemente angewandt werden sowie Methoden, die Attribute erfragen, die nur der Knoten eines Baumes besitzen kann, beispielsweise die Liste der Subelemente.

Mit Blick auf die geplante Verwendung der komplexen Materialien in einer verteilten Umgebung werden von einem ***MaterialNode***-Objekt außer den Subelementen selbst noch zusätzlich in einem anderen Behälter die IDs dieser Subelemente aufbewart. Von besonderer Bedeutung sind diese IDs der Subelemente im Umgang mit dem Archiv. Da das Archiv wie in Kapitel 5.3.1.4. beschrieben, keinerlei Wissen über den Typ oder die Struktur der von ihm verwalteten Materialien besitzt, müssen wir komplexe Materialien vor der Übergabe an das Archiv in die einzelnen Bestandteile zerlegen, bzw. nach dem Erhalt aus dem Archiv wieder zusammensetzen.

Beim Zerlegen eines MaterialElement-Objektes wird die Methode

41 ebenda, S. 217-218

conList flattenElements(conList elementList) dieses Objektes aufgerufen. Handelt es sich bei dem Objekt um ein ***MaterialLeaf***-Objekt, setzt dieses seine Referenz auf sein Elternobjekt auf ***null*** und fügt sich selbst der als Parameter übergebenen Liste hinzu, die schließlich zurückgegeben wird. Handelt es sich dagegen um ein ***MaterialNode***-Objekt, ruft dieses zunächst an jedem seiner Subelemente die Methode ***flattenElements()*** auf. Nach dem dies geschehen ist, leert das ***MaterialNode***-Objekt seinen Subelement-Behälter, setzt die Referenz auf sein Elternobjekt auf ***null,*** fügt sich in die als Parameter übergebene Liste ***elementList*** ein und gibt diese zurück. Nachdem die Methode ***flattenElements()*** beispielsweise an einem ***MaterialNode***-Objekt »ExampleNode« aufgerufen worden ist, hat das aufrufende Objekt eine Liste mit ***MaterialElement***-Objekten zurückerhalten, in der sowohl »ExampleNode« selbst, als auch alle Elemente des Unterbaumes, dessen Wurzel »ExampleNode« bildet, als einzelne und voneinander unabhängige Elemente enthält. Alle in dieser Liste enthaltenen ***MaterialNode***-Objekte enthalten nur noch die IDs ihrer Subelemente.

```
public conList flattenElements(conlist elementList)
{
  MaterialElement TempElement;
  Cursor cur = new Cursor();
  cur.login(_subElements);
// 1. Alle Subelemente einsammeln
  while (cur.forth())
  {
// Jedes Subelement fuegt sich in die Liste ein
   TempElement = (MaterialElement)cur.current();
   TempElement.flattenElements(elementList);
  };
// 2. Verbindungen zu den Subelementen kappen
  _subElements.makeEmpty();
```

```
// 3. Anzeigen, dass ich keine direkte Verbindung mehr zu meinen
//    Subelementen habe
  _flatFlag = true;
// 4. Verbindung zum Elternobjekt kappen
  setSuperNode(null);
// 5. "Mich selber einpacken"
  elementList.add(this);
// 6. Liste zurueckgeben
  return elementList;
};
```

Mit Hilfe der IDs der Subelemente kann aus einem Wurzelelement jederzeit wieder die gesamte Baumstruktur rekonstruiert werden.

Die vorangegangenen Erläuterungen lassen sich also wie folgt zusammenfassen: Als Grundlage der Materialkonstruktion diente uns die Annahme, daß sich alle vom Pausenplaner verwandten komplexen Materialien auf eine hierarchische Struktur abbilden lassen. Daraufhin haben wir auf Basis des Kompositum-Musters ein Grundgerüst aus Klassen aufgebaut, die den Umgang mit hierarchischen Strukturen ermöglichen. Als letzen Schritt mußten wir bei jedem vom Pausenplaner benutzten Material entscheiden, ob es sich um einen Knoten oder ein Blatt der Baumstruktur handelt. Nach dieser Einordnung brauchten wir die konkreten Materialklassen nur noch als Unterklassen von ***MaterialNode*** beziehungsweise ***MaterialLeaf*** zu implementieren. Diese methodische Vorgehensweise haben wir in der Praxis allerdings nicht befolgt. Im Rahmen der zyklischen Softwareentwicklung hatten wir zunächst den Schwerpunkt auf die Konstruktion der Werkzeuge gelegt und daher die Materialien als von rein fachlichen Gesichtspunkten geleitete Prototypen implementiert. Erst als wir uns näher mit den Problemen der Materialkonsistenz befaßten, haben wir die Beziehungen der Materialien genauer untersucht und schließlich die in diesem Kapitel vorgestellte Architektur entworfen und implementiert. Die im letzen Schritt vorgenommene Anpassung der schon vorhandenen Materialklassen an die erst im Nachhinein erstellte Basisstruktur erwies sich dabei allerdings als sehr zügig zu erledigende Aufgabe, die keinerlei Schwierigkeiten bereitete.

6.3.2 Das Material Lehrkörper

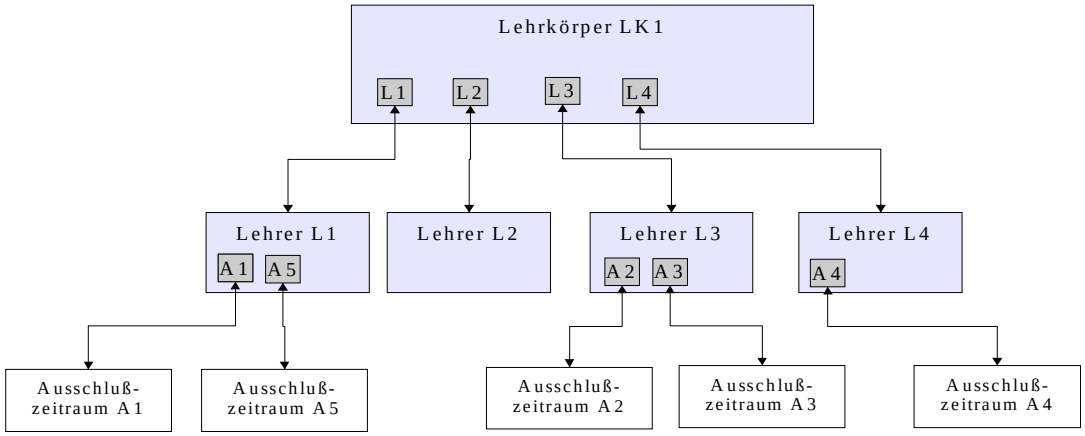


Abbildung 20: Die Struktur des Materials Lehrkoerper (schematische Darstellung)

Das Material Lehrkörper stellt einen Knoten in einer Baumstruktur dar, da der Lehrkoerper mehrere Subelemente enthalten kann. Aus diesem Grund haben wir die Klasse ***Lehrkoerper*** als Unterklasse von ***MaterialNode*** implementiert. Einem ***Lehrkoerper*** kann man Subelemente vom Typ ***Lehrer*** hinzufügen, wieder entfernen oder durch einen anderen ***Lehrer*** ersetzen. Des Weiteren kann man prüfen, ob ein ***Lehrer***-Objekt Mitglied dieses ***Lehrkoerper***-Objektes ist, sowie sich ein Lehrer-Objekt mit einer bestimmten ID herausgeben lassen.

Die Objekte der Klasse ***Lehrer***, die man dem ***Lehrkoerper*** als Subelemente hinzufügen kann, können wiederum Subelemente vom Typ ***Zeitraum*** beinhalten, die die Ausschlußzeiträume des Lehrers angeben. Daher bildet auch ***Lehrer*** eine Unterklasse von ***MaterialNode.*** Demgegenüber kann ein ***Zeitraum*** keine Subelemente enthalten und erbt daher von **MaterialLeaf** . Insgesamt kann also ein Objekt vom Typ ***Lehrkoerper*** eine beliebige Anzahl von ***Lehrer***-Objekten enthalten, und wenn es mindestens ein ***Lehrer***-Objekt enthält, Null oder mehrere Objekte vom Typ

Zeitraum beinhalten.

6.3.3 Das Material Pausenliste

Das Material Pausenliste kann ebenso wie das Material ***Lehrkörper*** mehrere Subelemente enthalten. Aus diesem Grund haben wir auch die Klasse ***Pausenliste*** als Unterklasse von ***MaterialNode*** implementiert. Ein Objekt vom Typ ***Pausenliste*** kann als Subelemente eine Anzahl von ***Pausen***-Objekten beinhalten. Diese wiederum enthalten ein Objekt vom Typ ***Zeitraum***, das den Termin der Pause beschreibt, sowie eine Liste von ***Aufsichtsort***-Objekten. Aus diesem Grund erbt auch die Klasse Pause von ***MaterialNode***.

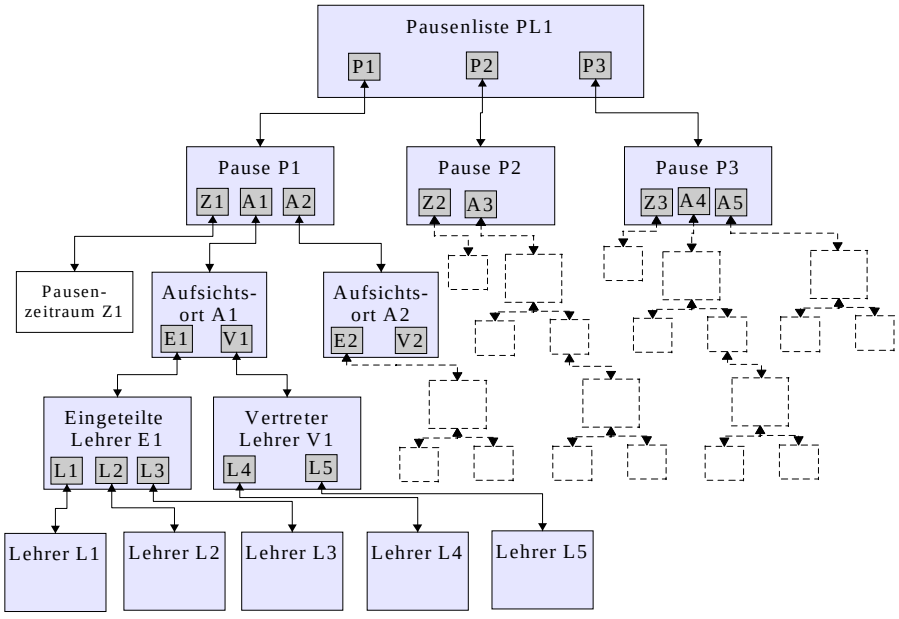


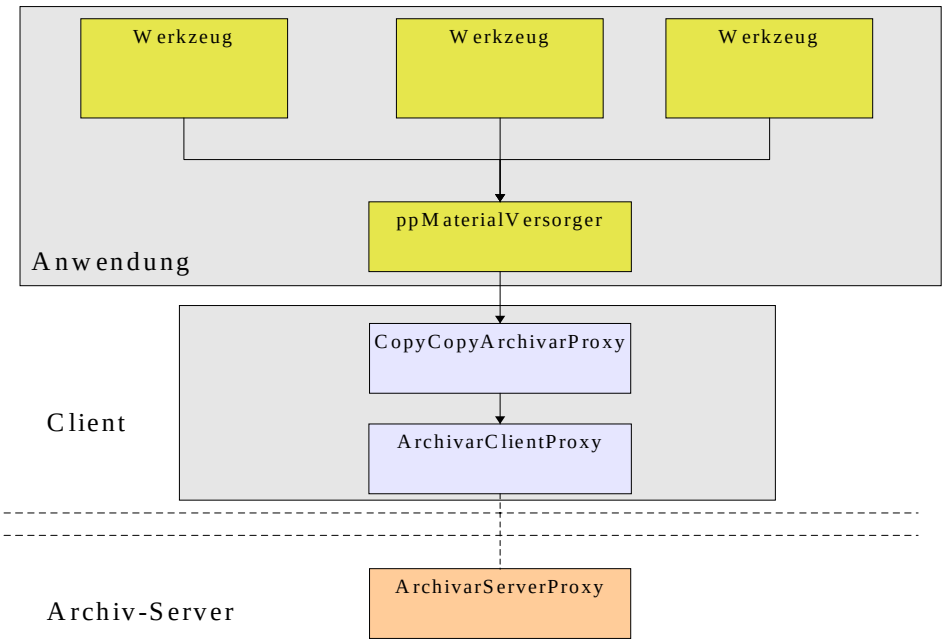
Abbildung 21: Die Struktur des Materials Pausenliste (schematische Darstellung)

6.4 Implementation der Automaten

6.4.1 Implementation des ppMaterialVersorgers

Der ***ppMaterialVersorger*** hat die fachliche Aufgabe, die Werkzeuge des Pausen-

planers mit den Materialien aus dem Archiv zu versorgen, bzw. die bearbeiteten Materialien wieder in das Archiv zurückzustellen. Aus technischer Sicht dient der ***ppMaterialVersorger*** zusätzlich dazu, die relativ komplexe Schnittstelle des Archives von dem Pausenplaner abzukapseln.



Die einzelnen Werkzeuge greifen über den als Singleton implementierten ***ppMaterialVersorger*** auf den Materialbestand des Archives zu. In der derzeitigen Version des ***ppMaterialVersorgers*** ist der Zugriff auf die Materialien des Archives auf Kopien beschränkt, d.h. der ***ppMaterialVersorger*** benutzt eine Instanz des ***CopyCopyArchivarProxys***. Um ein bestimmtes Material zu erhalten, ruft das betreffende Werkzeug die Methode

materialElement besorgeMaterial(Identificator materialID, IDUser toolID) an einer Instanz von ***ppMaterialVersorger*** auf. Handelt es sich bei dem gewünschten Material um ein Objekt vom Typ ***MaterialNode***, besorgt der ppMaterialVersorger rekursiv alle in der Baumstruktur vorkommenden Elemente. Aus diesem Grund brauchen ***fpLehrkoerperBearbeiter*** und ***fpPausenplanBearbeiter*** lediglich die ID desPausenplans bzw. des Lehrkörpers zu übergeben und erhalten jeweils das gesamte komplexe Material. Beim Zurückstellen dieser komplexen Materialien

müssen die Werkzeuge die Methode ***void speicherMaterial(MaterialElement material, IDUser toolID)*** des ***ppMaterialVersorgers*** aufrufen, wobei als Parameter ***material*** das komplexe Material übergeben wird. Der ***ppMaterialVersorger*** übernimmt dabei das Auseinandernehmen der Materialien mit Hilfe des in Kapitel 6.3.1. beschriebenen Algorithmus und legt die einzelnen Materialien unter den aus ihren Klassennahmen gebildeten Kategorien ab. So wird beispielsweise ein ***Lehrer*** -Objekt unter der Kategorie *pausenplaner.material.Lehrer* in das Archiv gestellt, während die in diesem ***Lehrer***-Objekt enthaltenen ***Zeitraum***-Objekte unter *pausenplaner.material.Zeitraum* abgelegt werden. Die nach den Klassennamen der Materialien benannten Kategorien erleichtern den Werkzeugen wiederum das Auffinden der benötigten Materialien, da sie die Methode ***conList besorgeInhaltsListe(String kategorie)*** des ***ppMaterialVersorgers*** mit dem jeweiligen Klassennamen des gesuchten Materials als Parameter ***kategorie*** aufrufen können.

Eine der Hauptaufgaben des ***ppMaterialVersorgers*** ist zudem die Abkapselung der für den Materialaustausch mit dem Archiv benötigten ***MaterialWrapper***-Objekte. Die Werkzeuge des Pausenplaners übergeben und erhalten vom ppMaterialVersorger nur Objekte vom Typ ***MaterialElement***. Der ***ppMaterialVersorger*** übernimmt die Aufgabe, diese Objekte in ***MaterialWrapper***-Objekten »einzupakken«, bzw. die vom Archiv erhaltenen ***MaterialWrapper*** »auszupacken«. Der Umgang mit den ***MaterialWrapper***-Objekten hat sich im nachhinein als recht unhandlich und fehlerträchtig erwiesen. Einerseits muß dafür Sorge getragen werden, daß ein schon einmal verpacktes Material wieder in dem gleichen Umschlag verpackt wird, da die Identifizierung der Materialien innerhalb des Archives nur durch die im ***MaterialWrapper*** enthaltene ***IDMaterial*** erfolgt (siehe Kapitel 5.3.1.4). Zum Anderen müssen die von ***Thing*** geerbten in den ***MaterialElement*** -Objekten vorhandenen IDs vom Typ ***Identificator*** auf die von den ***MaterialWrappern*** verwandten ***IDMaterials*** abgebildet werden.

6.5 Die Unterstützung kooperativer Arbeit mit gemeinsam genutzten Materialien durch den Pausenplaner

An einem kleinen Beispiel soll nun abschließend erläutert werden, wie die in diesem Kapitel beschriebenen Werkzeuge, Automaten und Materialien im Zusammenspiel mit dem Archiv auch die kooperative Arbeit an gemeinsamen Materialien unterstützen können. Als Fallbeispiel dient das Hinzufügen eines neuen Ausschlußzeitraumes zu einem Lehrer des Lehrkörpers. Um diesem Lehrer einen Ausschlußzeitraum hinzuzufügen, wird zunächst das Werkzeug *LehrkoerperBearbeiter* gestartet. Die FK dieses Werkzeuges meldet sich daraufhin beim ***ppMaterialVersorger*** an und fordert bei diesem das entsprechende Material ***Lehrkoerper*** an. Danach wählt der Benutzer in der von der ***ipLehrkoerperBearbeiter*** angezeigten Liste der in diesem Lehrkörper enthaltenen Lehrer den gewünschten aus, und kann dessen Ausschlußzeiträume mit dem *AusschlußzeitraumBearbeiter-Werkzeug* manipulieren. Die Änderung an diesem ***Lehrer***-Objekt wird auch von der ***fpLehrkoerperBearbeiter*** registriert, der daraufhin dem ***ppMaterialVersorger*** den kompletten ***Lehrkoerper*** zum Speichern im Archiv übergibt. Der ***ppMaterialVersorger*** zerlegt das komplexe Material ***Lehrkoerper*** in seine Einzelmaterialien und überprüft an jedem, ob ein Flag gesetzt ist, das eine Veränderung signalisiert.

Dieses Flag ist sowohl an dem neuen ***Zeitraum***-Objekt, als auch an dem entsprechenden ***Lehrer***-Objekt gesetzt. Als nächstes ruft der ***ppMaterialVersorger,*** um eine *begrenzte Transaktion*[42] zu beginnen, die Methode ***startLimitedTransaction()*** des von ihm benutzten ***CopyCopyArchivarProxys*** auf, wobei die ***IDUser*** des ***fpLehrkoerperBearbeiters*** und die Liste mit den IDs der veränderten Materialien als Parameter übergeben werden. Nachdem sich der ***ppMaterialVersorger*** somit den exklusiven Zugriff auf diese Materialien gesichert hat, wird das im Archiv vorhandene Original der Materialien durch die neue Version ersetzt, bzw. als neues Original in das Archiv aufgenommen. Im Falle der neuen Version des ***Lehrer*** -Objektes sendet das Archiv eine Nachricht vom Typ ***msgArchivarOriginalUpdated*** aus. Diese Nachricht wird wiederum von allen beim ***MessageBroker*** registrierten ***ppMaterialVersorgern*** aufgefangen.

42 siehe Kapitel 5.3.1.2.

In unserem Beispiel wird gerade an einem anderen Arbeitsplatz ein neuer Pausenplan aufgestellt. Der dortige ***ppMaterialVersorger*** überprüft anhand der in ***msgArchivarOriginalUpdated*** enthaltenen Kategorie des aktualisierten Materials, ob es sich um ein von dem Pausenplaner benutztes Material handelt. Da dieses in unserem Fall zutrifft, wird eine neue Nachricht generiert und über einen lokalen ***MessageBroker*** abgesandt. Die FK des *PausenplanBearbeiter-Werkzeuges* hat sich für diesen Nachrichtentyp bei ihrem lokalen ***MessageBroker*** mit der Methode ***neueMaterialVersion***() registriert. Innerhalb dieser Methode wird überprüft, ob das Material, dessen ID in dem Nachrichten-Objekt enthalten ist, ein Bestandteil des bearbeiteten ***Pausenplan***-Materials ist. Wenn dem so ist, fordert die ***fpPausenplanBearbeiter*** von seinem ppMaterialVersorger die aktuelle Version dieses Materials aus dem Archiv an. Sobald diese neue Version eingetroffen ist, überprüft die ***fpPausenplanBearbeiter*** die Konsistenz des veränderten ***Pausenplans*** und zeigt eventuell entstandene Konflikte an.

7 Fazit

Die größten Schwierigkeiten bei der Entwicklung einer größeren Beispielanwendung für das JWAM-Rahmenwerkes hatten ihre Ursache nicht in diesem Rahmenwerk, sondern in der noch unzureichenden Unterstützung des Debuggens durch die vorhandenen Entwicklungsumgebungen. Diese stellten zumeist nur eine grafische Benutzeroberfläche für den im JDK enthaltenen Debugger »jdb« zur Verfügung, der jedoch oft schon bei dem Debuggen nichtverteilter Programme abstürzte, spätestens jedoch von der Kombination *RMI + core-reflection* endgültig überfordert wurde. Aufgrund dieser Tatsache wurde das Auffinden und Beseitigen von Fehlern häufig zu einer langwierigen und mühsamen Aufgabe. Der mangelhafte Debugger war nicht der einzige Umstand, welcher unsere Gelduld des öfteren auf eine harte Probe stellte. So erwiesen sich sämtliche für Java erhältliche Entwicklungsumgebungen in mindestens einem Punkt als nicht oder nur bedingt brauchbar. Entweder unterstützten die Entwicklungsumgebungen die neueste JDK-Version nicht, der GUI-Editor war unbrauchbar, es gab keine Schnittstelle zum Debugger, es exi-

stierte kein Projekt-Management, oder aber sie stürzten gleich ab. Eines der größten Übel war jedoch die oft fehlende Performance. Zum Ende unserer Arbeit scheint sich die Situation langsam zu verbessern, da die Entwicklungsumgebungen immer ausgereifter und funktioneller werden. Einzig die Performance läßt sich wohl nicht so recht in den Griff bekommen.

Einige der oben erwähnten Mängel sind zweifelsohne auf die rasante Entwicklung im gesamten Bereich Java zurückzuführen. Dies zeigt sich schon an den schnell aufeinander folgenden JDK-Versionen. Kaum haben die Hersteller der Entwicklungsumgebungen ihre Programme an die neueste JDK-Version angepaßt, folgt auch schon die nächste. Da ist es schwer, wenn man in einem länger dauernden Projekt aktuell bleiben möchte.

Dieser schnelle Fortschritt macht natürlich erst recht nicht beim JWAM-Rahmenwerk halt, welches selbst ja noch ein junges Produkt ist. Während unserer Arbeit an dem Pausenplaner-System machten wir zwei volle Versionswechsel und etliche kleinerer Updates mit. Aus diesem Grund mußten wir des öfteren unseren Source-Code anpassen.

In Bezug auf die Performance und die Dokumentation sollte am JWAM-Rahmenwerk noch einiges getan werden. Ansonsten ließen sich die vorhandenen Komponenten des Rahmenwerks gut verwenden. Das Konzept für die Anbindung der Oberflächen an die Werkzeuge funktioniert sehr gut (von den im Abschnitt 6.1.4 erwähnten Einschränkungen bei Sub-Werkzeugen vom gleichen Typ abgesehen) und die Werkzeuge lassen sich recht einfach implementieren. Für die Verwendung des Rahmenwerks in einer verteilten Umgebung wäre eine integrierte Benutzerverwaltung sehr sinnvoll. Bis jetzt ist man diesbezüglich auf sich selbst angewiesen. Ebenfalls auf sich gestellt ist der Benutzer in Bezug auf komplexere Präsentationsformen. Bei der Erstellung eigener Präsentationsformen wird der Programmierer aber sehr gut vom Rahmenwerk unterstützt; so gab es bei der Erweiterung des Rahmenwerks durch unsere Präsentationsformen pfTable und pfTextLabel keine größeren Schwierigkeiten.

Insgesamt haben wir die Erfahrung gemacht, daß das JWAM-Rahmenwerk auch in einem etwas umfangreicheren Anwendungsbeispiel der Konstruktion interaktiver

Software nach der WAM-Metapher die erhoffte Unterstützung gewährt, ohne dabei eine unverhältnismäßig lange Einarbeitungszeit zu erfordern. Es gibt ausgereifte Oberklassen für die Interaktions- und die Funktionskomponente der Werkzeuge, mehrere verhältnismäßig einfach zu handhabende Benachrichtigungsmechanismen sowie eine gut ausgestattete Container-Bibliothek. Die Materialkonstruktion wird aber nur durch die Oberklasse ***Thing*** unterstützt und dieses ist, wie wir feststellten, für die Konstruktion von komplexen Materialien zu wenig.

Die Entwicklung des Archives hat sich als die vorraussehbar umfangreiche Aufgabe erwiesen. Aufgrund dieses großen Umfangs war es uns nicht möglich, alle für die Benutzung eines Archives notwendigen Komponenten zu implementieren. Als eines der wichtigsten dieser noch fehlenden Bestandteile ist sicherlich die Anbindung an eine Datenbank zu nennen. Damit einher geht auch der Verzicht auf verschiedene, von Datenbanken erbrachten Dienste. Zuallererst gibt es bei der Benutzung unseres Archives keine Möglichkeit, eine Menge von Materialien als Resultat einer einzigen Anforderung zu erhalten. Unser Archiv liefert dagegen jeweils nur ein Material, das zudem bei der Anforderung mit der zugehörigen ID genau identifiziert sein muß. Andererseits ist unserer Ansicht nach der Umgang mit einem Archiv in der »realen Welt« auch dadurch gekennzeichnet, daß der Archivbenutzer anhand der ihm vorliegenden Inhaltsverzeichnisse das gewünschte Material auffindet und auch nur dieses Material anfordert.

Neben dem Fehlen einer Anforderung von Materialmengen ist der Verzicht auf die Implementierung eines bei Datenbanken üblichen *Commit/Rollback*-Mechanismus ein weiterer Punkt, der die Benutzung unseres Archives derzeit auf kleine Anwendungen beschränkt.

Ein anderer noch vollkommen unberührter Aspekt ist die Sicherung gegen Systemausfälle. Dieses für eine ernsthafte Anwendung sicherlich äußerst wichtige Gebiet haben wir ebenfalls aus naheliegenden Gründen ausgeklammert.

Trotz der gerade erwähnten Unzulänglichkeiten können wir jedoch feststellen, daß unser Konzept eines Archives, welches nur die in Umschlägen verpackten Materialien verwaltet und damit von der Struktur der verpackten Materialien unabhängig ist, sich zumindest am Beispiel des Pausenplaners als tragfähig erwiesen hat. Die

Verlagerung des Umganges mit komplexen Materialien in den Bereich des Anwendungssystems bietet zum einen die Möglichkeit, eine Standardimplementation nach Beispiel des ***ppMaterialVersorgers*** und der von diesem verwandten Materialklassen anzubieten, zum Anderen läßt dies aber auch andere, auf spezielle Anwendungsfälle zugeschnittene Lösungen zu. Als ebenfalls brauchbarer Ansatz hat sich unsere rudimentäre Unterstützung der Kooperationsformen herausgestellt. Bei der Implementierung des Pausenplaners erwies sich unsere statische Lösung, die schon zur Kompilierzeit mehrere Fehler im kooperativen Umgang mit Materialien aufzeigte, als recht vorteilhaft.

Insgesamt kommen wir zu dem Ergebnis, daß unser Archivkonzept sich zum größeren Teil bewährt hat, unsere Implementierung dieses Konzeptes jedoch nicht zuletzt aufgrund der oben beschriebenen Mängel für einen ernsthaften Einsatz untauglich ist.

Damit sich dieses in Zukunft ändert, wird noch im Rahmen anderer Studienarbeiten an Archiven gearbeitet[43]. Uns bleibt die Hoffnung, mit unserer Arbeit dafür vielleicht ein paar Anregungen zu liefern, bzw. manchen Weg in eine Sackgasse zu ersparen.

8 Literaturverzeichnis

Holger Bohlmann:

Christiane Floyd: Einführung indie Softwaretechnik.Scriptum zur gleichnamigen Vorlesung, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softtwaretechnik, Hamburg 1997.

Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides: Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software. Deutsche Übersetzung von Dirk Riehle, (Bonn 1996).

Andreas Hartmann: Softwarekonstruktion nach WAM zur Unterstützung von Kooperation an einem Ort am Beispiel eines Pausenplaner-Systems. Diplomarbeit, Uni Hamburg 1998

Andreas Havenstein, (Titel noch unbekannt), Studienarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik , begonnen im Oktober 1998.

Javasoft: „Java-Beans 1.0 API specification“, Sun Microsystems, 1996.

Javasoft: „Java Object Serialization specification“, Sun Microsystems, 1996.

43 z.B. Andreas Havenstein: (Titel noch unbekannt), Studienarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik , begonnen im Oktober 1998

Eike Jessen, Rüdiger Valk: Rechensysteme. Grundlagen der Modellbildung. Berlin, Heidelberg, New York, Paris, Tokyo (1987).

Tim Krauß: Softwarekonstruktion nach WAM unter Nutzung des MFC Rahmenwerks. Diplomarbeit, Uni Hamburg 1998

Sven Lammers, (Titel noch unbekannt) Diplomarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik 1998.

Carola Lilienthal: Kooperation mit komplexen Materialien in verteilten Systemen. Exposé zum Dissertationsvorhaben. Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik April 1998.

Martin Lippert: „Konzeption und Realisierung eines GUI-Frameworks in Java nach der WAM-Metapher“, Studienarbeit Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik 1997.

Michael Otto, Norbert Schüler: (Titel noch unbekannt), Studienarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik , 1998.

Dirk Riehle: Entwurfsmuster für Softwarewerkzeuge. Gestaltung und Entwurf von Anwendungen mit grafischer Benutzungsoberfläche. Bonn (1997).

Dirk Weske, Martina Wulf: Konzepte zur Materialversorgung verteilter Werkzeugumgebungen am Beispiel der Anbindung einer objektorientierten Datenbank. Studienarbeit, Universität Hamburg, Fachbereich Informatik, Arbeitsbereich Softwaretechnik März 1994.

Heinz Züllighoven: Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material-Ansatz, Heidelberg (1998).