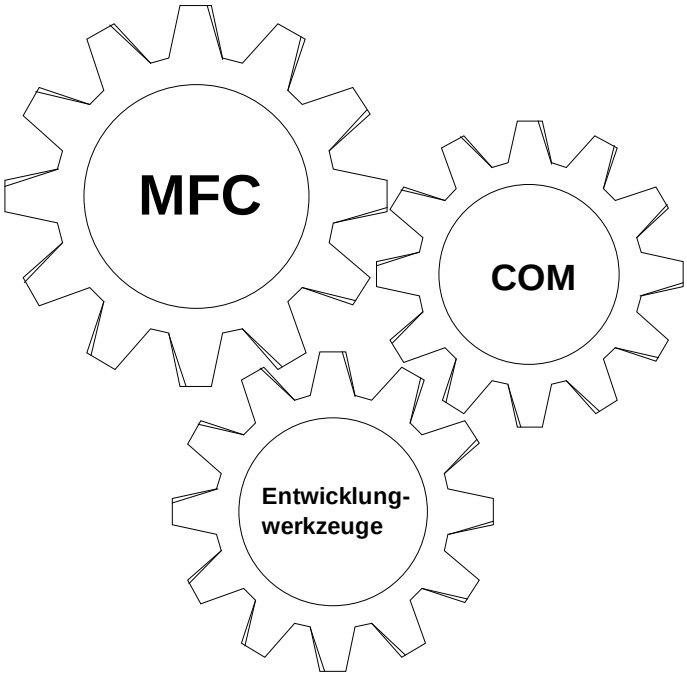
Softwarekonstruktion nach WAM unter Nutzung des MFC Rahmenwerks

Diplomarbeit

Universität Hamburg
Fachbereich Informatik
Arbeitsbereich Softwaretechnik



Tim-Oliver Krauß
Alte Wöhr 15
22307 Hamburg

Betreuer: Prof. Dr. Heinz Züllighoven
Zweitbetreuer: Dr. Daniel Moldt

Diese Diplomarbeit wurde am Fachbereich Informatik der Universität Hamburg eingereicht.

Ich versichere hiermit, diese Arbeit selbständig und unter ausschließlicher Zuhilfenahme der in der Arbeit aufgeführten Hilfsmittel erstellt zu haben.

Tim-Oliver Krauß, Hamburg den 11. März 1999

Inhaltsverzeichnis

1 Einleitung

"Frameworks werden immer wichtiger. Mit ihrer Hilfe erreichen objektorientierte Systeme den höchsten Grad an Wiederverwendung...Der Großteil des Anwendungsentwurfs und Anwendungscodes wird aus den verwendeten Frameworks stammen oder von ihnen beeinflußt sein."

Gamma et al. [GHJ+96]

Das MFC (Microsoft Foundation Classes, [MFC97], [Feu97], [Bra96], [Pro96]) Rahmenwerk[1] findet zunehmende Verbreitung in der Welt der objektorientierten Anwendungsentwicklung. Zur Zeit gibt es viele industrielle Projekte, in denen das Rahmenwerk und die dazugehörigen Entwicklungswerkzeuge zum Einsatz kommen. Das MFC Rahmenwerk ermöglicht die Entwicklung von Anwendungen für die Betriebssysteme Windows 95, Windows NT, Windows 3.11 und MacOS. Es ist unabhängig von einer Anwendungsdomäne, stellt jedoch die Bearbeitung von Dokumenten in den Vordergrund.

Der zentrale Bestandteil des MFC Rahmenwerks ist die Document/View Architektur. Sie läßt dem Entwickler erhebliche Freiheiten, auf welche Art und Weise er seine Softwaresysteme konstruieren möchte. Technische Unterstützung für die Konstruktion findet der Entwickler im Rahmenwerk und vor allem in den Werkzeugen der Entwicklungsumgebung. Ich meine jedoch, daß die Document/View Architektur dem Entwickler in seiner Arbeit nicht genug methodische Unterstützung gibt.

WAM (Werkzeug Automat Material, [Zül98]) ist ein am Arbeitsbereich Softwaretechnik (FB Informatik) der Universität Hamburg entwickelter Methodenrahmen, der die Konstruktion interaktiver Software ermöglicht. WAM basiert auf den Entwurfsmetaphern Werkzeug und Material, die den Anwendungsentwurf charakterisieren, sowie auf den Entwurfsmustern, welche die Umsetzung der Metaphern beschreiben.

Der Kern der Arbeit beschreibt, wie das MFC Rahmenwerk benutzt werden kann, um mit dem WAM Methodenrahmen Software zu konstruieren. Dabei analysiere ich, inwiefern die für WAM benötigten Entwurfsmuster Bestandteil des MFC Rahmenwerks sind und ob diese Muster qualitativ den Ansprüchen nach WAM genügen. Der Schwerpunkt meiner Arbeit liegt auf der Werkzeugkonstruktion nach WAM, wobei ich zeige, welche komponentenbasierten Konzepte das MFC Rahmenwerk für eine erweiterte Werkzeugkonstruktion bietet. Die Schwerpunkte, die in meiner Arbeit bezüglich der Werkzeugkonstruktion gesetzt werden, kann man wie folgt zusammenfassen:

- Trennung von Funktion und Interaktion
- Umsetzung von Reaktionsmustern
- Komponentenbasierte Konstruktion
- Nutzung der Entwicklungsumgebung

Der an die Arbeit geknüpfte Anspruch besteht darin, praktikable Lösungen zu erarbeiten. Die Entwickler sollten nicht vor eine Unmenge von Problemen und Anpassungsschwierigkeiten gestellt werden, bevor ein effizientes Arbeiten möglich wird. Dies soll dadurch erreicht werden, indem die gegebenenfalls notwendigen Änderungen und Erweiterungen einen möglichst geringen Umfang haben. Die in dieser Arbeit vorgestellten Lösungen ermöglichen die Softwarekonstruktion nach WAM, ohne daß man auf die Hilfsmittel der Entwicklungsumgebung verzichten muß.

Um die praktische Anwendbarkeit der ausgearbeiteten Lösungen zu überprüfen, wurde das **Pausenplanersystem** als Beispielanwendung entworfen und implementiert. Das Pausenplanersystem kann verwendet werden, um die Pausenaufsichten an einer Schule einzuteilen. Es wurde als verteilte Anwendung entworfen, die die Gruppenarbeit der an der Pausenplanung beteiligten Personen unterstützt.

1 Gleichbedeutend mit den englischen Begriffen Framework oder Application Framework. Ich verwende in meiner Arbeit durchgehend die deutsche Übersetzung Rahmenwerk.

1.1 Zusammenarbeit und Aufteilung der Themen

Diese Diplomarbeit ist kein Werk einer Einzelperson. Sie hat sich aus einer gemeinschaftlichen Arbeit entwickelt und ist daher ein Teil dieser Zusammenarbeit. Gemeinsam mit Andreas Hartmann habe ich längere Zeit über WAM Softwarekonstruktion mit dem MFC Rahmenwerk sowie über neue Kooperationsformen und Verteilung des Pausenplanersystems nachgedacht. In diesem Arbeitsprozeß wurden Vorschläge diskutiert und Lösungen erarbeitet. Gemeinsam haben wir das Pausenplanersystem entworfen und implementiert.

Die Beschreibung dieser Zusammenarbeit findet sich nun in zwei verschiedenen Diplomarbeiten wieder. Schon zu einem recht frühen Zeitpunkt im gemeinsamen Arbeitsprozeß fand eine Aufteilung der Themen statt.

Die Diplomarbeit von Andreas Hartmann [Har98] beschäftigt sich schwerpunktmäßig mit Kooperationsformen und der Entwurfsmetapher Gruppenarbeitsumgebung. Sie beschreibt die Konstruktion der Gruppenarbeitsumgebung und der Materialien unter Nutzung des MFC Rahmenwerks.

Wie in der Einleitung beschrieben, beschäftigt sich meine Arbeit schwerpunktmäßig mit der Werkzeugkonstruktion. Dennoch möchte ich dem Leser einen vollständigen Einblick in sämtliche Bereiche der Softwarekonstruktion nach WAM geben. Daher werden in diesem Text die dazu benötigten Themen aus der Arbeit von Andreas Hartmann einführend beschrieben. Das MFC Rahmenwerk und die COM Technologie liefern die technische Grundlage für das Pausenplanersystem. Beide Themen sowie die MFC Entwicklungswerkzeuge werden schwerpunktmäßig in meiner Arbeit behandelt.

1.2 Aufbau des Textes

Kapitel 2 folgt dieser Einleitung und beschreibt die Werkzeugkonstruktion nach WAM. Es stellt die Entwurfsmetapher Werkzeug vor und vermittelt dem Leser grundlegende Entwurfsmuster der Werkzeugkonstruktion.

Kapitel 3 stellt die Beispielanwendung Pausenplanersystem vor. Dabei werden zunächst die Tätigkeiten der Pausenplanung beschrieben und Formen der Zusammenarbeit analysiert. Die danach folgenden Abschnitte des Kapitels beschreiben Kooperationsmodell, Werkzeuge, Materialien und Verteilung des Pausenplanersystems.

Kapitel 4 stellt die Entwicklung des MFC Rahmenwerks dar und macht deutlich für welche Projekte das Rahmenwerk verwendet werden kann. Klassen und Anwendungsarchitektur des Rahmenwerks sowie die verwendeten Nachrichtenmechanismen werden vorgestellt.

In Kapitel 5 erhält der Leser einen Überblick über die wesentlichen Aspekte der Microsoft COM Technologie. Der Abschnitt stellt Teile der Komponententechnologie vor, die bei der Konstruktion des Pausenplanersystems verwendet wurden. Dabei wird der Automationsdienst und die Integration des Dienstes in das MFC Rahmenwerk beschrieben.

Im Anschluß daran erfolgt in Kapitel 6 die Beschreibung der Entwicklungswerkzeuge des MFC Rahmenwerks.

Das anschließende Kapitel 7 beschreibt die komponentenbasierte Werkzeugkonstruktion mit dem MFC Rahmenwerk. Es bildet damit den Kern dieser Arbeit. Ausgehend von der Konstruktionsidee wird die Umsetzung der Werkzeug Entwurfsmuster beschrieben. Die danach folgenden Abschnitte des Kapitels beschreiben den verwendeten Ereignismechanismus.

Kapitel 8 beschreibt die komponentenbasierte Konstruktion des gesamten Pausenplanersystems und geht dabei auf die Gruppenarbeitsumgebung, die Materialien und die Kopplung von Werkzeug und Material genauer ein.

In der Zusammenfassung gehe ich noch einmal auf die Arbeitsergebnisse ein und versuche diese zu bewerten. Abschließend erfolgt ein Ausblick auf mögliche Entwicklungen in der Zukunft.

1.3 Graphische Konventionen

Die verwendeten Klassendiagramme und Objektdiagramme basieren auf der Notation des Objektmodells der Object Modeling Technique (OMT, siehe [RBP+93]). Die Notation der Objektkommunikation basiert auf den Interaktionsdiagrammen von Jacobson [Jac92]. Ich verwende in meiner Arbeit zusätzliche Notationsformen, welche die oben genannten Klassen- und Objektdiagramme erweitern. Zu diesen zusätzlichen Notationsformen zählen die Darstellung von Rahmenwerken und Prozeßräumen sowie eine Notation, welche die Kommunikation zwischen COM Komponenten [COM97] beschreibt.

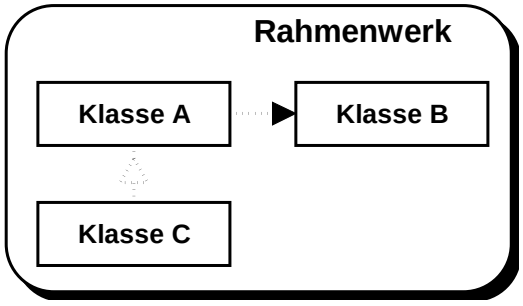


Abb. a: Ein Rahmenwerk enthält die Klassen A,B und C

Ein Rahmenwerk oder Subrahmenwerk faßt eine Menge semantisch abhängiger Klassen zusammen. Man verwendet die Rahmenwerk Notation, um diesen Zusammenhang graphisch abzubilden. Die Notation kann die enthaltenen Klassen und deren interne Struktur verbergen, um eine übersichtlichere Darstellung von großen Rahmenwerken zu erreichen.

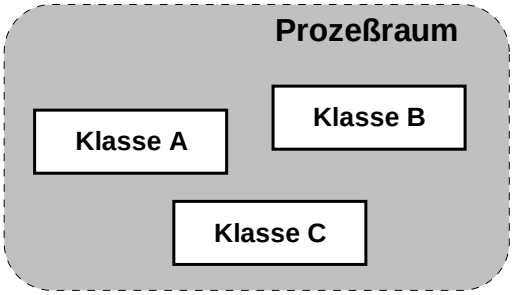


Abb. b: Objekte der Klassen A,B und C befinden sich gemeinsam innerhalb eines Prozeßraums

Besteht ein Anwendungssystem aus mehreren Prozeßräumen, so kann man mit dieser Notation beschreiben welche Komponenten des Systems welchen Prozeßräumen zugeordnet sind. Ein Prozeßraum existiert nur zur Laufzeit des Systems und ist damit ein dynamisches Gebilde. Es ist jedoch möglich, auch statische Komponenten innerhalb eines Prozeßraums anzuordnen. Bei der Anordnung von Klassen bedeutet die Darstellung, daß sich die Objekte dieser Klassen zur Laufzeit innerhalb des Prozeßraums befinden.

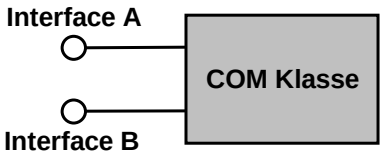


Abb. c: Eine COM Klasse bietet die Interfaces A und B an ihrer Schnittstelle an

Diese Darstellung wird auch „Plug-in Jack“ Notation genannt. Sie verwendet die Metapher von Komponenten einer Stereoanlage, die mit Hilfe von Steckern (Pfeile) und Buchsen (Kreise) miteinander verbunden werden können.

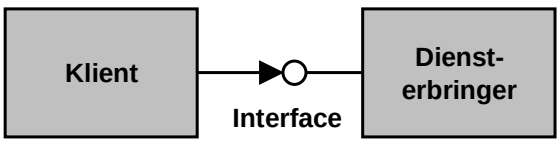


Abb. d: Ein COM Klient greift über ein Interface auf einen COM Diensterbringer zu

2 Werkzeugkonstruktion nach WAM

Der am Arbeitsbereich SWT entwickelte Methodenrahmen WAM (Werkzeug Automat Material. Siehe [KGZ94], [Gry95], [Zül98]) stellt die Entwurfsmetaphern Werkzeug, Material und Automat für die Konstruktion objektorientierter interaktiver Anwendungssysteme zur Verfügung. Da der Schwerpunkt meiner Arbeit auf der Konstruktion von Werkzeugen liegt, möchte ich in diesem einführenden Kapitel auf die Entwurfsmetapher Werkzeug genauer eingehen.

2.1 Entwurfsmetapher Werkzeug

Werkzeuge sind Arbeitsmittel einer Anwendung, sie eignen sich für die Bearbeitung von Materialien und vergegenständlichen Erfahrungen und Traditionen. Materialien sind Arbeitsgegenstand einer Anwendung, sie lassen sich in bestimmter Weise bearbeiten, d.h. verändern oder sondieren. In [Gry95] wird der Begriff Werkzeug wie folgt definiert:

> „Ein Werkzeug ist ein Arbeitsmittel, mit dem in einer bestimmten Arbeitssituation ein Arbeitsgegenstand, das Material, bearbeitet wird. Ein Werkzeug verändert ein Material oder sondiert seinen Zustand. Mit einem Werkzeug wird sowohl eine fachliche Funktionalität („es ist zu etwas gut“) als auch eine bestimmte Art der Handhabung („es muß richtig gehandhabt werden“) verbunden.“

Dabei verhalten sich Werkzeuge stets reaktiv. Dies bedeutet, daß Aktionen immer durch Handlungen des Benutzers ausgelöst werden. Das Werkzeug wird zu keiner Zeit selbst aktiv und es bearbeitet das Material niemals unbemerkt vom Benutzer. Die vom Werkzeug angebotenen Kommandos können vom Benutzer frei gewählt werden. Dabei macht das Werkzeug keine Annahmen über eine bestimmte Reihenfolge der Werkzeugkommandos. Es kann natürlich möglich sein, daß der Zustand des Werkzeugs die Auswahl bestimmter Werkzeugkommandos einschränkt.

In [KGZ94] sind die Eigenschaften eines Werkzeugs folgendermaßen zusammengefaßt:

- Das Werkzeug hat einen Namen und kann über ein graphisches Symbol aktiviert werden,
- zeigt immer eine Sicht auf das gerade bearbeitete Material, ...
- bleibt bei der Benutzung selbst „im Blick“, d.h. bei der Bearbeitung des Materials ist immer erkennbar, mit welchem Werkzeug es bearbeitet wird,
- bietet mögliche Handhabungen an, um das Material unterschiedlich zu bearbeiten, ...
- zeigt jederzeit seine Einstellungen, die die Materialsicht bestimmt, ...

Bei der technischen Konstruktion von Werkzeugen verwendet man Entwurfsmuster (siehe [Rie95]). Diese dienen als Vorlage für die Konstruktion der Werkzeuge. Ein Entwurfsmuster besteht aus Klassen, Objekten und Operationen. Es beschreibt Struktur sowie Dynamik der beteiligten Komponenten, ihre Zusammenarbeit und die Verteilung der Verantwortlichkeiten.

2.2 IAK-FK Entwurfsmuster

Werkzeuge vereinigen in sich fachliche Funktionalität sowie ihre Darstellung und Handhabung. Sie bestehen demnach aus einer Funktionskomponente und einer Interaktionskomponente. In [Gry95] werden beide Komponenten des Entwurfsmusters wie folgt beschrieben:

> Die **Funktionskomponente** (FK) ist der bewirkende und sondierende Teil des Werkzeugs. In ihr wird die fachliche Funktionalität des Werkzeugs festgelegt. In der FK werden die Operationen zur Bearbeitung des Materials implementiert.
>
> Die **Interaktionskomponente** (IAK) legt die Benutzungsoberfläche des Werkzeugs fest. Dazu nimmt sie Systemereignisse entgegen, ruft die FK und steuert die Präsentation an der Oberfläche. Damit die IAK von den konkreten Handhabungsformen und dem verwendeten Fenstersystem ab-

strahieren kann, benutzt sie im Regelfall für die konkrete Handhabung und Präsentation sogenannte **Interaktionstypen** (IAT), die Komponenten des Fenstersystems kapseln.

Durch geeignete Trennung von Funktion und Interaktion erreicht man die Unabhängigkeit zwischen fachlicher Funktionalität und Handhabung eines Werkzeugs. Diese Eigenschaft vereinfacht die Werkzeugkonstruktion sowie die Änderbarkeit eines Systems und verbessert die Möglichkeiten einer späteren Wiederverwendung.

Das Trennungsprinzip von Interaktion und Funktion ist ein wichtiger Grundsatz innerhalb der Anwendungsentwicklung. Seine Bedeutung wurde von den Mitarbeitern des Xerox Park erkannt und ist in Form des Model-View-Controller Entwurfsmusters in die Smalltalk Entwicklungsumgebung eingegangen (siehe [KP88]). In abgewandelter Form wird das MVC Entwurfsmuster heute in fast jeder Entwicklungsumgebung umgesetzt.

In Abbildung 1 sind die Beziehungen zwischen Interaktionstyp (IAT), Interaktionskomponente (IAK) und Funktionskomponente (FK) dargestellt.

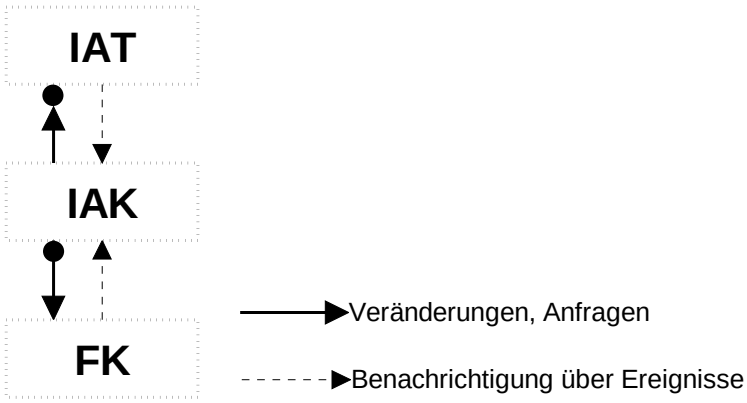


Abbildung 1: Das FK-IAK Entwurfsmuster

Die Interaktionskomponente legt die Benutzungsoberfläche des Werkzeugs fest. Sie stellt Werkzeug und bearbeitetes Material auf der Oberfläche dar und realisiert die Handhabung des Werkzeugs. Die IAK benutzt die FK, wobei sie sondierend oder verändernd auf die FK zugreifen kann. Die Funktionskomponente realisiert die Funktionalität des Werkzeugs sowie die Bearbeitung des Materials. Sie benachrichtigt die IAK im Fall einer Zustandsänderung, so daß die IAK den veränderten Zustand an der Werkzeugoberfläche darstellen kann. Durch die Verwendung mehrerer Interaktionskomponenten (IAKs) können verschiedene Benutzungsoberflächen zu einem Werkzeug realisiert werden. Ein Werkzeug könnte beispielsweise mehrere Sichten auf ein Material zur Verfügung stellen.

Die IAK verwendet mehrere Interaktionstypen (IATs), die die Ein/Ausgabe-Komponenten des zugrundeliegenden Fenstersystems kapseln. Die IAK kann den Zustand der IATs sondieren oder verändern. Diese benachrichtigen die IAK über vom Benutzer ausgelöste Ereignisse, wobei handhabungsbezogene Ereignisse direkt durch die IAK bearbeitet und anwendungsbezogene Ereignisse von der IAK an ihre FK weitergeleitet werden.

Sowohl zwischen IAK und IATs, als auch zwischen IAKs und FK besteht eine besondere Benutztbeziehung. In beiden Fällen benachrichtigen die verwendeten Komponenten ihren Beobachter über eine Zustandsänderung, ohne daß sie den Beobachter konkret kennen. Die Tatsache, daß die Komponenten einen abstrakten Beobachter benachrichtigen, erleichtert die mehrfache Verwendung der Komponenten. Der beschriebene Mechanismus der Benachrichtigung heißt **Reaktionsmechanismus**. Eine allgemeine Beschreibung möglicher Reaktionsmechanismen sowie die zugehörige Konzeption einiger Entwurfsmuster findet man in [RW96].

2.3 Werkzeugkomposition

Heute gebräuchliche Werkzeuge sind komplexe Gebilde, deren programmiersprachliche Beschreibung meist umfangreich sind. Das Werkzeugkomposition-Entwurfsmuster ermöglicht es dem Entwickler seine Werkzeuge hierarchisch zu strukturieren. In [Rie95] wird das Werkzeugkomposition-Entwurfsmuster wie folgt beschrieben:

> „Werkzeuge werden rekursiv aus Werkzeugen aufgebaut. Jedem Werkzeugobjekt in der resultierenden Baumstruktur kommt eine Aufgabe zu, welche es durch eigene Leistung sowie Delegation von Teilaufgaben an Subwerkzeuge realisiert."

Jedes Werkzeug hat demnach eine wohldefinierte Aufgabe. Verwendet es zur Bearbeitung dieser Aufgabe ein oder mehrere **Subwerkzeuge**, so ist es aus der Sicht der Subwerkzeuge ein **Kontextwerkzeug**. In Abbildung 2 habe ich die Beziehung zwischen einem Kontextwerkzeug und zwei Subwerkzeugen dargestellt.

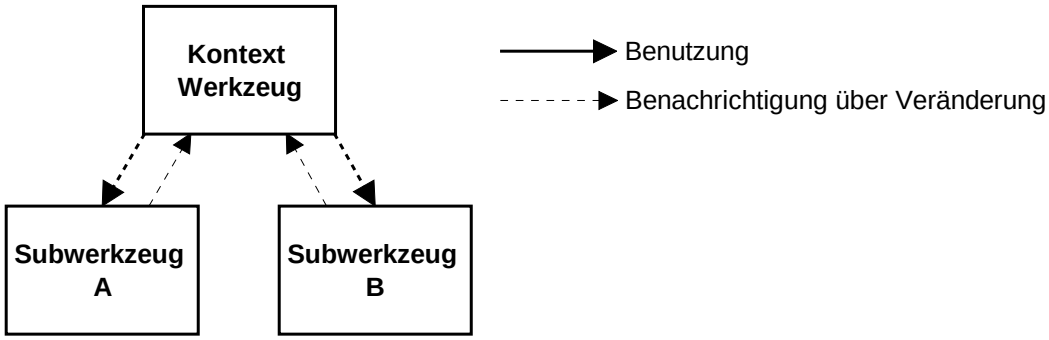


Abbildung 2: Ein zusammengesetztes Werkzeug

Ein zusammengesetztes Werkzeug wirkt nach außen wie ein Werkzeug. Es ist für die Erzeugung und Löschung seiner Subwerkzeuge verantwortlich. Mit der Benutzungsart legt das Kontextwerkzeug den Kontext fest, in dem die Subwerkzeuge verwendet werden. Ausschließlich das Kontextwerkzeug bestimmt, wie es ein Subwerkzeug benutzt. Ein Subwerkzeug sollte daher keine Annahmen über seinen Kontext oder seine Verwendung machen, da sonst seine Wiederverwendung eingeschränkt wird.

Subwerkzeuge kennen aus diesem Grund ihr Kontextwerkzeug nicht direkt. Wie bei der Beziehung zwischen IAK und FK benachrichtigen sie einen abstrakten Beobachter, wenn sich ihr Zustand verändert.

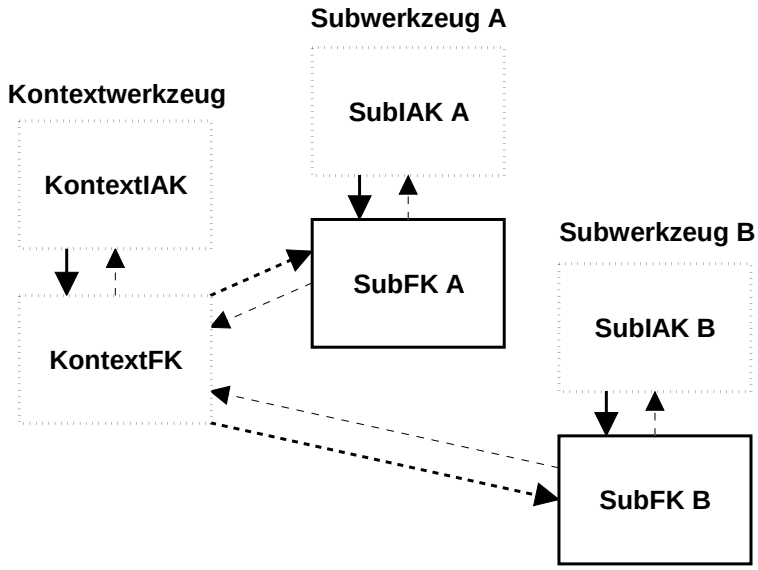


Abbildung 3: Werkzeugkomposition im Detail

Will man Werkzeuge wie Komponenten benutzen und wiederverwenden, so muß ein Werkzeug von außen über eine Schnittstelle ansprechbar sein. Bei der vorgestellten Werkzeugkonstruktion schließt die Funktionskompo-

nente das Werkzeug nach außen hin ab. Dabei vertritt die FK das Werkzeug gegenüber anderen Kontextwerkzeugen oder Systemkomponenten. Die Schnittstelle der Funktionskomponente wird daher auch Werkzeugschnittstelle genannt. An der Schnittstelle der FK übernimmt meist nur ein Teil der Methoden die Aufgabe der Werkzeugschnittstelle. Der andere Teil besteht aus sondierenden und verändernden Methoden, welche von der Interaktionskomponente des Werkzeugs benutzt werden. Abbildung 3 zeigt, welche Werkzeugkomponenten in einem zusammengesetzten Werkzeug miteinander kommunizieren.

3 Das Pausenplanersystem

Die Pausenplanung einer Schule ist eine Beispielanwendung, die in Arbeiten, Seminaren und Praktika am Arbeitsbereich Softwaretechnik der Universität Hamburg Verwendung findet (siehe [FL97] und [BGW97]). Die Aufgabe der Pausenplanung besteht darin, die Lehrer der Schule zu den Pausenaufsichten einzuteilen. Um die Einteilung der Lehrer zu den Pausenaufsichten festzuhalten, wird ein Pausenplan erstellt. Das hier vorgestellte Pausenplanersystem soll den Benutzern helfen eine zweckmäßige Pausenplanung durchzuführen.

Im Rahmen dieser Arbeit haben wir das Pausenplanersystem als Prototypen entworfen und konstruiert, um die Anwendbarkeit der, in dieser Arbeit ausgearbeiteten, Konstruktionslösungen zu überprüfen. Das Pausenplanersystem wurde als verteilte Anwendung entworfen, die die Gruppenarbeit der an der Pausenplanung beteiligten Personen unterstützt. Grundlagen zur rechnergestützter Gruppenarbeit in verteilten Systemen werden in [Sch96] beschrieben.

Damit sich der Leser in die Beispielanwendung hineindenken kann, werden in den nächsten Abschnitten zunächst die fachlichen Aspekte der Pausenplanung beschrieben. In den weiteren Kapiteln werde ich dann genauer auf den technischen Entwurf und die Konstruktion des Pausenplanersystems eingehen.

3.1 Pausenplanung und Kooperation

Dieser Abschnitt beschreibt Aufgaben und Zusammenarbeit der an der Pausenplanung beteiligten Personen. Dabei soll der Leser einen Überblick über die Tätigkeiten der Personen bekommen, um das Kooperationsmodell des Pausenplanersystems verstehen zu können.

Ausgangspunkt für die Analyse der Pausenplanung waren die Szenarien, Systemvisionen und CRC-Karten, welche uns vom Arbeitsbereich Softwaretechnik der Universität Hamburg zur Verfügung gestellt wurden. Im Rahmen der Analysetätigkeit sind weitere Entwurfsdokumente entstanden, welche die Kooperation der an der Pausenplanung beteiligten Personen genauer beschreiben.

An einer Schule wird die Aufgabe der Pausenplanung meist von mehreren Personen erledigt. Diese gemeinsame Aufgabe erfordert die Kooperation aller beteiligten Personen, welche dabei ihre Handlungen wechselseitig abstimmen müssen. Die beteiligten funktionellen Rollen sind:

- Pausenplanersteller

Der Pausenplanersteller ist für die Erstellung des Pausenplans verantwortlich. Dabei muß er dafür Sorge tragen, daß die Aufsichtsorte der Schule während der Pausenzeiträume von Lehrern beaufsichtigt werden. Bei der Erstellung des Pausenplans muß der Pausenplanersteller darauf achten, daß die Einteilung der beaufsichtigenden Lehrer keine Konflikte hervorruft. Ein Lehrer darf beispielsweise nicht während seiner Abwesenheit eingeteilt werden; auch kann ein Lehrer zur selben Zeit nicht mehreren Aufsichtsorten zugeteilt sein. Zusätzlich sollte die Einteilung der Pausenaufsichten auf den Lehrkörper der Schule gerecht verteilt werden, wobei halbe und dreiviertel Stellen zu berücksichtigen sind. Der Pausenplanersteller bestimmt die Aufsichtsorte und entscheidet wie viele Pausenaufsichten an einem Aufsichtsort nötig sind. Zusätzlich bestimmt er zu welchen Zeiten eine Beaufsichtigung der Aufsichtsorte nötig ist. Fällt eine Pausenaufsicht aus, so muß der Pausenplanersteller für eine Vertretung sorgen. Zu diesem Zweck sollte er neben den Pausenaufsichten auch Lehrer für die Vertretung einteilen.

- Sekretariatsmitarbeiter

Der Sekretariatsmitarbeiter ist für die Verwaltung des Lehrkörpers und die Unterrichtseinteilung zuständig. Im Rahmen der Pausenplanung sind die Zeiträume der Abwesenheit, im weiteren Text Ausschlußzeiträume genannt, sowie die Stelle eines Lehrers von Interesse. Die Ausschlußzeiten ergeben sich aus dem Lehrplan und der Unterrichtseinteilung eines Lehrers. Bei Änderungen informieren die Lehrer den Sekretariatsmitarbeiter. Dieser sorgt für die Modifikation der Ausschlußzeiten.

Sowohl der Pausenplanersteller als auch der Sekretariatsmitarbeiter halten ihre Arbeitsergebnisse fest. Zu diesem Zweck verwenden sie geeignete Materialien. Diese werden üblicherweise in einem Verwaltungsraum der Schule

aufbewahrt, so daß alle an der Pausenplanung beteiligten Personen darauf zugreifen können. Die Bearbeitung der Materialien findet normalerweise am Arbeitsplatz der Person statt, ist jedoch auch am Aufbewahrungsort der Materialien möglich. Eine zeitlich umfangreiche Bearbeitung wird der Pausenplanersteller an seinem Arbeitsplatz durchführen. Zeitlich kurze Bearbeitung findet oft am Aufbewahrungsort des Materials statt.

Möchte eine Person ein Material bearbeiten und das Material befindet sich nicht an seinem Aufbewahrungsort, so muß sie davon ausgehen, daß das Material bereits von einer anderen Person bearbeitet wird. Sie kann nun abwarten und die Änderungen durchführen, sobald das Material wieder an seinem Platz steht. Es ist jedoch sinnvoller, wenn sie sich zu dem momentanen Aufenthaltsort des Materials begibt, um sich dort mit der bearbeitenden Person zu koordinieren.

Durch die sofortige Kooperation beider beteiligter Personen können unnötige Handlungen vermieden werden. Beispielsweise sollte der Pausenplanersteller das Ausscheiden eines Lehrers aus dem Lehrkörper vom Sekretariatsmitarbeiter erfahren, bevor er diesen Lehrer für Pausenaufsichten einteilt. Die hier beschriebene Art der Kooperation beteiligter Personen nennt man **synchrone** Kooperation, da die Kooperation zeitlich übereinstimmend oder zeitgleich stattfindet.

Wartet die Person bis die Änderungen am Material durchgeführt sind und beginnt dann mit der Bearbeitung, so findet die Kooperation anhand des Materials statt. Für den nachfolgenden Bearbeiter kann es dabei wichtig sein, daß der Vorgänger seine Änderungen am Material kenntlich gemacht hat. Die Kooperation der bearbeitenden Personen in dieser Situation findet nicht gleichzeitig statt. Kooperation, die nicht gleichzeitig stattfindet, nennt man **asynchrone** Kooperation.

Arbeitsplatz, Aufbewahrungsort und Aufenthaltsort des Materials sind **Orte**, die bei der Pausenplanung eine große Rolle spielen. Man kann beobachten, daß die Orte an denen gemeinsame Arbeit stattfindet, oft die Art der Kooperation beeinflussen.

Im Rahmen der Pausenplanung findet synchrone Kooperation normalerweise an einem Ort statt. Die beteiligten Personen begeben sich an einem gemeinsamen Ort, um ihre Handlungen zu koordinieren. An der Schule könnte das ein Konferenzraum oder das Zimmer des Pausenplanerstellers sein. Wichtig ist, daß alle beteiligten Personen den Ort kennen und ihn auch erreichen können. Synchrone Kooperation, an verschiedenen Orten, kann meist nur durch die Verwendung technische Hilfsmittel, wie zum Beispiel Telefon oder Videokonferenzsystem, erfolgen.

Bei asynchroner Kooperation ist es nicht notwendig, daß sich die beteiligten Personen an einem gemeinsamen Ort treffen, um ihre Handlungen zu koordinieren. Das Material wird üblicherweise am persönlichen Arbeitsplatz bearbeitet. Gemeinsame Orte spielen demnach bei asynchroner Kooperation eine geringere Rolle als bei synchroner Kooperation.

Koordination nennt man die wechselseitige Abstimmung bei der Kooperation. Die Art der Kooperation bestimmt meist die Form der Koordination.

Im Falle synchroner Kooperation erfolgt die Koordination des Arbeitsprozesses meist durch Kommunikation zwischen den beteiligten Personen. Bei der gemeinsamen Bearbeitung eines Materials, kann sowohl der Bearbeitungszustand des Materials als auch die Tätigkeit der beteiligten Personen zusätzliche Informationen bereitstellen. Diese indirekt wahrgenommene Information über den Arbeitsprozeß kann die Kommunikation zwischen den Kooperationspartnern zum Teil ersetzen.

Bei asynchroner Kooperation erfolgt die Koordination der beteiligten Personen oft durch die Einhaltung von Regeln. Diese legen die Verantwortlichkeiten von Personen und Tätigkeiten im kooperativen Arbeitsprozeß fest. Die Einhaltung der Regeln kann dabei die Kommunikation zwischen den kooperierenden Personen teilweise ersetzen.

Für die hier vorgestellte Aufgabe der Pausenplanung an einer Schule hat die synchrone Kooperation der beteiligten Personen an einem gemeinsamen Ort viele Vorteile. Diese Vorteile lassen sich folgendermaßen begründenden:

Die Anzahl der an der Pausenplanung beteiligten Personen ist gering. Dabei ist es nachvollziehbar, daß sich die Koordinationsmöglichkeiten mit sinkender Anzahl der beteiligten Kooperationspartnern verbessern. Die an der Pausenplanung beteiligten Personen kennen die Aufgaben und Verantwortlichkeiten der anderen Kooperationspartner und sind um eine möglichst intensive Kooperation der Gruppenmitglieder bemüht. Bei der Pausenpla-

nung gibt es wenige Regeln, die den kooperativen Arbeitsprozeß festlegen, daher ist flexibles Handeln der beteiligten Personen nötig. Die Kooperationspartner werden im Rahmen der Pausenplanung immer wieder auf neue Situationen im Arbeitsprozeß stoßen, welche sie durch synchrone Kooperation am einfachsten bewältigen können.

3.2 Die Entwurfsmetapher Gruppenarbeitsumgebung

Menschen benötigen für die Durchführung von Aufgaben einen Ort, an dem sie die benötigten Arbeitsmittel und Arbeitsgegenstände entsprechend der Aufgaben einrichten können. Die Anordnung der Gegenstände an diesem Arbeitsplatz wird dabei vom arbeitenden Menschen selbst vorgenommen und hängt von den Ordnungsprinzipien und Gewohnheiten desselben ab. Der WAM Methodenrahmen konkretisiert das Leitbild vom Arbeitsplatz für qualifizierte menschliche Tätigkeit durch die Entwurfsmetapher **Arbeitsumgebung** (siehe [Gry96]).

Die Arbeitsumgebung ist der Ort an dem die Werkzeuge, Automaten und Materialien ihren Platz haben. Der qualifizierte Anwender kann sich seine Arbeitsumgebung gemäß seinen Aufgaben einrichten und bestimmt wann er welche Werkzeuge für die Bearbeitung von Materialien verwenden möchte. Bei der Arbeit ist er an keine festen Arbeitsabläufe oder Vorschriften zur Benutzung von Werkzeugen, Materialien oder Automaten gebunden.

Die Entwurfsmetapher Arbeitsumgebung beschreibt den Arbeitsplatz einer Person. Bei der Kooperation mehrerer Personen, die jeweils in ihrer eigenen Arbeitsumgebung tätig sind, muß die Koordination auf Informationswegen zwischen den Arbeitsumgebungen stattfinden (siehe [Gry96] und [WW94]).

Das Pausenplanersystem soll synchrone und asynchrone Kooperation der beteiligten Personen an einem Ort ermöglichen. Um die Gruppenarbeit zu modellieren haben wir die Entwurfsmetapher Arbeitsumgebung um die Möglichkeit erweitert, daß mehrere Personen in einer Umgebung tätig sein können. Die neue Entwurfsmetapher nennt sich Gruppenarbeitsumgebung und wird in [Har98] wie folgt definiert:

> „Eine **Gruppenarbeitsumgebung** zur Unterstützung qualifizierter menschlicher Tätigkeit, ist der Ort für eine anwendungsfachlich motivierte Zusammenstellung von Werkzeugen, Automaten und Materialien für alle am Kooperationsprozeß beteiligten Personen.“

Die Gruppenarbeitsumgebung ist der Ort an dem die Arbeit einer Personengruppe mit einer gemeinsame Aufgabe stattfindet. Gemeinschaftlich verwendete Materialien, Werkzeuge und Automaten befinden sich in der Gruppenarbeitsumgebung. Damit ein Gruppenmitglied an der gemeinsamen Aufgabe mitwirken kann, muß er sich zur Gruppenarbeitsumgebung begeben, wodurch er Teilnehmer am Kooperationsprozeß wird.

Die Gruppenmitglieder richten sich gemeinsam ihre Arbeitsumgebung ein und sollten gleichberechtigten Zugriff auf Material und Werkzeuge haben. Ob simultaner Zugriff mehrerer Personen auf ein Material notwendig ist, hängt von der Kooperationssituation ab und kann somit nur von den beteiligten Personen entschieden werden. Eine Gruppenarbeitsumgebung sollte daher die Verfügbarkeit der Materialien nicht künstlich einschränken.

Synchrone Kooperation innerhalb der Gruppenarbeitsumgebung macht die Kommunikation zwischen den Mitgliedern der Umgebung besonders wichtig. Eine Gruppenarbeitsumgebung sollte daher die Kommunikation zwischen den Mitgliedern ermöglichen.

Für jedes Mitglied einer Gruppenarbeitsumgebung ist ein hohes Maß an fachlicher Transparenz notwendig. Jede Person sollte erkennen können, welche Gruppenmitglieder sich in der Arbeitsumgebung aufhalten und welche Tätigkeiten sie verrichten. Dabei ist es wichtig, daß die Person sieht welche Werkzeugen seine Kooperationspartner verwenden und welche Materialien gerade in Bearbeitung sind. Diese Beobachtungen sind eine grundlegende Voraussetzung für die Koordination der beteiligten Personen.

Die Entwurfsmetaphern Arbeitsumgebung und Gruppenarbeitsumgebung harmonieren miteinander. Es ist durchaus vorstellbar, daß eine Person Arbeit sowohl an ihrem persönlichen Arbeitsplatz als auch an einem Gruppenarbeitsplatz verrichten kann. Dabei sollte es möglich sein, daß die Materialien zwischen den Arbeitsplätzen transportiert werden können.

Der nächste Abschnitt beschreibt die Gruppenarbeitsumgebung des Pausenplanersystems. Weitere Grundlagen zu Gruppenarbeit und WAM, sowie alternative Konzepte zur Gruppenarbeitsumgebung kann der Leser in [RW97] nachlesen.

3.3 Die Gruppenarbeitsumgebung des Pausenplanersystems

An einer Schule arbeiten die an der Pausenplanung beteiligten Personen in ihren Arbeitsräumen und besitzen für ihre Tätigkeit einen Arbeitsplatzrechner, die über ein Netzwerk miteinander verbunden sind. Die Arbeitsräume und Arbeitsplätze der beteiligten Personen liegen meist räumlich getrennt. Um diese räumliche Trennung im Sinne der Pausenplanung überwinden zu können, setzt das Pausenplanersystem die Entwurfsmetapher Gruppenarbeitsumgebung durch die Verwendung eines **virtuellen Schwarzen Bretts** um. Im Pausenplanersystem begeben sich die beteiligten Personen an einen gemeinsamen virtuellen Ort, an dem sie ihre Zusammenarbeit koordinieren können.

Jedes Gruppenmitglied muß sich zunächst mit seinem Namen im System anmelden, daraufhin bekommt das Mitglied am Bildschirm seines Arbeitsplatzrechners eine Abbildung des Schwarzen Bretts präsentiert (siehe Abbildung 4). Dort sind die für die Pausenplanung relevanten Arbeitsgegenstände angebracht. Werkzeuge und Materialien werden in Form von Symbolen am Schwarzen Brett sichtbar gemacht.

Die Mitglieder finden die Materialtypen **Pausenplan** und **Lehrkörper** am Schwarzen Brett vor. Sie können neues Material anheften oder nicht mehr benötigtes Material abnehmen. Neue Materialien eines bestimmten Typs können dabei von dem jeweiligen Materialstapel entnommen werden.

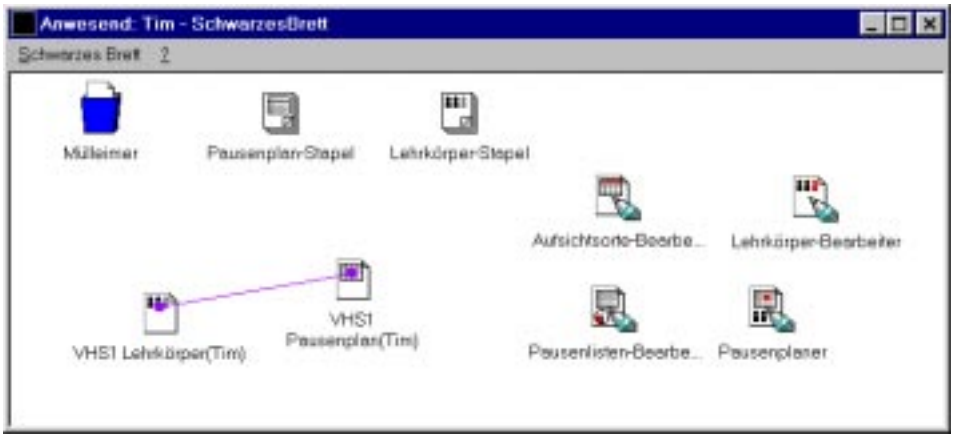


Abbildung 4: Das Schwarze Brett

Am Schwarzen Brett des Pausenplanersystems stehen die Werkzeuge **Lehrkörper-Bearbeiter**, **Aufsichtsorte-Bearbeiter, Pausenlisten-Bearbeiter** und **Pausenplaner** den Gruppenmitgliedern für die Bearbeitung der angehefteten Materialien zur Verfügung. Von einem Werkzeug können während der Bearbeitung beliebig viele Exemplare verwendet werden, was an einem realen Arbeitsplatz naturgemäß nicht möglich ist.

Die Gruppenmitglieder können beobachten welche Personen am Schwarzen Brett anwesend sind, und welches Material gerade von wem bearbeitet wird. Dabei ist der Name der bearbeitenden Person in Klammern hinter dem Materialnamen vermerkt. Besondere Materialzustände, wie zum Beispiel der Belegungskonflikt im Pausenplan-Material, werden durch Veränderung des Materialsymbols kenntlich gemacht. Die simultane Bearbeitung eines Materials durch mehrere Gruppenmitglieder ist möglich. Die Werkzeuge bilden bei der Bearbeitung immer den aktuellen Materialzustand ab und informieren darüber, welche Personen das bearbeitete Material im Zugriff haben. Wichtige Ereignisse im Arbeitsprozeß, wie das An- bzw. Abmelden einer Person oder das Anheften bzw. Abnehmen von Materialien, werden dem Benutzer durch ein Benachrichtigungsfenster signalisiert.

Die Handhabungs- und Werkzeugvisionen im Anhang repräsentieren Dokumente des objektorientierten Entwurfsprozesses nach WAM. Sie beschreiben detailliert die gemeinsame Arbeit am Schwarzen Brett und die Handhabung der Werkzeugen.

Am Schwarzen Brett ist nicht nur synchrone Kooperation, sondern auch asynchrone Kooperation der Gruppenmitglieder möglich. Zu diesem Zweck besitzen die Materialien eine Art Zugriffsstempel, der den Zeitpunkt der letzten Bearbeitung und den Namen der bearbeitenden Person speichert. Die Mitglieder können zusätzlich Notizzettel an das Schwarze Brett hängen, um ihren Kooperationspartnern wichtige Nachrichten zukommen zu lassen.

Möglichkeiten zur expliziten Kommunikation zwischen den Gruppenmitgliedern wurden von uns nicht im Prototypen des Pausenplanersystems umgesetzt. Beim Entwurf des Schwarzen Bretts sind wir zunächst davon ausgegangen, daß die Gruppenmitglieder die Kommunikation außerhalb des Pausenplanersystems organisieren. An einer Schule liegen die Räume des Pausenplanerstellers und das Sekretariat meist nicht weit auseinander. Ein Gruppenmitglied könnte sich daher kurzerhand in den Arbeitsraum seines Kooperationspartners begeben, um dort mit ihm zu kommunizieren. Sind die Wege weiter, könnte die Kommunikation unter Verwendung des Telefons stattfinden. Die Integration einer Videokonferenzsystem-Software in das Pausenplanersystem würde die Kommunikation der Gruppenmitglieder auf ihren Arbeitsplatzrechnern ermöglichen.

3.4 Technische Interpretation der Gruppenarbeitsumgebung

Dieser Abschnitt beschreibt den technischen Entwurf der Gruppenarbeitsumgebung des Pausenplanersystems. Der hier vorgestellte Entwurf lehnt sich an die technische Interpretation der Entwurfsmetapher Arbeitsumgebung an (siehe [Gry96]), wurde jedoch um einige Konzepte und Komponenten erweitert.

Das Pausenplanersystem ist ein verteiltes System. Die bei der Pausenplanung beteiligten Personen arbeiten an ihren Arbeitsplatzrechnern, welche über das Netzwerk der Schule verbunden sind. Die einzelnen Komponenten des Pausenplanersystems können sich daher auf unterschiedlichen Arbeitsplatzrechnern der Gruppenmitglieder befinden.

Die Entwurfsmetapher Arbeitsumgebung beschreibt einen Einzelarbeitsplatz. Innerhalb ihrer technische Interpretation besteht die Umgebung aus einem Gebilde. Um die Verteilung der Gruppenarbeitsumgebung technisch zu realisieren, wurde dieses Umgebungsgebilde in eine Handhabungs- und eine Verwaltungskomponenten aufgeteilt.

Die **Handhabungskomponente (HK)** realisiert die Handhabung und Präsentation der Gruppenarbeitsumgebung des Pausenplanersystems. Sie stellt die Werkzeuge, Materialien und weitere Arbeitsgegenstände der Gruppenarbeitsumgebung dar und ermöglicht den Gruppenmitgliedern den Zugang zum System.

Die **Verwaltungskomponente (VK)** verwaltet den simultanen Materialzugriffs der Werkzeuge und versorgt diese mit Materialien. Zusätzlich werden die am Schwarzen Brett angemeldeten Gruppenmitglieder von ihr verwaltet.

Die Handhabungskomponente der Pausenplaner Gruppenarbeitsumgebung wurde von uns „Schwarzes Brett“ genannt. Im Anhang dieser Arbeit gibt es zur Schwarzes Brett HK eine Vision, welche die Oberfläche sowie die Handhabung durch den Benutzer detailliert beschreibt. Für jedes Gruppenmitglied, das sich im Pausenplanersystem angemeldet, wird ein Exemplar der Handhabungskomponente erzeugt.

Die Verwaltungskomponente existiert im gesamten System nur einmal und wird von den verwendeten Handhabungskomponenten beobachtet. Die Handhabungskomponenten benutzen die Verwaltungskomponente und können dadurch eine Zustandsveränderung auslösen. Die Verwaltungskomponente benachrichtigt daraufhin alle beobachtenden Handhabungskomponenten über diese Zustandsveränderung. Abbildung 5 zeigt die Beziehung zwischen Handhabungs- und Verwaltungskomponente der Gruppenarbeitsumgebung.

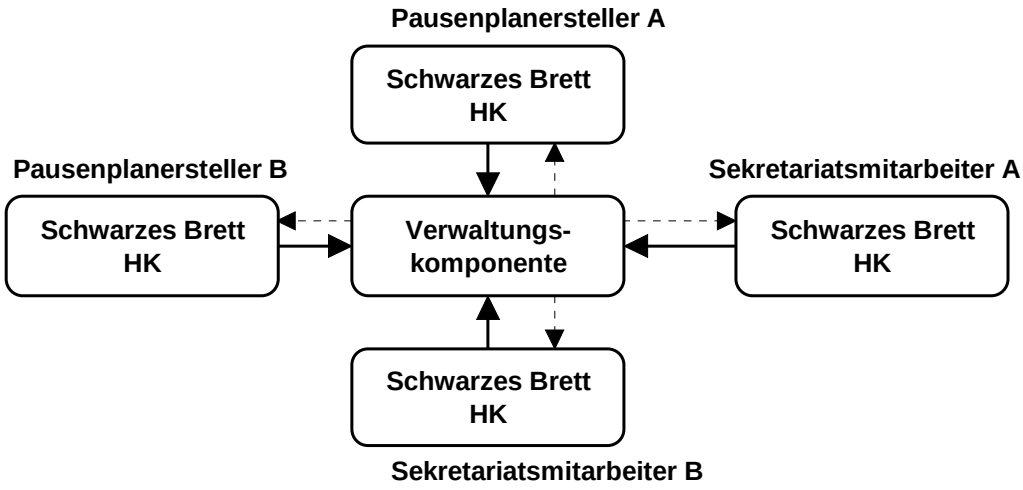


Abbildung 5: Mehrere Gruppenmitglieder arbeiten gemeinsam in der Gruppenarbeitsumgebung

Der hier vorgestellte komponentenbasierte Entwurf der Verwaltungskomponente besteht aus mehreren Subkomponenten, welche die oben beschriebenen Verwaltungsaufgaben umsetzen. Die folgende Liste beschreibt die einzelnen Subkomponenten der Verwaltungskomponente. Eine detaillierte Beschreibung der einzelnen Subkomponenten findet der Leser in der Arbeit von Andreas Hartmann [Har98].

- Gruppenarbeitsumgebung (Zentralkomponente, ZK)

Diese Zentralkomponente repräsentiert die Verwaltungskomponente, sie erzeugt und verwaltet die einzelnen Subkomponenten. Die Zentralkomponente ermöglicht der Handhabungskomponente und den Werkzeugen des Pausenplanersystems den Zugriff auf die einzelnen Subkomponenten.

- Benutzerverwalter

Der Benutzerverwalter ist für das An- und Abmelden der Benutzer zuständig. Er kann darüber Auskunft geben welche Benutzer am Schwarzen Brett angemeldet sind und welche Materialien von welchem Benutzer gerade bearbeitet werden. Die Werkzeuge stellen diese Information, bei simultaner Bearbeitung eines Materials, an ihrer Oberfläche dar.

- Materialverwalter

Beim Materialverwalter können die Materialien der Gruppenarbeitsumgebung angefragt werden. Das Material wird dabei durch den Namen oder den Materialtyp identifiziert. Der Materialverwalter wird den Werkzeugen auf den Arbeitsplatzrechnern der Gruppenmitglieder bekannt gemacht, diese beobachten den Materialverwalter, um bei Veränderung des Materials benachrichtigt zu werden. Für die Verwaltung der Materialien verwendet der Materialverwalter ein Materialmagazin. Der Materialverwalter ist zusätzlich für die Erzeugung neuer Materialien verantwortlich und übernimmt die „Entsorgung“ von nicht mehr benötigtem Material.

- Materialmagazin

Das Materialmagazin ist ein Behälter, der sämtliche Materialien der Gruppenarbeitsumgebung enthält. Diese werden bei der Initialisierung der Umgebung durch den Materialversorger in das Materialmagazin geladen. Die Materialien verbleiben während der gesamten Lebenszeit der Gruppenarbeitsumgebung im Materialmagazin. Werkzeuge auf den Arbeitsplatzrechnern der Gruppenmitglieder haben entfernten Zugriff auf die Materialien im Magazin. Ein simultaner Zugriff von mehreren Werkzeugen auf ein Material ist dabei möglich. Wird die Gruppenarbeitsumgebung geschlossen, so werden sämtliche Materialien durch den Materialversorger gesichert.

- Materialversorger

Ein Materialversorger ist für die Persistenz der Materialien verantwortlich. Für den Prototypen des Pausenplanersystems wurde nur ein Materialversorger implementiert. Dieser verwendet den Serialisierungsmechanismus des MFC Rahmenwerks, um sämtliche Materialien des Schwarzen Bretts persistent zu machen (siehe Abschnitt 4.4.3).

- Materialkonsistenzverwalter

Der Materialkonsistenzverwalter ist für die Konsistenz der Materialien des Pausenplanersystems verantwortlich. Das fachliche Wissen über die Abhängigkeiten der Materialien wird im Materialkonsistenzverwalter zentral verwaltet. Die Einhaltung der Materialkonsistenz kann dadurch an einer Stelle im System implementiert werden, ohne daß die Information über die Materialabhängigkeiten auf die verschiedenen Werkzeuge oder Materialien des Systems verteilt werden muß.

- Ereignisverwalter

Die Werkzeuge tragen beim Ereignisverwalter die Ereignisse ein, von deren Auslösung sie benachrichtigt werden sollen. Tritt das Ereignis ein, so benachrichtigt der Ereignisverwalter seine Beobachter.

Die Beziehungen zwischen den Subkomponenten der Verwaltungskomponente sind in Abbildung 6 dargestellt.

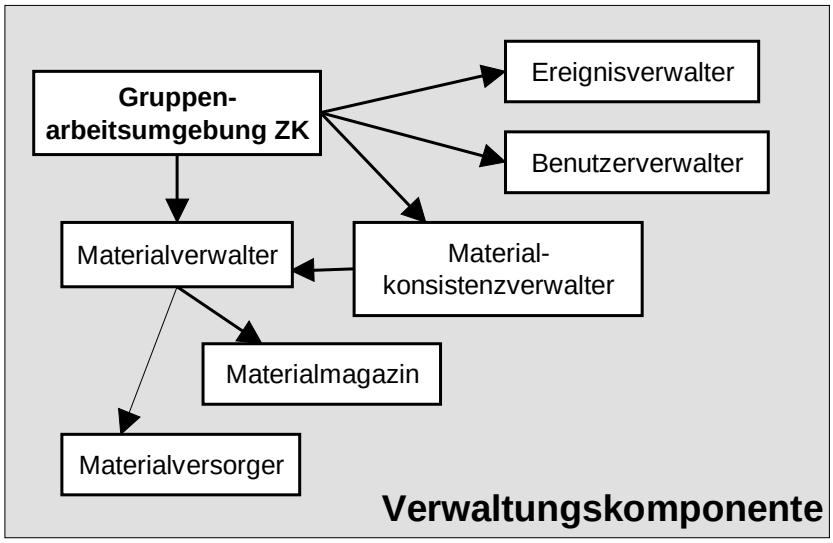


Abbildung 6: Subkomponenten der Verwaltungskomponente

Die Subkomponenten der Verwaltungskomponente können in externe und interne Komponenten gegliedert werden.

Externe Subkomponenten sind die Gruppenarbeitsumgebung, der Materialverwalter, der Ereignisverwalter sowie der Benutzerverwalter. Diese Subkomponenten können von der Handhabungskomponente und den Werkzeugen des Pausenplanersystems benutzt werden. Außerdem können sie die beobachtenden Komponenten über aufgetretene Ereignisse benachrichtigen.

Interne Subkomponenten sind das Materialmagazin, die Materialversorger und der Materialkonsistenzverwalter. Diese Subkomponenten sind von Außen nicht sichtbar. Sie dienen ausschließlich der internen Verarbeitung innerhalb der Verwaltungskomponente.

3.5 Die Werkzeuge des Pausenplanersystems

Dieser Abschnitt stellt die Werkzeuge des Pausenplanersystems vor und beschreibt ihre Beziehung zueinander. Die Konstruktion der Werkzeuge mit dem MFC Rahmenwerks wird in Kapitel 6 beschrieben. Im Pausenplanersystem gibt es vier Werkzeuge, die die Bearbeitung der Materialien ermöglichen. Zu jedem Werkzeug gibt es im Anhang eine Vision, die Oberfläche und Materialsicht des Werkzeugs darstellt und die Handhabung durch den Benutzer beschreibt. Die Visionen sind sehr detailliert, daher ist die Beschreibung der Werkzeuge in diesem Abschnitt stark komprimiert.

Die Werkzeuge des Pausenplanersystems sind:

- Pausenplaner

Das Pausenplaner Werkzeug wird vom Pausenplansteller verwendet, um die Lehrer zu den Pausenaufsichten einzuteilen. Im Rahmen dieser Tätigkeit belegt er die Pausenaufsichtsorte der Schule mit den Lehrer des Lehrkörpers. Das Werkzeug zeigt den verwendeten Lehrkörper und das bearbeitete Pausenplan-Material sowie Informationen über Belegungskonflikte und -statistiken.

- Aufsichtsorte-Bearbeiter

Mit dem Aufsichtsorte-Bearbeiter Werkzeug trägt der Pausenplansteller die Aufsichtsorte der Schule in den Pausenplan ein. Zu jedem Aufsichtsort kann er dabei die Anzahl der vorher eingestellten Pausenbeaufsichtiger und Vertreter festlegen.

- Pausenlisten-Bearbeiter

Mit dem Pausenlisten-Bearbeiter Werkzeug kann der Pausenplansteller die Pausen im Pausenplan der Schule festlegen. Er kann neue Pausen zum Pausenplan hinzufügen oder nicht mehr benötigte Pausen entfernen. Pausenzeitraum, Aufsichtsorte sowie Anzahl der Pausenaufsichten und Vertreter einer Pause können bestimmt werden.

- Lehrkörper-Bearbeiter

Mit dem Lehrkörper-Bearbeiter Werkzeug kann der Sekretariatsmitarbeiter den Lehrkörper der Schule bearbeiten. Neue Lehrer können in den Lehrkörper aufgenommen werden und von der Schule abgehende Lehrer können aus dem Lehrkörper entfernt werden. Name, Stelle und Ausschlußzeiträume eines Lehrers können mit dem Lehrkörper-Bearbeiter Werkzeug verändert werden.

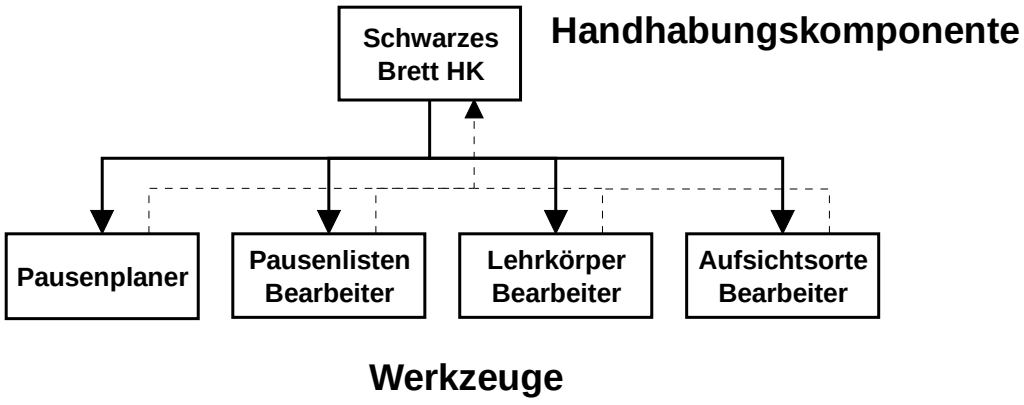


Abbildung 7: Werkzeuge des Pausenplanersystems

Die Werkzeuge des Pausenplanersystems werden für die Bearbeitung der angehefteten Materialien am Schwarzen Brett benutzt. Die Werkzeuge stehen unter der Verwaltung der Handhabungskomponente (siehe Abbildung 7).

Die Handhabungskomponente initialisiert die Verwendung der Werkzeuge. Sie muß dabei die Anwendung von Werkzeugen auf Materialien auf ihre Ausführbarkeit hin kontrollieren. In diesem Zusammenhang sorgt die Handhabungskomponente auch für die Erzeugung und Löschung der Werkzeuge.

Die Werkzeuge benachrichtigen die Handhabungskomponente über wichtige Ereignisse, die während der Arbeit durch den Benutzer ausgelöst werden können.

3.6 Die Materialien des Pausenplanersystems

Im Rahmen der Pausenplanung verwenden die Benutzer des Pausenplanersystems die beiden Materialtypen Pausenplan und Lehrkörper. Materialien dieses Typs können an das Schwarzen Brett angeheftet, abgenommen und verknüpft werden.

Bei der Bearbeitung der Materialien mit den am Schwarzen Brett befindlichen Werkzeugen wird man feststellen, daß sowohl das Pausenplan-Material als auch das Lehrkörper-Material aus mehreren Submaterialien bestehen. Der Pausenplan enthält Pausen und Aufsichtsorte. Der Lehrkörper setzt sich aus den Lehrern der Schule zusammen.

Der folgende Abschnitt stellt die Materialien des Pausenplanersystems vor und beschreibt Beziehungen zwischen den Materialien durch Veranschaulichung der Materialhierarchie (siehe Abbildung 8). Zu jedem der hier beschriebenen Materialien gibt es im Anhang eine Vision, die auf die Aufgaben und Bearbeitungsmöglichkeiten eines jeden Materials genauer eingeht. Die Materialien des Pausenplanersystems sind:

- Pausenplan

Der Pausenplan enthält die Pausen und Aufsichtsorte einer Schule und speichert die Einteilung der Pausenaufsichten. Um diese Einteilung vorzunehmen muß das Pausenplan-Material zunächst mit einem Lehrkörper verknüpft werden. Danach kann der Benutzer die Lehrer des verknüpften Lehrkörpers zu Pausenaufsichten im Pausenplan einteilen.

- Lehrkörper

Ein Lehrkörper repräsentiert das Lehrerpersonal einer Schule. Das Lehrkörper-Material ist daher als Behälter für eine Menge von Lehrern entworfen worden. Der Benutzer kann ein Lehrkörper-Material mit mehreren Pausenplänen verknüpfen, wodurch die mehrfache Verwendung eines Lehrkörpers möglich wird.

- Pausenliste

Eine Pausenliste enthält die Unterrichtspausen einer Schule. Das Pausenlisten-Material ist daher als Behälter für eine Menge von Pausen entworfen worden. Der Pausenplan verwendet den Behälter, um die Pausen besser verwalten zu können.

- Pause

Eine Pause ist ein wöchentlich wiederkehrender Zeitraum. Jede Pause hat Aufsichtsorte, die während dem Pausenzeitraum beaufsichtigt werden müssen. Zu jeder Pause kann man daher festlegen, welche Aufsichtsorte dies sind. Diese Zuordnung wird im Pausenaufsichtsort Material festgehalten.

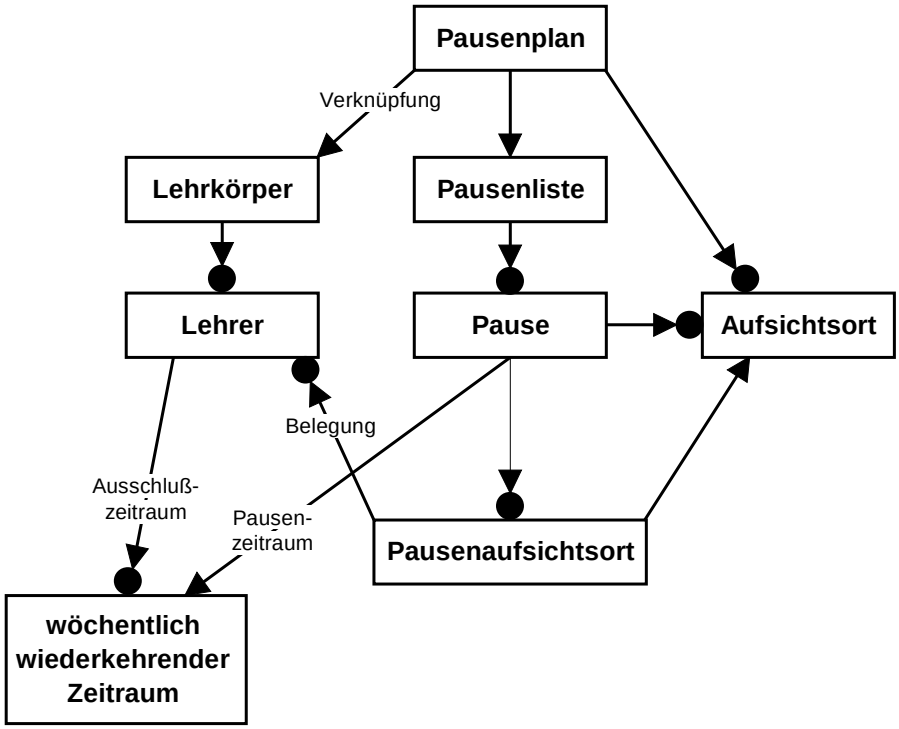


Abbildung 8: Materialien des Pausenplanersystems

- Lehrer

Dieses Material modelliert den Lehrer/die Lehrerin an einer Schule. Ein Lehrer hat einen Namen und eine Stelle, welche die Anzahl der zu leistenden Unterrichtsstunden bestimmt. Jeder Lehrer ist dazu verpflichtet eine gewisse Anzahl von Pausenaufsichten zu führen. Die Anzahl dieser Pausenaufsichten richtet sich nach der Stelle des Lehrers. Die Zeiten an denen ein Lehrer nicht an der Schule anwesend ist, werden in wöchentlich wiederkehrenden Zeiträumen (Ausschlußzeiträumen) festgehalten.

- Aufsichtsort

Ein Aufsichtsort ist ein örtlicher Bereich einer Schule, an dem sich die Schüler während der Pause aufhalten. Jeder Aufsichtsort hat einen Namen. Zusätzlich kann man die Mindestanzahl von Pausenaufsichten und Vertretern für den Aufsichtsort festlegen.

- Pausenaufsichtsort

Im Pausenaufsichtsort Material wird die Einteilung von Pausenaufsichten zu Pause und Aufsichtsort festgelegt. Im Krankheitsfall eines Lehrers muß eine Vertretung eingeteilt sein, die dann die Beaufsichtigung der Pause übernimmt. Für jeden Pausenaufsichtsort kann die Mindestanzahl von Pausenaufsichten und Vertretern individuell festgelegt werden.

- Wöchentlich wiederkehrender Zeitraum

Das Material wöchentlich wiederkehrender Zeitraum modelliert eine Zeitpanne zwischen zwei Zeitpunkten, die sich periodisch in jeder Woche wiederholt. Das Material kann eine Überschneidung mit einem anderen Zeitraum feststellen. Der Zeitraum wird bei der Erstellung des Materials festgelegt, kann jedoch nachträglich durch Werkzeuge verändert werden.

Beim Entwurf des Pausenplanersystems haben wir entschieden den wöchentlich wiederkehrender Zeitraum als Material und nicht als Fachwert zu modellieren. Fachwerte sind wie die Entwurfsmetapher Material Teil des WAM Methodenrahmens und werden verwendet um Werte der Anwendungswelt zu modellieren (siehe [BRS+98]). Bei dieser Entscheidung spielte die Tatsache der nachträglichen Veränderung des Zeitraums eine große Rolle.

3.7 Verteilung des Pausenplanersystems

Dieser Abschnitt beschreibt die technische Verteilung der Werkzeuge, der Materialien sowie der Gruppenarbeitsumgebung auf Prozeßräumen und Rechner.

Die Komponenten des Pausenplanersystems befinden sich in unterschiedlichen Prozeßräumen auf den Arbeitsplatzrechnern der Gruppenmitglieder oder auf den Serverrechnern der Schule. Im Pausenplanersystem kann man zwei Arten von Prozeßräumen unterscheiden:

- Lokaler Prozeßraum

Die Lokalen Prozeßräume enthalten die Handhabungskomponente (HK) der Gruppenarbeitsumgebung und die für die Bearbeitung der Materialien des Pausenplanersystems benötigten Werkzeuge. Die in Kapitel 6 beschriebene Werkzeugkonstruktion benötigt für jedes Werkzeug oder Subwerkzeug einen lokalen Prozeßraum, welcher sich auf dem Arbeitsplatzrechner eines Gruppenmitglieds befindet.

- Administrativer Prozeßraum

Der Administrative Prozeßraum enthält die Verwaltungskomponente (VK) der Gruppenarbeitsumgebung. Er existiert im gesamten Pausenplanersystem nur einmal und kann sich auf einem beliebigen Arbeitsplatzrechner oder einem Server der Schule befinden.

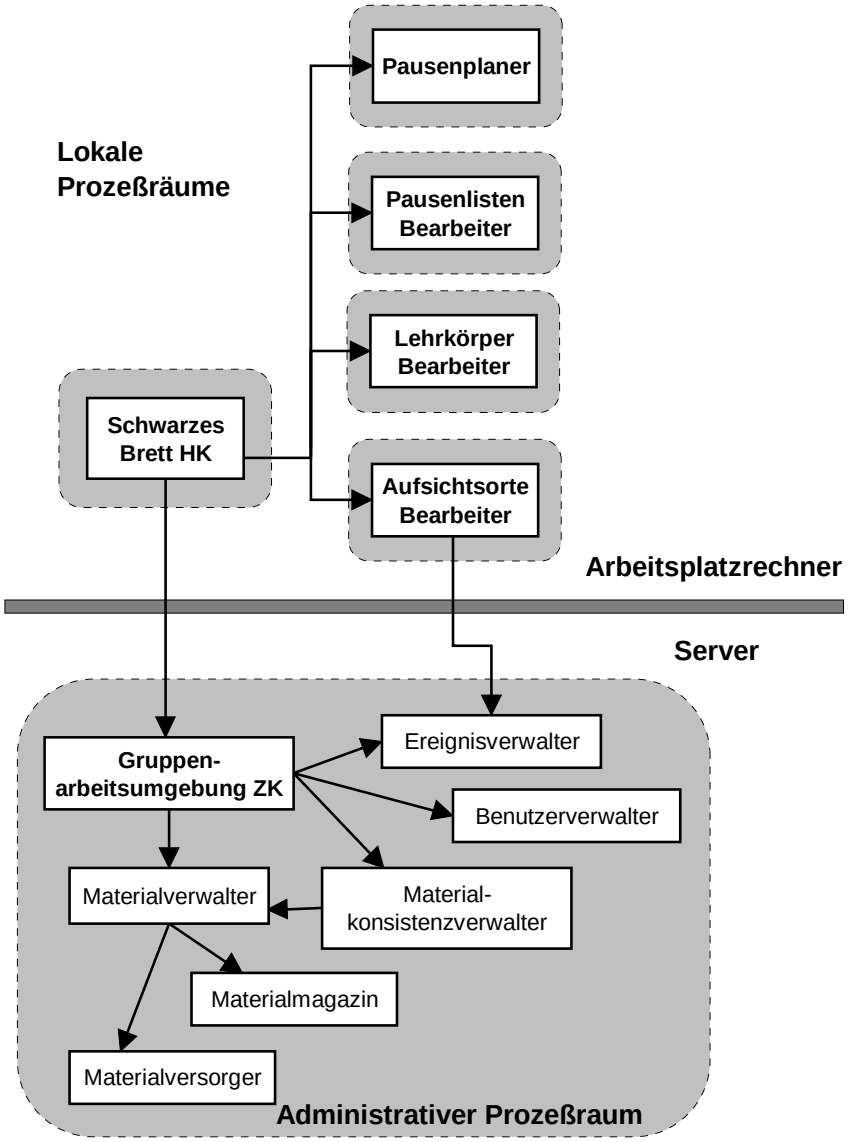


Abbildung 9: Verteilung des Pausenplanersystems

Abbildung 9 zeigt lokale Prozeßräume und den administrativen Prozeßraum der Pausenplanersystem Architektur. Komponenten aus unterschiedlichen Prozeßräumen kommunizieren miteinander, die Kommunikation findet dabei zwischen zwei lokalen Prozeßräumen oder zwischen einem lokalen und dem administrativem Prozeßraum statt.

In der Abbildung ist beispielhaft die Kommunikation zwischen Schwarzem Brett und Werkzeugen sowie zwischen Werkzeugen und Ereignisverwalter dargestellt. Aus Gründen der Übersichtlichkeit habe ich nicht alle Kommunikationswege der Komponenten des Pausenplanersystems dargestellt.

Abbildung 10 zeigt wie Pausenplanersteller und Sekretariatsmitarbeiter gemeinsam am Schwarzen Brett arbeiten. Die Abbildung zeigt einen möglichen Zustand des Pausenplanersystems. Sie soll deutlich machen in welchen Prozeßräumen sich Werkzeuge und Materialien der Gruppenmitglieder während der Bearbeitung befinden.

Die Werkzeuge auf den Arbeitsplatzrechnern der Gruppenmitglieder haben entfernten Zugriff auf die Materialien im Materialmagazin der Gruppenarbeitsumgebung. Diese verbleiben im Materialmagazin während der gesamten Lebensdauer der Gruppenarbeitsumgebung.

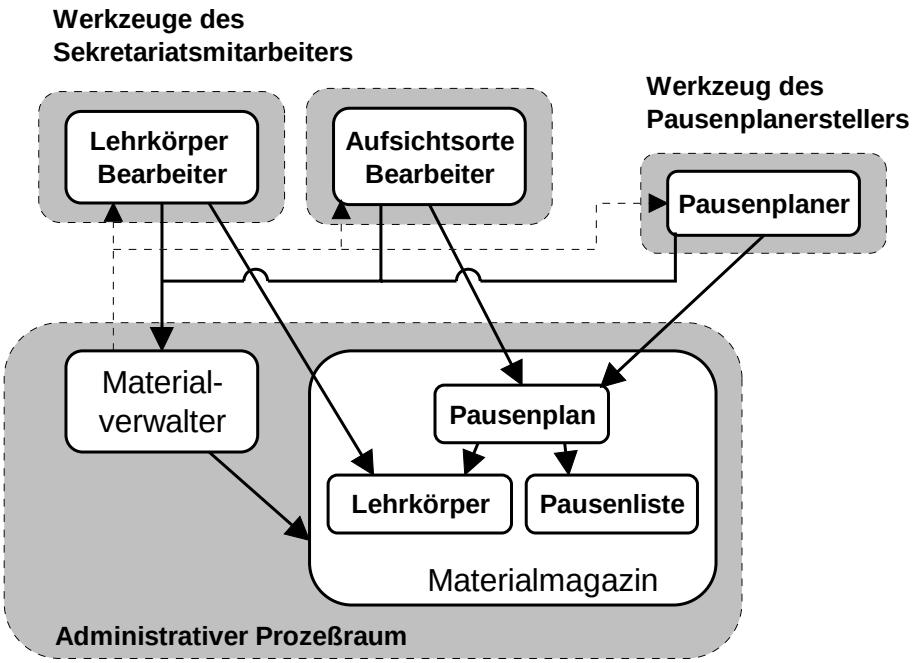


Abbildung 10: Pausenplanersteller und Sekretariatsmitarbeiter bearbeiten gemeinsam das Pausenplan Material

Abbildung 10 zeigt außerdem den simultanen Zugriff auf das Pausenplan Material. Dieser erfolgt vom Aufsichtsorte-Bearbeiter Werkzeug des Sekretariatsmitarbeiter sowie vom Pausenplaner Werkzeug des Pausenplanerstellers.

Sämtliche Werkzeuge beobachten den Materialverwalter der Gruppenarbeitsumgebung, verändert sich der Zustand des Pausenplans bei der Bearbeitung durch ein Werkzeug. So werden alle anderen Werkzeuge vom Materialverwalter über diese Zustandsänderung benachrichtigt. Daraufhin können sie den neuen Zustand des Pausenplans darstellen.

Für eine genaue Beschreibung der Konstruktion des gesamten Pausenplanersystems, muß ich den Leser auf spätere Kapitel verweisen. Kapitel 7 beschreibt detailliert die Werkzeugkonstruktion. Kapitel 8 geht auf die Konstruktion der Gruppenarbeitsumgebung und Materialien ein. In den nun folgenden Kapiteln versuche ich dem Leser die technischen Grundlagen für die Konstruktion des Pausenplanersystems zu vermitteln. Zu den technischen Grundlagen gehört das MFC Rahmenwerk, die Komponententechnologie COM und der Umgang mit den im Pausenplaner-Projekt verwendeten Entwicklungswerkzeugen.

4 Das MFC Rahmenwerk

Dieses Kapitel gibt dem Leser eine Einführung in das MFC Rahmenwerk, wobei die Entstehungsgeschichte wiedergegeben und die Entwurfsziele beschrieben werden. In den weiteren Abschnitten folgt eine Beschreibung der wichtigsten Klassen und Architekturen des Rahmenwerks.

Diese Einführung kann sicherlich nicht auf alle Aspekte des MFC Rahmenwerks im Detail eingehen. Daher möchte ich an dieser Stelle auf weitere Quellen hinweisen. Zum Kennenlernen des MFC Rahmenwerks kann man die Bücher [Hor97], [Cro97], [Sma96] und [Sch96b] empfehlen. Sie führen den MFC unerfahrenen Entwickler in die Verwendung des Rahmenwerks ein und zeigen, wie man mit Hilfe der MFC Anwendungen konstruieren kann.

Detailliertere Beschreibungen des MFC Rahmenwerks findet man in den Büchern [SW96], [Sch96], [Bla97], [Wil97] und [Dra98]. Sie beschreiben auch die Interna des Rahmenwerks und gehen dabei genauer auf die Implementation der Klassen ein. Das „MFC 5 Black Book“ von Al Williams [Wil97] beispielsweise vermittelt dem erfahrenen MFC Entwickler sehr detaillierte Kenntnisse über das Rahmenwerks, wodurch dieser in der Lage ist, die Möglichkeiten des Rahmenwerks voll auszuschöpfen. Das Werk von David Schmitt [Sch96] beschreibt darüberhinaus nützliche Erweiterungen des MFC Rahmenwerks.

Hervorheben möchte ich an dieser Stelle das Buch "MFC Internals" von G. Shepherd und S. Wingo [SW96], das mit Sicherheit aus der Menge der MFC Bücher herausragt, da es die Interna des MFC Rahmenwerks auf einfache und verständliche Art und Weise beschreibt.

Ein sehr umfangreiches Nachschlagewerk stellt die „Documentation Library“ der Visual C++ Entwicklungsumgebung dar. Diese ist in die Entwicklungswerkzeuge integriert und beinhaltet die Referenz des MFC Rahmenwerks [MFC97].

4.1 Entwurfsziele und Entwicklung

Im Jahr 1989 gründete Microsoft die "Application Framework Technology Development Group" kurz AFX[2] Group genannt. Die Mitarbeiter der Gruppe waren erfahrene Microsoft Entwickler, die aus den unterschiedlichsten Entwicklungsabteilungen zusammengeführt wurden. Ihr Ziel war es, die neuesten objektorientierten Technologien für die Entwicklung von Softwarewerkzeugen und Bibliotheken einzusetzen. Diese sollten später von Anwendungsentwicklern genutzt werden, um damit die modernsten GUI Anwendungen des Marktes zu entwikkeln.

Nach einem Jahr Entwicklungszeit hatte die AFX Gruppe den ersten Prototyp eines Rahmenwerks hergestellt. Wie man feststellen mußte, hatte dieser Prototyp recht wenig mit dem Windows Betriebssystem gemeinsam, da er ein komplett eigenes Fenstersystem, ein eigenes graphisches Subsystem, eine Objekt-Datenbank und sogar eine eigene Garbage Collection hatte.

Als die AFX Gruppenmitglieder anfingen Anwendungen mit dem Rahmenwerk zu entwickeln, stellte sich heraus, daß die Benutzung des Rahmenwerks zu kompliziert war und das sich die Verwendung des Prototypen zu weit von der bisherigen Art und Weise der Windows Programmierung entfernt hatte. Daher wurde beschlossen den ersten Prototypen zu verwerfen und die Ziele neu zu formulieren. Der Schwerpunkt der Entwicklung lag nun auf einem einfachen und eleganten Rahmenwerk, das einem Anwendungsentwickler die unkomplizierte Erstellung von Windows Anwendungen in der Programmiersprache C++ (siehe [Str92]) ermöglicht.

[2] Der Legende nach hat das "X" im Akronym keine Bedeutung. Es wurde vermutlich aus stilistischen Gründen hinzugefügt.

Die AFX Gruppe stellte fünf Entwurfsziele für die weitere Entwicklung des Rahmenwerks auf:

- Eine **praxisnahe Anwendungsentwicklung** soll möglich sein. Anstatt neue Programmierkonzepte zu definieren, soll das bestehende objektorientierte Programmiermodell von Windows mit Hilfe von gebräuchlichen C++ Sprachkonstrukten erweitert werden. Da C++ eine sehr komplexe Programmiersprache ist und nicht jeder Anwendungsentwickler damit umzugehen weiß, haben die AFX Gruppenmitglieder beschlossen, den Sprachumfang von C++ bei der Implementierung des MFC Rahmenwerks nicht vollständig auszuschöpfen. Beispielsweise nutzt das Rahmenwerk nicht die Mehrfachvererbung von Klassen. Der Anwendungsentwickler kann, wenn er möchte, in seinem Programmcode weiterhin alle Sprachkonstrukte von C++ nutzen.

- Das Rahmenwerk soll die **Erstellung von Windows Anwendungen** für erfahrene und weniger erfahrene Entwickler vereinfachen. In der Windows SDK[3] Dokumentation sind alle API[4] Funktion alphabetisch aufgeführt. Abhängigkeiten zwischen den einzelnen API Funktionen sind daher für den Entwickler nur schwer erkennbar. Die Verwendung der API Funktionen kann durch eine logische Gruppierung erleichtert werden. Die C++ Klassen des Rahmenwerks fassen Funktionsgruppen zusammen und kapseln die manchmal verwirrenden Details der Windows Programmierung. Die strenge Typisierung von C++ erhöht dabei zusätzlich die Sicherheit in der Entwicklung.

- Erlerntes **Wissen** über die Windows Programmierung kann weiter genutzt werden. Wo immer es möglich ist, sollen daher bestehende Eigenschaften von Windows verwendet werden. Um den Lernaufwand für erfahrene Windows Entwickler zu minimieren, wurden die Klassenhierarchie und die Namenskonventionen den zugrunde liegenden API Funktionen angeglichen. Alte C/SDK-Anwendungen können daher relativ leicht in eine MFC Anwendung konvertiert werden.

- Es wird ein **Fundament** für Windows Entwickler gegründet. Diese sollen mit dem Rahmenwerk umfangreiche Anwendungen erstellen, welche die neuesten Windows Erweiterungen, wie zum Beispiel OLE[5] und ODBC[6], verwenden. Mit jeder Erweiterung des Windows Betriebssystems wächst auch die Anzahl der Funktionen im Windows API. Der Entwurf des Rahmenwerks ermöglicht einen einfachen Ausbau des Rahmenwerks um neue Erweiterungen des Windows Betriebssystems.

- Das Rahmenwerk soll **klein und schnell** sein. Das ursprüngliche Ziel war, zusätzliche 20 KByte Speicher beim Programmcode in Kauf zu nehmen, wobei die neuen Anwendungen nicht langsamer ablaufen sollten als die alten C/SDK Anwendungen. Rahmenwerke mit hohem Abstraktionsgrad und vielen virtuellen Methoden neigen dazu, große und langsame Anwendungen zu erzeugen. Daher haben die AFX Gruppenmitglieder andere Mechanismen (siehe Abschnitt 4.4 und 4.6) entwickelt, um die Anwendungen kleiner und schneller zu machen.

Im Jahr 1992 brachte Microsoft das MFC Rahmenwerk Version 1.0 heraus. Es beinhaltete über 60 Klassen für die Entwicklung von Windows Anwendungen sowie allgemein nutzbare Klassen für Zeichenketten, Behälter, Dateien, Persistenz, Speichermanagement und Ausnahmebehandlung. Das Rahmenwerk fand große Beachtung und wurde von vielen Entwicklern in ihrer praktischen Arbeit eingesetzt. Die AFX Gruppenmitglieder erhielten viele Verbesserungsvorschläge für die erste Version des Rahmenwerks, die sie in der Version 2.0 umsetzten. In dieser Version hatte das Rahmenwerk bereits über 100 Klassen. In Tabelle 1 habe ich die weitere zeitliche Entwicklung des MFC Rahmenwerks zusammengefaßt. Die Version 4.0 beinhaltet über 240 Klassen. Anhand dieser Entwicklungszahl kann man einschätzen, wie das Rahmenwerk in seinem Umfang zugenommen hat.

[3] Software Development Kit. Eine Sammlung von Werkzeugen, Bibliotheken und Dokumenten zu einer Microsoft Technologie.
[4] Application Programming Interface. Eine Bezeichnung für Schnittstellen. APIs werden von Applikationen verwendet, um auf die Dienste des Betriebssystems zuzugreifen.
[5] Object Linking and Embedding: Microsofts objektbasierte Technologie, die eine gemeinsame Nutzung von Information und Diensten über Prozeß- und Maschinengrenzen hinweg ermöglicht. Bestandteile von OLE, welche keine Linking oder Embedding Funktionalität haben, werden in aktueller Dokumentation auch mit ActiveX betitelt.
[6] Open Database Connectivity: Eine standardisierte Schnittstelle für den Zugriff auf relationale Datenbanken.

Tabelle 1: Zeitliche Entwicklung des MFC Rahmenwerks

Jahr	MFC Version	Compiler Version	Neue Eigenschaften
1992	1.0	C/C++ 7.0	
1993	2.0	Visual C++ 1.0	Architektur für Anwendungsentwicklung, OLE 1.0 Unterstützung
1993	2.5	Visual C++ 1.5	OLE 2.0, ODBC
1994	3.0	Visual C++ 2.0	Thread Sicherheit
1995	3.1	Visual C++ 2.1	MAPI (Messaging Application Programming Interface), Socket Dienst
1995	4.0	Visual C++ 4.0	Win95 Unterstützung, Erweiterung der Entwicklungsumgebung
1997	4.21	Visual C++ 5.0	Internet Dienste, ActiveX

4.2 Verwendung des MFC Rahmenwerks

Das MFC Rahmenwerk ermöglicht die Entwicklung von Anwendungen. Ein solches Rahmenwerk wird in der englischen Literatur „Application-Framework“ genannte (siehe [Lew95]). Eine Definition für „Application-Framework“ liefert [Wei97]:

> „Ein Application-Framework realisiert die Infrastruktur einer vollständigen Anwendung und legt deren Kontrollfluß fest. Konkrete Anwendungen werden erstellt, indem deren spezifische Komponenten in das Framework integriert werden. ...“

Application-Frameworks definieren eine Systemumgebung für die Anwendungsentwicklung. Die unter Verwendung des Rahmenwerks erstellten Anwendungen sollten unabhängig vom Fenstersystem und, wenn möglich, auch unabhängig vom Betriebssystem sein.

Das MFC Rahmenwerk ermöglicht die Entwicklung von Anwendungen für die Betriebssysteme Windows 95, Windows NT, Windows 3.11 und MacOS. Es ist demnach möglich, eine Anwendung, die auf allen vier Betriebsystemen laufen soll, nur einmal zu erstellen. Für die Windows Betriebssystem ist dabei nicht einmal eine Neucompilation notwendig. Im Rahmen der Konstruktion des Pausenplanersystems wurde das Windows 95 Betriebssystem als Zielplattform eingesetzt. Die Verwendung des Pausenplanersystems auf den alternativen Plattformen wurden von uns nicht erprobt. Daher beziehen sich die dargestellten Ergebnisse auf den Einsatz innerhalb des Windows 95 Systems.

Das MFC Rahmenwerk wurde entworfen um dokumentenbasierte Anwendungen zu entwickeln. Tatsächlich wird die Entwicklung dokumentenbasierter Anwendungen vom Rahmenwerk und den Entwicklungswerkzeugen sehr gut unterstützt. Der Kern des Rahmenwerks, die in Kapitel 4.5 vorgestellte Document/View Architektur, legt dabei eine sehr allgemeine Infrastruktur fest. Dadurch kann das MFC Rahmenwerk innerhalb eines relativ breiten Anwendungsbereiches Verwendung finden.

Die Wiederverwendung funktioniert im MFC Rahmenwerk nach dem „White-Box“ Prinzip. Der Entwickler kann dabei anwendungsspezifisches Verhalten durch das Überschreiben von Methoden in abgeleiteten Klassen definieren. Da der Entwickler bei diesem Vorgang relativ viel Wissen über die interne Struktur des Rahmenwerks haben muß, nennt man diese Art der Wiederverwendung „White-Box“. Im Gegensatz dazu steht die „Black-Box“ Wiederverwendung, bei der der Entwickler die verwendeten Klassen des Rahmenwerks anwendungsspezifisch parametrisiert, ohne daß er dabei über ein weitreichendes Wissen der internen Struktur verfügen muß. Grundlagen zur Black- und White-Box Wiederverwendung können in [JF88] nachgelesen werden.

Die Frage nach dem Verwendungszweck des MFC Rahmenwerks könnte man mit einer recht kurzen Antwort zusammenfassen: „Das MFC Rahmenwerk ermöglicht die Erstellung *typischer* Windows Applikationen“. Was aber sind typische Windows Applikationen?

Im „Interface Application Design Guide“ [GUI94] wird das Aussehen und Verhalten (Look&Feel) von Windows Applikationen genau beschrieben. Ohne daß man auf die Einzelheiten des Design Guide eingehen muß, kann man alle Windows Applikationen in zwei Applikationstypen unterscheiden:

- **Single Document Interface** (SDI) Applikationen
- **Multiple Document Interface** (MDI) Applikationen

Single Document Interface (SDI) Applikationen können nur ein Dokument gleichzeitig bearbeiten. Will der Benutzer ein neues Dokument laden, um es zu bearbeiten, so muß er zunächst das alte Dokument schließen. Ein Beispiel für eine SDI Applikation ist der WordPad Editor des Windows Betriebssystems (siehe Abbildung 11).

Multiple Document Interface (MDI) Applikationen können mehrere Dokumente gleichzeitig bearbeiten. Jedes Dokument kann dann in einem eigenen Fenster dargestellt werden. Diese Bearbeitungsfenster befinden sich innerhalb des Hauptfensters der Applikation. MDI Applikationen können meist Dokumente verschiedenen Typs bearbeiten. Beispiele für MDI Applikationen sind Word (siehe Abbildung 12) oder Excel.

Abbildung 11: SDI Applikation WordPad

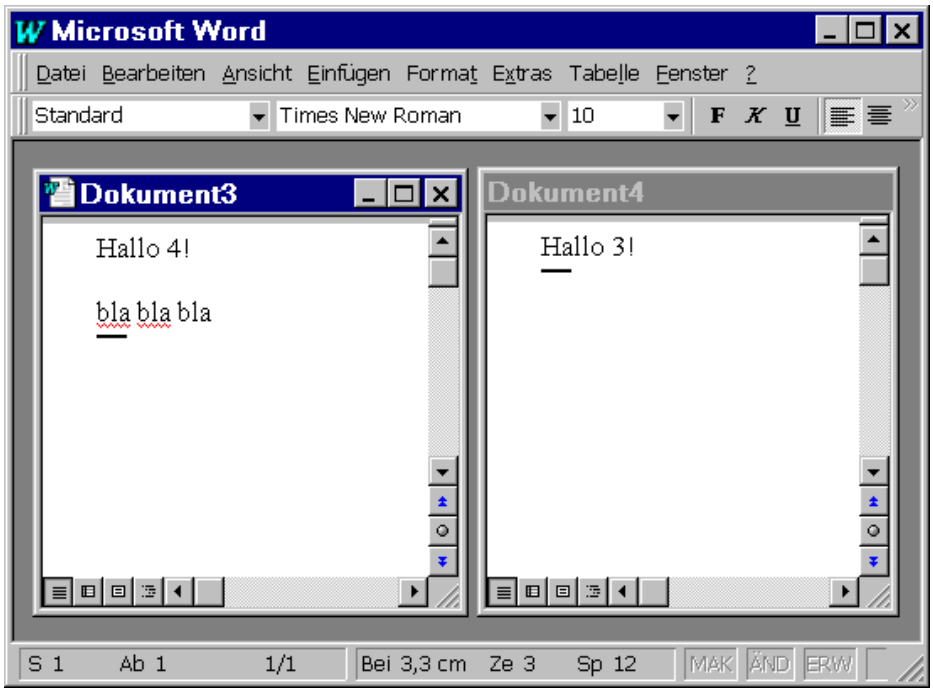


Abbildung 12: MDI Applikation Word

Wie man sehen kann, orientiert sich der SDI Applikationstyp am bearbeiteten Dokument. Die Verwendung dieses Typs eignet sich daher für kleinere Applikationen, die die Bearbeitung eines bestimmten Dokumenttyps in den Vordergrund stellen. Will der Benutzer mehrere Dokumente dieses Typs bearbeiten, so verwendet er parallel mehrere Exemplare der Applikation.

Beim MDI Applikationstyp orientiert sich das Geschehen um die Applikation selbst. Meist verwendet man den MDI Applikationstyp, wenn man viel Funktionalität innerhalb einer großen Applikation konzentrieren möchte. Die Applikation stellt dann den Ausgangspunkt für die Bearbeitung der unterschiedlichen Dokumente dar. Auch bei der parallelen Bearbeitung mehrerer Dokumente existiert immer nur ein Applikationsexemplar. Beide Applikationstypen werden durch das MFC Rahmenwerk unterstützt. Die Unterschiede der beiden Applikationstypen werden dabei innerhalb der Document/View Architektur abgebildet.

4.3 Klassen des MFC Rahmenwerks

Das MFC Rahmenwerk besteht aus einer Vielzahl von Klassen, die den unterschiedlichsten Zwecken dienen. Die Klassen des MFC Rahmenwerks decken weite Bereiche der Softwareentwicklung ab. In diesem Kapitel sollen nicht die einzelnen Klassen des Rahmenwerks im Detail beschrieben werden. Um dem Leser trotzdem einen geeigneten Überblick über die Struktur des MFC Rahmenwerks geben zu können, habe ich die Zuordnung einzelner Klassen in Subrahmenwerke vorgenommen. Dabei werden die Klassen nach ihrer semantischen Bedeutung den Rahmenwerken zugeordnet. Die Zuordnung der Klassen orientiert sich an der Klassifikation der MFC Referenz [MFC97]. Mehr über den Entwurf und die Struktur großer Rahmenwerke kann man in [BGK+97] nachlesen.

Die Klassenhierarchie des MFC Rahmenwerks auf Abbildung 13 zeigt die Vererbungsbeziehung zwischen wichtigen Basisklassen und Rahmenwerken. Die Notation der Rahmenwerke in der Abbildung verbirgt die enthaltenen Klassen und deren interne Struktur. Man sieht, daß fast alle Klassen des MFC Rahmenwerks über eine gemeinsame Wurzelklasse verfügen. Die Klassenhierarchie bildet damit einen Baum („single rooted“). Andere Rahmenwerke verfügen über mehrere Wurzeln, bilden mehrere Klassenbäume und werden daher auch Wald genannt. Die Vor- und Nachteile einer gemeinsamen Wurzelklasse werden in [Wei97] diskutiert.

Das MFC Rahmenwerk nutzt die **CObject** Wurzelklasse, um diverse Mechanismen in allen erbenden Klassen umzusetzen. Bei diesen Mechanismen handelt es sich um Laufzeitinformation, Persistenz, dynamische Objekterzeugung und Diagnosefunktionen. In Kapitel 4.4 werde ich genauer auf die Mechanismen und ihre Verwendung eingehen.

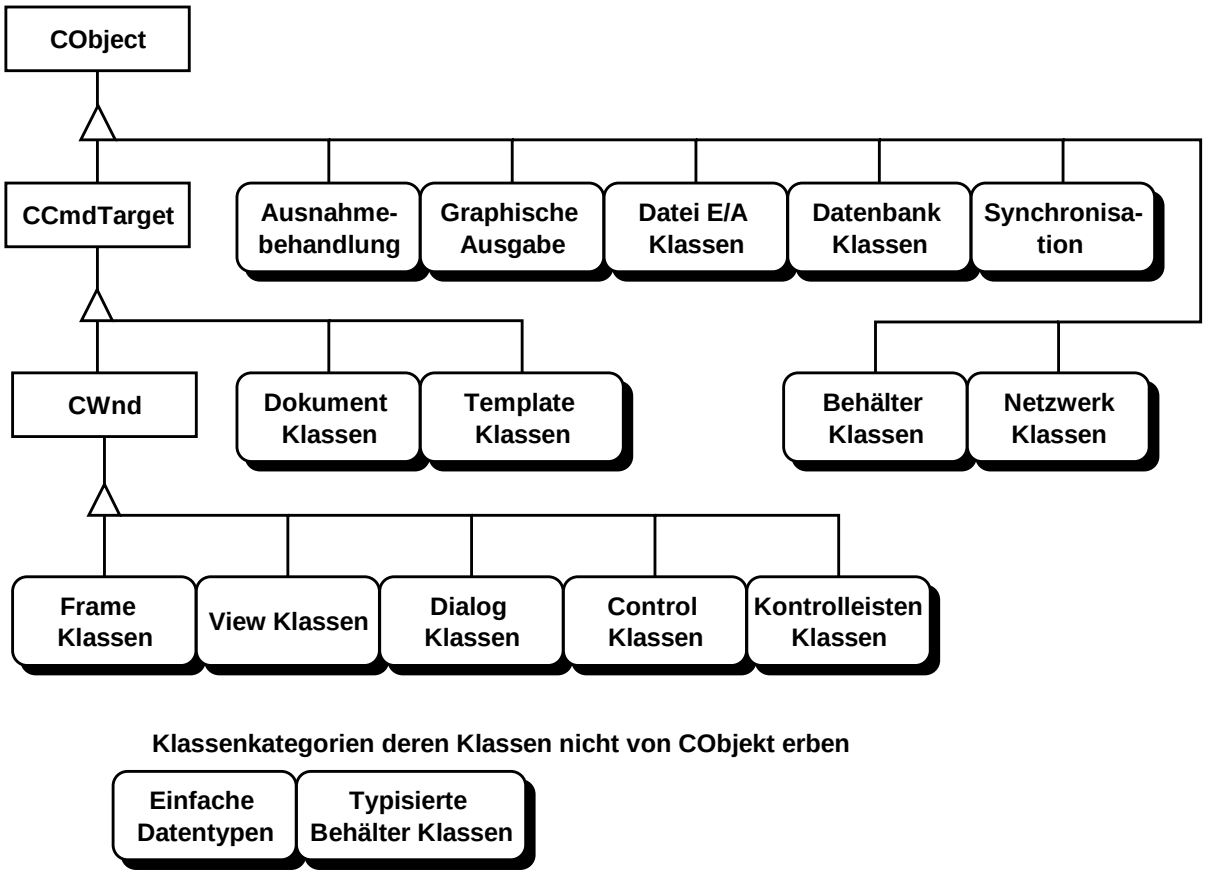


Abbildung 13: Die Subrahmenwerke des MFC Rahmenwerks

Die **CCmdTarget** Klasse ist die Basisklasse aller Nachrichten verarbeitenden Klassen. Objekte dieser Klasse können Nachrichten des Betriebssystems empfangen und verarbeiten (siehe Abschnitt 4.5). Zusätzlich ist es möglich, Objekte dieser Klasse entfernt zugreifbar zu machen. Der entfernte Zugriff auf ein Objekt wird im MFC Rahmenwerk über den COM Dienst „Automation“ realisiert. COM und Automation sowie die Integration in das MFC Rahmenwerk, werde ich in den folgenden Kapiteln beschreiben (siehe Kapitel 5).

Die **CWnd** Klasse ist die Basisklasse aller Fensterklassen des MFC Rahmenwerks. Sie kapselt große Teile des Window Fenstersystem. Objekte dieser Klasse können Nachrichten des Windows Fenstersystems empfangen und verarbeiten. Von der Klasse CWnd erben alle Klassen der View-, Frame-, Dialog-, Control- und Kontrolleisten-Rahmenwerke. Diese Klassen haben daher alle die Basisfunktionalität eines Fensters des Windows Fenstersystems.

Die folgenden Absätze charakterisieren die einzelnen Rahmenwerke. Jeder Absatz beschreibt dabei kurz den Verwendungszweck der Klassen, die sich innerhalb eines Rahmenwerks befinden.

- Klassen für Ausnahmebehandlung und Debugging

Dieses Rahmenwerk stellt Klassen für die Ausnahmebehandlung in verschiedensten Einsatzbereichen zur Verfügung. Gemeinsamkeiten aller im MFC Rahmenwerk existierender Ausnahmeklassen werden dabei in der Basisklasse CException zusammengefaßt. Zusätzlich umfaßt das Rahmenwerk Klassen für das Debugging und Testen von Anwendungen. Dazu gehören Klassen, die Ausgabetexte für Diagnosezwecke bereitstellen sowie Klassen, die Information über die Speichernutzung verwalten.

Zusammen ermöglichen die Klassen für Ausnahmebehandlung und Debugging die Konstruktion robuster und zuverlässiger Software.

- Klassen für graphische Ausgaben

In Windows Betriebssystemen werden graphische Ausgaben auf einer virtuellen Zeichenfläche dargestellt. Dadurch kann der Entwickler von der Ausgabe auf einem realen Ausgabegerät, wie Bildschirm oder Drucker, abstrahieren. Diese virtuelle Zeichenfläche nennt man auch Gerätekontext (engl. Device Context, DC). Das MFC Rahmenwerk stellt in diesem Rahmenwerk einige Kontextklassen (Paint, Window, MetaFile) sowie Ausgabeelementeklassen (Brush, Pen, Font) für die geräteunabhängige graphische Ausgabe zur Verfügung.

- Datei E/A und Datenbankklassen

Die Klassen dieser beiden Rahmenwerke ermöglichen die Persistenz von Objekten. Sie erlauben eine Speicherung und Wiederherstellung von Informationen innerhalb einer Datenbank oder Datei. Das MFC Rahmenwerk unterstützt zwei Datenbankschnittstellen, die DAO (Data Access Object) und ODBC (Open Database Connectivity) Schnittstellen. Beide Schnittstellen stellen die Funktionalität einer relationalen Datenbank bereit.

Die Datenbankanbindung wurde von den Entwicklern des MFC Rahmenwerk in die Document/View Architektur integriert.

- Klassen für Synchronisation

Klassen dieses Rahmenwerks ermöglichen die Synchronisation mehrerer Prozesse oder Threads. Unter anderem existieren Klassen für Events, Semaphore und den gegenseitigen Ausschluß.

- Behälterklassen

Im MFC Rahmenwerk gibt es drei Arten von Behältern: Arrays, Listen und Maps. Array-Behälter kann der Entwickler wie C++ Arrays verwenden, wobei die Speicherverwaltung von der Behälterklasse übernommen wird. MFC Listen-Behälter sind als doppelt verkettete Listen implementiert. Maps implementieren assoziative Behälter. Sie verwalten ihre Objekte anhand eines eindeutigen Schlüssels. Jede Behälterart wird im MFC Rahmenwerk als Template Klasse, als Template Klasse für typsichere Referenzen (Rahmenwerk „Typisierte Behälterklassen“) und als direkt verwendbare Klasse angeboten.

Die MFC Behälterklassen sich recht einfacher Natur und können mit den Klassen guter Behälterbibliotheken, wie z.B. der ConLib (siehe [Tra96]) oder der Standard Template Library der C++ Programmiersprache (siehe [MS96]) im direkten Vergleich nicht konkurrieren. Es gibt keine Queue und Stack Behälter und die MFC Behälterklassen kennen nicht das Konzept eines Cursors. Trotzdem wurden sie von uns bei der Konstruktion des Pausenplanersystems verwendet, da ihre Integration in das MFC Rahmenwerk Vorteile mit sich bringt. Die Behälterklassen implementieren beispielsweise den Serialisierungsmechanismus des Rahmenwerks, so daß ein kompletter Behälter samt Inhalt serialisiert werden kann. Voraussetzung dabei ist natürlich, daß die Objekte im Behälter auch serialisiert werden können.

- Netzwerkklassen

Die Klassen dieses Rahmenwerks ermöglichen den Austausch von Informationen über ein Netzwerk. Der Schwerpunkt liegt dabei auf der Verwendung des Internet/Intranet. Dabei ermöglichen die Klassen die Kommunikation über Sockets oder die Nutzung der Internet Protokolle Http, Ftp oder Gopher. Die Klassen dieses Rahmenwerks basieren auf dem Internet Service API oder auf dem Windows Socket API.

Das Netzwerk Rahmenwerk wurde mit der MFC Version 4.21 für Internet/Intranet Anwendungen sehr gut ausgebaut und mit Sicherheit eine der Stärken des MFC Rahmenwerks.

- Document/View Architekturklassen

Die Document-, Template-, View- und Frame-Rahmenwerke stellen die Komponenten der Document/View Architektur bereit. Sie bilden damit den Kern des MFC Rahmenwerks und stellen dem Entwickler eine Infrastruktur für den Entwurf seiner Anwendung zur Verfügung. Auf die Document/View Architektur werde ich in Kapitel 4.5 genauer eingehen.

- Einfache Datentypen

Die Klassen dieses Rahmenwerks ermöglichen eine einfache Handhabung strukturierter Werte (z.B. Punkt, Rechteck, Strecke, Zeitpunkt, Zeitspanne, Zeichenkette ...). Dabei handelt es sich um strukturierter Werte, die im MFC Rahmenwerk von vielen Klassen gemeinsam genutzt werden.

Oft handelt es sich bei diesen Klassen um eine Erweiterung von Strukturen des Windows API. Da in der Programmiersprache C++ Klassen und Strukturen gleich behandelt werden, sind diese kompatibel zueinander. Ein Großteil der Klassen dieses Rahmenwerks sind Adapterklassen für COM Automation Werte, dabei wird das Adapter Strukturierungsmuster aus [GHJ+96] implementiert.

4.4 Die Wurzelklasse CObject

Dieser Teil der Arbeit beschreibt die Möglichkeiten der CObject Wurzelklasse. Klassen, die von CObject erben, können Laufzeittypinformation, dynamische Objekterzeugung, Persistenz und Diagnosefunktionen anbieten. Ich beschreibe im folgenden, wie man diese Eigenschaften in eigenen Klassen einsetzen kann. Auf die Diagnosefunktionalität sowie weitere Implementierungsdetails werde ich dabei nicht eingehen und verweise dafür auf das Buch "MFC Internals" [SW96], in dem die technischen Hintergründe der Implementation ausführlich beschrieben werden.

4.4.1 Laufzeittypinformation

Objektorientierte Programmiersprachen unterstützen Inklusionspolymorphie, was bedeutet, daß ein Objekt zu mehr als einem Typ gehören kann. In statisch getypten Sprachen (z.B. C++, Eiffel) wird die Inklusionspolymorphie meist durch Vererbung eingeschränkt. Objekte eines Subtyps sind in diesen Programmiersprachen immer auch Objekte des Supertyps. Spricht man über den Typ eines Objekts, so verwendet man die Begriffe „statischer Typ“ und „dynamischer Typ“, um zwischen dem Typen der Deklaration und dem Laufzeittypen des Objekts zu unterscheiden. Der dynamische Typ ist dabei stets ein Subtyp des statischen Typs (siehe [Mey90]).

Die Verwendung von Inklusionspolymorphie macht es möglich, daß ein Objekt mit feststehendem statischen Typ während der Laufzeit zu unterschiedlichen dynamischen Typen gehören kann. Dabei tritt der dynamische Typ in den Hintergrund, da das Objekt über die Schnittstelle des statischen Typs angesprochen wird. Manchmal ist es in der objektorientierten Programmierung jedoch notwendig, daß man den dynamischen Typ eines Objekts kennt.

Um in C++ die Möglichkeit zu haben, den dynamischen Typ eines Objekts zur erfahren, wurde die Programmiersprache durch den standardisierten RTTI[7] Mechanismus erweitert. Diese Erweiterung erfolgte jedoch erst nach der Entwicklung des MFC Rahmenwerks. Daher waren die MFC Entwickler zum damaligen Entwicklungsstand

[7] Run-Time Typ Identification (siehe [Eck96]) wurde mit Visual C++ Version 4.0 in den Microsoft C++ Compiler integriert.

gezwungen, einen eigenen Mechanismus zu entwerfen. Der im MFC Rahmenwerk verwendete Mechanismus wird RTCI[8] genannt.

Von CObject abgeleitete Klassen, die Laufzeittypinformationen bereitstellen, speichern diese innerhalb der CRuntimeClass Struktur. In dieser Struktur sind beispielsweise die Namen der Klasse und der Basisklasse, sowie die Speichergröße erzeugter Objekte enthalten. Obwohl der Name „CRuntimeClass“ eine Klasse vortäuscht, handelt es sich bei CRuntimeClass um eine C++ Struktur. C++ Strukturen verhalten sich wie C++ Klassen, deren sämtliche Variablen und Methoden öffentlich zugänglich (public) sind. Die CRuntimeClass Struktur wird als statische Variable in der entsprechenden Klasse deklariert, so daß für jede Klasse nur ein Exemplar dieser Struktur im Speicher vorhanden ist.

Das folgende Beispiel zeigt, wie man einen polymorphen Cast zu einer abgeleiteten Klasse durch Aufruf der Methode *CObject::IsKindOf(...)* typsicher machen kann. Im Beispiel ist die Klasse CDynamicClass von CObject abgeleitet und kann Laufzeittypinformationen bereitstellen.

```
CObject* pObject = new CDynamicClass;

if ( pObject -> IsKindOf( RUNTIME_CLASS( CDynamicClass )))
        CDynamicClass* pMyObject = (CDynamicClass* ) pObject;
```

Die *IsKindOf(...)* Methode bekommt als Parameter eine Referenz auf eine CRuntimeClass Struktur einer Klasse. Das RUNTIME_CLASS Makro liefert diese Referenz zurück. Der Aufruf der *IsKindOf(...)* Methode klärt, ob der dynamische Typ des Objekts ein Subtyp der als Parameter übergebenen Klasse ist.

4.4.2 Dynamische Erzeugung

In diesem Abschnitt wird beschrieben, wie die dynamische Erzeugung von Objekten realisiert werden kann. Es kommt vor, daß man die Erzeugung von Objekten einer Klasse auf einen späteren Zeitpunkt verschieben möchte. Zu diesem Zweck verwendet man die dynamische Erzeugung von Objekten, die in der Literatur auch „Späte Erzeugung“ genannt wird. In [BR96] wird die „späte Erzeugung“ in Form eines Entwurfsmusters beschrieben und ihre Verwendung wird motiviert.

Der Mechanismus der dynamischen Erzeugung im MFC Rahmenwerk verwendet die CRuntimeClass Struktur. Diese besitzt die Methode *CreateObject()*, mit der Objekte einer vorher hinterlegten Klasse erzeugt werden können. Die CRuntimeClass Struktur wird mit dem RUNTIME_CLASS Makro initialisiert.

Ein Beispiel soll dies verdeutlichen: Die CDyncreateClass Klasse ist von CObject abgeleitet und kann dynamisch erzeugt werden.

```
CRuntimeClass* pRuntimeClass = RUNTIME_CLASS( CDyncreateClass );
CObject* pObject = pRuntimeClass -> CreateObject();
ASSERT( pObject -> IsKindOf( RUNTIME_CLASS( CDyncreateClass )))
```

Das MFC Rahmenwerk nutzt häufig diese Möglichkeit der dynamischen Erzeugung. Ein Beispiel aus dem Bereich der Document/View Architektur ist die Erzeugung einer Template-Komponente.

```
pDocTemplate = new CSingleDocTemplate( IDR_MAINFRAME,      //Ressourcen
      RUNTIME_CLASS(CLehrkoerperBearbeiterDoc),            // Document
      RUNTIME_CLASS(CMainFrame),                           // Frame
      RUNTIME_CLASS(CLehrkoerperBearbeiterView));          // View
```

Dem Konstruktor der Template-Komponente wird die Klasseninformation der zu verwendenden View-, Frame- und Document-Objekte innerhalb der CRuntimeClass Strukturen übergeben. Dadurch können die beteiligten Objekte zu einem späteren Zeitpunkt innerhalb der Template-Komponente erzeugt und verknüpft werden. Die Template-Komponente kennt dabei jedoch nur die Oberklasse von View-, Frame- und Document-Objekt. Der dynamische Typ muß nicht bekannt sein.
Details über die Verwendung der Template-Komponente und die Document/View Architektur beschreibe ich im Kapitel 4.5.

[8] Run-Time Class Information

4.4.3 Persistente Objekte

Im folgenden Abschnitt wird beschrieben, wie man persistente Objekte entwirft. Persistenz ist eine Eigenschaft, die es ermöglicht, den Zustand eines Objekts dauerhaft zu speichern und ihn zu einem späteren Zeitpunkt wiederherstellen zu können. Dieser **Serialisierungsmechanismus** wird im MFC Rahmenwerk „Serialization“ genannt (siehe [Oni96a] und [Oni96c]).

Um eine Klasse persistenzfähig zu machen, muß man die Klasse von CObject ableiten und die virtuelle Methode *CObject::Serialize(...)* innerhalb der Klasse überschreiben. Die *Serialize(...)* Methode beschreibt, wie der Zustand des Objekts gespeichert und wiederhergestellt werden kann:

```
void CSerialClass: Serialize( CArchive& ar)
{
        if ( ar.IsStoring() )
        {
              //Zustand speichern
              ar << m_IntegerVariable;
              ar << m_RealVariable;
              m_ExemplarVariable.Serialize( ar );
        }
        else
        {
              //Zustand wiederherstellen
              ar >> m_IntegerVariable;
              ar >> m_RealVariable;
              m_ExemplarVariable.Serialize( ar );
        }
}
```

Das Beispiel zeigt, wie die Methode *Serialize(...)* implementiert werden muß, damit sie den Zustand des CSerialClass Objekts einfrieren kann. Die Implementation von Speichern und Wiederherstellen der Exemplarvariablen wird dabei in den Klassen CArchive und CObject gekapselt. Variablen einfachen Typs werden mit Hilfe des Einfügeoperator (<<) gespeichert und mit Hilfe des Einleseoperator (>>) wiederhergestellt. Bei Exemplarvariablen wird einfach die *Serialize(...)* Methode mit dem Archiv als Parameter aufgerufen. Auf diese Weise können große zusammenhängende Objektstrukturen in einen Datenstrom geschrieben werden, ohne daß man sich um ein spezielles Datenformat kümmern muß.

Der Serialisierungsmechanismus des MFC Rahmenwerks muß nicht ausschließlich für die Persistenz von Objekten Verwendung finden. Er eignet sich allgemein für das Serialisieren von komplexen Objektstrukturen. Mit dem Serialisierungsmechanismus können beispielsweise Objekte über ein Netzwerk versendet werden. Dazu muß der Konstruktor der CArchive Klasse mit einer entsprechenden Unterklasse von CFile aufgerufen werden.

4.4.4 Technische Umsetzung

Für die Umsetzung der in den obigen Abschnitten beschriebenen Eigenschaften verwendet das MFC Rahmenwerk Paare von „DECLARE...“ und „IMPLEMENT...“ Makros. Die DECLARE Makros deklarieren Variablen und Methoden einer Klasse, die sich in der Deklarations Datei (.H) der Klasse befinden. Die IMPLEMENT Makros implementieren die Methoden und befinden sich in der Implementationsdatei (.CPP) der Klasse. Tabelle 2 zeigt, welche Eigenschaften die jeweiligen Makropaare implementieren.

Tabelle 2: Makros zur Umsetzung von Eigenschaften

DECLARE_DYNAMIC IMPLEMENT_DYNAMIC	Objekt stellt Laufzeitinformation über seinen Typ zur Verfügung.
DECLARE_DYNCREATE IMPLEMENT_DYNCREATE	Zusätzlich zu Laufzeitinformation: dynamische Erzeugung von Objekten.
DECLARE_SERIAL IMPLEMENT_SERIAL	Zusätzlich zu dynamischer Erzeugung: Persistenz von Objekten.

Die Makropaare und damit die Funktionalität der einzelnen Stufen bauen dabei aufeinander auf. Beispielsweise setzten die „...DYNCREATE“ Makros dynamische Erzeugung und Laufzeitinformation um. Würde man zusätzlich das „...DYNAMIC“ Makropaar im Quelltext der Klasse einfügen, so würde der Compiler Fehler melden.

4.4.5 Zusammenfassung

Innerhalb der Wurzelklasse CObject sind einige recht nützliche Objektmechanismen von den Entwicklern des MFC Rahmenwerks implementiert worden. Diese Basisfunktionalität wird von fast allen Klassen des Rahmenwerks genutzt und kann auch vom Anwendungsentwickler für eigene Klassen verwendet werden.

In dieser Eigenschaft ist die CObject Klasse des MFC Rahmenwerks anderen Wurzelklassen alternativer Rahmenwerke und Klassenbibliotheken recht ähnlich. Erwähnt sei hier die „Object“ Klasse von Smalltalk (siehe [Gol83]) und die Wurzelklasse des ET++ Rahmenwerks (siehe [Lew95]) gleichen Namens. Auch hier wird die Wurzelklasse verwendet, um Mechanismen für ein komplettes Rahmenwerk bereitzustellen, die in der Programmiersprache fehlen. Sowohl Smalltalk als auch ET++ integrieren zusätzlich zu den oben beschriebenen Mechanismen der CObject Klasse einen Beobachtermechanismus in ihre Wurzelklassen.

Mit dem RTCI Mechanismus für die Bereitstellung von Laufzeitinformationen stellt das MFC Rahmenwerk dem Anwendungsentwickler eine geeignete Methode zur Verfügung den dynamischen Typ eines Objekts zur Laufzeit zu erfahren. Der RTTI Mechanismus der Programmiersprache C++ leistet gleiches, ist jedoch in die verwendete Sprache integriert. Diese Integration der Laufzeitinformationen in die Programmiersprache ist mit Sicherheit als Vorteil zu werten.

Bei der Verwendung des RTTI Mechanismus im Kontext des MFC Rahmenwerks sollte der Anwendungsentwickler beachten, daß sowohl die dynamische Erzeugung als auch der Persistenz Mechanismus in ihrer Implementierung den RTCI Mechanismus verwenden. Nachteilig könnte sich evtl. auch die Abhängigkeit des RTTI Mechanismus vom verwendeten C++ Compiler auswirken.

Mit dem Mechanismus der dynamischen Erzeugung hat der Anwendungsentwickler ein gutes Instrument in der Hand, um die Erzeugung von Objekten zu kontrollieren. Im MFC Rahmenwerk sorgt die Verwendung von dynamischer Erzeugung für eine starke Vereinfachung der Erzeugungsprozesse.

Der Persistenz von Objekten kommt mit der Verbreitung Objektorientierter Programmiersprachen eine immer größere Bedeutung zu. Mit dem Serialisierungsmechanismus besitzt das MFC Rahmenwerk eine gut funktionierende Methode den Zustand eines Objekts dauerhaft zu speichern und ihn zu einem späteren Zeitpunkt wiederherstellen zu können. Der Mechanismus wurde konsequent im Rahmenwerk umgesetzt.

Die Implementierung der Mechanismen innerhalb der CObject Klasse ist durch die Verwendung von Strukturen und Makros leichtgewichtig und verursacht keinen großen Overhead. Dies ist für die performante Verwendung des MFC Rahmenwerks in der Anwendungsentwicklung von großer Bedeutung, da die Mechanismen innerhalb des Rahmenwerks sehr häufig verwendet werden.

4.5 Die Document/View Architektur

Die Document/View Architektur ist das Herzstück des MFC Rahmenwerks und ist deshalb für diese Arbeit von großer Bedeutung. Mit der Document/View Architektur stellt das MFC Rahmenwerk dem Entwickler eine Infrastruktur für seine Applikation zur Verfügung, deren Hauptaufgabe die Trennung von Interaktion und Funktion ist. In diesem Kapitel werde ich genauer auf die Verwendung der einzelnen Architektur-Komponenten und deren Zusammenwirken eingehen. Der Leser kann weitergehende Details in [For96a] und [For96b] nachlesen.

Neben der Trennung von Interaktion und Funktion realisiert die Document/View Architektur weitere Bestandteile der Applikation, wie zum Beispiel Datenpersistenz, Einbettung von externen Dokumenten, sowie graphische Bildschirm- oder Druckausgabe. Der Entwickler kann bestimmen, welche Bestandteile er von der Document/View Architektur verwenden möchte. Die restlichen Elemente seiner Applikation kann er dann alternativ nach seinen Vorstellungen entwickeln. Die an der Document/View Architektur beteiligten Komponenten sind:

- Die **Document-Komponente**, verwaltet die Daten der Applikation und benachrichtigt die View-Komponente(n) über Zustandsänderungen.
- Die **View-Komponente** übernimmt die Interaktion mit dem Benutzer, d.h. sie kümmert sich um die Darstellung und nimmt die Benutzereingaben entgegen. Die View-Komponente greift ausschließlich über die Schnittstelle der Document-Komponente auf die Daten der Applikation zu. Dabei können mehrere View-Komponenten zu einer Document-Komponente existieren.
- Die **Frame-Komponente** erzeugt die View-Komponente(n), rahmt sie ein und verwaltet Menüs, Symbol- und Statusleisten.
- Die **Template-Komponente** faßt Document, View und Frame zu einer Einheit zusammen. Sie erzeugt Document und Frame. Für jeden Dokumenttypen der Applikation gibt es eine Template-Komponente.
- Die **Applikation-Komponente** schließlich kapselt den Prozeß der Applikation und übernimmt deren Initialisierung. Sie verwaltet und erzeugt die Template-Komponente(n) der Applikation.

Abbildung 14 zeigt, wie die Komponenten der Document/View Architektur zusammenwirken.

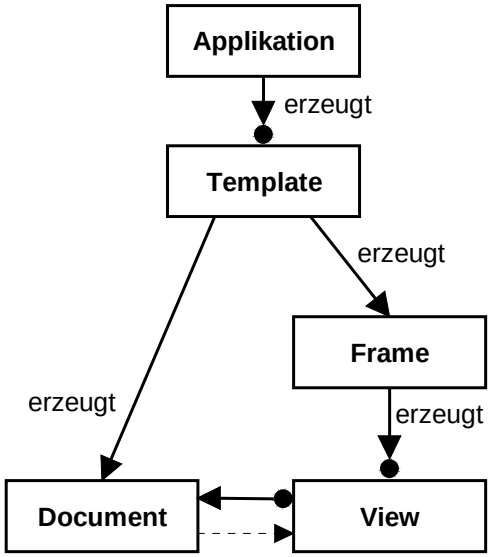


Abbildung 14: Die Document/View Architektur

Ich werde nun die Komponenten der Document/View Architektur im einzelnen näher beschreiben.

4.5.1 Document-Komponente

Die Document-Komponente verwaltet die Daten der Applikation. Über ihre Schnittstelle haben die Views lesenden und schreibenden Zugriff auf diese Daten. Sie benachrichtigt die Views über Zustandsveränderungen der Daten und sorgt so für Konsistenz. Weiterhin übernimmt die Document-Komponente das Laden und Speichern der Daten.

Im MFC Rahmenwerk wird die Document-Komponente durch die Basisklasse CDocument repräsentiert (siehe Abbildung 15). Möchte man eine eigene Document-Komponente erstellen, so erbt man von CDocument oder von einer ihrer abgeleiteten Klassen. Die von CDocument abgeleitete Klasse bewirkt noch nicht viel. Sie sollte Klassen benutzen, in denen die Daten verwaltet werden können, und Methoden, die die anderen Komponenten aufrufen, um auf diese Daten zugreifen zu können.

Die in Abbildung 15 dargestellte MFC Vererbungshierarchie zeigt, daß die CDocument Klasse von CCmdTarget abgeleitet wird. CCmdTarget wiederum wird von der Wurzelklasse CObject abgeleitet. Da die CDocument Klasse von CObject erbt, kann sie die von CObject Mechanismen wie Laufzeittypinformationen, dynamische Objekterzeugung, Persistenz und Diagnosefunktionen nutzen. Durch das Erben von CCmdTarget können die Objekte der CDocument Klasse versendete Nachrichten empfangen sowie Automation implementieren.

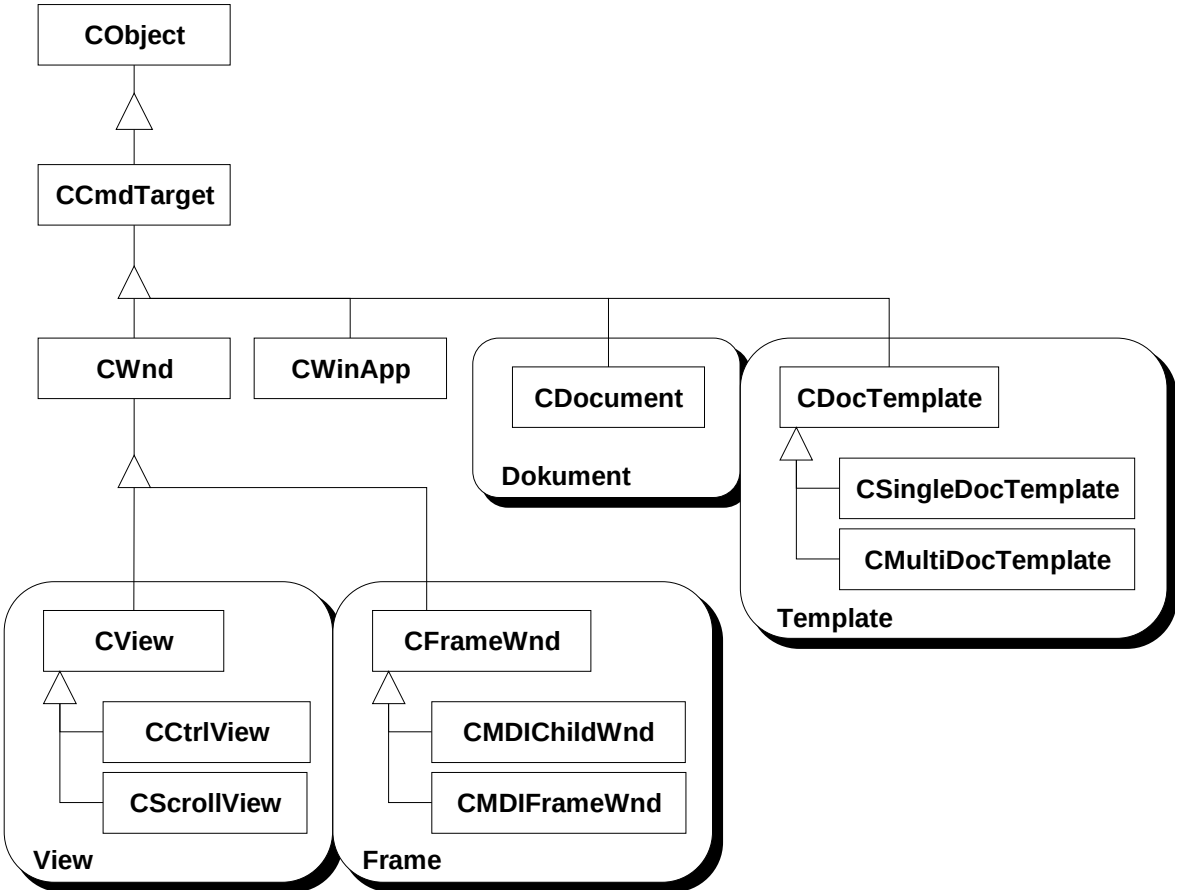


Abbildung 15: Die Klassen der Document/View Architektur im MFC Rahmenwerks

4.5.2 View-Komponente

Die View-Komponente übernimmt die Interaktion mit dem Benutzer, das heißt sie sorgt für die Darstellung der Daten und nimmt die Eingaben des Benutzers entgegen. Zu einer Document-Komponente kann es mehrere View-Komponenten geben, welche sämtliche Daten des Dokuments oder eine Untermenge darstellen. Die View-Komponente hat ausschließlich über die Schnittstelle der Document-Komponente Zugriff auf die Daten. Erhält sie eine Benachrichtigung über eine Zustandsveränderung der Daten, so ist sie für die Aktualisierung der Darstellung verantwortlich.

Im MFC Rahmenwerk wird eine View-Komponente durch die Basisklasse CView repräsentiert (siehe Abbildung 15). Möchte man eine eigene View-Komponente erstellen, so erbt man von CView oder von einer ihrer abgeleiteten Klassen. Die CView Klasse wird von CWnd abgeleitet und diese wiederum von CCmdTarget. Durch das Erben von CWnd kann jede View-Komponente zusätzlich Nachrichten des Windows Fenstersystems empfangen und verarbeiten.

4.5.3 Frame-Komponente

Die Frame-Komponente stellt einen Rahmen zur Verfügung, der eine oder mehrere View-Komponenten einrahmt. Sie erzeugt und verwaltet diese Views. Die Frame-Komponente enthält zusätzlich zum eigentlichen Rahmen Interaktionstypen. Dazu gehören Menüs, Symbolleisten und Statusleisten. Es werden demnach nicht alle Benutzerinteraktionen innerhalb der View-Komponente verarbeitet. Die MFC Entwickler argumentieren dabei mit verbesserten Möglichkeiten für die Wiederverwendung von Komponenten. In Abbildung 16 kann man sehen, wo sich die Frame, View und die zusätzlichen Interaktionstypen einer SDI Applikation befinden. Als Beispiel ist die Oberfläche der WordPad Editors abgebildet.

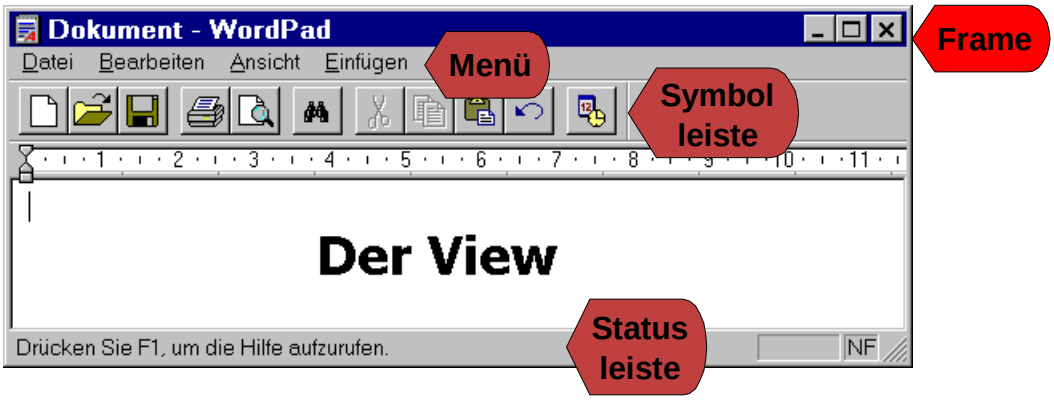


Abbildung 16: Frame- und View-Komponente der WordPad Applikation

Die beiden unterschiedlichen Applikationstypen SDI und MDI wirken sich auf die Umsetzung von Template und Frame-Komponenten aus. SDI Applikationen verwenden eine von CFrameWnd abgeleitete Klasse, um die View-Komponente zu umrahmen. MDI Applikationen verwenden dazu eine von CMDIChildWnd abgeleitete Klasse. Für das Hauptfenster verwenden MDI Applikationen die CMDIFrameWnd Klasse. Diese stellt einen äußeren Rahmen für die einzelnen MDI Child-Fenster zur Verfügung.

4.5.4 Template-Komponente

Die Template-Komponente fügt die Document-, View- und Frame-Komponente zu einer Einheit zusammen. Sie erzeugt und verwaltet die ihr zugeordneten Komponenten. Die jeweiligen Objekte der Komponenten werden über den Mechanismus der dynamischen Erzeugung (siehe Kapitel 4.4.2) zum geeigneten Zeitpunkt erzeugt und geeignet miteinander verknüpft. Eine Applikation besitzt für jeden Dokumenttypen, den sie bearbeiten kann, ein Template, das die Document-, View- und Frame-Komponenten dieses Typs enthält.

Im MFC Rahmenwerk wird die Template-Komponente durch die abstrakte Basisklasse CDocTemplate repräsentiert. Von CDocTemplate erben die beiden vorgefertigten Template Klassen CSingleDocTemplate und CMultiDocTemplate, die die SDI und MDI Applikationstypen realisieren.

4.5.5 Applikation-Komponente

Es gibt in jeder MFC Applikation ein CWinApp Objekt, das die Initialisierung der Anwendung und Erzeugung der Templates übernimmt. Die CWinApp Klasse kapselt die zentrale Nachrichtenschleife („Message Loop“) und repräsentiert den Prozeß der Applikation. Die CWinApp Klasse erbt von CCmdTarget und kann damit auch Nachrichten empfangen. Beispielsweise können Objekte der CWinApp Klasse die Kommandos "Datei|Neu" und "Datei|Öffnen" empfangen, und sie können diese an die korrekten Template Objekte weiterleiten.

4.5.6 Zusammenfassung und Ausblick

Die Document/View Architektur verwirklicht die Infrastruktur der zu entwickelnden Anwendung. Dieser Abschnitt hat für den Leser die einzelnen Komponenten der Document/View Architektur und deren Zusammenwirken zusammengefaßt.

Die Document/View Architektur wurde entworfen um dokumentenbasierte Anwendungen zu konstruieren. In der Tat wird die Konstruktion dokumentenbasierter Anwendungen von der Document/View Architektur und den Entwicklungswerkzeugen der Visual C++ Entwicklungsumgebung besonders gut unterstützt.

Wie der Leser in den noch folgenden Kapiteln sehen wird, kann die Document/View Architektur auch für die Konstruktion alternativer Softwareentwürfe genutzt werden. Beispielsweise wird sie von uns für die Werkzeugkonstruktion und die Konstruktion der Gruppenarbeitsumgebung des Pausenplanersystems verwendet. Diese Verwendung der Document/View Architektur im Kontext einer WAM Konstruktion werde ich in den Kapiteln 7 und 8 detailliert beschreiben.

Die Erzeugung und Anpassung der Document/View Komponenten wird von den Entwicklungswerkzeugen der Visual C++ Entwicklungsumgebung sehr gut unterstützt. In Abschnitt 6.1 beschreibe ich, wie man mit dem AppWizard Kodegenerator ein Document/View Komponentengerüst automatisch erstellen kann.

Im nächsten Abschnitt werden Nachrichten und Kommandos einer Windows Applikation vorgestellt. Außerdem wird beschrieben, wie die Komponenten der Document/View Architektur Nachrichten und Kommandos empfangen und verarbeiten können.

4.6 Ereignisse, Nachrichten und Kommandos

Dieser Abschnitt zeigt die Konstruktion von Windows Applikationen aus einer ereignisorientierten Perspektive und beschreibt, wie der Anwendungsentwickler dabei durch die Verwendung des MFC Rahmenwerks unterstützt wird.

Das Windows Betriebssystem ist eine ereignisorientierte Umgebung; das bedeutet das Betriebssystem wartet permanent auf Aktionen des Benutzers. Eine Benutzeraktion, wie zum Beispiel die Bewegung der Maus oder ein Tastendruck, löst ein Hardware Ereignis aus. Dieses wird vom Betriebssystem aufgefangen und an die betroffene Windows Applikation in Form einer Nachricht weitergeleitet. Jede Windows Applikation gibt dann diese Nachricht an eine Funktion weiter, die die Nachricht verarbeitet. Genauer gesagt beschreibt die Funktion, wie auf die Benutzeraktion reagiert werden soll. Eine solche nachrichtenverarbeitende Funktion wird **Event-Handler** genannt.

In Abbildung 17 ist der oben beschriebene Ablauf von der Aktion bis zur Verarbeitung der Nachricht innerhalb der Applikation dargestellt.

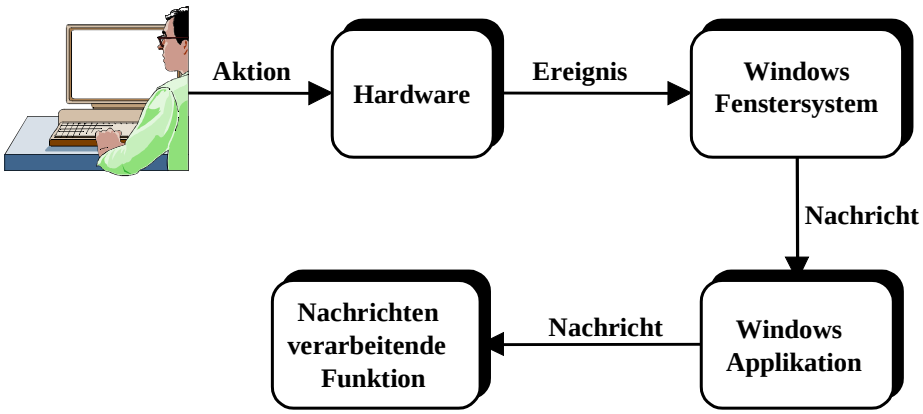


Abbildung 17: Aktion, Ereignis und Nachricht

Applikationen des Windows Fenstersystems bestehen meist aus mehreren Fenstern, die sich die Arbeit teilen und zusammen eine Fensterhierarchie bilden. Innerhalb jeder Applikation existiert ein Hauptfenster, welches die Wurzel der Hierarchie bildet. Jedes Applikationsfenster kann weitere Fenster enthalten, Diese nennt man **Kind-Fenster** (Child-Window) des Applikationsfensters. Aus Sicht der Kind-Fenster wird das verantwortliche Fenster **Eltern-Fenster** (Parent-Window) genannt. In der Fensterhierarchie ist es üblich, daß die Kind-Fenster ihre Eltern benachrichtigen, wenn sie wichtige Nachricht vom Fenstersystem erhalten haben.

Für jedes Applikationsfenster einer Windows Applikation muß eine Event-Handler Funktion beim Fenstersystem angemeldet werden. Durch die Anmeldung weiß das Fenstersystem, an welches Fenster die Nachricht gesendet werden muß. Die Event-Handler Funktion muß alle ankommenden Nachrichten erkennen und auf diese geeignet reagieren. Jede Windows Applikation implementiert außerdem eine zentrale Nachrichtenschleife („Message Loop“), welche die Nachrichten vom Windows Fenstersystem entgegennimmt und sie an die Event-Handler der jeweiligen Applikationsfenster weiter reicht.

Der Leser kann sehen, daß die Aufgaben eines Windows Anwendungsentwickler ohne eine geeignete Unterstützung recht umfangreich sind. In den nun folgenden Abschnitten möchte ich zeigen, wie sich diese Aufgaben durch die Verwendung des MFC Rahmenwerks vereinfachen.

4.6.1 Unterstützung durch das MFC Rahmenwerk

Das MFC Rahmenwerk und die im Rahmenwerk enthaltene Document/View Architektur unterstützen den Anwendungsentwickler bei der Konstruktion einer Applikation für das Windows Fenstersystem.

Die Event-Handler Funktion, die für jedes Applikationsfenster benötigt wird, ist innerhalb der CWnd Oberklasse gekapselt. Bei der Verwendung der von CWnd abgeleiteten Klassen innerhalb der Applikation, bleiben die Anmeldung der Event-Handler Funktion beim Fenstersystem, sowie die interne Verarbeitung der Nachrichten, vor dem Anwendungsentwickler verborgen (siehe Message-Mapping Technologie).

Die CWinApp Klasse kapselt die zentrale Nachrichtenschleife. Auch diese Tatsache bleibt dem Anwendungsentwickler bei der Verwendung der Applikation-Komponente der Document/View Architektur verborgen.

Allgemein kann man die Nachrichten, die innerhalb von Windows Applikationen versandt werden, in zwei verschiedenartig Nachrichtentypen gruppieren:

1. Die **Windows-Nachrichten** sind festgelegte Nachrichten des Windows Fenstersystems. Sie sind an die Fenster einer Applikation adressiert und werden vom Event-Handler der Fenster behandelt. Beispiele sind die WM_MOUSEMOVE, WM_CREATE, WM_CLOSE und WM_HELP Nachrichten. Im MFC Rahmenwerk existieren für die Behandlung der Windows-Nachrichten meist standardisierte Methoden in der CWnd Oberklasse und den von CWnd abgeleiteten Klassen. Möchte der Entwickler das standardisierte Verhalten verändern, so überschreibt er diese Methoden in den abgeleiteten Klassen seiner Applikation.

2. Die **Kommandos** sind spezifische Nachrichten der Applikation. Sie bestehen immer aus der WM_COMMAND Nachricht, die als Parameter eine applikationsweit eindeutige Identifikation (ID) des Kommandos mitführt. Kommandos sind immer an das Hauptfenster der Applikation adressiert und werden daher auch von der Event-Handler Funktion des Hauptfensters verarbeitet. Es ist für den Anwendungsentwickler leicht, neue Kommandos seiner Applikation hinzuzufügen. Innerhalb des MFC Rahmenwerks existiert bereits eine große Anzahl standardisierter Kommandos. So zum Beispiel die Kommandos mit der Identifikation „ID_FILE_NEW“ und „ID_FILE_OPEN“. Diese werden an die Applikation versendet, wenn der Benutzer die Menüeinträge „Datei|Neu“ und „Datei|Öffnen“ auswählt. Innerhalb der Document/View Architektur werden Kommandos normalerweise von Menüs und Symbolleisten der Frame-Komponente ausgesendet.

Die beiden Nachrichtentypen kann man außerdem auf folgende Weise unterscheiden: Ausgehend von einem Ereignis kann man zwei unterschiedliche Nachrichtenwege innerhalb der Document/View Architektur für Windows-Nachrichten und Kommandos beobachten.

Windows-Nachrichten erreichen auf direktem Wege die Fenster der Applikation. In der Document/View Architektur sind Frame- und View-Komponente sowie deren Subkomponenten die Applikationsfenster. Abbildung 18 zeigt die Nachrichtenwege der Windows-Nachrichten innerhalb der Document/View Architektur.

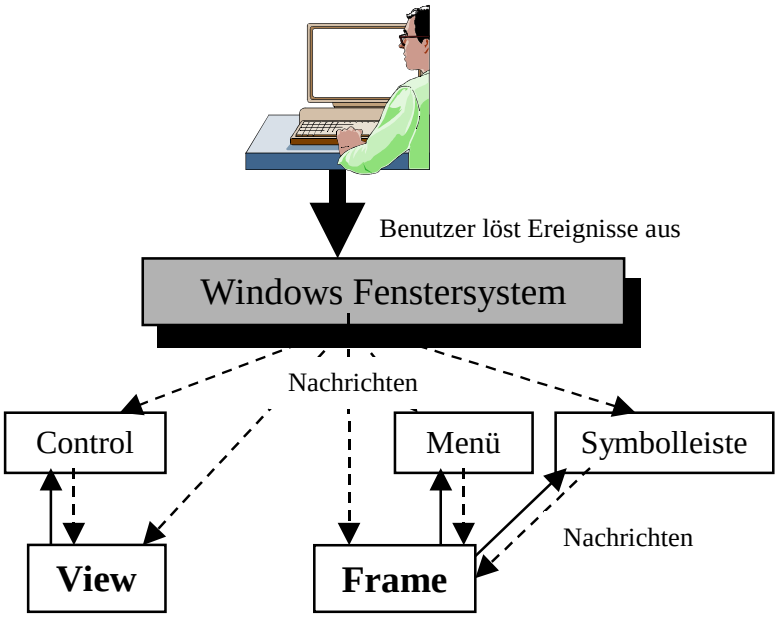


Abbildung 18: Nachrichten des Windows Fenstersystems an die Komponenten der Document/View Architektur

Handelt es sich beim Nachrichtentyp um ein **Kommando**, so wird jeder Komponente der Document/View Architektur die Möglichkeit gegeben, auf dieses Kommando zu reagieren. Abbildung 19 zeigt, welche Route ein Kommando durch die Komponenten der Document/View Architektur nehmen kann.

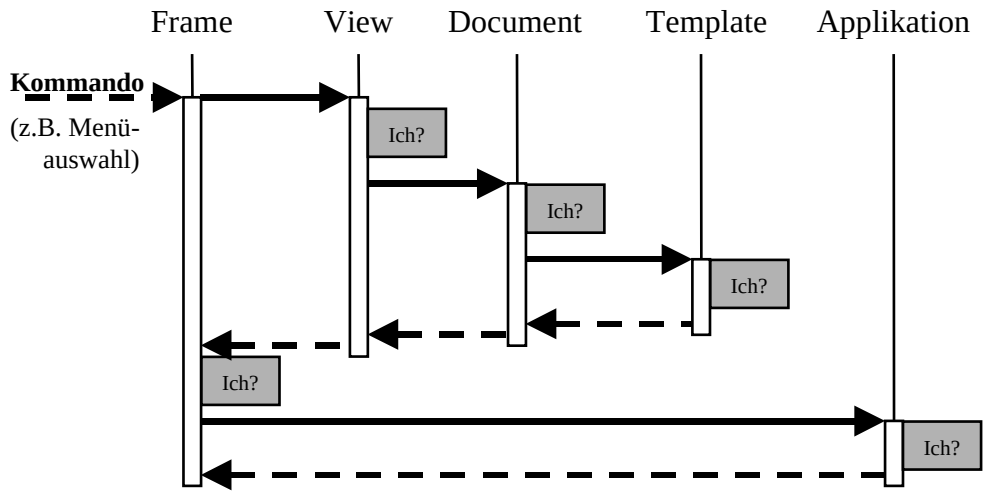


Abbildung 19: Route eines Kommandos durch die Komponenten der Document/View Architektur

Die Route beginnt bei der Frame-Komponente (das Hauptfenster der Applikation), die das Kommando ohne eigene Prüfung sofort an die View-Komponente weiterleitet. Ist eine Event-Handler Funktion für ein Kommando gefunden worden (siehe Message-Mapping Technologie), so wird das weitere Routing abgebrochen. Nicht bearbeitete Kommandos landen am Ende der Route bei der Applikations-Komponente.

Das Routing der Kommandos durch die Komponenten der Document/View Architektur wird innerhalb der *OnCmdMsg(...)* Methode der CCmdTarget Klasse festgelegt. Die von der Architektur festgelegte Route kann vom Anwendungsentwickler durch überladen der *OnCmdMsg(...)* Methode in abgeleiteten Klassen verändert werden.

4.6.2 Die Message-Mapping Technologie

Die Verwendung der Message-Mapping Technologie vereinfacht die Aufgaben des Anwendungsentwickler erheblich. Sie ermöglicht die Verarbeitung von Windows-Nachrichten und Kommandos durch Methoden einer Klasse.

Die Message-Mapping Technologie besteht aus zwei Teilen:

1. **CCmdTarget**: Die CCmdTarget Klasse ist die Basisklasse aller Nachrichten verarbeitenden Klassen. Sie implementiert den internen Nachrichtenmechanismus der Message-Maps.
2. **Message-Maps**: Die Message-Maps ordnen Nachrichten (Windows-Nachrichten oder Kommandos) Methoden einer Klasse zu, welche diese Nachrichten verarbeiten.

Betrachtet man die Vererbungshierarchie des MFC Rahmenwerks, so sieht man, daß relativ viele Klassen von CCmdTarget erben. Alle Objekte dieser Klassen können Nachrichten empfangen, indem man eine Message-Map in den Quellcode der Klasse einfügt.

Eine Message-Map wird durch die Verwendung von Makros generiert. Im unteren Text ist der Quellcode eines Message-Map Beispiels dargestellt:

```
BEGIN_MESSAGE_MAP(CTestView, CView)
  //{{AFX_MSG_MAP(CTestView)
  ON_WM_MOUSEMOVE()
  ON_COMMAND(ID_STRING_CENTER, OnStringCenter)
  //}}AFX_MSG_MAP
END_MESSAGE_MAP()
```

Die Expansion dieser Makros generiert eine Message-Map innerhalb der Klasse CTestView, die von CView erbt.

Die BEGIN_MESSAGE_MAP und END_MESSAGE_MAP Makros befinden sich in der Implementations Datei „CTestView.CPP“, sie bestimmen Beginn und Ende der Message-Map. Damit bilden sie einem Rahmen für die einzelnen Einträge in der Message-Map.

Die Oberklasse CView wird im BEGIN_MESSAGE_MAP Makro angegeben, da bei erfolgloser Suche nach einer Event-Handler Funktion in der lokalen Message-Map, auch die Message-Map der Oberklasse durchsucht wird.

Betrachten wir nun die Makros für die Einträge innerhalb der Message-Map. Das ON_WM_MOUSEMOVE Makro sorgt dafür, daß beim Empfang der Mausbewegungsnachricht WM_MOUSEMOVE die Methode *OnMauseMove(...)* der Klasse CTestView aufgerufen wird. Technisch wird dies über den Eintrag in einer Tabelle gelöst. Das ON_WM_MOUSEMOVE Makro bewirkt, daß in der Tabelle die WM_MOUSEMOVE Nachricht mit der *OnMauseMove(...)* Methode verknüpft wird.

Die Parameter der WM_MOUSEMOVE Nachricht beinhalten generische Informationen über die x und y Koordinaten des Mauszeigers, sowie Informationen über den gleichzeitigen Druck auf eine virtuelle Taste, wie zum Beispiel das Verwenden der rechten Maustaste. Der Message-Map Mechanismus wandelt diese Informationen in korrekt getypte Parameter der Event-Handler Funktionen um. Die x und y Koordinatenwerte des Mauszeigers werden in einer CPoint Variable übergeben und die Tastendruck-Information in einer Unsigned Integer Variable. Der *OnMauseMove()* Methodenkopf hat demnach folgendes Aussehen:

```
void CTestView::OnMouseMove(UINT nFlags, CPoint point) { ... }
```

Das ON_COMMAND Makro in der Message-Map sorgt dafür, daß beim Empfang des „ID_STRING_CENTER“ Kommandos die parameterlose Methode *CTestView::OnStringCenter()* aufgerufen wird.

Die Erzeugung und Änderung der Message-Map übernimmt das ClassWizard Entwicklungswerkzeug der Visual C++ Entwicklungsumgebung (siehe Kapitel 6)

4.6.3 Zusammenfassung

Dieser Abschnitt hat dem Leser die Konstruktion von Windows Applikationen unter Verwendung des MFC Rahmenwerks beschrieben. Für das Verständnis nachfolgender Kapitel ist besonders die Unterscheidung der beiden Nachrichtentypen, sowie der Empfang und die Verarbeitung von Nachrichten und Kommandos durch die Komponenten der Document/View Architektur, von Bedeutung.

Der Abschnitt konnte andeuten, wie das MFC Rahmenwerk die Verarbeitung von Nachrichten unter Verwendung der CCmdTarget Klasse und der Message-Mapping Technologie bewerkstelligt. Betreffend der Message-Mapping Technologie ist mir bei der Ausarbeitung dieses Abschnitts noch folgende Frage durch den Kopf gegangen.

Warum implementierten die Entwickler des MFC Rahmenwerks den Nachrichtenmechanismus durch Message-Maps und nicht durch virtuelle Methoden?

Das hat folgende Gründe: Bei der großen Anzahl von möglichen Nachrichten im Windows Betriebssystem würde der VTable[9] einer C++ Klasse, der die virtuellen Methoden verwaltet, sehr groß werden. Ein weiterer Nachteil ist, daß Veränderungen im Nachrichtenkonzept des Betriebsystems auch zu Veränderungen im Quellcode des Rahmenwerks führen würden. Zusätzlicher Vorteil der Message-Map Technik ist ihre Compilerunabhängigkeit und der geringe Ressourcenverbrauch.

[9] Auch "Virtual Funktion Table" genannt. (siehe [Eck96])

5 COM

Dieses Kapitel gibt eine Einführung in die Microsoft COM (Component Object Model) Technologie. Die Einführung beschränkt sich nur auf die für diese Arbeit wesentlichen Aspekte von COM. Eine detailliertere Perspektive auf COM kann man in den Büchern [Bro95], [Tem97] und [Rog97] erhalten. Besonders interessant ist das Buch „Beginning MFC COM Programming“ von Julian Templeman [Tem97] da es COM aus der Sichtweise des MFC Rahmenwerks beschreibt. Einen guten Überblick über COM und andere Objekt Infrastrukturen kann der Leser in der "Essential Distributet Objects Survival Guide" [OHE96] bekommen.

Was genau ist COM eigentlich? Im "COM Programmer’s Reference" [COM97] steht dazu:

> "COM is a technology that allows objects to interact across process and machine boundaries as easily as objects within a single process interact".

Dieser Satz beschreibt die Hauptaufgabe der COM Technologie. Zusammenfassend kann man sagen, daß COM die Kommunikation zwischen Objekten über Prozeßgrenzen hinweg ermöglicht. Wobei man erwähnen sollte, daß erst die Distributet COM Erweiterung (DCOM) die Kommunikation über Rechnergrenzen hinweg ermöglicht (siehe [Gri97]).

Damit ist COM die CORBA[10] Alternative von Microsoft. Für die Konstruktion des Pausenplanersystems haben wir COM vorgezogen, da die Technologie in das MFC Rahmenwerk integriert ist. Dieser Umstand vereinfacht die Entwicklung verteilt arbeitender Softwarekomponenten unter Verwendung des Rahmenwerks erheblich.

Der Kern der COM Technologie besteht aus einer binären Spezifikation, die beschreibt wie Softwarekomponenten kommunizieren können. Die binäre Spezifikation ist unabhängig von einer Programmiersprache oder einem Compiler, was bedeutet, daß COM Komponenten in beliebigen Programmiersprachen implementiert werden können. Die COM Technologie besteht weiterhin aus einigen Basisinterfaces, sowie Windows API Funktionen, die von allen COM Komponenten genutzt werden.

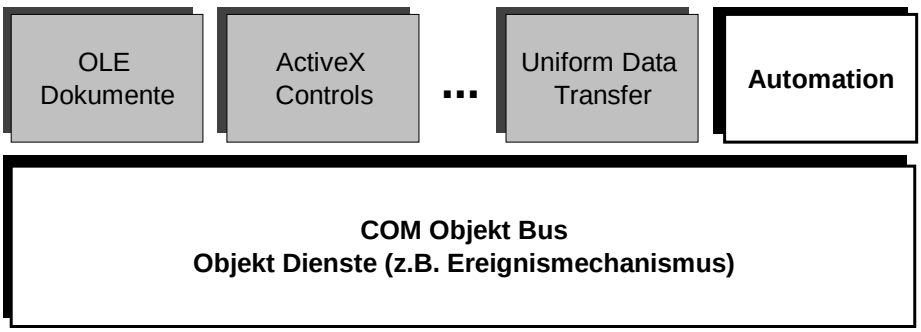


Abbildung 20: COM ist Basistechnologie für höhere Dienste

COM bildet die Basistechnologie für weitere objektbasierte Technologien von Microsoft, die in der Literatur meist unter den Oberbegriffen OLE[11] bzw. ActiveX[12] zusammengefaßt werden. Diese Technologien bieten höhere Dienste und Standards für den Entwickler an. Einer dieser höheren Dienste, den ich in meiner Arbeit genauer beschreiben möchte, nennt sich Automation (siehe Kapitel 5.3) und wurde bei der Konstruktion des Pausenplanersystems häufig verwendet.

Die COM, ActiveX und OLE Technologien sind Erweiterungen der Windows Betriebssysteme und daher in diese integriert. Die DCOM Erweiterung ist in Windows NT 4.0 enthalten. Für Windows 95 kann man eine Erweiterung installieren.

[10] Die "Common Object Request Broker Architecture" (siehe [Cor95]) ist die Objekt Infrastruktur der Object Management Group (OMG)
[11] Object Linking and Embedding
[12] Teile der OLE Technologien, die nicht im Zusammenhang mit „Linking and Embedding“ stehen, werden in der aktuellen Literatur von Microsoft unter dem Oberbegriff ActiveX zusammengefaßt

Ein Vorteil dieser Integration ist, daß man mit dem Erwerb eines Windows Betriebsystems die COM, ActiveX und OLE Technologien mit erwirbt. Diese starke Einbindung in die Betriebssysteme läßt auch auf eine große Verbreitung schließen.

Ein Nachteil ist die Abhängigkeit vom Betriebsystem, da COM keine offene Objekt Infrastruktur darstellt, und deshalb die Technologie von keinem anderen Hersteller angeboten wird. Ein Lichtblick sind eventuell die gemeinsamen Bemühungen von Microsoft und der Software-AG, die COM Technologie auch auf anderen Rechnerplattformen (Solaris, HP-UX, AIX) umzusetzen.

5.1 Interfaces in COM

Ein elementarer Bestandteil von COM sind die Interfaces. Was genau ist ein COM Interface?
Im "COM Programmer's Reference" [COM97] steht dazu:

> „A group of semantically related functions that provide access to a COM object. Each interface defines a contract that allows objects to interact according to the Component Object Model (COM)."

Ein Interface ist also eine Menge von semantisch zusammengehörigen Funktionen, die als vertragliche Bindung zwischen Kunde und Dienstanbieter angesehen werden können. Die Funktionen des Interface werden in statisch getypter Form definiert. Eine Kommunikation zwischen Kunde und Dienstanbieter erfolgt ausschließlich über den Aufruf der Interface Funktionen. Das Interface definiert lediglich die Funktionen unabhängig von der Implementierung des Dienstanbieters, man kann es daher mit einer abstrakten Klasse einer objektorientierten Programmiersprache vergleichen.

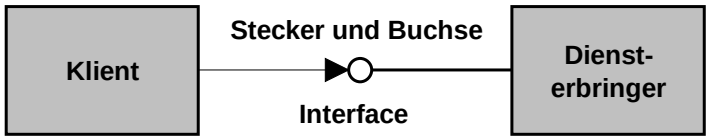


Abbildung 21: Kommunikation zwischen Klient und Diensterbringer über ein Interface (Stekker/Buchse Notation)

Abbildung 21 zeigt den Zugriff eines Klienten auf ein Interface eines Diensterbringers. Die COM Technologie realisiert dabei die technische Ortstransparenz des Diensterbringers. Das bedeutet, daß der Klient nicht wissen muß, wo sich der Diensterbringer befindet. Dieser kann sich mit dem Klienten im selben Prozeß befinden, in einem anderen Prozeß oder in einem Prozeß auf einer anderen Maschine (DCOM).

Eine eindeutige Identifikation hat jedes Interface durch seinen Interface Identifier (IID). Durch die Vergabe dieser Identifikation wird die Definition eines Interfaces vertraglich festgeschrieben. Eine nachträgliche Änderung des Interface ist nun nicht mehr möglich. Das bedeutet, daß bei Erweiterungen oder Veränderungen des Interfaces ein neuer Interface Identifier verwendet werden muß.

Jedes COM Interface hat zusätzlich einen symbolischen Namen, der üblicherweise mit dem Buchstaben „I" beginnt. Ein Beispiel ist das Interface „IUnknown", das ich in diesem Abschnitt noch vorstellen werde.

COM Interfaces werden in einer eigenen IDL (Interface Definition Language) beschrieben, die nicht mit der CORBA IDL der Object Management Group (siehe [Cor95]) kompatibel ist.

5.2 Klassen, Objekte und Vererbung in COM

Der Einsatz der COM Technologie weicht von der Verwendung klassischer objektorientierter Programmiersprachen ab. Daher möchte ich im folgenden Abschnitt auf die Unterschiede in der Benutzung eingehen. Zunächst möchte ich beschreiben wie Klassen und Objekte, die Grundelemente aller objektorientierter Programmiersprachen, in der COM Technologie umgesetzt werden und beginne mit der Frage:

„Was ist eine COM Klasse?“

Eine **COM Klasse** ist die statische Beschreibung einer Menge möglicher COM Objekte. Die Beschreibung kann mehrere Interfaces beinhalten, die von der Klasse implementiert werden. Die Implementation ist unabhängig von einer bestimmten Programmiersprache, kann also in jeder geeigneten Sprache erfolgen. Jede COM Klasse hat eine eindeutige Identifikation, die „Class ID“ (CLSID) genannt wird.

„Was genau ist dann ein COM Objekt?“

Ein **COM Objekt** ist das Exemplar einer COM Klasse und existiert demnach nur zur Laufzeit. Ein COM Objekt bietet an seiner Schnittstelle alle Interfaces der COM Klasse an. Für COM Objekte selber gibt es keine Identifikation! Man kann sich demnach nicht mit einem bestimmten Objekt über den COM Objektbus verbinden lassen. Ein Umstand, der den Unterschied zu den klassischen objektorientierten Programmiersprachen deutlich macht.

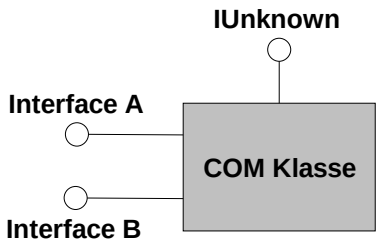


Abbildung 22: Eine COM Klasse implementiert die Interfaces A, B und IUnknown

Abbildung 22 zeigt die Stecker/Buchse Notation einer COM Klasse, die die Interfaces A, B und IUnknown, anbietet. Sie werden sich nun bestimmt fragen, was es mit dem IUnknown Interface dieser COM Klasse auf sich hat?

Das **IUnknown** Interface ist ein vordefiniertes COM Basisinterface, daß von jeder COM Klasse implementiert werden muß. Es wird verwendet, um dem Klienten den Zugriff auf die Interfaces des COM Objekts zu ermöglichen. Zusätzlich kann der Klient über das IUnknown Interface die Lebensdauer des COM Objekts beeinflussen.

Über die *QuerryInterface(...)* Funktion des IUnknown Interfaces können die Klienten herausfinden, ob das Objekt ein bestimmtes Interface anbietet. Das Interface wird dabei durch den Interface Identifier (IID) bestimmt. Bietet das COM Objekt das gewünschte Interface an, so liefert die *QuerryInterface(...)* Funktion eine Referenz darauf zurück und erhöht gleichzeitig den internen Referenzzähler des Objekts.

Über die *AddRef()* und *Release()* Funktionen des IUnknown Interfaces können die Klienten den internen Referenzzähler des Objekts direkt beeinflussen. *AddRef()* erhöht den internen Referenzzähler des COM Objekts um eine Referenz, während *Release()* ihn um eine Referenz erniedrigt. Die Lebenszeit des Objekts endet, wenn keine Referenzen mehr auf das Objekt existieren, der Speicher und andere Ressourcen können dann freigegeben werden.

„Wie funktioniert Vererbung?“

Vererbung findet innerhalb der COM Technologie nur auf der Ebene von Interfaces statt. Ein Interface Y erbt die Spezifikation eines Interfaces X, indem es die Funktionen von X übernimmt.

Dabei gibt es eine feste Grundregel: Alle COM Interfaces müssen vom IUnknown Interface erben. Das bedeutet, daß eine implementierende COM Klasse in jedem angebotenen Interface die drei Funktionen des IUnknown Interfaces umsetzen muß (siehe Abbildung 23). Man kann sich das IUnknown Interfaces somit als Basisspezifi-

kation aller Interfaces vorstellen. Ein Klient kann demnach über jedes Interface eines Diensterbringers andere Interfaces erfragen und den internen Referenzzähler des Diensterbringers beeinflussen.

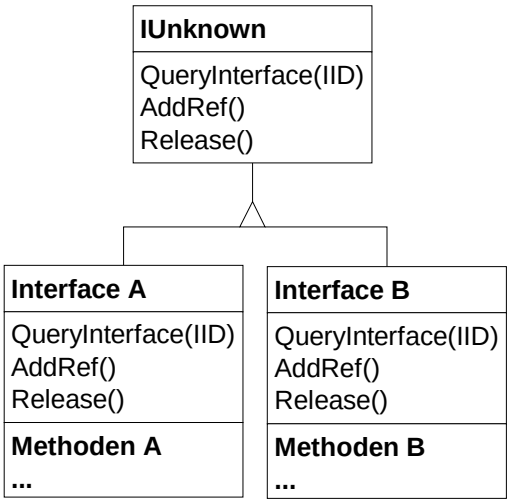


Abbildung 23: Vererbung von COM Interfaces

Die COM Technologie unterstützt nicht die Vererbung von Klassen, wie es bei objektorientierten Sprachen der Fall ist. Die Implementation eines bestimmten Interfaces innerhalb einer COM Klasse bildet zwar das Erben von einer Spezifikationsklasse nach, bei dieser Art der Vererbung bezieht man sich jedoch nicht auf eine Klasse, sondern auf ein Interface.

Um die Erweiterung und Spezialisierung einer COM Klasse dennoch zu ermöglichen hat Microsoft zwei alternative Mechanismen entworfen, welche „Aggregation“ und „Containment“ genannt werden. Beide Mechanismen versuchen Klassen in Klassen zu kapseln, wozu sie unterschiedliche Ansätze verwenden. Die Vorgehensweise dabei ist eher technischer Natur, sie läßt sich in eigenen Projekten schwer umsetzen. Details zu dieser Art der Wiederverwendung möchte ich an dieser Stelle nicht beschreiben, sie können in [Bro95] nachgelesen werden.

Meiner Meinung nach hat COM hier einen klaren Nachteil gegenüber CORBA, welches die Vererbung von Klassen unterstützt. Microsoft begründet die Einschränkung mit der Aussage, daß ein Vertrag zwischen Klient und Diensterbringer innerhalb einer Implementationshierarchie nicht klar definiert sei.

Diese Problematik wurde auch von Bertrand Meyer erkannt und in seinem Paradigma „Programmieren durch Vertrag“ beschrieben (siehe [Mey90]). Meyer führt hier das Konzept des Unterauftrags ein.

> *„Inheritance is dangerous, so Microsoft’s COM won’t support it. It’s like saying divide-by-zero is dangerous, so let’s remove the „devide“ operation from our microprocessors.“*
>
> ***Cliff Reeves, IBM Director of Objects (Feb, 1995)***

5.3 Automation

Automation ist ein höherer Dienst der auf der Basistechnologie COM aufbaut. Der Automationsdienst ermöglicht eine abstraktere Sicht auf die Kommunikation zwischen Komponenten und wurde daher bei der Konstruktion des Pausenplanersystem verwendet. Dieser Abschnitt gibt dem Leser zunächst einen Überblick über den Automationsdienst.

Ursprünglich wurde Automation entwickelt um einem Klienten die Steuerung großer Applikationen zu ermöglichen, wobei die Klienten Zugriff auf die Methoden und Eigenschaften (Properties) von Objekten dieser Applikationen haben. Applikationen, die Automation unterstützen, sind **Automationsserver**. Die Objekte eines Automationsservers können über eine Automations-Schnittstelle von außerhalb angesprochen werden. Es handelt sich also um entfernt zugreifbare Objekte. Die Objekte eines Automationsservers werden **Automationsobjekte** genannt.

Eine Automations-Schnittstelle besteht aus Methoden und Eigenschaften. Automations-Klienten rufen die Methoden auf, um innerhalb des Automationsservers eine Aktion auszulösen. Über die Eigenschaften haben sie sondierenden bzw. verändernden Zugriff auf den Automationsserver. Dabei löst Automation ältere Techniken, wie zum Beispiel DDE (Dynamic Data Exchange) oder proprietäre Makrosprachen, ab (siehe [Mat97]). Zur Zeit unterstützt jede neu entwickelte Windows Applikation Automation und kann dadurch wie eine Komponente wiederverwendet werden.

Da viele Automations-Klienten durch interpretierte Programmiersprachen implementiert werden, muß ein Automationsserver einen dynamischen Zugriff auf Methoden und Eigenschaften unterstützen. Eine Bindung sollte daher erst zur Laufzeit erfolgen. Dieses Vorgehen nennt man auch „späte Bindung“, sie ermöglicht es in Programmiersprachen wie Smalltalk oder Visual Basic, die ausschließlich späte Bindung unterstützen, Automation zu verwenden.

Das Fundament auf dem Automation aufbaut, ist das **IDispatch** Interface. Über dieses vordefinierte COM Basisinterface greifen die Automations-Klienten dynamisch auf die Objekte des Automationsservers zu.

Wichtigster Bestandteil des IDispatch Interfaces ist die *Invoke(...)* Funktion. Mit dem Aufruf dieser Funktion, initialisiert ein Klient die Bearbeitung eines Zugriffs beim Automationsserver. Der Zugriff des Klienten besteht aus dem Aufruf einer Methode oder dem Zugriff auf eine Eigenschaft des Automationsobjekts. Die Implementation des IDispatch Interfaces prüft zur Laufzeit die Anfrage des Klienten. Dabei wird festgestellt, ob das Automationsobjekt überhaupt die gewünschte Methode oder Eigenschaft besitzt, außerdem werden Typ und Anzahl der übergebenen Parameter überprüft. Sind alle Angaben korrekt, so startet die Bearbeitung und das Automationsobjekt übergibt den Rückgabewerk an den Klienten, andernfalls wird ein Fehler gemeldet.

Wie man sieht realisiert das IDispatch Interface die späte Bindung zwischen Automations-Klienten und Automationsserver. Im Gegensatz dazu unterstützen COM Interfaces nur eine frühe Bindung und sind damit für Programmiersprachen mit später Bindung nur schwerlich ansprechbar. Der Automationsdienst hat noch einen weiteren Vorteil gegenüber der COM Basistechnologie. Findet die Kommunikation zwischen zwei verschiedenen Rechnerarchitekturen statt, so kümmert sich der Automationsdienst um die geeignete Umwandlung der Parameter- und Rückgabewerte. Die COM Basistechnologie leistet diese Umwandlung nicht.

Der Automationsdienst besitzt eigene **Automationstypen** für Parameter- und Rückgabewerte, diese sind unabhängig von Programmiersprachen und Rechnerarchitekturen. Es existieren Automationstypen für die gebräuchlichsten Werte (Integer, Real, String ...) und für Referenzen auf weitere Automationsserver.

Durch die Verwendung von Referenzen auf Automationsserver kann die Referenz-Parameter Problematik vieler RPC (Remote Procedure Call) Implementierungen aufgelöst werden. RPC Implementierungen verzichten meist auf Referenz-Parameter oder verwenden alternativ einen „Copy/Restore“ Parametermechanismus (vergl. [Tan92]).

„Warum wurde bei der Konstruktion des Pausenplanersystems Automation verwendet?“

Wir haben Automation verwendet, anstatt die COM Basistechnologie direkt zu verwenden, da der Automationsdienst eine abstraktere Sicht auf die Kommunikation zwischen den Komponenten ermöglicht und innerhalb des MFC Rahmenwerks sehr gut integriert ist. Diese Integration wird von den Entwicklungswerkzeugen der Visual

C++ Entwicklungsumgebung ausgezeichnet unterstützt. Die Entwicklungswerkzeuge erleichtern die Konstruktion von Automations-Klienten und -Diensterbringern erheblich.

Sowohl die Werkzeugkonstruktion als auch die Konstruktion der anderen Komponenten des Pausenplanersystems hat sich durch diese Tatsachen deutlich vereinfacht. Der Leser möge sich zu diesem Zeitpunkt zunächst mit dieser kompakten Antwort zufrieden geben, da ausführliche Antworten in den nächsten Kapiteln folgen.

Der nächste Abschnitt beschreibt, wie der Automationsdienst in das MFC Rahmenwerk integriert ist.

5.4 Automation im MFC Rahmenwerk

Nachdem vorgestellt wurde, wie die Automation auf der COM Ebene realisiert wird, möchte ich in diesem Abschnitt beschreiben, wie die Automation im MFC Rahmenwerk integriert ist.

Die CCmdTarget Klasse ist die Basisklasse aller Nachrichten verarbeitenden Klassen des MFC Rahmenwerks. Auch die Automation ist innerhalb der CCmdTarget Klasse gekapselt. Daraus ergibt sich, daß jede Klasse die von CCmdTarget erbt Automation unterstützen kann. Die Betonung liegt hier auf kann, da nicht unbedingt jeder Nachkomme von CCmdTarget über den Automationsdienst angesprochen werden soll.

Um ein CCmdTarget Objekt zu einem Automationsobjekt zu machen, muß die Methode *EnableAutomation()* aufgerufen werden. Diese verbindet das CCmdTarget Objekt zur Laufzeit mit einem COleDispatchImpl Objekt, welches die Implementation des IDispatch Interfaces enthält. CCmdTarget Klassen ohne Automation müssen durch den beschriebenen Mechanismus nicht den gesamten Automations Overhead mitführen, dennoch ist es möglich auf alle Objekte von CCmdTarget angeleiteter Klassen über Automation zuzugreifen.

Neben der CCmdTarget Klasse verwendet das MFC Rahmenwerk die Dispatch-Map Technologie, um Automation umzusetzen. Wie die Message-Maps des MFC Nachrichtenmechanismus, werden auch die Dispatch-Maps aus Makros aufgebaut. Diese beschreiben die einzelnen Methoden und Eigenschaften des Automationsobjekts.

Betrachten wir dazu ein Beispiel aus dem Pausenplanersystem. Die Document-Komponente „CAufsichtsorteBearbeiterDoc“ des Aufsichtsorte-Bearbeiter Werkzeug ist ein Automationsobjekt und stellt zwei Automationsmethoden zur Verfügung. Die beiden Methoden *Init(...)* und *NeuDarstellen()* bilden im C++ Quellcode folgende Dispatch-Map:

```
BEGIN_DISPATCH_MAP(CAufsichtsorteBearbeiterDoc, CDocument)
//{{AFX_DISPATCH_MAP(CAufsichtsorteBearbeiterDoc)
   DISP_FUNCTION(CAufsichtsorteBearbeiterDoc, "NeuDarstellen", NeuDarstellen,
      VT_EMPTY, VTS_NONE)
   DISP_FUNCTION(CAufsichtsorteBearbeiterDoc, "Init", Init, VT_EMPTY, VTS_DISPATCH
      VTS_BSTR)
//}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

Die BEGIN_ DISPATCH _MAP und END_ DISPATCH _MAP Makros bestimmen Beginn und Ende der Dispatch-Map. Für die beiden Automationsmethoden *Init(...)* und *NeuDarstellen()* werden in der Dispatch-Map mit Hilfe des DISP_FUNCTION Makros die internen und externen Methodennamen sowie die Automationstypen der Parameter und Rückgabewerte festgehalten.

Dabei benennt der interne Methodenname die Methode der CAufsichtsorteBearbeiterDoc Klasse, die die Automationsmethode implementiert. Der externe Name ist der Name der Automationsmethode, unter dem ein Klient die Methode aufruft. Dieser Methodenname wird dem DISP_FUNCTION Makro als Zeichenkette übergeben. Die Automationstypen der Parameter und Rückgabewerte werden über definierte Makros festgelegt.

Die Erzeugung und Änderung der Dispatch-Map übernimmt das ClassWizard Entwicklungswerkzeug der Visual C++ Entwicklungsumgebung (siehe Kapitel 6)

5.5 Zusammenfassung und Ausblick

In diesem Kapitel habe ich die für diese Arbeit wesentlichen Aspekte der COM Technologie beschrieben. Ich konnte zeigen, daß die COM Technologie ein großes und komplexes Gebilde ist auf dessen Basis viele weitere Dienste und Standards aufbauen.

Ich habe die COM Interfaces als elementaren Bestandteil von COM beschrieben und gezeigt, wie die Microsoft Entwickler die grundlegenden Konstrukte objektorientierter Programmiersprachen wie Objekte, Klassen und Vererbung in COM abgebildet haben.

Damit sich der Leser einen zusammenfassenden Überblick über diese Abbildung machen kann, habe ich in der folgenden Tabelle Konstrukte und Verwendung der objektorientierten Programmiersprache C++ und der COM Technologie miteinander verglichen.

Tabelle 3: C++ vs. COM

	C++	COM
Objekte	ja	ja
Klassen	ja	ja
Vererbung von Klassen	ja	nein
Objekterzeugung	new	COM API Funktion
Objektzerstörung	delete	Objekt löscht sich, wenn interner Referenzzähler=0
Benutztbeziehung	Zeiger auf Objekt	Zeiger auf Interface
Zugriff auf Objektzustand über	Methoden, Variablen	Methoden
Objekt Aufenthaltsort	im selben Prozeßraum	technische Transparenz
Laufzeitinformation	RTTI Mechanismus	über Methode der Klasse

Um dem Leser die Basis der komponentenbasierten Konstruktion des Pausenplanersystems näher zu bringen, habe ich danach den Automationsdienst und dessen Integration in das MFC Rahmenwerk beschrieben. Auf die Konstruktion von Werkzeugen, Materialien und Gruppenumgebung des Pausenplaners und die Kommunikation zwischen diesen Komponenten werde ich in den Kapiteln 7 und 8 detailliert eingehen.

Abschließend noch ein Überblick über Vor- und Nachteile der COM Technologie aus meiner Perspektive:

Ein großer Vorteil der COM Technologie ist die Integration der Technologie in die Windows Betriebssysteme, Anwendungsprogramme, Rahmenwerke und Entwicklungswerkzeuge. Die Technologie ist in allen Produkten und Systemen von Microsoft präsent. Die Verbreitung und Akzeptanz der Komponententechnologie ist enorm. Beispielsweise existieren Tausende von vorgefertigten qualitativ hochwertigen ActiveX Komponenten. Teile der Windows Betriebssysteme und Windows Anwendungsprogramme werden zunehmend aus Komponenten konstruiert. Betriebssysteme und Anwendungsprogramme stellen ihre Komponenten auch anderen Entwicklern für deren Softwareentwicklung zur Verfügung. Rahmenwerke und Entwicklungswerkzeuge unterstützen den Entwickler bei einem komponentenbasierten Entwurf seiner Software.

Nachteile ergeben sich aus der Abhängigkeit von der COM Technologie. Microsoft versucht hier eigene Standards zu setzen. Das äußert sich beispielsweise in der Verwendung einer eigenen nicht kompatiblen IDL (Interface Definition Language) für die Beschreibung der Interfaces. CORBA wird als direkter Konkurrent zu COM gesehen und in Produkten von Microsoft überhaupt nicht unterstützt.

6 Die MFC Entwicklungswerkzeuge

Dieses Kapitel liefert dem Leser eine Beschreibung der wichtigsten Entwicklungswerkzeuge für das MFC Rahmenwerk. Im Bereich der Windows Betriebssysteme existieren mehrere Entwicklungsumgebungen, die Entwicklungswerkzeuge für das MFC Rahmenwerk bereitstellen. Bei der Umsetzung des Pausenplanersystems wurde von uns die Visual C++ 5.0 Entwicklungsumgebung verwendet (siehe [Kru96] und [Tot96]). Ich beschränke mich daher auf die Beschreibung der Entwicklungswerkzeuge dieser Umgebung.

Visual C++ ist eine integrierende Entwicklungsumgebung, welche die unterschiedlichsten Entwicklungswerkzeuge, Bibliotheken und Sprachen zusammenfaßt. Aus der Vielzahl der angebotenen Werkzeuge möchte ich die zwei wichtigsten herausgreifen und beschreiben. Dabei handelt es sich um die Werkzeuge **AppWizard**[13] und **ClassWizard**. Beide Werkzeuge sind **Kodegeneratoren**, die dem Entwickler den Umgang mit dem MFC Rahmenwerk erleichtern sollen.

Der folgende Text beschreibt beide Kodegeneratoren sowie den Dialogeditor. Der Dialogeditor ist ein Werkzeug der Visual C++ Entwicklungsumgebung, das bei der Umsetzung des Pausenplanersystems für die Gestaltung der Werkzeugoberflächen eingesetzt wurde. Ich zeige, was der Entwickler mit den Werkzeugen bewirken kann und welche Arbeitsvorgänge ihm abgenommen werden.

6.1 Das AppWizard Entwicklungswerkzeug

Die Hauptfunktion des MFC AppWizard Entwicklungswerkzeugs ist die Erzeugung der grundlegenden Bestandteile einer Applikation. Zu den grundlegenden Bestandteilen gehören die einzelnen Komponenten der Document/View Architektur. Der AppWizard erzeugt für die View-, Document-, Frame- und Applikation-Komponenten jeweils eine aus dem Rahmenwerk abgeleitete Klasse. Die abgeleiteten Klassen sind Klassen des aktuellen Projekts, ihr Quelltext wird in Dateien gespeichert. Gemeinsam bilden die erzeugten Klassen ein elementares Gerüst, das der Entwickler nutzt, um seine Applikation aufzubauen. Der AppWizard erstellt zusätzlich weitere notwendige Dateien, die für die Übersetzung und Verwaltung der Quelldateien benötigt werden.

Der Entwickler entscheidet innerhalb eines Prozesses, wie seine Applikation aussehen soll und welche Bestandteile er von der Document/View Architektur verwenden möchte. Dieser Entscheidungsprozeß untergliedert sich in sechs Schritte:

1. Der Entwickler legt fest, ob er den SDI oder den MDI Applikationstyp verwenden möchte und welche nationale Sprache er in der Applikation benutzen will.
2. Er bestimmt, ob die Applikation Zugriff auf eine Datenbank erhalten soll.
3. Er legt fest, welche OLE-Dokument Funktionalität die Applikation unterstützen kann. Weiterhin bestimmt er, ob die zukünftige Applikation über Automation gesteuert werden kann. Der Entwickler kann zusätzlich angeben, ob er OLE/ActiveX Controls verwenden möchte.
4. In diesem Schritt legt der Entwickler zusätzliche Funktionalität (Drucken, Seitenansicht, Kontextsensitive Hilfe) sowie zusätzliche zu verwendende Benutzerinteraktionskomponenten (Symbol-, Statusleiste) fest.
5. Der Entwickler kann entscheiden, ob in den erzeugten Quelltexten Kommentare erscheinen sollen. Zusätzlich kann er bestimmen, ob die Applikation das MFC Rahmenwerk statisch oder als dynamische Bibliothek (DLL) verwenden soll.
6. Im letzten Schritt kann der Entwickler die Namen der zu erzeugenden Klassen seinen Wünschen anpassen und bestimmen, welche View Klasse er in seiner Applikation verwenden möchte.

Abbildung 24 zeigt, wie ein vom AppWizard erzeugtes Applikationsgerüst aussehen könnte. Das Beispiel zeigt ein SDI Applikationsgerüst ohne Datenbank- und OLE-Dokument Unterstützung.

[13] Application Wizard

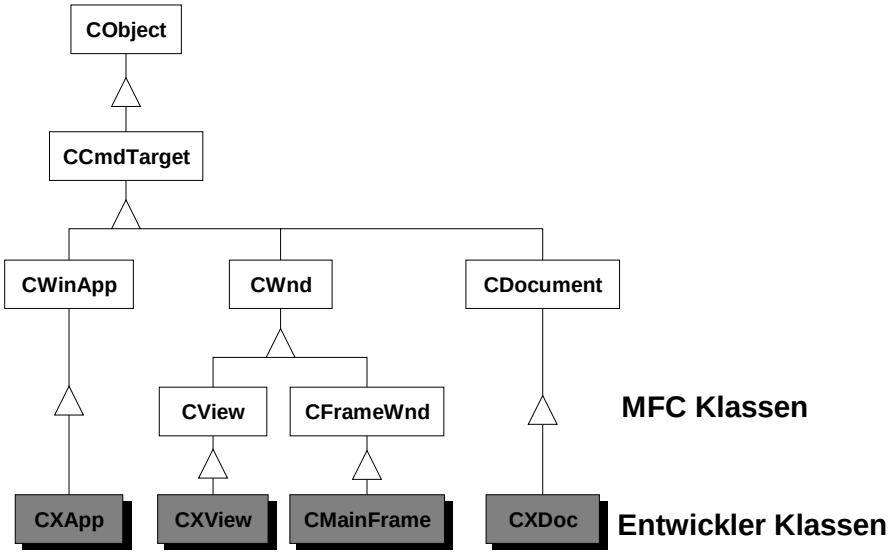


Abbildung 24: Vom AppWizard erzeugtes Applikationsgerüst

Ein solches vom AppWizard erzeugtes Applikationsgerüst kann der Entwickler mit dem C++ Compiler übersetzen. Alle verwendeten Komponenten der Document/View Architektur implementieren das Standardverhalten einer SDI oder MDI Applikation. Die erstellte Binärdatei ist sofort lauffähig, daher ist es möglich, daß die prototypische Applikation bereits betrachtet und erprobt werden kann. Der Entwickler kann dadurch einen frühen Eindruck von Funktionalität und Handhabung seiner Applikation gewinnen.

Innerhalb der Visual C++ Entwicklungsumgebung gibt es für die verschiedenen Projektarten unterschiedliche AppWizards. In diesem Abschnitt wurde der AppWizard für die Konstruktion von Applikationen mit dem MFC Rahmenwerk vorgestellt. Die Visual C++ Umgebung ermöglicht zusätzlich die Entwicklung eigener AppWizards, um die Erzeugung eines Gerüsts den Bedürfnissen eigener Projekte anpassen zu können. Die Entwicklung eigener AppWizards wird wiederum durch einen (Meta) AppWizard unterstützt (siehe [Tot96]).

6.2 Das ClassWizard Entwicklungswerkzeug

Das ClassWizard Entwicklungswerkzeug unterstützt den Entwickler bei der Verwendung der MFC Rahmenwerk Technologien und bei der Ableitung neuer Klassen aus dem Rahmenwerk. Bei den Technologien handelt es sich um die Message-, Dispatch- und Connection-Maps sowie die Dialog-Data-Exchange (DDX) Technologie. Die Message- und Dispatch-Maps haben wir bereits in den Abschnitten 4.6.2 und 5.4 kennengelernt.

Dialog-Data-Exchange ist der Oberbegriff für eine Menge von Transferfunktionen, die Daten zwischen Controls und Exemplarvariablen transferieren können (siehe [Oni96b]). Die Connection-Maps werden verwendet, um Events von ActiveX Controls zu verwalten. Da bei der Umsetzung des Pausenplanersystems diese Technologie nicht verwendet wurde, werde ich auch in diesem Teil der Arbeit nicht weiter auf diesen Teilbereich des ClassWizard Entwicklungswerkzeugs eingehen.

Auf Abbildung 25 ist die Oberfläche des ClassWizard Entwicklungswerkzeugs abgebildet. Man kann erkennen, daß das Werkzeug durch die Verwendung von „Karteireitern“ in fünf Teilbereiche aufgeteilt wird. Jeder Teilbereich, ausgenommen der Karteireiter „Class Info“, unterstützt dabei die Verwendung einer MFC Rahmenwerk Technologie. Im folgenden wird die Handhabung der einzelnen Teilbereiche des ClassWizard detaillierter beschrieben.

6.2.1 Karteireiter „Message Maps“

Die Auswahl des Karteireiters „Message Maps“ aktiviert den Message-Map Teilbereich des ClassWizard Werkzeugs. Hier kann der Entwickler neue Message-Map Einträge erstellen oder bestehende Einträge löschen. Das Erstellen eines neuen Eintrags bewirkt zusätzlich, daß der Rumpf einer nachrichtenverarbeitenden Funktion der ausgewählten Klasse hinzugefügt wird. Ein Beispiel soll die Arbeitsweise mit dem ClassWizard verdeutlichen.

In Abbildung 25 kann man sehen, wie die „CAufsichtsorteBearbeiterView“ Klasse des Aufsichtsorte-Bearbeiter Werkzeugs bearbeitet wird. Die Oberfläche enthält ein Textfeld, in dem der Benutzer eine Personenanzahl angeben kann. Nachdem der Benutzer die Zahl eingegeben hat und das Textfeld verläßt, möchte der Entwickler auf dieses Ereignis innerhalb einer Methode der View Klasse reagieren. Zu diesem Zweck selektiert der Entwickler in der „Object IDs“ Liste den Namen „IDC_EDIT_VERTRETER“ des beteiligten Textfeldes.

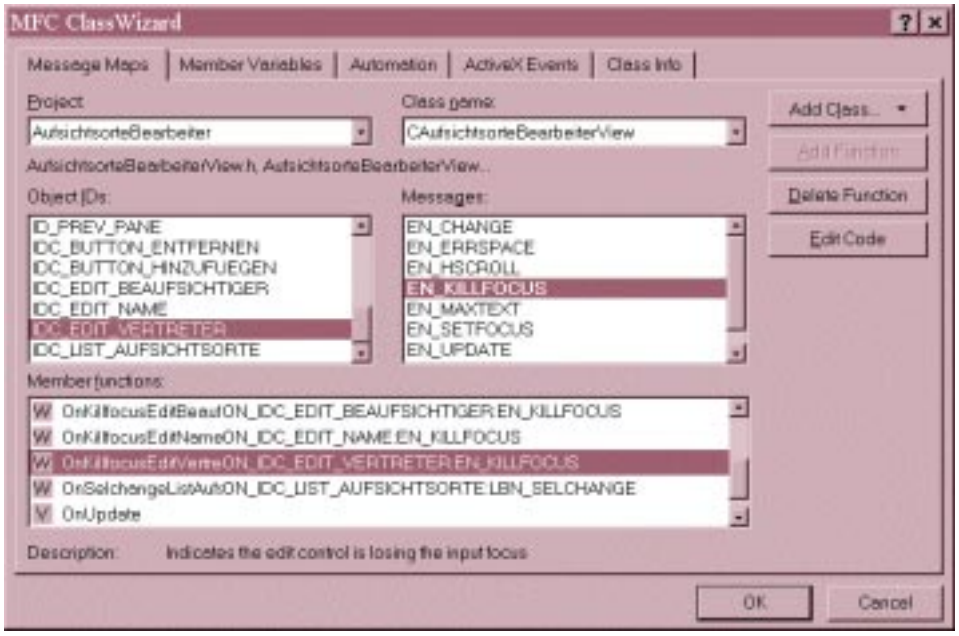


Abbildung 25: Der ClassWizard verwaltet Message-Maps

In der "Messages" Liste rechts daneben werden daraufhin alle Nachrichten aufgelistet, die das Textfeld versenden kann. Nun wählt man in der Liste die Nachricht „EN_KILLFOCUS“ aus, welche versendet wird wenn der Benutzer den Textbereich des Feldes verläßt. Ein Druck auf den „Add Funktion“ Knopf fügt einen neue Methodenrumpf in die CAufsichtsorteBearbeiterView Klasse ein. Gleichzeitig wird der Eintrag in die Message-Map dieser Klasse getätigt. In der unteren "Member functions" Liste des ClassWizard sieht man, daß der vom Entwicklungswerkzeug vergebene Methodenname *OnKillfocusEditVertreter()* lautet. Mit dem „Delete Function“ Knopf könnte der Entwickler jetzt diese behandelnde Methode aus der Klasse entfernen und den Eintrag aus der Message-Map löschen.

6.2.2 Karteireiter „Member Variables“

Die Auswahl des Karteireiters „Member Variables“ (Exemplarvariablen) aktiviert den Dialog-Data-Exchange Teilbereich des ClassWizard Werkzeugs. CDialog und CFormView sind Klassen des MFC Rahmenwerks, die mehrere Controls innerhalb eines Fensters darstellen und verwalten können. Die Dialog-Data-Exchange (DDX) Technologie ermöglicht durch ihre Transferfunktionen den Datenaustausch zwischen den Controls und den Variablen der verwaltenden Klasse. Zusätzlich können die Werte durch die Dialog-Data-Validation (DDV) Funktionen auf bestimmte Kriterien hin überprüft werden. Mit Hilfe des ClassWizard kann der Entwickler Exemplarvariablen und zugehörige Transfer-/Validationsfunktionen erstellen und löschen.

Abbildung 26 zeigt, wie die Exemplarvariablen der „CAufsichtsorteBearbeiterView“ Klasse des Aufsichtsorte-Bearbeiter Werkzeugs im ClassWizard aufgelistet werden. In der linken Spalte der Liste ist der Name des Controls angegeben, in der mittleren Spalte steht der verwendete Typ und in der rechten Spalte der dazugehörige Name der Exemplarvariable. Zu einem Control können mehrere Exemplarvariablen erstellt werden, was durchaus sinnvoll ist. Betrachten wir dazu als Beispiel das „IDC_EDIT_VERTRETER“ Textfeld. In der CAufsichtsorteBearbeiterView Klasse gibt es zwei Exemplarvariablen, die mit diesen Control verknüpft sind.

Die *m_AnzahlVertreter* Variable besitzt den Typ unsigned integer[14]. Ihr Wert repräsentiert die Eingabe des Benutzers in das Vertreter-Textfeld. Die Transferfunktion sorgt dafür, daß der eingegebene Text in einen positiven ganzzahligen Wert umgewandelt wird, sie überprüft auch ob die Umwandlung überhaupt möglich ist. Zusätzlich können Maximal- und Minimalwerte angegeben werden, diese werden in den Validationsfunktionen überprüft.

Die *m_AnzahlVertreter_Feld* Variable besitzt den Typ CEdit. Über diese Exemplarvariable kann auf den Control selbst zugegriffen werden. Beispielsweise könnte die Eingabe in das Textfeld deaktiviert werden, um dem Benutzer temporär keine Eingabe zu ermöglichen.

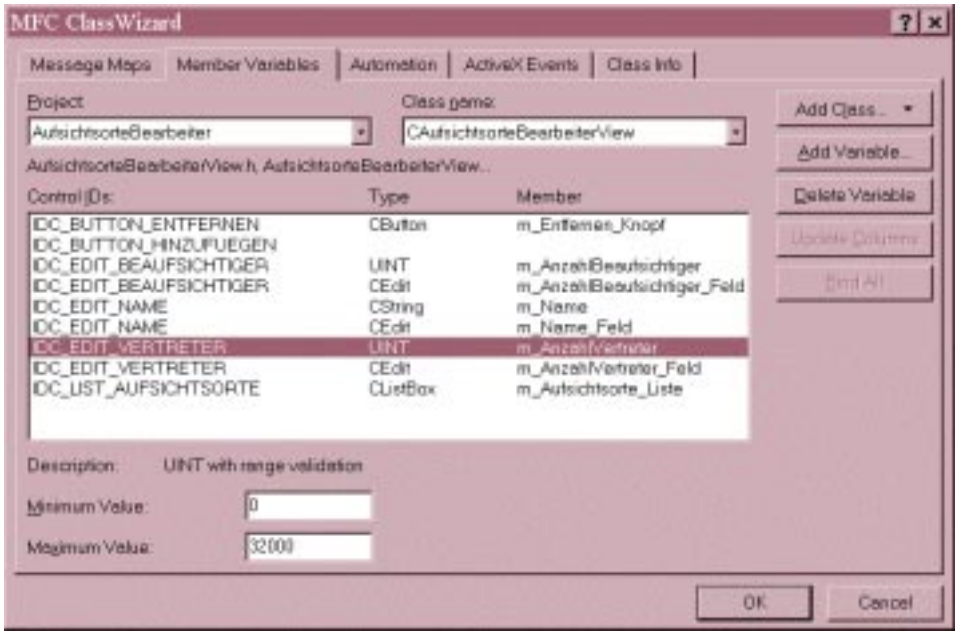


Abbildung 26: Der ClassWizard verwaltet Variablen für Controls

Die Transfer- und Validationsfunktionen fügt der ClassWizard in die *DoDataExchange(...)* Methode der ausgewählten Klasse ein. Im unteren Text ist die *DoDataExchange(...)* Methode des Aufsichtsorte-Bearbeiter Views dargestellt. Der Programmcode entspricht dem Zustand des ClassWizard, wie er in Abbildung 26 zu sehen ist.

```
void CAufsichtsorteBearbeiterView::DoDataExchange(CDataExchange* pDX)
{
 CFormView::DoDataExchange(pDX);
 //{{AFX_DATA_MAP(CAufsichtsorteBearbeiterView)
 DDX_Control(pDX, IDC_EDIT_VERTRETER, m_AnzahlVertreter_Feld);
 DDX_Control(pDX, IDC_EDIT_NAME, m_Name_Feld);
 DDX_Control(pDX, IDC_EDIT_BEAUFSICHTIGER, m_AnzahlBeaufsichtiger_Feld);
 DDX_Control(pDX, IDC_LIST_AUFSICHTSORTE, m_Aufsichtsorte_Liste);
 DDX_Control(pDX, IDC_BUTTON_ENTFERNEN, m_Entfernen_Knopf);
 DDX_Text(pDX, IDC_EDIT_NAME, m_Name);
 DDV_MaxChars(pDX, m_Name, 20);
 DDX_Text(pDX, IDC_EDIT_BEAUFSICHTIGER, m_AnzahlBeaufsichtiger);
 DDV_MinMaxUInt(pDX, m_AnzahlBeaufsichtiger, 0, 32000);
 DDX_Text(pDX, IDC_EDIT_VERTRETER, m_AnzahlVertreter);
```

[14] Das MFC Rahmenwerk verwendet aus Gründen der Portabilität zwischen 32 und 16 Bit Betriebssystemen das Makro "UINT".

```
  DDV_MinMaxUInt(pDX, m_AnzahlVertreter, 0, 32000);
  //}}AFX_DATA_MAP
}
```

Wie man oben sieht, verwendet die DDX Technologie in der *DoDataExchange(...)* Methode ein CDataExchange Objekt, das Kontextinformationen wie Transferrichtung und verwaltendes Fenster für die Transfer-/Validationsfunktionen festlegt. Die *DoDataExchange(...)* Methode wird aus der Applikation nie direkt aufgerufen, statt dessen wird die *CWnd::UpdateData(bool)* Methode verwendet, deren Parameterwert die Transferrichtung angibt. *UpdateData(true)* speichert den aktuellen Zustand der Controls in den Exemplarvariablen und *UpdateData(false)* verwendet man, um die Werte der Exemplarvariablen den Controls zu übermitteln. Die *UpdateData(...)* Methode erzeugt und initialisiert das CDataExchange Objekt, sie behandelt auch die Ausnahmen, die in den Transfer- und Validationsfunktionen ausgelöst werden können.

6.2.3 Karteireiter „Automation“

Die Auswahl des Karteireiters „Automation“ aktiviert den Automationsteilbereich des ClassWizard Werkzeugs. Hier kann der Entwickler Methoden und Eigenschaften einer Automationsklasse erstellen und löschen. Der ClassWizard verwaltet dabei die Einträge in der Dispatch-Map und aktualisiert zusätzlich die Texte der ODL Schnittstellenbeschreibung. In Abbildung 27 sind die Automationsmethoden des Aufsichtsorte-Bearbeiter Werkzeug im ClassWizard aufgelistet.

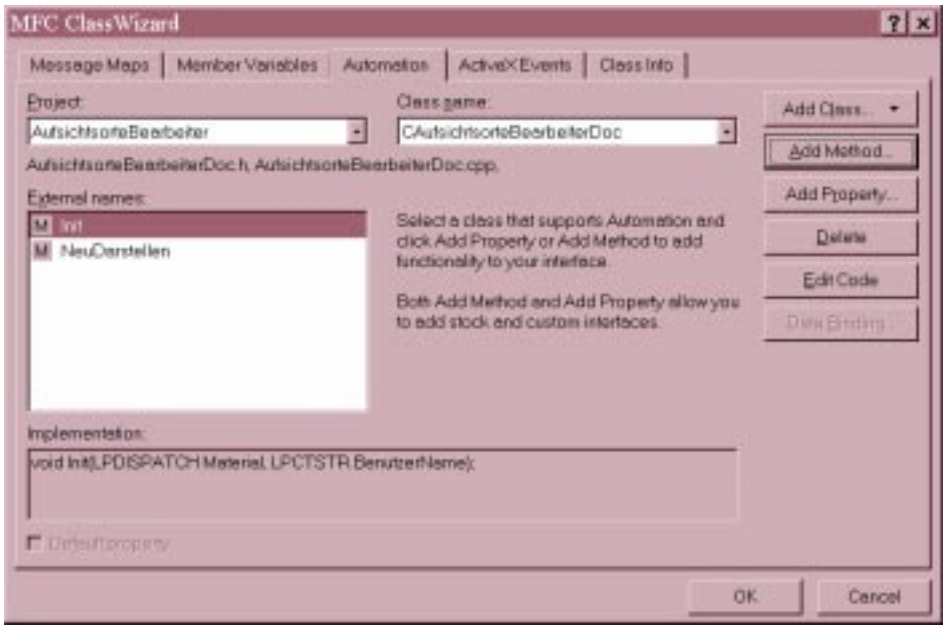


Abbildung 27: Der ClassWizard verwaltet Automationsmethoden

Aus den beiden Methoden *Init(...)* und *NeuDarstellen()* erstellt der ClassWizard in der CAufsichtsorteBearbeiterDoc Klasse folgende Dispatch-Map:

```
BEGIN_DISPATCH_MAP(CAufsichtsorteBearbeiterDoc, CDocument)
  //{{AFX_DISPATCH_MAP(CAufsichtsorteBearbeiterDoc)
  DISP_FUNCTION(CAufsichtsorteBearbeiterDoc, "NeuDarstellen", NeuDarstellen,
VT_EMPTY,VTS_NONE)
  DISP_FUNCTION(CAufsichtsorteBearbeiterDoc, "Init", Init, VT_EMPTY,
VTS_DISPATCH VTS_BSTR)
  //}}AFX_DISPATCH_MAP
END_DISPATCH_MAP()
```

6.3 Der Dialogeditor

Der Dialogeditor ermöglicht dem Entwickler die visuelle Oberflächengestaltung seiner Applikation. Bei der Umsetzung des Pausenplanersystems konnten wir den Dialogeditor gewinnbringend einsetzen, daher möchte ich in diesem Teil meiner Arbeit dieses nützliche Entwicklungswerkzeug kurz beschreiben.

CDialog und CFormView sind Klassen des MFC Rahmenwerks, die mehrere Controls in einem Fenster darstellen und verwalten können. Informationen über die Anordnung, Größe und Attributwerte der Controls, speichern beide Klassen in einem Programmiersprachen unabhängigen Text namens „Dialog-Template“. Mit dem Dialogeditor-Werkzeug kann der Entwickler Dialog-Templates auf visuellem Wege bearbeiten.

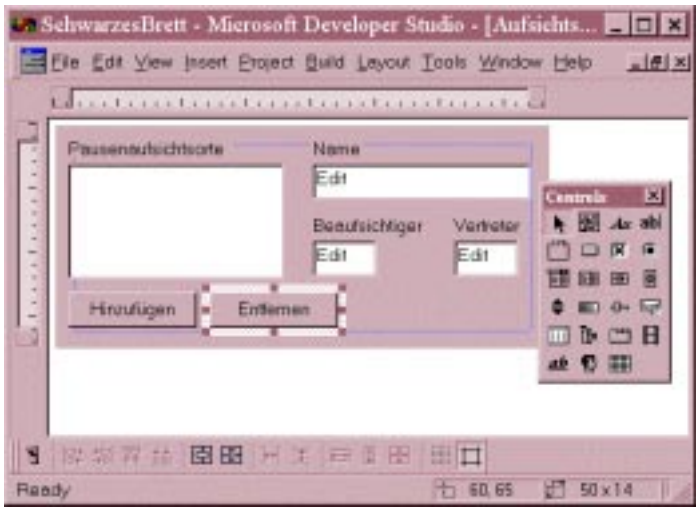


Abbildung 28: Der Dialogeditor

In Abbildung 28 sieht man die Oberfläche des Dialogeditors. Im zentralen Fenster befindet sich dort der Dialog-Template der Aufsichtsorte-Bearbeiter Werkzeugoberfläche gerade in der Bearbeitung. Der Entwickler kann von der Control Palette, die sich im kleinen Fenster auf der rechten Seite befindet, ein Control mittels Drag&Drop im Dialog-Template anordnen. Größe und Attribute der Controls können verändert werden. Zum Abschluß der Bearbeitung kann der Entwickler Erscheinungsbild und Verhalten des Dialog-Templates testen. Ist er mit dem Ergebnis des Tests zufrieden, so verknüpft er das von ihm erstellte Dialog-Template mit einer von CFormView oder CDialog abgeleiteten Klasse seiner Applikation. Die Dialog-Template Informationen werden in einer Programmiersprachen unabhängigen Ressource Datei gespeichert.

Die untenstehende Ressource Datei beschreibt das Dialog-Template der Aufsichtsorte-Bearbeiter Werkzeugoberfläche. Ein Vorteil der Dialog-Template Technik ist die Unabhängigkeit von der Programmiersprache, so daß Informationen über Anordnung, Größe und Attributwerte von Controls in verschiedenen Sprachen verwendet werden können. Ein zusätzlicher Vorteil dabei ist, daß die Informationen nicht im Quelltext einer Klasse der Applikation beschrieben werden müssen.
Ressource Datei für Aufsichtsorte-Bearbeiter Dialog-Template:

```
IDD_AUFSICHTSORTEBEARBEITER_FORM DIALOG DISCARDABLE  0, 0, 192, 87
STYLE WS_CHILD
FONT 8, "MS Sans Serif"
BEGIN
    LISTBOX         IDC_LIST_AUFSICHTSORTE,5,15,84,45,LBS_SORT |
                    LBS_NOINTEGRALHEIGHT | WS_VSCROLL | WS_TABSTOP
    EDITTEXT        IDC_EDIT_NAME,100,15,85,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT_BEAUFSICHTIGER,100,45,25,14,ES_AUTOHSCROLL
    EDITTEXT        IDC_EDIT_VERTRETER,155,45,25,14,ES_AUTOHSCROLL
    PUSHBUTTON      "Hinzufügen",IDC_BUTTON_HINZUFUEGEN,5,65,50,14
    PUSHBUTTON      "Entfernen",IDC_BUTTON_ENTFERNEN,60,65,50,14
    LTEXT           "Name",IDC_STATIC,100,5,20,8
    LTEXT           "Beaufsichtiger",IDC_STATIC,100,35,46,8
    LTEXT           "Pausenaufsichtsorte",IDC_STATIC,5,5,65,8
    LTEXT           "Vertreter",IDC_STATIC,155,35,30,8
END
```

6.4 Zusammenfassung

Dieses Kapitel hat die wichtigsten Entwicklungswerkzeuge für das MFC Rahmenwerk vorgestellt. Außer AppWizard, ClassWizard und Dialog Editor gibt es noch viele weitere nützliche Entwicklungswerkzeuge innerhalb der Visual C++ Entwicklungsumgebung. Erwähnen möchte ich an dieser Stelle den Ressource Editor, der unter anderem die Konstruktion von Menüs, Symbolleisten und Icons erleichtert.

Außerdem unterstützen die Entwicklungswerkzeuge die Vererbung von Klassen aus dem MFC Rahmenwerk mit einem Speziellen Assistenten. Dieser erzeugt die C++ Quelldateien der Klasse und die trägt auch die benötigten Makros für Message-, Dispatch-Map ein.

Zusammenfassend lassen sich folgende Punkte für dieses Kapitel aufstellen:

- Der Zusammenhang zwischen MFC Entwicklungswerkzeugen und dem MFC Rahmenwerk ist stark ausgeprägt.
- Die Verwendung der Werkzeuge ermöglicht dem Entwickler einen sicheren und effizienten Umgang mit dem MFC Rahmenwerk.
- Die gesamte Entwicklungsumgebung vereinfacht die Arbeit des Entwicklers, da sie eine abstrakte Sichtweise auf das Rahmenwerk ermöglichen.
- Die textuelle Beschreibung von Schnittstellen, Maps und Klassen weicht einer bildlichen Darstellung innerhalb der Entwicklungswerkzeuge.

7 Werkzeugkonstruktion mit dem MFC Rahmenwerk

Werkzeuge sind Arbeitsmittel im Rahmen einer Aufgabenerledigung. Der Benutzer verwendet sie, um mit ihrer Hilfe ein Arbeitsergebnis zu erreichen. Ein Werkzeug stellt sich selbst und die Materialien, die es bearbeitet, in einer Form dar, die dem Benutzer eine interaktive Bearbeitung der Materialien ermöglicht (siehe Abschnitt 2.1).

Dieses Kapitel beschreibt die Konstruktion von Werkzeugen nach WAM unter Verwendung des MFC Rahmenwerks. Hauptziel der hier vorgestellten Werkzeugkonstruktion ist die einfache Integration der Entwurfskomponenten in das MFC Rahmenwerk sowie gute Handhabbarkeit durch den Entwickler.

Bereits bei der Analyse des MFC Rahmenwerks wurde deutlich, daß man einen Kompromiß zwischen den Vorgaben des MFC Rahmenwerks und der Werkzeugkonstruktion nach WAM finden muß. Das MFC Rahmenwerk unterstützt die Konstruktion dokumentenbasierter Anwendungen und stellt dem Anwendungsentwickler für diese Aufgabe die Document/View Architektur zur Verfügung. Die WAM Entwurfsmetapher Werkzeug wird durch das IAK-FK und das Werkzeugkompositions Entwurfsmuster verwirklicht (siehe Abschnitt 2.2 und 2.3).

Die hier vorgestellte Idee für die Werkzeugkonstruktion nach WAM kann sicherlich nicht alle Anforderungen der WAM Entwurfsmuster erfüllen. Sie enthält jedoch eine praktikable Lösung, die die Verwendung des MFC Rahmenwerks auf ein methodisches Fundament stellt und es dem Entwickler ermöglicht, die Hilfsmittel der Visual C++ Entwicklungsumgebung einzusetzen.

7.1 Verwendung der Document/View Architektur

Den Kern des MFC Rahmenwerks bildet die Document/View Architektur, deren eingehende Beschreibung sich in Kapitel 4.5 befindet. Die Hauptaufgabe der Document/View Architektur ist die Trennung von Interaktion und Funktion innerhalb einer Applikation. Sie realisiert diese Trennung durch Verwendung mehrerer Komponenten, welche die Interaktion mit dem Benutzer umsetzen, beziehungsweise die Funktionalität der Applikation bereitstellen.

Die Konstruktionsidee schlägt vor, diese Einteilung für die Werkzeugkonstruktion nach WAM zu verwenden. Eine erste Gegenüberstellung der Komponenten aus Document/View Architektur und FK-IAK Entwurfsmuster (siehe Abschnitt 2.2) ergibt folgende Zuordnung:

View-Komponente und Frame-Komponente ≈ IAK
Document-Komponente ≈ FK

Auf den ersten Blick haben sowohl die View- und Frame- als auch die Document-Komponente ähnliche Aufgaben, wie die ihnen zugeordneten Komponenten des FK-IAK Entwurfsmusters. Die Document-Komponente entspricht dem bewirkenden und sondierenden Teil der Applikation (Werkzeug) und in ihr wird die fachliche Funktionalität der Applikation festgelegt. View- und Frame-Komponenten legen die Benutzungsoberfläche der Applikation fest, steuern die Präsentation an der Oberfläche und nehmen Nachrichten vom Windows Fenstersystem entgegen.

Es scheint zunächst etwas unverständlich zu sein, daß in der Konstruktionsidee die Document-Komponente der Funktionskomponente gegenübergestellt wird. Im WAM Kontext wird man den Begriff Dokument eher mit Materialien assoziieren. Betrachtet man die Aufgaben der Document-Komponente in einer Applikation, so wird verständlich, daß der Name Document für diese im Grunde falsch gewählt wurde. In der Document-Komponente implementiert der Entwickler die Funktionalität der Applikation. Normalerweise benutzt die Document-Komponente dabei Klassen, welche die Funktionalität eines bearbeiteten Dokuments kapseln.

Betrachtet man die Frame-Komponente der Document/View Architektur genauer, so stellt sich heraus, daß sie mit dem Menü sowie den Symbol- und Statusleisten nicht geringe Teile der Oberfläche verwaltet. Es scheint dabei so, als würden die Aufgaben der IAK auf zwei verschiedene Komponenten der Document/View Architektur aufgeteilt werden.

Eine weitere Schwierigkeit ergibt sich aus der Tatsache, daß die View- und Frame-Komponenten auf direktem Wege Nachrichten vom Windows Fenstersystems erhalten (siehe Abbildung 29). Nach dem IAK-FK Entwurfsmuster sollte eine IAK von den konkreten Handhabungsformen und dem verwendeten Fenstersystem abstrahieren. Zu diesem Zweck verwendet sie Interaktionstypen, die die Ein-/Ausgabekomponenten des Fenstersystems kapseln. Nachrichten des verwendeten Fenstersystem werden dadurch in den Interaktionstypen abgefangen und erreichen nicht die IAK.

Sowohl die View- als auch die Frame-Komponente abstrahieren nicht von dem verwendeten Fenstersystem. Sie sind in das System integriert und übernehmen Aufgaben, die man im IAK-FK Entwurfsmuster einem IAT zuordnen könnte. Die regulären Nachrichten des Windows Fenstersystems werden jedoch bereits in den Basisklassen der beiden Komponenten bearbeitet, so daß innerhalb der Schnittstellen der abgeleiteten Klassen keine bearbeitenden Methoden für das Fenstersystem auftauchen.

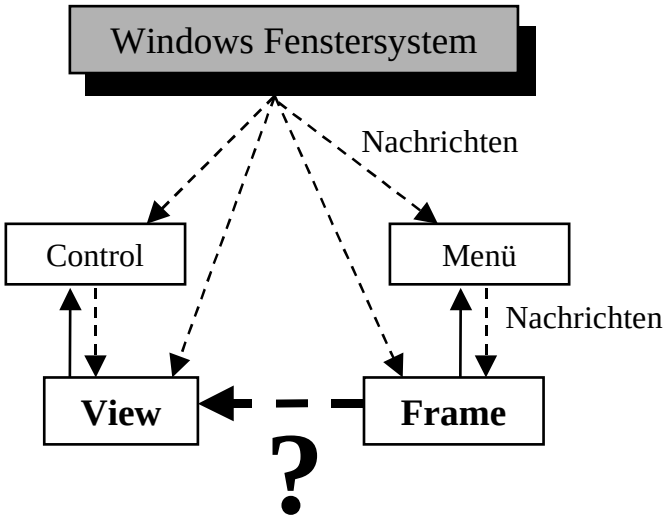


Abbildung 29: Nachrichten des Window Fenstersystems

Abbildung 29 zeigt, daß die von View und Frame verwenden Komponenten ebenfalls Nachrichten des Windows Fenstersystems erhalten. Die View-Komponente verwendet in diesem Beispiel einen Control und die Frame-Komponente ein Menü. Dabei sind mögliche Status- und Symbolleisten der Frame-Komponente in der Abbildung nicht dargestellt.

Die von View und Frame verwendeten Komponenten kapseln die Ein-/Ausgabekomponenten des Windows Fenstersystems. Sowohl die Beziehung zwischen Control und View-Komponente als auch die Beziehung zwischen Menü und Frame-Komponente ähnelt dabei der Beziehung zwischen IAT und IAK im IAK-FK Entwurfsmuster.

Es ist für die Werkzeugkonstruktion unvorteilhaft, daß die Bearbeitung von Nachrichten beziehungsweise Kommandos sowohl in der Frame- als auch in der View-Komponente stattfindet. Gerade die Bearbeitung von Kommandos sollte ausschließlich innerhalb der View-Komponente stattfinden. Dadurch können sämtliche Aufgaben, die die Handhabung und Präsentation des Werkzeugs betreffen, in einer Komponente angeordnet werden. Wir benötigen demnach eine Möglichkeit, die Kommandos von der Frame-Komponente an die View-Komponente weiterzuleiten.

Der Routingmechanismus des MFC Rahmenwerks (siehe Abschnitt 1.1) gibt jeder Komponente der Document/View Architektur die Möglichkeit, auf ein vom Benutzer ausgelöstes Kommando zu reagieren. Man kann den Routingmechanismus demnach benutzen, um sämtliche Kommandos innerhalb der View-Komponente zu bearbeiten.

Ein Beispiel soll dieses Vorgehen beschreiben. Der Benutzer hat im Menü den Menüpunkt „Info“ Ausgewählt, daraufhin erhält die Frame-Komponente ein „ID_INFO“ Kommando, das sie auf direktem Wege an die View-Komponente weiterleitet. Innerhalb der View-Komponente kann nun auf die Auswahl des Menüpunkts reagiert werden.

Abbildung 30: Verwendung von Kommandos

Durch den Routingmechanismus des MFC Rahmenwerks kann man die Frame-Komponente als Teil der IAK verwenden, ohne daß die Bearbeitung von Kommandos auf mehrere Komponenten aufgeteilt werden muß. Innerhalb der View-Komponente können dadurch sämtliche Aufgaben, welche die Handhabung und Präsentation des Werkzeugs betreffen, angeordnet werden.

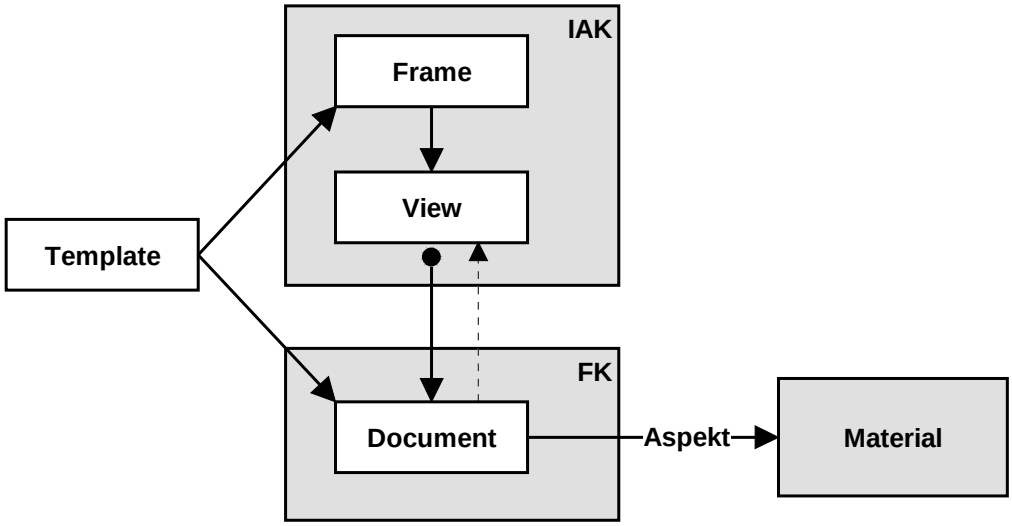


Abbildung 31: Zuordnung von Document/View Komponenten zu IAK und FK

In der Abbildung 31 möchte ich die Konstruktionsidee abschließend verdeutlichen. Die Abbildung zeigt, wie die Frame-, View- und Document-Komponenten genutzt werden, um die Trennung von Interaktion und Funktion des IAK-FK Entwurfsmusters umzusetzen. View- und Frame-Komponente übernehmen dabei gemeinsam die Aufgaben der Interaktionskomponente (IAK). Die Document-Komponente übernimmt die Aufgaben der Funktionskomponente (FK). Sie implementiert die Operationen zur Bearbeitung des Materials, wobei sie gegebenenfalls über einen Aspekt auf das Material zugreift. Details der Werkzeug und Material Kopplung im Pausenplanersystem werden in Abschnitt 8.4 beschrieben.

Die Template-Komponente repräsentiert als technische Hülle die Gesamtheit des Werkzeugs, indem sie Document-, View- und Frame-Komponente zu einer Einheit zusammenfügt. Sie erzeugt, initialisiert und verwaltet die ihr zugeordneten Werkzeugkomponenten. Die Verwendung der Applikations-Komponente wird im Abschnitt 7.5 beschrieben.

In diesem Abschnitt habe ich beschrieben, wie die zentralen Komponenten der Document/View Architektur den Komponenten des IAK-FK Entwurfsmusters zugeordnet werden können. Weiterhin habe ich gezeigt, wie man es mit Hilfe des Routingmechanismus vermeiden kann, daß die Bearbeitung von Kommandos auf View- und Frame-Komponente aufgeteilt werden muß. Der nächste Abschnitt beschreibt, wie man die Kopplung zwischen IAK und FK unter Verwendung des Beobachtermusters der Document/View Architektur realisieren kann.

7.2 Das Beobachtermuster der Document/View Architektur

Das Beobachtermuster der Document/View realisiert einen Reaktionsmechanismus, der eine lose Kopplung zwischen den abhängigen Komponenten der Architektur erlaubt. Dieser Abschnitt gibt dem Leser zunächst eine allgemeine Darstellung des Beobachtermusters. Daraufhin folgt eine Beschreibung, wie dieses Beobachtermuster innerhalb der Document/View Architektur implementiert worden ist und wie man diese Implementierung nach der oben beschriebenen Konstruktionsidee verwenden kann.

Der Zweck des Beobachtermusters wird in [GHJ+96] wie folgt beschrieben:

> „Definiere eine 1-zu-n-Abhängigkeit zwischen Objekten, so daß die Änderung des Zustands eines Objekts dazu führt, daß alle abhängigen Objekte benachrichtigt und automatisch aktualisiert werden."

Im Beobachtermuster nennt man das zentrale Objekt „Subjekt" und die von ihm abhängigen Objekte „Beobachter". Man sagt: „Die Beobachter beobachten das Subjekt". Ein Subjekt kann mehrere abhängige Beobachter haben, die benachrichtigt werden, sobald das Subjekt seinen Zustand ändert. Die Beobachter synchronisieren sich mit dem neuen Zustand des beobachteten Subjekts, indem sie Anfragen an die Schnittstelle des Subjekts stellen. Das Beobachtermuster dient der Aufrechterhaltung der Konsistenz zwischen Beobachtern und Subjekt, ohne daß die Objekte eng miteinander gekoppelt werden müssen. Abbildung 32 verdeutlicht diesen Zusammenhang:

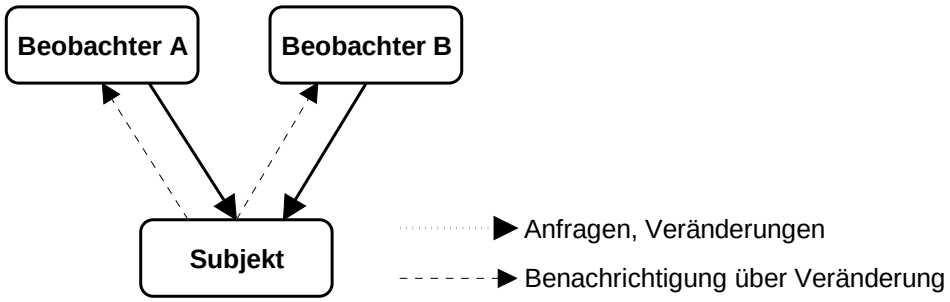


Abbildung 32: Beobachter und Subjekt

Verwendet man das Beobachtermuster innerhalb der Werkzeugkonstruktion um eine lose Kopplung von Interaktionskomponente und Funktionskomponente zu erreichen, so übernimmt die IAK die Rolle des Beobachters und die FK die Rolle des Subjekts (FK-IAK Entwurfsmuster siehe Abschnitt 2.2).

Abbildung 33 zeigt in einem Klassendiagramm die statischen Aspekte des Document/View Beobachtermusters. In der Abbildung ist die Verwendung des Musters für die Konstruktion des Pausenplaner Werkzeugs dargestellt.

Die Klassen CDocument und CView setzen das Konzept von Subjekt und Beobachter um. Über die Methoden *AddView()* und *RemoveView()* der CDocument Schnittstelle können sich die Beobachter (Views) beim Subjekt (Document) an- und abmelden. Dies geschieht während der Erzeugung beziehungsweise Zerstörung eines View Objekts automatisch. Bei einer Zustandsänderung im Document werden alle angemeldeten Views über die Methode *OnUpdate()* benachrichtigt. Dieser Vorgang wird innerhalb der Methode *UpdateAllViews()* implementiert. CDocument kennt seine Beobachter demnach nur über die CView Schnittstelle.

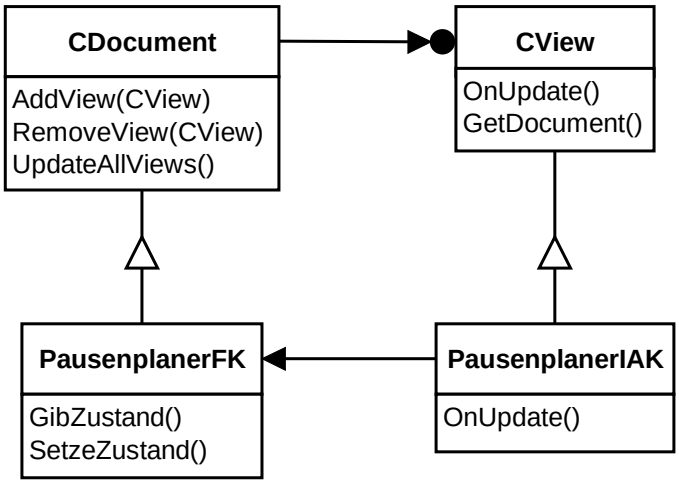


Abbildung 33: Verwendung des Document/View Beobachtermusters

Will man die Konstruktionsidee aus Abschnitt 7.1 umsetzen, so erbt die Funktionskomponente des Pausenplaners von CDocument. Die Interaktionskomponente des Pausenplaners erbt von CView und überschreibt dabei die Methode *OnUpdate()*, innerhalb derer die IAK den Zustand der FK sondiert, um daraufhin die Darstellung zu aktualisieren.

Durch Aufruf der Methode *GetDocument()* erhält eine IAK die Referenz auf ihre FK. Die Referenz wurde während der Erzeugung des CView Objekts durch den Aufruf der *CDocument::AddView()* Methode gesetzt.

Besonders bei mehreren Beobachtern ist es entscheidend, daß jede beobachtende IAK die FK bei der Sondierung in ein und demselben Zustand vorfindet. Dabei ist es wichtig zwischen sondierenden (z.B. *GibZustand()*) und verändernden Methoden (z.B. *SetzeZustand()*) an der Schnittstelle der FK strikt zu unterscheiden.

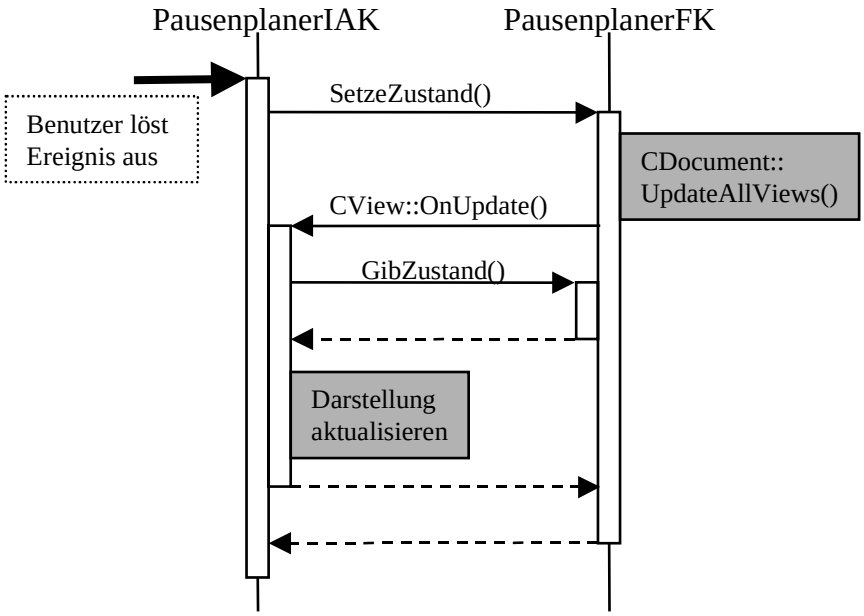


Abbildung 34: Interaktion im Document/View Beobachtermuster

Abbildung 34 zeigt in einem Interaktionsdiagramm den dynamischen Aspekt des Document/View Beobachtermusters. Abgebildet ist die Kommunikation zwischen Interaktionskomponente und Funktionskomponente des Pausenplaners, die nach dem Auslösen eines Ereignisses durch den Benutzer erfolgt.

7.3 Werkzeuginteraktion und View-Komponenten

In diesem Kapitel soll genauer auf die Konstruktion des interaktiven Teils eines Werkzeugs eingegangen werden. Dabei wird gezeigt, wie man mit dem MFC Rahmenwerk Interaktionskomponenten konstruieren kann. Beispiele aus dem Pausenplanersystem werden diesen Arbeitsabschnitt abrunden.

7.3.1 View-Komponenten

Innerhalb der Konstruktionsidee wurde beschrieben, daß die View-Komponente der Document/View Architektur die zentralen Aufgaben der IAK übernehmen soll. Die CView Klasse des MFC Rahmenwerks stellt die Oberklasse aller View-Komponenten der Document/View Architektur dar. Daher werde ich im folgenden Text auf die CView Klasse und ihre Schnittstelle näher eingehen, um die Unterschiede zu einer IAK darzustellen. Eine vollständige Tabelle aller Methoden der CView Klasse ist im Anhang 12 aufgeführt.

Betrachten wir zunächst, wo sich die CView Klasse innerhalb der Struktur des MFC Rahmenwerks befindet. In Abbildung 13 ist die Klassenhierarchie des gesamten MFC Rahmenwerks abgebildet. Abbildung 35 zeigt im Detail, welche Klassen im View-Rahmenwerk vorzufinden sind und wie die Klassen in Beziehung zueinander stehen. CView ist die Oberklasse des View-Rahmenwerks. Dies bedeutet, daß alle Klassen des Rahmenwerks direkt oder indirekt von der CView Klasse erben. Die Basisfunktionalität aller View-Komponenten, ist innerhalb der CView Klasse zusammengefaßt.

Die CView Klasse erbt die gesamte Implementation eines Windows Fensters von der Klasse CWnd. Sie hat keine abstrakten Methoden, ist demzufolge auch keine abstrakte Klasse. Dabei besitzen alle Methoden der CView Klasse Implementierungen, die das Standardverhalten eines Views in einer „typischen“ Windows Applikation umsetzen. Die Klasse ist damit direkt „einsetzbar“, ohne daß der Entwickler eine einzige Methode überladen muß.

Die Schnittstelle der CView Klasse beinhaltet neben den Methoden, die das Document/View Beobachtermuster umsetzen, zwei weitere Methodengruppen. Die Methoden der ersten Gruppe realisieren die graphische Ausgabe eines View, dazu zählt sowohl die Ausgabe auf dem Bildschirm als auch die Möglichkeit des Ausdrucks oder der Seitenansicht eines Dokuments. Das MFC Rahmenwerk verwendet dabei die Metapher der virtuellen Zeichenfläche, um die graphischen oder textuellen Informationen auf unterschiedlichen Geräten auszugeben zu können.

Mit den Methoden der zweiten Gruppe kann der Entwickler Drag&Drop Operationen innerhalb des Views implementieren. Durch die Verwendung von Drag&Drop Operationen kann der Austausch von Informationen zwischen Werkzeugen realisieren werden. Bei der Konstruktion des Pausenplanersystems hat sich die Verwendung der Drag&Drop Operationen innerhalb der Werkzeuge als vorteilhaft herausgestellt.

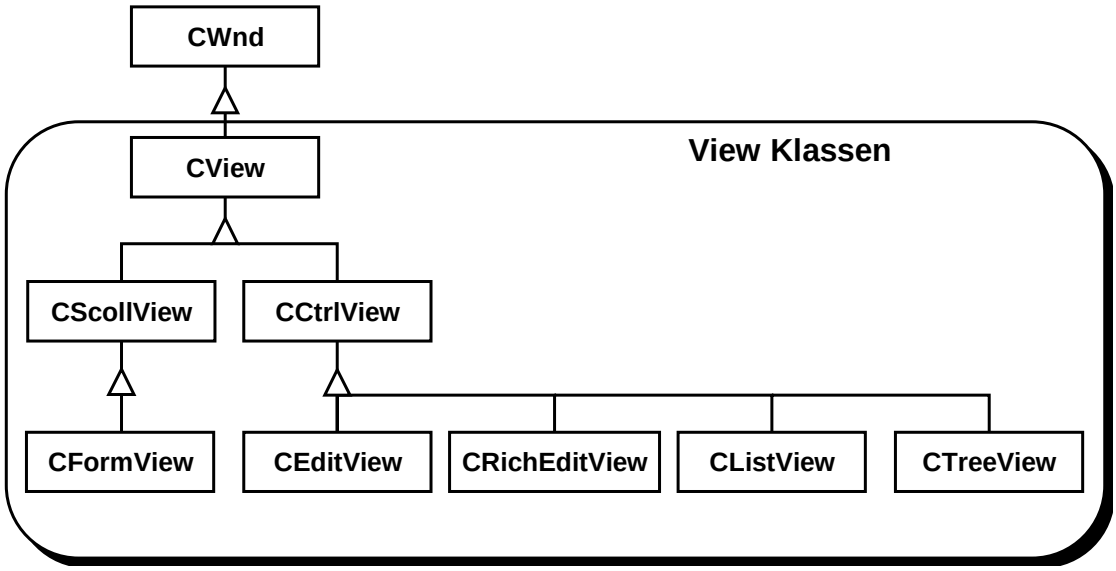


Abbildung 35: Das View-Rahmenwerk im Detail

Bei der Konstruktion der Werkzeuge des Pausenplanersystems wurden verschiedene Unterklassen von CView eingesetzt. Daher möchte ich nun genauer auf die Verwendung dieser Unterklassen eingehen, um die Unterschiede zu der gemeinsamen Basisklasse CView deutlich zu machen.

Sowohl die CFormView Klasse als auch die CCtrlView Klasse verwenden nicht das Konzept der virtuellen Zeichenfläche, sie ermöglichen statt dessen die Verwendung von Controls in einem Fenster. Controls sind die Ein-/Ausgabekomponenten des Windows Fenstersystem. Beide Klassen haben ihre eigene Vorgehensweise bei der Verwendung der Controls, die ich in den nun folgenden Unterkapiteln beschreiben möchte.

7.3.1.1 Formviews

Die CFormView Klasse ermöglicht die Verwendung mehrerer Controls in einem View. Die Anordnung der einzelnen Controls wird in einer sogenannten „Form“ festgelegt. Position und Größe eines Controls innerhalb einer Form kann der Entwickler mit dem Dialogeditor der Visual C++ Entwicklungsumgebung bestimmen und in einer Resource Datei speichern. Die CFormView Klasse liest bei ihrer Erzeugung die Informationen aus der Resource Datei und sorgt dafür, daß die Controls an der richtigen Stelle innerhalb der Form mit den korrekten Einstellungen erzeugt werden. Sie sorgt auch für die Zerstörung einzelnen Controls, wenn die Form nicht mehr benötigt wird. Ein Beispiel für die Verwendung der CFormView Klasse ist das Aufsichtsorte-Bearbeiter Werkzeug des Pausenplanersystems, dessen Werkzeugoberfläche in Abbildung 36 dargestellt ist.

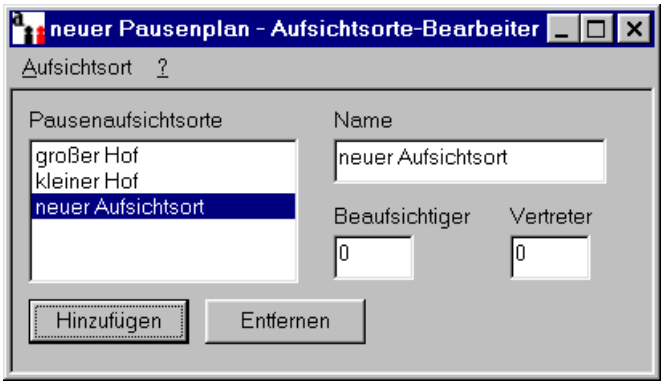


Abbildung 36: Das Aufsichtsorte-Bearbeiter Werkzeug

Um auf die Controls innerhalb der CFormView Klasse zugreifen zu können verwendet man Exemplarvariablen, die im Umfeld des MFC Rahmenwerks „Member Variables“ genannt werden. Der Entwickler kann mit dem ClassWizard Entwicklungswerkzeug der Visual C++ Umgebung diese Exemplarvariablen erstellen oder bestehende Exemplarvariablen löschen. Als Benutzer eines Controls will man nicht nur Controls verändern und Anfragen an sie stellen, man möchte auch benachrichtigt werden, sobald der Benutzer ein Ereignis an einem Control ausgelöst hat. Üblicherweise gibt es dann eine Event-Handler Funktion, die beim Auslösen des Ereignisses aufgerufen wird. Verknüpfungen zwischen Ereignis, Control und Event-Handler Funktion können auf einfache Weise mit dem ClassWizard Entwicklungswerkzeug erzeugt werden. Eine einführende Beschreibung der Funktionalität dieses Werkzeugs kann man in Kapitel 6 nachlesen.

7.3.1.2 Controlviews

Die CCtrlView Klasse ermöglicht die Verwendung eines Controls in einem Fenster. Sie kapselt in sich die Erzeugung und Initialisierung des Controls. Der Entwickler wird die CCtrlView Klasse nie direkt verwenden, statt dessen verwendet er die Klassen, die von ihr erben. Dies sind die Klassen CEditView, CRichEditView, CListView und CTreeView (siehe Abbildung 35). Im folgenden Text möchte ich auf diese Klassen näher eingehen.

Die CListView Klasse kann eine Menge von Elementen mit Elementnamen und Icon darstellen. Dabei besitzt sie verschiedene Möglichkeiten der Anordnung, die bestimmt, wo die Elemente innerhalb des Fensters dargestellt werden und wieviel Information dem Benutzer über ein Element präsentiert wird.

Ein Beispiel für die Verwendung der CListView Klasse ist die Schwarzes Brett Handhabungskomponente, welche zur Gruppenarbeitsumgebung des Pausenplanersystems gehört und daher nicht Teil eines Werkzeugs ist. Trotzdem habe ich das Schwarze Brett als Beispiel gewählt, damit der Leser sehen kann, daß die Nachkommen der CView Klasse auch für andere Komponenten des Pausenplanersystems verwendet worden sind. Kapitel 8 beschreibt die Konstruktion von Gruppenarbeitsumgebung und Materialien.

Die Oberfläche der Handhabungskomponente ist in Abbildung 37 dargestellt. Die Elemente repräsentieren hier die Materialien, Materialstapel und Werkzeuge, die am Schwarzen Brett angeordnet sind. Der Benutzer kann die Anordnung der Elemente frei bestimmen und Elemente durch eine Drag&Drop Operation einander zuordnen.

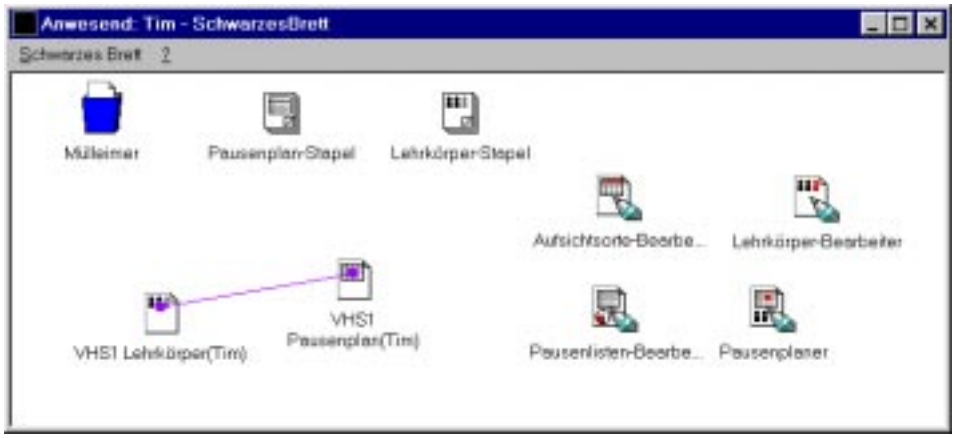


Abbildung 37: Die Schwarzes Brett Handhabungskomponente

Der Tree Control ähnelt in seiner Funktionalität dem List Control, im Unterschied zu ihm ordnet er die Elemente jedoch hierarchisch an. Die CEditView und CRichEditView Klassen stellen die Funktionalität eines Texteditors zur Verfügung. CEditView stellt Funktionen für den Druck sowie Suchen/Ersetzen zur Verfügung. Die CRichEditView Klasse erweitert die Funktionalität der CEditView Klasse um Text- und Absatzformatierung sowie die Einbettung von externen Dokumenten.

7.3.2 Controls

Innerhalb der Microsoft Dokumentation werden die Ein-/Ausgabekomponenten des Windows Fenstersystems „Controls“ genannt. Das MFC Rahmenwerk kapselt diese E/A-Komponenten innerhalb von Controlklassen. Dabei drängt sich ein Vergleich zwischen den Controlklassen und den Interaktionstypen des IAK-FK Entwurfsmusters auf. Controls stellen demjenigen der sie verwendet, verändernde und sondierende Methoden an ihrer Schnittstelle zur Verfügung. Der benutzenden Komponente ist es möglich, sich vom Control über bestimmte Ereignisse benachrichtigen zu lassen, um so auf Aktionen des Benutzers reagieren zu können. Jeder Control verfügt dabei über eine individuelle Schnittstelle und über mehrere individuelle Ereignisse.

Abbildung 38 zeigt das Control-Rahmenwerk. Aus Gründen der Übersichtlichkeit habe ich in der Abbildung nicht sämtliche Controlklassen dargestellt. Wie man sehen kann, gibt es innerhalb des Rahmenwerks keine gemeinsame Oberklasse. Alle Controlklassen erben von der CWnd Klasse, welche die Grundfunktionalität eines Fensters des Windows Fenstersystems kapselt.

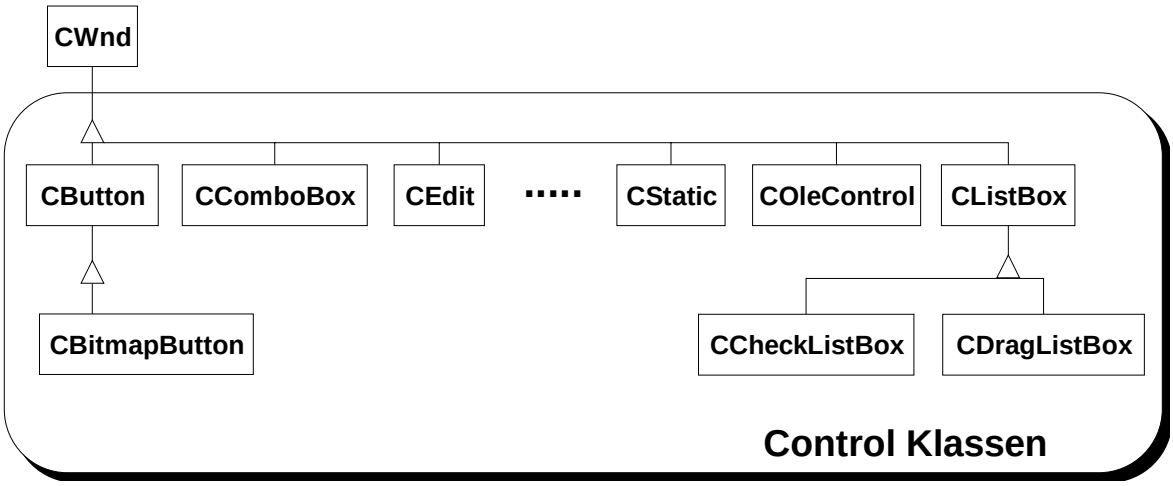


Abbildung 38: Das Control-Rahmenwerk im Detail

Im Windows Fenstersystem haben sich innerhalb der letzten Jahre vielen Veränderungen und Erweiterungen ergeben, insbesondere im Bereich der Ein-/Ausgabekomponenten. Mit jeder neuen Ein-/Ausgabekomponente steigt auch die Anzahl der Controlklassen im MFC Rahmenwerk. Zum heutigen Zeitpunkt gibt es bereits 23 Controlklassen im MFC Rahmenwerk[15]. Eine vollständige Liste aller Controlklassen mit zugehöriger Beschreibung ist im Anhang 12.3 aufgeführt.

Für jeden Entwurfszweck ist in der Regel ein Control vorhanden. Nur in Ausnahmefällen kann es vorkommen, daß ein Entwickler einen eigenen Control entwerfen muß. Es sei erwähnt, daß das MFC Rahmenwerk und die Entwicklungswerkzeuge die Konstruktion eigener Controls gut unterstützen.

7.4 Werkzeugfunktion und Document-Komponenten

In der Konstruktionsidee wurde erläutert, daß die Document-Komponente der Document/View Architektur die zentralen Aufgaben der Funktionskomponente übernehmen soll, welche die fachliche Funktionalität eines Werkzeugs festlegt. Die Schnittstelle einer FK besteht aus Methoden, die diese Funktionalität implementieren und die man in verändernde und sondierende Methoden unterteilt. Innerhalb der FK werden auch die Operationen zur Bearbeitung des Materials implementiert.

[15] MFC Version 4.0

Man kann die Document-Komponente für die Konstruktion einer Funktionskomponente nutzen, die die oben beschriebenen Aufgaben erfüllt. Der Entwickler muß dabei jedoch einige Besonderheiten der CDocument Klasse beachten. Ich werde daher im folgenden Text die CDocument Klasse und ihre Schnittstelle genauer beschreiben. Eine vollständige Tabelle aller Methoden der CDocument Klasse kann der Leser im Anhang 12.2 betrachten.

Innerhalb dokumentenbasierter Anwendungen realisiert die Document-Komponente üblicherweise die Persistenz der Dokumente. In der Schnittstelle der CDocument Klasse gibt es daher Methoden für das Öffnen, Schließen und Speichern von Dokumenten. Das MFC Rahmenwerk verwendet den Serialisierungsmechanismus (siehe Kapitel 4.4), um Dokumente in einer Datei zu speichern oder um Dokumente aus einer Datei auszulesen.

Es finden sich noch weitere Methoden in der Schnittstelle der CDocument Klasse, die die Konstruktion dokumentenbasierter Anwendungen unterstützen. Zusammenfassend möchte ich sagen, daß man diese Methoden bei der Konstruktion eines Werkzeugs nach WAM normalerweise nicht verwenden wird.

Bei der Werkzeugkonstruktion des Pausenplanersystems hat sich gezeigt, daß die CDocument Klasse eine gute Basis für die Realisierung der Funktionskomponenten liefert.

7.5 Werkzeuge als Komponenten

Im Rahmen der Arbeit habe ich über mehrere Möglichkeiten einer Umsetzung der Werkzeugkonstruktion nachgedacht. Innerhalb des MFC Rahmenwerks gibt es sicherlich mehrere Alternativen. Dabei ist die hier vorgestellte Umsetzung „Werkzeuge als Komponenten“ vom Konzept des Gesamtsystems abhängig. Das Pausenplanersystem wurde von uns als komponentenbasiertes System entworfen (siehe Kapitel 8). Der hier vorgestellte Weg ist demnach eine Lösung, die zu der grundlegenden Entwurfsidee am besten paßt.

Die WAM Entwurfsmuster besagen, daß die Funktionskomponente das Werkzeug gegenüber den anderen Komponenten des Systems vertritt. Die Schnittstelle der Funktionskomponente wird daher auch **Werkzeugschnittstelle** genannt. Im vorherigen Kapitel wurde beschrieben, wie die Document-Komponente der Document/View Architektur die Aufgaben der Funktionskomponente übernimmt. Daher ist zu klären, wie andere Komponenten über eine geeignete Schnittstelle der Document-Komponente auf das Werkzeugs zugreifen können.

Im MFC Rahmenwerk ist ein solcher Zugriffsmechanismus in Form des Automationsdienstes (siehe Kapitel 5.4) bereits integriert. Der Entwickler kann die Document-Komponente zu einem **entfernt zugreifbaren Objekt** machen, indem er innerhalb des Konstruktors der CDocument Klasse die Methode *EnableAutomation()* aufruft.

Durch den Aufruf wird das gesamte Werkzeug zu einem Automationsserver und die Document-Komponente zu einem Automationsobjekt. Die Document-Komponente kann nun über das „IDispatch“ Interface Automationsmethoden und Eigenschaften für andere Komponenten zugreifbar machen. Das Interface stellt somit den anderen Komponenten des Systems eine Werkzeugschnittstelle zur Verfügung. Abbildung 39 zeigt die Architektur des Werkzeugs im Überblick.

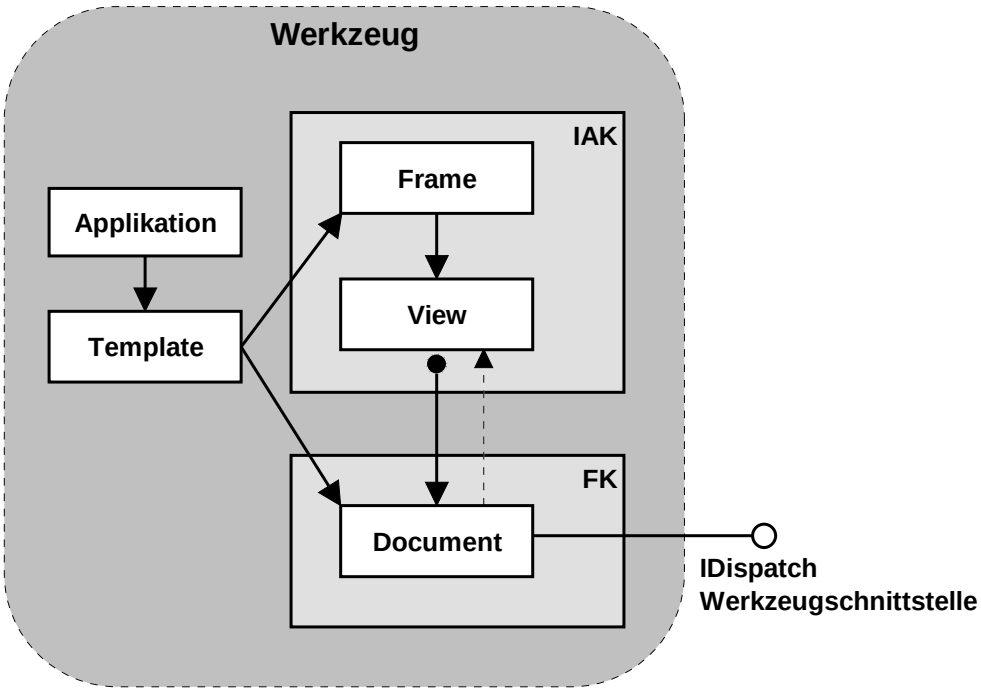


Abbildung 39: Ein Werkzeug als Automationsserver

Das Werkzeug wird durch die Verwendung des Automationsdienstes zu einer abgeschlossenen Softwarekomponente, die der Anwendungsentwickler nach dem Black-Box Prinzip verwendet. Die Aktivitäten des Werkzeugs finden innerhalb eines eigenen Prozesses statt. Die Application-Komponente der verwendeten Document/View Architektur hat dabei die Rolle des Prozeßverwalter übernommen. Sie kapselt die zentrale Nachrichtenverarbeitung des Windows Fenstersystems („Message Pump“) und repräsentiert den Prozeß der abgeschlossenen Softwarekomponente. Die Application-Komponente erzeugt außerdem die Template-Komponente des Werkzeugs, welche die Document, View und Frame Werkzeugkomponenten initialisiert.

7.6 Werkzeugkomposition

Das Werkzeugkomposition-Entwurfsmuster ermöglicht es dem Entwickler, seine Werkzeuge hierarchisch zu strukturieren. Dabei wird ein Werkzeug aus mehreren Subwerkzeugen aufgebaut, die es für die Erledigung seiner Aufgabe verwendet. Die existierenden Konzepte der Werkzeugkomposition werden in Abschnitt 2.3 beschrieben. Dieser Abschnitt zeigt, wie eine Umsetzung des Entwurfsmusters unter Nutzung des MFC Rahmenwerks erfolgen kann.

Das Werkzeugkomposition-Entwurfsmuster besagt, daß die Funktionskomponente das Werkzeug gegenüber dem Kontextwerkzeug vertritt. Im vorherigen Abschnitt wurde gezeigt, wie Werkzeuge durch die Verwendung des Automationsdienstes zu einer abgeschlossenen Softwarekomponente werden. In dem nun folgenden Text werde ich beschreiben, wie diese Art der Werkzeugkonstruktion innerhalb der Realisierung des Werkzeugkomposition-Entwurfsmusters Verwendung findet.

Der hier beschriebene Mechanismus ist nicht auf die Kommunikation zwischen Kontext- und Subwerkzeug beschränkt, er kann auch zwischen anderen Komponenten des Systems verwendet werden.

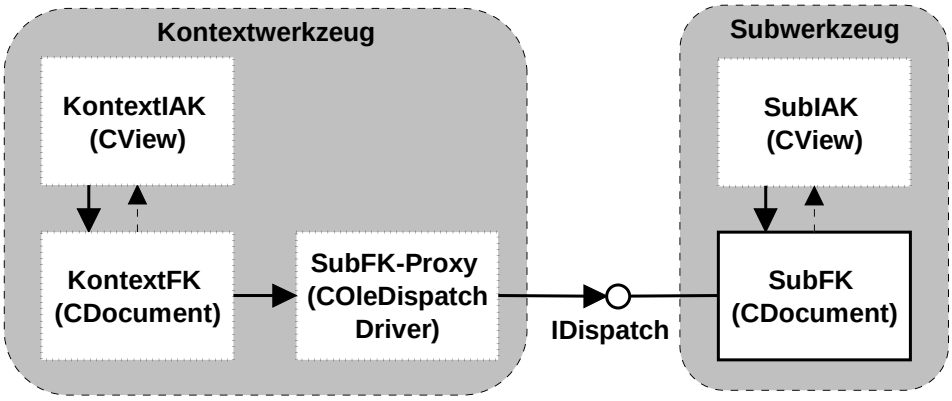


Abbildung 40: Automation für Werkzeuge

Man erkennt in Abbildung 40, wie das Kontextwerkzeug das Subwerkzeug unter Verwendung des Automationsdienstes benutzt. Innerhalb der Abbildung beschränke ich mich zunächst auf die Benutzungskommunikation zwischen den Werkzeugen. Auf die Kommunikation des Ereignismechanismus gehe ich erst in nächsten Kapitel genauer ein. Die abgerundeten Rechtecke um die Einzelkomponenten des Kontext- und Subwerkzeugs stehen für die Prozeßgrenzen, in denen sich die Werkzeuge befinden. Die Namen der aus dem MFC Rahmenwerk beteiligten Klassen stehen unter der Komponentenbezeichnung in Klammern. Die dargestellte Werkzeugkomponente erbt dabei von der beteiligten MFC Klasse.

Die KontextFK benutzt ein Proxy-Objekt, um auf die entfernte SubFK zugreifen zu können. Dieses Proxy-Objekt entspricht dem Stellvertreterobjekt des Proxy-Strukturierungsmusters im „Entwurfsmuster“ Buch von Gamma et al [GHJ+96]. Es repräsentiert das Subwerkzeug im Prozeßraum des Kontextwerkzeugs. Dabei stellt es der KontextFK alle Methoden an seiner Schnittstelle zur Verfügung, die auch an der SubFK als Automationsmethoden nach außen sichtbar sind.

Die eigentliche Kommunikation erfolgt über das **IDispatch** Interface und wird in den Klassen COleDispatchDriver und CCmdTarget, ein Vorfahre von CDocument, vor dem Entwickler verborgen. Die Kommunikation erfolgt synchron. Das bedeutet, daß das Kontextwerkzeug nach einem Methodenaufruf wartet, bis die Bearbeitung der Methode im Subwerkzeug abgeschlossen ist.

Mit dem ClassWizard Entwicklungswerkzeug aus der Visual C++ Umgebung kann man die Automationsmethoden einer CCmdTarget Klasse oder von ihr abgeleiteten Klassen erstellen und verwalten. Darüber hinaus ermöglicht das Werkzeug die automatische Generierung von Proxy-Klassen.

7.7 Der Ereignismechanismus

Ein Ereignismechanismus ist ein spezieller Reaktionsmechanismus (siehe [RW96]), der die lose Kopplung zwischen abhängigen Komponenten eines Systems realisiert, indem er die Versendung und den Empfang von Ereignissen durchführt. In diesem Kapitel beschreibe ich die Umsetzung eines Ereignismechanismus unter Nutzung des MFC Rahmenwerks. Die Beschreibung findet im Kontext der Werkzeugkomposition statt. Die Umsetzung ist jedoch nicht auf die Kommunikation zwischen Kontext- und Subwerkzeug eingeschränkt, sie kann auch für die Kommunikation zwischen beliebigen Komponenten des Systems verwendet werden.

Bei der Umsetzung eines Ereignismechanismus mit dem MFC Rahmenwerk kann der Entwickler auf einen vorhandenen COM Basisdienst zurückgreifen. Die Grundlagen zu COM werden in Kapitel 5 beschrieben. Der erwähnte Basisdienst nennt sich „Connectable Objects“ und ist in des MFC Rahmenwerk integriert. Im folgenden werde ich die beteiligten Komponenten, COM Interfaces und die Funktionalität der „Connectable Objects“ genauer erläutern.

Betrachten wir dazu Abbildung 41, in der die beteiligten Komponenten sowie die verwendeten Interfaces dargestellt sind. Das Subwerkzeug bietet den Dienst eines „Connectable Objects“ an. Das Kontextwerkzeug nimmt diesen Dienst in Anspruch, da es beim Auftreten eines bestimmten Ereignisses im Subwerkzeug benachrichtigt werden möchte. Sowohl Kontext- als auch Subwerkzeug lagern den Ereignismechanismus in

Klassen aus. Auf Seiten des Kontextwerkzeugs ist dies die EventStub Klasse, die das Ziel eines Ereignisses darstellt. Die Funktionskomponente des Kontextwerkzeugs sollte das EventStub Objekt beobachten, so daß eine gegenseitige Benutztbeziehung vermieden werden kann. Dadurch wird es möglich, daß ein EventStub auch von anderen Komponenten verwendet werden kann.

Das Subwerkzeug verwendet zu diesem Zweck die Event Klasse, die die Quelle eines Ereignisses repräsentiert. Ähnlich dem Ereignis-Muster aus [RW96] stellt diese Struktur die Ereignisse in den Vordergrund. Dadurch wird es möglich, die Ereignisse des Systems durch Ereignisklassen zu differenzieren, wobei sich stets Paare aus zusammengehörigen Event und EventStub Klassen bilden. Ein anderer Reaktionsmechanismus, das Beobachtermuster aus Kapitel 7.2, stellt die Beobachtung in den Vordergrund. Einen ausführlichen Vergleich beider Mechanismen findet man in [RW96].

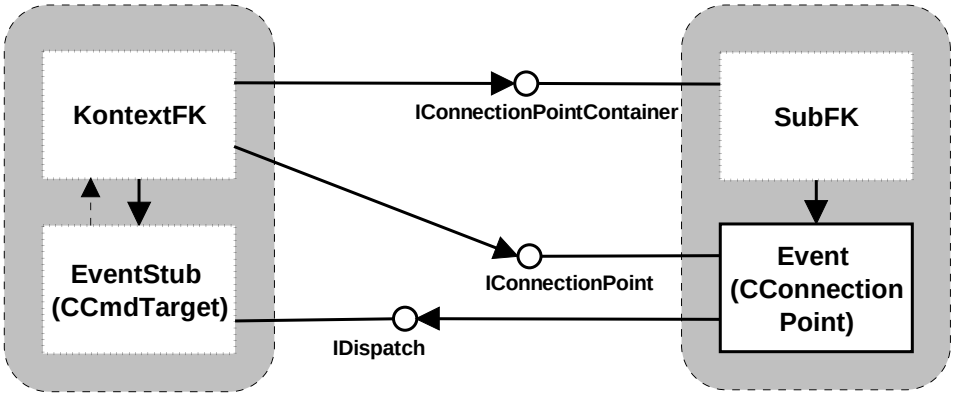


Abbildung 41: Der Ereignismechanismus „Connectable Objects“

Das An- und Abmelden für eine Benachrichtigung über Ereignisse findet an den sogenannten „Connection Points“ statt. Jeder Connection Point repräsentiert dabei ein bestimmtes Ereignis und jedes Connectable Object kann mehrere Connection Points anbieten.

Will sich das Kontextwerkzeug beim Subwerkzeug anmelden, so muß es beim Subwerkzeug erst einmal Zugriff auf den richtigen Connection Point erhalten. Zu diesem Zweck bietet das Subwerkzeug das **IConnectionPointContainer** Interface an. Die Funktionen des IConnectionPointContainer Interface ermöglichen es dem Kontextwerkzeug herauszubekommen, welche Connection Points vom Subwerkzeug angeboten werden und geben Zeigerreferenzen der sondierten Connection Points heraus.

Zugriff auf den Connection Point erhält das Kontextwerkzeug über die Funktionen des **IConnectionPoint** Interfaces. Die *Advise(...)* und *Unadvise(...)* Funktionen ermöglichen das An- und Abmelden des Kontextwerkzeugs beim Subwerkzeug. Das IConnectionPoint Interface wird in der CConnectionPoint Klasse des MFC Rahmenwerks implementiert.

Bei der Anmeldung übergibt das Kontextwerkzeug die Referenz seines EventStub Objekts dem Connection Point. Diese Referenz ist ein **IDispatch** Interfacezeiger. Die EventStub Klasse implementiert das IDispatch Interface indem sie von der CCmdTarget Klasse erbt. Nach dem Anmelden ist die Verbindung zwischen Event (Ereignisquelle) und EventStub (Ereignisziel) hergestellt.

Wird nun das Ereignis durch eine Zustandsänderung in der Funktionskomponente des Subwerkzeugs ausgelöst, so werden alle beim Connection Point angemeldeten Beobachter benachrichtigt. Die Benachrichtigung erfolgt innerhalb des Event Objekts, das über das IDispatch Interface eine Methode im EventStub Objekt des Kontextwerkzeugs aufruft. Das EventStub Objekt kann daraufhin die Funktionskomponente des Kontextwerkzeugs benachrichtigen, die auf das Ereignis reagiert.

Leider unterstützt die Visual C++ Entwicklungsumgebung den Entwickler bei der Umsetzung des Ereignismechanismus nicht. Im ClassWizard Entwicklungswerkzeug gibt es zwar einen „ActiveX Events“ Karteireiter, jedoch kann man in diesem Teil des ClassWizard nur Events von ActiveX Controls verwalten.

7.8 Technische Konsequenzen dieser Umsetzung

Dieses Kapitel beschreibt und diskutiert die technischen Konsequenzen, die sich aus der vorgestellten Werkzeugkonstruktion mit dem MFC Rahmenwerk ergeben. Die technischen Konsequenzen wirken sich auf den Entwurf der Werkzeuge sowie auf den Entwurf des Gesamtsystems aus. Man kann vier grundlegende Konsequenzen bei der Werkzeugkonstruktion erkennen. Diese sind:

- Jedes Werkzeug/Subwerkzeug hat einen eigenen Prozeß und ein eigenes Fenster:

Der erste Punkt ist sicherlich der auffallendste. Es ist sicherlich ungewöhnlich, daß man bei der Konstruktion eines Systems für jedes Werkzeug und Subwerkzeug einen eigenen Prozeß bereitstellt.

Innerhalb der vorgestellten Werkzeugkonstruktion ist jedes Werkzeug eine abgeschlossene Softwarekomponente. Der Entwickler verwendet das Werkzeug nach dem Black-Box Prinzip, wobei die Aktivitäten jeder Black-Box innerhalb eines eigenen Prozesses stattfinden. Werkzeuge als Softwarekomponenten bieten einige Vorteile. Verwendung und Wiederverwendung von Werkzeugen findet auf binärer Ebene und nicht auf Programmiersprachen Ebene statt. Der Entwickler kann, wenn er möchte, unterschiedliche Sprachen bei der Konstruktion einsetzen. Durch die Verteilung der Anwendung auf mehrere Prozesse kann die Robustheit des Gesamtsystems erhöht werden.

Nachteilig bei der vorgestellten Werkzeugkonstruktion ist die Tatsache, daß jedes Werkzeug und Subwerkzeug in einem eigenen Fenster dargestellt wird. Die Umsetzung von eingebetteten Subwerkzeugen, die sich ein Fenster teilen, ist mit der vorgestellten Konstruktion daher nicht möglich.

- IAK ist in View- und Frame-Komponente aufgespalten:

Die Interaktionskomponente ist bei der vorgestellten Werkzeugkonstruktion in View- und Frame-Komponente aufgespalten, da Menü, Symbol- und Statusleisten bei der Verwendung der Document/View Architektur von der Frame-Komponente verwaltet werden, ein Umstand, den man deutlich als Nachteil ansehen muß. Diesen Nachteil können wir jedoch relativieren, da man durch die Verwendung des Routingmechanismus sämtliche Aufgaben, die Handhabung und Präsentation des Werkzeugs betreffen, innerhalb der View-Komponente bearbeiten kann.

- Mehrere verschiedene IAKs zu einer FK sind problematisch:

Die Verwendung mehrere verschiedener IAKs, bestehend aus View- und Frame-Komponente zu einer FK, gestaltet sich bei der vorgestellten Werkzeugkonstruktion problematisch, ein Umstand, der aus der Nutzung der Document/View Architektur hervorgeht. Innerhalb der Architektur kann man einer Template-Komponente nur jeweils einen bestimmten View-, Frame- und Document-Typ zuweisen. Die Verwendung unterschiedlicher View-Komponenten ist zwar möglich, wird aber von der Architektur und den Entwicklungswerkzeugen nur ungenügend unterstützt.

- Verwendung unterschiedlicher Reaktionsmuster:

Die hier vorgestellte Konstruktionslösung verwendet für die Realisierung der Beobachtungsbeziehung unterschiedliche Reaktionsmuster. Zwischen IAK und FK eines Werkzeugs wird das Beobachtermuster der Document/View Architektur verwendet. Zwischen Kontext- und Subwerkzeug wird der COM Ereignismechanismus verwendet. Die Verwendung von zwei Reaktionsmustern kann auf die unterschiedlichen Anforderungen bei der Realisierung der Beobachtungsbeziehung zurückgeführt werden.

7.9 Zusammenfassung und Ausblick

In diesem Kapitel habe ich die Konstruktion von Werkzeugen unter Nutzung des MFC Rahmenwerks beschrieben. Dabei kam der Document/View Architektur des Rahmenwerks eine besondere Bedeutung zu. Die vorgestellte Konstruktionsidee ermöglicht die geeignete Zuordnung der Document/View Komponenten zu den Komponenten des IAK-FK Entwurfsmusters.

In dem hier vorliegenden Kapitel wurde das Beobachtermuster der Document/View Architektur sowie die Verwendung vorgefertigter Document- und View-Komponenten des MFC Rahmenwerks von mir beschrieben.

Des weiteren konnte ich zeigen, wie man ein Werkzeug durch die Verwendung des Automationsdienstes zu einer abgeschlossenen Softwarekomponente machen kann. Die Idee, die Werkzeuge zu Automationsservern zu machen, war für die komponentenbasierte Konstruktion des Gesamtsystems von großer Bedeutung.

Als nächsten Schritt habe ich die Realisierung der Werkzeugkomposition beschrieben. Am Beispiel dieser Realisierung wurden die Benutzungskommunikation und der Ereignismechanismus erläutert. Sowohl Benutzungskommunikation als auch Ereignismechanismus werden, so wie sie hier beschrieben sind, für die Kommunikation zwischen allen Komponenten des Systems verwendet.

Bei der Konstruktion der Werkzeuge des Pausenplanersystems hat sich gezeigt, daß die hier vorgestellte Art der Werkzeugkonstruktion gut anwendbar ist. Zusätzlich konnte durch den Einsatz der Visual C++ Entwicklungswerkzeuge der Konstruktionsprozeß erheblich vereinfacht werden. Die Diskussion der technischen Konsequenzen, die sich aus der Werkzeugkonstruktion ergeben, trübt dabei den durchweg positiven Eindruck ein wenig.

Damit nicht ausschließlich die Werkzeugkonstruktion Thema dieser Arbeit bleibt, wird im nächsten Kapitel die Konstruktion der noch fehlenden Komponenten des Pausenplanersystems beschrieben. Ein Überblick über Gruppenarbeitsumgebung, Material sowie die Kopplung zwischen Werkzeug und Material ist Gegenstand des weiteren Verlaufs dieser Arbeit.

8 Konstruktion des Pausenplanersystems

Werkzeuge, Materialien und Gruppenarbeitsumgebung des Pausenplanersystems wurden von mir bereits in Kapitel 3 vorgestellt. Die Konstruktion der Werkzeuge wurde im vorherigen Kapitel ausführlich beschrieben. Dieses Kapitel soll dem Leser den Gesamtüberblick über die Konstruktion des Pausenplanersystems geben.

Dabei gehe ich zunächst auf die komponentenbasierte Konstruktion des Gesamtsystems ein und werde in den Abschnitten danach die Konstruktion der Gruppenarbeitsumgebung und der Materialien genauer betrachten. Außerdem möchte ich einen Blick auf die Koppelung von Werkzeug und Material werfen und die Kommunikation zwischen den Komponenten des Pausenplanersystems genauer erläutern.

8.1 Komponentenbasierte Konstruktion

Sämtliche Komponenten des Pausenplanersystems werden über ihre Automations-Schnittstelle angesprochen. Klienten, die eine Komponente benutzen möchten, besorgen sich eine Referenz auf die Automations-Schnittstelle der Komponente. Sie können über die Automations-Schnittstelle mit der Komponente kommunizieren. Die Klienten erhalten die benötigte Referenz entweder vom Betriebssystem oder von einer anderen Komponente des Systems.

Eine Besonderheit komponentenbasierter Konstruktion ist, daß der Klient den Aufenthaltsort der angesprochenen Komponente nicht wissen muß. Die Komponente kann sich im selben Prozeßraum, in einem anderen Prozeßraum auf dem selben Rechner oder auf einem anderen Rechner des Netzwerks befinden. Außerdem muß der Klient nicht wissen in welcher Programmiersprache die Komponente implementiert wurde. Die Verwendung einer Komponente findet auf binärer Ebene und nicht auf Ebene der Programmiersprache statt.

Der Klient benutzt „einfach“ die Komponente und das unterliegende Betriebssystem sorgt für die notwendige technische Transparenz. Für den Entwurf eines Verteilten Systems ist dieser angesprochene Punkt wichtig, da er eine gewisse Flexibilität gewährleistet.

Abbildung 42 zeigt einige Komponenten des Pausenplanersystems und ihre Automations-Schnittstellen.

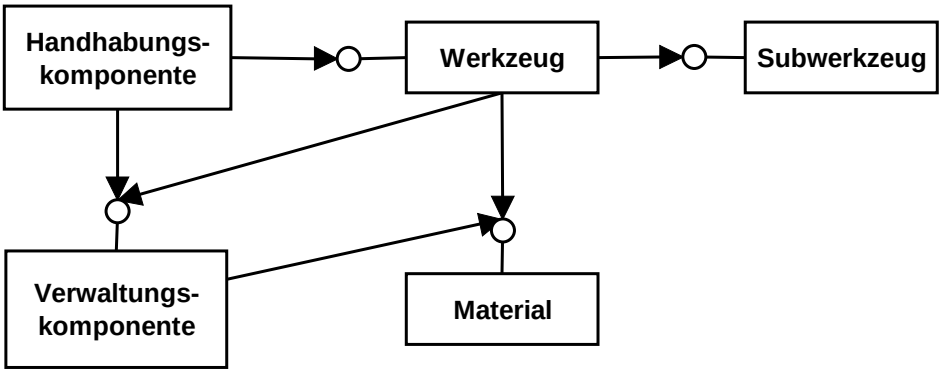


Abbildung 42: Komponenten des Pausenplanersystems

Sämtliche Komponenten des Pausenplanersystems werden unter Nutzung des MFC Rahmenwerks konstruiert. Dabei kann die Werkzeugkonstruktion, wie sie in Kapitel 7 von mir beschrieben wurde, als gutes Beispiel für die Konstruktion der anderen Komponenten des Pausenplanersystems angesehen werden. Die nächsten Abschnitte beschreiben die Konstruktion der anderen Komponenten.

8.2 Konstruktion der Gruppenarbeitsumgebung

Der folgende Abschnitt beschreibt, wie die Gruppenarbeitsumgebung des Pausenplanersystem konstruiert wurde. Die hier vorgestellte Konstruktion setzt den Entwurf der Gruppenarbeitsumgebung, bestehend aus Handhabungs- und Verwaltungskomponente, um (siehe Abschnitt 3.4). Eine detaillierte Beschreibung der Konstruktion der Gruppenarbeitsumgebung findet der Leser in der Arbeit von Andreas Hartmann [Har98].

Handhabungs- und Verwaltungskomponente der Gruppenarbeitsumgebung sind als Automationsserver konstruiert worden. Die externen Subkomponenten der Verwaltungskomponente sind Automationsobjekte, die den anderen Komponenten des Pausenplanersystems den entfernten Zugriff über ihre Automations-Schnittstelle anbieten. Abbildung 43 soll diese Konstruktion und den Zugriff auf die einzelnen Automations-Schnittstellen verdeutlichen.

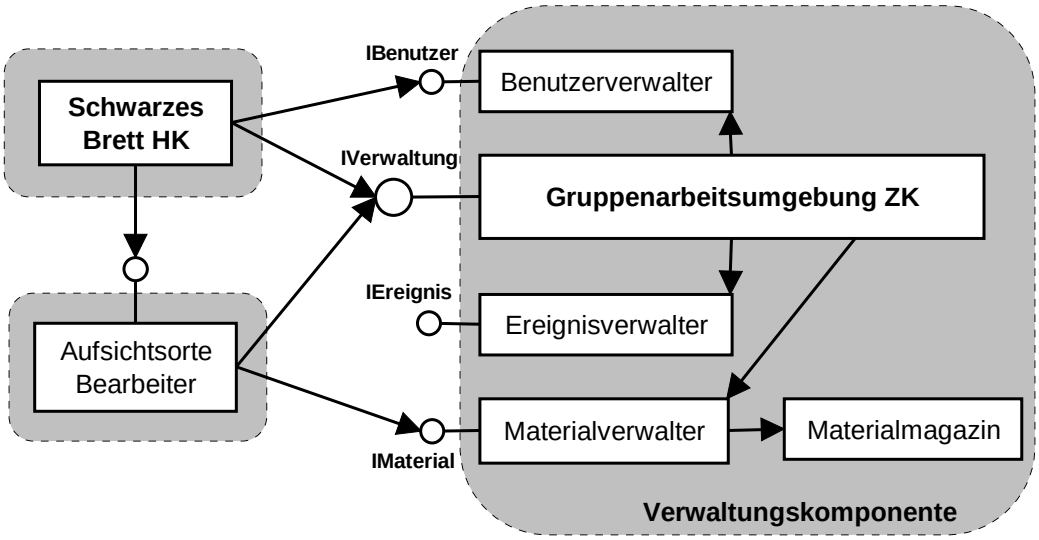


Abbildung 43: Zugriff auf die Automations-Schnittstellen der Verwaltungskomponente

Die Automations-Schnittstelle **IVerwaltung** der Gruppenarbeitsumgebung Zentralkomponente ist die zentrale Schnittstelle der Verwaltungskomponente. Bei der Erzeugung eines Werkzeugs übergibt die Handhabungskomponente dem Werkzeug die Referenz auf die IVerwaltung Schnittstelle. Über die IVerwaltung Schnittstelle kann sich die Handhabungskomponente oder ein Werkzeug nach Bedarf Referenzen auf die Automations-Schnittstellen der Subkomponenten geben lassen.

Die Initialisierung des gesamten Systems erfolgt durch die Anmeldung des ersten Gruppenmitglieds am Schwarzen Brett. Zunächst startet der Benutzer die Schwarzes Brett Handhabungskomponente auf seinem Arbeitsplatzrechner. Die Handhabungskomponente versucht nun auf die Verwaltungskomponente der Gruppenarbeitsumgebung zuzugreifen. Ist der Automationsserver der Verwaltungskomponente nicht aktiv, so wird dieser gestartet und die Handhabungskomponente erhält eine Referenz auf die IVerwaltung Automations-Schnittstelle der Verwaltungskomponente.

Der Automationsserver der Verwaltungskomponente bleibt aktiv, solange sich ein Gruppenmitglied am Schwarzen Brett befindet. Meldet sich das letzte Gruppenmitglied ab, so werden die Materialien gespeichert und der Serverprozeß wird beendet.

Die hier vorgestellte Konstruktion der Gruppenarbeitsumgebung verwendet sowohl für die Verwaltungskomponente als auch für die Handhabungskomponente die Infrastruktur der Document/View Architektur. Das Applikationsgerüst für Verwaltungs- und Handhabungskomponente wurde unter Verwendung des AppWizard Entwicklungswerkzeugs erzeugt (siehe Kapitel 6.1).

8.3 Konstruktion der Materialien

Dieser Abschnitt beschreibt die Konstruktion der Materialien des Pausenplanersystems. Die Übersicht über den Materialentwurf in Abschnitt 3.6 zeigt, daß die Materialien des Pausenplanersystems eine Hierarchie bilden. Die hier vorgestellte Konstruktion geht daher auch auf zusammengesetzte Materialien ein.

In Abschnitt 3.6 wurde bereits beschrieben, daß die Materialien des Pausenplanersystems im Materialmagazin der Gruppenarbeitsumgebung erzeugt werden und während der gesamten Bearbeitung im Materialmagazin verbleiben. Die Materialien sind von uns als Automationsobjekte konstruiert worden. Werkzeuge bearbeiten Materialien und deren Submaterial daher immer über eine Referenz der Automations-Schnittstelle des Materials. Implementiert werden die Materialklasse durch Vererbung von der CCmdTarget Klasse des MFC Rahmenwerks.

Ein Beispiel soll diese Materialkonstruktion verdeutlichen. Abbildung 44 zeigt im oberen Teil einen beliebiger Bearbeitungszustand des Pausenplanersystems, Materialobjekte vom Typ Lehrkörper, Lehrer und Ausschlußzeitraum, befinden sich innerhalb des Materialmagazin. Der Lehrkörper VHS1 enthält die Lehrer Müller, Meier und Schultze, der Lehrer Schultze hat zusätzlich einen Ausschlußzeitraum. Im unteren Teil der Abbildung ist die statische Benutztbeziehung zwischen den verwendeten Materialklassen noch einmal dargestellt.

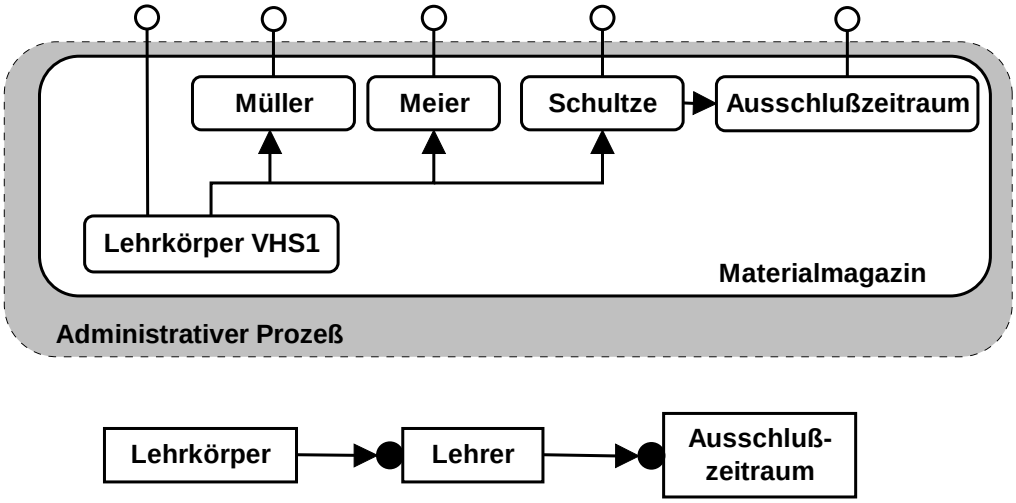


Abbildung 44: Komponentenbasierte Konstruktion der Materialien

Das Lehrkörper-Bearbeiter Werkzeug möchte den Lehrkörper VHS1 bearbeiten und holt sich die Referenz auf die Automations-Schnittstelle des Lehrkörpers über den Materialverwalter der Gruppenarbeitsumgebung. Das Werkzeug möchte im Laufe der Bearbeitung natürlich auch Zugriff auf die im Lehrkörper enthaltenen Lehrer bekommen. An der Schnittstelle des Lehrkörpers kann es dazu die Automationsmethode *GibLehrer(...)* aufrufen, die eine Referenz auf die Automations-Schnittstelle des gewünschten Lehrerobjekts zurückgibt. Über die Schnittstelle des Lehrers kann das Werkzeug Zugriff auf dessen Ausschlußzeiträume erhalten.

Zusammenfassend kann man folgende Punkte der Materialkonstruktion aufführen:

- Werkzeuge des Pausenplanersystems bearbeiten die Materialien immer über einen entfernten Zugriff.
- Die Materialien können dadurch im Materialmagazin verbleiben und sind dort für alle Werkzeuge zu jedem Zeitpunkt verfügbar.
- Ausschließlich Referenzen und Werte der Materialien werden über die Prozeßgrenzen hinweg übertragen.

8.4 Werkzeug und Material Kopplung

Bisher habe ich im Rahmen der Konstruktion ausschließlich über Werkzeuge und Materialien gesprochen. In diesem Abschnitt möchte ich genauer auf die Kopplung zwischen Werkzeug und Material eingehen.

Da Werkzeuge für die Bearbeitung verschiedener Materialien geeignet sind und Materialien von verschiedenen Werkzeugen bearbeitet werden können benötigt man eine Schnittstelle, welche das „Zueinanderpassen“ von Werkzeug und Material beschreibt. Diese Schnittstelle beschreibt die für das Funktionieren eines Werkzeugs relevanten Eigenschaften eines Materials.

In objektorientierten Programmiersprachen wird diese Schnittstelle durch eine Aspektklasse modelliert. Die Materialklasse erbt von der abstrakten Aspektklasse und implementiert deren Methoden. Die Werkzeugklasse benutzt die Aspektklasse, ohne daß sie die konkrete Materialklasse kennen muß.

Die oben beschriebene Lösung für das „Zueinanderpassen“ von Werkzeug und Material, findet auf Ebene von Programmiersprachen statt und geht davon aus, daß sich Werkzeug, Aspekt und Material in einem gemeinsamen Prozeßraum befinden (siehe Abbildung 45 linke Seite).

Das Pausenplanersystem ist ein Verteiltes System und wurde komponentenbasiert entworfen. Die Werkzeuge des Pausenplanersystems bearbeiten die Materialien über einen entfernten Zugriff. Material und Werkzeug befinden sich dabei in unterschiedlichen Prozeßräumen, die oben beschriebene Lösung konnten wir also nicht verwenden. Daher haben wir eine eigene Lösung gefunden, welche die Anforderungen des komponentenbasierten Entwurfs erfüllt (siehe Abbildung 45 rechte Seite).

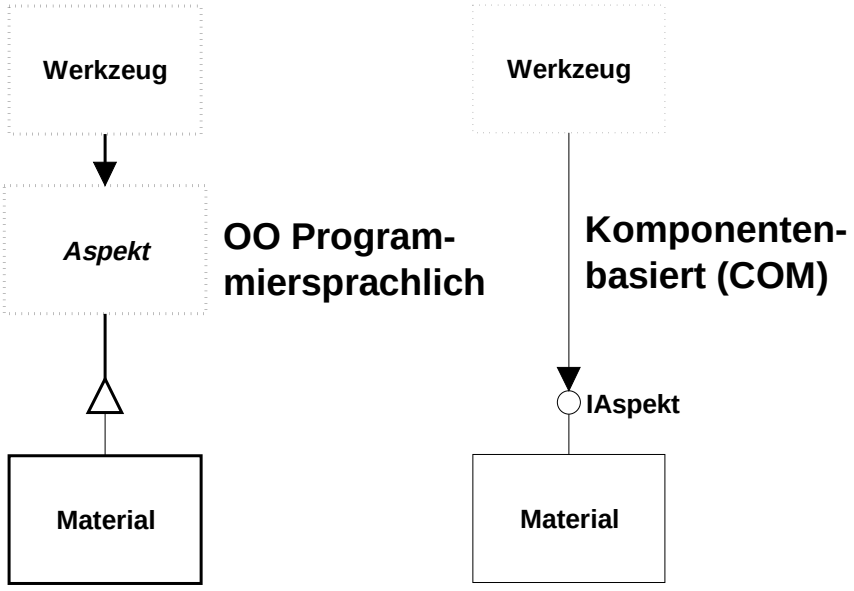


Abbildung 45: Zwei Möglichkeiten der Werkzeug und Material Kopplung

Bei der komponentenbasierten Lösung greift die Werkzeugkomponente über eine Automations-Schnittstelle auf die Materialkomponente zu. Für jeden benötigten Aspekt, der das „Zueinanderpassen“ von Werkzeug und Material beschreibt, kann eine Aspektschnittstelle definiert werden. Die Werkzeugkomponenten kennen die konkreten Materialkomponenten nicht, sondern nur die Aspektschnittstelle. Die Materialkomponenten müssen die in der Aspektschnittstelle spezifizierten Funktionen implementieren. Die Details der Implementierung mit dem MFC Rahmenwerks wurden bereits im Kontext von Werkzeug und Subwerkzeug beschrieben (siehe Abschnitt 7.6).

Im jetzigen Zustand des Pausenplanersystems wurde die hier vorgestellte Kopplung von Werkzeug und Material noch nicht umgesetzt. Die Werkzeugkomponente greift zwar über eine Automations-Schnittstelle auf die Materialkomponente zu, diese enthält jedoch die gesamte Schnittstelle der Materialkomponente. Ich denke jedoch, daß der Schritt von direkter Benutzung der Materialkomponente zu Aspektschnittstellen nicht sehr groß ist.

8.5 Kommunikation

Da sich die Komponenten des Pausenplanersystems in unterschiedlichen Prozessen aufhalten, muß die Kommunikation zwischen den Komponenten über Prozeßgrenzen hinweg stattfinden. Diese Art der Kommunikation nennt man **Inter Prozeß Kommunikation** (engl. Inter Process Communication, IPC). Dabei ergibt sich für die Inter Prozeß Kommunikation ein höherer Aufwand, als für die Kommunikation innerhalb eines Prozesses. Besonders interessant für den Anwender ist natürlich der höhere zeitliche Aufwand.

Im Fall des Pausenplanersystems konnte dieser höhere Aufwand von uns bei der Verwendung des Prototypen beobachtet werden, wurde jedoch nicht als störend empfunden. Das Arbeiten mit dem System unter Laborbedingungen war „flüssig“. Dennoch sollte man die zeitlichen Einbußen durch die Inter Prozeß Kommunikation nicht unterschätzen. Exakte Performancetests des Gesamtsystems wurden von uns nicht durchgeführt.

Die Kommunikation zwischen den Komponenten des Pausenplanersystems findet **synchron** statt. Das bedeutet, daß bei einem Methodenaufruf die aufrufende Komponente solange mit der weiteren Ausführung von Anweisungen wartet, bis die aufgerufene Komponente die Bearbeitung des Methodenaufrufs beendet hat. Die Reihenfolge der Bearbeitung von Anweisungen bleibt also streng sequentiell.

Bei **asynchroner** Kommunikation wartet die aufrufende Komponente nicht, sondern fährt sofort mit der Ausführung weiterer Anweisungen fort. Beide Komponenten arbeiten jetzt parallel. Ist die aufgerufene Komponente mit der Bearbeitung des Methodenaufrufs fertig, so benachrichtigt sie die aufrufende Komponente und sendet das Ergebnis der Bearbeitung zurück. Asynchrone Kommunikation zwischen zwei Komponenten bewirkt die Ausführung von nebenläufigen Prozessen. Dadurch verbessert sich die Performance, durch die Nebenläufigkeit wird das Gesamtsystem aber auch komplexer.

Die Kommunikation im Pausenplanersystem findet synchron statt, da die COM Technologie nur synchrone Kommunikation unterstützt. Die Konsistenz aller Komponenten im System ist demnach durch die sequentielle Bearbeitung aller Anweisungen gewährleistet. Das Pausenplanersystem ist durch die Einschränkung der Nebenläufigkeit übersichtlich, Abläufe im System sind für den Anwendungsentwickler verständlich. Durch die Einschränkung der Nebenläufigkeit bleiben jedoch die Möglichkeiten paralleler Verarbeitung ungenutzt. Es bleibt zu hoffen, daß zukünftige Versionen von COM auch asynchrone Kommunikation unterstützen.

9 Zusammenfassung und Ausblick

Dieses Kapitel faßt die wichtigsten Aspekte der hier vorliegenden Arbeit noch einmal zusammen und gibt einen Überblick über die erarbeiteten Lösungen. Der nachfolgende Ausblick versucht darüber hinaus mögliche Entwicklungen darzustellen.

Zu Beginn der Zusammenarbeit von Andreas Hartmann und mir lautete die zentrale Fragestellung: „Wie kann man einen WAM Softwareentwurf unter Verwendung des MFC Rahmenwerks konstruieren?“ Um diese Frage umfassend zu beantworten, wurde von uns die Pausenplaner Beispielanwendung ausgewählt und mit Hilfe des WAM Methodenrahmens entworfen.

In Kapitel 3 meiner Arbeit stelle ich die Pausenplaner Beispielanwendung vor. Dabei werden die Tätigkeiten der Pausenplanung beschrieben und Formen der Zusammenarbeit analysiert. Im Anhang dieser Arbeit kann das Ergebnis des Pausenplaner Entwurfs in Form von Handhabungs-, Werkzeug- und Materialvisionen genauer begutachtet werden. Schnell traten beim Pausenplaner Entwurf zusätzliche Fragen über Verteilungsaspekte des Systems und das zu verwendende Kooperationsmodell auf. Wir mußten diese Fragen in die zentrale Fragestellung mit einbeziehen, wobei aus einer Frage zwei unterschiedliche zentrale Fragen wurden:

- Gibt es für ein verteiltes Pausenplanersystem ein geeignetes Kooperationsmodell, welches sich in den WAM Methodenrahmen einfügt?
- Wie kann man den WAM Entwurf eines verteilten Systems unter Nutzung des MFC Rahmenwerks konstruieren?

Der Entwurf des verteilten Pausenplanersystems ist kein typischer Entwurf nach WAM (siehe [Gry96]). Hier wurden von uns neue Wege gegangen. Die im gemeinsamen Arbeitsprozeß entstandene Entwurfsmetapher Gruppenarbeitsumgebung konnte die erste zentrale Frage hinreichend beantworten.

Die Entwurfsmetapher Gruppenarbeitsumgebung wird in Kapitel 3 meiner Arbeit vorgestellt, sie erlaubt die Zusammenarbeit mehrere Personen an einem Ort. Im gleichen Kapitel wird die Verwendung der Entwurfsmetapher im Pausenplanersystems und ihre technische Umsetzung beschrieben. Grundsätzlich hat sich bei der Arbeit gezeigt, daß die Abbildung der Zusammenarbeit mehrerer Personen auf ein technisches System, Probleme bereiteten kann. Ausschlaggebend sind bei der Zusammenarbeit die Kommunikationsmöglichkeiten der beteiligten Personen im technischen System. Mit dem Pausenplanersystem konnten wir eine Beispielanwendung für eine konkrete Form der Zusammenarbeit an einem virtuellen Ort bereitstellen. Wir denken, daß die Zusammenarbeit der beteiligten Personen in gelungener Weise im Pausenplanersystem abgebildet worden ist.

Um die zweite zentrale Frage zu beantworten, habe ich in den folgenden Kapiteln meiner Arbeit zunächst einmal versucht dem Leser die technischen Grundlagen, die für die Konstruktion des Pausenplanersystems erforderlich sind, näher zu bringen.

In Kapitel 4 habe ich zunächst das MFC Rahmenwerks vorgestellt, den Aufbau der Klassenhierarchie beschrieben und danach gezeigt welche Anwendungen „normalerweise“ mit dem Rahmenwerk konstruiert werden. Im weiteren Verlauf des Kapitels habe ich die Objektmechanismen der Wurzelklasse beschrieben und gezeigt, wie diese gewinnbringend in eigenen Entwürfen eingesetzt werden können. Die Document/View Architektur verwirklicht die Infrastruktur einer mit dem MFC Rahmenwerk konstruierten Anwendung und ist damit der Kern des Rahmenwerks. Ich habe daher in Kapitel 4 die einzelnen Komponenten der Document/View Architektur und deren Zusammenwirken zusammengefaßt.

In Kapitel 5 habe ich dem Leser die Elemente der COM Komponententechnologie vorgestellt, die bei der Konstruktion des Pausenplanersystems verwendet wurden. Weitere Abschnitte des Kapitels beschreiben den Automationsdienst und die Integration der Komponententechnologie in das MFC Rahmenwerk. Diese Integration hat die komponentenbasierte Konstruktion des Pausenplanersystems sicherlich beeinflußt.

In Kapitel 6 habe ich die wichtigsten Entwicklungswerkzeuge für das MFC Rahmenwerks beschrieben. Wie sich bei der Konstruktion des Pausenplanersystems herausgestellt hat, ermöglicht die Verwendung der Werkzeuge, dem Anwendungsentwickler einen sicheren und effizienten Umgang mit dem MFC Rahmenwerk.

Seine Arbeit wird durch die Entwicklungsumgebung vereinfacht, da diese dem Entwickler eine abstraktere Sichtweise auf das Rahmenwerk ermöglicht.

Wie sich im Arbeitsprozeß gezeigt hat, ist der Zusammenhang zwischen MFC Rahmenwerk, COM Komponententechnologie und den Entwicklungswerkzeugen stark ausgeprägt. In den Kapiteln und Abschnitten meiner Arbeit habe ich immer wieder durch Querverweise auf diese Tatsache hingewiesen. MFC, COM und Entwicklungswerkzeuge verhalten sich wie drei Zahnräder, die sich gegenseitig antreiben[16]. Abbildung 46 soll das verzahnte Zusammenspiel der drei Zahnräder verdeutlichen.

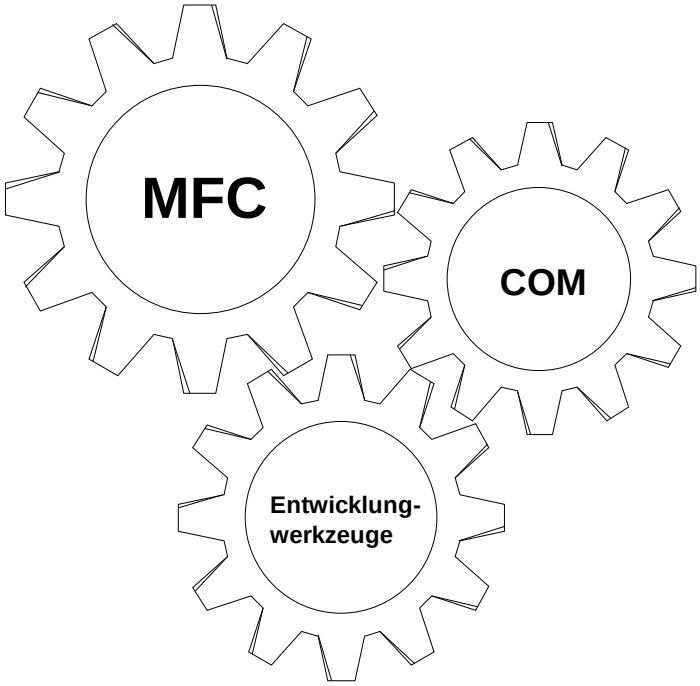


Abbildung 46: Verzahntes Zusammenspiel von MFC, COM und Entwicklungswerkzeugen

Aufbauend auf den technischen Grundlagen, der Kapitel 4,5 und 6 beschreiben die Kapitel im weiteren Verlauf meiner Arbeit, die Konstruktion von Werkzeugen, Materialien und Gruppenarbeitsumgebung.

Kapitel 7 beschreibt die komponentenbasierte Werkzeugkonstruktion mit dem MFC Rahmenwerk. Das Kapitel bildet damit den Kern meiner Arbeit. Dabei konnte ich aufzeigen, daß der Document/View Architektur des Rahmenwerks bei der komponentenbasierten Werkzeugkonstruktion eine besondere Bedeutung zukommt. Die im Kapitel vorgestellte Konstruktionsidee ermöglicht die geeignete Zuordnung der Komponenten der Document/View Architektur zu den Komponenten der WAM Entwurfsmusters. Die dargestellte Werkzeugkonstruktion zeigt wie man Funktion und Interaktion in Werkzeugen trennen kann, beschreibt die Umsetzung von geeigneten Reaktionsmustern und demonstriert die Komposition von Werkzeugen. Dabei wurde das Beobachtermuster der Document/View Architektur sowie die Verwendung vorgefertigter Document- und View-Komponenten von mir beschrieben. Zusätzlich zeige ich, wie Werkzeuge durch die Verwendung des Automationsdienstes zu abgeschlossenen Softwarekomponenten werden.

Kapitel 8 beschreibt die komponentenbasierte Konstruktion des gesamten Pausenplanersystems und geht dabei auf die Gruppenarbeitsumgebung, die Materialien und die Kopplung von Werkzeug und Material genauer ein. Dabei hat sich gezeigt, daß die Anforderungen eines verteilten Pausenplanersystems durch die hier vorgestellte komponentenbasierte Konstruktion sehr gut erfüllt werden konnten.

[16] Gelegentliches Verhaken der drei Zahnräder sei nicht ausgeschlossen ☺.

Die Konstruktionslösungen beziehen sich dabei nicht ausdrücklich auf das Pausenplanersystem, sondern könnten auch bei anderen WAM Softwareentwürfen eingesetzt werden. Im Rahmen dieser Arbeit konnten wir alternative Kooperationsmodelle jedoch nicht durch die Konstruktion eines weiteren Prototypen überprüfen.

Die vorgestellten Konstruktionslösungen sind praktikabel und ermöglichen eine effiziente Softwarekonstruktion nach WAM, ohne daß man auf die komplette Unterstützung des MFC Rahmenwerks oder die Hilfsmittel der Entwicklungsumgebung verzichten muß. Die Lösungen beschreiben die Verwendung des Rahmenwerks im Sinne einer Softwarekonstruktion nach WAM. Eine Erweiterung der MFC Klassenhierarchie war bei der Realisierung nicht notwendig, wodurch sich die Verwendung des Rahmenwerks für den Entwickler vereinfacht.

Das Pausenplanersystem ist zweifelsohne ein guter Beleg für die Brauchbarkeit der hier vorgestellten Konstruktionslösungen. Die bei der Konstruktion gesammelten Erfahrungen wurden in meiner Arbeit beschrieben und können sicherlich für den Entwurf und die Konstruktion zukünftiger Systeme genutzt werden.

Anwendungsentwickler die die hier vorgestellten Konstruktionslösungen verwenden möchten, müssen mit folgenden positiven als auch negativen Konsequenzen rechnen:

- Durch die Verwendung des MFC Rahmenwerks bei der Softwarekonstruktion erreicht der Entwickler die Unabhängigkeit von einigen Betriebssystem. Zur Zeit unterstützt das Rahmenwerk die Betriebssysteme Windows 95, Windows NT, Windows 3.11 und MacOS. Das Pausenplanersystem wurde von uns auf Windows 95 und Windows NT Systemen installiert und getestet. Über den Einsatz auf Windows 3.11 und MacOS Systemen kann ich an dieser Stelle keine genaue Aussage machen. Die konstruierten Komponenten des Pausenplanersystems, wie Werkzeuge, Materialien und Gruppenarbeitsumgebung, sind vom MFC Rahmenwerk abhängig, da sie Teile des Rahmenwerks verwenden. Eine Wiederverwendung einzelner Komponenten ist demnach nur im Kontext des Rahmenwerks möglich.

- Die Verwendung der MFC Entwicklungswerkzeuge erleichtert die Konstruktion von Softwaresystemen erheblich. In sämtlichen wichtigen Konstruktionsbereichen leisten die in der Arbeit vorgestellten Werkzeuge der Visual C++ Entwicklungsumgebung gute Dienste.

- Die Integration der COM Komponententechnologie in das MFC Rahmenwerk ist ein großer Vorteil. Besonders die komponentenbasierte Konstruktion eines (verteilten) Systems wird durch die Integration der Komponententechnologie erheblich erleichtert.

- Die Verwendung des MFC Rahmenwerks kann die Entwicklung eines eigenen Rahmenwerks überflüssig machen. Das Rahmenwerk kann an eigene Entwürfe und Vorstellungen angepaßt werden. Selbst die Entwicklungswerkzeuge des Rahmenwerks unterstützen eine solche Erweiterung (Meta AppWizard). Die Erfahrung hat gezeigt, daß der Entwurf eigener Rahmenwerke große Mühen und Kosten verursacht. Anwendungsentwickler sollten nach meiner Meinung möglichst wenig selber entwerfen, sondern bestehende Entwurfsmuster und Klassen in ihrem Entwurf wiederverwenden.

Große Stärken des MFC Rahmenwerks sehe ich in den Entwicklungswerkzeugen des Rahmenwerks, der weiten Verbreitung und in der schnellen Integration neuer Technologien in das Rahmenwerk. Dabei möchte ich besonders die COM Komponententechnologie hervorheben, die die Konstruktion des Pausenplanersystems enorm vereinfacht hat. Das MFC Rahmenwerk ist besonders für die Softwarekonstruktion auf Windows Betriebssystemen geeignet und hat sich dort im Laufe der Jahre etabliert.

In Zukunft wird die Entwicklung des MFC Rahmenwerks weiter voranschreiten. Die Document/View Architektur wird wie ich denke von den Veränderungen weitgehend unbetroffen bleiben. Entwicklungen sehe ich eher im Bereich der Verteilung (DCOM Technologie, COM+) und im Bereich Internet/Intranet. Die Portierung des MFC Rahmenwerks auf weitere Betriebssysteme ist, meiner Einschätzung nach eher unwahrscheinlich.

Auch bei dem Entwurf und der Konstruktion verteilter Anwendungen nach WAM wird sich in Zukunft vieles tun. Die Unterstützung von synchroner und asynchroner kooperativer Arbeit gewinnt immer mehr an Bedeutung. Die Entwurfsmetapher Gruppenarbeitsumgebung und die Konstruktion des Pausenplanersystems sind ein Schritt in diese Richtung. Hier werden noch einige Erfahrungen im Umgang, Entwurf und Konstruktion dieser Kooperationsmodelle von Nöten sein.

T.O.K.

10 Bibliographie

[BGK+97] D. Bäumer, G. Gryczan, R. Knoll, C. Lilienthal, D. Riehle, H. Züllighoven
Framework Development for Large Systems
CACM Oct 1997 Vol.40 Nr.10

[BGW97] D. Bäumer, G. Gryczan, M. Wulf
Ein Pausenplaner von SWT
Entwurfsdokumente vom Arbeitsbereich Softwaretechnik
Fachbereich Informatik der Universität Hamburg 1997

[Bla97] Mike Blaszczak
Professional MFC with Visual C++ 5
Wrox Pr 1997

[BR96] Dirk Bäumer, Dirk Riehle
Late Creation - A Creational Pattern
in Pattern Languages of Program Design
Addison-Wesley 1996

[Bra96] Marshall Brain
Developing Professional Applications for Windows95 and NT using MFC
Prentice Hall 1996

[Bro95] Kraig Brockschmidt
Inside OLE, Second Edition
Microsoft Press 1995

[BRS+98] D. Bäumer, D. Riehle, W. Siberski, C. Lilienthal, D. Megert, K. Sylla, H. Züllighoven
Values in Large Object Systems
Fachbereich Informatik der Universität Hamburg 1997 (in Arbeit)

[COM97] COM Programmers Reference
Microsoft Press 1997

[Cor95] The Common Object Request Broker: Architecture and Specification Revision 2.0
Im Internet auf „http://www.omg.org“
July 1995

[Cro97] Frank Crockett
MFC Developer’s Workshop
Microsoft Press 1997

[Dra98] Laura Draxler
Windows Programming under the hood of MFC
Prentice Hall 1998

[Eck96] Bruce Eckel
In C++ denken
Prentice Hall 1996

[Feu97] Alan R. Feuer
MFC Programming
Addison Wesley 1997

[FL97] Andreas Felten, Christian Langmann
Entwicklung einer CORBA-basierten verteilten Anwendung mit Distributed Smalltalk am Beispiel eines Pausenplaners
Studienarbeit am Fachbereich Informatik der Universität Hamburg 1997

[For96a] Doug Forguson
Programming with the MFC Document Template Architecture
C++ Report, March 1996

[For96b] Doug Forguson
Extending the MFC Document Template Architecture
C++ Report, April 1996

[GHJ+96] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides
Entwurfsmuster: Elemente wiederverwendbarer objektorientierter Software
Addison-Wesley 1996

[Gol83] Adele Goldberg
Smalltalk-80: The Interactive Programming Environment
Addison-Wesley, Menlo Park 1984

[Gri97] Richard Grimes
Professional DCOM Programming
Software Masters 1997

[Gry96] Guido Gryczan
Prozeßmuster zur Unterstützung kooperativer Tätigkeit
Deutscher Universitätsverlag 1996

[GUI94] The Windows Interface: An Application Design Guide
Microsoft Press 1994

[Har98] Andreas Hartmann
Softwarekonstruktion nach WAM zur Unterstützung von Kooperation an einem Ort am Beispiel eines Pausenplanersystems
Diplomarbeit am Fachbereich Informatik der Universität Hamburg 1998 (in Arbeit)

[Hor97] Ivor Horton
Beginning MFC Programming
Wrox Pr 1997

[Jac92] Ivar Jacobson
Object-Oriented Software Engineering: A Use Case Driven Approach
Addison-Wesley 1992

[JF88] Ralph E. Johnson, Brian Foote
Designing Reusable Classes
The Journal of Object-Oriented Programming, Vol. 1 Nr. 2, 1988

[KGZ94] Klaus Kilbert, Guido Gryczan, Heinz Züllighoven
Objektorientierte Anwendungsentwicklung: Konzepte, Strategien, Erfahrungen
Vieweg Verlag 1994

[KP88] Glenn E. Krasner, S. Pope
A Cookbook for using the Model-View-Controler User Interface Paradigm in Smalltalk-80
Journal of OOP, 1(3), Aug-Sep 1988

[Kru96] David J. Kruglinski
Inside Visual C++
Microsoft Press 1996

[Lew95] Ted Lewis
Object-Oriented Application Frameworks
Manning Publications 1995

[Mat97] Mirko Matytschak
Verteilte Anwendungen mit OLE-Remote-Automation
Objekt Spektrum 5/1997

[Mey90] Bertrand Meyer
Objektorientierte Softwareentwicklung
Hanser, Prentice Hall 1990

[MFC97] Microsoft Visual C++ MFC Library Reference, Part 1 und 2
Volume 1 und 2, Visual C++ Version 5.0 Documentation Library
Microsoft Press 1997

[MS96] David R. Musser, Atul Saini
STL Tutorial and Reference Guide: C++ Programming with the Standard Template Library
Addison-Wesley, Reading, Messachusetts 1996

[OHE96] R. Orfali, D. Harkey, J. Edwards
The Essential Distributed Objects Survival Guide
Wiley&Sons 1996

[Oni96a] Fritz Onion
Object Persistence in MFC
C++ Report Vol. 8 Iss. 1

[Oni96b] Fritz Onion
Dialog Data Exchange in MFC
C++ Report Vol. 8 Iss. 5

[Oni96c] Fritz Onion
Dynamic Data Structure Serialization in MFC
C++ Report Vol. 8 Iss. 3

[Pro96] Jeff Prosise
Programming Windows95 with MFC
Microsoft Press 1996

[RBP+93] J. Rumbaugh, M. Blaha, W. Premerlani, F. Eddy, W. Lorensen
Objektorientiertes Modellieren und Entwerfen
Carl Hanser, Prentice-Hall 1993

[Rie95] Dirk Riehle
Muster am Beispiel der Werkzeug und Material Metapher
Diplomarbeit am Fachbereich Informatik der Universität Hamburg 1995

[Rog97] Dale Rogerson
Inside COM
Microsoft Press 1997

[RW96] Stefan Roock, Henning Wolf
Konzeption und Implementierung eines Reaktionsmusters für objektorientierte Softwaresysteme, Studienarbeit am Fachbereich Informatik der Universität Hamburg 1996

[RW97] Stefan Roock, Henning Wolf
Die Raummetapher zur Entwicklung kooperationsunterstützender Softwaresysteme für Organisationen
Diplomarbeit am Fachbereich Informatik der Universität Hamburg 1997

[Sch96] Alexander Schill
Rechnergestützte Gruppenarbeit in verteilten Systemen
Prentice Hall

[Sch96] David Schmitt
Extending the MFC Library: Add Useful Reusable Features to the Microsoft Foundation class Library
Addison Wesley 1996

[Sch96b] Herbert Schildt
MFC Programming from the Ground Up
McGraw-Hill 1996

[Sma96] MFC Smartlabs: An Intelligient Tutoring System
Prentice Hall 1996

[Str92] Bjarne Stroustrup
Die C++ Programmiersprache
2. Aufl. Bonn, Addison-Wesley, 1992

[SW96] George Shepherd, Scot Wingo
MFC Internals
Addison-Wesley 1996

[Tan92] Andrew S. Tannenbaum
Computer-Netzwerke
2. Aufl. Attenkirchen, Wolfram's Fachverlag, 1992

[Tem97] Julian Templeman
Beginning MFC COM Programming
Wrox Pr 1997

[Tot96] Viktor Toth
Visual C++ 4 Unleashed
Sams Publishing 1996

[Tra96] Horst-Peter Traub
Objektorientierte Behälterklassen-Bibliothek – Konzepte, Entwurf und Implementation
Fachbereich Informatik Universität Hamburg 1996

[Wei97] Ulfert Weiß
Konzeptionelle und technische Weiterentwicklung eines objektorientierten Frameworks nach der Werkzeug-Material-Metapher, Diplomarbeit am Fachbereich Informatik der Universität Hamburg 1997

[Wil97] Al Williams
MFC 5 Black Book
Coriolis Group Books 1997

[WW94] Martina Wulf, Dirk Weske
Konzepte zur Materialversorgung verteilter Werkzeugumgebungen am Beispiel der Anbindung einer objektorientierten Datenbank, Studienarbeit am Fachbereich Informatik der Universität Hamburg 1994

[Zül98] Heinz Züllighoven
Das objektorientierte Konstruktionshandbuch nach dem Werkzeug & Material Ansatz
dpunkt Verlag 1998 (in Arbeit)

11 Anhang Entwurf des Pausenplanersystems

11.1 Überblicksvision des Pausenplanersystems

Die Pausen einer Schule müssen beaufsichtigt werden. Für die Beaufsichtigung sind die Lehrer des Lehrkörpers zuständig. Das Pausenplanersystem soll dem Pausenplanersteller helfen, eine zulässige und sinnvolle Pausenplanung durchführen zu können. Das Szenario „Pausenaufsichtsplan erstellen“ beschreibt, wie diese Aufgabe bisher gelöst wurde. Die in dem Szenario beschriebene Vorgehensweise der Erstellung von Lehrerkarten und die Belegung eines Pausenplanes mit diesen Lehrerkarten, soll durch das Pausenplanersystem voll unterstützt werden. Die Arbeitsmittel Lehrerkarten und Pausenplan werden im Rechner durch Materialien modelliert. Das Pausenplanersystem soll die Teilaufgaben, die notwendig sind, um dieser Aufgabe nachzukommen, vereinfachen.

- Das Durchführen einer Pausenbelegung wird, wie bisher, durch Ablegen einer Lehrerkarte auf einem Übersichtsplan realisiert. Einziger Unterschied besteht darin, daß Plan und Karten auf dem Bildschirm zu sehen sind. Das Feststellen von Konflikten in der Pausenzuordnung, welches bisher sehr kompliziert gewesen ist, wird dem Benutzer durch das System abgenommen. Er wird sofort nach erfolgter Belegung auf diesen Konflikt hingewiesen.
- Die Berechnung der Aufsichtspflichten wird dem Benutzer durch das System abgenommen.
- Die Pflege des Lehrkörpers ist einfach zu realisieren. Kam im bisherigen System ein Lehrer neu an die Schule, so mußte der Pausenplanersteller mehrfach Lehrerkarten ausfüllen, sowie manuell eine Sortierung vornehmen. Im zukünftigen System beschränken sich die notwendigen Tätigkeiten auf die Eingabe, der für den Lehrer relevanten Informationen.
- Analog zu der Pflege des Lehrkörpers, gestaltet sich die Pflege der Pausen ebenso einfach und effizient. Eine Veränderung in den Pausen führt zur sofortigen Anpassung des Pausenplanes.
- Das Erstellen von Kopien eines Pausenplanes, um ihn beispielsweise im Lehrerzimmer aufzuhängen, oder jedem Lehrer eine Kopie zukommen zu lassen, läßt sich leicht durch mehrfaches Ausdrucken realisieren.

Jedem Benutzer wird durch das Pausenplanersystem ein virtuelles schwarzes Brett am Bildschirm präsentiert. Alle Benutzer müssen sich mit ihren Namen im System anmelden. Das Schwarze Brett ist eine allgemein zugängliche Gruppenarbeitsumgebung, an dem jeder Benutzer die vorgefundenen Materialien betrachten oder bearbeiten kann. Benutzer können die angehefteten Materialien mit den Werkzeugen, welche sie am Schwarzen Brett vorfinden, bearbeiten. Sie können neues Material anheften oder nicht mehr benötigtes Material abnehmen. Jedes Material erhält nach der Bearbeitung durch einen Benutzer einen Stempel, in dem der Zeitpunkt der letzten Bearbeitung und der Name des Benutzers vermerkt ist.

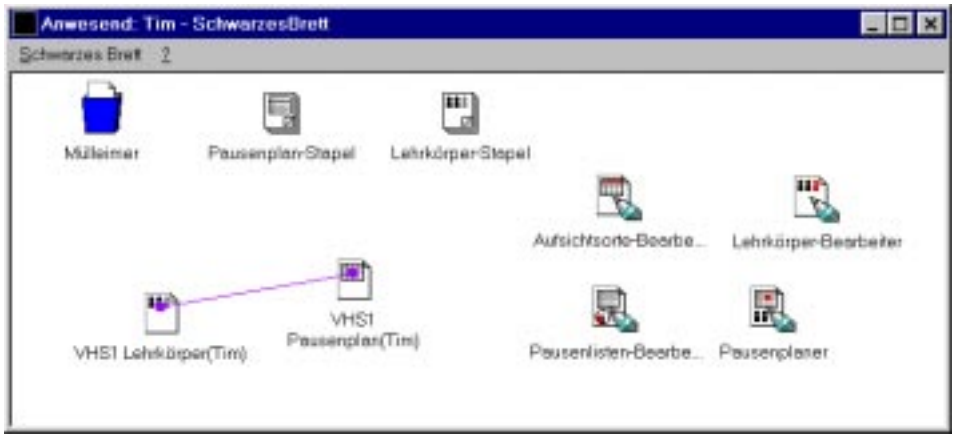


Abbildung 47: Das Schwarze Brett

Am Schwarzen Brett des Pausenplanersystems gibt es die Werkzeuge **Lehrkörper-Bearbeiter**, **Aufsichtsorte-Bearbeiter, Pausenlisten-Bearbeiter** und den **Pausenplaner**. Sie werden durch Symbole repräsentiert. Im Pausenplanersystem gibt es die Materialien Pausenplan und Lehrkörper. Auch sie werden in Form von Symbolen am Schwarzen Brett sichtbar gemacht. Von diesen Materialien kann es mehrere Exemplare geben.

11.2 Einbettungsvision

Da dieser Entwurf im Rahmen unserer Diplomarbeit „Softwareentwicklung nach WAM unter Nutzung des MFC Rahmenwerkes“ erfolgt, ist eine Einbettung des Pausenplanersystems in das MFC Rahmenwerk vorgesehen. Es wird auf dem Betriebssystem Windows NT oder Windows 95 aufsetzen. Das Pausenplanersystem ist ein Mehrbenutzersystem. Mehrere Benutzer können gleichzeitig mit dem System arbeiten. Die Verteilung soll durch die COM Technologie realisiert werden.

11.3 Fachliche Ablaufvision Bearbeiten des Lehrkörpers durch das Sekretariat

Eine Veränderung des Lehrkörpers durch den Sekretariatsmitarbeiter hat zur Folge, daß der Pausenplan auf seine Konsistenz geprüft werden muß. Diese Prüfung wird vom Pausenplanersteller vorgenommen. Er erkennt durch die Farbe des Pausenplansymbols, daß eine Überprüfung durchzuführen ist. Anhand des Materialstempels kann er sehen, welcher Benutzer den Lehrkörper bearbeitet hat und zu welchem Zeitpunkt die Bearbeitung erfolgte. Sobald er die Überprüfung abgeschlossen hat wird das Symbol des Pausenplanes wieder in seiner ursprünglichen Farbe angezeigt. Die neue Darstellung signalisiert dem Sekretariatsmitarbeiter, daß der Pausenplanersteller auf die Veränderung des Lehrkörpers reagiert hat. Er kann auch anhand des Materialstempels erkennen, wer den Lehrkörper zuletzt sondiert hat und zu welchem Zeitpunkt die Betrachtung erfolgte.

11.4 Handhabungsvisionen

11.4.1 Handhabungsvision Erstellen eines neuen Pausenplanes

Um einen neuen Pausenplan zu erstellen, zieht der Pausenplanersteller mit der Maus einen neuen Pausenplan vom Pausenplanstapel des Schwarzen Brettes und läßt ihn auf eine frei Stelle fallen. Es erscheint ein Symbol mit der Bezeichnung „neuer Pausenplan“. Die Bezeichnung des Pausenplanes kann nun geändert werden, indem der Pausenplanersteller die Schrift unterhalb des Symbols mit der Maus anklickt und eine andere Bezeichnung einträgt. Um die Aufsichtsorte der Schule in den Pausenplan einzutragen, startet der Pausenplanersteller die Bearbeitung des neuen Pausenplanes mit dem Aufsichtsorte-Bearbeiter Werkzeug, indem er das Symbol des Pausenplanes auf das Werkzeugsymbol zieht. Das Starten der Bearbeitung kann auch ausgeführt werden, indem er das Werkzeug auf das Material zieht.

Der Pausenplanersteller definiert nun die verschiedenen Pausenaufsichtsorte. Er klickt auf den „Hinzufügen“ Knopf oder wählt im Menü „Aufsichtsort/hinzufügen“. In der Liste der Aufsichtsorte erscheint ein neuer Eintrag mit der Bezeichnung „neuer Aufsichtsort“. Er kann diese Bezeichnung sowie die Mindestanzahl der Pausenaufsichten und Vertreter, welche für diesen Ort als Voreinstellung dienen sollen, verändern. Nachdem der Pausenplanersteller alle Aufsichtsorte in das System eingegeben hat, beendet er die Bearbeitung durch Klicken auf den „Beenden“ Systemknopf. Dieser befindet sich ganz rechts auf der oberen Fensterleiste.

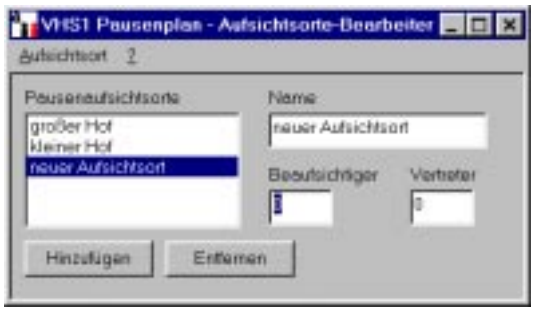


Abbildung 48: Aufsichtsorte-Bearbeiter Werkzeug

Abbildung 49: Pausenlisten-Bearbeiter Werkzeug

Um die Pausen der Schule in den Pausenplan einzutragen, startet der Pausenplanersteller die Bearbeitung des neuen Pausenplanes mit dem Pausenlisten-Bearbeiter Werkzeug, indem er das Symbol des Pausenplanes auf das Werkzeugsymbol zieht.

Nun fügt er die Pausen im Pausenlisten-Bearbeiter Werkzeug hinzu. Der Pausenplanersteller klickt auf den „Hinzufügen“ Knopf oder wählt den Menüpunkt „Pause/Hinzufügen“. In der Pausenliste erscheint daraufhin eine neue Pause mit der Darstellung „neue Pause“. Diese ist sofort selektiert und der Pausenplanersteller kann jetzt, im rechten Teil des Fensters, Wochentag und Anfangs- bzw. Endzeitpunkt mit Stunde und Minute eingeben. Daraufhin ändert sich die Darstellung der Pause in der Pausenliste. Als Voreinstellung sind alle Pausenaufsichtsorte der Schule in der Ortsliste vermerkt. Zu jedem Aufsichtsort gilt zunächst die voreingestellte Anzahl der Pausenaufsichten und Vertreter, so wie sie im Aufsichtsorte-Bearbeiter Werkzeug eingegeben wurde. Der Pausenplanersteller kann nun nicht gewünschte Pausenaufsichtsorte aus der Liste entfernen, in dem er den zu entfernenden Ort in der Liste selektiert und auf den „Entfernen“ Knopf der Pausenaufsichtsorte klickt. Er kann auch die Anzahl der Pausenaufsichten pro Pausenort im „Beaufsichtigter“ Textfeld ändern.

Der Pausenplanersteller beendet die Bearbeitung der Pausenliste, nachdem er alle Pausen hinzugefügt hat. Dazu klickt er auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Pause/Bearbeitung beenden“.

11.4.2 Handhabungsvision Erstellen eines neuen Lehrkörpers

Der Benutzer zieht einen neuen Lehrkörper vom Lehrkörperstapel des Schwarzen Brettes und läßt ihn auf eine freie Stelle fallen. Es erscheint ein neues Lehrkörpersymbol mit dem Namen „neuer Lehrkörper. Die Bezeichnung des Lehrkörpers kann nun geändert werden, indem der Benutzer die Schrift unterhalb des Symbols mit der Maus anklickt und eine andere Bezeichnung einträgt. Danach startet er die Bearbeitung des noch leeren Lehrkörpers mit dem Lehrkörper-Bearbeiter Werkzeug, indem er das Symbol des Lehrkörpers auf das Werkzeug zieht. Nun kann der Benutzer die Lehrer der Schule in den Lehrkörper aufnehmen.

Dazu klickt der Benutzer auf den „Aufnehmen“ Knopf oder er wählt den Menüpunkt „Lehrer/Aufnehmen“. In der Lehrerliste erscheint daraufhin ein neuer Lehrer mit dem Namen „neuer Lehrer“. Dieser ist sofort selektiert und der Benutzer kann jetzt, im rechten Teil des Fensters, den echten Namen und die Stelle des Lehrers eingeben. Daraufhin ändert sich die Darstellung des Namens in der Lehrerliste.

Abbildung 50: Lehrkörper-Bearbeiter Werkzeug

Die Liste der Ausschlußzeiträume ist zunächst leer. Um einen neuen Ausschlußzeitraum für den Lehrer angeben zu können, muß der Benutzer auf den „Zeitraum Hinzufügen“ Knopf klicken oder den Menüpunkt „Ausschlußzeitraum/Hinzufügen“ wählen. In der Liste der Ausschlußzeiträume erscheint daraufhin ein neuer Ausschlußzeitraum mit der Darstellung „neuer Ausschlußzeitraum“. Dieser ist sofort selektiert und der Benutzer kann jetzt im rechten Teil des Fensters den Anfangs- und Endzeitpunkt mit Wochentag, Stunde und Minute eingeben. Daraufhin ändert sich die Darstellung des Zeitraumes in der Liste der Ausschlußzeiträume. Der neue Zeitraum erscheint jetzt in textueller Form gleich den anderen Zeiträume in der Liste. Um einen Ausschlußzeitraum eines Lehrers wieder zu entfernen, muß der Benutzer in der Liste der Ausschlußzeiträume den gewünschten Ausschlußzeitraum selektieren und dann auf den „Zeitraum Entfernen“ Knopf klicken oder den Menüpunkt „Ausschlußzeitraum/Entfernen“ wählen.

Hat der Benutzer alle gewünschten Lehrer in den Lehrkörper aufgenommen, so klickt er auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Lehrer/Bearbeitung beenden“.

11.4.3 Handhabungsvision neuen Pausenplan bearbeiten

Bevor der Pausenplanersteller mit der Bearbeitung des neuen Pausenplanes beginnen kann, muß er den Pausenplan zunächst mit einem Lehrkörper am Schwarzen Brett verknüpfen. Dazu zieht er das Lehrkörpersymbol mittels der Maus auf des Pausenplansymbol. Es erscheint eine Linie zwischen dem Symbol des Pausenplanes und dem Symbol des Lehrkörpers.

Der Pausenplanersteller startet nun die Bearbeitung des Materials Pausenplan mit dem Pausenplaner Werkzeug. Dazu zieht er das Symbol des Pausenplanes auf das Symbol des Pausenplaners. Der Lehrkörper und der noch leere Pausenplan werden daraufhin im Werkzeug dargestellt.

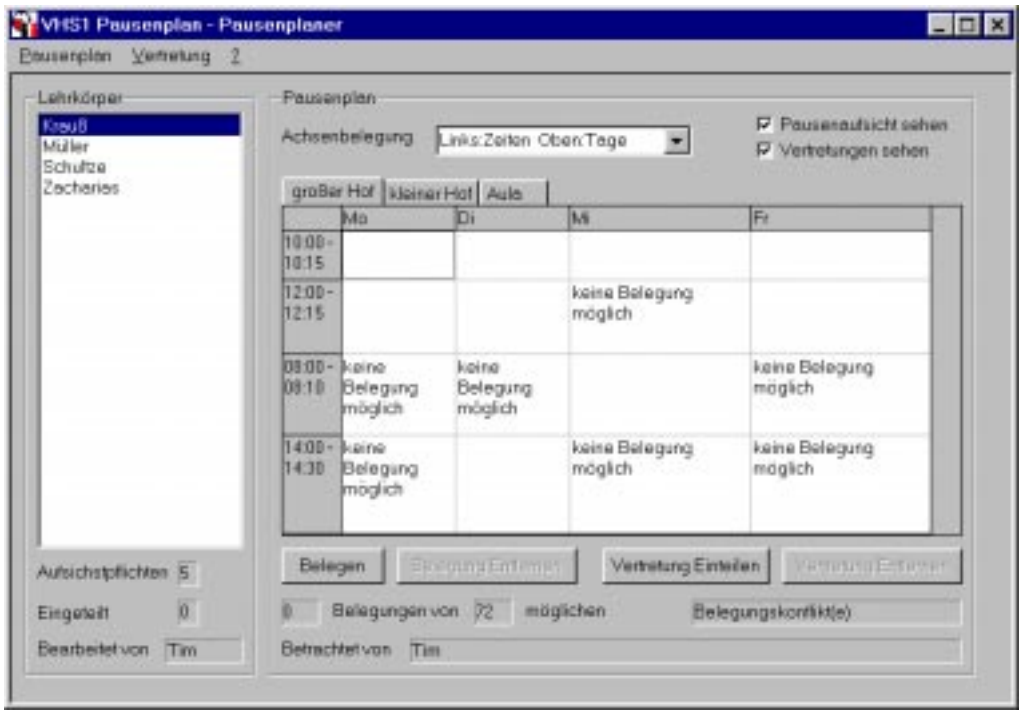


Abbildung 51: Pausenplaner Werkzeug

Der Benutzer kann die Darstellung von Pausen, Wochentagen und Orten im Pausenplan beliebig einstellen. Dazu klickt er mit der Maus auf die Auswahlbox „Achsenbelegung“ und kann die gewünschten Belegung für die x-, y- und z-Achse. Die Materialpräsentation des Pausenplans ändert sich daraufhin sofort. Abbildung 51 zeigt die Einstellung „Links: Zeiten Oben: Tage“. Mit dieser Einstellung befinden sich die Pausenaufsichtsorte auf den Aktenreitern. Der Pausenplanersteller wählt nun den zu bearbeitenden Aufsichtsort, indem er mit der Maus auf den gewünschten Karteireiter klickt.

Jetzt kann der Pausenplanersteller die Pausenbelegung durchführen. Um eine Pause mit einem Lehrer aus dem Lehrkörper zu belegen, muß der Pausenplanersteller in der Lehrerliste einen Lehrer selektieren. Das Werkzeug zeigt an, wie viele Aufsichtspflichten der Lehrer hat und wie oft er schon eingeteilt wurde. Im Pausenraster werden die Felder, in denen der Lehrer Ausschlußzeiträume hat, farbig hervorgehoben. Jetzt kann der Pausenplanersteller mittels Drag&Drop den Lehrer auf ein Feld des Pausenplanes ziehen. Bei diesem Vorgang kann der Pausenplanersteller zusätzlich, anhand der Darstellung des Mauszeigers erkennen, ob ein Belegen des Feldes überhaupt möglich ist, oder ob ein Konflikt besteht. Das Belegen der Pause mit dem Lehrer ist im Konfliktfall dennoch möglich. Der Lehrername wird dann in diesem Feld farbig hervorgehoben.

Zieht der Pausenplanersteller den Lehrer auf ein schon belegtes Feld, so wird die alte Belegung vorher entfernt. Der Pausenplanersteller kann auch Belegungen innerhalb des Pausenplans mittels Drag&Drop verschieben. Ein Ziehen auf ein schon belegtes Feld bewirkt dann einen Lehrertausch der Pausenbelegungen.

Der Pausenplanersteller wird im Pausenplaner Werkzeug über den Zustand des Pausenplanes informiert. Ihm wird angezeigt, wie viele Pausenbelegungen für eine vollständige Pausenplanung notwendig sind, die Anzahl der bereits getätigten Pausenbelegungen, sowie die Anzahl der vorliegenden Belegungskonflikte. Belegungskonflikte eines Pausenplanes werden auch im Symbol des Pausenplan-Materials am Schwarzen Brett sichtbar.

Der Pausenplanersteller beendet die Bearbeitung des Pausenplanes, nachdem er die notwendigen Pausenbelegungen durchgeführt hat. Dazu klickt er auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Pause/Bearbeitung beenden“.

11.5 Vision der Handhabungskomponente Schwarzes Brett

Die Gruppenarbeitsumgebung des Pausenplanersystems ist ein allgemein zugänglicher Ort. Benutzer können sich dorthin bewegen und sich die Materialien betrachten oder bearbeiten. Es können sich gleichzeitig mehrere Benutzer dort einfinden um verschiedene Tätigkeiten auszuführen. Sie müssen, sofern sie gleichzeitig aktiv sind, miteinander kooperieren. Benutzern wird vergegenwärtigt, welche anderen Benutzer sich zur selben Zeit am Schwarzen Brett befinden. Daher müssen Benutzer im System namentlich bekannt sein. Die Benutzer können die angehefteten Materialien mit den Werkzeugen, welche sie am Schwarzen Brett vorfinden, bearbeiten. Sie können neues Material anheften oder nicht mehr benötigtes Material abnehmen.

Die Handhabungskomponente Schwarzes Brett soll synchrone und asynchrone Kooperation der Benutzer ermöglichen. Man kann am eigenen Bildschirm erkennen, welche anderen Benutzer sich zur selben Zeit in der Gruppenarbeitsumgebung befinden und welche Materialien sie bearbeiten. Die Benutzer können nacheinander oder simultan auf die Materialien zugreifen. Gemeinsamer lesender sowie schreibender Zugriff auf ein Material ist möglich. Die gemeinsame Bearbeitung eines Materials, kann den beteiligten Benutzern durch die Handhabungskomponente mitgeteilt werden, wenn dies im Anwendungskontext sinnvoll erscheint.

Über die Verwaltungskomponente der Gruppenarbeitsumgebung werden einzelne Veränderungen an einem Material durch ein bearbeitendes Werkzeug, wie z.B. eine Pausenbelegung, den anderen Werkzeugen des Pausenplanersystems, die das gleiche Material betrachten, sofort gemeldet. Alle Benutzer sehen somit immer den aktuellen Materialzustand.

11.5.1 Oberflächendesign

Jedem Benutzer wird durch das Pausenplanersystem ein virtuelles Schwarzes Brett am Bildschirm präsentiert. In der oberen Fensterleiste werden die Namen der Benutzer aufgelistet, die sich zur selben Zeit ebenfalls am Brett befinden. Werkzeuge, Materialien und Materialstapel werden durch Symbole repräsentiert. Diese sind mit Namen versehen, die unter den Symbolen sichtbar sind. Befindet sich ein Material in Bearbeitung, so erscheint unter dem Symbol in Klammern auch die Namen der Bearbeiter. In Abbildung 52 befinden sich Andreas und Tim am Schwarzen Brett.

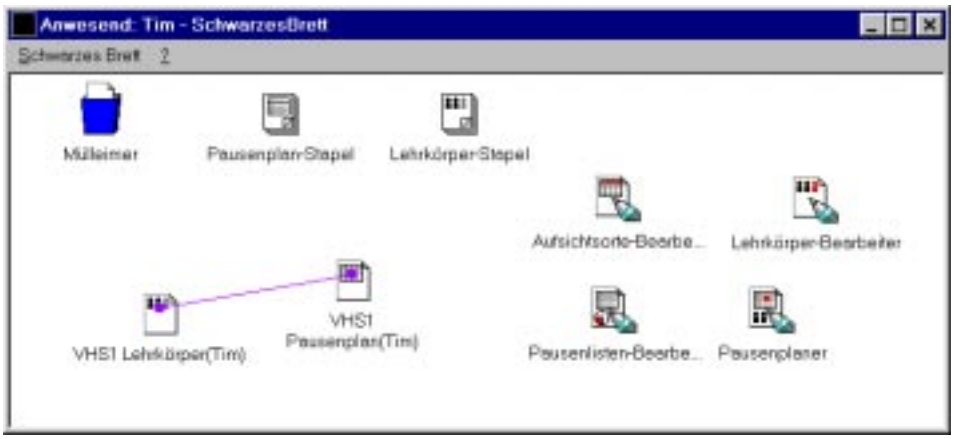


Abbildung 52: Das Schwarze Brett

Jedes Symbol auf dem Schwarzen Brett hat ein Kontextmenü. Dieses wird durch Klicken mit der rechten Maustaste auf das Symbol sichtbar. Im Kontextmenü der Materialien befinden sich weiterhin Informationen über den Zeitpunkt der letzten Bearbeitung, ebenso die Namen der Bearbeiter. Diese Informationen werden aktualisiert, sobald der letzte Benutzer die Bearbeitung des Materials abgeschlossen hat, so können andere Benutzer erkennen, wer zuletzt das Material bearbeitet.

11.5.2 Aktionen des Benutzers

Neue Materialien anheften:
Will der Benutzer neue Materialien an das Schwarze Brett anheften, so zieht er mit der Maus vom gewünschten Materialstapel ein neues Material und läßt es an einer freien Stelle fallen. Zu jedem Materialtyp gibt es einen eigenen Stapel. Es erscheint ein Symbol mit der Bezeichnung „neuer/neue <Materialtyp>“. Die Bezeichnung des Materials kann nun geändert werden, indem der Benutzer die Schrift unterhalb des Symbols mit der Maus anklickt und eine andere Bezeichnung einträgt.

Materialien abnehmen:
Will der Benutzer nicht mehr benötigte Materialien vom Schwarze Brett abnehmen, so zieht er mit der Maus das gewünschte Material auf das Papierkorb Symbol. Eine Sicherheitsabfrage folgt und fragt den Benutzer, ob er das Material tatsächlich vernichten will. Materialien, welche eine Verknüpfung besitzen können nicht vom Schwarzen Brett abgenommen werden. Will der Benutzer verknüpfte Materialien abnehmen, so muß er zuvor die Verknüpfung lösen.

Material bearbeiten oder Betrachten:
Das Bearbeiten oder Betrachten eines Materials durch ein Werkzeug kann durch einen Drag&Drop Vorgang mit den Symbolen auf dem Schwarzen Brett initiiert werden. Es wird entweder ein Werkzeug auf ein Material gezogen oder ein Material auf ein Werkzeug. Kann ein Werkzeug ein Material nicht bearbeiten so wird dies angezeigt. Andernfalls öffnet sich das Werkzeug und die Bearbeitung des Materials kann beginnen.

Materialien verknüpfen:
Im Pausenplan System ist nur die Verknüpfung von Pausenplan und Lehrkörper vorgesehen. Der Benutzer initiiert die Verknüpfung, indem er den Lehrkörper mittels der Maus auf den Pausenplan zieht. Der Pausenplan darf vorher keine Verknüpfung besitzen. Es erscheint eine Linie zwischen dem Symbol des Pausenplanes und dem Symbol des Lehrkörpers. Der Pausenplan benutzt nach einer getätigten Verknüpfung den Lehrkörper. Lehrer aus dem Lehrkörper können nun die Pausen des Pausenplanes beaufsichtigen. Ein Lehrkörper kann mit mehreren Pausenplänen verknüpft sein.

Verknüpfung von Materialien lösen:
Will der Benutzer nicht mehr benötigte Verknüpfungen zwischen Pausenplan und Lehrkörper lösen, so klickt er mit der rechten Maustaste auf das Symbol des beteiligten Pausenplan-Materials. Daraufhin erscheint das Kontextmenü und der Benutzer wählt dann „Verknüpfung lösen“. Eine Sicherheitsabfrage folgt und fragt den Benutzer, ob er die Verknüpfung tatsächlich lösen will.

Schwarzes Brett verlassen:
Will der Benutzer die Bearbeitung oder Betrachtung der Materialien beenden, so klickt er auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Aktionen/Schwarzes Brett verlassen“. Ein Verlassen ist nur möglich, wenn zum Zeitpunkt des Verlassens keine Materialien bearbeitet werden. Ist dies dennoch der Fall so muß der Benutzer die Bearbeitung des Materials beenden.

11.6 Werkzeugvisionen

11.6.1 Werkzeugvision Lehrkörper-Bearbeiter

Mit dem Lehrkörper-Bearbeiter Werkzeug kann der Benutzer das Material Lehrkörper bearbeiten. Neue Lehrer können in den Lehrkörper aufgenommen werden. Von der Schule abgehende Lehrer können aus dem Lehrkörper entfernt werden. Name, Stelle und Ausschlußzeiträume eines Lehrers können verändert werden. Der Lehrkörper wird als Teil des Materials Pausenplan verwendet.

11.6.1.1 Materialpräsentation

Eine Liste aller Lehrer, die sich im Lehrkörper befinden, wird auf der linken Seite des Fensters gezeigt. Die Namen der Lehrer werden alphabetisch geordnet aufgelistet. Der Benutzer kann in der Liste einen Lehrernamen selektieren. Dadurch werden Name, Stelle und Ausschlußzeiträume des selektierten Lehrers auf der rechten Seite des Fensters gezeigt. Die Ausschlußzeiträume werden in einer Liste temporär geordnet dargestellt. Der Benutzer kann im unteren Teil des Werkzeuges sehen, wer das Lehrkörper-Material im Moment bearbeitet.

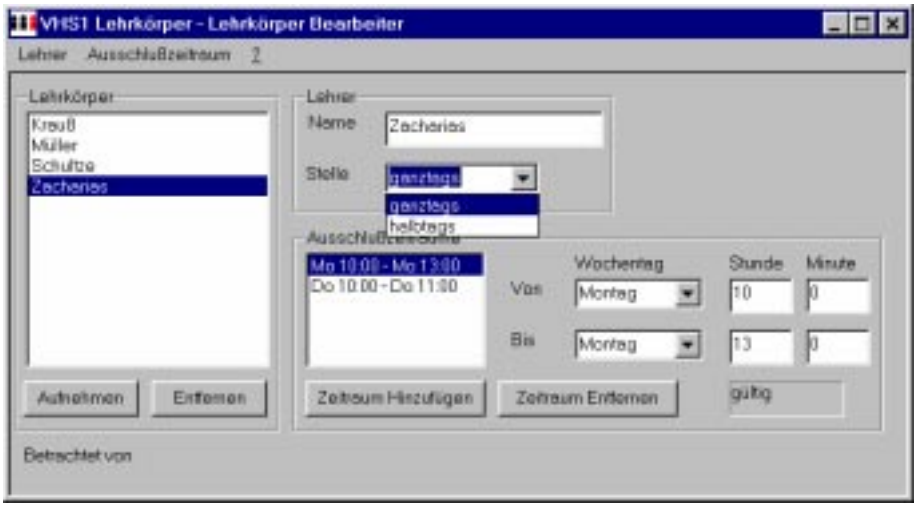


Abbildung 53: Lehrkörper-Bearbeiter Werkzeug

11.6.1.2 Aktionen des Benutzers

Neuen Lehrer in den Lehrkörper aufnehmen:
Der Benutzer klickt auf den „Aufnehmen“ Knopf oder er wählt den Menüpunkt „Lehrer/Aufnehmen“. In der Lehrerliste erscheint daraufhin ein neuer Lehrer mit dem Namen „neuer Lehrer“. Dieser ist sofort selektiert und der Benutzer kann jetzt, im rechten Teil des Fensters, den echten Namen und die Stelle des Lehrers eingeben. Daraufhin ändert sich die Darstellung des Namens in der Lehrerliste. Die Liste der Ausschlußzeiträume ist zunächst leer. Um einen neuen Ausschlußzeitraum für den Lehrer angeben zu können, muß der Benutzer auf den „Zeitraum Hinzufügen“ Knopf klicken oder den Menüpunkt „Ausschlußzeitraum/Hinzufügen“ wählen. (siehe Operation Ausschlußzeitraum eines Lehrers hinzufügen)

Lehrer aus dem Lehrkörper entfernen:
Um einen Lehrer aus dem Lehrkörper zu entfernen, muß der Benutzer in der Lehrerliste einen Lehrer selektieren und dann auf den „Entfernen“ Knopf klicken oder den Menüpunkt „Lehrer/Entfernen“ wählen. Eine Sicherheitsabfrage folgt und fragt den Benutzer, ob der Lehrer tatsächlich entfernt werden soll.

Ausschlußzeitraum eines Lehrers hinzufügen:
Der Benutzer klickt auf den „Zeitraum Hinzufügen“ Knopf oder er wählt den Menüpunkt „Ausschlußzeitraum/Hinzufügen“. In der Liste der Ausschlußzeiträume erscheint daraufhin ein neuer Ausschlußzeitraum mit der Darstellung „neuer Ausschlußzeitraum“. Dieser ist sofort selektiert und der Benutzer kann jetzt im rechten Teil des Fensters den Anfangs- und Endzeitpunkt mit Wochentag, Stunde und Minute eingeben. Daraufhin ändert sich die Darstellung des Zeitraumes in der Liste der Ausschlußzeiträume. Der neue Zeitraum erscheint jetzt in textueller Form gleich den anderen Zeiträume in der Liste.

Ausschlußzeitraum eines Lehrers entfernen:
Um einen Ausschlußzeitraum eines Lehrers zu entfernen, muß der Benutzer in der Liste der Ausschlußzeiträume den gewünschten Ausschlußzeitraum selektieren und dann auf den „Zeitraum Entfernen“ Knopf klikken oder den Menüpunkt „Ausschlußzeitraum/Entfernen“ wählen.

Name eines Lehrers verändern:
Um den Namen eines Lehrers zu verändern, muß der Benutzer in der Lehrerliste den gewünschten Lehrer selektieren. Daraufhin werden die Eigenschaften des ausgewählten Lehrers angezeigt. Jetzt kann der Name in dem Textfeld verändert werden. Die Änderung erscheint sofort in der Lehrerliste.

Stelle eines Lehrers verändern:
Um die Stelle eines Lehrers zu verändern, muß der Benutzer in der Lehrerliste den gewünschten Lehrer selektieren.
Daraufhin werden die Eigenschaften des ausgewählten Lehrers angezeigt. Jetzt kann die Art der Stelle in der Auswahlbox verändert werden.

Ausschlußzeitraum eines Lehrers verändern:
Um den Ausschlußzeitraum eines Lehrers zu verändern muß der Benutzer in der Lehrerliste den gewünschten Lehrer selektieren. Daraufhin werden die Ausschlußzeiträume des ausgewählten Lehrers innerhalb der Liste der Ausschlußzeiträume angezeigt. Der Benutzer wählt nun den gewünschten Ausschußzeitraum aus dieser Liste. Jetzt können Anfangs- und Endzeitpunkt des Ausschlußzeitraums in den Textfeldern und den Auswahlboxen verändert werden. Die Änderungen erscheinen sofort in der Liste der Ausschlußzeiträume.

Bearbeitung beenden:
Der Benutzer klickt auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Lehrer/Bearbeitung beenden“.

11.6.2 Werkzeugvision Pausenlisten-Bearbeiter

Mit dem Pausenlisten-Bearbeiter Werkzeug kann der Pausenplanersteller die Pausenliste bestehender Pausenpläne pflegen. Pausen können zur Pausenliste hinzugefügt oder entfernt werden. Der Zeitraum, die Aufsichtsorte und die Anzahl der Pausenaufsichten einer Pause können verändert werden. Die Pausenliste wird als Teil des Materials Pausenplan verwendet.

11.6.2.1 Materialpräsentation

Eine Liste aller Pausen wird auf der linken Seite des Fensters gezeigt. Die Zeiträume der Pausen werden temporär geordnet aufgelistet. Der Pausenplanersteller kann in der Liste eine Pause selektieren. Dadurch werden Wochentag, Zeitraum und eine Liste der Pausenaufsichtsorte der selektierten Pause auf der rechten Seite des Fensters gezeigt. Die Liste der Pausenaufsichtsorte enthält neben den Namen derselben ebenso die Anzahl der Pausenaufsichten. Selektiert der Pausenplanersteller einen Ort aus dieser Liste, so wird die Anzahl der Pausenaufsichten in einem Textfeld angezeigt und kann dort verändert werden.

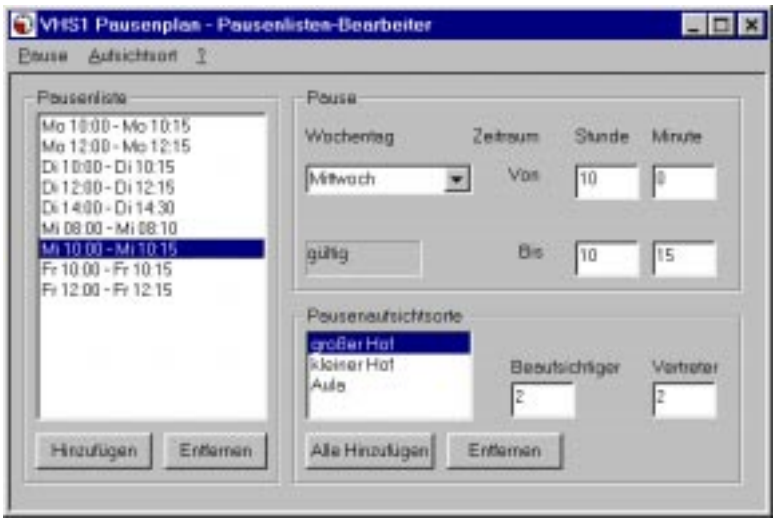


Abbildung 54: Pausenlisten-Bearbeiter

11.6.2.2 Aktionen des Pausenplanerstellers

Pause zur Pausenliste hinzufügen:
Der Pausenplanersteller klickt auf den „Hinzufügen“ Knopf oder wählt den Menüpunkt „Pause/Hinzufügen“. In der Pausenliste erscheint daraufhin eine neue Pause mit der Darstellung „neue Pause“. Diese ist sofort

selektiert und der Pausenplanersteller kann jetzt, im rechten Teil des Fensters, Wochentag und Anfangs- bzw. Endzeitpunkt mit Stunde und Minute eingeben. Daraufhin ändert sich die Darstellung der Pause in der Pausenliste. Als Voreinstellung sind alle Pausenaufsichtsorte der Schule in der Ortsliste vermerkt. Zu jedem Aufsichtsort gilt die voreingestellte Anzahl der Pausenaufsichten und Vertreter. Der Pausenplanersteller kann nun die Pausenaufsichtsorte, die in dieser Pause nicht beaufsichtigt werden müssen, aus der Liste entfernen (siehe unten „Pausenaufsichtsort einer Pause entfernen“). Er kann auch die Anzahl der Pausenaufsichten pro Pausenort ändern (siehe unten „Anzahl der Pausenaufsichten an einem Pausenaufsichtsort ändern“).

Pause aus der Pausenliste entfernen:
Um eine Pause aus der Pausenliste zu entfernen, muß der Pausenplanersteller in der Pausenliste eine Pause selektieren und dann auf den „Entfernen“ Knopf klicken oder den Menüpunkt „Pause/Entfernen“ wählen.

Zeitraum einer Pause verändern:
Um den Zeitraum einer Pause zu verändern, muß der Pausenplanersteller in der Pausenliste die gewünschte Pause selektieren. Daraufhin wird der Zeitraum der ausgewählten Pause angezeigt. Der Pausenplanersteller kann nun den Wochentag in der Auswahlbox verändern. Anfangs- und Endzeitpunkt der Pause können in den Textfeldern verändert werden. Die Änderungen erscheinen sofort in der Pausenliste.

Alle Pausenaufsichtsorte einer Pause hinzufügen:
Um alle an der Schule existierenden Pausenaufsichtsorte zu einer Pause hinzuzufügen, muß der Pausenplanersteller in der Pausenliste eine Pause selektieren und dann auf den „Alle Hinzufügen“ Knopf klicken oder den Menüpunkt „Aufsichtsort/Alle Hinzufügen“ wählen.

Pausenaufsichtsort einer Pause entfernen:
Der Pausenplanersteller selektiert in der Pausenliste die gewünschte Pause. Daraufhin werden alle Pausenaufsichtsorte angezeigt. Der Pausenplanersteller selektiert in der Liste den zu entfernenden Ort und klickt auf den „Entfernen“ Knopf der Pausenaufsichtsorte oder wählt im Menü „Aufsichtsort/Aufsichtsort entfernen“. Der Pausenaufsichtsort wird gelöscht.

Anzahl der Pausenaufsichten oder Vertreter an einem Pausenaufsichtsort ändern:
Der Pausenplanersteller selektiert in der Pausenliste die gewünschte Pause. Daraufhin werden alle Pausenaufsichtsorte angezeigt. Der Pausenplanersteller selektiert in der Liste den gewünschten Ort und kann nun die Anzahl der Pausenaufsichten oder Vertreter im Textfeld ändern.

Bearbeitung beenden:
Der Pausenplanersteller klickt auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Pause/Bearbeitung beenden“.

11.6.3 Werkzeugvision Aufsichtsorte-Bearbeiter

Mit dem Aufsichtsorte-Bearbeiter Werkzeug kann der Pausenplanersteller die Orte, an denen in der Schule Pausenaufsichten wahrgenommen werden müssen, eingeben und pflegen. Er kann neue Pausenaufsichtsorte hinzufügen oder entfernen. Zu jedem Pausenaufsichtsort kann er die Anzahl der voreingestellten Pausenaufsichten und Vertreter festlegen.

11.6.3.1 Materialpräsentation

Eine Liste aller Pausenaufsichtsorte wird auf der linken Seite des Fensters gezeigt. Die Namen der Orte werden alphabetisch geordnet aufgelistet. Der Pausenplanersteller kann in der Liste einen Pausenort selektieren. Dadurch werden Name und die Anzahl der Pausenaufsichten und Vertreter des selektierten Ortes auf der rechten Seite des Fensters gezeigt.

Abbildung 55: Aufsichtsorte-Bearbeiter Werkzeug

11.6.3.2 Aktionen des Pausenplanerstellers

Pausenaufsichtsort hinzufügen:
Der Pausenplanersteller klickt auf den „Hinzufügen“ Knopf oder er wählt den Menüpunkt „Pausenaufsichtsort/Hinzufügen“. In der Liste erscheint daraufhin eine neuer Ort mit der Darstellung „neuer Aufsichtsort“. Dieser ist sofort selektiert und der Pausenplanersteller kann jetzt, im rechten Teil des Fensters, den Namen und die voreingestellte Anzahl der Pausenaufsichten eingeben. Daraufhin ändert sich die Darstellung des Ortes in der Liste.

Pausenaufsichtsort entfernen:
Um einen Ort aus der Liste zu entfernen, muß der Pausenplanersteller in der Liste einen Pausenaufsichtsort selektieren und dann auf den „Entfernen“ Knopf klicken oder den Menüpunkt „Pausenaufsichtsort/Entfernen“ wählen.

Namen eines Pausenaufsichtsortes verändern:
Der Pausenplanersteller muß in der Liste den gewünschten Pausenaufsichtsort selektieren. Daraufhin wird der Name des ausgewählten Ortes in der Textbox angezeigt. Der Pausenplanersteller kann nun den Namen in der Textbox verändern. Die Änderungen erscheinen sofort in der Liste.

Anzahl der voreingestellten Pausenaufsichten und Vertreter an einem Pausenaufsichtsort ändern:
Der Pausenplanersteller muß in der Liste den gewünschten Pausenaufsichtsort selektieren. Daraufhin wird die Anzahl der voreingestellten Pausenaufsichten und Vertreter des ausgewählten Ortes in den Textboxen angezeigt. Der Pausenplanersteller kann nun die Anzahl innerhalb der Textboxen verändern.

Bearbeitung beenden:
Der Pausenplanersteller klickt auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Pausenaufsichtsort/Bearbeitung beenden“.

11.6.4 Werkzeugvision Pausenplaner

Mit dem Pausenplaner Werkzeug kann der Pausenplanersteller alle Pausenaufsichten der Schule an die Lehrer des Lehrkörpers vergeben. Die Vergabe wird im Material festgehalten. Eine Pausenaufsicht vergibt der Pausenplanersteller dadurch, das er eine Pause aus der Pausenliste mit einem Lehrer des Lehrkörpers belegt. Der Pausenplanersteller kann Belegungskonflikte erkennen und daraufhin Belegungen entfernen oder tauschen. Die verwendete Pausenliste des Pausenplans wird im Pausenlisten-Bearbeiter Werkzeug verwaltet. Der verwendete Lehrkörper des Pausenplans wird im Lehrkörper-Bearbeiter Werkzeug verwaltet.

11.6.4.1 Materialpräsentation

Der Lehrkörper des Pausenplans wird in einer Liste auf der linken Seite des Fensters gezeigt. Die Namen der Lehrer werden dort alphabetisch geordnet aufgelistet. Unter der Liste werden die Aufsichtspflichten des gerade selektierten Lehrers angezeigt. Man kann auch sehen, wie viele Aufsichtspflichten man schon eingeteilt hat. Darunter kann man sehen, wer das Lehrkörper-Material im Moment Bearbeitet.

Der Pausenplan wird in einem Raster auf der rechten Seite des Fensters gezeigt. Zeilen und Spalten dieses Rasters sind dynamisch veränderbar und passen sich der Pausenliste des Pausenplans an. Das bedeutet, daß im Plan beliebige Wochentage, Aufsichtsorte und Zeiträume für Pausen dargestellt werden können. Pausen mit gleichen Zeiträumen werden in einer Zeile des Rasters dargestellt. Dadurch kann es Beispielsweise vorkommen, daß einzelne Felder einer Zeile gesperrt sind, da an den Wochentagen dieser Felder keine Pause zu dieser Zeit vorgesehen ist.

Der Benutzer kann die Darstellung von Pausen, Wochentagen und Orten im Pausenplan beliebig einstellen. Dazu klickt er mit der Maus auf die Auswahlbox „Achsenbelegung“ und kann die gewünschten Belegung für die x-, y- und z-Achse wählen. Die Materialpräsentation des Pausenplans ändert sich daraufhin sofort. Abbildung 56 zeigt die Einstellung „Links: Zeiten Oben: Tage“. Mit dieser Einstellung befinden sich die Pausenaufsichtsorte auf den Aktenreitern. Unter den Knöpfen „Belegen“ und „Belegung Entfernen“ wird die Gesamtzahl der Belegungen des Pausenplans und die Anzahl der schon getätigten Belegungen aufgezeigt. Hier kann man auch sehen, ob Belegungskonflikte bestehen oder nicht. Darunter kann der Pausenplanersteller sehen, wer das Pausenplan-Material im Moment betrachtet.

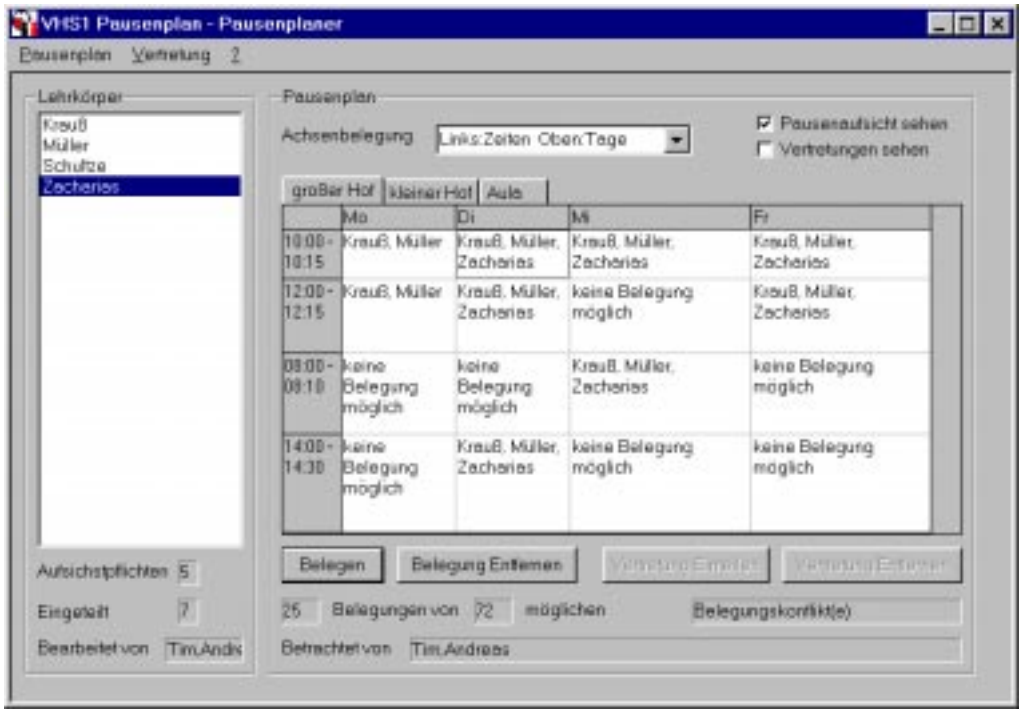


Abbildung 56: Pausenplaner Werkzeug

11.6.4.2 Aktionen des Pausenplanerstellers

Pause mit Lehrer belegen:
Um eine Pause mit einem Lehrer aus dem Lehrkörper zu belegen, muß der Pausenplanersteller in der Lehrerliste einen Lehrer selektieren. Das Werkzeug zeigt an, wie viele Aufsichtspflichten der Lehrer hat und wie oft er schon eingeteilt wurde. Im Pausenraster werden die Felder, in denen der Lehrer Ausschlußzeiträume hat, farbig hervorgehoben. Jetzt kann der Pausenplanersteller mittels Drag&Drop den Lehrer auf ein Feld des Pausenplanes ziehen. Bei diesem Vorgang kann der Pausenplanersteller zusätzlich, anhand der Darstellung des Mauszeigers erkennen, ob ein Belegen des Feldes überhaupt möglich ist, oder ob ein Konflikt besteht. Das Belegen der Pause mit dem Lehrer ist im Konfliktfall dennoch möglich. Das Feld wird dann farbig hervorgehoben. Zieht der Pausenplanersteller den Lehrer auf ein schon belegtes Feld, so wird die alte Belegung vorher entfernt. Der Pausenplanersteller kann auch Belegungen innerhalb des Pausenplans mittels Drag&Drop verschieben. Ein Ziehen auf ein schon belegtes Feld bewirkt dann einen Lehrertausch der Pausenbelegungen.

Mehrere Pausen mit einem Lehrer belegen:

Der Pausenplanersteller selektiert ein oder mehrere Felder des Pausenplanes und klickt dann auf den „Belegen“ Knopf oder er wählt den Menüpunkt „Pause/Mit Lehrer belegen“. Die ausgewählten Pausen werden jetzt mit dem in der Lehrerliste selektierten Lehrer belegt. Alte Belegungen werden vorher entfernt.

Belegung einer Pause entfernen:
Der Pausenplanersteller selektiert ein oder mehrere Felder des Pausenplanes und klickt dann auf den „Entfernen“ Knopf oder wählt den Menüpunkt „Pause/Belegung entfernen“. Die Belegung sämtlicher ausgewählter Pausen wird daraufhin entfernt.

Alle Pausenbelegungen entfernen:
Der Pausenplanersteller wählt den Menüpunkt „Pausenplan/Alle Belegungen entfernen“. Nach einer Sicherheitsabfrage, ob der Pausenplanersteller den Pausenplan tatsächlich leeren möchte, werden sämtliche Pausenbelegungen entfernt.

Auswahl eines Pausenaufsichtsortes (Vorausgesetzte Einstellung „Links: Zeiten Oben: Tage“)**:**
Über dem Pausenplan gibt es für jeden Pausenaufsichtsort der Schule einen Aktenreiter. Die Namen der Aufsichtsorte sind auf den Aktenreitern zu sehen. Der Pausenplanersteller wählt den gewünschten Pausenaufsichtsort durch einen Klick auf den richtigen Aktenreiter. Daraufhin erscheint der Pausenplan für den ausgewählten Ort[17].

Pausenplan drucken:
Der Pausenplanersteller wählt den Menüpunkt „Pausenplan/Drucken“. Der Pausenplan wird ausgedruckt.

Bearbeitung beenden:
Der Pausenplanersteller klickt auf den „Beenden“ Systemknopf oder wählt den Menüpunkt „Pausenplan/Bearbeitung beenden“.

11.7 Materialvisionen

11.7.1 Materialvision Lehrer

Das Material Lehrer repräsentiert den Lehrer einer Schule. Dieser Lehrer ist charakterisiert durch seinen Namen und die Stelle, die er bekleidet. Neben der Aufgabe, an der Schule zu unterrichten ist er, im Rahmen seiner Stelle, verpflichtet, eine gewisse Anzahl von Pausenaufsichten zu führen. Es gibt ganze, halbe und viertel Stellen. Aufgrund seiner Stelle, oder auch anderer Gründe, ist ein Lehrer nicht immer an der Schule anwesend. Die Informationen über die Zeiten, zu denen er nicht verfügbar ist, werden in Ausschlußzeiträumen festgehalten.

- Veränderung des Lehrers
  Die Eigenschaften eines Lehrers werden im Rahmen des Pausenplaner-Systems bearbeitet. Name und Stelle des Lehrers soll der Benutzer mit Hilfe eines Werkzeuges verändern können. Ebenso kann er die Ausschlußzeiträume des Lehrers bearbeiteten. Es können Ausschlußzeiträume hinzugefügt und entfernt werden.
- Verfügbarkeit eines Lehrers
  Soll ein Lehrer die Aufsicht einer Pause übernehmen, so wird die Pause mit ihm belegt. In diesem Zusammenhang kann der Lehrer darüber Auskunft geben, ob er innerhalb eines Zeitraumes einsatzfähig ist. So würde es wenig Sinn machen, einen Lehrer für die Pause von 8.40 einzuteilen, obwohl er erst in der vierten Stunde (um 11.00) das erste Mal Unterricht gibt und demnach zur Pausenzeit nicht anwesend ist.

11.7.2 Materialvision Aufsichtsort

[17] Es sollte evtl. möglich sein ein zweites Fenster des Pausenplans zu öffnen, um zwei Pausenaufsichtsorte parallel zu sehen und zu bearbeiten.

Ein Pausenaufsichtsort ist ein Bereich in einer Schule, in dem sich die Schüler während der Pause aufhalten. In einer Pause muß eine definierte Mindestanzahl von Pausenaufsichten zugegen sein. Die Anzahl der einzuteilenden Vertreter stimmt in der Regel mit diesem Wert überein.

- Veränderung der Mindestanzahl von Pausenaufsichten.
  Die Mindestanzahl von Pausenaufsichten muß verändert werden können.
- Veränderung der Bezeichnung eines Aufsichtsortes.

11.7.3 Materialvision Pausenaufsichtsort

Ein Pausenaufsichtsort ist ein Material, welches sich auf einen Aufsichtsort bezieht und in einer Pause mit einer Mange von Beaufsichtigern und Vertretern belegt wird.

- Veränderung der Mindestanzahl von Pausenaufsichten.
  Die Mindestanzahl von Pausenaufsichten muß verändert werden können. Geschieht dieses, so wird die durch den Aufsichtsort vorgegebene Voreinstellung überschrieben.
- Zuordnung von Lehrern
  Einem Pausenaufsichtsort, als Eigenschaft einer Pause, werden eine Menge von Lehrern zugeordnet.
- Zuordnung von Vertretern
  Dem Pausenaufsichtsort muß ebenso wie Lehrer, eine Menge von Vertretern zugeordnet werden können.

11.7.4 Materialvision Pause

Eine Pause bezieht sich auf einen wöchentlich wiederkehrenden Zeitraum[18]. Sie ist dadurch gekennzeichnet, daß dieser Zeitraum beaufsichtigt werden muß. An einer Schule kann es mehrere Aufsichtsorte geben. Es existieren beispielsweise verschiedene Pausenhöfe, beziehungsweise es muß sowohl innerhalb als auch außerhalb des Schulgebäudes Aufsicht geführt werden. Die Beaufsichtigung eines Pausenaufsichtsortes erfordert einen oder mehrere Lehrer. Im Krankheitsfall einer Pausenaufsicht muß eine Vertretung eingeteilt sein. Diese übernimmt dann die Beaufsichtigung der Pause. Die Mindestanzahl von Pausenaufsichten kann durch eine untere Grenze festgelegt werden. Diese ist mit der Anzahl der eingeteilten Vertretungen identisch.
Eine Pausenaufsicht wird eingeteilt, indem ein Aufsichtsort einer Pause mit Lehrern belegt wird.

- Veränderung einer Pause
  Da jede Schule eine eigene Pauseneinteilung vornimmt, muß der Zeitraum und die Mindestanzahl von Pausenaufsichten einer Pause bearbeitet werden können. Diese Eigenschaften sollen auch nachträglich verändert werden können.
- Belegung eines Aufsichtsortes einer Pause mit einem Lehrer (Pausenaufsicht oder Vertretung der Pausenaufsicht)
  Im Falle der Belegung einer Pause mit einem Lehrer, wird der Lehrer bei der Pause mit dem gewünschten Pausenaufsichtsort eingetragen.
- Feststellen eines Konfliktes
  Durch die Belegung eines Aufsichtsortes einer Pause kann ein Belegungskonflikt entstehen, wenn sich die Pausenzeit mit einem Ausschlußzeitraum des Lehrers überschneidet. Die Pause kann diesen Konflikt feststellen.

11.7.5 Materialvision Lehrkörper

Der Lehrkörper stellt das Lehrerpersonal an einer Schule dar. Der Lehrkörper enthält eine Menge von Lehrern.

- Verwaltung des Lehrerbestandes
  Der Bestand an Lehrer kann sich ändern. Der Lehrkörper ist ein Behälter, der Lehrer verwaltet. Er kann neue Lehrer aufnehmen und es können Lehrer aus ihm entfernt werden. Er kann Lehrer herausgeben und nach ihnen suchen.

[18] Siehe dazu den Abschnitt über den wöchentlich wiederkehrenden Zeitraum

11.7.6 Materialvision Pausenliste

An einer Schule gibt es neben einer Menge von Unterrichtsstunden eine Menge von Pausen. Diese Pausen liegen zwischen den Unterrichtsstunden. Die Pausenliste nimmt alle Pausen der Schule auf.

- Verwaltung der Pausen
  Jede Schule hat andere Pausen. Es kann eine unterschiedliche Anzahl von Pausen geben. Die Pausenliste wird verändert. Es können Pausen hinzugefügt sowie entnommen werden. Die Pausenliste kann Pausen herausgeben und nach ihnen suchen.

11.7.7 Materialvision Pausenplan

An einer Schule gibt es eine Menge von Pausen und eine Menge von Lehrern, sowie eine Menge von Aufsichtsorten. Eine Pausenaufsicht kommt zustande, indem ein Aufsichtsort eines Elementes der Pausenliste mit einem Element des Lehrkörpers belegt wird. Der Pausenplan enthält eine Pausenliste. Er kann mit einem Lehrkörper verknüpft werden. Sobald es eine solche Verknüpfung gibt, ist es möglich Pausenaufsichten einzuteilen. Der Pausenplan benutzt die Materialien Lehrkörper und Pausenliste. Es handelt sich somit um einen fachlichen Behälter, der den Lehrkörper benutzt und die Pausenliste verändert.

- Durchführung von Pausenbelegungen
  Der Pausenplan führt die notwendigen Operationen durch um Pausenaufsichten zu erstellen. Dafür benötigt er ein hohes Maß an fachlichem Wissen. Er muß die Pausenbelegung durchführen, Konfliktsituationen feststellen sowie erkennen, ob ein Pausenplan vollständig belegt ist.
- Konflikterkennung
  Der Pausenplanersteller möchte wissen, wo und wie viele Konflikte bei der Planung der Pausenaufsichten aufgetreten sind. Der Pausenplan kann diese Konflikte für alle Lehrer, oder auch nur für einzelne Lehrer, sowie für Pausen ermitteln.
- Ermittlung statistischer Informationen
  Für jeden Lehrer resultiert aus seiner Stelle sowie der Gesamtanzahl der Pausen eine Anzahl von Aufsichtspflichten. Die Ermittlung der Aufsichtspflichten, sowie der Pausenaufsichten des Lehrers gehört zu den Verantwortlichkeiten des Pausenplanes.

11.7.8 Materialvision Wöchentlich wiederkehrender Zeitraum

Ein wöchentlicher wiederkehrender Zeitraum bezeichnet einen Zeitspanne zwischen zwei definierten Zeitpunkten, die sich jede Woche wiederholt. Sie kann als Vergegenständlichung der Aussage : „Immer Montags von 17°° bis 18°°Uhr“ verstanden werden. Ein wöchentlicher Zeitraum kann sich über mehrere Tage erstrekken.

- Veränderung des Zeitraumes
  Ein wöchentlich wiederkehrender Zeitraum kann verändert werden.
- Feststellung von Überschneidungen
  Ein wöchentlich wiederkehrender Zeitraum kann feststellen, ob er sich mit einem anderen überschneidet.

11.8 Glossar des Pausenplanersystems

Aufsichtsort Ein Ort an einer Schule, an dem sich in den *Pausen* Schüler aufhalten und dort demzufolge Aufsicht geführt werden muß.

Aufsichtsortebearbeiter Eine Werkzeug, mit dessen Hilfe die Benutzer des *Pausenplanersystems* die *Aufsichtsorte* an einer Schule definieren.

Ausschlußzeitraum Ein *Ausschlußzeitraum* ist eine Zeitspanne, in welcher ein *Lehrer* nicht zur *Pausenaufsicht* eingeteilt werden sollte.

Benutzer Der Benutzer ist die Person, die mit dem späteren System arbeiten wird. Innerhalb des *Pausenplanersystems* gibt es mehrere Arten von Benutzern. Sie verkörpern dabei unterschiedliche funktionelle Rollen. Der *Pausenplanersteller* oder *Sekretariatsmitarbeiter* sind Benutzer.

Lehrer Der *Lehrer* an einer Schule. Er ist im Kontext des *Pausenplanersystems* eine Person, die *Pausenaufsichten* führen muß, also ein *Pausenbeaufsichtiger*.

Lehrerkarte Eine Karte, auf der die für die Pausenplanung relevanten Informationen notiert sind. Diese Informationen sind im einzelnen der Name des *Lehrers*, die Stelle, die Anzahl der Aufsichtspflichten und die Ausschlußzeiträume des *Lehrers*. Die Anzahl der *Lehrerkarten* zu einem *Lehrer* entspricht der Anzahl der Aufsichtspflichten, denen er im Rahmen seiner Stelle nachzukommen verpflichtet ist. (siehe dazu Szenario Pausenaufsichtsplan erstellen)

Lehrkörper Der *Lehrkörper* ist anwendungsfachlich die Menge der *Lehrer*, die an einer Schule beschäftigt sind. Systemtechnisch bezeichnet der *Lehrkörper* einen Behälter, der die *Lehrer* der Schule enthält und verwaltet.

Lehrkörperbearbeiter Werkzeug um einen *Lehrkörper* zu pflegen

Notiz Eine *Notiz* ist ein Kommunikationsmittel. Sie dient der Mitteilung einer Information an einen oder mehrere Personen. Sie kann konkret an eine Person gerichtet sein, oder ungerichtet sein. Ungerichtet bedeutet, daß sie alle Personen betrifft, die zufällig oder bewußt mit dieser *Notiz* konfrontiert werden. Die Adresse ist üblicherweise auf einer *Notiz* vermerkt. Außerdem enthält sie das Datum der Erstellung, sowie ihren Autor.

Pause Meyers Taschenlexikon beschreibt die *Pause* als „Unterbrechung einer (körperlichen oder geistigen) Tätigkeit, um Ermüdung und damit Leistungsabfall zu vermeiden“. Im Kontext des *Pausenplanersystems* wird der Begriff der *Pause* jedoch in einer spezialisierteren Form verwendet. Er bezeichnet konkreter die *Pausen*, die es an einer Schule gibt und Unterrichtsstunden unterbrechen.

Pausenaufsicht Die *Pausenaufsicht* bezeichnet eine Menge von *Pausenbeaufsichtigern*, mit denen eine *Pause* belegt ist, um sie zu beaufsichtigen Eine *Pausenaufsicht* bezieht sich immer auf eine *Pause*. Der Begriff meint einerseits die Tätigkeit eine *Pause* zu beaufsichtigen (eine Person führt die *Pausenaufsicht*). Andererseits meint der Begriff ebenso die funktionelle Rolle und die damit verbundenen Verantwortlichkeiten (eine Person ist die *Pausenaufsicht*).

Pausenaufsichtsort Ein Ort einer zu beaufsichtigenden *Pause*. Dieser eher technisch definierte Begriff orientiert sich einerseits an dem Ort, an dem Aufsicht zu führen ist, andererseits aber auch an dem Zeitpunkt einer *Pause* und definiert somit beide Eigenschaften einer.

Pausenaufsichtsplan siehe *Pausenplan*

Pausenbeaufsichtiger	Eine funktionelle Rolle an einer Schule. Die Person, die mit der Beaufsichtigung einer *Pause* an einer Schule betraut ist. Es gibt *Pausenbeaufsichtiger* mit unterschiedlicher Qualifikation (siehe dazu das Szenario Pausenaufsichtsplan erstellen). Es gibt *Lehrer*, die verantwortlich *Pausen* beaufsichtigen dürfen, andere *Lehrer*, die dieses nicht verantwortlich tun dürfen und Zivildienstleistende, die ebenso der Qualifikation eines verantwortlichen *Pausenbeaufsichtigers* nicht entsprechen.
Pausenlisten-Bearbeiter	Werkzeug um die *Pausen* des *Pausenplans* zu pflegen
Pausenplan	Der *Pausenplan* ist die Verknüpfung eines *Lehrkörpers* mit einer Menge von *Pausen*. Gemeint ist damit sowohl der anwendungsfachlich vorliegenden Papierplan, auf dem der *Pausenplanersteller Lehrerkarten* ablegt (siehe Szenario Pausenaufsichtsplan erstellen), als auch das systemtechnische Pendant in Form des Materials(siehe Materialvision Pausenplan).
Pausenplaner	Softwarewerkzeug um *Pausenaufsichten* einzuteilen Es unterstützt den *Pausenplanersteller* bei der Erfüllung seiner Aufgabe, *Pausen* mit *Lehrern* zu belegen.
Pausenplanersystem	Das zu erstellende Gesamtsystem.
Pausenplanersteller	Eine funktionelle Rolle, für die Person, die an einer Schule die Aufgabe hat, die Pausenplanung vorzunehmen. In der Regel wird ein *Lehrer* mit dieser Aufgabe betraut.
Sekretariatsmitarbeiter	Eine Person, die im Sekretariat der Schule beschäftigt ist. Sie ist auch ein Benutzer des Pausenplanersystems, da sie mit der Pflege des *Lehrkörpers* betraut ist.

12 Anhang MFC Rahmenwerk

12.1 Schnittstelle CView

Beobachtermuster	
OnUpdate	Wird gerufen, wenn das Dokument den View über seine Zustandsänderung benachrichtigt.
OnInitialUpdate	Wird gerufen, wenn der View innerhalb der Initialisierungsphase an das Dokument gekoppelt wird.
GetDocument	Gibt das Dokument, welches im View dargestellt werden soll, zurück.
Graphische Darstellung	
OnActivateView	Wird gerufen, wenn der View aktiviert wird.
OnActivateFrame	Wird gerufen, wenn das Rahmenfenster (Frame Window), welches den View umschließt, aktiviert oder deaktiviert wird.
OnBeginPrinting	Wird gerufen, wenn ein Druckauftrag beginnt. Der Entwickler kann hier Graphics Device Interface[19] (GDI) Ressourcen anfordern.
OnEndPrinting	Wird gerufen, wenn ein Druckauftrag endet. Der Entwickler kann hier GDI Ressourcen freigeben.
OnEndPrintPreview	Wird gerufen, wenn die Seitenansicht eines Ausdrucks beendet ist.
OnPrepareDC	Wird gerufen, bevor die OnDraw oder OnPrint Methode ausgeführt wird.
OnPreparePrinting	Wird gerufen, bevor die Seitenansicht oder der Ausdruck eines Dokuments beginnt. Der Entwickler kann hier die Drucken Dialogbox initialisieren
OnPrint	Wird gerufen, wenn eine Seite eines Dokuments gedruckt oder in der Seitenansicht betrachtet werden soll.
OnDraw	Wird gerufen, wenn die Bildschirm- oder Druckdarstellung eines Dokuments erstellt werden soll.
DoPreparePrinting	Öffnet die Drucken Dialogbox. Wird verwendet, wenn der Entwickler die OnPreparePrinting Methoden überschreibt.
Drag&Drop	
OnDragEnter	Wird gerufen, wenn der Benutzer innerhalb einer Drag&Drop Operation den Bereich eines View betritt.
OnDragLeave	Wird gerufen, wenn der Benutzer innerhalb einer Drag&Drop Operation den Bereich eines View verläßt.
OnDragOver	Wird gerufen, wenn der Benutzer den Mauszeiger innerhalb einer Drag&Drop Operation über den Bereich eines View bewegt (Dragging).
OnDrop	Wird gerufen, wenn der Benutzer eine Drag&Drop Operation im Bereich eines View beendet (Dropping).
OnDragScroll	Wird gerufen, wenn der Benutzer den Mauszeiger bei einer Drag&Drop Operation in den Scroll Bereich eines Fensters bewegt.

[19] Das GDI ist die Schnittstelle zu einem virtuellen Gerätetreiber für grafische Darstellungen. Das Windows Betriebssystem abstrahiert so von der direkten Ausgabe auf Bildschirm oder Drucker.

12.2 Schnittstelle CDocument

Beobachtermuster	
AddView RemoveView GetFirstViewPosition GetNextView	An und Abmelden von Views beim Document, sowie Methoden, die das Iterieren durch die Liste der angemeldeten Views ermöglichen.
UpdateAllViews	Benachrichtigung der Zustandsänderung an alle angemeldeten Views.
OnChangedViewList	Wird gerufen, wenn sich ein View beim Dokument an- oder abmeldet.
Modifikationsflag und Dokumenttitel	
IsModified SetModifiedFlag	Verändern und Sondieren des Modifikationsflags. Wurde das Dokument modifiziert, so fragt das Rahmenwerk bei Beendigung des Werkzeugs oder Öffnen eines neuen Dokuments, ob das alte Dokument gesichert werden soll.
GetTitle SetTitle	Verändern und Sondieren des Dokumenttitels. Dieser erscheint automatisch als Überschrift in der Fensterleiste.
GetDocTemplate	Liefert das Document-Template welches den Typ des Dokuments beschreibt.
CanCloseFrame	Wird gerufen, bevor ein Frame geschlossen werden soll, dessen View das Dokument darstellt. Der Rückgabewert bestimmt, ob das Frame geschlossen werden kann.
DeleteContents	Wird gerufen, wenn das Rahmenwerk den Inhalt eines Dokuments löschen will.
Dateizugriff	
OnCloseDocument	Wird gerufen, wenn das Dokument geschlossen werden soll.
OnNewDocument	Wird gerufen, wenn ein neues Dokument erstellt werden soll.
OnOpenDocument	Wird gerufen, wenn ein Dokument geöffnet werden soll.
OnSaveDocument	Wird gerufen, wenn ein Dokument gespeichert werden soll.
ReportSaveLoadException	Wird gerufen, wenn beim Öffnen oder Schließen eines Dokuments eine Ausnahmebehandlung aufgetreten ist.
GetFile	Ermöglicht den Zugriff auf die Datei des Dokuments.
ReleaseFile	Macht die Datei des Dokuments auch anderen Applikationen zugänglich.
GetPathName SetPathName	Verändern und Sondieren des Dateipfades.
SaveModified	Fragt den Benutzer ob ein modifiziertes Dokument vor dem Schließen gespeichert werden soll.

12.3 Die Controlklassen des MFC Rahmenwerks

Statische Darstellung	
Cstatic	Statische Controls werden verwendet, um andere Controls zu Beschriften oder zu Gruppieren. Zusätzlich zu der Text- oder Kastenansicht können auch Abbildungen von statischen Controls dargestellt werden.
Text Ein- und Ausgabe	
CEdit	Ein Feld für die Ein- und Ausgabe von Text.
CRichEditCtrl	Wie CEdit. Unterstützt zusätzlich Text- und Absatzformatierung sowie Einbettung von OLE/ActiveX Objekten.
Skalare Ein- und Ausgabe	
CSliderCtrl	Der Control besitzt einen Schieber, den der Benutzer bewegt, um einen bestimmten Wert einzustellen.
CSpinButtonCtrl	Zwei Knöpfe mit Pfeilsymbolen, die der Benutzer verwendet, um einen Wert zu erhöhen oder zu erniedrigen.
CProgressCtrl	Ein Rechteck wird von links nach rechts gefüllt, um den Fortgang einer längeren Operation anzuzeigen.
CScrollBar	Mit dem Schieber kann der Benutzer die Position innerhalb eines Bereichs festlegen.
Knöpfe	
CButton	Diese Klasse stellt dem Entwickler eine Schnittstelle zu „Pushbuttons“, „Check Boxes“ und „Radio Buttons“ zur Verfügung. Bei einem Pushbutton handelt es sich um einen normale Knopf. Eine Check Box ist ein beschriftetes Feld, welches der Benutzer mit einem Hacken versehen kann. Ein Radio Button sieht einer Check Box recht ähnlich, im Unterschied zu ihr kann innerhalb einer Gruppe von Radio Buttons immer nur einer selektiert sein.
CBitmapButton	Statt einer Beschriftung ziert diesen Knopf eine Abbildung.
Listen	
CListBox	Eine Liste von Textelementen, die der Benutzer Betrachten und Selektieren kann.
CDragListBox	Wie CListBox. Zusätzlich kann der Benutzer die Elemente der Liste durch Drag&Drop Operationen anordnen.
CComboBox	Kombination aus Edit und List Control.
CCheckListBox	Wie CListBox. Der Benutzer kann zusätzlich die Elemente mit einem Hacken versehen.
CListCtrl	Stellt eine Elementmenge mit Name und Icon dar. Siehe linke Seite des Windows-Explorer.
CTreeCtrl	Stellt eine hierarchische Anordnung von Elementen mit Name und Icon dar. Siehe rechte Seite des Windows-Explorer.
Status- und Symbolleisten	
CToolBarCtrl	Stellt die Funktionalität einer Windows Symbolleiste zur Verfügung. Das MFC Klassengerüst verwendet die CToolBar Klasse.
CStatusBarCtrl	Stellt die Funktionalität einer Windows Statusleiste zur Verfügung. Das MFC Klassengerüst verwendet die CStatusBar Klasse.
Andere Controls	
CAnimateCtrl	Kann dem Benutzer eine einfache Animation vorspielen.
CToolTipCtrl	Ein kleines einzeiliges pop-up Fenster, welches dem Benutzer den Gebrauch eines Werkzeugs erklären kann.
CHeaderCtrl	Stellt Titel oder Beschreibungen über Spalten dar.
CTabCtrl	Der Benutzer kann Karteireiter, wie in einem Karteikasten auswählen.
CHotKeyCtrl	Ermöglicht es dem Benutzer eine Tastenkombination zu definieren, mit deren Hilfe er eine Operation prompt ausführen kann.
COleControl	Diese Klasse stellt dem Entwickler eine Basis für die Implementation eigener OLE/ActiveX Komponenten zur Verfügung.