Diplomarbeit

zur Erlangung des akademischen Grades
Diplom-Wirtschaftsinformatiker (FH)

Entwicklung eines Frameworks für den standardisierten Zugriff auf Datenobjekte des
Stundenplansystem S-PLUS und Anwendung auf die Prüfungsverwaltung

Eingereicht von	Michael Schäfer
Studiengang	Wirtschaftsinformatik
Seminargruppe	I1/03
Matrikelnummer	02 277 0367
Betreuer	Herr Prof. Dr. rer. pol. Christian Müller Frau Prof. Dr. rer. nat. Ulrike Tippe

Inhaltverzeichnis

Abbildungsverzeichnis	IV
Tabellenverzeichnis	VI
Anlagenverzeichnis	VII
1 Einleitung	1
1.1 S-PLUS	2
1.2 Webpräsentationskomponente	6
1.3 Idee und Zielsetzung	7
2 Technische Grundlagen	10
2.1 Das .NET-Framework	10
2.1.1 Architektur	11
2.1.2 Anwendungen	12
2.1.3 Konzept	14
2.2 C#	15
2.2.1 Typen	16
2.2.2 Glossar	18
2.3 Component Object Model	19
2.3.1 Bereitstellung	19
2.3.2 Interopabilität	20
2.3.3 IDL	21
2.3.4 Objektorientierung	21
2.3.5 Identifikation	22
2.3.6 Automatisierung	23
2.3.7 Persistenz	24
2.3.8 Running Object Table	24
2.3.9 Interopabilität zwischen .NET und COM	24
3 Planung der Software	25
3.1 Framework	25
3.2 Entwurfsmuster	26
3.3 S-PLUS Automationsserver	27
3.4 Konvertieren der Typbibliothek	30

4	Entwicklung der Software	34
4.1	Framework	34
4.1.1	Einbinden der COM-Komponente	35
4.1.2	Objektimplementation	38
4.1.3	Objekte	43
4.1.4	Listen	49
4.1.5	Ereignisse	55
4.1.6	Installation	57
4.1.7	Anwendungen	58
4.2	Webpräsentationskomponente	66
4.2.1	Idee	66
4.2.2	Umsetzung	70
4.2.3	Anpassung	72
4.3	Prüfungsplaner	74
4.3.1	Konzept und Anforderungen	74
4.3.2	Konfiguration	77
4.3.3	Verwaltung	78
4.3.4	Prüfung löschen	83
4.3.5	Prüfung bearbeiten	83
4.3.6	Prüfung anlegen und planen	84
5	Fazit	93
	Quellenverzeichnis	95
	Anhang	99
	Selbständigkeitserklärung	102

Abbildungsverzeichnis

Abbildung 1	Entwicklung der Studierendenzahl an der TFH Wildau	1
Abbildung 2	Darstellung von Daten in S-PLUS	2
Abbildung 3	Einsatz als Netzwerkversion von S-PLUS	3
Abbildung 4	Objektmodell von S-PLUS	4
Abbildung 5	Modell der Webpräsentationskomponente	6
Abbildung 6	Prinzip des Frameworks	9
Abbildung 7	Grobarchitektur des .NET-Frameworks	11
Abbildung 8	Anwendungen für das .NET-Framework	12
Abbildung 9	Typenhierarchie in C#	16
Abbildung 10	Client-Server Kommunikation in COM	19
Abbildung 11	Zugriff auf COM-Objekt in einem lokalen Server	20
Abbildung 12	Aggregation	22
Abbildung 13	Registrierung einer COM-Klasse	22
Abbildung 14	Runtime Callable Wrapper	24
Abbildung 15	S-PLUS Automationsserver für Windows	27
Abbildung 16	Eintragen der ProgID	28
Abbildung 17	Eintrag des S-PLUS Images in der Systemregistratur	28
Abbildung 18	S-PLUS Typbibliothek in ITypeLib Viewer	30
Abbildung 19	Objektdiagramm Objektadapter	38
Abbildung 20	Basisklassenhierarchie im Framework	40
Abbildung 21	Musterstruktur der Fabrikmethode	45
Abbildung 22	Musterstruktur der Schablonenmethode	45
Abbildung 23	Verwalten von Standorten in S-PLUS	50
Abbildung 24	Installiertes Framework	57
Abbildung 25	Benutzeroberfläche ConfigFileBuilder	58
Abbildung 26	Formular zum Erstellen einer Option	60
Abbildung 27	Angelegte Konfigurationsdatei	61
Abbildung 28	Administrationsoberfläche des Frameworks	62
Abbildung 29	Einstellen der ProgID	62
Abbildung 30	Registrierte Konfigurationsdatei	63
Abbildung 31	Startseite für Stundenpläne	66
Abbildung 32	Aufgelistete Stundenpläne für Studenten	67
Abbildung 33	Alle Veranstaltungen für BB-05	68
Abbildung 34	Arbeitsablauf zur Stundenplanerstellung	69
Abbildung 35	Erstellte Verzeichnisstruktur der Webpräsentationskomponente	70
Abbildung 36	Registrierte Webpräsentationskomponente	72

Abbildung 37	Struktur für Studiengänge in S-PLUS	75
Abbildung 38	Konfigurationseinstellungen des Prüfungsplaners	77
Abbildung 39	Daten laden für den Prüfungsplaner	78
Abbildung 40	Hauptfenster der Prüfungsverwaltung	79
Abbildung 41	Filtern und Suchen von Prüfungen	80
Abbildung 42	Ausgabe des Suchergebnisses	81
Abbildung 43	Buttons zur Datenbanksteuerung	82
Abbildung 44	Aktivierte Datenbankkontrolle	82
Abbildung 45	Buttons zum Anlegen, Bearbeiten und Löschen	83
Abbildung 46	Prüfung anlegen	84
Abbildung 47	Auswahl von Dozenten	85
Abbildung 48	Ausgefülltes Datenformular einer Prüfung	86
Abbildung 49	Prüfung planen	87
Abbildung 50	Fehlermeldung bei der Planung einer Prüfung	89
Abbildung 51	Erfolgreich geplante Prüfung	90
Abbildung 52	Datenansicht einer geplanten Prüfung	91
Abbildung 53	Planungsbeschränkungen bearbeiten	92

Tabellenverzeichnis

Tabelle 1	Versionen des .NET-Frameworks	10
Tabelle 2	Dimensionen der Ausprägung von Entwurfsmuster	26
Tabelle 3	Namespaces des Frameworks	34
Tabelle 4	Basisklassen des Frameworks	41
Tabelle 5	Erweiterte Member der Klasse ObjectEditor	48
Tabelle 6	Member die alle Listen der COM-Schnittstelle implementieren	49
Tabelle 7	Member der Klasse ListBuilder	51
Tabelle 8	Erweiterte Member der Klasse ListEditor	54
Tabelle 9	Beispieloptionen einer Konfigurationsdatei	59
Tabelle 10	Methode der Klasse Options	64

Anlagenverzeichnis

Anlage A	Erstellte Adapter im Framework	99
Anlage B	CD	100

1 Einleitung

Die Technische Fachhochschule Wildau bietet Studiengänge in den Fachbereichen Ingenieurwesen/Wirtschaftsingenieurwesen, Betriebswirtschaft/Wirtschaftsinformatik sowie Wirtschaft, Verwaltung und Recht an. Zum Stand des Wintersemesters 2006 / 2007 gab es insgesamt 28 Studiengänge (Webseiten der TFH Wildau, vgl. [TFHWZA]).

Zu den 198 Mitarbeitern an der Technischen Fachhochschule Wildau gehören Lehrkräfte, Professoren, wissenschaftliche sowie technische Mitarbeiter und Verwaltungspersonal. Seit dem Wintersemester 1990 / 1991 steigt die Anzahl der Studierenden stetig an und zum Wintersemester 2006 / 2007 betrug die Anzahl bereits 3509 (vgl. [TFHWZA]). Diese Entwicklung wird durch Abbildung 1 verdeutlicht.

Abbildung 1: Entwicklung der Studierendenzahl an der TFH Wildau (vgl. [TFHWZA])

Die genannten Fakten verdeutlichen die steigende Komplexität bei der Verwaltung von Daten und dem Planen und Erstellen von Lehrplänen. Mehr Studiengänge und mehr Studenten erfordern mehr Räume, mehr Dozenten und mehr Ausrüstungsgegenstände.

Ausgehend von der Menge der Daten stellt das Planen von Veranstaltungen einen sehr komplexen Vorgang dar. Dabei spielen neben der Planung des Zeitpunkts oder der Größe einer Veranstaltung auch Beschränkungen oder Präferenzen von Teilnehmern einer Veranstaltung eine Rolle. Bei der großen Menge der Veranstaltungen gibt es unzählige Entscheidungsalternativen zur Erstellung des Lehrplans. Das Erstellen einer optimalen Lösung für dieses Entscheidungsproblem ist kaum von Hand zu bewältigen. Aus diesem Grund wird seit 2005 die von der Scientia GmbH (siehe [SCIENTIA]) entwickelte Planungssoftware S-PLUS (siehe [SPLUS]) zum Erstellen von Stundenplänen an der Technischen Fachhochschule Wildau eingesetzt.

1.1 S-PLUS

Die Planungssoftware S-PLUS ist bereits in mehreren Versionen veröffentlicht worden. Im Rahmen der Diplomarbeit wird mit der Version 3.5.1 gearbeitet.

S-PLUS ist eine Windows-Anwendung und bietet mit einer grafischen Benutzeroberfläche Zugriff auf Daten und Funktionen.

Die Aufgaben von S-PLUS kann man in zwei Bereiche unterteilen. Der erste Bereich ist das Verwalten von Daten. Dabei geht es um das Sammeln, Speichern und Verarbeiten von Informationen über Dozenten, Fachbereiche oder Semester. Alle diese Objekte werden über den Menüpunkt "Lehrangebot" in der grafischen Benutzeroberfläche verwaltet. Für jedes Objekt steht ein Formular zur Verfügung welches das Anlegen, Löschen und Bearbeiten dieses Typs ermöglicht. Auf Abbildung 2 wird diese Möglichkeit exemplarisch für Fachbereiche dargestellt.

Abbildung 2: Darstellung von Daten in S-PLUS (Bildschirmkopie)

Auf Abbildung 2 ist zu erkennen, dass die Datensätze auf der linken Seite aufgelistet sind und die entsprechenden Eigenschaften über Tabulatoren (oder Reiter) eines Objektes auf rechten Seite präsentiert werden.

Eine genaue Beschreibung aller in S-PLUS genutzten Objekte und zur Installation der Software ist im Handbuch zu S-PLUS (siehe [HB350]) zu finden.

Die Verwaltung der Daten kann zentral als Einzelplatzversion oder dezentral als Netzwerkversion stattfinden (vgl. [SPLUS]).

Der Einsatz der Software erfolgt an der Technischen Fachhochschule Wildau als Netzwerkversion. Dieses Prinzip wird auf Abbildung 3 dargestellt.

Abbildung 3: Einsatz als Netzwerkversion von S-PLUS (eigene Darstellung)

Die Abbildung 3 verdeutlicht, dass auf einem Server mit Hilfe einer Datenbank die Daten der Technischen Fachhochschule Wildau hinterlegt werden. Die dezentrale Bearbeitung dieser Daten kann durch einen oder mehrere Clients erfolgen. Diesen steht die Möglichkeit zur Verfügung, die entsprechenden Daten aus der Datenbank zu auffrischen oder diese im umgekehrten Fall in die Datenbank zurückzuschreiben.

Auf der Seite eines Clients werden Daten in einem speziellen Dateiformat gespeichert. Dieses Format hat die Endung „img“ und wird im weiteren Verlauf als Image bezeichnet. In einem Image befindet sich neben den Daten der Hochschule auch Software-Code. Dieser Code umfasst die grafische Benutzeroberfläche von S-PLUS sowie Methoden und Funktionalitäten zur Bearbeitung der Daten. Somit kann ein Image als COM-Server (siehe [COM]) fungieren. Die COM-Technologie wird in Abschnitt 2.3 näher beschrieben.

Wichtig im Zusammenhang mit dem Component Object Model ist die Tatsache, dass mit dieser Technologie der Zugriff auf Daten, Methoden und Funktionalitäten des Images von außerhalb möglich ist. Des Weiteren können Daten auch in anderen Dateiformaten gespeichert und ausgelesen werden. Dazu zählt zum Beispiel das Format UEF (Universal Exchange Format). Diese sind jedoch im Zusammenhang mit der Diplomarbeit weniger von Bedeutung und werden nicht weiter betrachtet.

Die durch S-PLUS verwalteten Objekte stehen in einem Zusammenhang. Das Objektmodell wird auf Abbildung 4 gezeigt.

Abbildung 4: Objektmodell von S-PLUS (eigene Darstellung nach Handbuch, vgl. [SAS])

Abbildung 4 beschreibt die notwendigen Objekte und ihr Verhältnis zur Erstellung eines Stundenplans. Ein Punkt der durch Abbildung 4 dargestellt wird ist die Zuweisung von Mitteln (Ressourcen) zu einer Veranstaltung. Zu den Ressourcen zählen Dozenten, Räume oder Studenten-Sets. Es wird ersichtlich, dass ohne eine Zuweisung der benötigten Ressourcen eine Veranstaltung nicht geplant werden kann.

Aus dieser Erkenntnis kann man den zweiten Aufgabenbereich von S-PLUS ableiten. Dabei handelt es sich um das Erstellen eines Stundenplans mit Hilfe der verwalteten Daten. Der Prozess der Planung umfasst dabei verschiedene Schritte. Damit ist aber nicht das Planen einer einzelnen Veranstaltung gemeint, sondern die Erstellung eines gesamten Stundenplans. Der Anwender legt Rahmenbedingungen für den Plan fest. Dabei kann es sich um nachfolgende Punkte handeln:

- § **Beschränkungen** – Während des Planungsprozesses werden alle Ressourcen auf ihre Verfügbarkeit geprüft. Wenn eine Ressource zu dem gewünschten Zeitpunkt einer Veranstaltung nicht zur Verfügung steht kann diese nicht geplant werden. Es besteht aber die Möglichkeit dieses Verhalten zu umgehen und die entsprechenden Ressourcen später zuzuweisen.
- § **Präferenzen** – Für Ressourcen wie beispielsweise Dozenten können Präferenzen festgelegt werden. Das schließt Einstellungen zum bevorzugten Beginn des Arbeitstages oder zum bevorzugten Einsatzzeitraum innerhalb einer Woche ein.

Der Veranstaltungsplaner versucht nun mit den gegebenen Beschränkungen und Präferenzen eine optimale Lösung für den Lehrplan zu entwickeln.

Wenn Veranstaltungen den gesetzten Rahmenbedingungen nicht entsprechen, kann dies in einer Auflistung von Problemen nachvollzogen werden. Dabei kann es sich um Veranstaltungen handeln, die nicht an dem vorgeschlagenen Zeitpunkt in den Plan gesetzt werden können oder eine Ressource nutzen, die zum selben Zeitpunkt schon von einer anderen Veranstaltung genutzt wird.

Insgesamt bietet die Software viele Möglichkeiten Veranstaltungen unter bestimmten Bedingungen zu planen. Eine genauere Beschreibung befindet sich im Handbuch zur Software, welches den Anlagen beiliegt (siehe [HB350]).

Zusammenfassend kann die Aufgabe von S-PLUS als das Erstellen von Lehrplänen und die Verwaltung der dafür benötigten Daten bezeichnet werden. Die Präsentation erfolgt über eine grafische Benutzeroberfläche.

1.2 Webpräsentationskomponente

Die Technische Fachhochschule Wildau stellt Stundenpläne für Studentengruppen, Räume und Dozenten im Internet zur Verfügung. Die Möglichkeiten die S-PLUS bietet reichen allerdings nicht aus, um die Publikation zu realisieren. Aus diesem Grund wurde von Prof. Dr. Christian Müller und Jürgen Lübeck eine Komponente entwickelt, die dies ermöglicht (Webseite der Webpräsentationskomponente, vgl. [WPK]).

Zur Umsetzung des Projektes wird die im Abschnitt 1.1 angedeutete Möglichkeit genutzt, Daten des Images von außerhalb auszulesen. Somit können Informationen über Dozenten, Räume und Studentengruppen in einem gesonderten Softwareprojekt gelesen und verarbeitet werden. Dieses Prinzip wird durch Abbildung 5 verdeutlicht.

Abbildung 5: Modell Webpräsentationskomponente (vgl. [WPK])

Nachdem die entsprechenden Daten ausgelesen wurden, werden die benötigten Informationen in XML-Dateien gespeichert. Die XML-Dateien werden nun auf den Server der Technischen Fachhochschule Wildau geladen. Auf dem Server befinden sich auch die entsprechenden XSLT-Stylesheets, mit denen die Stundenplandaten verknüpft werden und dadurch eine Darstellung der Stundenpläne in Webbrowsern erfolgt (vgl. [WPK]).

1.3 Idee und Zielsetzung

In den beiden vorhergehenden Punkten wurde der Ist-Zustand der Veranstaltungsplanung an der Technischen Fachhochschule Wildau beschrieben.

Die ursprüngliche Idee der Diplomarbeit war die Entwicklung einer Windows-Anwendung, die das Verwalten von Prüfungen an der Technischen Fachhochschule Wildau erlaubt. Die Software soll genau wie die Webpräsentationskomponente den Zugriff auf die von S-PLUS verwalteten Daten mit Hilfe der COM-Technologie erhalten. Das Projekt wird im weiteren Verlauf mit dem Arbeitstitel Prüfungsplaner bezeichnet und soll folgenden Anforderungen genügen:

- § **Prüfungen filtern** – Es sollen nur Prüfungen verwaltet werden. Aus diesem Grund müssen Prüfungen bei der Nutzung der Software aus der Masse von allen Veranstaltungen gefiltert werden.
- § **Prüfungen suchen und auflisten** – Über die grafische Benutzeroberfläche des Prüfungsplaners soll das Suchen von Prüfungen auf möglichst einfache und übersichtliche Weise erfolgen und das entsprechende Suchergebnis in einer Liste dargestellt werden. Des Weiteren sollen über die Einträge der Ergebnisliste Detailinformationen zu einer Prüfung abrufbar sein.
- § **Prüfungen erstellen und bearbeiten** – Mit Hilfe eines Formulars soll das Erstellen beziehungsweise Bearbeiten einer einzelnen Prüfung ermöglicht werden. Dabei müssen die entsprechenden Ressourcen zugewiesen und die Prüfung in den Lehrplan gesetzt werden.
- § **Prüfung planen** – Nachdem eine Prüfung erstellt wurde, soll die Software die Planung der Prüfung über die automatisierten Funktionalitäten des Images von S-PLUS vornehmen. Wenn während des Planungsprozesses Probleme oder Fehler auftreten, sollen diese dem Anwender gemeldet werden und dieser kann entscheiden, ob die Prüfung trotz bestimmter Probleme geplant wird oder ob die Rahmenbedingungen für die Prüfung noch einmal bearbeitet werden müssen.
- § **Image speichern** – Der Anwender soll über den Prüfungsplaner die Möglichkeit erhalten das Image zu speichern.
- § **Datenbank** – Wie bereits erläutert wird S-PLUS an der Technischen Fachhochschule Wildau als Netzwerkversion genutzt. Dementsprechend muss dem Nutzer des Prüfungsplaners eine Funktionalität zur Verfügung stehen, die es erlaubt die Daten des aktuellen Images aufzufrischen beziehungsweise zurückzuschreiben.

Die Auflistung soll einen kurzen Überblick über die gewünschten Funktionalitäten des Prüfungsplaners geben. Tiefergehende Details werden im Kapitel 4.3 dargestellt.

Der Zweck der die Entwicklung des Prüfungsplaners bedingt, ist das Verstecken der Komplexität des Verwaltens und Planens von Prüfungen innerhalb von S-PLUS. Die grafische Benutzeroberfläche des Prüfungsplaners soll im Gegensatz zur Oberfläche von S-PLUS nur die nötigsten Informationen liefern und diese auch in einer vereinfachten Art und Weise präsentieren. Insgesamt soll es also zu einer vereinfachten, schnelleren und effizienteren Verwaltung von Prüfungen durch Mitarbeiter der Technischen Fachhochschule Wildau kommen.

Nachdem die Anforderungen und der Zweck der Softwareentwicklung spezifiziert wurden, standen mir der Quellcode der Webpräsentationskomponente sowie Beschreibungen und Spezifikationen von Objekten, Methoden sowie Funktionalitäten des COM-Servers zur Verfügung. Ausgehend von dieser Basis bestand der erste Schritt darin, den Quellcode genauer zu analysieren. Dabei wurde vor allem ersichtlich, dass die Webpräsentationskomponente viele Funktionalitäten im Zusammenhang mit dem COM-Server implementiert, die auch für den Prüfungsplaner benötigt werden. Dabei handelt es sich zum Beispiel um Berechnungen im Zusammenhang mit Zeitangaben, um das Kapseln von Methoden oder um das Bereitstellen von angepassten Ansichten auf die Datensätze des COM-Servers.

Im zweiten Schritt folgte die Untersuchung der bereitgestellten COM-Komponente. Dabei wurde deutlich, dass die Anzahl der Objekte, Schnittstellen und Methoden sehr hoch ist. Vor allem für Programmierer die unerfahren im Umgang mit der COM-Technologie sind, stellt dies ein großes Hindernis dar. Es erfordert einen hohen Zeitaufwand mit Hilfe von Beschreibungen (Handbuch COM-Server, siehe [SAS]) die ganze Bandbreite der angebotenen Möglichkeiten zu verstehen. Ein weiterer Nachteil besteht darin, dass Methoden oder Eigenschaften eines Objekts nicht immer das erwartete Ergebnis liefern und somit ein gewisser Frustrationsgrad bei der Arbeit mit der COM-Komponente entsteht.

Aus den beschriebenen Erfahrungen im Umgang mit der COM-Komponente und dem schon vorhandenen Know-how der Webpräsentationskomponente entstand die Idee, dass bereits vorhandene Wissen nicht noch einmal neu für den Prüfungsplaner implementieren zu müssen. Auf dieser Grundlage wurde ein Konzept entwickelt, dass die Erfahrungen im Zusammenhang mit S-PLUS in einer Komponente zusammenfasst und diese somit für den Prüfungsplaner und zukünftige Entwicklungen zur Verfügung stehen. Damit ist im Anfangsstadium der Diplomarbeit eine weitere Zielstellung hinzugekommen.

Auf Abbildung 6 wird das Prinzip des Frameworks noch einmal visualisiert.

Abbildung 6: Prinzip des Frameworks (eigene Darstellung)

Abbildung 6 soll zeigen, dass Anwendungen die mit dem COM-Server von S-PLUS zusammenarbeiten dies in Zukunft über eine Zwischenschicht tun. Das Framework verfolgt das Ziel, den COM-Server zu abstrahieren. Neben der Zusammenfassung von Wissen und Erfahrungen soll das Framework eine grundlegend vereinfachte Arbeit mit dem COM-Server ermöglichen. Wie das Kapseln und das Anpassen der Funktionalitäten des COM-Servers durch das Framework erfolgt wird in Kapitel 4.1 beschrieben.

Nach den Betrachtungen zum Framework und seinem Nutzen lassen sich die notwendigen Arbeitsschritte während der Diplomarbeit folgendermaßen gliedern:

1. Entwicklung des Frameworks;
2. Anpassung der Webpräsentationskomponente an das Framework;
3. Ausarbeitung des Prüfungsplaners.

2 Technische Grundlagen

Alle Applikationen die im Rahmen der Diplomarbeit entwickelt wurden, basieren auf einer von Microsoft entwickelten Softwareplattform, dem .NET-Framework. Die Entwicklung fand mit der Programmiersprache C# statt. Das Interagieren mit *S-PLUS* wird durch die COM-Technologie ermöglicht.

Das folgende Kapitel soll eine Einführung in die verwendeten Technologien geben.

2.1 Das .NET-Framework

Das .NET-Framework (offizielle Webseite, siehe [MSDNNET]) ist eine von Microsoft entwickelte Softwareplattform, die den internationalen Standard Common Language Infrastructure (CLI-Standard, siehe [ECMA335]) implementiert. Der offene Standard, der durch die ECMA (siehe [ECMA]) im Juni 2006 in der 4. Version veröffentlicht wurde, spezifiziert Rahmenbedingungen und Regeln für Softwareplattformen, welche die Voraussetzungen für das sprachneutrale und plattformunabhängige Entwickeln bzw. Ausführen von Anwendungen schaffen.

Die Softwareplattform ist im Jahr 2002 erstmals in der Version 1.0 erschienen und im Jahr 2007 wurde die bis hierher letzte Version herausgebracht. Parallel zum Ausbau der Plattform wurde die Entwicklungsumgebung Visual Studio weiterentwickelt. Die Tabelle 1 listet alle bisherigen Versionen des .NET-Frameworks mit den dazugehörigen Versionen der Entwicklungsumgebung auf.

Erscheinungsjahr	Version	Entwicklungsumgebung
2002	1.0	Visual Studio 2002
2003	1.1	Visual Studio 2003
2005	2.0	Visual Studio 2005
2006	3.0	
2007	3.5	Visual Studio 2008

Tabelle 1: Versionen des .NET-Frameworks

Die Entwicklung der Software im Rahmen der Diplomarbeit wurde mit der Version 2.0 und der dazu korrespondierenden Version von Visual Studio „Microsoft Visual Studio 2005“ realisiert.

Die Auslieferung der Version 2.0 des .NET-Frameworks erfolgt ab Windows Vista automatisch. Für ältere Versionen des Betriebssystems Windows besteht die Möglichkeit das .NET-Framework im Nachhinein zu installieren oder im Zusammenhang mit einer Anwendung weiterzugeben.

Auf Grundlage des offenen Standards der CLI gibt es verschiedene Implementationen und Varianten des .NET-Frameworks, die das Ziel verfolgen, die Erstellung und die Ausführung von .NET-Anwendungen auch auf anderen Betriebssystemen zu ermöglichen. Zur Gewährleistung der Plattformunabhängigkeit hat Microsoft eigene Varianten der Plattform für FreeBSD oder Mac OS X zur Verfügung gestellt. Außerdem existieren Projekte von Drittanbietern wie Mono (siehe [MONO]).

2.1.1 Architektur

Das .NET-Framework kann man als Schicht betrachten, die auf Windows aufsetzt und eine Laufzeitumgebung sowie eine objektorientierte Klassenbibliothek hinzufügt (vgl. [MÖS06], S. 4). Dieser Ansatz zur groben Beschreibung der Architektur wird durch Abbildung 7 verdeutlicht.

Abbildung 7: Grobarchitektur des .NET-Frameworks (vgl. [BBMHPW06], S. 3)

Die Common Language Runtime stellt die Laufzeitumgebung zur Verfügung auf deren Basis alle .NET-Anwendungen ausgeführt werden. Sie besitzt eigene Komponenten wie die Automatische Speicherverwaltung (Garbage Collection), verschiedene Sicherheitssysteme, die Verwaltung eines Threadpools oder einen Dateilader. Die Laufzeitumgebung basiert auf einer virtuellen Maschine (siehe [VM]) und besitzt mit der Common Intermediate Language (kurz CIL, in der Literatur findet man auch die Begriffe Microsoft Intermediate Language oder kurz Intermediate Language) einen eigenen Befehlssatz (vgl. [MÖS06], S. 5).

Alle .NET-Anwendungen werden mit Hilfe eines entsprechenden Compilers in diesen Befehlssatz übersetzt.

Unmittelbar bevor eine Anwendung ausgeführt wird, übersetzt der JIT-Compiler (just in time) den CIL-Code in Maschinencode der entsprechenden Plattform. Zusätzlich definiert die CLR mit dem Common Type System (CTS) ein einheitliches Typsystem und mit der Common Language Specification (CLS) Regeln für die Verwendung von Typen innerhalb des .NET-Frameworks (vgl. [MÖS06], S. 5).

Die Klassenbibliothek (englisch: Base Class Library oder Framework Class Library) des .NET-Frameworks stellt einen objektorientierten API-Satz bereit (vgl. [RIC06], S. 18). Sie bietet tausende von Typen an und wird aus Gründen der Übersichtlichkeit in Namespaces unterteilt. Es werden Funktionalitäten für die Erstellung von grafischen Benutzeroberflächen, für Sicherheitsmechanismen oder für andere auf Abbildung 7 dargestellte Bereiche zur Verfügung gestellt.

2.1.2 Anwendungen

Mit der Abbildung 8 wird der Zusammenhang zwischen erstellten Anwendungen, Assemblies und dem .NET-Framework visualisiert. Des Weiteren wird der Umwandlungsprozess von Quellcode zu Maschinencode verdeutlicht.

Abbildung 8: Anwendungen für das .NET-Framework (vgl. [MSDNC#2])

Abbildung 8 stellt auf der obersten Ebene ein Projekt in der Programmiersprache C# dar. Neben den Quellcodedateien können zu einem Projekt Verweise auf Bibliotheken oder andere Anwendungen sowie Ressourcen (Bilder etc.) gehören.

Mit Hilfe eines C#-Compilers wird dieses Projekt in eine .NET-Binärdatei, ein Assembly, umgewandelt. Code der nach diesem Prinzip für das .NET-Framework erstellt wurde, wird als *verwalteter Code* bezeichnet. Ein Assembly stellt die kleinste Komponente im Zusammenhang mit der Entwicklung von .NET-Anwendungen dar. Folgende Bestandteile hat ein Assembly:

- § **CIL-Code** – Der Code der später über das .NET-Framework in Maschinencode umgewandelt und ausgeführt wird.
- § **Metadaten** – Die Metadaten beschreiben alle in einem Assembly enthaltenen Typen (Klassen, Schnittstellen usw.) sowie derer Mitglieder (Methoden, Ereignisse usw.) (vgl. [TRO02], Seite 39).
- § **Manifest** – Beschreibt den internen Aufbau des Assemblys.
- § **Ressourcen** – Ressourcen sind optional und können zum Beispiel Bilder sein die mit dem Projekt ausgeliefert werden.

Durch Metadaten und Manifest ist ein Assembly vollständig selbstbeschreibend.

Des Weiteren enthält ein Assembly eine Versionsnummer. Diese mehrstufige Nummer garantiert, dass andere Komponenten immer das Assembly mit der entsprechenden Versionsnummer laden und somit eine Existenz mehrerer Bibliotheken (DLLs) mit unterschiedlichen Versionsnummern auf einem Computer möglich ist. Frühere in diesem Zusammenhang bekannte Probleme, wie das gegenseitige Überschreiben von Komponenten, beispielsweise bei der Installation neuer Software, sind damit behoben (vgl. [MÖS06], Seite 6 f.).

Ein Assembly stellt die kleinste Komponente der Softwareentwicklung im Zusammenhang mit dem .NET-Framework dar. Die Speicherung und Weitergabe erfolgt als ausführbare EXE-Datei oder als Bibliothek in Form einer DLL-Datei (dynamic link library) (vgl. MöS06], Seite 6 f.).

2.1.3 Konzept

In den letzten Abschnitten wurde die Architektur des .NET-Frameworks erläutert und beschrieben wie eine Anwendung für die Plattform erstellt und gespeichert wird. Auf dieser Grundlage kann man einige konzeptionelle Gesichtspunkte des .NET-Frameworks zusammenfassen:

- § **Sprachintegrität** – Das .NET-Framework wurde von Anfang an darauf ausgerichtet sprachneutral zu sein. Voraussetzung für den Einsatz einer Programmiersprache zum Erstellen von .NET-Anwendungen ist ein Compiler der Code für das .NET-Framework erzeugen kann. Dieser Code muss dem durch die CLR definierten Programmiermodell (CTS) entsprechen. Es existieren Compiler für viele verschiedene Programmiersprachen, Beispiele sind C#, Visual Basic, J#, Fortran, COBOL, Perl oder Pascal.
- § **Sprachinteropabilität** – CIL-Code der durch einen .NET-fähigen Compiler erstellt wurde und der dem CTS entspricht ist für andere Programmiersprachen zugänglich. Es besteht also zum Beispiel die Möglichkeit eine in C# erstellte Bibliothek in einer anderen Sprache wie Visual Basic vollständig zu nutzen.
- § **Abstraktion des Anwendungsszenarios** – Unter Zuhilfenahme eines einzigen Programmiermodells (CLR) und der Klassenbibliothek des .NET-Frameworks kann ein Entwickler verschiedenste Anwendungsszenarios abdecken. Es können Konsolenanwendungen, Web-Anwendungen oder eine grafische Windows-Anwendungen als Anwendungsszenario in Frage kommen (vgl. [RIC06], S. 18). Die Abstraktion des Szenarios verfolgt weiterhin das Ziel ältere Modelle von Microsoft wie die WIN-32 Programmierung oder die COM-Programmierung abzulösen. Die genannten und andere Modelle sind nur auf ein Anwendungsszenario ausgerichtet und damit weit weniger flexibel als das .NET-Framework.
- § **Komponentenorientierung** – Die Bereitstellung von Komponenten erfolgt im Gegensatz zu älteren Modellen vereinfacht. Es müssen keine Registrierungseinträge vorgenommen werden. Die Wiederverwendung von Code wird durch das selbstbeschreibende Prinzip eines Assemblys und der Sprachinteropabilität stark vereinfacht. Ein Assembly kann ohne Systemregistrierungseinträge oder ähnlichem separat ausgeliefert werden.
- § **Plattformunabhängigkeit** – Wie beschrieben erzeugt ein Compiler einer .NET-Sprache keinen Maschinencode sondern plattformunabhängigen CIL-Code. Die Übersetzung in Maschinencode erfolgt erst zur Laufzeit durch die Common Language Runtime. Somit kann eine .NET-Anwendung auf jedem Computer genutzt werden auf dem eine ECMA-kompatible Version der Common Language Runtime und der Klassenbibliothek installiert ist (vgl. [RIC06], Seite 20).

2.2 C#

Im vorhergehenden Kapitel wurde dargestellt, dass die Entwicklung von Applikationen für das .NET-Framework in verschiedenen Programmiersprachen erfolgen kann. Mit C# (offizielle Webseite, siehe [MSDNC#]) hat Microsoft eine Programmiersprache entworfen, die speziell auf die Softwareplattform ausgerichtet ist und alle Möglichkeiten des .NET-Frameworks ausnutzen soll.

Die Programmiersprache ist im Jahr 2001 erschienen und wurde im Jahr 2008 in der bis zum jetzigen Zeitpunkt letzten Version 3.0 veröffentlicht. Parallel zur Weiterentwicklung erfolgt die Standardisierung von C# durch die Norm ECMA-334 (siehe [ECMA334]) beziehungsweise dem ISO-Standard 23270 (siehe [ISO23270]). Alle weiteren Aussagen über C# beziehen sich auf die im Rahmen der Diplomarbeit verwendete Version 2.0 der Programmiersprache.

C# ist als Konkurrenzprodukt zu Java konzipiert und kombiniert Eigenschaften von Java, C++ und Visual Basic (vgl. [MÖS06], Seite 1).

In C# wurden die bekannten Konzepte der objektorientierten Programmierung umgesetzt. Dazu zählt zum Beispiel die Vererbung, das Kapseln von Werten, Polymorphismus, abstrakte und statische Klassen sowie das Festlegen der Sichtbarkeit von Typen und Eigenschaften.

Weitere Merkmale von C# sind das Überladen von Operatoren, die attributbasierte sowie die schnittstellenbasierte Programmierung und die nicht notwendige Benutzung von Zeigern. Des Weiteren zählen die automatische Speicherverwaltung und formale Syntaxkonstrukte für Strukturen, Aufzählungen oder Klasseneigenschaften zu den Eigenschaften von C# (vgl. [TRO02], Seite 34).

C# zählt zu den stark typisierten Programmiersprachen. Damit wird garantiert, dass alle Variablen und Objekte zu jeder Zeit durch einen Typ deklariert sind und dass diese Bindung während der Lebenszeit des Objektes bestehen bleibt. Im Zusammenhang mit C# wird auch von einer typsicheren Sprache gesprochen. Typsicherheit bedeutet, dass Fehler, die beim Deklarieren von Variablen oder in Anweisungen entstehen, bereits vor der Laufzeit einer Anwendung durch den Compiler erkannt werden. Dadurch lassen sich viele Programmierfehler von vornherein vermeiden (vgl. [MÖS06], Seite 2).

2.2.1 Typen

Die Programmiersprache C# verfügt über eine Vielzahl von definierten Datentypen. Dabei kann es sich um einfache Typen wie Zeichenfolgen oder ganzen Zahlen handeln, aber auch um benutzerdefinierbare Typen wie Klassen oder Schnittstellen. Grundsätzlich unterscheidet man zwischen drei verschiedenen Arten:

- § **Werttypen** (value-types) – Dabei handelt es sich um einfache Typen wie bool, int oder float sowie Aufzählungstypen (Enumerationen) und Strukturen (Structs). Variablen dieser Typen werden am Methodenkeller oder im enthaltenden Objekt gespeichert und enthalten direkt einen Wert. Bei der Zuweisung wird dieser Wert kopiert (vgl. [MÖS06], Seite 18).
- § **Referenztypen** (reference-types) – Referenztypen umfassen Klassen, Delegates, Felder (Arrays) und Schnittstellen (Interfaces). Im Gegensatz zu Werttypen enthält eine Variable nicht direkt einen Wert sondern eine Referenz auf ein Objekt. Die Zuweisung und Speicherung der Variable erfolgt auf einem verwalteten Heap (vgl. [MÖS06], Seite 18).
- § **Zeigertypen** (pointer-types) – Zeiger verweisen auf Adressen im Speicher. Die Verwendung eines Zeigertyps ist in C# nur in einem als unsicher (unsafe) gekennzeichneten Block möglich. Zeigertypen sollten nur eingeschränkt genutzt werden.

Abbildung 9 stellt die eben beschriebenen Datentypen in einer Hierarchie dar. Dabei werden die in C# seltener genutzten Zeigertypen außen vor gelassen.

Abbildung 9: Typenhierarchie in C# (vgl. [MÖS06], Seite 17)

Ein ausführliche Beschreibung über die Unterschiede und Gemeinsamkeiten von Verweistypen, Referenztypen und Primitiven Typen ist in den Fachbüchern [RIC06] (Seite 142 ff.) und [MÖS06] (Seite 17 ff.) zu finden.

Alle Typen die in C# vorgeben sind oder durch einen Entwickler definiert wurden sind direkt oder indirekt vom Typ `Object` abgeleitet. Somit werden standardmäßig folgende öffentliche Methoden von jedem Typ implementiert:

- § **Equals** – Liefert eine Aussage darüber ob zwei Objekte denselben Wert haben.
- § **GetHashCode** – Berechnet einen Hashwert für den Wert des Objektes. Dieser Wert kann zum Beispiel im Zusammenhang mit Hashtabellen verwendet werden.
- § **ToString** – Wenn diese Methode nicht überschrieben wurde, gibt sie den Namen des Typs des aktuellen Objektes zurück.
- § **GetType** – Liefert eine Instanz eines Type-Objekts, das Informationen über den Typ des Objektes enthält.

Des Weiteren werden durch den Typ `Object` noch zwei geschützte Methoden zur Verfügung gestellt. Dabei handelt es sich um die Methode *MemberwiseClone*, die das aktuelle Objekt kopiert und eine neue Instanz anlegt, sowie um die Methode *Finalize*, welche aufgerufen wird bevor der Garbage Collector das Objekt löscht (vgl. [RIC06], Seite 121 f.).

Im weiteren Verlauf der Diplomarbeit spielt die Möglichkeit Typen zu konvertieren eine entscheidende Rolle. Die Typkonvertierung (type casting) kann in zwei Richtungen erfolgen. Zum einen besteht die Möglichkeit einen Typ in dessen Basistyp umzuwandeln. Wenn also alle Typen direkt oder indirekt den Typ `Object` als Basistypen haben, ist folgende Typumwandlung immer gültig:

```
Object obj = new Klasse();
```

Der umgekehrte Weg ist die Umwandlung eines Objektes in einen abgeleiteten Typ:

```
Klasse obj2 = (Klasse) obj;
```

Es wird deutlich, dass die Anweisung im zweiten Fall explizit den gewünschten Typ beinhaltet. Für diesen Vorgang wird in C# auch der Operator `as` angeboten. Durch den Operator `is` besteht außerdem die Möglichkeit zu prüfen ob eine Umwandlung möglich ist. Folgender Quellcode soll die Funktionsweise der Operatoren zeigen:

```
if (obj is MyClass) { MyClass obj2 = obj as MyClass; }
```


2.2.2 Glossar

In diesem Abschnitt sollen einige Sprachkonstrukte von C# dargestellt werden, die im weiteren Verlauf der Diplomarbeit eine Rolle spielen. Dies soll dem besseren Verständnis in späteren Kapiteln dienen. Es wird davon ausgegangen, dass allgemeine Begrifflichkeiten im Zusammenhang mit objektorientierter Programmierung, wie Klasse, Schnittstelle oder Vererbung, bekannt sind.

- § **Eigenschaft** – Properties oder Eigenschaften stellen Felder eines Objektes dar. Das Nutzen von Eigenschaften erlaubt das Abrufen und das Setzen von Werten eines Objektes und dient häufig der Kapselung von privaten Feldern.
- § **Delegate** – Ein Delegate ist ein Objekt in dem Methoden gespeichert werden können. Die Deklaration erfolgt mit dem Schlüsselwort `delegate` und einer Methodensignatur. Daraufhin können Methoden mit dieser Signatur in einer Variablen dieses Delegate-Typen gespeichert werden (vgl. [MÖS06], Seite 105).
- § **Event** – Ereignisse werden durch das Schlüsselwort `event` gekennzeichnet und bieten eine Vereinfachung der Verwendung von Delegates. Sie werden als Felder einer Klasse deklariert. Mit Hilfe eines Ereignisses ist es Objekten möglich sich gegenseitig zurückzurufen. Die häufigste Anwendung von Ereignissen findet man bei Windows-Anwendungen mit einer grafischen Benutzeroberfläche (vgl. [TRO02]).
- § **Generika** – Generische Typen bieten die Möglichkeit einer Parametrisierung ihrerseits mit anderen Typen. Dieses Verfahren wird beispielsweise durch die Typen im Namespace `System.Collections.Generic` der .NET-Basisklassenbibliothek angewendet um stark typisierte Auflistungen von Objekten zu realisieren. Generische Bausteine können Klassen, Methoden, Schnittstellen oder auch Delegates sein.
- § **Exception** – Das .NET-Framework bietet ein einheitliches Modell für die Behandlung von Ausnahmen. Alle Ausnahmeklassen leiten sich von der Klasse `Exception` im Namespace `System` der .NET-Basisklassenbibliothek ab. Das Auswerten, Abfangen und Reagieren auf Ausnahmen wird mit Hilfe von `try`-Anweisungen und `catch`-Klauseln umgesetzt.

Für alle genannten Begriffe finden sich in der Fachliteratur ausführliche Erläuterungen.

2.3 Component Object Model

Das Component Object Model (COM) ist ein von Microsoft entwickeltes Fundament zur einheitlichen Kommunikation verschiedener Komponenten. Die Interaktion zwischen voneinander unabhängigen Anwendungen stellt das Hauptanliegen dar.

Ein oft zitiertes Beispiel in diesem Zusammenhang ist das Einbinden einer Excel-Tabelle in ein Word-Dokument. Dafür wird ein standardisiertes Verfahren der Kommunikation dieser beiden unterschiedlichen Applikationen benötigt. Das Component Object Model ermöglicht genau diese Kommunikation verschiedenster Komponenten auf einer binären Basis (vgl. [STAL]).

Die COM-Technologie kann als Vorgängerstandard zu .NET betrachtet werden.

2.3.1 Bereitstellung

Anwendungen die ihre Funktionalitäten für andere Software zur Verfügung stellen tun dies mit Hilfe von COM-Objekten. Abbildung 4 stellt die Einbettung eines COM-Objekts in einem Server dar und visualisiert den Zugriff eines Clients auf die Schnittstellen des COM-Objektes.

Abbildung 10: Client-Server Kommunikation in COM (eigene Darstellung)

Die Abbildung zeigt, dass ein COM-Objekt innerhalb eines Servers implementiert ist. Die Bereitstellung eines COM Servers kann auf drei Arten erfolgen (vgl. [CHA96], S. 53):

- § **In-Process Server** – Die COM-Komponente liegt als Bibliothek vor (DLL) und wird in den Adressraum des aufrufenden Prozesses geladen.
- § **Local Server** – Ein lokaler Server, der auf dem Computer des Clients gestartet wurde, stellt die COM-Objekte bereit.
- § **Remote Server** – Der Aufruf einer COM-Komponente erfolgt über ein Netzwerk.

Ein COM-Objekt stellt eine Instanz einer COM-Klasse dar (Bezeichnung: CoClass) welche eine, oder in den meisten Fällen, mehrere Schnittstellen zur Verfügung stellt. Der Zugriff auf das COM-Objekt erfolgt nicht direkt, sondern immer über diese Schnittstellen. Eine Schnittstelle (Interface) definiert Methoden, auf die mit einem Schnittstellenzeiger zugegriffen werden kann. Die direkte Interaktion mit dem COM-Objekt ist also nicht möglich.

Jede COM Schnittstelle ist von der Basisschnittstelle IUnknown abgeleitet und stellt damit 3 Methoden automatisch zur Verfügung. Die Methoden AddRef und Release dienen der Referenzzählung, also der Clients welche die Schnittstelle referenzieren. Die 3. Methode QueryInterface ermöglicht den Zugriff auf andere Schnittstellen des COM-Objektes (vgl. [STAL]).

2.3.2 Interoperabilität

In COM liegen Client und Server als unabhängige binäre Komponenten vor. Dabei kann die Realisierung der Anwendungen in unterschiedlichen Programmiersprachen erfolgen. (vgl. [STAL]). Voraussetzung für den Einsatz einer Programmiersprache zur Entwicklung eines COM-Objektes ist die Unterstützung der durch COM definierten binären Schnittstelle. Ein Client muss nun in der Lage sein, Methoden über diese Schnittstelle des Objektes aufzurufen.

In diesem Zusammenhang ist es notwendig, auf die Kommunikation von Client und Server einzugehen. Die beschriebene Methode des Aufrufs der Methoden über einen Schnittstellenzeiger ermöglicht lediglich die Einbindung lokaler Objekte (als DLL), welche zur Laufzeit in den Adressraum des Clients geladen werden. Wenn ein Client jedoch auf ein COM-Objekt zugreifen möchte, das sich in einem anderen Adressraum befindet, zum Beispiel in einem anderen Prozess als ausführbares Programm, ist der Einsatz von Proxies und Stubs notwendig. Auf Abbildung 11 wird die Kommunikation zwischen einem Client und einem lokalen Server dargestellt.

Abbildung 11: Zugriff auf COM-Objekt in einem lokalen Server (vgl. [CHA96], Seite 70)

Die Abbildung 11 zeigt, dass bei dem Zugriff auf COM-Komponenten in einem anderen Prozess Proxies und Stubs eine Rolle spielen. Weil der Client nicht direkt einen Schnittzeiger auf die Schnittstelle des Objekts im Server setzen kann, übernimmt der Proxy eine Stellvertreterrolle. Er präsentiert dem Client die erwartete Schnittstelle des lokalen Servers und übernimmt die Kommunikation mit dem COM-Objekt, konvertiert die Parameter (Marshalling) und meldet etwaige Ergebnisse (vgl. [CHA96], S. 70 f.).

Ein Stub Objekt hat die entgegengesetzte Rolle des Proxies. Es liegt im Adressraum des aufgerufenen Prozesses und empfängt die Aufrufe, entpackt die Parameter (Demarshaling) und ruft damit das angeforderte COM-Objekt auf (vgl. [STAL]).

Bei prozessübergreifenden Zugriffen auf COM-Objekte erfolgt also im Gegensatz zu prozessinternen Zugriffen eine indirekte Kommunikation mit Hilfe vorgelagerter Stellvertreterobjekte, welche die Kommunikation übernehmen.

2.3.3 IDL

Die Bereitstellung von Proxy- und Stub Objekten und Code für das Marshaling sowie Demarshaling kann mit Hilfe der Definitionssprache Interface Definition Language erfolgen. Mit Hilfe des IDL Compilers von Microsoft (MDIL) wird aus den in IDL definierten Schnittstellen der benötigte Code generiert (vgl. [STAL]).

Des Weiteren erzeugt der Compiler eine Typenbibliothek (Type Library). Diese liefert eine Beschreibung der Schnittstellen, die durch ein COM-Objekt zur Verfügung gestellt wird. Das umfasst die Methoden sowie deren Signaturen. Eine Typenbibliothek liegt entweder als eine Datei vor (Dateieindung: *tlb*) oder wird direkt mit der entsprechenden Datei geliefert.

2.3.4 Objektorientierung

In "klassischen" objektorientierten Programmiersprachen wie C++ spielt der Mechanismus der Implementierungsvererbung eine entscheidende Rolle. Die Vererbung zwischen einer Basisklasse und einer erbbenden Klasse wird durch COM nicht unterstützt. (vgl. [CHA96], Seite 13 f.).

Aus diesem Grund gibt es in COM andere Methoden zur Wiederverwendung entwickelter Komponenten.

Zum einen ist es möglich mit Hilfe von Kapselung (containment) die Methoden einer anderen Komponente zu nutzen.

Der zweite Weg wird als Aggregation bezeichnet und erlaubt einem COM-Objekt die Schnittstellen eines anderen Objektes als seine eigenen darzustellen. Abbildung 12 stellt schematisch das Prinzip der Aggregation dar.

Abbildung 12: Aggregation (COM-Technologie, vgl. [STAL])

Des Weiteren wird Polymorphismus unterstützt. Es ist also für Objekte möglich dieselbe Methode auf verschiedene Art und Weise zu implementieren.

2.3.5 Identifikation

Zur Identifikation von COM-Objekten, Schnittstellen, Typbibliotheken und Klassen ist es notwendig, diese eindeutig zu identifizieren. Weil es nicht garantiert ist, dass die gewählten lesbaren Bezeichnungen eindeutig sind, geschieht die Identifikation mit Hilfe von Global Unique Identifiers (GUIDs). Dabei handelt es sich um eine Zeichenfolge mit einer Größe von 128 Bit. Um COM-Objekte zu nutzen müssen die Identifikatoren registriert werden. Im Falle von Windows wird dabei die Systemregistratur genutzt, was Abbildung 13 darstellt.

Abbildung 13: Registrierung einer COM-Klasse (vgl. [STAL])

Auf Abbildung 13 wird ein Beispiel für das Registrieren einer COM-Klasse in der Sektion HKEY_CLASSES_ROOT der Systemregistrator veranschaulicht. Es wird eine konkrete Implementierung der Klasse Search als ausführbares Programm in `c:\bin\shape.exe` in der Systemregistrator hinterlegt (vgl. [STAL]).

In diesem Zusammenhang ist es noch notwendig die Funktion einer ProgID (programmatische ID) zu erläutern. Die ProgID stellt ein lesbares Synonym für die CLSID einer Klasse dar. In der Systemregistrator wird mit der ProgID auf die entsprechende CLSID verwiesen. Sie setzt sich aus dem Namen der COM-Komponente, dem Namen der COM-Klasse und einer Versionsnummer zusammen.

2.3.6 Automatisierung

COM-Komponenten stellen anderen Anwendungen oder Komponenten ihre Dienste zur Verfügung. Wenn in diesem Zusammenhang durch ein COM-Objekt Dienste angeboten werden, die ein Benutzer auch über die grafische Oberfläche erledigen kann, wird von Automatisierung gesprochen. Somit ist eine direkte Steuerung einer Software von außerhalb möglich.

Ein Beispiel das den Nutzen der Automatisierung zeigt, ist die programmatische Steuerung von Excel- oder Worddokumenten. Demzufolge wird zum Beispiel nicht nur Nutzern über eine grafische Oberfläche erlaubt eine Excel-Tabelle anzulegen und in ein Word-Dokument zu exportieren, sondern auch externen Anwendungen (vgl. [CHA96], S. 85 ff.).

Die universelle COM Schnittstelle IDispatch erleichtert die programmatische Steuerung von Anwendungen. Sie ermöglicht Anwendungen einen zentralen Zugriffspunkt auf ihre automatisierte Funktionalität zu bieten. Die Schnittstelle erbt von *IUnknown* und bietet folgende Methoden (vgl. [STAL]):

- § **getTypeInfoCount** – Liefert Informationen darüber ob ein COM-Objekt zur Laufzeit Typinformationen enthält (vgl. [CHA96], Seite 98),
- § **getTypeInfo** – Liefert Informationen über die Funktionalitäten der Applikation.
- § **getIDsOfNames** – Ruft die ID ab, welche einer Methode zugeordnete ist.
- § **invoke** – Bietet einer Anwendung die Möglichkeit Methoden der COM-Schnittstelle mit deren Parametern aufzurufen.

2.3.7 Persistenz

Im Zusammenhang mit der Schnittstelle *IUnknown* wurde erläutert, dass eine Referenzzählung auf ein COM-Objekt erfolgt. Somit existiert ein COM-Objekt immer solange wie eine Referenz durch ein oder mehrere Clients darauf besteht. Unter Persistenz versteht man die Fähigkeit eines COM-Objektes, seine Daten dauerhaft zu speichern und wieder abzurufen. Eine ausführliche Beschreibung dieser Eigenschaft erfolgt in [CHA96] (Seite 107 ff.).

2.3.8 Running Object Table

Bei der Running Object Table (kurz: ROT) handelt es sich um eine öffentliche Tabelle auf einem Computer. Bei Bedarf können sich laufende COM-Server in diese Tabelle ein-bzw. austragen. Dadurch wird es Anwendungen, die diese Komponente nutzen wollen, ermöglicht, eine bereits laufende Instanz zu benutzen (siehe [CHA96], Seite 145).

2.3.9 Interopabilität zwischen .NET und COM

Bevor das .NET-Framework als Entwicklungsplattform von Microsoft eingeführt wurde sind weltweit viele COM-Komponenten entwickelt worden. Es würde hohe Kosten verursachen, den entwickelten Code neu zu planen und zu implementieren. Aus diesem Grund bieten die Common Language Runtime Mechanismen an, mit denen es möglich ist, nicht verwalteten Code in verwaltetem Code zu nutzen. COM-Objekte stehen dadurch in .NET-Anwendungen zur Verfügung (vgl. [RIC06], S. 54).

Aus der Sicht des Entwicklers muss es ermöglicht werden, dass ein COM-Typ als .NET-Äquivalente zur Verfügung steht. Die in COM definierten Typen werden durch eine Zwischenschicht, dem Runtime Callable Wrapper (RCW), in .NET verfügbar gemacht und können dort wie eigene Typen behandelt werden. Auf Abbildung 14 wird verdeutlicht wie der RCW ein COM-Objekt kapselt (vgl. [TRO02], S. 613).

Abbildung 14: Runtime Callable Wrapper (vgl. [TRO02], Seite 613)

Im Abschnitt 3.4 wird diese Vorgehensweise genauer betrachtet und im Zusammenhang mit dem Softwareprojekt erläutert.

3 Planung der Software

Das folgende Kapitel legt einige theoretische Grundlagen zum Entwurf der Software. Als erstes werden grundlegende Überlegen zu einem Framework präsentiert und im nachfolgenden Abschnitt wird eine kurze Einführung zum Thema Entwurfsmuster in der Softwareentwicklung gegeben. Die Entwicklung des Frameworks wurde durch den Einsatz von Entwurfsmustern realisiert. An konkreten Beispielen wird die Verwendung der Muster beschrieben. Als letztes wird die COM-Komponente von S-PLUS analysiert. Auf dieser Grundlage wird das vierte Kapitel aufgebaut.

3.1 Framework

Ein Framework ist eine wieder verwendbare und objektorientierte Bibliothek. Es stellt eine Grundstruktur für die Entwicklung von Anwendungen zur Verfügung und bietet in einem einheitlichen Gesamtkonzept wiederverwendbare Bausteine, eine Menge von zusammenarbeitenden Klassen und Hilfsmitteln. Die Aufgabe eines Frameworks besteht in der Erstellung eines wiederverwendbaren Entwurfsmusters. Entwicklern soll eine breite Basis zur Gestaltung von Applikationen für unterschiedliche Szenarien und Anforderungen im Zusammenhang mit einer bestimmten Art von Software oder System geboten werden. Die Spannweite der Größe von Frameworks reicht von System-Frameworks wie dem .NET-Framework bis zu kleinen Komponenten, die wiederverwendbare Bibliotheken für spezifische Anforderungen zur Verfügung stellen.

Im Rahmen der Diplomarbeit wurde ein Framework entworfen, das einen standardisierten Zugriff auf die Datenobjekte des Veranstaltungsplaners S-PLUS ermöglicht. Es handelt sich dabei also um eine wiederverwendbare Komponente, die Klassen und Schnittstellen sowie Hilfsmittel in Bezug auf S-PLUS zusammenfasst. Das Ziel besteht darin, ein einheitliches und umfangreiches Programmiermodell zur Verfügung zu stellen, das verschiedenste Anforderungen für die Entwicklung von Anwendungen im Zusammenhang mit S-PLUS erlaubt.

Neben der Wiederverwendbarkeit sind Einfachheit, Zuverlässigkeit, Sicherheit, Erweiterbarkeit und Konsistenz entscheidende Merkmale und Qualitätskriterien eines Frameworks (vgl. [CA07], Seite 27 ff.).

3.2 Entwurfsmuster

Das Buch "Entwurfsmuster - Elemente wiederverwendbarer objektorientierter Software" gilt als Standardwerk zur Thematik Entwurfsmuster. Das Autorenteam beschreibt in diesem Buch wiederkehrende Probleme in der objektorientierten Programmierung und liefert entsprechende Lösungsansätze durch Entwurfsmuster. Im folgenden Kapitel getroffene Aussagen, beziehen sich somit auf diese theoretischen Ansätze (siehe [GHJV07]).

Unter einem Entwurfsmuster versteht man die Lösung für ein wiederkehrendes Problem in der objektorientierten Softwareentwicklung. Die Lösung bzw. Schablone soll beliebig oft angewendet werden können (vgl. [GHJV07], Seite 3). Durch ein Muster wird die Lösung für eine Problemstellung allerdings nur beschrieben. Eine Umsetzung erfolgt dann mit dem Mitteln der objektorientierten Programmierung.

Tabelle 2 zeigt, dass Entwurfsmuster an Hand ihrer Aufgabe und ihres Gültigkeitsbereiches eingeteilt werden.

		Aufgabe		
		Erzeugungsmuster	Strukturmuster	Verhaltensmuster
Gültigkeitsbereich	klassenbasiert	Fabrikmethode	Adapter (klassenbasiert)	Interpreter Schablonenmethode
	objektbasiert	Abstrakte Fabrik Erbauer Prototyp Singleton	Adapter (objektbasiert) Brücke Dekorierer Fassade Fliegengewicht Kompositum Proxy	Befehl Beobachter Besucher Iterator Memento Strategie Vermittler Zustand Zuständigkeitskette

Tabelle 2: Dimensionen der Ausprägungen von Entwurfsmustern (vgl. [GHJV07], Seite 14)

In Tabelle 2 ist zu erkennen, dass die Aufgaben von Entwurfsmustern sich in den Prozess der Objekterzeugung (Erzeugungsmuster), der Zusammensetzung von Klassen, Objekten und Implementierungen (Strukturmuster) sowie in die Charakterisierung der Interaktion von Objekten (Verhaltensmuster) unterscheiden lassen. Der Gültigkeitsbereich unterscheidet ob ein Muster klassenbasiert (durch Vererbung) oder objektbasiert (Objektbeziehungen) ist. Bei der Fabrikmethode handelt es sich beispielsweise um ein klassenbasiertes Erzeugungsmuster das die Erzeugung von Objekten auf Basis der Vererbung durch Klassen beschreibt.

Bei der Entwicklung des Frameworks mussten die Kriterien und Einsatzmöglichkeiten von Entwurfsmustern überdacht und ihre Verwendung geplant werden.

In Kapitel 4.1 wird an Hand von konkreten Problemstellungen der Einsatz bestimmter Entwurfsmuster im Framework erläutert und begründet.

3.3 S-PLUS Automationsserver

Im Kapitel 2.3 wurden Grundlagen der COM-Technologie dargestellt. Scientia stellt mit dem Veranstaltungsplaner S-PLUS eine COM-Komponente zur Verfügung, die das programmatische Steuern der Anwendung von außerhalb erlaubt. Daten und Funktionalitäten werden durch das S-PLUS Image zur Verfügung gestellt. Auf Abbildung 15 wird das Prinzip der Bereitstellung des Images für Windows abgebildet.

Abbildung 15: S-PLUS Automationsserver für Windows (COM-Server Handbuch, vgl. [SAS])

In dem Image werden Daten, Methoden und Funktionalitäten dauerhaft und somit persistent gespeichert. Zur Laufzeit meldet es sich über die Running Object Table beim Betriebssystem an. Anderen Anwendungen wird es somit ermöglicht über diesen Eintrag auf die Schnittstellen und Methoden des COM-Objektes zuzugreifen und die Applikation von außen zu steuern. Die mitgelieferte Typbibliothek liefert Informationen über Schnittstellen, Methoden und deren Parameter sowie zu Klassen und Aufzählungstypen.

In ersten Schritt der Planung des Frameworks war es notwendig, die Art der Bereitstellung der COM-Komponente zu betrachten. Danach war es erforderlich, die Typbibliothek zu analysieren und im letzten Schritt wurde die Einbindung der COM-Komponente in das .NET-Projekt realisiert.

Die Registrierung des COM-Servers kann mit Hilfe der grafischen Benutzeroberfläche von S-PLUS erfolgen. Der Menüpunkt COM ermöglicht das Eintragen der ProgID und das Registrieren beziehungsweise Ausregistrieren des COM-Servers. Auf Abbildung 16 wird die ProgID "SS07" (an der Technischen Fachhochschule Wildau wird die ProgID an Hand des Semesternamens vergeben, in diesem Fall handelt es sich um das Sommersemester 2007) eingetragen. Anschließend wird der COM-Server registriert.

Abbildung 16: Eintragen der ProgID (Bildschirmkopie)

Wie im Kapitel 2.3.5 erläutert, muss nach der Registrierung des COM-Servers die Eintragung eines eindeutigen Bezeichners (CLSID) in die Systemregistratur erfolgen. Wenn man nach den Registrierungseinträgen des Servers in der Systemregistratur von Windows sucht, findet man den entsprechenden Eintrag unter dem Pfad CLSID, der sich unter *HKEY_CLASSES_ROOT* befindet. Einfacher geht es jedoch mit dem Dienstprogramm *OLE/COM Object Viewer* (Download, siehe [OCOV]), welches einen Überblick über installierte COM-Klassen liefert.

Auf der linken Seite des Programms befindet sich ein Menübaum, in dem sich unter Object Classes / All Objects der Eintrag "SPLUS server (3.5.1 xxi)" befindet. Abbildung 17 zeigt den Eintrag der COM-Klasse, ihre Schnittstellen und weitere Informationen.

Abbildung 17: Eintrag des S-PLUS Images in der Systemregistratur (Bildschirmkopie)

Im Kapitel 2.3.5 wurde erläutert, dass die ProgID ein eindeutiger Bezeichner einer COM-Klasse ist. In Abbildung 17 ist zu erkennen, dass die ProgID der COM-Klasse *SS07.Application.69* lautet. Daraus lässt sich die Information gewinnen, dass der Name der registrierten COM-Komponente *SS07*, der Name der COM-Klasse *Application* und die Versionsnummer *69* ist.

Der Eintrag *LocalSever32* zeigt, dass die COM-Klasse *Application* in einem lokalen COM-Server eingebettet ist. Der Prozess des lokalen COM-Servers wird durch die Ausführung der Datei *splus.exe* gestartet.

Um die Anwendung S-PLUS also programmatisch von außen zu steuern, ist es erforderlich ein Objekt der Klasse *Application* zu erzeugen. Wie das Erstellen eines Objektes der Klasse *Application* erfolgt, wird im Abschnitt 4.1.1 beschrieben.

Die Klasse bietet folgende Eigenschaften um die Anwendung S-PLUS zu steuern:

- § **ActiveCollege** – Diese Eigenschaft liefert alle Daten des aktuellen Images. Dazu zählen beispielsweise Dozenten, Studenten oder Räume. Zu Arbeit mit den Datenobjekten des Images nutzt man diese Eigenschaft.
- § **MainWindow** – Mit Hilfe dieser Eigenschaft ist ein beschränkter Zugriff auf das Hauptfenster der grafischen Benutzeroberfläche des Veranstaltungsplaners möglich.
- § **SDB** – Das Kürzel SDB steht für Scientia Database. Über diese Eigenschaft liefert die Klasse den Zugriff auf die in Kapitel 1.1 angesprochenen Datenbankfunktionalitäten des Veranstaltungsplaners.

3.4 Konvertierung der Typbibliothek

Wie bereits angesprochen liefert S-PLUS eine Typbibliothek, welche Informationen über Klassen, Schnittstellen und Methoden der COM-Komponente enthält. Die entsprechende Typbibliothek hat den Namen *splus.tlb*.

Die Typbibliothek lässt sich mit Hilfe des Programms *ITypeLib Viewer* betrachten, welches in *OLE/COM Object Viewer* integriert ist. Abbildung 18 zeigt die Startansicht des *ITypeLib Viewer* für die Typbibliothek.

Abbildung 18: S-PLUS Typbibliothek in *ITypeLib Viewer* (Bidschirmkopie)

Neben Aufzählungen (Enums) werden COM-Klassen (CoClasses) und Interfaces in der Typbibliothek definiert.

Alle COM-Klassen in der Typbibliothek haben das Attribut *noncreatable* und somit kann von ihnen keine Instanz über sie selber erstellt werden. Nur die Klasse *Application* hat dieses Attribut als Einstiegspunkt für den COM-Server nicht. Diese Klasse hat das Attribut *appobject*, was sie als ein Applikationsobjekt im Zusammenhang mit einer ausführbaren .EXE-Datei identifiziert.

Um die beschriebene Typbibliothek im .NET-Framework nutzen zu können, muss sie in ein äquivalentes Assembly umgewandelt werden. Diese Konvertierung wird mit Hilfe des Dienstprogramms *Tlbimp.exe* (Type Library Importer) vollzogen. Dabei werden die Typdefinitionen der Typbibliothek in Metadaten eines Assemblys umgewandelt. Der Befehl zum Bereitstellen der COM-Typen als .NET-Äquivalente sieht folgendermaßen aus:

```
tlbimp splus.tlb /out:SplusServer.dll
```

Das erzeugte Assembly im Rahmen der Diplomarbeit heißt also *SplusServer.dll*. Weitere Informationen über die Aufgabe von *Tlbimp.exe* und mögliche Argumente für den Befehl der Konvertierung sind auf der entsprechenden Webseite von Microsoft (siehe [TLI]) zu finden.

Interessant ist nun wie die Umwandlung stattgefunden hat und welche Typen im Assembly *SplusServer.dll* zur Verfügung stehen. Die Konvertierung der Typen erfolgt nach bestimmten Regeln (Konvertierungsregeln, siehe [TLISUM]).

Als Grundlage soll die COM-Klasse Tag (Flagge) zur Darstellung der Konvertierung einer COM-Klasse und ihrer Schnittstellen genutzt werden. Auf dieselbe Weise werden auch alle anderen COM-Klassen in der Typbibliothek definiert und durch *Tlbimp.exe* konvertiert. Die Darstellung einer Flagge erfolgt in der Typbibliothek in IDL auf folgende Weise:

```
dispinterface ITag {
    // Methoden
}
dispinterface ITagEvents {
    // Methoden
}

[
    /* .. */
    noncreatable
]
coclass Tag {
    [default] dispinterface ITag;
    [default, source] dispinterface ITagEvents;
    [source] dispinterface IObjectWithDictionaryEvents;
    [source] dispinterface IObjectWithIdEvents;
    [source] dispinterface IObjectWithUserTextEvents;
    [source] dispinterface IObjectWithNameEvents;
    [source] dispinterface IObjectWithTagsEvents;
    dispinterface IObjectWithDictionary;
    dispinterface IObjectWithId;
    dispinterface IObjectWithUserText;
    dispinterface IObjectWithName;
    dispinterface IObjectWithTags;
};
```

Das Attribut *default* zeigt an, dass diese Schnittstelle die Standardschnittstelle der COM-Klasse ist. Mit Hilfe des Attributs *source* wird gezeigt, dass diese Schnittstelle Ereignisse liefert.

Neben den beiden Standardschnittstellen können die COM-Klassen noch weitere Schnittstellen aggregieren. Diese Schnittstellen stellen jedoch keine Erweiterung an Signaturen für die COM-Klasse dar, sondern können als Basisschnittstellen bezeichnet werden.

Beim Konvertieren einer COM-Klasse wird eine verwaltete Klasse erstellt, die den Zusatz *Class* erhält. Für die verwaltete Klasse wird automatisch ein Standardkonstruktor hinzugefügt, außer die COM-Klasse hat das Attribut *noncreateable*. Somit haben die Klassen im konvertierten Assembly *SplusServer.dll* keinen Standardkonstruktor. Die in den Schnittstellen der COM-Klasse definierten Member werden der verwalteten Klasse hinzugefügt (vgl. [TLISUM]).

Des Weiteren wird zu jeder Klasse eine Schnittstelle hinzugefügt, die den Namen der COM-Klasse hat. Generell werden bei allen importierten Schnittstellen die Methoden der Schnittstellen *IUnknown* und *IDispatch* sowie anderen Standardschnittstellen entfernt (vgl. [TLISUM]).

Folgende Typen stehen im Assembly nach der Konvertierung unter anderem bezüglich einer Flagge zur Verfügung (Deklaration in C#):

```
// Dispinterface ITag - Umwandlung
public interface ITag {...}

// Dispinterface ITagEvents – Umwandlungen
public interface ITagEvents {...}
public interface ITagEvents_Event {...}

// Schnittstelle aus COM-Klasse
public interface Tag : ITag, ITagEvents_Event {...}

// CoClass Tag - Umwandlung
public class TagClass : Tag, ITag, ITagEvents_Event, IObjectWithDictionary,
IObjectWithDictionaryEvents_Event, IObjectWithId, IObjectWithIdEvents_Event,
IObjectWithName, IObjectWithNameEvents_Event, IObjectWithTags,
IObjectWithTagsEvents_Event, IObjectWithUserText,
IObjectWithUserTextEvents_Event {...}
```

Entscheidend ist die erstellte Schnittstelle *Tag*. Diese Schnittstelle implementiert die beiden Standardschnittstellen der COM-Klasse und bietet Zugriff auf alle Member. Diese Schnittstelle ist damit der Runtime Callable Wrapper für Objekte der COM-Klasse *Tag*.

Die erstellte Klasse *TagClass* hat keinen Konstruktor und somit kann keine Instanz dieser Klasse erzeugt werden. Ein wichtiger Punkt ist jedoch die Erkenntnis, dass die Klasse alle Schnittstellen implementiert, die in der COM-Klasse *Tag* angegeben wurden. Das erlaubt die Konvertierung eines Objektes vom Typ *Tag* in eine der Schnittstellen, die durch die Klasse *TagClass* implementiert werden. Welche Auswirkungen dieser Mechanismus für die Implementierung von Objekten im Framework hat, wird in Kapitel 4.1.2 erläutert.

Des Weiteren werden für alle Ereignisse, die in Ereignisschnittstellen definiert sind, Delegates im Assembly erzeugt. Mit diesen Delegates können die Ereignisse eines COM-Objektes abgerufen werden.

Als letztes wird immer eine Klasse angelegt, die als Ereignissenke dient. Im Falle des Imports der COM-Klasse *Tag* und der Ereignisschnittstelle *ITagEvents* heißt die erzeugte Klasse *ITagEvents_SinkHelper*. Diese Klassen spielen bei der weiteren Betrachtung jedoch keine Rolle.

Durch die Konvertierung der Typbibliothek wurde die Anzahl der Typen zur Darstellung eines Objektes drastisch erhöht. Für die COM-Klasse *Tag* und ihrer beiden Standardschnittstellen sind im Assembly *SplusServer.dll* vier Schnittstellen, zwei Klassen und zwei Delegates deklariert. Nach der Konvertierung werden durch das Assembly mehrere tausend Typen zur Verfügung gestellt.

Dies ist vor allem für Entwickler, die sich mit der COM-Technologie nicht auskennen und sich in die COM-Schnittstelle einarbeiten wollen, verwirrend. Das Framework soll die Menge der Typen auf das Nötigste reduzieren. Mit welchen Mitteln dieses Ziel verfolgt wird, erläutert das nächste Kapitel.

4 Entwicklung der Software

Dieses Kapitel beschäftigt sich mit der Entwicklung des Frameworks, der Anpassung der Webpräsentationskomponente und dem auf dem Framework basierenden Prüfungsplaner. Es verfolgt das Ziel, die Grundideen zu präsentieren und aufgetretene Probleme und ihre Lösungsansätze zu beschreiben.

4.1 Framework

Im dritten Kapitel wurden die Grundlagen eines Frameworks, von Entwurfsmustern sowie das Konvertieren der Typbibliothek erläutert. Diese Themen stellen den theoretischen Unterbau für das folgende Kapitel dar.

Bei dem im Rahmen der Diplomarbeit entwickelten Framework handelt es sich um eine Klassenbibliothek. Die Klassenbibliothek fasst notwendige Typen und Operationen zusammen um die Nutzung der COM-Schnittstelle zu vereinfachen. Durch das Einbinden des Frameworks soll es nicht mehr nötig sein, die Typbibliothek der COM-Schnittstelle zu benutzen. Zu diesem Zweck wurden Typen zum Adaptieren, Erstellen und Bearbeiten der COM-Objekte entworfen. Diese werden in Namespaces angeordnet. Tabelle 3 listet die wichtigsten Namespaces der Klassenbibliothek auf.

Name	Typen
Splus.Framework	Enthält globale statische Klassen.
Splus.Framework.Data	Enthält Klassen und Schnittstellen der Datenobjekte.
Splus.Framework.Data.Model	Enthält Basisklassen und Schnittstellen für die Datenobjekte.
Splus.Framework.Data.Groups	Enthält Klassen und Schnittstellen der Datenobjekte die eine Gruppe darstellen.
Splus.Framework.Event	Enthält Delegates und Ereignisklassen.

Tabelle 3: Namespaces des Frameworks (eigene Darstellung)

In den folgenden Abschnitten wird beschrieben, wie das Framework die COM-Schnittstelle einbindet und die Objekte der COM-Schnittstelle zur Verfügung stellt. Dabei wird an Hand von konkreten Beispielen der Zugriff auf die Objekte direkt über die COM-Schnittstelle und über das Framework verglichen. Verschiedene Probleme beim Zugriff auf Typen werden erläutert und die Lösung dafür im Framework aufgezeigt. Insgesamt soll der Vorteil durch die Nutzung des Frameworks vermittelt und im Zuge dessen die Erfüllung der gestellten Anforderungen geklärt werden.

4.1.1 Einbinden der COM-Komponente

Im Abschnitt 3.3 wurde erläutert, dass die COM-Klasse *Application* den Einstiegspunkt für die Arbeit mit dem Automationsserver von S-PLUS bildet. Das folgende Kapitel beschäftigt sich mit der Frage, wie eine Instanz dieser Klasse erzeugt werden kann und wie dieser Vorgang durch das Framework gekapselt wird.

Die Schnittstellen der Klasse *Application* befindet sich in der konvertierten Typbibliothek, welche im Abschnitt 3.4 erklärt wurde. Nachdem das Assembly *SplusServer.dll* in ein Projekt eingebunden wurde, kann man annehmen, dass die Erzeugung eines Objektes der Klasse *Application* folgendermaßen funktioniert:

```
SplusServer.ApplicationClass Application = new SplusServer.ApplicationClass();
```

Diese Vorgehensweise führt jedoch nicht zu dem gewünschten Ergebnis. Bei diesem Vorgang wird keine Referenz auf einen aktiven COM-Server erzeugt und somit können keine Datenobjekte abgerufen werden.

Es muss vielmehr die Möglichkeit bestehen, ein Objekt der Klasse *Application* zur Laufzeit zu erzeugen. Voraussetzung dafür ist ein aktiver COM-Server. Man kann den COM-Server selber starten oder ein Objekt eines gestarteten COM-Servers erstellen.

Um zu verstehen wie ein COM-Server zur Laufzeit gestartet werden kann, muss man sich noch einmal die in Kapitel 3.3 erwähnten Einträge der COM-Komponente in der Systemregistrator in Erinnerung rufen. Entscheidend sind folgende Einträge:

- § **ProgID** – Eindeutiger Bezeichner der COM-Komponente. Setzt sich aus dem Namen der Komponente ("SS07"), dem Namen der COM-Klasse ("Application") und einer Versionsnummer zusammen.
- § **LocalServer32** – Beschreibt wie der lokale Prozess gestartet wird, in dem der COM-Server vorliegt. Wenn man sich den Eintrag anschaut wird deutlich, dass durch den Start der ausführbaren Datei *splus.exe* ein lokaler Prozess gestartet wird und das Image mit dem Namen SS07 den COM-Server darstellt.

Die Basisklassenbibliothek von .NET bietet entsprechende Möglichkeiten, um mit diesen Angaben einen COM-Server zu starten und ein Objekt der Klasse *Application* zur Laufzeit zu erzeugen.

Dieser, auch als spät gebundener Methodenaufruf bezeichneter, Vorgang wird in zwei Schritten durchgeführt.

Als erstes wird der Typ der COM-Klasse abgefragt. Das Abfragen des Typs kann auf folgende Weise geschehen:

```
// Typ über ProgID abrufen  
Type type = Type.GetTypeFromProgID("SS07.Application");
```

Die Basisklassenbibliothek von .NET liefert im Namespace *System* die Klasse *Type*. Diese Klasse stellt Typdeklarationen dar und mit Hilfe der Methode *GetTypeFromProgID* lässt sich der Typ der COM-Klasse *Application* über die ProgID der COM-Komponente abrufen.

Im zweiten Schritt wird mit Hilfe des abgefragten Typs ein COM-Server gestartet.

```
// COM-Server starten  
Activator.CreateInstance(type);
```

Auch die Klasse *Activator* befindet sich im Namespace *System* der Basisklassenbibliothek des .NET-Frameworks. Mit Hilfe der Methode *CreateInstance* und dem Typen der COM-Klasse erzeugt die Klasse *Activator* ein Objekt der COM-Klasse.

Beim Erzeugen dieses Objektes wird der COM-Server gestartet und die Datenobjekte des Automationsservers stehen über dieses Objekt zur Verfügung.

Der Rückgabewert der Methode *CreateInstance* ist vom Typ *object*. Um auf die Methoden und Eigenschaften der COM-Klasse zuzugreifen, wird eine Typumwandlung für den in der Typbibliothek definierten Typ *SplusServer.Application* vorgenommen. Somit lautet der Code zum Erzeugen des COM-Servers:

```
// COM-Server starten, mit Typumwandlung  
SplusServer.Application application =  
    (SplusServer.Application) Activator.CreateInstance(type);
```

Die Erzeugung des COM-Servers wird durch das Framework übernommen. Die statische Klasse *SPLUS* im Namespace *Splus.Framework* kapselt den Vorgang in der Methode *Start*. Demzufolge muss eine Anwendung, die das Framework nutzen will, als erstes immer die Methode *Start* aufrufen.

Bisher wurde nur das Starten eines COM-Servers dargestellt. Eine weitere Möglichkeit ist es, einen bereits gestarteten COM-Server zu nutzen. Wie in Kapitel 2.3.8 erläutert tragen sich gestartete COM-Server in die *Running Object Table* des Betriebssystems ein. Wenn der Automationsserver mehrmals gestartet wird, werden auch mehrere Einträge in der *Running Object Table* hinterlegt.

Die Methode *CreateInstance* der Klasse *Activator* startet einen COM-Server, wenn noch kein Eintrag in der *Running Object Table* vorhanden ist. Falls es schon einen Eintrag in der *Running Object Table* gibt, wird kein neuer COM-Server erzeugt, sondern nur eine Referenz auf einen bestehenden COM-Server geliefert. Somit übernimmt eine Anwendung die Kontrolle über eine bereits gestartete Instanz des Veranstaltungsplaners. Dieses Verhalten ist aus den folgenden Gründen nicht wünschenswert:

- § **Benutzerfreundlichkeit** – Wenn das Framework einen bestehenden COM-Server bindet, führt das zu benutzerunfreundlichen Konsequenzen. Beim Start einer Anwendung die das Framework benutzt, würde die Kontrolle einer offenen Instanz des Veranstaltungsplaner an diese Anwendung gehen. Somit können vorgenommene Datenänderungen zerstört werden. Des Weiteren kann das Hauptfenster der Anwendung plötzlich verschwinden oder Methoden, wie das Sichern des Images aufgerufen werden, obwohl dies der Benutzer nicht vorgesehen hat.
- § **Inkonsistenz** – Zu den Funktionalitäten des Veranstaltungsplaners gehört es, dass Image zu speichern, Daten an die Datenbank zurückzuschreiben oder Veranstaltungen neu zu planen. Das sind Prozesse die Zeit in Anspruch nehmen und zu der Zeit stattfinden können, in der das Framework das COM-Objekt erzeugen möchte. Der Zustand der Anwendung und die Ausführung von Prozessen kann nicht mehr durch das Framework kontrolliert werden. Das ist kein wünschenswerter Zustand und kann zu Fehlern bei der Ausführung führen.

Aus diesem Grund wird in der Methode *Start* abgefragt, ob schon ein COM-Server in der *Running Object Table* vorhanden ist. Wenn dies der Fall ist, wird eine Ausnahme ausgelöst und kein COM-Server gestartet. Eine Anwendung, die mit Hilfe des Frameworks auf die COM-Schnittstelle zugreift, kann also nur gestartet werden, wenn keine Instanz des Veranstaltungsplaners geöffnet ist.

4.1.2 Objektimplementation

In Kapitel 3.4 wurde die Umwandlung der Typbibliothek *sp/us.tlb* in ein Assembly erläutert. Es wurde dabei deutlich, dass für eine COM-Klasse und den beiden korrespondierenden Schnittstellen eine Vielzahl von Typen im erzeugten Assembly erstellt werden. Insgesamt stehen nach der Umwandlung mehrere tausend Typen in dem Assembly zur Verfügung.

Ein Hauptziel bei der Entwicklung des Frameworks war es, die Vielzahl der durch die Typbibliothek angebotenen Typen zu verkleinern und somit eine höhere Übersichtlichkeit im Umgang mit dem COM-Server zu erreichen. Des Weiteren sollen die Schnittstellen der Typen in der Typbibliothek an die Anforderungen potenzieller Anwendungen angepasst werden.

Die Anpassung einer Schnittstelle erfolgt mit Hilfe eines Adapters. Ein Adapter zählt innerhalb der Entwurfsmuster zu den Strukturmustern und wird auch als Wrapper bezeichnet. Das Strukturmuster kann klassen- sowie objektbasiert sein (siehe [GHJV95], S. 171 ff.). Im Falle des Frameworks wird immer von einem objektbasierten Adapter gesprochen, der sich wie auf dem in Abbildung 19 dargestellten Objektdiagramm klassifizieren lässt.

Abbildung 19: Objektdiagramm Objektadapter (vgl. [GHJV95], S. 174)

Die auf dem Objektdiagramm dargestellten Teilnehmer haben folgende Eigenschaften und Funktionen (vgl. [GHJV95], S. 174):

- § **Ziel** – Die erwartete Zielschnittstelle des Klienten.
- § **Klient** – Der Klient arbeitet nicht mit der adaptierten Klasse, sondern mit Objekten die der Zielschnittstelle entsprechen zusammen.

- § **Adaptierte Klasse** – Die Schnittstelle dieser Klasse soll durch den Adapter angepasst werden. Im Falle des Frameworks handelt es sich dabei um die Schnittstelle einer COM-Klasse.
- § **Adapter** – Der Adapter übernimmt die Aufgabe eine Klasse zu adaptieren und an die entsprechende Zielschnittstelle anzupassen. Bei Aufruf einer Methode des Adapters leitet dieser den Aufruf an die adaptierte Klasse weiter. Im Framework hält der Objektadapter eine Referenz auf eine Schnittstelle einer COM-Klasse und passt diese an.

Die Adaption der Schnittstelle einer COM-Klasse ist im Framework das durchgängige Prinzip um ein Objekt darzustellen. Das folgende Code-Beispiel zeigt die Adaption eines Objektes vom Typ Dozent (Name in der Typbibliothek: *StaffMember*) durch das Framework:

```
// Adaptiertes Objekt, Dozent
SplusServer.StaffMember dozent;
// Adapter
class Dozent {
    // Adaptiertes Objekt
    private SplusServer.StaffMember dozent;

    // Konstruktor
    Dozent(SplusServer.StaffMember dozent) {
        // Zuweisung
        this.dozent = dozent;
    }

    // Schnittstelle anpassen
    public void SchreibeEMail() {
        Console.WriteLine(dozent.Email);
    }
}

//Klient
Dozent adaptierterDozent;
adaptierterDozent.SchreibeEMail();
```

Das Beispiel zeigt, dass die Klasse Dozent den Adapter darstellt und ein Objekt des zu adaptierenden Typs über den Konstruktor geliefert bekommt. Jetzt kann der Adapter die Schnittstelle von SplusServer.StaffMember anpassen. Das umfasst das Hinzufügen, das Entfernen und das Ändern von Signaturen des adaptierten Typs. Dieser Vorgang soll durch die angelegte Methode SchreibeEMail verdeutlicht werden. Der Klient, im Falle des Frameworks eine Anwendung für die Klassenbibliothek, arbeitet mit dem Typ Dozent. Auf diese Weise können alle Typen der COM-Schnittstelle adaptiert werden.

Der zweite Schritt der Objektimplementation besteht in der Umwandlung der Vererbung durch Schnittstellen in eine Implementierungsvererbung. Es soll also eine Hierarchie aus Basisklassen und abgeleiteten Klassen erzeugt werden, die auf der Schnittstellenimplementierung der COM-Klassen basiert.

Im Abschnitt 3.4 wurde beschrieben, dass COM-Klassen des Automationsservers neben zwei Standardschnittstellen von weiteren Schnittstellen erben können. Nach einer Analyse der Schnittstellenvererbung innerhalb der Typbibliothek konnten mehrere Schnittstellen spezifiziert werden, die durch viele COM-Klassen implementiert werden. Wenn man diese Basisschnittstellen adaptiert, können Basisklassen für eine Implementierungsvererbung erstellt werden. Nach der Analyse wurde die auf Abbildung 20 dargestellte Vererbungsstruktur abstrakter Basisklassen entworfen.

Abbildung 20: Klassenhierarchie im Framework (eigene Darstellung)

Die in Abbildung 20 dargestellten Basisklassen sind alle im Namespace *Splus.Framework.Data.Model* definiert.

In Tabelle 4 wird die Bedeutung der Basisklassen dargestellt.

Name	Beschreibung
BasisObject	Fasst die Basiseigenschaften aller Objekte des Lehrangebots zusammen. Adaptiert dafür die Schnittstelle <i>SplusServer.ObjectWithName</i> .
BasisObjectWithTimePreferences	Abgeleitete Klasse von <i>BasisObject</i> . Erweitert <i>BasisObject</i> um Eigenschaften die zeitliche Präferenzen eines Objektes darstellen. Adaptiert dafür die Schnittstelle <i>SplusServer.ObjectWithTimePreferences</i> .
Resource	Stellt eine Klasse für Ressourcen dar. Leitet sich von der Klasse <i>BasisObjectWithTimePreferences</i> ab und erweitert diese um eine Liste, welche die Veranstaltungen einer Ressource enthält. Die Klasse <i>Resource</i> adaptiert die Schnittstelle <i>Resource</i> .
ResourceWithTimeConstraints	Stellt eine Basisklasse für Ressourcen mit zeitlichen Beschränkungen dar. Die Klasse leitet sich von der Klasse <i>Resource</i> ab und adaptiert die Schnittstelle <i>SplusServer.ResourceWithTimeConstraints</i> . Unter einer zeitlichen Beschränkung versteht man z.B. das Verfügbarkeitsmuster einer Ressource.

Tabelle 4: Basisklassen des Frameworks (eigene Darstellung)

Nachdem die Basisklassen sowie deren Eigenschaften und Methoden definiert wurden, wurden die entsprechenden Klassen zur Darstellung der Objekte des Lehrangebots entworfen. Die Klassen leiten sich entsprechend ihrer implementierten Schnittstellen in der Typbibliothek von einer der vier Basisklassen ab. Während die Klasse *Flagge* von der Basisklasse *BasisObject* erbt, leitet sich die Klasse *Dozent* von *ResourceWithTimeConstraint* ab. Die abstrakten Basisklassen können somit als Steckplätze angesehen werden, von denen andere Klassen entsprechend ihrer Schnittstelle erben können. Das folgende Code-Beispiel soll zeigen wie die Vererbung über die Adapter funktioniert:

```
// Adapter eines Dozenten, leitet sich von ResourceWithTimeConstraint ab
internal class Dozent : ResourceWithTimeConstraints {
    internal Dozent(SplusServer.StaffMember dozent)
        : base(dozent as SplusServer.ResourceWithTimeConstraints) {}
}
```


Das Code-Beispiel zeigt, dass die Klasse *Dozent* sich von der Basisklasse *ResourceWithTimeConstraints* ableitet. Durch eine Typumwandlung mit dem Operator *as* wird die Basisschnittstelle von *SplusServer.StaffMember* an den Adapter weitergeleitet, der die Basisklasse für die Klasse *Dozent* darstellt.

Durch diese Vorgehensweise wurde eine Implementierungsvererbung im Framework realisiert, mit der die Typen der Typbibliothek adaptiert werden. Die Implementierungsvererbung erhöht zudem die Erweiterbarkeit. Wenn eine Methode oder Eigenschaft vielen Objekten zur Verfügung stehen soll, muss dies nur über eine einmalige Erweiterung in einer der Basisklassen erfolgen.

Die Adapter sind im Framework alle durch das Schlüsselwort *internal* gekennzeichnet. Im nächsten Kapitel wird erläutert, dass die Implementation eines Objektes von seiner Konstruktion im Framework getrennt wird. Aus diesem Grund können die Adapter als *internal* gekennzeichnet werden und ein Nutzer des Frameworks arbeitet nur mit der Schnittstelle des entsprechenden Adapters.

4.1.3 Objekte

Im vorherigen Abschnitt wurde erläutert wie die Objekte des COM-Servers durch das Framework adaptiert werden. Dabei ging es also nur um die Struktur und die Zusammensetzung einzelner Objekte. Eine Anwendung die das Framework nutzt, soll aber auch konkrete Objekte erstellen und bearbeiten können.

Im folgenden Kapitel wird das Erstellen von Objekten der entsprechenden Adapter betrachtet. Es wird geklärt, wie das Framework die Vorgehensweise der COM-Schnittstelle abstrahiert und welche Vorteile dadurch für einen Entwickler entstehen.

Als erstes wird am Beispiel eines Dozenten erläutert, wie das Erstellen eines Objektes mit Hilfe der COM-Klasse *Application* funktioniert. Es wird davon ausgegangen, dass ein Objekt dieser Klasse mit dem Namen *application* erzeugt wurde. Der Quellcode dafür kann folgendermaßen aussehen:

```
// Dozent erstellen  
SplusServer.StaffMember dozent = application.ActiveCollege.StaffMembers[1];
```

Das Quellcodebeispiel verdeutlicht noch einmal, dass der Zugriff auf die Objekte über die Eigenschaft *ActiveCollege* erfolgt. Diese Eigenschaft stellt alle Objekte eines bestimmten Typs in einer Sammlung zur Verfügung. Dementsprechend ist es möglich, über einen Index auf einen Dozenten in der Sammlung aller Dozenten zuzugreifen.

Um einen bestimmten Datensatz abzurufen, bieten Listen der COM-Schnittstelle die Methode *Find*. Diese Methode sucht einen Datensatz an Hand des Primärschlüssels, des Namens und der ID innerhalb einer Sammlung. Um den Dozenten mit dem Namen *Müller* zu suchen und ein Objekt zu erstellen würde der Quellcode so aussehen:

```
// Dozent erstellen  
SplusServer.StaffMember dozent =  
    application.ActiveCollege.StaffMembers.Find("", "Müller", "")[1];
```

Man sieht, dass nur ein Parameter nötig ist um ein Objekt zu suchen. In diesem Beispiel handelt es sich um den Namen des zu erstellenden Objektes. Der erste Parameter der Methode *Find* stellt den Primärschlüssel dar und der dritte Parameter die ID eines Objektes.

Dabei ist nur die ID eines Objektes ein eindeutiger Bezeichner. Der Name sowie der Primärschlüssel müssen nicht zwangsläufig auf ein eindeutiges Objekt in der Sammlung verweisen. Aus diesem Grund liefert die Methode *Find* eine Sammlung an gefundenen Objekten. Auf das Suchergebnis wird ebenfalls über einen Index zugegriffen.

Diese Vorgehensweise bei der Erstellung von Objekten hat einige Nachteile.

Ein Fallstrick ist die Tatsache, dass der Startindex der Sammlungen 1 ist. Das nicht der übliche Startindex 0 genutzt wird kann zu unnötigen Programmierfehlern führen. Des Weiteren kann man die Eindeutigkeit bei der Erstellung eines Objektes nur mit Hilfe der Eigenschaft *Count* einer Sammlung überprüfen. Das folgende Beispiel zeigt die Abfrage des Suchergebnisses mit Hilfe dieser Eigenschaft:

```
// Dozenten erstellen
SplusServer.StaffMembers dozenten =
    application.ActiveCollege.StaffMembers.Find("", "Müller", "");

// Abfragen ob nur ein Element in der Sammlung vorhanden ist
if (dozenten.Count == 1) {
    SplusServer.StaffMember dozent = dozenten[1];
}
```

Das Beispiel soll verdeutlichen, dass die Erstellung eines eindeutigen Objektes schwierig zu realisieren ist. Wenn die Abfrage über die Anzahl der gefundenen Objekte nicht erfolgt, ist es nicht sicher, ob auch wirklich ein eindeutiges Objekt erstellt wurde. Zum einen können mehrere Suchergebnisse vorliegen und zum anderen gar keins.

Wenn durch die Methode *Find* kein Objekt erzeugt werden konnte und man das Suchergebnis über den Startindex abfragt, ist der Rückgabewert *null*. Objekte die dadurch mit *null* initialisiert werden, können nicht genutzt werden und erzeugen bei jeder Anfrage eine Ausnahme.

Insgesamt ist die Erstellung von Objekten direkt über die COM-Schnittstelle fehleranfällig und unsicher.

Aus den genannten Gründen wird die Erstellung von Objekten durch das Framework in eigenen Klassen gekapselt. Bei der Entwicklung einer Anwendung können dadurch leichter und komfortabler eindeutige Objekte erstellt werden.

Das Bereitstellen von Klassen, die der Erstellung von Objekten dienen, wird über das Erzeugungsmuster mit dem Namen Fabrikmethode realisiert. Dabei handelt es sich um ein klassenbasiertes Muster. Das Prinzip der Fabrikmethode ist es, eine abstrakte Klasse zu definieren, die bestimmte Operationen zum Erstellen eines Objektes enthält. Von dieser abstrakten Klasse leiten sich konkrete Klassen ab, die bestimmen welcher Objekttyp durch die abstrakte Klasse erzeugt werden soll (Entwurfsmuster, vgl. [GHJV95], Seite 131).

Abbildung 21 zeigt die Struktur der Fabrikmethode.

Abbildung 21: Musterstruktur der Fabrikmethode (Entwurfsmuster, vgl. [GHJV95], Seite 132)

Die Darstellung des Musters soll zeigen, dass ein Erzeuger eine Fabrikmethode definiert, mit welcher Objekte des Typs *Produkt* erzeugt werden. Ein *KonkreterErzeuger* überschreibt die Fabrikmethode und erzeugt ein Objekt vom Typ *KonkretesProdukt*. Im Falle des Frameworks könnte das Produkt zum Beispiel die Basisklasse *Resource* sein und das *KonkreteProdukt* ein *Dozent*.

Die in Abbildung 21 gekennzeichneten Fabrikmethoden werden im Framework über die Schablonenmethode aufgerufen. Dabei handelt es sich um ein klassenbasiertes Verhaltensmuster, dass es Unterklassen ermöglicht, bestimmte Schritte eines Algorithmus zu überschreiben (Entwurfsmuster, vgl. [GHJV95], Seite 366). Die Struktur des Musters wird auf Abbildung 22 dargestellt.

Abbildung 22: Musterstruktur der Schablonenmethode (Entwurfsmuster, vgl. [GHJV95], Seite 368)

Das Muster der Schablonenmethode zeigt, dass eine abstrakte Klasse eine Methode die als Schablone dient implementiert. Des Weiteren definiert die abstrakte Klasse primitive Operationen, die durch eine konkrete Klasse überschrieben werden müssen. Die Ausführung der primitiven Operationen erfolgt dann in der Schablonenmethode. Somit kann die konkrete Klasse Teile der Schablonenmethode überschreiben und anpassen.

Im Framework werden die beiden beschriebenen Muster eingesetzt. In der abstrakten Basisklasse *ObjectBuilder* wird die Fabrikmethode zum Erstellen von Objekten definiert. Zusätzlich wird in *ObjectBuilder* eine abstrakte Methode deklariert, die durch einen konkreten Erzeuger überschrieben werden muß.

Am Beispiel des konkreten Erzeugers für Dozenten *DozentBuilder* soll die Erstellung von Objekten auf der theoretischen Grundlage der beschriebenen Entwurfsmuster erklärt werden. Teile des Quellcodes der Klasse *ObjectBuilder*:

```
public abstract class ObjectBuilder<T> {  
  
    // Objektmenge  
    protected internal SplusServer.ObjectsWithName objects = null;  
  
    // Konstruktor  
    internal ObjectBuilder(SplusServer.ObjectsWithName objects) {  
        // Zuweisung  
        this.objects = objects;  
    }  
  
    // Fabrikmethode  
    public T Build(string objectId, string name, string primärschlüssel) {  
        try {  
            // Suchen  
            SplusServer.IObjectsWithName ergebnis =  
                this.objects.Find(primärschlüssel, name, objectId);  
            // Wenn genau ein Treffer vorliegt  
            if (ergebnis.Count == 1) {  
                // Konvertiertes Objekt  
                return this.Convert(ergebnis[1] as SplusServer.ObjectWithName);  
            }  
            else { // Ausnahme auslösen }  
        }  
        catch (COMException e) { // Ausnahme auslösen }  
    }  
  
    // Abstrakte Methode  
    internal abstract T Convert(SplusServer.ObjectWithName obj);  
}
```

Der Quellcode zeigt, dass eine konkrete Klasse die sich von der Klasse *ObjectBuilder* ableitet, die abstrakte Methode *Convert* überschreiben muss. Der generische Typparameter *T* bestimmt den Typ des zu erstellenden Objektes. Das Erstellen des Objektes erfolgt aus einer über den Konstruktor übergebenen Liste aller Objekte eines Typs.

Nun folgt der Quellcode der entsprechenden konkreten Klasse, in diesem Falle der Klasse *DozentBuilder*.

```
// DozentBuilder, leitet sich von ObjectBuilder ab
public class DozentBuilder : ObjectBuilder<IDozent> {

    // Konstruktor
    public BevorzugterBeginnBuilder()
        : base(application.ActiveCollege.StaffMembers as
        SplusServer.ObjectsWithName)
    { }

    // Konkretisiert die abstrakte Methode Convert.
    internal override IDozent Convert(SplusServer.ObjectsWithName obj) {
        return new Dozent(obj as SplusServer.StaffMember);
    }
}
```

Man soll an diesem Beispiel erkennen, dass die Klasse *DozentBuilder* als generischen Typparameter die Schnittstelle *IDozent* übergibt. Dieser Typparameter bestimmt die Rückgabewerte der Methoden *Build* und *Convert*. Die Methode *Convert* muss nun noch durch die Klasse überschrieben werden. Sie bekommt das Suchergebnis der Methode *Build* übergeben und wandelt es in den Typen *IDozent* um. Es soll deutlich werden, dass durch die konkreten Erzeugerklassen die in Kapitel 4.1.2 erläuterten Adapter erstellt werden. Ein Entwickler könnte nun den Datensatz des Dozenten *Müller* auf folgende Art und Weise erstellen:

```
DozentBuilder builder = new DozentBuilder();
IDozent dozent = builder.Build("", "Müller", "");
```

Wenn man die Methode *Build* noch einmal betrachtet, erkennt man, dass bei einem nicht eindeutigen Ergebnis Ausnahmen ausgelöst werden. Somit weiß ein Entwickler immer sofort, ob ein erstelltes Objekt eindeutig ist und kann Ausnahmen entsprechend abfangen.

Zusätzlich wird in der Klasse *ObjectBuilder* noch die Methode *Check* deklariert. Mit Hilfe dieser Methode kann die Existenz und Einmaligkeit eines Objektes vor der Erstellung getestet werden.

Insgesamt kann man also feststellen, dass die Erstellung von Objekten von der eigentlichen Implementation über Sammlungen für einen Entwickler nicht mehr erkennbar ist. Sie wird im Framework in der Klasse *ObjectBuilder* durch die Methode *Build* gekapselt. Diese Abstraktion erlaubt eine flexible und erweiterbare Erstellung von Objekten und trägt zu dem Ziel bei, die COM-Schnittstelle komplett zu adaptieren.

Neben dem Erstellen von Objekten soll das Framework auf das Erzeugen neuer Datensätze und das Löschen von Datensätzen ermöglichen. Zu diesem Zweck wurde die Klasse *ObjectEditor* erstellt. Diese Klasse leitet sich von *ObjectBuilder* ab und erweitert die Schnittstelle der Basisklasse um die in der Tabelle 5 aufgelisteten Methoden.

Name	Bedeutung
Create()	Erzeugt ein neues Objekt.
Delete()	Löscht ein vorhandenes Objekt.

Tabelle 5: Erweiterte Member der Klasse *ObjectEditor* (eigene Darstellung)

Beim Erzeugen eines neuen Objektes erwartet die Methode *Create* als Parameter den Namen des zu erzeugenden Objektes. Somit soll durch das Framework sichergestellt werden, dass keine Objekte mit denselben Namen in einer Objektmenge des gleichen Typs erzeugt werden können. Die beiden Methoden werden ebenfalls über das Verhaltensmuster der Schablonenmethode bereitgestellt.

Nachdem im letzten Kapitel die Objektimplementation und in diesem Abschnitt das Erstellen, Erzeugen und Löschen von Objekten besprochen wurde, kann man die Schlußfolgerung ziehen, dass für einen Typen immer eine Schnittstelle und eine Klasse vorliegt, die sich je nach Anforderung entweder von der Klasse *ObjectBuilder* oder *ObjectEditor* ableitet. Für einen Dozenten gibt es beispielsweise die Typen *IDozent* und *DozentEditor*.

Im Anlage A befindet sich eine Liste in der alle adaptierten Objekte und die entsprechende Erbauerklasse aufgelistet sind.

4.1.4 Listen

Die COM-Schnittstelle stellt Sammlungen für Objekte desselben Typs bereit. Tabelle 6 fasst die Methoden und Eigenschaften zusammen, die durch alle Listen der COM-Schnittstelle implementiert werden.

Name	Bedeutung
Add()	Fügt der Liste ein neues Element hinzu.
Remove()	Entfernt ein Element aus der Liste.
Find()	Sucht in der Liste Elemente und gibt das Ergebnis in Form einer Liste zurück.
GetEnumerator()	Ruft einen Enumerator ab, mit dem es möglich ist die Elemente der Auflistung zu durchlaufen.
Count	Ruft die Anzahl der Elemente ab, die sich in der Liste befinden.

Tabelle 6: Member die alle Listen der COM-Schnittstelle implementieren (eigene Darstellung)

Die COM-Schnittstelle stellt Listen auf zwei verschiedenen Ebenen zur Verfügung. Zum einen ist es möglich, alle Objekte eines Typs abzurufen. Das Abrufen aller Dozenten und das Durchlaufen der Elemente der Liste wird im folgenden Beispiel dargestellt:

```
// Liste mit allen Dozenten erstellen
SplusServer.StaffMembers dozenten = application.ActiveCollege.StaffMembers;
// Liste durchlaufen
foreach (SplusServer.StaffMember dozent in dozenten) {
    // Aktion durchführen
}
```

Zum anderen gibt es Listen die einem Objekt zugeordnet sind. Im folgenden Beispiel wird die Liste von Flaggen abgerufen und bearbeitet die einem Dozenten zugeordnet ist:

```
// Dozent erstellen
SplusServer.StaffMember dozent = application.ActiveCollege.StaffMembers[1];
// zugeordnete Flaggen abrufen
SplusServer.Tags flaggen = dozent.AssociatedTags;
// Flagge hinzufügen
flaggen.Add(/*Objekt vom Typ SplusServer.Tag*/);
// Flagge entfernen
flaggen.Remove(/*Objekt vom Typ SplusServer.Tag*/);
```


Nachdem beschrieben wurde wie der Zugriff auf Sammlungen der COM-Schnittstelle erfolgt und wie diese implementiert sind, werden nun Erfahrungen zum Umgang mit den Listen geschildert.

Die Verwaltung von Standorten erfolgt in S-PLUS auf zwei Ebenen. Einem Standort können auf der zweiten Ebene andere Standorte zugewiesen werden. Abbildung 23 zeigt die Standorte die dem Standort mit dem Namen *B* auf zweiter Ebene zugewiesen sind.

Abbildung 23: Verwaltung von Standorten in S-PLUS (Bildschirmkopie)

Ausgehend von den bisherigen Ausführungen sind das Hinzufügen und das Entfernen von Standorten auf der zweiten Ebene über die Methoden *Add* und *Remove* möglich. Bei der Ausführung des entsprechenden Quellcodes wird jedoch eine Ausnahme durch die COM-Schnittstelle erzeugt und die gewünschte Aktion führt zu keinem Erfolg. Um die Hierarchie der Standorte über die COM-Schnittstelle zu steuern, ist es vielmehr notwendig die erste Ebene eines Standortes zu ändern.

Dieses Beispiel soll zeigen, dass die von jeder Liste implementierten Methoden nicht immer das erwartete Ergebnis erzielen. Diese Erfahrungen sollen im Framework umgesetzt und dadurch sichergestellt werden, dass keine Methoden mehr aufgerufen werden die zu keinem Ergebnis und Ausnahmen führen.

Der zweite Aspekt der im Zusammenhang mit Listen angesprochen wird, ist die Entwicklung von Windows-Anwendungen. Eine häufige Anforderung einer Windows-Anwendung ist die Darstellung von Objektmengen in Steuerelementen. Beispielsweise sollen alle in S-PLUS verwalteten Dozenten in einer Auswahlliste eines Formulars präsentiert werden.

Die durch die Listen implementierten Methoden reichen nicht aus, um diese Anforderung zu erfüllen. Es werden keine Methoden implementiert die das Sortieren oder das Filtern einer Liste ermöglichen.

Das Framework kapselt den Zugriff auf die Listen der COM-Schnittstelle. Aus den beschriebenen Erfahrungen lassen sich folgende Ziele für die Implementierung der Listen durch das Framework ableiten:

- § es sollen nur Methoden implementiert werden, die das erwartete Ergebnis liefern;
- § durch das Kapseln des Zugriffs auf die Listen sollen Methoden, die das Sortieren und das Filtern erlauben implementiert werden.

Die Realisierung der gesetzten Zielstellung erfolgt durch zwei Klassen. Die erste Klasse hat den Namen *ListBuilder* und ermöglicht das Erstellen einer Liste. In Tabelle 7 werden die Methoden und Eigenschaften der Klasse dargestellt.

Name	Bedeutung
GetList()	Ruft die Objekte der aktuellen Liste in Form einer stark typisierten Liste vom Typ <i>List<T></i> ab. Diese Methode ist mehrmals überladen und ermöglicht es die eine Liste auf verschiedene Arten abzurufen.
ForEach()	Führt eine Aktion für alle aktuellen Objekte der aktuellen Auflistung durch. Dazu wird ein generischer Parameter vom Typ <i>System.Action<T></i> übergeben.
Contains()	Prüft die Existenz eines Objektes in der aktuellen Liste.
Count	Ruft die Anzahl der Elemente ab, die sich in der Liste befinden.

Tabelle 7: Member der Klasse *ListBuilder* (eigene Darstellung)

Die Aufgabe der Klasse *ListBuilder* ist es, die Implementierung der Listen in der COM-Schnittstelle zu ummanteln. Sie bietet nur Lesezugriff auf eine Liste und ermöglicht mit der Methode *GetList* die flexible Umwandlung von Listen der COM-Schnittstelle in eine Liste vom generischen .NET-Typen *List<T>*.

Die Bereitstellung aller Objekte eines Typs erfolgt im Framework über die Eigenschaft *Objects*, welche im entsprechenden *ObjectBuilder* eines Typs deklariert ist. Der folgende Quellcode zeigt die Erstellung einer Liste mit allen verwalteten Dozenten:

```
// Editor erstellen
DozentEditor editor = new DozentEditor();
// Liste erstellen
List<IDozent> dozenten = editor.Objects.GetList();
```

Der Quellcode zeigt noch einmal, dass die Klasse *ListBuilder* die Auflistung aller Dozenten kapselt und diese durch die Methode *GetList* in eine generische Liste umwandelt, die Objekte des entsprechenden Adapters enthält. In diesem Fall erfolgt also eine Konvertierung des Typen *SplusServer.StaffMembers* in den .NET Typen *List<IDozent>*.

Die Klasse *List<T>* stellt nun Methoden zum Suchen, Bearbeiten und Sortieren bereit. Sie implementiert zudem die Schnittstelle *IList<T>*, welche die Grundlage für das Binden von Daten an Steuerelemente in Windows-Anwendungen darstellt.

In Tabelle 7 wird erklärt, dass die Methode *GetList* mehrmals überladen ist. Das bedeutet, dass es mehrere Möglichkeiten gibt die Methode aufzurufen. Im folgenden Beispiel soll dargestellt werden, wie man eine Liste aus allen Dozenten abrufen deren Anfangsbuchstabe des Namens A lautet:

```
// Editor erstellen
DozentEditor editor = new DozentEditor();
// Liste erstellen
List<IDozent> dozenten = editor.Objects.GetList(new Predicate<IDozent>(Filter));

// Methode die der Signatur des Delegate System.Predicate<IDozent> entspricht
public static Boolean Filter(IDozent dozent) {
    if (dozent.Name.StartsWith("A")) return true;
    return false;
}
```

Man erkennt, dass die Methode *GetList* auch mit einem Delegate vom .NET-Typen *System.Predicate* aufgerufen werden kann, um schon bei der Erstellung der Liste eine Filterung der Objekte vorzunehmen.

Des Weiteren stellt die Klasse *ListBuilder* mit der Methode *ForEach* eine Möglichkeit zur Verfügung, eine Liste nicht erst erstellen zu müssen, sondern sofort eine angegebene Aktion mit den Objekten der Liste auszuführen. Das folgende Beispiel zeigt wie man die Namen aller verwalteten Dozenten in einer Konsole schreibt, ohne das zuerst eine Konvertierung erfolgt:

```
// Editor erstellen
DozentEditor editor = new DozentEditor();
// Liste durchlaufen
editor.Objects.ForEach(new Action<IDozent>(Aktion));

// Methode die der Signatur des Delegates System.Action<IDozent> entspricht
public static void Aktion(IDozent dozent) {
    Console.WriteLine(dozent.Name);
}
```

Diese Beispiele zeigen den Umgang mit der Klasse *ListBuilder* und sollen verdeutlichen, dass es verschiedene Möglichkeiten gibt, die Listen der COM-Schnittstelle über die Klasse abzurufen oder zu durchlaufen.

Wie schon erwähnt, bietet diese Klasse nur Lesezugriff auf die gekapselten Listen. Somit ist die Klasse für Listen der COM-Schnittstelle geeignet, die nicht bearbeitet werden sollen. Dazu zählt die bereits besprochene Verwaltung von Standorten auf zweiter Ebene. Um die Standorte abzurufen die einem Standort zugewiesen sind, könnte der Code zum folgendermaßen lauten:

```
// StandortEditor erstellen
StandortEditor editor = new StandortEditor();
// Standort
IStandort standort = editor.Build("", "B", "");
// 2. Ebene
List<IStandort> ebene2 = standort.Ebene2.GetList();
```

Alle Listen der COM-Schnittstelle die nur Lesezugriff bieten, werden durch die Klasse *ListBuilder* gekapselt. Über die Schnittstelle *IListBuilder<T>* werden die Methoden und Eigenschaften der Klasse zur Verfügung gestellt.

Im zweiten Schritt war es notwendig eine Implementierung zu erstellen die das Bearbeiten von Listen der COM-Schnittstelle ermöglicht. Aus diesem Grund wurde eine Klasse mit dem Namen *ListEditor* entwickelt.

Die Klasse *ListEditor* leitet sich von der Klasse *ListBuilder* ab und erweitert sie um zwei Methoden. Die beiden Methoden werden in der Tabelle 8 dargestellt.

Name	Bedeutung
Add()	Fügt der Liste ein neues Element hinzu.
Remove()	Entfernt ein Element aus der Liste.

Tabelle 8: Erweiterte Member der Klasse *ListEditor* (eigene Darstellung)

Diese beiden Methoden beeinflussen die Listen der COM-Schnittstelle direkt. Das folgende Beispiel zeigt das Hinzufügen einer Flagge zur Liste von Flaggen eines Dozenten:

```
// DozentEditor erstellen
DozentEditor editor = new DozentEditor();
// Dozent
IDozent dozent = editor.Build("", "Müller", "");
// Flagge hinzufügen
dozent.Flaggen.Add(/*Objekt vom Typ IFlagge*/);
```

Alle Listen der COM-Schnittstelle deren Bearbeitung möglich ist, werden durch die Klasse *ListEditor* gekapselt. In diesem Beispiel ist die Eigenschaft *Flaggen* des Dozenten vom Typ *ICollection<IFlagge>*.

Durch diese beiden Klassen soll eine genaue Trennung der Möglichkeiten erfolgen, die ein Entwickler hat. Zum einen gibt es Listen die nur ausgelesen werden können und zum anderen Listen die bearbeitet werden können.

4.1.5 Ereignisse

Die COM-Schnittstelle stellt für alle Aktionen Ereignisse bereit. Diese informieren über Änderungen von Werten eines Objektes oder über den Erfolg einer ausgeführten Methode. Bei jeder Aktion werden zwei Ereignisse ausgelöst. Das erste tritt vor der Aktion ein und das zweite danach. Der folgende Code soll am Beispiel der Änderung des Namens eines Dozenten zeigen wie die COM-Schnittstelle Ereignisse bereitstellt:

```
// Erstellter Dozent
SplusServer.StaffMember dozent;
// Verbindung zu Ereignissen herstellen
dozent.BeforeModify += new
SplusServer.IStaffMemberEvents_BeforeModifyEventHandler(dozent_BeforeModify);
dozent.AfterModify += new
SplusServer.IStaffMemberEvents_AfterModifyEventHandler(dozent_AfterModify);
// Namen ändern
dozent.Name = "Mustermann";

// Ereignissenken
void dozent_AfterModify(string PropertyName) {
    // Code...
}
void dozent_BeforeModify(string PropertyName) {
    // Code...
}
```

Der Code zeigt exemplarisch die Vorgehensweise zum Empfangen von Ereignissen der COM-Schnittstelle. Es wird eine Verbindung zu einem Ereignis hergestellt, das vor dem Ändern des Namens eintritt und dafür eine entsprechende Ereignissenke erstellt. Parallel dazu wird das Ereignis initialisiert, das nach dem Ändern ausgelöst wird. Die Ereignissenken bekommen über den Parameter *PropertyName* den Namen der zu ändernden oder geänderten Eigenschaft übergeben (in diesem Falle die Eigenschaft *Name*).

Der entscheidende Punkt ist, dass für alle Ereignisse eigene Delegates bei der Konvertierung der Typlibibliothek erzeugt wurden. Im obigen Beispiel sind das die Delegates *IStaffMemberEvents_BeforeModifyEventHandler* und *IStaffMemberEvents_AfterModifyEventHandler*. Somit enthält die Typlibibliothek sehr viele Delegates welche die gleiche Methode darstellen. Die Signatur dieser Methode im obigen Beispiel ist:

```
void MethodName(string PropertyName);
```

Bei anderen Aktionen, wie dem Sichern des Images, werden noch andere Parameter für die Ereignissenke erwartet. Alles in allem ist aber festzustellen, dass die meisten Delegates der COM-Schnittstelle überflüssig wären.

Aus diesem Grund wird das Senden von Ereignissen durch das Framework verändert. Alle notwendigen Typen sind im Namespace *Splus.Framework.Event* zu finden. Hier befinden sich die Delegates und Klassen die Daten von Ereignissen zusammenfassen. Klassen die Ereignisdaten bereitstellen werden üblicherweise mit dem Präfix *EventArgs* versehen.

Der prinzipielle Aufbau beim Senden der Ereignisse wird durch das Framework übernommen. Das Framework sendet also auch jeweils zwei Ereignisse für eine Aktion. Für Ereignisse welche jedoch die gleiche Signatur der Ereignissenke haben steht nur noch ein Delegate zur Verfügung. Die Implementierung der Delegates ist also nicht mehr an ein Objekt gebunden, sondern an die ausgeführte Aktion. Zusätzlich werden die Daten der Ereignisse in Klassen zur Verfügung gestellt. Das folgende Beispiel zeigt wie die Änderung des Namens eines Dozenten und das Empfangen der Ereignisse mit dem Framework realisiert wird:

```
using Splus.Framework.Event;

// Erstellter Dozent
IDozent dozent;
// Verbindung zu Ereignissen herstellen
dozent.BeforeUpdate += new BeforeUpdateObjectEventHandler(dozent_BeforeUpdate);
dozent.AfterUpdate += new AfterUpdateObjectEventHandler(dozent_AfterUpdate);
// Namen ändern
dozent.Name = "Mustermann";
// Ereignissenken
void dozent_BeforeUpdate(IBasisObject sender, BeforeUpdateEventArgs args) {
    // Code...
}
void dozent_AfterUpdate(IBasisObject sender, AfterUpdateEventArgs args) {
    // Code...
}
```

Der Quellcode soll verdeutlichen, dass die Delegates *BeforeUpdateObjectEventHandler* und *AfterUpdateObjectEventHandler* allgemein für alle Objekte bei Eigenschaftsänderungen gelten. Somit wird die Anzahl der Delegates der COM-Schnittstelle durch das Framework von mehreren hundert auf 10 bis 20 reduziert.

4.1.6 Installation

Das Framework wird als installierbares Paket bereitgestellt. Durch die Weitergabe des Projekts durch ein Installationspaket soll eine möglichst einfache Installation und Bereitstellung des Projektes erfolgen.

Das Bereitstellungsprojekt sowie die installierbare Datei sind auf der CD in Anlage B.

Nachdem ein Anwender die Installation abgeschlossen hat, sind im gewählten Zielordner der Installation die auf Abbildung 24 gezeigten Dateien vorhanden.

Abbildung 24: Installiertes Framework (Bildschirmkopie)

Nach der Installation des Frameworks stehen einem Entwickler die Klassenbibliothek in Form des Assemblys *Splus.Framework.dll* sowie die konvertierte Typbibliothek der COM-Schnittstelle (*SplusServer.dll*) zur Verfügung.

Des Weiteren befinden sich im Installationsverzeichnis die Anwendungen *Administration* und *ConfigFileBuilder* sowie die Assemblys *Splus.Framework.Management.dll* und *Splus.Framework.Utilities.dll*. Im Unterverzeichnis *config* befindet sich zusätzlich eine Binärdatei mit dem Namen *TimetablerConfig.dat*.

Welche Bedeutung den zusätzlich installierten Dateien zukommt wird im nächsten Kapitel beschrieben.

4.1.7 Anwendungen

Nachdem in den letzten Abschnitten erläutert wurde wie der Zugriff auf das Framework funktioniert und das Framework installiert wird, folgt nun eine allgemeine Beschreibung zur Erstellung von Anwendungen für das Framework.

Bisher wurde nur betrachtet, dass eine Anwendung Daten des Veranstaltungsplaners lesen, bearbeiten oder löschen kann. Des Weiteren muss es aber noch die Möglichkeit geben, bestimmte Einstellungen für eine Anwendung festzulegen. Dabei handelt es sich um Optionen die für jeden erstellen Client je nach Anforderung der Anwendung unterschiedlich sein können.

Die Konfiguration wurde bisher durch XML-Dateien realisiert. In Zukunft wird dies über serialisierte Objekte geschehen. Im Assembly *Splus.Framework.Management.dll* sind die entscheidenden Typen definiert. Sie definieren die Struktur einer Option oder genereller Konfigurationseinstellungen. Durch Serialisierung ist es möglich Objektzustände von Typen dauerhaft als Binärdatei zu speichern. Nach der Installation des Frameworks befindet sich im Unterverzeichnis *config* die Binärdatei *TimetablerConfig.dat*. In dieser Datei werden alle vorgenommenen Einstellungen gespeichert. Das Serialisieren und das Deserialisieren der Datei wird mit der Klasse *Serializer*, die im Assembly *Splus.Framework.Utilities.dll* definiert ist, vorgenommen.

Um eine Anwendung für das Framework zu erstellen, muss ein Entwickler zunächst eine Konfigurationsdatei erstellen. Dies wird mit der Anwendung *ConfigFileBuilder* ermöglicht. Auf Abbildung 25 wird die Windows-Anwendung dargestellt.

Abbildung 25: Benutzeroberfläche *ConfigFileBuilder* (Bildschirmkopie)

Die Anwendung *ConfigFileBuilder* kann Konfigurationsdateien erstellen und bearbeiten. Dafür sind in der oberen Menüleiste der Anwendung die entsprechenden Buttons zum Speichern, Öffnen und Neu anlegen vorhanden. Parallel werden diese Funktionen über den Menüpunkt *Datei* im Hauptmenü zur Verfügung gestellt.

Zu einer Konfigurationsdatei gehören zwei Elemente. Das erste ist der Name der Anwendung. Die Angabe des Namens ist zwingend erforderlich, weil eine Anwendung sich mit ihrem Namen gegenüber dem Framework identifiziert. Das zweite Element sind die Optionen. Es ist nicht erforderlich Optionen zu definieren.

Im unteren Bereich der Anwendung *ConfigFileBuilder* befindet sich die Verwaltung der Optionen einer Konfigurationsdatei. Es ist möglich Optionen hinzuzufügen, zu löschen oder zu bearbeiten. Auf der linken Seite werden alle Optionen der aktuellen Konfigurationsdatei aufgelistet. Nach der Auswahl einer Option wird auf der rechten Seite eine Vorschau der gewählten Option gezeigt.

Es gibt 6 verschiedene Arten von Optionen die angelegt werden können:

- § Zeichenfolgen;
- § Zahlen;
- § Datumsangaben;
- § Ja/Nein Werte;
- § einfache Auswahl aus einer Liste;
- § mehrfache Auswahl aus einer Liste.

In einem Beispiel soll eine Konfigurationsdatei mit dem Namen Test erstellt werden. Es werden drei Optionen erstellt. Die 1. Option ist eine Zeichenfolge, die 2. Option eine Datumsangabe und bei der 3. Option handelt es sich um eine Mehrfachauswahl. In Tabelle 9 werden die wichtigsten Angaben zu den Optionen zusammengefasst.

ID	Name	Typ	Standardwert	Optionen
option1	Zeichenfolge	String		
option2	Datum	Date	01.01.2008	
option3	Auswahlliste	MultiSelect	Wert2	Wert1;Wert2;Wert3

Tabelle 9: Beispielooptionen einer Konfigurationsdatei (eigene Darstellung)

Das Anlegen oder Bearbeiten einer Option erfolgt in einem gesonderten Formular. Dieses Formular ist über die Buttons Neu und Bearbeiten zu öffnen. Auf Abbildung 26 wird dieses Formular gezeigt.

TFH - Wildau, Veranstaltungsplaner

Option
Erstellen oder Bearbeiten einer Option.

ID
An Hand dieser ID kann der Wert der Option in der Anwendung bestimmt werden.

Name
Der eingebene Wert bestimmt den in Formularen angezeigten Namen.

Beschreibung
Machen Sie nähere Angaben zu der Option.

Statisch ☐ Statischer Wert (Konstante)

Optionstyp
Der Typ bestimmt wie eine Option angezeigt und ausgelesen wird.

Standardwert

Minimalwert
Bestimmt den kleinsten Ganzzahl-Wert den diese Option annehmen darf.

Maximalwert
Bestimmt den größten Ganzzahl-Wert den diese Option annehmen darf.

Minimalwert
Bestimmt das früheste Datum das diese Option annehmen darf.

Maximalwert
Bestimmt das späteste Datum das diese Option annehmen darf.

Optionen
Möglichkeiten für Auswahlliste. Diese werden mit einem Semikolon getrennt.

Abbildung 26: Formular zum Erstellen einer Option (Bildschirmkopie)

Auf Abbildung 26 wird die dritte Option der Testdatei angelegt.

Im oberen Abschnitt des Formulars werden die ID und der Name einer Option definiert. Wenn eine Option mit der angegebenen ID in der aktuellen Konfigurationsdatei bereits existiert oder eine Option ohne Name angelegt werden soll, wird dies durch eine Fehlermeldung kenntlich gemacht.

Des Weiteren kann im oberen Bereich eine Beschreibung der Option erfolgen und festgelegt werden, ob die aktuelle Option statisch ist.

Als nächstes wird der Optionstyp festgelegt. Hier hat der Anwender die Wahl zwischen String (Zeichenfolge), Int (Zahl), Date (Datumsangabe), Bool (Ja/Nein-Wert), Select (Einfachauswahl) und MultiSelect (Mehrfachauswahl).

Die Wahl eines Typs beeinflusst dynamisch die Darstellung des Formulars. Es werden das Steuerelement für den Standardwert sowie der untere Bereich des Formulars an die Wahl des Typs angepasst. Im gezeigten Beispiel auf Abbildung 26 wird eine Liste für die Mehrfachauswahl generiert, welche als Auswahlmöglichkeiten die Werte Wert1, Wert2 und Wert3 hat. Wert2 wird als Standardwert festgelegt. Die Trennung oder Separation von Werten für die Auswahlmöglichkeiten bzw. des Standardwertes erfolgt durch das Semikolon.

Nachdem alle drei Optionen der Testdatei auf diese Weise angelegt wurden, werden die Einstellungen der Testdatei wie auf Abbildung 27 dargestellt.

Abbildung 27: Angelegte Konfigurationsdatei (Bildschirmkopie)

Nachdem ein Entwickler die Konfigurationsdatei entworfen hat, muss er sein Projekt durch die Datei beim Framework anmelden. Dafür steht die bei der Installation mitgelieferte Anwendung *Administration* zur Verfügung. Auf Abbildung 28 wird die Oberfläche der Windows-Anwendung dargestellt.

Abbildung 28: Administrationsoberfläche des Frameworks (Bildschirmkopie)

Mit der Administrationsoberfläche können die registrierten Anwendungen des Frameworks bearbeitet werden. Über die beiden ersten Buttons in der Menüleiste werden Anwendungen registriert und entfernt. Der dritte Button dient zum Einstellen der ProgID. Abbildung 29 zeigt das Einstellen der ProgID über ein gesondertes Formular.

Abbildung 29: Einstellen der ProgID (Bildschirmkopie)

Um das Beispiel mit der Testdatei fortzuführen wird diese registriert. Bei der Registrierung einer Konfigurationsdatei werden die entsprechenden Werte in die Datei *Timetable-Config.dat* geschrieben. Beim Löschen werden diese wieder entfernt.

Nach einer erfolgreichen Registrierung erfolgt die Darstellung der Testdatei wie auf Abbildung 30 gezeigt.

Abbildung 30: Registrierte Konfigurationsdatei (Bildschirmkopie)

Die Konfigurationsdatei wird mit ihrem Namen in die linke Auflistung eingetragen. Auf der rechten Seite werden die in der Datei hinterlegten Optionen angezeigt. Man erkennt, dass die in Tabelle 9 beschriebenen Optionen der Testdatei angezeigt werden.

Ein Benutzer kann die Werte der Optionen ändern und über den Button *Übernehmen* abspeichern.

Auf dieser Basis können also beliebig viele Anwendungen die das Framework zum Zugriff auf die COM-Schnittstelle nutzen mit ihrem Namen und Optionen registriert werden. Der Benutzer kann die Optionen über die Administrationsoberfläche steuern.

Nachdem ein Entwickler die Konfigurationsdatei seiner Anwendung registriert hat, kann er das Framework nutzen.

In Kapitel 4.1.1 wurde beschrieben, dass der Start des COM-Servers über die Methode *Start* der Klasse *SPLUS* erfolgt. Diese Methode erwartet als Parameter den Namen der Anwendung. Um den COM-Server für die Testanwendung zu starten, lautet der Aufruf der Methode *Start* folgendermaßen:

```
SPLUS.Start("Test");
```

Es wird deutlich, dass sich der Entwickler nicht darum kümmern muss, mit welcher ProgID der COM-Server gestartet werden soll. Der Wert der ProgID wird wie beschrieben durch den Benutzer in der Administrationsoberfläche eingestellt. Wie nach dem Start des COM-Servers auf die Datenobjekte von S-PLUS über das Framework zugegriffen wird, wurde in den letzten Kapiteln ausführlich beschrieben.

Interessant ist jetzt noch der Zugriff auf die Optionen einer Anwendung. Im Namespace *Splus.Framework* befindet sich die statische Klasse *Options*. Die Klasse hat die in Tabelle 10 aufgelisteten Methoden.

Name	Bedeutung
GetStringValue()	Ruft den Wert einer Option ab, die eine Zeichenfolge darstellt.
GetDateTimeValue()	Ruft den Wert einer Option ab, die eine Datumsangabe darstellt.
GetIntValue()	Ruft den Wert einer Option ab, die eine Zahl darstellt.
GetBooleanValue()	Ruft den Wert einer Option ab, die einen Ja/Nein-Wert darstellt.
GetListValue()	Ruft den Wert einer Option ab, die eine Auswahl aus einer Liste darstellt.

Tabelle 10: Methoden der Klasse *Options* (eigene Darstellung)

Die 5 Methoden bieten Zugriff auf den Wert einer Option der Anwendung. Der Wert wird mit Hilfe der ID der Option abgerufen. Der Rückgabewert der Methoden ist auf die Optionstypen ausgerichtet. Somit kann jeder Wert einer Option in der Konfigurationsdatei über eine der Methoden abgerufen werden.

Die Testanwendung kann die Werte ihrer Optionen folgendermaßen abrufen:

```
// Option1 abrufen
String option1 = Options.GetStringValue("option1");
// Option2 abrufen
DateTime option2 = Options.GetDateTimeValue("option2");
// Option3 - Auswahl aus Liste abrufen
List<String> option3 = Options.GetListValue("option3");
```

Auf den letzten Seiten wurde beschrieben, wie ein Entwickler eine Anwendung für das Framework erstellt. Über eine Konfigurationsdatei meldet sich die entwickelte Anwendung beim Framework an. Die Optionen für die Anwendung können dann über das Framework abgerufen werden.

Der Grund für diesen Aufbau ist die Weitergabe von Anwendungen für das Framework. Wenn ein Entwickler seine Anwendung fertig gestellt hat, möchte er diese natürlich an eine Reihe von Anwendern weiter geben. Aus Anwendersicht muss ebenfalls das Framework installiert sein. Nun erhält ein Anwender die entwickelte Komponente. Mit Hilfe der Konfigurationsdatei kann er die Anwendung registrieren und administrieren. Mehr hat der Anwender nicht zu tun. Somit wird eine einfache Art der Weitergabe von Anwendungen für das Framework realisiert.

4.2 Webpräsentationskomponente

Die Webpräsentationskomponente ist eine Entwicklung von Prof. Dr. Christian Müller und Jürgen Lübeck. Die Anwendung basiert auf dem Automationsserver von S-PLUS und wurde vor der Entwicklung des Frameworks an der Technischen Fachhochschule Wildau eingesetzt. Dementsprechend gab es die Anforderung, die Webpräsentationskomponente in Zukunft mit Hilfe der Funktionalitäten des Frameworks zu benutzen.

Im folgenden Kapitel werden die Idee und das Konzept der Webpräsentationskomponente beschrieben. Des Weiteren wird auf die Anpassung der Anwendung an das Framework eingegangen.

4.2.1 Idee

Die Technische Fachhochschule Wildau präsentiert auf ihren Internetseiten die Stundenpläne für Dozenten, Studenten und Räume. Abbildung 31 zeigt die Startseite der veröffentlichten Stundenpläne.

Abbildung 31: Startseite für Stundenpläne (Bildschirmkopie, vgl. [TFHWPLAN])

Im unteren Bereich der gezeigten Webseite befinden sich drei Links. Diese führen zu den verschiedenen publizierten Stundenplänen an der Technischen Fachhochschule Wildau.

Um die Darstellung der veröffentlichten Stundenpläne für Studenten, Dozenten oder Räume zu zeigen, wird exemplarisch der Link *Studenten* gewählt. Abbildung 32 zeigt eine Auswahl der aufgelisteten Stundenpläne für Studenten.

Abbildung 32: Aufgelistete Stundenpläne für Studenten (Bildschirmkopie, vgl. [TFHWPLANSTU])

Die Stundenpläne werden für Studenten in Studenten-Set-Gruppen zusammengefasst. Für jede Studenten-Set-Gruppe wird eine Zeile mit Informationen und Links zu den Stundenplänen in der Auflistung gezeigt.

Eine Studenten-Set-Gruppe stellt eine Seminargruppe dar, die in einem bestimmten Jahr das Studium aufgenommen hat. Zum Beispiel weist der Name B1-04 darauf hin, dass die Studenten-Set-Gruppe im Jahr 2004 mit dem Betriebswirtschaftsstudium begonnen hat und die Seminargruppe 1 des Jahrgangs ist. Die zweite Seminargruppe dieses Jahrgangs heißt dementsprechend B2-04.

Jede Studenten-Set-Gruppe fasst nun eine oder mehrere zusammengehörige Studenten-Sets zusammen. Die Stundenpläne der Studenten-Sets einer Gruppe können sich in bestimmten Punkten noch unterscheiden. Beispielsweise muss zwischen Wahlpflichtfächern unterschieden werden oder die Gruppe auf verschiedene Räume aufgeteilt werden.

Parallel dazu verläuft die Veröffentlichung der Stundenpläne für Dozenten und Räume. Räume werden ähnlich wie Studenten-Set-Gruppen in Raum-Gruppen unterteilt. Dagegen wird für jeden Dozenten einzeln der Stundenplan publiziert.

Auf Abbildung 33 wird zusätzlich deutlich, dass eine Unterteilung der Stundenpläne in *Lehrveranstaltungen*, *Prüfungen* und *alle Veranstaltungen* erfolgt. Es können somit Pläne nur mit Lehrveranstaltungen oder nur mit Prüfungen abgerufen werden. Unter dem Link *alle Veranstaltungen* werden alle Veranstaltungen zusammengefasst. Neben Prüfungen und Lehrveranstaltungen können dazu z.B. auch Klausureinsichten gehören.

Abbildung 33 soll exemplarisch alle Veranstaltungen des Stundenplans der Studenten-Set-Gruppe BB-05 darstellen.

Veranstaltungsübersicht für StudentSetGruppe: BB-05
Biosystemtechnik/Bioinformatik (Bachelor) Jahrgang 2005
Sommersemester 2008
Semesterzeit vom: 01.03.2008 - 31.08.2008
Vorlesungszeit vom: 15.03.2008 - 28.06.2008
Plan erstellt am: 23.04.2008

Bei Fragen wenden Sie sich bitte an den [Stundenplaner](#).

Tag	(erstes) Datum	Beginn	Dauer [in min]	wann / Zeitmuster	Typ	Name	Dozenten	Räume	StudentenSets	letzte Änderung
Montag	17.03.2008	09:45	90	14täg.gerade	L	BioInf 4	Beyerlein Wang	13-136 Lab	BB/05 Set/01	13.03.2008
Montag	17.03.2008	11:30	90	14täg.gerade	L	BioInf 4	Beyerlein Wang	13-136 Lab	BB/05 Set/01	13.03.2008
Montag	17.03.2008	15:45	90	14täg.gerade	L	BioInf 4	Beyerlein Wang	13-136 Lab	BB/05 Set/01	13.03.2008
Dienstag	18.03.2008	08:00	90	wöchentlich	L	BioProzTech	Stein, F. Wildenauer	15-143/4 Lab	BB/05	06.03.2008
Dienstag	18.03.2008	09:45	90	wöchentlich	L	BioProzTech	Stein, F. Wildenauer	15-143/4 Lab	BB/05	06.03.2008
Dienstag	18.03.2008	11:30	90	wöchentlich	L	BioProzTech	Stein, F. Wildenauer	15-143/4 Lab	BB/05	06.03.2008
Dienstag	18.03.2008	14:00	90	wöchentlich	L	BioProzTech	Stein, F. Wildenauer	15-143/4 Lab	BB/05	06.03.2008
Donnerstag	20.03.2008	09:45	90	wöchentlich	L	MikroSystTech	Edel Foitzik Stein, F.	15-048 Lab	BB/05	06.03.2008
Donnerstag	20.03.2008	11:30	90	wöchentlich	L	MikroSystTech	Edel Foitzik Stein, F.	15-048 Lab	BB/05	06.03.2008
Montag	24.03.2008	09:45	90	14täg.ungeradeKW	L	BioInf 4	Beyerlein Wang	13-136 Lab	BB/05 Set/02	13.03.2008
Montag	24.03.2008	11:30	90	14täg.ungeradeKW	L	BioInf 4	Beyerlein Wang	13-136 Lab	BB/05 Set/02	13.03.2008
Montag	24.03.2008	15:45	90	14täg.ungeradeKW	L	BioInf 4	Beyerlein Wang	13-136 Lab	BB/05 Set/02	13.03.2008
Mittwoch	07.05.2008	09:45	180	19.KW	Prüfung 3	BioHybTech	Lisdat	15-022, 023 Lab	BB/05	23.04.2008

Anmerkung: Seit 2005 wird der Stundenplan mit dem Ressourcenplanungssystem S-Plus erstellt. Die Webpräsentation wurde von Prof. Dr. Christian Müller und Jürgen Lübeck entwickelt.

Abbildung 33: Alle Veranstaltungen für BB-05 (Bildschirmkopie, vgl. [TFHWPLANBB05])

Auf Abbildung 33 ist zu erkennen, dass die Veranstaltungen in zeitlich geordneter Form vorliegen. Des Weiteren liefert der Stundenplan Informationen über den Typ, den Namen, den Beginn und die Dauer der Veranstaltung. Alle an einer Veranstaltung beteiligten Dozenten, Studenten-Sets und Räume werden ebenfalls mit aufgelistet.

Auf den letzten Seiten wurde die Darstellung der Stundenpläne auf den Webseiten der Technischen Fachhochschule Wildau erläutert. Daraus ergibt sich die Frage, wie die Pflege und Wartung der veröffentlichten Stundenpläne erfolgt. Es ist eine mühsame Arbeit alle Stundenpläne per Hand zu erstellen und eventuelle Änderungen eines Planes auf den Webseiten nachzuvollziehen. Hier setzt die Webpräsentationskomponente an. Sie soll die Arbeit der Mitarbeiter an der Technischen Fachhochschule Wildau erleichtern und ermöglicht das automatische Generieren der Stundenpläne.

Abbildung 34 fasst grob die Arbeitsabläufe zur Veröffentlichung der Stundenpläne mit Hilfe der Webpräsentationskomponente zusammen.

Abbildung 34: Arbeitsablauf zur Stundenplanerstellung (eigene Darstellung)

Die Idee hinter der Webpräsentationskomponente ist das automatische Generierung von Stundenplänen auf Basis der in S-PLUS verwalteten Daten. Nach dem Auslesen der Daten über die COM-Schnittstelle kann die Webpräsentationskomponente alle nötigen Informationen verarbeiten und in Dateien zusammenfassen. Diese Dateien können dann auf den Server der Technischen Fachhochschule Wildau geladen und nun über den beschriebenen Weg durch Studenten, Dozenten oder Mitarbeitern abgerufen werden.

Dieser Vorgang kann beispielsweise für jedes neue Semester durchgeführt werden oder auch bei Änderungen des Plans innerhalb eines Semesters. Somit bietet die Webpräsentationskomponente eine schnelle und effiziente Möglichkeit auf Änderungen des Stundenplans zu reagieren und eine zeitnahe bzw. korrekte Veröffentlichung zu gewährleisten.

4.2.2 Umsetzung

Nachdem die Idee der Webpräsentationskomponente erläutert wurde, soll nun auf die technische Umsetzung eingegangen werden.

Die Webpräsentationskomponente ist eine Konsolenanwendung, die in C# für das .NET-Framework entwickelt wurde.

Im ersten Schritt liest die Anwendung die nötigen Daten zur Stundenplanerstellung über den Automationsserver von S-PLUS aus. Dabei handelt es sich um Dozenten, Studenten-Set-Gruppen und Raum-Gruppen.

In der folgenden Phase generiert die Anwendung die Stundenpläne. Für jedes Objekt werden drei Stundenpläne erzeugt und jeweils in einer XML-Datei gespeichert. Wie in Abbildung 33 dargestellt wird beispielsweise für jeden Dozenten ein Stundenplan für Lehrveranstaltungen, einer für Prüfungen und einer mit allen Veranstaltungen generiert.

Die im XML-Format erzeugten Pläne werden in einer bestimmten Verzeichnisstruktur erzeugt, welche auf Abbildung 35 dargestellt wird.

Abbildung 35: Erstellte Verzeichnisstruktur der Webpräsentationskomponente (Bildschirmkopie)

Alle Objekte eines Typs werden in einem Verzeichnis zusammengefasst. Abbildung 35 zeigt nicht alle Unterverzeichnisse auf der dritten Ebene. Es werden nur beispielhaft einige Unterverzeichnisse für *Dozenten*, *Räume* und *Studenten* gezeigt.

Für jedes Objekt wird ein Unterverzeichnis angelegt, welches sich aus dem Namen sowie der *ObjectId* zusammensetzt. In diesem Verzeichnis befinden sich die drei erstellten Stundenpläne des Objektes.

Zusätzlich zu den Stundenplänen der einzelnen Objekte wird für Dozenten, Studenten-Set-Gruppen und Raum-Gruppen eine Index-Datei im entsprechenden Ordner erzeugt. Diese XML-Datei liefert eine Übersicht über alle sich im Verzeichnis befindenden Objekte.

Nachdem die Stundenpläne erstellt wurden, können sie auf den Webserver der Technischen Fachhochschule Wildau geladen werden. Die generierten XML-Dateien sind mit XSLT-Stylesheets verknüpft, die sich auf dem Server befinden. Die Stylesheets erzeugen aus den XML-Dateien die optische Darstellung der Stundenpläne auf den Webseiten. Die Index-Dateien der Objekttypen werden wie in Abbildung 32 gezeigt in einem Browser dargestellt. Die Stylesheets bilden die Stundenpläne wie in Abbildung 33 gezeigt ab.

Die Konsolenanwendung wird über Konfigurationseinstellungen gesteuert. Mit Hilfe der vorgenommenen Einstellungen lassen sich folgende Bereiche der Anwendung steuern:

- § **Verzeichnisstruktur** – Ein Anwender hat die Möglichkeit zu bestimmen, in welchen Verzeichnissen und Unterverzeichnissen die generierten Stundenpläne abgelegt werden sollen.
- § **Stylesheets** – Der Benutzer kann über den Speicherort bestimmen auf welches Stylesheet eine XML-Datei verweist.
- § **Veröffentlichungseinschränkungen** – Über bestimmte Einstellungen kann ein Anwender die veröffentlichten Stundenpläne einschränken. Es ist beispielsweise möglich den Zeitraum einzugrenzen, in dem eine Veranstaltungen liegen muss um veröffentlicht zu werden.
- § **Veranstaltungsarten** – Um die drei verschiedenen Stundenpläne für ein Objekt erzeugen zu können, muss ein Anwender mit Hilfe der Art einer Veranstaltung festlegen, ob es sich um eine Prüfung oder eine Lehrveranstaltung handelt.

Das Steuern der Anwendung fand bisher über eine Konfigurationsdatei im XML-Format statt. Wie das in Zukunft mit dem Framework realisiert wird, erläutert der folgende Abschnitt.

4.2.3 Anpassung

Es wurde bereits erläutert, dass die Webpräsentationskomponente an das im Rahmen der Diplomarbeit erstellte Framework angepasst werden musste. In folgendem Abschnitt werden die wichtigsten Schritte bei der Anpassung besprochen.

In der ersten Phase wurde die Konfigurationsdatei der Webpräsentationskomponente erstellt. Die vorherige Variante mit einer Steuerung der Anwendung über eine XML-Datei wurde durch das neue, in Kapitel 4.1.7 beschriebene, Konzept ersetzt.

Abbildung 36 zeigt die Konfigurationsdatei der Webpräsentationskomponente in der Administrationsanwendung.

Abbildung 36: Registrierte Webpräsentationskomponente (Bildschirmkopie)

Nach der Registrierung der Anwendung für das Framework ist es nicht mehr nötig, die Konfigurationseinstellungen über eine XML-Datei einzulesen. Somit mussten die Stellen im Quellcode für das Auslesen der Einstellungen angepasst werden. Insgesamt konnte dadurch Quellcode eingespart werden.

Im nächsten Schritt wurde die Erstellung des COM-Servers überarbeitet. Diese Aufgabe übernimmt jetzt das Framework und muss nicht mehr durch eine Anwendung durchgeführt werden. Das folgende Quellcodebeispiel zeigt, wie man den COM-Server für die Webpräsentationskomponente startet:

```
try {  
    SPLUS.Start("Webpräsentation");  
}  
catch (Exception e) {  
    Console.WriteLine(e.Message);  
    Console.WriteLine("Die aktuelle Anwendung wird abgebrochen. Drücken Sie  
eine  
Taste.");  
    Console.ReadLine();  
    Environment.Exit(1);  
}
```

Im Quellcode ist zu sehen, dass sich die Anwendung nur mit Ihrem Namen (Webpräsentation) beim Framework über die Methode *Start* der Klasse *SPLUS* anmeldet. Wie bereits beschrieben, wird dadurch der COM-Server über das Framework gestartet. In der alten Variante musste das Starten des COM-Servers in der Anwendung selber geschehen.

Als letztes musste der Objektzugriff angepasst werden. Bevor die Webpräsentationskomponente an das Framework angepasst wurde, wurden in der Anwendung bereits mehrere Typen definiert, welche die gleichen Ziele verfolgten wie die Objektadapter des Frameworks. Es waren also Objekte vorhanden, die den Zugriff auf Dozenten oder Studenten-Set-Gruppen kapseln. Das Nutzen dieser Typen ist durch den Einsatz des Frameworks nicht mehr nötig. Somit werden in der Webpräsentationskomponente nur noch Typen definiert, die speziell der Ausführung dieser Anwendung gelten.

Die Logik zur Erstellung des Dateisystems und der Stundenpläne wurde übernommen und nicht verändert.

Insgesamt werden durch das Einbinden des Frameworks einige Vorteile erzielt:

- § Starten des COM-Servers wird durch das Framework übernommen;
- § bequemes Erzeugen und Auslesen von Konfigurationseinstellungen;
- § Konzentration auf die Entwicklung der entscheidenden Typen für eine spezielle Anwendung (Dateisystem, Stundenpläne im XML-Format).

4.3 Prüfungsplaner

Die Entwicklung einer Windows-Anwendung, mit der es möglich ist die Prüfungen an der Technischen Fachhochschule Wildau zu planen ist das Hauptziel der Diplomarbeit. In folgendem Abschnitt wird auf die Realisierung und Problemstellungen und deren Lösungen eingegangen. Des Weiteren werden Problematiken angesprochen die nicht zu bewältigen waren.

4.3.1 Konzept und Anforderungen

Das Hauptanliegen des Prüfungsplaners besteht in der Vereinfachung der Planung und Verwaltung von Prüfungen.

Zur Verwaltung von Prüfungen zählen das Erstellen, das Bearbeiten und das Löschen einer Prüfung. Um diese Anforderung umzusetzen, bestand der erste Schritt in der Analyse der notwendigen Objekte und Eigenschaften die einer Prüfung zugewiesen werden müssen. Weil es um eine Vereinfachung der Erstellung von Prüfungen geht, wurde sich nur auf die notwendigsten Zuweisungen und Angaben konzentriert.

Dozenten

Dozenten stellen eine Ressource für eine Prüfung dar. Es ist damit nicht möglich, eine Prüfung ohne die Angabe eines oder mehrerer Dozenten zu planen. Aus diesem Grund muss es im Prüfungsplaner möglich sein, Dozenten zuzuweisen und Zuweisungen zu bearbeiten.

Räume

Ein Raum stellt den Ort für eine Veranstaltung oder Prüfung bereit. Bei Räumen handelt es sich um Ressourcen für eine Prüfung und somit kann eine Prüfung nicht ohne die Zuweisung eines Raumes erstellt werden. Im Prüfungsplaner muss es also möglich sein, diese Zuordnung zu vollziehen.

Studenten-Sets

Ein Studenten-Set stellt die kleinste Gruppe von logisch zusammengefassten Studenten dar. Die Studenten eines Studenten-Sets haben die gleichen Lehrveranstaltungen und Prüfungen. Eine Prüfung kann nicht ohne die Zuweisung eines Studenten-Sets geplant werden. Es ist also notwendig, dass im Prüfungsplaner die Möglichkeit besteht diese Zuweisung vorzunehmen oder zu bearbeiten.

Neben den Zuweisungen von Ressourcen ist es notwendig, eine Prüfung in den Stundenplan einzuordnen. Dazu muss eine Zuweisung zu einem Semester sowie zu einem Modul erfolgen. Auf Abbildung 37 wird die Struktur dargestellt, in die eine Prüfung eingeordnet werden muss.

Abbildung 37: Struktur für Studiengänge in S-PLUS (Handbuch S-PLUS, vgl. [HB350])

Auf Abbildung 37 wird am Beispiel eines Architektur-Studienganges die in S-PLUS genutzte Struktur zur Abbildung der Studienstruktur dargestellt. Es ist zu erkennen, dass ein Semester die höchste Instanz in der Hierarchie darstellt und einen Studiengang beschreibt. Einem Semester sind verschiedene Module zugeordnet. Ein Modul stellt wiederum eine Kombination aus Veranstaltungen bereit (vgl. [HB350]).

Durch den Prüfungsplaner muss also die Zuordnung einer Prüfung zu einem Modul gewährleistet werden. Die Zuweisung zu einem Semester oder zu Semestern erfolgt dann automatisch über das gewählte Modul.

Die Zuweisung von Studenten-Sets erfolgt auch mit Hilfe des Moduls. Ein Studenten-Set kann zu einem oder mehreren Modulen zugeordnet werden. Somit wird der Stundenplan eines Studenten-Sets über die Module erstellt. Der Prüfungsplaner soll in diesem Zusammenhang die Zuweisung von bestimmten Studenten-Sets zu einer Prüfung über das gewählte Modul realisieren. Weitere Details und Probleme zu diesem Aspekt sind in Kapitel 4.3.6 zu finden.

Neben den beschriebenen Zuweisungen sollen als allgemeine Eigenschaften einer Prüfung zusätzlich der Name und die Veranstaltungsart angegeben bzw. bearbeitet werden können.

Nachdem die Zuweisungen feststehen, ist es notwendig zu analysieren wie eine Prüfung geplant wird. Der Unterschied zwischen dem Erstellen und der Planung einer Prüfung besteht darin, dass beim Erstellen nur der reine Datensatz angelegt wird und durch die Planung die entsprechende Veranstaltung durch S-PLUS in den Stundenplan eingefügt wird.

Beim Planen einer Veranstaltung sind vor allem die in S-PLUS definierten und definierbaren Beschränkungen zu beachten. Zu den Beschränkungen gehört beispielsweise das Verfügbarkeitsmuster einer Ressource. Das Verfügbarkeitsmuster stellt die Verfügbarkeit über den gesamten Planungszeitraum dar. Andere Beschränkungen sind beispielsweise die freien Tage die ein Dozent pro Woche haben muss, die Wegezeiten zwischen verschiedenen Standorten oder das Ressourcen des gleichen Typs nicht gleichzeitig geplant werden dürfen (für weitere Informationen siehe Handbuch S-PLUS, Erstellen des Veranstaltungsplans, vgl. [HB350]).

Eine Veranstaltung kann nicht geplant werden, sobald eine Beschränkung verletzt wird. Wenn zum Beispiel ein Dozent an einem Tag und zu einer Uhrzeit mit einer Veranstaltung geplant werden soll und er hat an diesem Termin schon eine andere Veranstaltung, ist es nicht möglich diese Veranstaltung zu planen.

Es ist jedoch möglich über bestimmte Einstellungen Beschränkungen zu umgehen. Welche Auswirkungen das auf den Prüfungsplaner hat, wird in Kapitel 4.3.6 beschrieben.

Besonders wichtig ist hierbei, dass der Prüfungsplaner beim Planen einer Prüfung erkennt, ob die Prüfung ordnungsgemäß geplant wurde. Wenn bestimmte Beschränkungen verletzt wurden, muss dies dem Anwender deutlich gemacht werden. Dem Anwender muss die Möglichkeit gegeben werden, auf verletzte Beschränkungen zu reagieren und die Prüfung erneut zu planen.

Wie eine Prüfung durch einen Anwender geplant werden kann wird in Abschnitt 4.3.6 erläutert.

4.3.2 Konfiguration

Um die Anwendung zu benutzen, ist es wie bei der Stundenplanausgabe notwendig eine Konfigurationsdatei anzumelden. Diese beinhaltet eine Option, mit der bestimmt wird, welche Veranstaltungsart als Prüfung gelten. Somit werden alle Veranstaltungen zu der diese Veranstaltungsart zugewiesen ist als Prüfung angesehen. Abbildung 38 zeigt diese Einstellungsmöglichkeit mit Hilfe der Administrationsanwendung.

Abbildung 38: Konfigurationseinstellungen des Prüfungsplaners

Diese Vorgehensweise fördert die Übersichtlichkeit der dargestellten Prüfungen durch den Prüfungsplaner, im Gegensatz zu S-PLUS. Es werden die durch den Benutzer über die Veranstaltungsart eingestellten Veranstaltungen gefiltert und angezeigt.

Die auswählbaren Veranstaltungsarten basieren auf dem Image für das Sommersemester 2007. Bei Änderungen oder Neuerungen in der Struktur der Veranstaltungsarten muss natürlich auch die Konfigurationsdatei angepasst und neu für das Framework angemeldet werden. Dies ist jedoch durch die beschriebenen Anwendungen im Kapitel 4.1.7 schnell und einfach zu erledigen.

4.3.3 Verwaltung

Nachdem die Konfigurationsdatei angemeldet wurde und die entsprechenden Einstellungen vorgenommen wurden, kann der Prüfungsplaner benutzt werden.

Nach dem Start des Prüfungsplaners sieht der Anwender das in Abbildung 39 abgebildete Fenster. Hier wird der Fortschritt des Ladevorgangs von Daten für den Prüfungsplaner grafisch dargestellt.

Abbildung 39: Daten laden für den Prüfungsplaner

In Kapitel 4.1.4 wurde die Erstellung von Listen erläutert. Das findet hier seine Anwendung. Folgende Listen werden für die Anwendung beim Laden erstellt:

- § **Dozenten** – Es werden alle im Image hinterlegten Mitarbeiter geladen.
- § **Räume** – Alle im Image gespeicherten Räume werden für die Anwendung in einer Liste hinterlegt.
- § **Studenten-Sets** – Alle Sets werden für die Anwendung geladen.
- § **Raum-Gruppen** – In einer Raum-Gruppe können Räume zusammengefasst werden. Alle Raum-Gruppen stehen in der Anwendung in einer Liste zur Verfügung.
- § **Studenten-Set-Gruppen** – StudentenSets können in Gruppen eingeteilt werden. Alle im Image hinterlegten StudentenSet-Gruppen werden geladen.
- § **Semester** – Alle Semester werden für die Anwendung geladen.
- § **Module** – Alle Module werden in einer Liste für die Anwendung hinterlegt.
- § **Verfügbarkeitsmuster** – Es werden nur die Verfügbarkeitsmuster geladen, die genau eine Woche im gesamten Planungszeitraum für eine Prüfung als verfügbar kennzeichnen.
- § **Prüfungen** – Es werden nur die Veranstaltungen geladen, die in der Konfiguration über die Veranstaltungsart als Prüfung gekennzeichnet wurden.

Das Laden der Daten erfolgt vor allem aus Gründen der Performance der Anwendung. Um größere Datenmengen in Steuerelementen einer Windows-Anwendung zu nutzen, werden diese von vornherein geladen. Wenn diese bei jeder Anforderung des Anwenders neu geladen werden müssten, würde die Anwendung ständig längere Pausen für das Laden einlegen müssen.

Nachdem das Laden abgeschlossen wurde, erhält der Benutzer zunächst die in Abbildung 40 dargestellte Ansicht auf alle Prüfungen.

Abbildung 40: Hauptfenster der Prüfungsverwaltung (Bildschirmkopie)

Abbildung 40 zeigt, dass im unteren Teil der Anwendung links die Auflistung aller geladenen Prüfungen erfolgt. Diese Liste kann der Anwender durchlaufen und erhält auf der rechten Seite die entsprechenden Informationen zu einer Prüfung. Dazu zählen neben dem Namen und der Veranstaltungsart die zugewiesenen Dozenten, Räume und Studenten-Sets. Des Weiteren bekommt der Anwender Informationen über den Planungsstatus der Prüfung. Es wird gezeigt ob die Prüfung bereits geplant wurde und wenn ja zu welchem Termin.

Über der Auflistung der Prüfungen stehen verschiedene Optionen zur Verfügung. Die ersten 3 Buttons ermöglichen das Anlegen, das Bearbeiten und das Löschen einer Prüfung. Auf diese Möglichkeiten wird im nächsten Kapitel eingegangen.

Als nächstes folgen ein Button zum Suchen (*Lupe*) und daneben ein Button der eine vorgenommene Filterung von Prüfungen wieder aufhebt. Wenn man den Button zum Suchen oder Filtern von Prüfungen drückt, erhält ein Anwender den auf Abbildung 41 dargestellten Dialog.

Abbildung 41: Filtern und Suchen von Prüfungen (Bildschirmkopie)

Wie im auf Abbildung 41 dargestellten Fall würden Anwender nun alle Prüfungen für den Dozenten *Bachmann* im Hauptfenster angezeigt bekommen.

Abbildung 42 soll noch einmal verdeutlichen was die Suche bewirkt hat. Es werden nur noch die Prüfungen des Dozenten *Bachmann* angezeigt. Dies wird über eine Meldung unterhalb der Auflistung deutlich gemacht (*Suchergebnis für Dozent: Bachmann*).

Abbildung 42: Ausgabe des Suchergebnisses (Bildschirmkopie)

Wie man im Vergleich zu Abbildung 40 erkennt ist auch der Button zum rückgängig machen der Suche aktiviert (rechts neben der Lupe). Bei Betätigung dieses Buttons werden wieder alle Prüfungen angezeigt.

Die Suche ist neben den Dozenten auch für Räume, Raum-Gruppen, Studenten-Sets, Studenten-Set-Gruppen, Module und Semester möglich.

In Kapitel 1.1 wurde erläutert, dass S-PLUS an der Technischen Fachhochschule Wildau als Netzwerkversion verwendet wird. Aus diesem Grund soll der Prüfungsplaner folgende Funktionalitäten anbieten:

- § Verbindung zur Datenbank herstellen;
- § Verbindung mit der Datenbank trennen;
- § Daten des aktuellen Images durch die Daten der Datenbank auffrischen;
- § Änderungen von Daten im Image zur Datenbank zurückschreiben.

Das Framework kapselt diese Funktionalitäten in der Klasse *Database*, die sich im Namespace *Splus.Framework* befindet. Der Prüfungsplaner setzt diese Anforderung mit dieser Klasse um. Die auf Abbildung 43 dargestellten Buttons, ermöglichen einem Nutzer den Zugriff auf die Datenbank zu steuern.

Abbildung 43: Buttons zur Datenbanksteuerung (Bildschirmkopie)

Der erste Button verbindet das aktuelle Image mit der Datenbank. Wenn das Verbinden mit der Datenbank funktioniert hat, werden die restlichen drei Buttons aktiviert. Auf Abbildung 44 wird dieser Zustand gezeigt.

Abbildung 44: Aktivierte Datenbankkontrolle (Bildschirmkopie)

Nach der Aktivierung kann der Benutzer mit dem zweiten Button die Verbindung zur Datenbank wieder beenden. Der dritte Button ermöglicht es die Daten des Images durch die Datenbank aufzufrischen. Wenn dieser Vorgang erfolgreich abgeschlossen wurde, erscheint der in Abbildung 39 dargestellte Ladebildschirm. Wie beschrieben werden die Daten für den Prüfungsplaner in Listen zwischengespeichert. Aus diesem Grund müssen diese Daten nach einer Auffrischung auch neu geladen werden. Der vierte Button schreibt die aktuellen Daten des Images an die Datenbank zurück.

Der Erfolg des Auffrischens bzw. Zurückschreibens der Daten wird dem Benutzer über Erfolgs- bzw. Fehlermeldungen transparent gemacht.

Die beschriebenen Funktionalitäten können parallel über den Menüpunkt *Datenbank* im Hauptmenü des Prüfungsplaners genutzt werden.

Der rechte Button des Menüs, der durch eine Diskette dargestellt wird, ermöglicht es das aktuelle Image zu speichern.

Das Speichern des Images führte in der Entwicklung zu Problemen. Wenn man die Methode *SaveImage* der COM-Schnittstelle über das Framework aufruft, wird das Image in den angegebenen Pfad gespeichert. Nach dem Speichern wird S-PLUS jedoch automatisch neu gestartet. Somit geht der Zugriff auf den COM-Server im Framework verloren. Aus diesem Grund wird der Prüfungsplaner nach dem Speichern des Images neu gestartet.

4.3.4 Prüfung löschen

Auf der linken Seite des Menüs bietet der Prüfungsplaner die Möglichkeit eine Prüfung zu löschen, eine neue anzulegen und eine vorhandene zu bearbeiten. Auf Abbildung 45 werden die entsprechenden drei Buttons dargestellt.

Abbildung 45: Buttons zum Anlegen, Bearbeiten und Löschen (Bildschirmkopie)

Durch den rechten Button auf der Abbildung 45 wird die aktuell in der Liste gewählte Prüfung gelöscht. Vor dem Löschen wird der Benutzer über eine Meldung gefragt, ob er die entsprechende Prüfung wirklich löschen will.

4.3.5 Prüfung bearbeiten

Mit dem auf Abbildung 45 in der Mitte dargestellten Button werden Prüfungen bearbeitet. Die Daten der in der Auswahlliste gewählten Prüfung werden in einem Formular zur Bearbeitung bereitgestellt. Wie dieses Formular aussieht und wie Daten einer Prüfung bearbeitet werden, wird im nächsten Abschnitt erläutert.

4.3.6 Prüfung anlegen und planen

Nachdem ein Benutzer den Button zum Anlegen einer Prüfung gewählt hat, erscheint das auf Abbildung 46 dargestellte Formular.

The screenshot shows a software window titled "TFH - Wildau, Prüfungsplaner". Inside, there's a sub-header "Prüfung" with the subtitle "Erstellen und Bearbeiten von Prüfungen". Below this are two tabs: "Daten" (selected) and "Planung". The "Daten" tab contains several input fields and buttons:

- Name:** A text input field.
- Typ:** A dropdown menu currently showing "Prüfung".
- Dozenten:** A large text area for listing lecturers, with "Hinzufügen..." and "Entfernen" buttons to its right.
- Räume:** A large text area for listing rooms, with "Hinzufügen..." and "Entfernen" buttons to its right.
- Kapazität:** A text field containing the value "0".
- Modul:** A text input field with an "Ändern..." button to its right.
- Studenten:** A large text area for listing students, with "Hinzufügen..." and "Entfernen" buttons to its right.

At the bottom right of the form area is a "Speichern" (Save) button. At the bottom center of the window is an "Abbrechen" (Cancel) button.

Abbildung 46: Prüfung anlegen (Bildschirmkopie)

Der erste Schritt beim Anlegen einer Prüfung ist die Angabe der wichtigsten Daten. Dazu gehören der Name und der Typ der Prüfung. Als nächstes müssen die Ressourcen der Prüfung bestimmt werden. Zu den Ressourcen zählen Dozenten, Räume und Studenten-Sets.

Für jede Ressource steht ein Button zur Verfügung, der es ermöglicht Objekte hinzuzufügen und einer der Objekte entfernt.

Am Beispiel der Dozenten soll gezeigt werden, wie ein Dozent zu einer Prüfung hinzugefügt wird. Durch Bestätigen des Buttons *Hinzufügen* neben der Liste mit den Dozenten wird das auf Abbildung 47 gezeigte Formular aufgerufen.

Abbildung 47: Auswahl von Dozenten (Bildschirmkopie)

Das auf Abbildung 47 dargestellte Formular ermöglicht es dem Benutzer, Dozenten aus einer Liste auszuwählen und der aktuellen Prüfung hinzuzufügen. Über der Liste befinden sich Suchfelder, die es erlauben die angezeigten Dozenten in der Liste einzuschränken. Es ist zum Beispiel möglich, nur die Dozenten anzuzeigen die zu einer bestimmten Dozenten-Gruppe zugeordnet sind. Des Weiteren besteht die Möglichkeit nur Dozenten anzuzeigen, die zu einem bestimmten Zeitpunkt zur Verfügung stehen. Die Verfügbarkeit eines Dozenten wird durch sein Verfügbarkeitsmuster bestimmt und ob der Dozent zum angegebenen Zeitpunkt keiner anderen Veranstaltung zugeordnet ist.

Für Räume und StudentenSets verläuft die Auswahl auf dieselbe Art und Weise.

Auf Abbildung 48 wird das Formular zum Anlegen von Prüfungen dargestellt, in dem alle Datenfelder ausgefüllt sind.

TFH - Wildau, Prüfungsplaner

Prüfung
Erstellen und Bearbeiten von Prüfungen

Daten Planung

Name: Testprüfung

Typ: Prüfung 2

Dozenten: Adam, Adler

Hinzufügen... Entfernen

Räume: 100-001

Hinzufügen... Entfernen

Kapazität: 49

Modul: B_VwL 1

Ändern...

Studenten: B1/06, B2/06

Hinzufügen... Entfernen

Speichern

Abbrechen

Abbildung 48: Ausgefülltes Datenformular einer Prüfung (Bildschirmkopie)

Über den Button *Entfernen* besteht für jede Ressource nun die Möglichkeit das gewählte Element aus der entsprechenden Liste zu entfernen.

Das Feld *Kapazität* informiert den Benutzer über die aktuelle Kapazität aller gewählten Räume.

Das Verwalten der Studenten-Sets, die an einer Prüfung teilnehmen erfolgt über ein Modul. Ein Modul besteht aus mehreren Studenten-Sets, aus denen sich der Anwender die teilnehmenden Studenten-Sets aussuchen kann.

Das Hinzufügen der Studenten-Sets über ein Modul ist allerdings in einigen Fällen zu unflexibel. Wenn zum Beispiel eine Prüfung angelegt werden soll, die für das vorherige Semester gilt, kann es vorkommen, dass zwar das Modul vorhanden ist, diesem aber nicht die Studenten-Sets zugeordnet sind. Aus diesem Grund kann ein Benutzer auch Studenten-Sets zuordnen, die nicht dem aktuell gewählten Modul zugeordnet sind. Beim Speichern der Prüfung wird dann die Zuordnung der Studenten-Sets zum Modul automatisch übernommen.

Nachdem alle Datenfelder ausgefüllt wurden, kann die Prüfung mit dem Button *Speichern* gespeichert werden. Wenn die Prüfung erfolgreich angelegt wurde, kann sie geplant werden. Auf Abbildung 49 wird das Formular zum Planen einer Prüfung dargestellt.

TFH - Wildau, Prüfungsplaner

Prüfung
Erstellen und Bearbeiten von Prüfungen

Daten Planung

Geplant Nein

Woche 12.KW 19.03.2007 - 24.03.2007

Dauer 15

Vorgeschlagener Termin

Tag Montag

Uhrzeit 08:00:00

Planen mit 53 ☒ Reale Größe

Aus dem Plan nehmen Planen Planen Ohne Uhrzeit

Abbrechen

Abbildung 49: Prüfung planen (Bildschirmkopie)

Wie bereits erwähnt versteht man unter der Planung, dass die Prüfung in Stundenplan der Technischen Fachhochschule Wildau eingefügt wird.

Für eine Prüfung gibt es dementsprechend zwei Zustände. Der erste Zustand ist eine geplante Prüfung und der zweite eine ungeplante Prüfung. Nachdem eine Prüfung neu angelegt wurde befindet sie sich im ungeplanten Zustand. Über die obere Box im Formular erhält ein Benutzer Informationen über den Planungszustand einer Prüfung.

Im ersten Schritt legt der Benutzer fest, in welcher Woche die Prüfung stattfinden soll. Das Aussuchen des Datums der Prüfung sollte eigentlich über ein Datumsfeld erfolgen, das durch den Semesterstart und das Semesterende begrenzt wird. Dieses Vorhaben konnte allerdings nicht realisiert werden.

Um zu verstehen wie das Datum einer Prüfung geplant wird, muss man sich die Verfügbarkeitsmuster in S-PLUS anschauen. Ein Verfügbarkeitsmuster stellt die Verfügbarkeit eines Objektes im Semesterzeitraum dar. Unter anderem erfolgt eine Einteilung der Verfügbarkeit in Wochen. Eine Veranstaltung die sich im Laufe des Semesters wiederholt, hat dementsprechend mehrere verfügbare Wochen im Semester. Prüfungen hingegen dürfen nur für eine Woche geplant werden. Um eine Prüfung in einer bestimmten Woche zu planen, muss also ein entsprechendes Verfügbarkeitsmuster durch S-PLUS verwaltet werden. Leider war es auch nach mehrmaligen und zeitaufwendigen Versuchen nicht möglich ein Verfügbarkeitsmuster mit diesen Anforderungen über die COM-Schnittstelle anzulegen.

Wenn die Angabe des Datums über eine Datumsfeld erfolgen würde, kann es vorkommen, dass eine Woche durch den Benutzer gewählt wird, die zu keinem entsprechenden Verfügbarkeitsmuster passt und somit die Prüfung nicht geplant werden kann. Aus diesem Grund werden für den Prüfungsplaner alle Verfügbarkeitsmuster geladen, die genau eine Woche als verfügbar im Semesterzeitraum markieren. Diese Muster werden mit ihrem Namen in der obersten ComboBox des Formulars zur Auswahl gestellt. Neben der Box werden Details über den Start und das Ende der durch das Verfügbarkeitsmuster dargestellten Woche angezeigt.

Damit stehen dem Benutzer alle Wochen zur Auswahl, die in S-PLUS über die entsprechenden Verfügbarkeitsmuster für eine Prüfung geplant werden können.

Neben der Woche kann der Benutzer die Dauer der Prüfung einstellen.

Als nächstes kann ein Termin für die Prüfung festgelegt werden, der sich innerhalb der gewählten Woche befindet. Dazu zählen der Wochentag und eine Uhrzeit.

Alle dargestellten Daten zur Uhrzeit, zum Wochentag oder zur Dauer beziehen sich auf die Angaben aus S-PLUS. In S-PLUS werden für das aktuelle Semester festgelegt, welche Wochentage gültig sind, wie lange ein Arbeitstag ist oder aus wie vielen Minuten eine Zeitperiode besteht. Mit diesen Angaben und im Framework bereitgestellten Berechnungen wird das Formular an das aktuelle Image angepasst.

Der letzte Punkt, der durch den Benutzer für die Planung der Prüfung angegeben wird, ist die Größe der Veranstaltung. Unter der Größe versteht man die Anzahl der teilnehmenden Studenten. Bei der Größe einer Prüfung wird zwischen der "realen Größe" und der "geplanten Größe" unterschieden. Die "reale Größe" spiegelt die Anzahl aller Studenten der gewählten Studenten-Sets wieder. Wenn aber zum Beispiel eine dritte Prüfung in einem Fach stattfindet, nehmen nicht mehr alle Studenten teil. Aus diesem Grund kann der Anwender die *geplante Größe* angeben.

Wenn alle Angaben für die Planung gemacht wurden, kann mit dem Button *Planen* die Prüfung geplant werden. Das Planen kann durch verschiedenste Gründe fehlschlagen. Es können zum Beispiel gewählte Ressourcen zum gewünschten Zeitpunkt nicht zur Verfügung stehen oder die Größe der Prüfung ist höher als die Kapazität der gewählten Räume. In diesen Fällen erhält der Benutzer eine Meldung über die Gründe einer fehlgeschlagene Planung. Auf Abbildung 50 wird beispielhaft eine Fehlerausgabe gezeigt.

Abbildung 50: Fehlermeldung bei der Planung einer Prüfung (Bildschirmkopie)

Abbildung 50 zeigt das die Studenten-Sets B1/06 und B2/06 zum gewünschten Termin nicht zur Verfügung stehen und dementsprechend nicht geplant werden können.

Neben dem Button *Planen* befindet sich der Button *Planen Ohne Uhrzeit*. Dieser Button ermöglicht es die Planung einer Prüfung unabhängig von einer angegebenen Uhrzeit zu vollziehen. In diesem Fall plant der Prüfungsplaner die Prüfung am erstmöglichen Zeitpunkt des angegebenen Tages.

Wenn eine Prüfung erfolgreich geplant wurde, verändern sich die Steuerelemente des Formulars. Auf Abbildung 51 wird dieser Punkt verdeutlicht.

The screenshot shows a software window titled "TFH - Wildau, Prüfungsplaner". Inside, there's a sub-header "Prüfung" with the subtitle "Erstellen und Bearbeiten von Prüfungen". Below this are two tabs: "Daten" and "Planung", with "Planung" being the active tab. The main area contains a form with the following fields and controls:

- A status bar at the top of the form says "Geplant Ja - 19.03.2007 19:15:00 Uhr, 15 Minuten".
- A "Woche" field with a dropdown menu showing "12.KW" and a date range "19.03.2007 - 24.03.2007".
- A "Dauer" field with a spinner box set to "15".
- A section titled "Vorgeschlagener Termin" containing:
  - A "Tag" dropdown menu set to "Montag".
  - An "Uhrzeit" dropdown menu set to "19:15:00".
- A "Planen mit" field with a spinner box set to "53" and a checked checkbox labeled "Reale Größe".
- At the bottom of the form are three buttons: "Aus dem Plan nehmen", "Planen", and "Planen Ohne Uhrzeit". The "Aus dem Plan nehmen" button is highlighted with a blue border.
- At the bottom right of the window is an "Abbrechen" button.

Abbildung 51: Erfolgreich geplante Prüfung (Bildschirmkopie)

Nach einer erfolgreich geplanten Prüfung ist ein erneutes Planen nicht möglich. Die entsprechenden Steuerelemente des Formulars sind deaktiviert. Um eine Prüfung neu zu planen muss sie erst durch den Button *Aus dem Plan nehmen* in einen ungeplanten Zustand gesetzt werden.

Die Ansicht auf die Daten der Prüfung verändert sich ebenfalls nach einer erfolgten Planung. Auf Abbildung 52 wird dieser Effekt verdeutlicht.

The screenshot shows a software window titled "TFH - Wildau, Prüfungsplaner" with a sub-header "Prüfung" and the description "Erstellen und Bearbeiten von Prüfungen". The window has two tabs: "Daten" (selected) and "Planung".

Under the "Daten" tab, the following fields and controls are visible:

- Name:** A text input field containing "Testprüfung".
- Typ:** A dropdown menu showing "Prüfung 2".
- Dozenten:** A list box containing "Abraham" and "Abel". To the right are buttons "Hinzufügen..." and "Entfernen".
- Räume:** A list box containing "100-001". To the right are buttons "Hinzufügen..." and "Entfernen".
- Kapazität:** A text input field containing "49".
- Modul:** A text input field containing "B_VwL 1". To the right is a button "Ändern...".
- Studenten:** A list box containing "B2/06" and "B1/06". To the right are buttons "Hinzufügen..." and "Entfernen".

At the bottom right of the window are two buttons: "Speichern" and "Abbrechen".

Abbildung 52: Datenansicht einer geplanten Prüfung (Bildschirmkopie)

Abbildung 52 zeigt, dass bei einer geplanten Prüfung die Ressourcen nicht bearbeitet werden können. Nur der Namen und der Typ der Prüfung können verändert werden. Um die Ressourcen zu verändern muss die Prüfung erst wieder aus dem Plan genommen werden. Danach kann zum Beispiel ein Dozent hinzugefügt und die Prüfung neu geplant werden.

Die Ausführungen auf den letzten Seiten sollen verdeutlichen, dass Daten und Planungsvariablen abhängig vom Planungsstatus der Prüfung eingestellt werden können.

Ein Aspekt bei der Planung einer Prüfung soll auf der folgenden Seite noch besprochen werden.

Dabei geht es um das bewusste Überbuchen von Ressourcen. Zum Beispiel sollte es ermöglicht werden, zwei Prüfungen zur gleichen Zeit und im gleichen Raum für verschiedene Studenten-Sets zu planen. In diesem Falle gibt der Prüfungsplaner eine Fehlermeldung aus die besagt, dass der Raum zur angegebenen Zeit nicht zur Verfügung steht. In diesem Moment sollte dem Benutzer die Wahl erlaubt werden die Prüfung trotzdem zu planen.

S-PLUS stellt die auf Abbildung 53 dargestellten Optionen zum Beeinflussen der Planung von doppelt gebuchten oder nicht verfügbaren Ressourcen bereit.

Abbildung 53: Planungsbeschränkungen bearbeiten (Bildschirmkopie)

Über die CheckBoxen unter *BucheDoppelt* kann ein Benutzer von S-PLUS einstellen, welche Ressourcen doppelt gebucht werden dürfen. Genau diese Einstellungsmöglichkeit entspricht der Anforderung an den Prüfungsplaner.

Leider war es auch nach vielen Versuchen nicht möglich, über die COM-Schnittstelle diese Werte zu beeinflussen. Aus diesem Grund kann die gestellte Anforderung nicht direkt mit Hilfe des Prüfungsplaners erfüllt werden. Der einzige Weg Ressourcen doppelt zu buchen ist S-PLUS zu starten, die entsprechenden Optionen zu setzen und danach den Prüfungsplaner zu benutzen, um eine Prüfung unter den gewünschten Bedingungen zu planen.

5 Fazit

Die vorliegende Diplomarbeit beschreibt die Entwicklung eines Frameworks für die COM-Schnittstelle des an der Technischen Fachhochschule Wildau verwendeten Veranstaltungsplaners. Das erstellte Framework wird für eine Anwendung genutzt, welche die Prüfungen an der Technischen Fachhochschule Wildau verwaltet und durch Mitarbeiter benutzt werden soll.

Bei der Erfüllung der gestellten Anforderungen an das Framework und der Anwendung zum Planen von Prüfungen muss differenziert werden.

Das Auslesen und Verarbeiten von Daten mit Hilfe des Frameworks ist problemlos realisierbar. Am Beispiel der Webpräsentationskomponente ist zu erkennen, dass diese Funktionalität durch das Framework abgedeckt wird.

Beim Bearbeiten von Daten des Veranstaltungsplaners sind jedoch diverse Probleme aufgetreten. Die Möglichkeiten der COM-Schnittstelle sind leider an einigen Stellen begrenzt. Es gibt auch einige Fälle in denen Methoden oder Operationen der COM-Schnittstelle nicht das erwartete Resultat erzielen. Als Beispiel dienen die Ausführungen in Kapitel 4.3.6. Aus diesem Grund war es in einigen Fällen notwendig, Abstriche bei den Funktionalitäten des Prüfungsplaners zu machen.

Das einige Funktionalitäten nicht realisiert werden konnten liegt nicht am Framework oder einer entwickelten Anwendung, sondern an der COM-Schnittstelle. Gegebenenfalls können die in der jetzigen Version des Veranstaltungsplaners nicht realisierten Anforderungen in Nachfolgeversionen verbessert werden.

Änderungen der COM-Schnittstelle in folgenden Versionen können durch das Framework nachvollzogen werden. Es wurde für Erweiterbarkeit und Wartbarkeit ausgerichtet, so dass Erweiterungen in der Funktionalität sowie Änderungen der COM-Schnittstelle schnell implementiert werden können.

Insgesamt wurde mit dem Framework eine Plattform geschaffen, die den gestellten Anforderungen entspricht. So weit es möglich war, wurde der Datenzugriff auf die COM-Schnittstelle erleichtert und verbessert. Zudem wird durch die XML-Kommentierung von Klassen, Methoden und Eigenschaften die Arbeit mit S-Plus wesentlich erleichtert. Einem Entwickler werden beschreibende Kommentar im *Visual Studio* über die im Framework definierten Typen geliefert.

Neben der Erleichterung beim Zugriff auf Datenobjekte von S-Plus ermöglicht das Framework durch eine Administrationsoberfläche und einer Anwendung zum Erstellen von Konfigurationsdateien die bequeme Weitergabe von Projekten. In diesem Bereich wurden die im Rahmen der Diplomarbeit gestellten Anforderungen übertroffen.

Das Anpassen der Webpräsentationskomponente ist wie bereits erwähnt ohne Probleme verlaufen. Der größte Nutzen der durch die Anpassung entsteht ist der Wegfall von allgemeinen Typen für die Webpräsentationskomponente. Diese Typen werden nun im Framework deklariert und stehen allen Entwicklern zur Verfügung.

Für die Zukunft sind weitere Anwendungen die ähnlich der Webpräsentationskomponente Daten auslesen und weiterverarbeiten möglich. Denkbar ist beispielsweise eine Anwendung die das automatisierte verschicken von E-Mails an Dozenten übernimmt und diese über Änderungen ihres Stundenplans informiert.

Obwohl es die genannten Schwierigkeiten bei der Umsetzung einiger Funktionalitäten beim Bearbeiten von Daten des Veranstaltungsplaners gab, empfehle ich den Einsatz des Prüfungsplaners an der Technische Fachhochschule Wildau. Das Erstellen und Planen von Prüfungen wird zuverlässig und einfach ermöglicht. Um Spezialfälle wie das mehrfache Buchen einer Ressource zu realisieren, muss aber leider trotzdem noch die grafische Benutzeroberfläche des Veranstaltungsplaners genutzt werden.

Nach einer kurzen Einführungsphase ist die Handhabung der Anwendung leicht zu verstehen. Sie erfüllt die Anforderung das Planen von Prüfungen zu vereinfachen. Mit dem mitgelieferten Handbuch wird zusätzlich Unterstützung geboten.

Natürlich werden in der Anfangsphase bei der Einführung des Prüfungsplaners Probleme auftreten. Trotz intensiver Tests und dem Durchspielen verschiedener Szenarien ist es schwer möglich alle Anwendungsfälle zu überprüfen. Des Weiteren kann es zu Wünschen kommen die Anwendung in bestimmten Punkten zu ändern, zu verbessern oder etwas wegzulassen.

Aus diesem Grund möchte ich der Technischen Fachhochschule Wildau in der Einführungsphase des Prüfungsplaners zur Verfügung stehen. Eventuelle Änderungswünsche oder Verbesserungsvorschläge werden gerne entgegengenommen und umgesetzt.

Literatur

- [BBC01] BROWN, Ray / BARON, Wade / CHADWICK III, William D.:
Designing Solutions with COM+ Technologies
Microsoft Press, Redmond, Washington 2001
- [BBMHPW06] BEER, Wolfgang / BIRNGRUBER, Dietrich / MÖSSENBOCK,
Hanspeter / PRÄHOFFER, Herbert / WÖß, Albrecht:
Die .NET-Technologie
2. Auflage, dpunkt.verlag, Heidelberg 2006
- [CA07] CWALINA, Krzysztof / ABRAMS, Brad:
Richtlinien für das Framework-Design
Addison-Wesley Verlag, München 2007
- [CHA96] CHAPPELL, David: Understanding ActiveX and OLE
Microsoft Press, Redmond, Washington 1996
- [GHJV95] GAMMA, Erich / HELM, Richard / JOHNSON, Ralph / VLISSIDES,
John: Entwurfsmuster
Addison-Wesley Verlag, München 1995
- [FF04] FREEMAN, Eric / FREEMAN, Elisabeth: Head First Design Patterns
O'Reilly, Gravenstein Highway North, Sebastopol 2004
- [MÖS06] MÖSSENBOCK, Hanspeter: Softwareentwicklung mit C# 2.0
2. Auflage, dpunkt.verlag, Heidelberg 2006
- [RIC06] RICHTER, Jeffrey:
Microsoft .NET-Framework-Programmierung mit C#
2. Auflage, Microsoft Press Deutschland, Unterschleißheim 2006
- [TH04] TEILHET, Stephen / HILLYARD, Jay: C# Cookbook
O'Reilly, Gravenstein Highway North, Sebastopol 2004
- [TRO02] TROELSEN, Andrew: C# und die .NET-Plattform
1. Auflage, mitp-Verlag, Bonn 2002

Internet

Alle angegebenen Internetadressen beziehen sich auf den März 2008.

[COM]	Microsoft, Component Object Model Technologies http://www.microsoft.com/com/default.msp
[ECMA]	ECMA International http://www.ecma-international.org/
[ECMA334]	Standard ECMA-334: C# Language Specification http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-334.pdf
[ECMA335]	Standard ECMA-335: Common Language Infrastructure (CLI) http://www.ecma-international.org/publications/files/ECMA-ST/Ecma-335.pdf
[ISO23270]	ISO/IEC 23270:2003; Information technology – C# Language Specification http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=36768
[MONO]	Mono-Project: Was ist Mono? http://www.mono-project.de/was-ist-mono/
[MSDNNET]	msdn, .NET Framework Developer Center http://msdn2.microsoft.com/de-de/netframework/default.aspx
[MSDNC#]	msdn, Visual C# Developer Center http://msdn2.microsoft.com/de-de/vcsharp/default.aspx
[MSDNC#2]	Einführung in die Programmiersprache C# und in .NET Framework http://msdn2.microsoft.com/de-de/library/z1zx9t92(vs.80).aspx
[OCOV]	Windows 2000 Resource Kit Tool: OLE/COM Object Viewer http://www.microsoft.com/downloads/details.aspx?familyid=5233b70d-d9b2-4cb5-aeb6-45664be858b6&displaylang=en

---

[TLI]	msdn, .NET-Framework Developer Center .NET-Framework-Tools: Type Library Importer-Tool (Tlbimp.exe) http://msdn2.microsoft.com/de-de/library/tt0cf3sx(VS.80).aspx
[TLISUM]	msdn, .NET-Framework Developer Center Zusammenfassung: Konvertieren einer Typbibliothek in eine Assembly http://msdn2.microsoft.com/de-de/library/k83zzh38(VS.80).aspx
[SCIENTIA]	Scientia – ein Weltmarktführer http://www.scientia.de/scientia/index.html
[SPLUS]	S-PLUS, Überblick http://www.scientia.de/s-plus/ueberblick.htm
[STAL]	Michael Stal: Microsoft DCOM http://www.stal.de/Downloads/DCOM/DCOM.html
[TFHWZA]	TFH Wildau – die Entwicklung in Zahlen http://www.tfh-wildau.de/ueberuns/index_statistik.html
[TFHWPLAN]	Lehrveranstaltungen und Prüfungen an der TFH Wildau http://www.tfh-wildau.de/stundenplan/
[TFHWPLANSTU]	Lehrveranstaltungen und Prüfungen an der TFH Wildau am Beispiel Der StudentenSet-Gruppen http://www.tfh-wildau.de/stundenplan/
[TFHWPLANBB05]	Veranstaltungen der Seminargruppe BB-05 / Sommersemester 08 http://www.tfh-wildau.de/stundenplan/stundenplan_ss08/studenten/BB-05_DA451CBD0F9365859C24A1CC51E6D459/AlleVeranstaltungen.xml
[VM]	IT Wissen: Definition einer Virtuellen Maschine http://www.itwissen.info/definition/lexikon/_VMVM_VMvirtual%20maschineVM_VMVirtuelle%20Maschine.html

- [WPK]   Webpräsentationskomponente im Stundenplansystem der
Technischen Fachhochschule Wildau
<http://www.tfh-wildau.de/stundenplan/formulare/Praesentationskomponente.pdf>

S-PLUS Handbücher

Die hier verzeichneten Handbücher befinden sich auf der CD in Anlage B.

- [HB350]   Handbuch für S-PLUS 3.5.1 Veranstaltungsplaner mit Modulplaner
Scientia GmbH, Hansaring 61, 50670 Köln
- [SAS]   Syllabus Plus Automation Server 3.5.0
Scientia GmbH, Hansaring 61, 50670 Köln

Anlagen

Erstellte Adapter im Framework

Die folgende Tabelle listet alle im Framework zur Verfügung gestellten Objektadapter und ihre Schnittstelle auf. Für jeden Adapter wird der Name der adaptierten COM-Schnittstelle genannt. Zusätzlich zeigt die Tabelle Informationen darüber ob für einen Adapter eine Klasse erstellt wurde die sich von ObjectBuilder oder von ObjectEditor ableitet.

Adapter-Schnittstelle	COM-Schnittstelle	Builder / Editor
IBevorzugerBeginn	StartPreferencePattern	Builder
IBevorzugterEinsatz	UsagePreferencePattern	Builder
IDozent	StaffMember	Editor
IDozentGruppe	StaffMemberGroup	Editor
IEquipment	EquipmentItem	Editor
IEquipmentGruppe	EquipmentItemGroup	Editor
IFachbereich	Department	Editor
IFachbereichGruppe	DepartmentGroup	Editor
IFlagge	Tag	Editor
IFlaggeGruppe	TagGroup	Editor
IHochschule	College	
IKurs	Course	Editor
IKursGruppe	CourseGroup	Editor
IModul	Module	Editor
IModulGruppe	ModuleGroup	Editor
IRaum	Location	Editor
IRaumGruppe	LocationGroup	Editor
IRessourcenPool	PooledResource	Editor
IRessourcenPoolGruppe	PooledResourceGroup	Editor
ISektion	Section	Editor
ISektionGruppe	SectionGroup	Editor
ISemester	ProgrammeOfStudy	Editor
ISemesterGruppe	ProgrammeOfStudyGroup	Editor
IStandort	Zone	Editor
IStandortGruppe	ZoneGroup	Editor
IStudentenSet	StudentSet	Editor
IStudentenSetGruppe	StudentSetGroup	Editor
IVeranstaltung	Activity	Editor
IVeranstaltungGruppe	ActivityGroup	Editor
IVeranstaltungsart	ActivityType	Editor
IVerfügbarkeitsmuster	AvailabilityPattern	Builder

CD

Auf der beiliegenden CD zur Diplomarbeit befinden sich das Softwareprojekt sowie weitere Inhalte. Folgende Verzeichnisse gliedern den Aufbau der CD:

- § **Diplomarbeit** – In diesem Verzeichnis befindet sich die Diplomarbeit im PDF-Format.
- § **Installation** – Dieses Verzeichnis umfasst das Installationspaket des Frameworks sowie die in der Diplomarbeit beschriebenen Anwendungen Prüfungsplaner und Webpräsentationskomponente. Des Weiteren befinden sich die Handbücher in diesem Verzeichnis.
- § **Quellen** – Hier sind einige Webseiten, die als Quellen für die Diplomarbeit dienen Offline zu finden. Außerdem sind hier die als Quellen angegebenen Handbücher von S-PLUS gespeichert.
- § **Ressourcen** – Der Quellcode des Softwareprojektes sowie das Assembly *SplusServer.dll* befinden sich in diesem Verzeichnis.

Selbständigkeitserklärung

Hiermit erkläre ich, dass die vorliegende Diplomarbeit selbständig und ohne unerlaubte Hilfe angefertigt wurde. Sinngemäße oder wörtliche Zitate sind kenntlich gemacht und durch das Quellenverzeichnis prüfbar.

Ort, Datum

Unterschrift