



JOHANNES KEPLER
UNIVERSITÄT LINZ

Netzwerk für Forschung, Lehre und Praxis



Implementierung des Model-View-Controller Paradigmas für das WeLearn-System (Web Environment for Learning)

DIPLOMARBEIT

zur Erlangung des akademischen Grades

DIPLOMINGENIEUR

in der Studienrichtung

INFORMATIK

Angefertigt am *Institut für Informationsverarbeitung und Mikroprozessortechnik*

Von:

Dietmar Stoiber

Betreuung:

o.Univ.-Prof. Dr. Jörg R. Mühlbacher

Dipl.-Ing. Dr. Susanne Loidl

Beurteilung:

o.Univ.-Prof. Dr. Jörg R. Mühlbacher

Marchtrenk, 16 Februar 2005

Kurzfassung

Das Ziel des Model-View-Controller Paradigmas (MVC) ist die Dekomposition von interaktiven Systemen in Teile, die soweit voneinander unabhängig sind, so dass sie einzeln geändert oder ausgetauscht werden können. In der Softwarewelt existiert eine Trennung in Ausgabe, Eingabe und Verarbeitung schon lange. Das MVC-Paradigma überträgt dies auf GUI (Graphical User Interface)-basierte Systeme, wobei das "Model" mit der Verarbeitung, die "View" mit der Ausgabe und der "Controller" mit der Eingabe übereinstimmen. Die drei Komponenten werden separat behandelt, wodurch die Erstellung von Anwendungen mit mehreren Sichten auf die selben Daten vereinfacht wird. Das MVC-Paradigma hat unter anderem das Ziel, dass Änderungen an den Views möglichst ohne Änderungen an Model und Controller vorgenommen werden können.

- Das Model stellt die Repräsentation der Daten einer Anwendung dar. Neben der Verwaltung der Daten ist es auch für alle Änderungen derselben zuständig. Ein Modell kann mit beliebig vielen Views verbunden sein.
- Das View-Objekt übernimmt die grafische Darstellung der Daten, wobei es nur eine der möglichen Darstellungen des Modells ist. Ein View-Objekt ist immer mit genau einem Modell verbunden.
- Der Controller definiert die Aktionsmöglichkeiten des Benutzer mit der Applikation. Er nimmt Eingaben vom Benutzer entgegen und bildet diese auf Funktionen des Modells oder der View ab.

Im Laufe dieser Diplomarbeit wurde für das WeLearn System (Web Environment for Learning) des Instituts für Informationsverarbeitung und Mikroprozessortechnik (FIM) der Johannes Kepler Universität Linz das MVC-Paradigma implementiert.

Die Systemobjekte entsprechen den wesentlichen 'Objekten' von HTML (z.B: Tabellen, Listen, Links, Bilder, Formulare, etc.), welche um einige spezielle komplexe Objekte, wie Datumswähler, Bäume, etc. erweitert wurden.

Abstract

The goal of the Model-View-Controller paradigm (MVC) is the decomposition of interactive systems into parts, which are independent from each other in so far that they can be changed or exchanged individually without infecting the other components. In the software world a separation in output, input and processing already exists for a long time. The MVC paradigm transfers this to GUI (Graphical User Interface) based systems, whereby the "Model" corresponds with the processing, the "View" with the output and the "Controller" with the input. These three components are handled separately, whereby the building of applications with several views to the same data is simplified. The MVC paradigm has among other things the aim that changes at the views can be made as far as possible without changes at Model and Controller.

- The Model is a representation of the data of an application. In addition to the administration of the data it is responsible also for all changes of the same. A model can be connected with as any desired View.
- The View-object is responsible for the representation of the data, whereby it is only one of the possible representations of the Model. A View-object is always connected with exactly one Model.
- The Controller defines the possible actions, which the user can use within the application. It receives input from the user and maps it on functions of the Model or the View.

During this diploma thesis the MVC-paradigm was implemented for the WeLearn System (Web Environment for Learning) of the Institute for Information Processing and Microprocessor Technology (FIM) of the Johannes Kepler University Linz.

The symbol objects are equivalent to the basic 'objects' of HTML (e.g.: tables, lists, links, images, forms, etc.), extended with some special complex objects like date-choosers, trees, etc.

Danksagung

An dieser Stelle möchte ich mich bei allen bedanken, die zum Gelingen dieser Arbeit beigetragen haben.

Als erstes möchte ich meiner Mutter danken, die mir das Studium ermöglicht hat und mich in allen Bereichen unterstützt hat. Weiters möchte ich meiner Verlobten für ihre Unterstützung danken.

Für die gute Betreuung und wertvollen wissenschaftlichen und organisatorischen Tipps möchte ich Frau Dipl.-Ing. Dr. Susanne Loidl besonders danken.

Weiters danke ich Herrn o.Univ.-Prof. Dr. Jörg R. Mühlbacher für die wertvollen Tipps und Anregungen, sowie die mir gebotene Möglichkeit an dem Projekt WeLearn mitzuarbeiten.

Sehr hilfreich waren auch die vielen Anregungen der WeLearn-Mitarbeiter, besonders möchte ich mich bei Mag. iur. Dipl.-Ing. Dr. Michael Sonntag, MSc. Alexandros Paramythis und Thomas Göbl bedanken.

Inhaltsverzeichnis

1 E-Learning.....	1
1.1 Definition.....	1
1.2 Motivation.....	2
1.2.1 Ortsunabhängig und/oder Zeitunabhängig.....	2
1.2.2 An den Lernenden anpassend.....	4
1.2.3 Interaktiv.....	4
1.2.4 Multimedial.....	4
1.3 Vorteile.....	4
1.4 Nachteile.....	5
2 WeLearn.....	7
3 Das Model-View-Controller Paradigma.....	10
3.1 Geschichte.....	10
3.2 Konzept.....	10
3.3 Model.....	12
3.4 View.....	13
3.5 Controller.....	14
3.6 Kommunikation zwischen den Komponenten.....	14
3.6.1 Zustandsabfrage: View → Model.....	15
3.6.2 Zustandsänderung: Controller → Model.....	16
3.6.3 Viewselektion: Controller → View.....	16
3.6.4 Änderungsnachricht: Model → View.....	16
3.6.5 Benutzerinteraktion: Benutzer → Controller.....	17
3.6.6 Ausgabe: View → Benutzer.....	17
3.6.7 Kommunikation zwischen den Komponenten in der Realität.....	17
3.6.8 Ablauf der Kommunikation zwischen den Komponenten.....	18
3.7 Beispiel.....	19
3.8 Vorteile und Nachteile des MVC-Paradigmas.....	19
3.9 Umsetzungen des MVC-Paradigmas im Java-Umfeld.....	20
3.10 Fazit.....	23
4 Millstone.....	24
4.1 Idee.....	24
4.2 Umsetzung.....	25
4.3 Vorteile und Nachteile von Millstone.....	29
4.4 Einblicke.....	30
4.4.1 Übermittlung von Parametern von den Benutzerschnittstellen-Komponenten an den Web-Browser.....	30
4.4.2 Übermittlung von Parametern vom Web-Browser an die Benutzerschnittstellen-Komponenten.....	32
4.4.3 Funktionsweise einer einfachen Benutzerschnittstellen-Komponenten am Beispiel der Button-Klasse.....	35
5 Millstone – WeLearn-Edition.....	39

5.1 Ziele.....	39
5.2 Architektur.....	39
5.2.1 BaseLib.....	39
5.2.2 Themes.....	41
5.2.3 WebAdapter.....	41
5.3 Umsetzung.....	42
5.3.1 Erweiterung der Tabelle.....	42
5.3.2 Implementierungen der ImageMap.....	44
5.3.3 Änderung von JavaScript- in HTML-Funktionalität.....	45
5.3.4 Fehlerbeseitigung.....	46
5.4 Verwendung.....	47
6 Fallstudie – Millstone installieren, verwenden, erweitern.....	48
6.1 Millstone installieren (mit aller benötigter Software).....	48
6.2 Millstone verwenden (eine eigene Applikation schreiben).....	54
6.3 Millstone erweitern (ein bestehendes Theme erweitern).....	57
7 Fazit.....	59
8 Appendices.....	61
8.1 Appendix A – Benutzerhandbuch für das modifizierte Millstone.....	61
8.1.1 Label.....	65
8.1.2 Layout.....	66
8.1.3 Button.....	68
8.1.4 Textfield.....	69
8.1.5 Datefield.....	71
8.1.6 Form.....	72
8.1.7 Link	75
8.1.8 Panel.....	76
8.1.9 TabSheet.....	78
8.1.10 Select.....	79
8.1.11 Table.....	84
8.1.12 Tree.....	89
8.1.13 Window.....	91
8.1.14 Embedded.....	93
8.1.15 Upload.....	95
8.1.16 ImageMap.....	97
8.1.17 FrameWindow.....	100
8.1.18 Beispiel mit allen Elementen.....	103
8.2 Appendix B – Codebeispiele.....	112
9 Quellverzeichnis.....	125
9.1 Literaturverzeichnis.....	125
9.2 Abbildungsverzeichnis.....	130

1 E-Learning

1.1 Definition

Für den Begriff Electronic-Learning (e-Learning) gibt es keine allgemeingültige Definitionen. In der Literatur stößt man auf unterschiedlichste Definitionen. Einige werden hier exemplarisch angeführt:

"If someone is learning in a way that uses information and communication technologies (ICTs), they are e-learning. They could be a pre-school child playing an interactive game; a group of pupils collaborating on a history project with pupils in another country via the Internet; a group of geography students watching an animated diagram of a volcanic eruption their lecturer has just downloaded; a nurse taking her driving theory test online with a reading aid to help her dyslexia - the list goes on and it all counts as e-learning."

Department of education and skills (UK) [DfES]

"Distributed and electronic learning can be represented as a spectrum ranging from Internet-supported distance learning in which the learner has limited physical contact with the tutor or other learners, to teacher-led, classroom-based activity which is interspersed with occasional computer-delivered or facilitated assignments."

Distributed and Electronic Learning Group (UK) [DELG02]

"E-learning is Internet-enabled communication and training that allows quick, effective delivery at lower costs than traditional methods. Companies can use these solutions to provide an enhanced learning experience that is scalable, up-to-date, can be self-paced, and provides learners with anytime, anywhere access to knowledge."

Cisco [Cis01]

"E-learning most often means an approach to facilitate and enhance learning by means of personal computers, CDROMs and the internet. [...] Advantages are seen in that just-in-time learning is possible, courses can be tailored to specific needs and asynchronous learning is possible. e-learning may also be used to support distance learning through the use of WANs (Wide area networks), and may also be considered to be a form of flexible learning. Often, but not always, e-learning will also attempt to be a student-centred learning solution."

Wikipedia [Wiki/1]

Aus den verschiedenen Definitionen kann man einige Eigenschaften ableiten:

- E-Learning ist ortsunabhängig und/oder zeitunabhängig
- E-Learning passt sich an den Lernenden an
- E-Learning ist interaktiv
- E-Learning ist multimedial
- E-Learning gibt es in jeder Konstellation (m zu n, 1 zu n und 1 zu 1)

Ein E-Learning-Projekt muss nicht alle oben genannten Eigenschaften umfassen. Es ist zum Beispiel nicht nötig, dass sich die Lernplattform an den Lernenden anpasst, auch ist es möglich dass E-Learning-Unterlagen nur als Text angeboten werden, also nicht multimedial sind.

Die Motivation für E-Learning ergibt sich zum Großteil aus den ersten vier Eigenschaften. Auf diese werden im folgenden Unterkapitel näher betrachtet.

1.2 Motivation

1.2.1 Ortsunabhängig und/oder Zeitunabhängig

Beim traditionellen Lernen ist es zwar auch teilweise möglich orts- und zeitunabhängig zu lernen, allerdings gilt dies meist nur für das Lernen für Prüfungen. Die eigentlichen Kurse muss man dabei aber zu gewissen Zeiten an der jeweiligen Bildungsanstalt besuchen.

Beim e-Learning können nun auch die Kurse orts- und zeitunabhängig durchgeführt werden. Dafür sind in der Regel alle Lehrveranstaltungen geeignet, die nicht durch Hard- oder Software-Beschränkungen an einen bestimmten Raum gebunden sind - beispielsweise Software, die aus lizenzrechtlichen Gründen nur auf wenigen Instituts-Rechnern verwendet werden darf oder spezielle Maschinen (z.B.: Drehbänke für Mechatroniker). Ein Programmierpraktikum wäre zum Beispiel ein ideales Fach, um es (zumindest bis auf einen eventuellen Abschlußvortrag) online abzuhalten, jedoch selbst chemische Experimente können, eine geeignete Simulationssoftware vorausgesetzt, am Computer durchgeführt werden (und verringern zusätzlich noch den Chemikalienbedarf des Labors) .

Es gibt daher mehrere Möglichkeiten der Umsetzung, die nicht alle orts- und

zeitunabhängig zugleich sind:

- ortsunabhängig, aber zeitabhängig:

Fächer, die zwar eine unmittelbare Interaktion der Lernenden erfordern, aber nicht deren physische Anwesenheit benötigen. Ein Beispiel hierfür wäre eine Vorlesung, von der ein live Video-Stream im Internet zur Verfügung gestellt wird, und es über einen Rückkanal (E-Mail, Instant-Messaging, usw.) den entfernten Studenten ermöglicht wird Fragen zu stellen, als ob diese persönlich anwesend wären.

- Zeitunabhängig aber ortsabhängig:

Fächer, die zwar einen bestimmten Ort (dessen Ausstattung) erfordern, aber nicht unbedingt gemeinsam abgehalten werden müssen. Ein Beispiel hierfür wäre ein chemisches Experiment, für das zwar die Apparaturen benötigt werden, aber etwa bei Studenten höherer Semester keine Lehrperson mehr zur Überwachung erforderlich ist.

- Orts- und Zeitunabhängig

Dies bietet sowohl Lehrenden, als auch Lernenden maximale Flexibilität. Ein Beispiel hierfür wäre eine Vorlesung, die auf einem ausführlichen Skriptum – eventuell mit multimedialen Erweiterungen (zum Beispiel Videoaufzeichnung) – basiert und bei der alle Fragen in einem Online-Forum behandelt werden.

Ortsunabhängigkeit ist insbesondere für Lernende interessant, die weit von der Bildungsanstalt entfernt wohnen und pendeln müssen. Mittels E-Learning haben diese die Möglichkeit die Stehzeiten (im Zug, am Bahnhof, etc.) besser für Studienzwecke nutzen zu können, wodurch ihnen mehr Freizeit zur Verfügung steht.

Zeitunabhängigkeit ist vermutlich für berufstätige Lernende einer der interessantesten Aspekte von E-Learning, da es für diese meist nicht einfach möglich ist zu beliebigen Zeiten von ihrer Arbeit fernzubleiben, die Lehrenden aber darauf meist keine Rücksicht nehmen können (besonders bei Lehrveranstaltungen mit vielen Teilnehmern). Aber natürlich bietet die Zeitunabhängigkeit auch nicht arbeitenden Studenten Vorteile, da etwa Probleme bei einer Übung im Forum gepostet werden können, wo diese von anderen Studenten, Tutoren, oder Lehrenden auch schon vor der nächsten Lehrveranstaltungsstunde beantwortet werden können.

1.2.2 An den Lernenden anpassend

Dieser Punkt ist eine der größten Motivationen für E-Learning. Eine traditionelle

Lehrveranstaltung ist eine Gratwanderung zwischen den Fähigkeiten und dem Vorwissen der Teilnehmer. Nimmt man den Stoff zu schnell durch überfordert man die Schwächeren, nimmt man ihn zu langsam durch unterfordert man die Besseren. In der Realität läuft dies meist darauf hinaus, dass man den Unterricht am unteren Durchschnitt ausrichtet, um die Schlechtesten nicht zu sehr zu überfordern, erreicht dadurch allerdings, dass die Besten dem Unterricht nicht mehr aufmerksam folgen und schließlich den Anschluss verlieren. [Wiki/2]

Für diese Situation ist E-Learning, das sich an den Lernenden anpasst natürlich optimal, denn dabei hat jeder die Möglichkeit den Stoff in der für ihn passenden Geschwindigkeit zu lernen ohne einen anderen Lernenden dadurch zu benachteiligen. Weiters kann die Lernplattform mittels automatisierter Tests das Wissen der Lernenden überprüfen und etwa Studenten mit geringerem Vorwissen zusätzliche Information anbieten bzw. die Wiederholung eines Kapitels empfehlen.

1.2.3 Interaktiv

Nicht umsonst heißt es "Learning by doing". Das menschliche Gehirn ist so gebaut, dass es Abläufe, die man selbst ausführt, leichter und besser aufnimmt, als wenn man sie nur liest oder hört. [Mensa][Qis]

Beim traditionellen Unterricht sind der Interaktivität bei hunderten Personen in einer Lehrveranstaltung Grenzen gesetzt. Bei E-Learning ist es problemlos möglich jedem Lernenden – zumindest mittels einer Simulation – einen Ablauf selbst durchführen zu lassen und unmittelbar eine Rückmeldung zu geben.

1.2.4 Multimedial

Beim traditionellen Unterricht wird meist auf statische (gedruckte) Skripten zurückgegriffen, beim E-Learning sollen Animation in das Unterrichtsmaterial Einzug halten, da man anhand einer Animation bzw. Simulation meist einen Ablauf einfacher und effektiver erklären kann, als es mit Bildern oder reinem Text möglich ist.

1.3 Vorteile

Das orts- und zeitunabhängig Lernen ist ein großer Vorteil, da sowohl berufstätige, als auch entfernt wohnende Lernende unterstützt werden, aber es kommt insgesamt allen Lernenden zugute, da nicht jeder Mensch während des ganzen Tages gleich aufnahmebereit ist, und sich, bei freier Einteilung, dann mit dem Stoff beschäftigen kann, wenn es für ihn am optimalsten ist.

Da jeder Mensch unterschiedlich schnell lernt und auch unterschiedliche Strategien zum Lernen verwendet, ist ein E-Learningsystem, das sich an die jeweiligen Bedürfnisse des Lernenden anpasst insbesondere für die Lernenden von großem Vorteil, die ein anderes Lernverhalten haben als der Durchschnitt (etwa Hochbegabte mit einer raschen Auffassungsgabe, die bei durchschnittlichem Lerntempo durch mangelnde Konzentration den Anschluss verlieren können).

Interaktives Lernen fördert durch die intensivere Beschäftigung mit der Materie das Lernen. Durch den aktiven Lernvorgang prägt sich das zu Lernende schneller und intensiver ins Gedächtnis ein. Außerdem kann man durch Wiederholungen feststellen, welche Stoffteile ein Lernender noch nicht gut beherrscht, und denjenigen individuell genau diese Teile intensivieren lassen. Außerdem ist es durch die Interaktivität möglich kleine Beispiele vorzubereiten, die die Lernenden bearbeiten können und sofort eine Rückmeldung bekommen, ob sie alles richtig gemacht haben.

Durch multimediale Inhalte wird das Vermitteln von Wissen beim E-Learning gegenüber dem traditionellen Lernen vereinfacht, da man Abläufe nicht mehr textuell oder mit ein paar Bildern darstellen muss, sondern dafür eine Animation oder ein Video verwenden kann.

Ein weiterer Vorteil von E-Learning ist die Art des Lernmaterials. Beim herkömmlichen Lernmaterial hat man mit mehreren Lehrveranstaltungen meist mehrere Ordner an Unterlagen, und kann in der Regel nicht alle bei sich haben. Beim E-Learning kommt man mit einem Notebook bzw. Handheld (und eventuell ein paar zusätzlichen Speicherkarten) aus. Wodurch es möglich ist, immer das passende Skript zur Hand zu haben.

1.4 Nachteile

Einer der wenigen Nachteile an E-Learning ist, dass man Menschen ohne Computerkenntnisse, bzw. ohne eigenen Computer benachteiligt, da sie zu weniger Information Zugang haben als Lernende mit Computer.

Weiters könnten Lernende mit geringer Selbstdisziplin durch die freiere Einteilung des Lernens dazu verleitet werden, erst knapp vor den Prüfungen mit dem Lernen zu beginnen, was sich negativ auf die Ergebnisse auswirken könnte.

Von technischer Seite wird die Ortsunabhängigkeit bei elektronischen Dokumenten allerdings im Moment auch noch eingeschränkt, da ein Notebook im Akku-Betrieb kaum mehr

als vier Stunden durchhält, wogegen ein Skriptum aus Papier keinen solchen Einschränkungen unterliegt. Eine weitere Einschränkung erfolgt durch interaktive, sich anpassende Lernsoftware, wenn diese nicht lokal lauffähig ist, da bei Interaktion mit einem Server meist eine schnelle Internetverbindung benötigt wird, die aber im mobilen Bereich (etwa Verbindung mittels eines Handy) noch nicht zur Verfügung steht.

2 WeLearn

WeLearn (Web Environment for Learning) ist eine virtuelle Lernumgebung, die vom Institut für Informationsverarbeitung und Mikroprozessortechnik [FIM] der Johannes Kepler Universität Linz [JKU] mittlerweile in der Version 3 entwickelt wird.

WeLearn geht dabei weit über die bloße online Verfügbarkeit von Lehr-, bzw. Lernmaterialien hinaus. So ist es durch ein umfassendes Rechtemanagement personalisierbar, unterstützt beliebige Formate der Unterrichtsmaterialien und ermöglicht Kommunikation und Interaktion zwischen Lehrenden (Professoren, Assistenten, Tutoren, ...) und Lehrenden (Studenten) sowohl gegenseitig als auch untereinander. Folgende Definition von WeLearn stammt aus der Dissertation von Frau DI. Dr. Susanne Loidl (geb. Reisinger) [Rei02]:

WeLearn (Web Environment for Learning) ist ein Framework, das aus drei wesentlichen Teilen besteht:

- *Einer Plattform für Distance Teaching/Coaching/Learning, die die Präsentation und Organisation von Lernmaterial, die Administration von Kursen, Kommunikation und Interaktion und auch Self-Assesment mit einschließt.*
- *Webbasierten Lehr- und Lerninhalten, wie "Propädeutikum aus Informatik", "Einführung in Betriebssysteme", etc. und*
- *Beispiel-Settings oder Lernarrangements zur didaktischen Umsetzung von Lernmodellen, die Lehrende als Grundlage für ihre Kurse heranziehen oder adaptieren, aber auch individuelle Lernmodelle umsetzen können.*

Ergänzend zur obigen kurzen Erwähnung von einigen WeLearn-Features wird folgende Liste angefügt:

- Die WeLearn-Plattform: Das Framework selbst stellt die Lernmaterialien online, per Web zur Verfügung, ermöglicht kollaboratives Arbeiten, etc.
- Agenten: Agenten werden bei WeLearn beispielsweise verwendet um nach Lehrveranstaltungen zu suchen oder auch für die Koordination von Lehrveranstaltungen, indem sie automatisch alle Teilnehmer über eine Terminänderung informieren.

- Kurs-Templates: Vorgefertigte Kurs-Templates ermöglichen dem Lehrenden einfach eigene Kurse nach seinen Bedürfnissen zu erstellen.
- Personalisierung: Die Personalisierung ist die Möglichkeit das System individuell an die Bedürfnisse und Vorlieben einzelner Benutzer anzupassen. Wobei sowohl eine optische als auch funktionale Anpassung möglich ist.
- Rechtemanagement: Es können die Rechte für verschiedene Anwendergruppen und Anwendungsszenarien verwaltet werden. Zum Beispiel für Lehrende, Tutoren und Lernende.
- WeLearn Offline Konverter: Der Offline Konverter wurde entwickelt um aus dem für das online-WeLearn verwendete CPS-Format (ein Standard des Instructional Management System (IMS) des Global Learning Consortiums [IMS]) ein offline verwendbares Format wie HTML oder Applets zu generieren, damit Kursinhalte auch auf CD verteilt werden bzw. offline verwendet werden können.
- Workflowmanagement: Mittels des Workflowmanagements können Abläufe automatisiert werden. Etwa eine Benachrichtigung der Tutoren über abgegebene, zu korrigierende Übungen, und eine automatisierte Rückmeldung an die Lernenden über Punkte und Korrekturen.

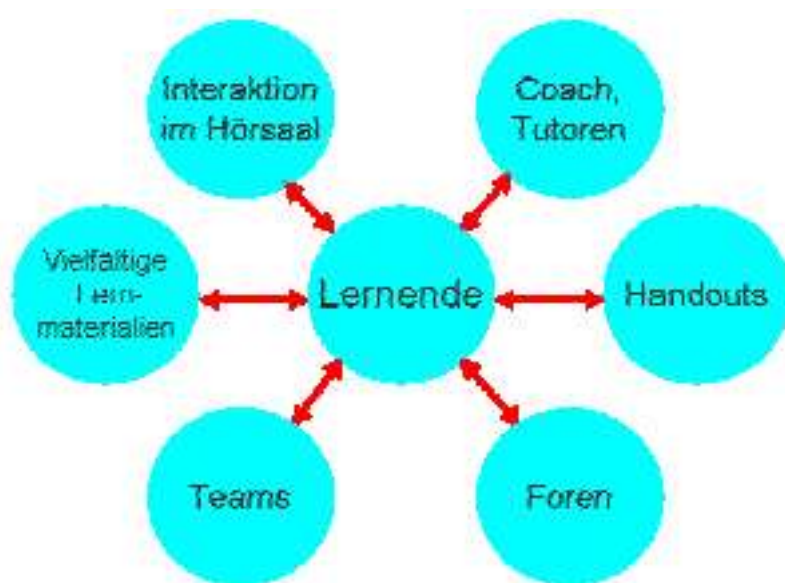
Zur Verwendung von WeLearn benötigt man nur einen Web-Browser und Programme, um die unterschiedlichen Kursmaterialien darstellen zu können, wodurch WeLearn prinzipiell plattformunabhängig ist. Es können hierfür alle Formate verwendet werden, es sollte allerdings darauf geachtet werden, dass man die Plattformunabhängigkeit von WeLearn nicht durch Kursmaterial in proprietären Formaten aushebelt, für die es auf vielen Plattformen keine Anzeigemöglichkeit gibt. Da WeLearn in Java programmiert wurde, gibt es serverseitig keine Einschränkungen.

WeLearn unterstützt sowohl die Lehrenden - etwa durch ein Authoringtool zur Kurserstellung ohne Informatikkentnisse - als auch die Lernenden - etwa durch individuellen Webspace, geteilte Ordner und Personalisierung von Kursen.

Weiters ist in der WeLearn-Plattform bereits multimedial aufbereitetes Kursmaterial eingebettet, wie beispielsweise die Kurse: "Propädeutikum aus Informatik", "Einführung in die

Betriebssysteme", "Java-Programmierung", etc. Die Inhalte liegen beispielsweise sowohl als Langtext (reiner Text und HTML), als auch als Folienpräsentation vor. Es gibt auch Möglichkeiten etwa mittels Lückentexten oder Multiple-Choice-Fragen sein Wissen zu einzelnen Themengebieten zu überprüfen bzw. durch Applets Simulationen und Experimente selbständig durchzuführen und so selbst die Lerninhalte zu erfahren.

WeLearn bietet Unterstützung für viele verschiedene Lernformen. Abbildung 1 zeigt ein typisches Beispiel für ein hybrides Lernmodell (Blended Learning):



Lernende sitzen wie üblich in einem Hörsaal, wo der Stoff vorgetragen und Fragen gleich direkt geklärt werden (Präsenzphasen).

Abbildung 1: Typisches WeLearn-Setting für ein hybrides Lernmodell

[Abb_Lernmodell]

Weiters werden die Lernenden durch vielfältiges Lernmaterial in ihren unterschiedlichen Lernweisen unterstützt (Langtext, multimedial aufbereiteter Text, Folienpräsentation, praktische Beispiele, kleine Tests, usw.). Die Lernenden können auch Teams bilden, in denen sich die Lernenden gegenseitig unterstützen, bzw. anderen Gruppen als Coaches oder Tutoren dienen, wobei sie von WeLearn als gemeinsame Kommunikationsplattform unterstützt werden. Eine weitere Austauschmöglichkeit zwischen den Lernenden aber auch zwischen Lehrenden und Lernenden bilden Foren, in denen sowohl Organisatorisches als auch Stoffspezifisches behandelt werden kann. Handouts unterstützen schließlich noch Lernende, die temporär ohne Online-Zugriff, bzw. technische Hilfsmittel lernen wollen.

Mehr Informationen über WeLearn und dessen Aufbau sind auch in ([Mue02a], [Mue02b]) zu finden und unter [FIM] kann ein Test-Account angefordert werden.

3 Das Model-View-Controller Paradigma

Da die englischen Begriffe Model, View und Controller auch in der deutschsprachigen Literatur üblich sind, werde ich sie in dieser Arbeit ohne Anführungszeichen im deutschen Text verwenden.

3.1 Geschichte

Das Model-View-Controller Paradigma wurde erstmals 1978/79 von Xerox für Smalltalk (eine der ersten objektorientierten Programmiersprachen) in die GUI-Programmierung (Graphical-User-Interface – Graphische-Benutzer-Schnittstelle) eingeführt [Ree]. Smalltalk war lange Zeit die favorisierte objektorientierte Programmiersprache im akademischen Bereich, konnte sich später aber aufgrund einiger Design-Schwächen nicht gegen Java behaupten (zum Beispiel keine privaten Methoden, keine Typprüfung beim Kompilieren, ...). In Java kommt das MVC-Paradigma sowohl bei der 'normalen' GUI-Programmierung (mittels einer Klassenbibliothek wie zum Beispiel Swing oder SWT) zum Einsatz, als auch bei Java-Server-Pages und Servlets, um die Applikationslogik vom HTML-Code zu trennen (wobei es natürlich vom Programmierer abhängt, in welchem Umfang er das MVC-Paradigma umsetzt).

Vorteile des MVC-Paradigmas sind beispielsweise übersichtlicherer Code, einfachere Wartung bzw. Änderungen und Austauschbarkeit (von Code-Teilen im Allgemeinen und der Komponenten im Speziellen. Dieses Thema wird später noch näher behandelt (siehe Kapitel 3.8 - *Vorteile und Nachteile des MVC-Paradigmas*).

3.2 Konzept

In den Anfangsjahren der Informatik wurde nur sogenannter Spaghetti-Code [Wiki/3] produziert, d.h. es erfolgte keine Strukturierung in Methoden, sondern es wurde nur mit Schleifen und Sprünge gearbeitet. Zusätzlich waren dabei Ausgabe, Eingabe und Verarbeitung völlig ineinander verwoben. Als die Programme immer länger und daher unübersichtlicher wurden, führte man als Abhilfe erst die strukturierte Programmierung (1970er) [Rec02/1, S.525ff] und schließlich die objektorientierte Programmierung (1980er) [Rec02/2, S.529ff] ein.

Nun war es möglich Programme zu strukturieren. Viele nutzten diese neue Möglichkeit anfangs nur, um Spaghetti-Code in mehrere Segmente aufzuspalten. Doch mit der Zeit setzte es sich durch, dass die Strukturierung auch nach Ausgabe, Eingabe und Verarbeitung vorgenommen

wurde. Zu dieser Zeit war die GUI-Programmierung noch nicht erfunden und die textorientierten Programme konnten eine Interaktionen nur entgegennehmen, wenn eine Methode darauf (blockierend) gewartet hat. Wie etwa folgendes Codebeispiel illustriert:

```
begin
  while(input != 'quit') do
    input = waitForInput();
    output = doSomething(input);
    write(output);
  end while;
end.
```

Bei Einführung der grafischen Benutzerschnittstelle und des Multi-Tasking (gleichzeitige Ausführung mehrerer Programme) wurde es notwendig eine andere Art der Eingabe-Verarbeitung zu finden. Es wurden nicht mehr an bestimmten Stellen Kommandos eingegeben, wie bei textorientierten Programmen, und auch das 'GUI', das zum Beispiel DOS anbot (es wurden einfache grafische Formen (Rechtecke, Kreise, Linien, Punkte) auf den Bildschirm gezeichnet und die Pixel-Position einer Mausinteraktion abgefragt), war nicht mit diesem neuen GUI vergleichbar. Es wurde nicht mehr ein Rechteck mit Text darin gezeichnet, das für das Programm selbst keine logische Bedeutung hatte, sondern es wurde ein Button mit einem Titel definiert. Eine Mausinteraktion wurde nicht mehr auf dem Pixel xy gemeldet, sondern auf dem definierten Button, der für die Weiterverarbeitung des Ereignisses nunmehr selbst zuständig ist.

Diese Art der Eingabe-Verarbeitung bezeichnet man als ereignisgesteuerte Programmierung [Rec02/3, S.802ff]. Eine Funktion, die auf eine Eingabe wartet und währenddessen den Computer blockiert, wurde ersetzt durch eine Methode, die erst durch ein Ereignis aufgerufen wird und die Verarbeitung dieses Ereignisses übernimmt.

Analog zur Differenzierung in Eingabe, Ausgabe und Verarbeitung in der Zeit der Konsolenanwendungen wird bei der ereignisgesteuerten GUI-Programmierung eine Differenzierung in Model, View und Controller vorgenommen, wobei Model der Verarbeitung, View der Ausgabe und Controller der Eingabe entspricht.

Durch diese Aufteilung wird es ermöglicht, dass leichter Änderungen an einem Teil vorgenommen werden können, ohne dass die anderen Teile dadurch beeinflusst werden. Das ist natürlich nicht uneingeschränkt möglich – so würden Änderungen an den Datentypen der

Models, bzw. Änderungen von Interfaces natürlich auch Änderungen am Controller und den Views nach sich ziehen – aber für Änderungen an den Views und der Applikationslogik sollte bei sauberer Umsetzung der Änderungsaufwand in den jeweils anderen Teilen gleich Null sein.

Ein weiterer großer Vorteil dieser Trennung ist die starke Modularität. Einerseits können dadurch Teile leichter bei anderen Projekten wiederverwendet werden und andererseits ist es möglich für dasselbe Model mehrere Views zu definieren und je nach Bedarf beliebig auf das Model anzuwenden. Dabei kann es sich um rein optisch unterschiedliche Darstellungen (anderes Layout, andere Farben, ...) handeln, aber natürlich auch um verschiedene Ausgabeformate (HTML ohne JavaScript, HTML mit JavaScript, Betriebssystem-GUI, ...). Weiters ist es damit auch möglich durch unterschiedliche Views eine nach Hardwareplattform differenzierte Behandlung des Models zu erreichen (zum Beispiel WML für Handys, einfaches HTML (nach Spezifikation 4.01 oder gar 3.2) für Handhelds und ältere Web-Browser und dynamisches XHTML für neuere Web-Browser).

3.3 Model

Das Model ist die zentrale Komponente im MVC-Paradigma, es kann intern ebenso wie Views und Controller leicht geändert werden, lediglich die Schnittstellen zu den Views und dem Controller müssen konstant bleiben.

Das Model enthält sämtliche Daten und die gesamte Applikationslogik. Zudem findet jegliche Kommunikation nach außen (außerhalb der Applikation) ebenfalls im Model statt, sowohl der Zugriff auf externe Daten (Datenbanken, Dateisysteme, etc.) als auch die Kommunikation mit anderen Applikationen. Obwohl der Applikationsstatus zu den Daten zählt, sollte hier jedoch explizit erwähnt werden, dass das Model auch für die Verwaltung des momentanen Applikationsstatus zuständig ist. Dies ist bei lokalen Applikationen nicht relevant, doch bei Web-Applikationen "übersteht" der Status einen Interaktionszyklus mit dem Benutzer nicht automatisch, sondern es muss vom Entwickler darauf geachtet werden, dass nach einem solchen Interaktionszyklus im korrekten Applikationsstatus fortgefahren wird.

Eine weitere wichtige Aufgabe des Models ist die Verwaltung der aktiven Views, die es zu benachrichtigen gilt, falls sich der Zustand des Model ändert. Dafür führt das Model eine Liste mit allen momentan aktiven Views, die bei einer Model-Änderung (wie Änderung von dargestelltem Inhalt) mittels eines Ereignisses über diese Änderung informiert werden. Auch hier gibt es bei Web-Anwendungen eine Besonderheit zu beachten. Lokale Anwendungen und auch

viele verteilte Anwendungen teilen eine Änderung des Models unmittelbar der View bzw. den Views mit, und die Darstellung wird für den Benutzer sofort angepasst. Bei Anwendungen, die als Web-Seiten im Browser dargestellt werden ist dies technologiebedingt nicht möglich, da der Server den Browser nicht über eine Änderung informieren und somit auch keine Aktualisierung der Anzeige erfolgen kann bis der Browser das nächste mal Kontakt zum Server aufnimmt. Mit HTTP 1.1 existiert zwar die Möglichkeit einer server-initiierten Kommunikation, wofür allerdings sowohl Browser als auch Server HTTP 1.1 unterstützen müssen und auch die Applikation von dieser Möglichkeit Gebrauch machen muss. Eine Lösungsmöglichkeit wäre, den Browser beispielsweise alle 60 Sekunden mittels Meta-Tags die Seite neu laden zu lassen. Dies schwächt das Problem allerdings nur etwas ab, da diese Lösung gleichzeitig vermehrtes Transfervolumen erzeugt. Weiters besteht hierbei das Problem, dass der Benutzer bei der Dateneingabe gestört werden könnte, wenn sich die Seite neu lädt. Das Problem des erhöhten Transfervolumens könnte man mittels Frames und JavaScript lösen, indem ein kleiner oder versteckter Frame immer wieder neu geladen wird, und bei Bedarf über eine JavaScript-Routine eine Aktualisierung des Hauptframes initiiert. Fatalerweise wirft dies allerdings wiederum das Problem auf, dass man durch die Verwendung von Frames und JavaScript nicht mehr alle Web-Browser unterstützen kann und Personen mit eingeschränkten Fähigkeiten - etwa Blinden – den Zugang erschwert oder gar unmöglich macht. Leider gibt es für dieses Problem keine allgemein gültige Lösung. Es muss von Fall zu Fall entschieden werden, welche Lösung am geeignetsten ist.

3.4 View

Die View ist die Komponente im MVC-Modell, die für die Ausgabe zuständig ist, und am häufigsten geändert wird. Views bilden eine Abstraktionsschicht zwischen der Repräsentation der Applikation, den Daten im Model und der Präsentation dem Benutzer gegenüber. Sie werden zur Erstellung von optisch als auch technisch unterschiedlicher Ansichten des selben Models verwendet. Einzig logisch unterschiedliche Sichten des Models müssen von diesem selbst verwaltet werden (zum Beispiel: für verschiedene Benutzer), da die View ja keine Kenntnisse über das Model hat und auch auf keine externen Daten zugreifen kann (bei verschiedenen Benutzern kann die View weder die Authentifizierung des Benutzers validieren, noch kann sie Entscheidungen treffen, welche Bereiche ein Benutzer sehen darf und welche nicht. Jede View ist in der Regel einem Model zugeordnet, kann aber von diesem Model beliebig oft instantiiert, und auch mit beliebigen anderen Views dieses Models kombiniert werden.

Die einfachste Methode eine neue View zu generieren, ist eine optisch unterschiedliche View mittels einer bestehenden zu erzeugen. Hierbei werden etwa Farben bzw. Bilder und eventuell das Layout angepasst.

Die komplexeste Methode ist eine neue View für ein Model zu entwickeln, die mithilfe einer neuen Technologie arbeitet. Ein Beispiel wäre ein Model mit HTML-View, für das noch eine WML-View geschaffen wird. In solchen Fällen ist es teilweise auch nötig den Controller anzupassen, da manchmal eine neue Technologie nicht die Möglichkeiten der Rückmeldung an den Controller bietet, die von der 'alten' View verwendet, und vom Controller verstanden wird.

3.5 Controller

Der Controller ist die Komponente im MVC-Modell, die dafür zuständig ist, die Eingaben aus verschiedensten Quellen entgegenzunehmen und standardisiert an das Model weiter zu leiten. Somit bildet der Controller eine zweite Abstraktionsschicht, die die Interaktionen, die der Benutzer an der View vornimmt, wieder in das Model zurückführt.

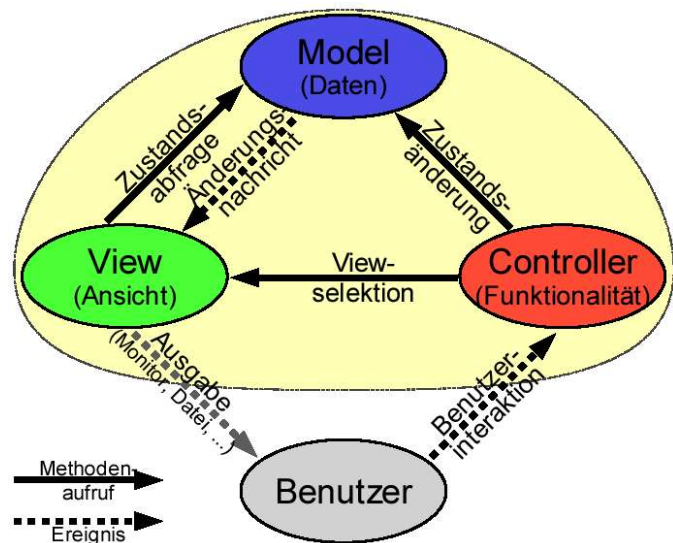
Er besitzt eine Instanz des Models, kennt alle dafür möglichen Views und ist auch für die Zuordnung der aktuellen verwendeten View(s) eines Models zuständig.

Der Controller hat im Idealfall, ebenso wie Model und View, kein Wissen über die Internas der anderen zwei Komponenten, sondern kennt ebenfalls nur die für ihn wichtigen Schnittstellen. Im Gegensatz zu diesem Idealfall hat der Controller in der Implementierungspraxis manchmal einen 'aktiveren' Part, indem er direkt in das Model eingreift und Änderungen darin vornimmt, was allerdings dem Prinzip widerspricht, die gesamte Logik im Model zu bündeln.

3.6 Kommunikation zwischen den Komponenten

Die Kommunikation zwischen den Komponenten erfolgt durch Methodenaufrufe und durch Ereignisse. Ereignisse werden hauptsächlich infolge von Benutzerinteraktionen erzeugt, können aber auch durch Änderungen des Models (etwa aufgrund eines zeitlich verzögerten Kommandos) hervorgerufen werden.

Abbildung 2 illustriert die möglichen Ereignisse und Methodenaufrufe der Kommunikation zwischen den drei MVC-Komponenten und dem Benutzer. Im Ablauf einer Kommunikation kommen an drei Stellen Methodenaufrufe und an zwei Stellen Ereignisse vor. Das dritte "Ereignis" in der Abbildung 2 – die Ausgabe von der View zum Benutzer – ist kein Ereignis im Sinn der ereignisgesteuerten Programmierung, es



bedeutet lediglich, dass den Benutzer eine erneuerte View angezeigt wird. Worauf in der Regel eine (Re)-Aktion des Benutzers folgt.

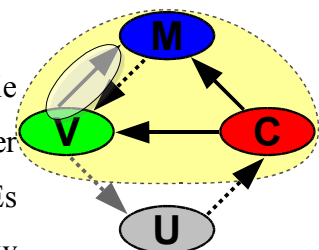
Abbildung 2 - Kommunikation zwischen den MVC-Komponenten

- Der Controller ist vom Ablauf der Interaktionen her die zentrale Komponente im MVC-Paradigma, verantwortlich für die Benutzerinteraktion und für die Verwaltung des Model und der Views.
- Das Model, die wichtigste Komponente des MVC-Paradigmas, ist für den eigentlichen Programmablauf (Änderung des Zustandes) und für die Benachrichtigung aller zugehörigen (aktiven) Views bei einer Änderung des Zustandes zuständig.
- Die View steht am Ende des Interaktionsverlaufs und ist für die Konvertierung des Datenmodells des Model (das von einem Ausgabemedium unabhängig ist) in die gewünschte Ausgabeform zuständig.

Im Folgenden werden die einzelnen Kommunikationsabläufe des MVC-Paradigmas näher diskutiert.

3.6.1 Zustandsabfrage: View → Model

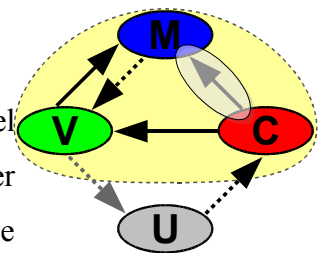
Die aktuelle View fragt alle Werte des Models ab, die für die Darstellung benötigt werden. Normalerweise wird dieser Kommunikationsteil bei jeder Erneuerung der Ausgabe durchgeführt. Es werden also alle Daten im Model gehalten und die Ausgabe der View wird neu generiert.



Aus Performance-Gründen wird bei manchen Implementierungen auch die letzte View zwischengespeichert und wiederverwendet, falls die Änderungen am Model für die View nicht relevant sind. Sollte es die Implementierung erlauben, ist es natürlich auch möglich die zwischengespeicherte View zu adaptieren um nur die geänderten Daten zu aktualisieren. Dabei sollte man aber beachten, dass dies bei einer Applikation, die sich bei dem Großteil der Interaktionen stark ändert, zu unnötigen Performance-Einbußen führen kann, da jedes mal geprüft werden muss, ob sich die alte View wiederverwenden lässt, jedoch ohne etwas Wiederverwendbares zu finden. In solch einem Fall wäre es vernünftig auf die Zwischenspeicherung der View zu verzichten.

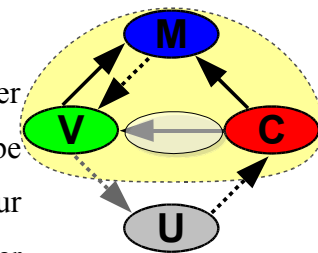
3.6.2 Zustandsänderung: Controller → Model

Hierbei ruft der Controller bei dem von ihm verwalteten Model Methoden zur Änderung von Parametern auf, um diese entsprechend der Benutzerinteraktion zu ändern. In weiterer Folge wird das Model diese Änderungen prüfen, entsprechende Zustandsänderungen durchführen und die View durch ein Ereignis über die Änderung informieren, damit von der View eine neue Ausgabe generiert wird.



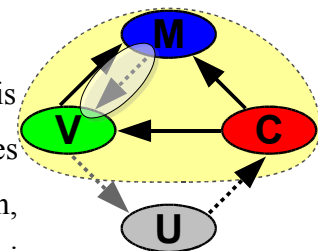
3.6.3 Viewselektion: Controller → View

Hierbei wird vom Controller (entweder per Standardwert oder durch Benutzerauswahl) eine View gewählt, die für die Ausgabe verwendet wird. Dieser Methodenaufruf erfolgt für gewöhnlich nur einmal zu Beginn einer Kommunikationssitzung. Wird die Änderung der gewählten View über die Applikation selbst ermöglicht, muss der Controller die zu verwendende View natürlich zuerst vom Model erfragen.



3.6.4 Änderungsnachricht: Model → View

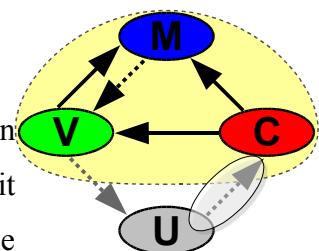
Hier wird für jede zugehörige (aktive) View ein Ereignis ausgelöst, um die Anpassung der Ausgabe an den geänderten Zustand des Models zu bewirken. Diese Anpassung erfolgt im Idealfall unverzüglich, was aber technologiebedingt nicht immer möglich ist. Während es bei einem normalen GUI kein Problem ist, mittels eines Ereignisses die sofortige Anpassung der Ausgabe zu bewirken, ist dies etwa bei HTML nicht möglich. Bei nur einem Benutzer wird dies kein Problem darstellen, da nach dem Senden der Benutzerinteraktion sowieso ein Ergebnis an den Browser geschickt und die Seite neu geladen wird, beim Mehrbenutzerbetrieb oder falls



zeitgesteuerte Zustandsänderungen auftreten, wird es aber unmöglich alle Ausgaben konsistent zu halten. Verändert zum Beispiel ein Benutzer einen Zustand, mit dem auch ein zweiter Benutzer arbeitet, bekommt der zweite Benutzer erst bei der nächsten Server-Interaktion eine Rückmeldung, dass dieser Zustand seit dem letzten Zugriff geändert wurde. Wenn die Implementierung nicht sorgfältig durchgeführt wurde, kann es passieren, dass der zweite Benutzer nicht einmal erfährt, dass ein Konflikt aufgetreten ist. Mit HTTP 1.1 existiert zwar die Möglichkeit einer server-initiierten Kommunikation, die allerdings von der Applikation initiiert werden muss. Außerdem müssen sowohl Browser als auch Server HTTP 1.1 unterstützen. Ein Ansatz zur Lösung dieses Problems wurde bereits in Kapitel 3.3 erläutert.

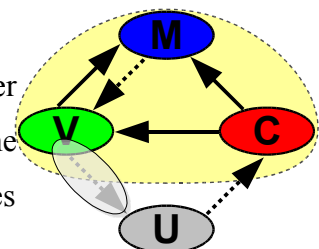
3.6.5 Benutzerinteraktion: Benutzer → Controller

Diese Interaktion verlässt den bisher betrachteten rein technischen Bereich des MVC-Paradigmas und schließt den Benutzer mit ein. Außer der ersten Benutzerinteraktion (Prozess starten) geschehen die meisten anderen als Reaktion auf die zuletzt angezeigte Ausgabe der View. Durch diese Interaktion wird ein Ereignis an den Controller übermittelt, der das Ereignis auswertet, und gegebenenfalls die Änderungen am Model vornimmt. Sollten die Benutzerinteraktion keine Änderung am Model bewirken wird normalerweise die letzte Ausgabe der View erneut gesandt, ansonsten wird eine neue Ausgabe generiert.



3.6.6 Ausgabe: View → Benutzer

Dieses 'Ereignis' ist genau genommen kein Ereignis im Sinn der ereignisgesteuerten Programmierung. Dabei wird für den Benutzer eine aktualisierte Anzeige erzeugt, worauf üblicherweise eine (Re)-Aktion des Benutzers folgt.



3.6.7 Kommunikation zwischen den Komponenten in der Realität

Dieser Ablauf der Kommunikation ist idealisiert und in der Praxis oft nicht anzutreffen. Bei einfachen Java-Server-Pages-Applikationen, mit nur einer View, ist es zum Beispiel üblich, dass Controller und View verschmolzen sind, und dass das Model (zum Beispiel eine Datenbank) die View nicht mit einem Ereignis informiert, sondern dem Benutzer die Änderung erst nach einer Interaktion mitgeteilt wird. Controller von MVC-Anwendungen, die mit XSL (XML Style Sheets)-Transformationen arbeiten, nehmen teilweise die Daten als XML-Baum

vom Model entgegen und die XSL-Transformation repräsentiert die View. Bei Web-Applikationen, die hierarchische MVC-Frameworks benutzen ist das Model außerdem nicht für das Neuinitiiieren des Applikationsstatus verantwortlich, da dies der Controller übernimmt.

3.6.8 Ablauf der Kommunikation zwischen den Komponenten

Am Beispiel einer Benutzeranmeldung (Abbildung 3):

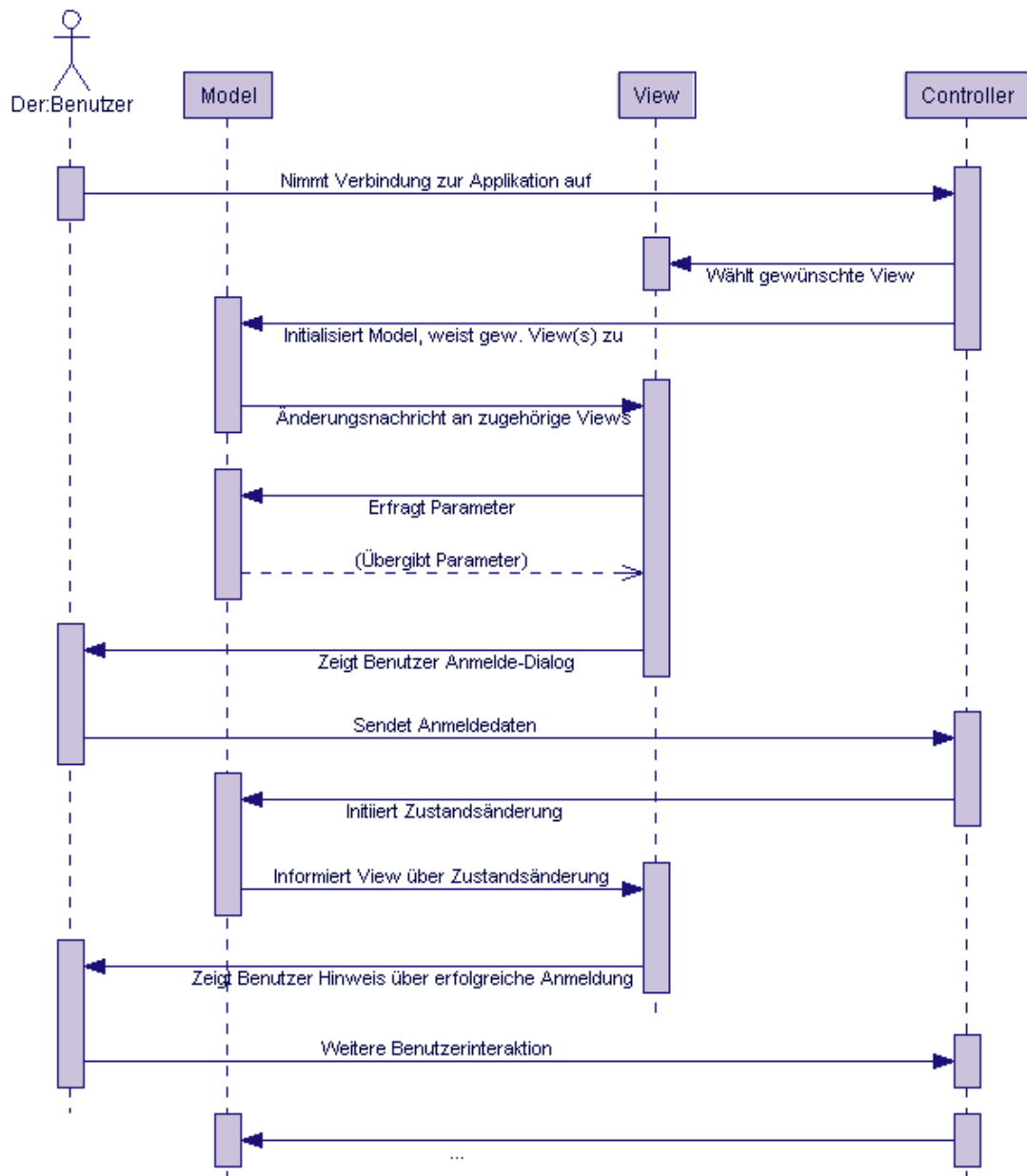


Abbildung 3: Beispielhafter Ablauf einer Benutzeranmeldung

3.7 Beispiel

Kurz ein praktisches Beispiel zur Erläuterung des MVC-Paradigmas. Dafür wurde eine simple Tabellenkalkulation gewählt:

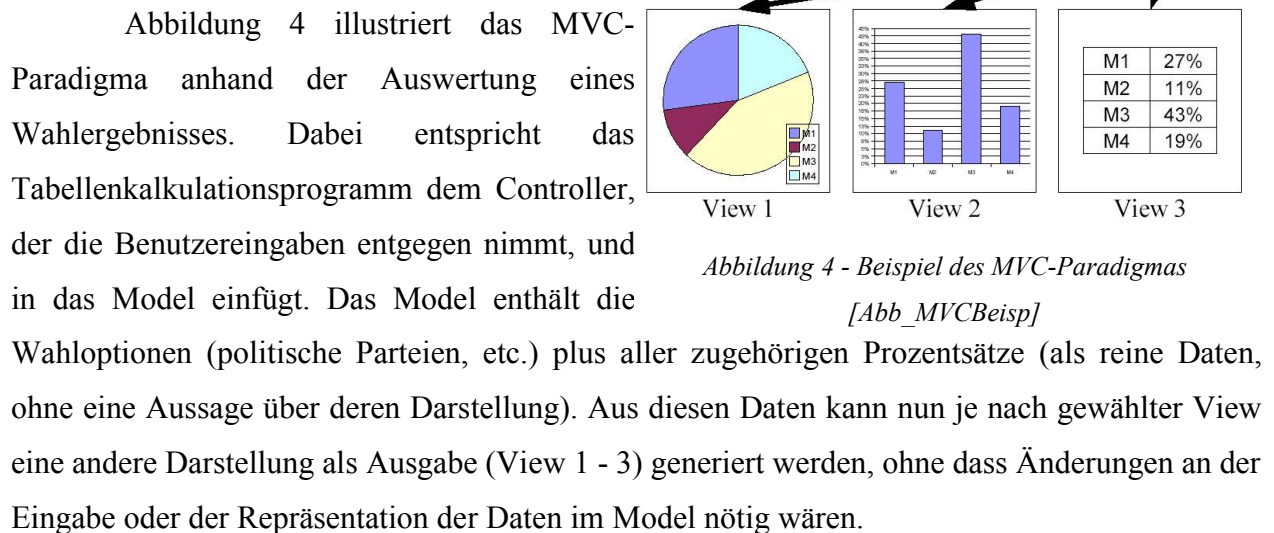


Abbildung 4 - Beispiel des MVC-Paradigmas

[Abb_MVCBeisp]

3.8 Vorteile und Nachteile des MVC-Paradigmas

Vorteile des MVC-Paradigmas sind zum Beispiel:

- Gut strukturiertes Design durch die Trennung von dem eigentlichen Applikations-Code und den zugehörigen Daten (Model), der Darstellung dem Benutzer gegenüber (View) und der Behandlung von Benutzerinteraktion (Controller).
- Durch diese Strukturierung ist es möglich Änderungen an einer Komponente vorzunehmen, ohne dass dies Auswirkungen auf die anderen beiden Komponenten hat. Dies ist natürlich nur dann möglich, wenn Änderungen nur die Interna einer Komponente betreffen und nicht die Schnittstellen dazwischen.
- Selbst das komplette Ersetzen einer Komponente ist problemlos möglich, falls die neue Komponente die gleichen Schnittstellen verwendet wie die alte.
- Dadurch ist es möglich einfach durch den Austausch der Views mehrere unterschiedliche Darstellungen des selben Models zu generieren.
- Ein weiterer Vorteil dieser Modularität ist, dass im Fehlerfall (fast immer) die Suche der Fehlerquelle auf eine der Komponenten beschränkt werden kann.
- Es ist auch möglich die Trennung von verschiedenen Benutzern (zum Beispiel Benutzer und Administrator) anhand von View und Controller zu implementieren, es ist allerdings

angeraten sich – aus Sicherheitsgründen – nicht nur darauf zu verlassen, sondern zumindest die Benutzerauthentifizierung im Model zu belassen.

- Die Komponenten können zusammen erweitert werden, wobei alte Views und Controller ein neueres Model verwenden können, solange die Schnittstellen nicht geändert werden.
- Das Konzept ermöglicht, dass die gleichen Daten von mehreren Personen auf unterschiedliche Weise betrachtet werden können.

Die Vorteile werden in der Regel die **Nachteile** überwiegen, jedoch:

- Es ist mehr Gründlichkeit bei der Planung und Implementierung erforderlich, um die Vorteile des MVC-Paradigmas nutzen zu können. Sollten zum Beispiel Daten noch über andere Wege als die Schnittstellen zwischen den Komponenten ausgetauscht werden, wird dies bei einer Änderung oder einem Austausch meist zu Problemen führen.
- Außerdem kann es bei kleinen Applikationen sein, dass der Mehraufwand für die Benutzung des MVC-Paradigmas dessen Vorteile bei der Entwicklung überwiegt.

3.9 Umsetzungen des MVC-Paradigmas im Java-Umfeld

Im Java-Umfeld gibt es mehrere Möglichkeiten, Web-Applikationen unter Verwendung des MVC-Paradigmas zu entwickeln.

- **Servlets** bieten zwar die meisten Möglichkeiten, allerdings auch den größten Aufwand bei der Entwicklung. Servlets sind eher für kleinere technische Aufgaben geeignet – zum Beispiel das Steuern von Downloads und Uploads oder für Sicherheitsüberprüfung. Für das Erstellen von komplexen Anwendungen sind Servlets meist zu aufwendig. Sollte man dafür trotzdem Servlets verwenden, ist es ratsam Hilfsklassen zu verwenden, die zum Beispiel das Verwenden von HTML-Templates vereinfachen.
- **Java-Server-Pages** sind gut geeignet für kleinere Web-Applikationen. Da bei Java-Server-Pages, wie bei PHP, der Java-Code zwischen dem HTML-Code steht, werden Java-Server-Pages bei größeren Anwendungen schnell unübersichtlich. Um dem vorzubeugen sollte man Teile der Seite in eigene Java-Server-Pages auslagern und per 'include' einbinden.
- **eXtensible-Server-Pages** vergleichbar mit Java-Server-Pages, allerdings sind es XML Dokumente. Durch die Verwendung von XML-Tags, die erst später in HTML, WML oder ein anderes Format umgewandelt werden, werden die Dokumente etwas

übersichtlicher, da keine Formatierungsdetails mehr enthalten sind.

- Beim **MVC-Paradigma** erfolgt eine Trennung in Model (Daten, Programm), View (Anzeige) und Controller (Eingabe). Bei vielen MVC-Frameworks wird ein zentrales Servlet verwendet, das als Controller agiert, der den Programmablauf mittels des Models durchführt, und dann das Ergebnis an eine Server-Page zur Aufbereitung der Ausgabe übergibt. Meist wird man solch ein MVC-Framework nicht selbst implementieren, sondern ein fertiges verwenden. Struts [Struts] von der Apache Software Foundation [Apache] ist ein solches MVC-Framework für Java-Server-Pages. Als weiteres Beispiel wäre Cocoon [Cocoon] zu nennen (ebenfalls von der Apache Software Foundation). Dieses ist umfangreicher als Struts, jedoch für eXtensible-Server-Pages bestimmt.
- Als fünfte Möglichkeit gibt es noch **hierarchische Model-View-Controller**, die eine grundsätzliche Aufteilung besitzen wie beim herkömmlichen MVC-Paradigma, allerdings mittels einer Klassen-Bibliothek verwendet werden, ähnlich der Programmierung konventioneller GUIs. Dabei ist es zum Beispiel möglich ein Menü als eine Komponente zu definieren und zu verwenden. Bei der Definition der Menüs einer Seite muss man nur mehr diese eine Komponente einfügen und parametrisieren. Bei einer Interaktion mit einem Knoten dieses Menüs wird ein Ereignis direkt an diese Menü-Komponente gesendet, die dann selbst für die weitere Verarbeitung zuständig ist. Diese Art der Web-Programmierung ist bei großen Applikationen am effektivsten, da man den wenigsten Overhead durch Beschäftigung mit Darstellungsdetails hat. Der Programmierer kommt weder mit der Verwaltung von Sitzungen, noch dem Wiederherstellen des Applikationsstatus nach einer Benutzerinteraktion in Berührung. Darüber hinaus ist es theoretisch möglich einen GUI-Builder zu verwenden, falls dieser das jeweilige Komponenten-Modell unterstützt, wobei mir allerdings keiner bekannt ist, der ein hierarchisches MVC-Komponenten-Modell für Web-Anwendungen implementiert.

Welche der vorgestellten Varianten man für ein Projekt verwendet hängt von mehreren Faktoren ab:

- Wissen der Entwickler: Verfügen die Entwickler ausschließlich über Java-Kenntnisse, ist es besser ein hierarchisches MVC-Framework zu verwenden. Bei weiterführenden Kenntnissen über HTML oder XML sind die nachstehenden Faktoren ausschlaggebend.

- Anforderungen an Stabilität und Ausfallsicherheit: Stellt man hier hohe Ansprüche, sollte man Servlets oder Server-Pages verwenden (eventuell in Zusammenarbeit mit einem gut getesteten MVC-Framework), da diese am Ausgereiftesten sind, insbesondere die hierarchischen MVC-Frameworks sind derzeit ziemlich neu und oft nicht ausführlich in der Praxis getestet.
- Komplexität der Oberfläche der Applikation: Bei sehr komplexen Layouts wird es mit hierarchischen MVC-Frameworks schnell umständlich, diese in die Realität umzusetzen. In diesem Fall wären Servlets oder Java-Server-Pages besser geeignet (eXtensible-Server-Pages bieten durch den Zwischenschritt der XSL-Transformation auch nicht mehr die Flexibilität von Java-Server-Pages). Bei komplexen Applikationen ist die Verwendung eines MVC-Frameworks (normal oder hierarchisch) zu bevorzugen. Zumindest sollte man die Server-Pages bzw. Servlets sehr sauber gliedern, da diese sonst mit der Zeit unübersichtlich werden.

Folgende Tabelle zeigt eine kurze Zusammenfassung der Faktoren [MVCFrmwrks]:

	Servlet	JSP	XSP	MVC	H-MVC
Komplexität	mittel	niedrig	hoch	hoch	sehr hoch
Abstraktionsgrad	niedrig	mittel	hoch	hoch	sehr hoch
Reife	hoch	hoch	mittel	mittel	niedrig
Performance	hoch	hoch	mittel - schlecht	mittel	mittel
Standardisierung	Finale Spezifikation	Finale Spezifikation	-	-	sehr frühe Phase
Notwendige Kenntnisse	HTML, HTTP, Java	HTML, (Java)	XML, XSLT, (Java)	HTML oder XML, Java	Java
Einstiegshürde	mittel	niedrig	hoch	hoch	hoch
Produktivität	gering	mittel	hoch	hoch	hoch

3.10 Fazit

Das Model-View-Controller-Paradigma ist eine wichtige Methode in der Programmierung, die es erlaubt modulare, leichter wartbare bzw. erweiterbare Applikationen mit beliebig austauschbaren Benutzerschnittstellen zu schreiben, die leichter an verschiedene Layouts

bzw. Technologien angepasst werden können.

Abgesehen von einigen wenigen Einzelfällen (zum Beispiel schlechte Umsetzungsmöglichkeit in der gewählten Programmiersprache, kleines Projekt, etc.) gibt es für ein Projekt keinen Grund, die Applikation nicht aufgrund des MVC-Paradigmas zu entwickeln, da die dadurch erhaltenen Vorteile die wenigen Nachteile bei weitem überwiegen.

4 Millstone

Das Unternehmen IT Mill [ITMill] wurde im Jahr 2000 gegründet, um eine neue Technologie zur Erstellung von Internetapplikationen zu entwickeln, die das Entwickeln von großen Unternehmensapplikationen erleichtern sollte. Auf Basis der J2EE-Plattform - nachdem man sich gegen ein proprietäres Framework entschieden hatte - wurde Millstone entwickelt. Die ersten beiden Versionen (2001 und 2002) waren nur für interne Projekte bestimmt und laut IT Mill auch nur für die Verwendung durch Programmierer mit guten Java-Kenntnissen geeignet. Version drei ist teilweise von Version zwei abgeleitet. Es wurden jedoch viele Änderungen zur einfacheren Verwendbarkeit vorgenommen, damit auch weniger erfahrene Programmierer damit umgehen können. Weiters wurde das Millstone-Projekt mit Version drei ein Open-Source-Produkt.

4.1 Idee

Bei der Entwicklung herkömmlicher Webapplikationen, zum Beispiel mittels Perl- oder PHP, oder auch Java-Servlets bzw. JavaServerPages kommt man als Applikationsentwickler auch immer mit dem endgültigem HTML-Code in Berührung. Dies bringt mehrere Nachteile mit sich. Erstens muss bzw. sollte der Applikationsentwickler auch HTML verstehen, um zu wissen wo er die Code-Templates einzufügen hat. Zweitens ist es fehleranfälliger, da der HTML-Code problemlos im Programm geändert werden kann, was zu fehlerhafter Darstellung oder invalidem Code führen kann. Und drittens ist es zeitaufwändiger als die Verwendung von fertigen Bibliotheken, da man sich um viele Details selbst kümmern muss. Der einzige große Vorteil besteht darin, dass man an jeder Stelle maximale Eingriffsmöglichkeit in den generierten Code hat, was aber wie gesagt nur dann ein Vorteil ist, wenn man über genügend HTML-Kenntnis verfügt.

Auch die Verwendung von Templates ist nicht der Weisheit letzter Schluss, da Templates oft nur für die wichtigsten Fälle (grundsätzlicher Seitenaufbau, Menüs, häufigste Designelemente, ...) sinnvoll verwendet werden können bzw. in dynamischen Umgebungen die Templates in vielen Fällen nicht (ohne größeren Aufwand) dynamisch genug gemacht werden können. Außerdem ist die Verwendung komplexer Templates eine weitere Fehlerquelle, ähnlich der direkten Verwendung des HTML-Codes.

Der größte Unterschied zwischen einer herkömmlichen Web-Applikation und einer

lokalen Applikation ist allerdings, dass bei einer lokalen Applikation die Variablen über eine Benutzerinteraktion hinweg erhalten bleiben. Bei einer Web-Applikation, wird nach jeder Benutzerinteraktion die Applikation erneut aufgerufen und neu initialisiert. Sollten Daten benötigt werden, die bei dem letzten Aufruf verwendet wurden, muss man sich selbst um eine Sicherung (Dateisystem, Datenbank, ...) kümmern, bzw. alle eventuell benötigten Daten über den Benutzer mittels versteckter Felder an die neue Instanz weiterleiten. Beide Lösungen sind allerdings zeitaufwändig bei der Umsetzung und auch eine zusätzliche Fehlerquelle.

Die Idee für Millstone war eine Anwendungsbibliothek zu schaffen, die genau so einfach zu verwenden ist wie eine gewöhnliche GUI-Bibliothek (zum Beispiel Swing oder SWT) [Swing][SWT]. Diese Bibliothek, bzw. das dahinter liegende Framework sollte die gesamte Abwicklung der Benutzerinteraktion vornehmen, damit sich der Programmierer nur auf die eigentliche Applikationslogik konzentrieren kann. Weiters kümmert sich das Framework um eine Vorverarbeitung der Benutzerinteraktion und um das Neuinitialisieren der Variablen auf die Werte vor der Benutzerinteraktion, so dass der Applikationsprogrammierer keinen Unterschied zu einer normalen GUI-Applikation erkennen kann.

4.2 Umsetzung

Millstone ist in drei Komponenten gegliedert:

- Die BaseLib-Bibliothek, die sämtliche UI-Objekte bereitstellt, und mit deren Hilfe die Applikationen geschrieben werden.
- Den WebAdapter, der die HTTP-Parameter vom Servlet-Container entgegen nimmt, für die Applikation aufbereitet und das Ergebnis der XSL-Transformation wieder an den Browser zurückschickt.
- Die Themes mit deren Hilfe der WebAdapter die XML-Ausgabe der BaseLib (per XSL-Transformation) in einen für das jeweilige Terminal (in der Regel einen Web-Browser) verständlichen Code umwandelt (zum Beispiel HTML).

In diesem Unterkapitel wird nur die grundlegende Funktionsweise von Millstone beschrieben, da eine detaillierte und komplette Beschreibung den Rahmen dieser Arbeit sprengen würde. Im nächsten Unterkapitel - *Einblicke* - wird aber auf einige ausgewählte Bereiche näher eingegangen. Wer sich näher informieren will, kann die API und den Quellcode lesen [MSAPI] [MSSource], eine weitere Dokumentation existiert nicht bzw. ist nicht öffentlich zugänglich.

Millstone kann mit dem gesamten Java Application Environment zusammenarbeiten (etwa mit Enterprise-Java-Beans, Servlets oder JDBC-Datenbanken), ist aber von allen Techniken unabhängig. Es wird nur ein Servlet-Container benötigt, in dem Millstone läuft und der die eigentliche Kommunikation mit dem Web-Browser vornimmt. Abbildung 5 soll dies illustrieren.

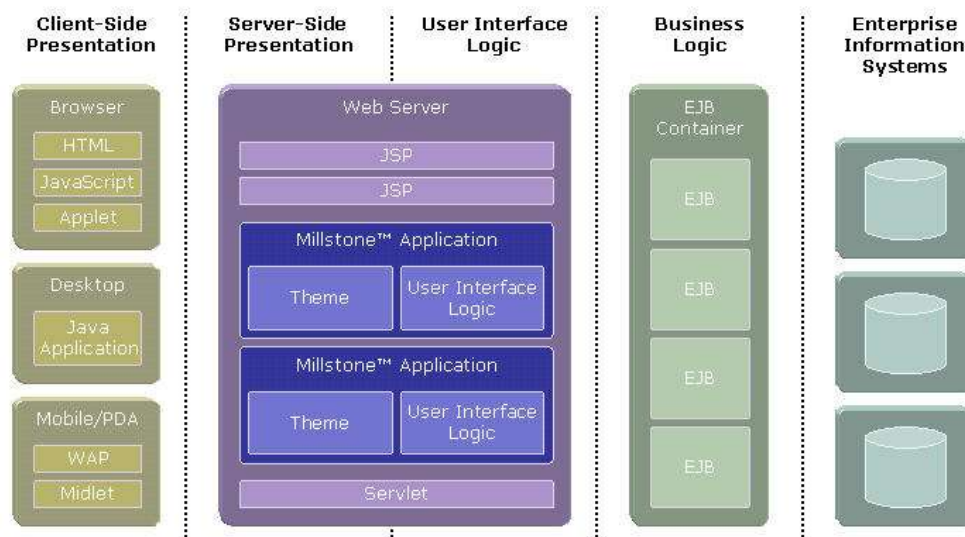


Abbildung 5 - Millstone in der Java Enterprise Edition Architektur [Abb_MilstJEEA]

Bei einer Millstone-Applikation ist es, im Vergleich zu einer herkömmlichen Web-Applikation, die mit Servlets gebaut wurde, nicht notwendig den momentanen Status bei jedem Request zu reinitialisieren und in der Session mitzuführen. Diese Arbeit wird im Hintergrund vom Millstone-Framework erledigt, wodurch man die Variablen, wie in einer lokalen Applikation verwenden kann.

Dies ist auf der einen Seite ein großer Vorteil für den Entwickler, da er weder HTML noch eine Script-Sprache (Perl, PHP, Python, Ruby, ...) beherrschen muss, sondern nur Java. Er kann sich durch die Abstraktion durch die BaseLib völlig auf die eigentliche Applikationslogik konzentrieren und muss sich nicht mit dem Design der HTML-Seite beschäftigen.

Auf der anderen Seite ist es aber auch ein Nachteil, da er zusätzlich zu den wenigen Möglichkeiten, die die BaseLib-Klassen an Manipulationen an der HTML-Ausgabe zulassen (einige Klassen erlauben das Definieren von Breite und Höhe, bzw. einiger einfacher HTML-Tags in ihrem Inhalt), keinerlei Einflussnahme auf das endgültige Erscheinungsbild der

Applikation hat – zumindest nicht aus der Millstone-Applikation selbst.

Um das Erscheinungsbild einer Applikation völlig nach seinen eigenen Vorstellungen zu gestalten, muss der Entwickler die XSL-Templates selbst adaptieren, wofür Kenntnisse über XML und XSL nötig sind. Außerdem ist es zum Beispiel - im Vergleich zu einer herkömmlichen Web-Applikation – nur umständlich möglich allen Tabellen ein einheitliches Layout zu geben, mit Ausnahme von einigen Tabellen, die jeweils mit einem anderen Layout dargestellt werden sollen. Bei einer herkömmlichen Applikation genügte es, einer Tabelle einfach andere CSS-Parameter (Cascading Style Sheets – die Methode um HTML (standardkonform) zu formatieren) zuzuweisen. Bei einer Millstone-Applikation ist es notwendig, für jede dieser Tabellen einen eigenen 'Style' in den XSL-Dateien einzufügen. Dadurch wird der Umfang der Applikation weit mehr vergrößert, als bei einer entsprechende Änderung einer herkömmlichen Web-Applikation. Weiters muss man XML und XSL besser beherrschen, um eine solche Anpassung vornehmen zu können, als dies bei HTML bzw. CSS der Fall wäre, da Web-Browser bei HTML (nach Spezifikation 3.2 und 4.01) für gewöhnlich ziemlich fehlertolerant sind und kleine Fehler das Ergebnis optisch kaum beeinflussen, während bei XML (und je nach Browser auch XHTML) bereits kleine Fehler einen Abbruch des Transformationsprozesses verursachen.

Dieser Transformationsprozess wird bei Millstone vom WebAdapter verwaltet, indem dieser mittels dem gewählten XSL-Theme eine Transformation des von der BaseLib gelieferten XML-Codes durchführt und das Ergebnis mittels eines Adapters an das jeweilige Terminal übermittelt. Dieser Terminal-Adapter ist in weiterer Folge auch dafür zuständig, die Rückmeldung des Terminals in für Millstone verständliche Kommandos umzuwandeln.

Neben den bereits erwähnten Komponenten einer Millstone-Applikation (BaseLib, Terminal-Adapter und Themes) gibt es noch Schnittstellen zur Datenanbindung – zum Beispiel zur Anbindung einer Datenbank. Und schließlich enthält eine Millstone-Applikation natürlich noch die eigentliche Applikationslogik.

Bei allen Komponenten (BaseLib, Datenanbindung, Terminal-Adapter und Themes) ist es möglich, die bestehenden Klassen durch eigene Klassen zu erweitern. So wurde zum Beispiel im Rahmen dieser Diplomarbeit die Tabelle sortierbar gemacht und eine ImageMap hinzugefügt (Hierauf wird im Detail in Kapitel 5 eingegangen).

Abbildung 6 soll diese Zusammenhänge näher illustrieren.

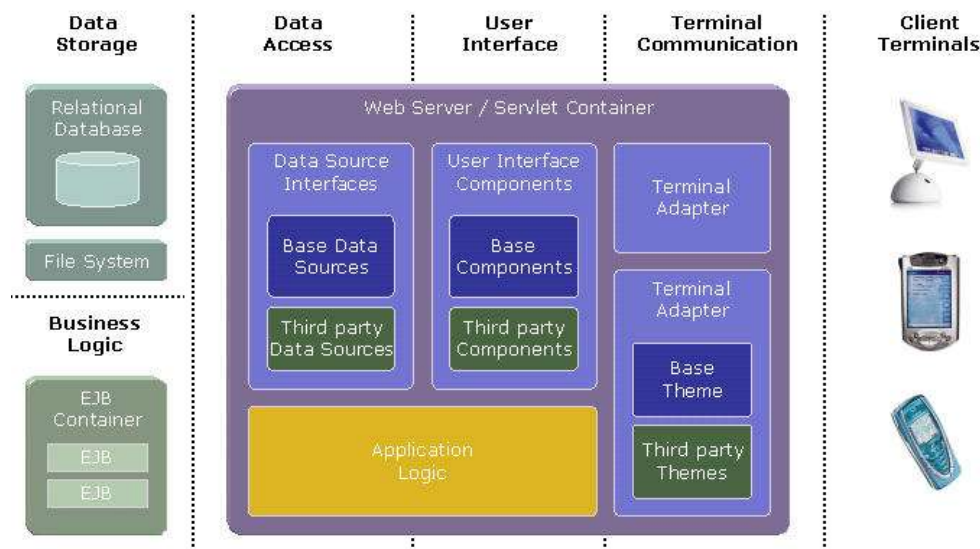


Abbildung 6 - Millstone Applikationsstruktur [Abb_MilstAppstr]

Die **BaseLib** enthält das Komponentenmodell von Millstone, mit dem alle Millstone-Applikationen umgesetzt werden. Die zentrale Klasse ist dabei die 'Application'-Klasse, von der alle Applikation abgeleitet sind. Zudem sind noch die Benutzerschnittstellen-Komponenten enthalten und zusätzliche Hilfs-Klassen. Die wichtigsten Benutzerschnittstellen-Komponenten sind in Kapitel 5.2.1 aufgelistet.

Zu Details zu den einzelnen Benutzerschnittstellen-Komponenten finden sie im *Appendix A– Benutzerhandbuch für das modifizierte Millstone*.

Die **Schnittstellen für die Datenanbindung** ermöglichen es zum Beispiel den Inhalt einer Tabelle direkt mit einer Datenbank zu verknüpfen, so können die Inhalte der Datenbank direkt in die Tabelle übernommen werden und auch Änderungen an der Tabelle automatisch mit der Datenbank abgeglichen werden. Weiters enthalten die Benutzerschnittstellen-Komponenten Schnittstellen um mit anderen Benutzerschnittstellen-Komponenten verknüpft zu werden.

Der **Terminal-Adapter** (WebAdapter) bietet der Millstone-Applikation bzw. den BaseLib-Klassen eine einheitliche Schnittstelle zu den Web-Browsern, indem er verschiedene Implementierungseigenheiten der verschiedenen Browser abfängt, die registrierten Variablen aus den Benutzerschnittstellen-Komponenten in Parameter der HTML-Seite umwandelt und die vom Web-Browser übermittelten HTTP-Parameter den Komponenten wieder in der ursprünglichen Benennung zur Verfügung stellt.

Themes sind dafür zuständig, die von den Benutzerschnittstellen-Komponenten gelieferte XML-Repräsentation der Ergebnisseite in die gewünschte Ausgabeform zu bringen. Dafür wird eine XSL-Transformation auf die XML-Repräsentation durchgeführt. Mit dem Standard-Millstone-Theme liefert diese Transformation HTML (4.0) es ist aber ohne weiteres möglich damit auch WML (für Handys) oder jedes andere Format zu generieren, das sich per XSL-Transformation erzeugen lässt. In dem Standard-Millstone Theme wird kaum eine Differenzierung nach JavaScript-fähigen bzw. nicht JavaScript-fähigen Web-Browsern vorgenommen. Weiters ist diese Zuweisung für manche Browser fehlerhaft, da sie JavaScript nicht genügend unterstützen, um die verwendeten JavaScript-Befehle zu verstehen. In anderen Fällen fehlen die JavaScript-freien Alternativen entweder ganz oder arbeiten nicht korrekt.

4.3 Vorteile und Nachteile von Millstone

Dadurch, dass eine Millstone-Applikation wie eine herkömmliche lokale Applikation programmiert wird ergeben sich einige Vorteile:

- Einfachere Programmierung – Der Entwickler muss sich nicht um Sitzungsparameter, die Wiederherstellung des Applikationsstatus nach einer Benutzerinteraktion und die Weiterleitung von Parametern aus der Instanz vor der letzten Benutzerinteraktion kümmern. Das wird vom WebAdapter erledigt.
- Man muss sich in der Applikation nicht um mögliche Einschränkungen einzelner Browser kümmern. Das wird von den Themes und dem WebAdapter erledigt.
- Durch die bessere Code-Strukturierung steigt die Wiederverwendbarkeit von Code-Teilen - ermöglicht durch das objektorientierte Benutzerschnittstellen-Komponentenmodell.
- Man benötigt nur Kenntnisse in einer Programmiersprache – Java. (Bei anderen Lösungen, wie in JSP, Perl, PHP, usw. benötigt man fast immer auch Kenntnisse von HTML und CSS).

Der einzige wirkliche Nachteil von Millstone ist, dass Flexibilität bei dem Aussehen einzelner Applikation umständlicher zu realisieren ist als wenn man direkt aus der Applikation Zugriff auf den endgültigen HTML-Code hat. Ein weiterer Nachteil für kleinere Projekte ist der benötigte Servlet-Container, da viele Shared-Hosting-Anbieter zwar Perl und PHP unterstützen, aber keine Java-Umgebung (wegen des höheren Ressourcenverbrauchs). Virtual- bzw. Dedicated-

Hosts sind allerdings meist wesentlich teurer.

Es ist daher eine Frage der Umstände ob eine komponenten-basierte Entwicklung (zum Beispiel Millstone) oder eine herkömmliche Entwicklung (JSP, Perl, PHP, usw.) effizienter ist. Bei großen aber auch mehreren kleinen Applikation mit einem einheitlichen Layout ist wahrscheinlich eine komponenten-basierte Entwicklung besser. Dies gilt auch wenn die Entwickler nur Java aber kaum HTML beherrschen. Bei Entwickler, die auch HTML gut beherrschen, und bei Bedarf nach maximaler Einflussmöglichkeit auf den Code (etwa zur Optimierung einzelner Seiten der Applikation ist man mit einem herkömmlichen Ansatz besser beraten.

4.4 Einblicke

In diesem Unterkapitel wird auf einige ausgewählte Funktionsweisen von Millstone näher eingegangen, um exemplarisch zu zeigen, wie Millstone intern funktioniert. Diese Bereiche sind:

- Die Übermittlung von Parametern von den Benutzerschnittstellen-Komponenten an den Web-Browser.
- Die Übermittlung von Parametern vom Web-Browser an die Benutzerschnittstellen-Komponenten.
- Die Funktionsweise einer einfachen Benutzerschnittstellen-Komponenten am Beispiel der Button-Klasse.

4.4.1 Übermittlung von Parametern von den Benutzerschnittstellen-Komponenten an den Web-Browser

Die Übermittlung von Parametern von den Benutzerschnittstellen-Komponenten an den Web-Browser erfolgt mittels der `paintContent`-Methode der jeweiligen Benutzerschnittstellen-Komponente. Dabei wird von der Komponente zuerst eine XML-Darstellung erzeugt, die in weiterer Folge vom `WebAdapter` mittels eines XSL-Themes in das gewünschte Endergebnis umgewandelt wird. Für dieses Beispiel wird die Methode der `Select`-Klasse verwendet, da diese alle wichtigen Befehle enthält, im Umfang aber übersichtlich bleibt.

```
/** Paint the content of this component.  
 * @param event PaintEvent.  
 * @throws PaintException The paint operation failed.  
 */  
public void paintContent(PaintTarget target) throws PaintException {
```

```
// Paint field properties
super.paintContent(target);
```

Ruft die `paintContent`-Methode der Klasse auf, von der diese Klasse abgeleitet ist, um die geerbten Attribute zu setzen – in diesem Fall wird durch das `super.paintContent` in der `Select`-Klasse die `paintContent`-Methode in der `AbstractField`-Klasse aufgerufen.

```
// Paint select attributes
if (isMultiSelect())
    target.addAttribute("selectmode", "multi");
if (isNewItemsAllowed())
    target.addAttribute("allownewitem", true);
```

Bestimmt, ob Mehrfachauswahl bzw. das Hinzufügen einer eigenen Option erlaubt ist. In weiterer Folge werden noch die verschiedenen Optionen in das XML-Ergebnis eingefügt.

```
// Paint options and create array of selected id keys
String[] selectedKeys;
if (isMultiSelect())
    selectedKeys = new String[((Set) getValue()).size()];
else
    selectedKeys =
        new String[(
            getValue() == null
                && getNullSelectedItemId() == null ? 0 : 1)];
```

Füllt ein String-Array mit den bereits selektierten Optionen

```
int keyIndex = 0;
target.startTag("options");
```

Markiert im XML-Ergebnis die folgenden Einträge als mögliche Optionen des Auswahlfeldes. Und fügt in weiterer Folge jede der Optionen einzeln hinzu.

```
for(Iterator i = getItemIds().iterator(); i.hasNext();) {

    // Get the option attribute values
    Object id = i.next();
    String key = itemIdMapper.key(id);
    String caption = getItemCaption(id);
    Resource icon = getItemIcon(id);

    // Paint option
    target.startTag("so");
```

Zeigt den Beginn der Definition einer Option an.

```
if (icon != null)
    target.addAttribute("icon", icon);
target.addAttribute("caption", caption);
target.addAttribute("key", key);
if (isSelected(id)) {
    target.addAttribute("selected", true);
    selectedKeys[keyIndex++] = key;
}
```

Fügt ein eventuell definiertes Icon ein, definiert die Beschriftung und den Rückgabewert der Option und schließlich wird noch das Attribut `selected` gesetzt, falls diese Option selektiert dargestellt werden soll.

```
target.endTag("so");
```

Schließt eine Option.

```
}
target.endTag("options");
```

Schließt die Definition der möglichen Optionen.

```
// Paint variables
target.addVariable(this, "selected", selectedKeys);
```

Definiert eine Variable, die die selektierten Optionen enthält, und aus der nach der Rückmeldung des Browsers Millstone ermittelt, ob sich an der Auswahl etwas geändert hat.

```
if (isNewItemsAllowed())
    target.addVariable(this, "newitem", "");
```

Definiert eine Variable für eine zusätzliche Option, falls dieses erlaubt ist.

```
}
```

Hiermit endet die Abbildung dieser Komponente in das XML-Ergebnis.

Unterschied von Attributen und Variablen:

- Attribute dienen zur Manipulation des optischen Erscheinungsbildes der Ergebnisseite,
- Variablen dienen hauptsächlich zur Rückgabe von Parametern vom Web-Browser an die Benutzerschnittstellen-Komponenten (können aber auch das Aussehen beeinflussen).

4.4.2 Übermittlung von Parametern vom Web-Browser an die Benutzerschnittstellen-Komponenten

Die Übermittlung von Parametern vom Web-Browser an die Benutzerschnittstellen-Komponenten erfolgt mittels der `changeVariables`-Methode der jeweiligen Benutzerschnittstellen-Komponente. Dabei fragt die Komponente die sie betreffenden Variablen aus der übergebenen Sammlung von Variablen aus, prüft ob ein Parameter geändert wurde und reagiert dementsprechend. Entsprechend dem vorherigen Unterkapitel - *Übermittlung von Parametern von den Benutzerschnittstellen-Komponenten an den Web-Browser* – wird als Beispiel wieder die entsprechende Methode der Select-Klasse verwendet.

```
/** Invoked when the value of a variable has changed.
 * @param event Variable change event containing the information about
 * the changed variable.
```

```

*/
public void changeVariables(Object source, Map variables) {
    // Try to set the property value
    try {

```

Folgende Code-Zeilen behandeln eine eventuell vorhandene selbst definierte Option:

```

// New option entered (and it is allowed)
String newitem = (String) variables.get("newitem");
if (newitem != null && newitem.length() > 0) {

```

Eine selbst definierte Option ist erlaubt und wurde eingegeben.

```

// Check for readonly
if (isReadOnly())
    throw new Property.ReadOnlyException();

```

Sollte das Feld schreibgeschützt gewesen sein muss ein Fehler aufgetreten sein, da ein Feld für eine zusätzliche Option entweder ganz fehlen kann, oder beschreibbar sein muss.

```

// Add new option
if (addItem(newitem) != null) {

```

Versucht die zusätzliche Option zu den bereits bestehenden hinzuzufügen, gelingt dies nicht implizit, wird es noch einmal explizit mit der ID des Auswahlfeldes versucht, sollte es auch damit nicht möglich sein die neue Option einzufügen wird eine Exception geworfen.

```

// Set the caption property, if used
if (getItemCaptionPropertyId() != null)
    try {
        getContainerProperty(
            newitem,
            getItemCaptionPropertyId()).setValue(
                newitem);
    } catch (Property.ConversionException ignored) {
        // The conversion exception is safely ignored,
        // the caption is just missing
    }
}

```

Folgende Code-Zeilen behandeln eine eventuell vorhandene selbst definierte Wahlmöglichkeit:

```

// Selection change
if (variables.containsKey("selected")) {
    String[] ka = (String[]) variables.get("selected");

```

Entnimmt das String-Array der selektierten Optionen aus der übergebenen Sammlung von Variablen (muss immer enthalten sein, außer es sind bei der XSL-Transformation oder im Web-Browser schwere Fehler aufgetreten).

Folgende Code-Zeilen beschreiben den Fall dass mehrere Optionen selektiert werden können:

```
// Multiselect mode
if (isMultiSelect()) {

    // Convert the key-array to id-set
    LinkedList s = new LinkedList();
    for (int i = 0; i < ka.length; i++) {
        Object id = itemIdMapper.get(ka[i]);
        if (id != null && containsId(id))
            s.add(id);
        else if (
            itemIdMapper.isNewIdKey(ka[i])
            && newItem != null
            && newItem.length() > 0)
            s.add(newItem);
    }
}
```

Obiger Code bestimmt alle selektierten Optionen. In weiterer Folge werden alle sichtbaren Optionen (es kann auch unsichtbare geben, die zwar in der Programmlogik verarbeitet werden, im Web-Browser aber nicht angezeigt werden) deselektiert und schließlich die vom Browser übermittelten neu selektiert.

```
// Limit the deselection to the set of visible items
// (non-visible items can not be deselected)
Collection visible = getVisibleItemIds();
if (visible != null) {
    Set newset = (Set) getValue();
    if (newset == null)
        newset = new HashSet();
    else
        newset = new HashSet(newset);
    newset.removeAll(visible);
    newset.addAll(s);
    super.setValue(newset);
}
}
```

Folgende Code-Zeilen beschreiben den Fall dass nur eine Optionen selektiert werden kann:

```
// Single select mode
else {
    if (ka.length == 0) {
```

In diesem Fall wurde nichts selektiert. In weiterer Folge wird geprüft, ob das betreffende Element sichtbar ist, und bei positivem Ergebnis wird die Selektion gelöscht.

```
// Allow deselection only if the item is visible
Object current = getValue();
Collection visible = getVisibleItemIds();
if (visible != null && visible.contains(current))
    setValue(null);
} else
```

In diesem Fall wurde eine Option selektiert. In weiterer Folge können drei Fälle auftreten. Ist die ID der selektierten Option gleich einer zu definierenden

`nullSelectionItemId` (ID eines Elements das 'Unselektiert' bedeutet – zum Beispiel: DropDown-Liste für Anrede: mit den Elementen 'Bitte wählen sie', 'Frau', 'Herr', etc. wobei 'Bitte wählen sie' keine korrekte Selektion im Kontext von Anrede ist. Wird nichts anders ausgewählt sendet der Browser die ID von 'Bitte wählen sie' als selektierten Eintrag. Ist diese ID nun gleich der `nullSelectionItemId` wird im Model nichts als selektiert vermerkt) wird die Selektion ebenfalls gelöscht. Bei einem neuen Element wird dieses selektiert (und somit zu den möglichen Optionen hinzugefügt) ansonsten wird das angegebene Objekt selektiert.

```

        Object id = itemIdMapper.get(ka[0]);
        if(id!=null && id.equals(getNullSelectionItemId()))
            setValue(null);
        else if (itemIdMapper.isNewIdKey(ka[0]))
            setValue(newitem);
        else
            setValue(id);
    }
}

```

Folgende Code-Zeilen fangen eventuell durch Fehler geworfene Exceptions ab und führen sie der Behandlung durch Millstone zu (auf sie wird in der Ergebnisseite hingewiesen):

```

    } catch (Throwable e) {
        if (e instanceof ErrorMessage)
            setComponentError((ErrorMessage) e);
        else
            setComponentError(new SystemError(e));
    }
}

```

Hiermit endet die Behandlung der Rückgabeparameter des Web-Browsers für diese Komponente.

4.4.3 Funktionsweise einer einfachen Benutzerschnittstellen-Komponenten am Beispiel der Button-Klasse

Benutzerschnittstellen-Komponenten sind alle ähnlich aufgebaut. Dieser Aufbau wird in diesem Unterkapitel exemplarisch an der Button-Klasse demonstriert. (Zwecks der Übersichtlichkeit wurden unwichtigere Kommentare und Methoden entfernt.)

```

package org.millstone.base.ui;

import java.util.Map;
import java.lang.reflect.Method;

import org.millstone.base.data.Property;
import org.millstone.base.terminal.PaintTarget;
import org.millstone.base.terminal.PaintException;
import org.millstone.base.terminal.ErrorMessage;
import org.millstone.base.terminal.SystemError;

```

```
public class Button extends AbstractField {
```

Alle Benutzerschnittstellen-Komponenten müssen direkt oder indirekt von der AbstractComponent-Klasse bzw. der AbstractField-Klasse abgeleitet sein.

```
/* Final members ***** */
```

Definition von Konstanten:

```
/** strings to be caught at adapter (transformer) */
private static final String BUTTON_VAR_NAME = "clicked";
/* Private members ***** */
```

Definition von Variablen: (hier keine)

Definition der Konstruktoren:

```
/** Creates a new push button.      */
public Button(String caption) {
    setCaption(caption);
}

/** Creates a new push button with click listener.      */
public Button(String caption, ClickListener listener) {
    this(caption);
    addListener(listener);
}

/** Creates a new push button with a method listening button clicks.
The method must have either no parameters, or only one parameter of
Button.ClickEvent type.      */
public Button(String caption, Object target, String methodName) {
    this(caption);
    addListener(ClickEvent.class, target, methodName);
}
```

Der UIDL-Tag (User Interface Definition Language) wird zur Benennung der Benutzerschnittstellen-Komponente in der XML-Repräsentation des Ergebnisses verwendet.

```
/** Get component UIDL tag.      */
public String getTag() {
    return "button";
}
```

Wie in Unterkapitel 4.4.1 bereits erläutert generiert die Methode paintContent die XML-Repräsentation des jeweiligen Buttons. Dafür wird nur eine Variable für den Wert des Buttons hinzugefügt (die Kennzeichnung des Buttons selbst und weitere Parameter werden von den Ober-Klassen aufgrund von super.paintContent eingefügt).

```
/** Paint the content of this component.      */
public void paintContent(PaintTarget target) throws PaintException {
    super.paintContent(target);

    boolean state;
    try {
```

```

        state = ((Boolean) getValue()).booleanValue();
    } catch (NullPointerException e) {
        state = false;
    }
    target.addVariable(this, "state", state);
}

```

Wie im vorherigen Unterkapitel (4.4.2) bereits erläutert verarbeitet die Methode `changeVariables` die Rückmeldung des Web-Browsers. Wobei hier im Falle einer Benutzerinteraktion mit dem Button ein Ereignis an den Listener geschickt wird.

```

/** Invoked when the value of a variable has changed. */
public void changeVariables(Object source, Map variables) {
    if (variables.containsKey("state")) {
        try {

            // Get the new and old button states
            Boolean newValue = (Boolean) variables.get("state");
            Boolean oldValue = (Boolean) getValue();

            // Only send click event if button is pushed
            if (newValue.booleanValue())
                fireClick();
            // If button is true for some reason, release it
            if (oldValue != null && oldValue.booleanValue())
                setValue(new Boolean(false));
        } catch (Throwable e) {
            if (e instanceof ErrorMessage)
                setComponentError((ErrorMessage) e);
            else
                setComponentError(new SystemError(e));
        }
    }
}

```

Definiert den Typ einer Benutzerschnittstellen-Komponente. Mögliche Werte sind zum Beispiel `Boolean`, `Integer`, `String` oder `Object`.

```

/** The type of the button as a property. */
public Class getType() {
    return Boolean.class;
}

```

Hilfs-Methoden und Hilfs-Klassen für die Behandlung des Drückens eines Buttons:

```

/* Click event ***** */

private static final Method BUTTON_CLICK_METHOD;
static {
    try {
        BUTTON_CLICK_METHOD =
            ClickListener.class.getDeclaredMethod(
                "buttonClick",
                new Class[] { ClickEvent.class });
    } catch (java.lang.NoSuchMethodException e) {
        // This should never happen
        throw new java.lang.RuntimeException();
    }
}

```

```
}
```

Klasse für das Ereignis bei einem Click:

```
/** Click event. Is thrown, when the button is clicked. */
public class ClickEvent extends Component.Event {

    /** New instance of text change event */
    public ClickEvent(Component source) {
        super(source);
    }

    /** Button where the event occurred */
    public Button getButton() {
        return (Button) getSource();
    }
}
```

Listener-Interface für ein Click-Ereignis:

```
/** Button click listener */
public interface ClickListener {

    /** Button has been pressed. */
    public void buttonClick(ClickEvent event);
}

/** Add button click listener */
public void addListener(ClickListener listener) {
    addListener(ClickEvent.class, listener, BUTTON_CLICK_METHOD);
}

/** Remove button click listener */
public void removeListener(ClickListener listener) {
    removeListener(ClickEvent.class, listener, BUTTON_CLICK_METHOD);
}

/** Emit options change event. */
protected void fireClick() {
    fireEvent(new Button.ClickEvent(this));
}
}
```

Ende der Button-Klasse.

Für Details wie man Millstone installiert und verwendet lesen sie bitte *Fallstudie – Millstone installieren, verwenden, erweitern* und *Appendix A – Benutzerhandbuch für das modifizierte Millstone*. Ein weiteres Beispiel einer Benutzerschnittstellen-Komponenten finden sie in *Appendix B – Codebeispiele*.

5 Millstone – WeLearn-Edition

5.1 Ziele

Bei dem praktischen Teil meiner Diplomarbeit – der Anpassung von Millstone (auf Basis der Version 3.0.3) an die Bedürfnisse des neuen WeLearn-System (Version 3) des Instituts für Informationsverarbeitung und Mikroprozessortechnik (FIM) – gab es drei Schwerpunkte:

- Die Erweiterung und Neuimplementierungen von Benutzerschnittstellen-Komponenten.
- Die Sicherstellung der Funktionalität mit älteren Web-Browsern, Text-Browsern, bzw. allgemein Web-Browsern ohne aktiven JavaScript.
- Die Behebung von Fehlern in Millstone.

Die Funktionalität mit Text-Browsern, die JavaScript meist gar nicht oder nur ziemlich eingeschränkt interpretieren können, ist besonders im Hinblick auf des universitäre Umfeld wichtig. Dort ist es gesetzlich vorgeschrieben Personen mit körperlichen Beeinträchtigungen nicht zu diskriminieren, was in diesem Zusammenhang hauptsächlich auf Personen mit eingeschränkten Sehvermögen, oder gar Blindheit zutrifft. Diese Personen benutzen zum Lesen Screen-Reader, die meist einen Text-Browser wie Lynx verwenden, um HTML-Seiten anzuzeigen bzw. vorzulesen [ScreenReader].

5.2 Architektur

Die Architektur der WeLearn-System-Anpassung ist größtenteils identisch mit dem original Millstone, nur in ein paar Details bestehen Unterschiede. Nähere Information zu den Unterschieden erhalten sie im Unterkapitel *Umsetzung*.

Die prinzipielle Aufteilung erfolgt auch hier in `BaseLib`, Daten-Schnittstellen, `Themes` und `WebAdapter`, wobei die Daten-Schnittstellen unverändert geblieben sind.

5.2.1 BaseLib

Die `BaseLib` enthält in der Originalversion folgende Benutzerschnittstellen-Komponenten:

- `Button`: Eine Schaltfläche, aber auch ein Link innerhalb der Applikation, je nach Style

- `DateField`: Ein Feld zum Setzen eines Datums, entweder nur Felder oder die Tage eines Monats in Kalenderform, je nach Style
- `Embedded`: Eingebettete Objekte: Bilder, Applets, Flash, etc.
- `Form`: Definiert ein Formular zur logischen (optischen) Gruppierung von Elementen
- `FrameWindow`: Definiert ein Fenster mit Frames
- `GridLayout`: Definiert ein NxM Layout zum Positionieren von Elementen
- `Label`: Ein Textfeld, kann neben reinen Zeichenfolgen zum Beispiel auch XML und XHTML enthalten
- `Link`: Definiert einen Link, allerdings nicht innerhalb der Applikation (dafür siehe `Button`: Link-Style) sondern auf eine Resource – zum Beispiel eine Datei zum herunterladen (die im Dateisystem innerhalb der Applikation liegt) oder auch externe Ziele, wobei als extern alles außerhalb von Millstone gilt.
- `OrderedLayout`: Ordnet eingefügte Elemente horizontal oder vertikal hintereinander (`OrderedLayout` ist das Standardlayout innerhalb von Komponenten - zum Beispiel im `Panel`)
- `Panel`: Gruppiert enthaltene Elemente
- `Select`: Bietet Auswahllisten, Drop-Down-Felder, Radiobuttons und Checkboxes
- `Table`: Definiert eine Tabelle (als optisch sichtbare, nicht im allgemeinen Sinn einer HTML-Table). Die Zeilen sind selektierbar, in einer kommerziellen Erweiterung bzw. als Ergebnis dieser Diplomarbeit auch sortierbar.
- `TabSheet`: Definiert Karteireiter
- `TextField`: Definiert ein Texteingabefeld (sowohl ein- als auch mehrzeilig)
- `Tree`: Definiert je nach Style einen Baum oder ein Menü (zum Beispiel für Navigationsbäume)
- `Upload`: Definiert ein Feld zum Transfer einer lokalen Datei vom Benutzer direkt in die Applikation
- `Window`: Definiert sowohl das eigentliche Hauptfenster der Applikation als auch Fenster, die zusätzlich geöffnet werden.

In der WeLearn-System-Anpassung kommt noch folgende Benutzerschnittstellen-Komponente hinzu:

- `ImageMap`: Definiert ein Bild mit verlinkbaren Bereichen (Rechteck, Kreis, frei)

Weiters erfolgte bei der WeLearn-System-Anpassung eine Erweiterung der Funktionalität einiger Benutzerschnittstellen-Komponenten, von denen die wichtigsten hier aufgelistet sind:

- `FrameWindow`: Verbesserte Funktionalität bei Text-Browsern und Browsern ohne aktivem JavaScript – Betrifft nur die Darstellung im Web-Browser, nicht die Verwendung in einer Applikation.
- `Tabelle`: Sortierbarkeit der Tabelle mit Windows-Explorer ähnlichen Look-And-Feel, hinzufügen der Optionen 'Select All' (gesamte Tabelle selektieren), 'Unselect All' (gesamte Tabelle deselektieren) und 'Invert Selection' (Selektierung invertieren).
- `Window`: Verbesserte Funktionalität bei Text-Browsern und Browsern ohne aktivem JavaScript – Betrifft nur die Darstellung im Web-Browser, nicht die Verwendung in einer Applikation.

Eine weitere Änderung an den Schnittstellen einer Benutzerschnittstellen-Klasse wurde in der Basisklasse aller Applikation – der `Application`-Klasse – vorgenommen (es wurden zwei neue Methoden zur `Application`-Klasse hinzugefügt). Dies wurde allerdings nicht infolge dieser Diplomarbeit nötig, sondern zum Zweck der Benutzer Autorisierung, die für das WeLearn-System nötig ist.

5.2.2 Themes

Die `Themes` haben den gleichen Aufbau wie die des originalen Millstone, allerdings wurde das `Default`-Theme für die WeLearn-Edition teilweise erheblich verändert.

Dies erfolgte einerseits, um die neuen Funktionalitäten abzubilden, und um andererseits die JavaScript Funktionalität durch reines HTML zu ersetzen. Nur für einige hilfreiche, aber nicht notwendige Funktionen wird weiterhin JavaScript verwendet, da sie mit HTML nicht sinnvoll nachbildbar sind.

5.2.3 WebAdapter

Natürlich musste auch der `WebAdapter` leicht angepasst werden, um mit den durch

die JavaScript-HTML-Umstellung verbundenen Änderungen an der Art der Variablenübergabe vom Browser zurecht zu kommen. So wird etwa eine Selektion im Original-Millstone durch einen komplex (per JavaScript) zusammengesetzten String übermittelt, was sich mit normalem HTML nicht nachbilden lässt.

Nähere Informationen findet man in der API [MSAPI] bzw. dem Quellcode [MSSource].

5.3 Umsetzung

5.3.1 Erweiterung der Tabelle

Die größte Änderung an einer Klasse wurde bei der `Table`-Komponente durchgeführt, um sie sortierbar zu machen und die Selektierbarkeit der Zeilen zu verbessern.

Die ursprüngliche Auswahlmöglichkeit bestand darin einzelne Zeilen direkt zu selektieren bzw. zu deselektieren, was allerdings bei vielen zu selektierenden Einträgen nicht sehr benutzerfreundlich ist. Um die Handhabung vieler Einträge zu vereinfachen wurde unter der Tabelle ein zusätzlicher Bereich für die Selektion hinzugefügt (im `Default-Theme`), in dem man die Möglichkeit hat, alle Tabelleneinträge zu selektieren (`Select All`), keine zu selektieren (`Select None`) und die momentane Selektion der Tabelle umzukehren (`Select Invert`).

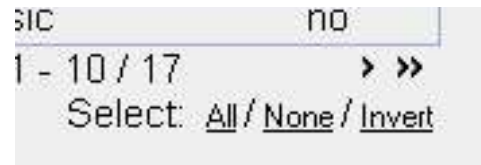


Abbildung 7 - Tabellen Selektionsoptionen

Die Sortiermöglichkeit der Tabelle wurde neu implementiert, wobei optisch Anleihe an Programmen wie dem Windows Explorer genommen wurde, um die Benutzung zu vereinfachen. Für die Sortierbarkeit, mussten einige größere Änderungen am Aufbau der Tabelle vorgenommen werden, da die Original-Tabelle nur reinen Text, aber keine Schaltflächen oder Verweise für die Spaltenbeschriftung unterstützte. Aus Kompatibilitätsgründen wurde

 A screenshot of a table component. The table has two columns: 'Class' and 'Category'. The 'Class' column contains icons (a document icon for 'Window', a folder icon for 'Upload', a tree icon for 'Tree', and a text field icon for 'Text Field'). The 'Category' column contains the corresponding category names: 'Layouts', 'Data Handling', 'Item Container', and 'Basic'.

Class	Category
Window	Layouts
Upload	Data Handling
Tree	Item Container
Text Field	Basic

Abbildung 8 - Tabellen Sortierung

die Definition der Spaltenbeschriftung mittels eines Strings belassen. Es wurde nur ein zusätzliches Attribut eingefügt, das bestimmt, ob die Tabelle sortierbar ist oder nicht. Ein

definierbares Ziel (wie bei der Button-Klasse) für einzelne Spaltenüberschriften wurde nicht implementiert, da es noch wesentlich größere Änderungen an der Tabelle verursacht hätte, und auf Grund der internen Behandlung der Sortier-Ereignisse in der Tabellen-Klasse selbst, auch nicht als sinnvoll erschien. Ist dieses Attribut gesetzt wird das Theme angewiesen, statt der herkömmlichen Spaltenbeschriftung nun Verweise zu verwenden, die die ausgewählte Spalte an die Tabelle zurückmelden, wodurch in weiterer Folge eine Neusortierung der Tabelle ausgelöst wird. Dafür wurde ein weiteres Attribut verwendet, das angibt, nach welcher Spalte und ob aufsteigend oder absteigend sortiert werden soll. Diese Zweiteilung wurde nicht nur zum Zweck der sauberen Implementierung vorgenommen, sondern auch um spezielleres Verhalten zu ermöglichen. Wird zum Beispiel die Tabelle als sortierbar definiert, wird aber nicht angegeben wie sortiert werden soll, erfolgt die erste Darstellung in der momentanen Reihenfolge (ohne vorhergegangene Sortierung beziehungsweise Änderungen an der Methode des Einfügens ist dies die Reihenfolge, in der die Elemente in die Tabelle eingefügt wurden). Wird zwar die Art der Sortierung angegeben, die Tabelle allerdings nicht als sortierbar definiert, wird dem Benutzer eine sortierte Tabelle angezeigt, deren Sortierung er aber nicht beeinflussen kann. Um in einer Spalte sortieren zu können, müssen die Objekte in dieser Spalte das Interface `java.lang.Comparable` implementierten (die Methode `int compareTo(Object obj)`).

Folgendes Code-Beispiel zeigt die Verwendung einer sortierbaren Tabelle:

```
Table tbl = new Table();
tbl.addContainerProperty("first", String.class, "", "Class", null,
    Table.ALIGN_LEFT);
tbl.addContainerProperty("second", String.class, "", "Category", null,
    Table.ALIGN_LEFT);
tbl.addContainerProperty("third", String.class, "", "Immediate",
    null, Table.ALIGN_CENTER);
tbl.setSortable(true);
tbl.setSortBy(1);
tbl.addItem(new Object[] {"Label", "Basic", "no"}, new Integer(1));
tbl.addItem(new Object[] {"Button", "Basic", "yes"}, new Integer(2));
tbl.addItem(new Object[] {"Link", "Basic", "yes"}, new Integer(3));
```

In diesem Beispiel wird eine Tabelle mit drei Spalten definiert ('Class', 'Category' und 'Immediate'). Dann wird die Tabelle als sortierbar definiert und angegeben, dass aufsteigend nach der ersten Spalte sortiert werden soll. Schließlich werden noch drei Elemente eingefügt (Die Tabelle könnte auch erst nach dem Einfügen der Elemente als sortierbar definiert werden).

Auf Wunsch der Projektleitung ist die `immediate`-Funktionalität für die Selektion, bzw. Deselektion einzelner Zeilen in der Table-Komponente erhalten geblieben, um es zum

Beispiel zu ermöglichen eine Schaltfläche auf der Seite nur zu aktivieren, wenn mindestens ein Element ausgewählt ist (z.B. Für Kopieren, Löschen, etc.) – was mittels einer, durch ein JavaScript ausgelösten, Serverinteraktion bei jeder Selektion bzw. Deselektion gesteuert wird.

Es sei hier ausdrücklich darauf hingewiesen, dass man im Falle der Verwendung dieser Funktion, darauf achten muss, dass bei einem Web-Browser, der kein JavaScript versteht bzw. bei dem JavaScript deaktiviert ist (vom Ergebnis her identisch), die Buttons nicht deaktiviert sind, falls nichts selektiert ist, da diese Web-Browser bei einer Selektion logischerweise keine Serverinteraktion initiieren können. Außerdem sollte man bei dieser Funktion bedenken, dass bei jeder Selektion die gesamte Seite neu geladen wird – also mehr Transfervolumen und Serverlast erzeugt – und somit sowohl für Modem-Nutzer als auch bei vielen Anwendern nicht optimal ist.

5.3.2 Implementierungen der ImageMap

Die ImageMap-Klasse wurde als neue Benutzerschnittstellen-Komponente implementiert, da diese Funktionalität für WeLearn gewünscht war, allerdings in Millstone nicht enthalten ist (nur in einer erweiterten kommerziellen Version).

Eine ImageMap ist ein Bild, das von verweissensitiven geometrischen Formen überdeckt ist (Bereiche, die als Verweis fungieren). So sind beispielsweise alle rechteckige bzw. polygone Bereiche in Abbildung 9 anwählbar, und verweisen zu einem individuell definierbaren Ziel. Theoretisch können sich diese auch überschneiden, wobei meist die zuletzt definierte Form für einen Punkt zur Anwendung kommt. Da das Verhalten im Überdeckungsfall nur von der Browserimplementierung abhängt, sollte man aber Überdeckungen möglichst vermeiden.

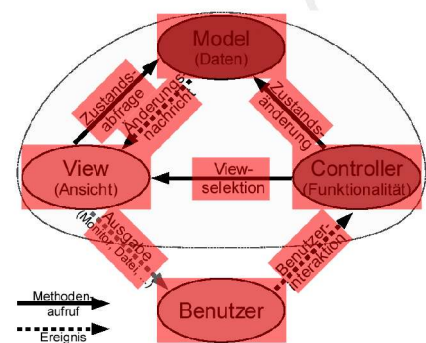


Abbildung 9 - Beispiel der Verlinkung einer ImageMap

Zur Realisierung der ImageMap wurden Klassen für die möglichen Umrisse (Shapes) implementiert, sprich Kreis, Rechteck und Polygon. Es wurde das Hintergrundbild und ein alternativer Text für das Bild und die Shapes definiert, und eine Variable mit der der verwendeten Shape vom Web-Browser zurückgemeldet wird. Als Verweisziel können die gleichen Verweisziele verwendet werden, die auch bei der Button- und Link-Klasse möglich sind (eine Methode der Applikation, eine Datei, ein externer Verweis, usw.).

Für den Java- sowie XSL-Code und weitere Kommentare zur Implementierung sei auf

Appendix B – Codebeispiele verwiesen.

5.3.3 Änderung von JavaScript- in HTML-Funktionalität

Der größten Arbeitsaufwand war durch die Änderung der JavaScript-Funktionalität in HTML-Funktionalität bei allen Benutzerschnittstellen-Komponenten bedingt.

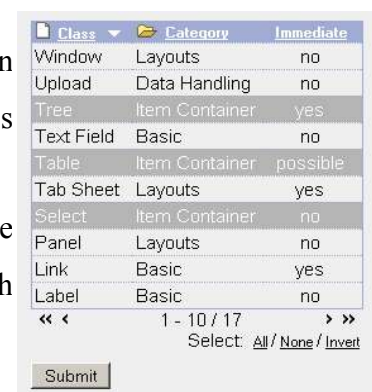
Das Hauptproblem bei der Umsetzung der Änderungen war die Art, wie die Parameter einiger Benutzerschnittstellen-Komponenten vom Web-Browser an Millstone übermittelt werden. Es wurden dabei komplexe Befehle an den WebAdapter per JavaScript zusammengebaut, um beispielsweise die Selektion einer Tabelle zu übermitteln, die mittels normaler HTML-Parameter nicht rekonstruierbar waren. Daher musste die Übermittlung dieser Variablen auf eine Methode geändert werden, die auch mit HTML realisierbar war. Wofür einige Änderungen an der Variablen-Verarbeitung des WebAdapters nötig waren.

Viele Funktionalitäten waren ohne technische Notwendigkeit in JavaScript ausgeführt. Etwa übermittelte der Baum bei der Auswahl eines Knotens in einer per JavaScript generierten Serverinteraktion nur die ID des jeweiligen Knotens. Hier wäre ohne weiteres auch eine Lösung durch HTML möglich gewesen. Dies wurde für die WeLearn-Edition auch derart umgesetzt.

Im Hinblick auf die Benutzbarkeit (Usability) wurde die gesamte HTML-Ausgabe der Benutzerschnittstellen-Komponenten derart geändert, dass damit gebaute Anwendungen sowohl ohne JavaScript als auch mit Text-Browsern bedienbar sind.

Die verbleibende JavaScript-Funktionalitäten sind:

- Die optisch ansprechendere Selektion der Tabellenzeilen durch Farbhinterlegung anstatt mit Checkboxes (falls JavaScript vorhanden ist).
- Die Kontrolle einer nur einfach-selektierbaren Tabelle durch JavaScript (in der JavaScript freien Alternative durch eine Radiogroup).
- Die immediate-Funktionalität einer Tabellenzeile (aus vorher genannten Gründen).
- Die Darstellung der Monatstage in Kalenderform beim Style 'calendar' der DataField-Komponente (ohne JavaScript wird automatisch eine DropDown Alternative angezeigt).



Class	Category	Immediate
Window	Layouts	no
Upload	Data Handling	no
Tree	Item Container	yes
Text Field	Basic	no
Table	Item Container	possible
Tab Sheet	Layouts	yes
Select	Item Container	no
Panel	Layouts	no
Link	Basic	yes
Label	Basic	no

« « 1 - 10 / 17 » »
Select: All / None / Invert
Submit

Abbildung 10 - Selektion in einer Tabelle

- Das Fokussieren von HTML-Elementen
- Das Öffnen von neuen Fenstern (ohne JavaScript wird am Seitenanfang ein Hinweis eingeblendet, dass ein oder mehrere Fenster hätten geöffnet werden sollen, mit entsprechenden Links um die Fenster einzeln zu öffnen).



Abbildung 11 - Neues Fenster ohne JavaScript

Bei folgenden Komponenten wurde JavaScript entfernt und durch HTML ersetzt:

- Button, DateField, Select, Table, TabSheet, TextField, Tree, Upload, Window

Dabei wurde in allen Klassen (ausser Table) die `immediate`-Funktionalität für die Komponenten, die diese Funktionalität in HTML nicht besitzen entfernt (nur die HTML-Elemente Link und Button (sowohl Button als auch Input von Typ 'submit') besitzen diese Funktionalität). Um zu verhindern, dass ein Web-Browser ohne aktiviertem JavaScript zum Beispiel auf einer Seite mit nur einer Texteingabe festsetzt, da diese eigentlich 'immediate' wäre, der Browser die Serverinteraktion aber ohne JavaScript nicht initiieren kann. Ohne 'immediate' benötigt die Applikation auch mit JavaScript einen Sende-Button, wodurch das Problem ohne JavaScript nicht mehr auftritt.

Weiters wurde in der TabSheet-, Tree- und Windows-Klasse die jeweiligen Interaktionen rein mit HTML gelöst.

5.3.4 Fehlerbeseitigung

Die Fehlerbeseitigung war das kleinste der hier erwähnten Aufgabengebiete. In diesem Unterkapitel werden drei der korrigierten Fehler kurz erwähnt:

Ein Fehler der bei der Implementierung der erweiterten Selektier-Funktionalität aufgetreten ist war ein Koordinationskonflikt bei der Verarbeitung der Rückmeldung des Web-Browsers. Da die Variablen in zufälliger Reihenfolge zur Verarbeitung eintreffen passierte es manchmal, dass (ohne `immediate`-Funktionalität bei der Selektion von Zeilen) manchmal zuerst eine Zeile selektiert wurde und dann die Selektion invertiert wurde (so wie es sein sollte, da eine Invertierung der Selektion sofort erfolgt – eine Änderung der Selektion in der Tabelle

muss also entweder vorher geschehen oder im Nachhinein in einem anderen Interaktionsschritt), manchmal war es aber auch umgekehrt, dass zuerst der Befehl zur Invertierung ausgeführt wurde und erst dann der zur Selektion einer Zeile. Dieses Problem wurde behoben indem alle `Button`-Ereignisse im ersten Verarbeitungsschritt zurückgestellt werden und nur die Änderungen der restlichen Komponenten abgearbeitet werden. Erst in einem zweiten Schritt werden auch die `Button`-Ereignisse bearbeitet (normalerweise sollte immer nur maximal ein `Button`-Ereignis auftreten, da ein `Button` immer sofort eine Serverinteraktion auslöst).

Ein weiterer Fehler wurde von einem Projekt-Kollegen in der `Upload`-Komponente gefunden. Genauer gesagt in der Klasse `MultipartRequest`, die für die `Upload`-Komponente den `UploadStream` verwaltet. Wurde eine leere Datei an die `Upload`-Komponente übermittelt, so war das die Daten enthaltende `byte`-Array gleich `'null'`, dies wurde auch überprüft. Allerdings wurde dann eine Methode aufgerufen, die auf diesen Datenstrom Zugriff und somit eine `NullPointerException` verursachte. Zur Lösung für dieses Problem wurde das `byte`-Array auf ein Array der Länge `Null` gesetzt.

Ein dritter Fehler war die `immediate`-Funktionalität der `TextField`-Komponente. Diese `immediate`-Funktionalität funktioniert nur bei einzeiligen Eingabefeldern, da bei mehrzeiligen Eingabefeldern der Web-Browser die Eingabebestätigung (Enter-Taste) nicht als Sende-Kommando versteht, sondern als Zeilenumbruch. Wurde also ein `TextField` als `'immediate'` definiert und ohne Sende-Button mehrzeilig verwendet, konnte der Web-Browser diese Seite nicht mehr abschicken. Dieses Problem wurde gelöst, indem die `immediate`-Funktionalität für das `TextField` entfernt wurde.

5.4 Verwendung

Die Millstone WeLearn-Edition wird genau so verwendet, wie das Original-Millstone, mit Ausnahme der vorgenommenen Erweiterungen und Korrekturen. Bei der Applikations-Entwicklung ist es ratsam, die API der modifizierten Version (zu finden auf der beiliegenden CD) zu verwenden und nicht die Original-API. Als einführendes Tutorial ist auch *Appendix A – Benutzerhandbuch für das modifizierte Millstone* zu empfehlen (das aus Strukturierungsgründen in den Appendix verschoben wurde). Wer Hilfe bei der Einrichtung von Millstone benötigt, findet diese in *Fallstudie – Millstone installieren, verwenden, erweitern*.

6 Fallstudie – Millstone installieren, verwenden, erweitern

6.1 Millstone installieren (mit aller benötigter Software)

Die Eclipse-Entwicklungsumgebung bietet ein hervorragendes Umfeld, um Applikationen mit Java und Millstone zu entwickeln. Für Programmierer, die noch nie mit Eclipse und Millstone gearbeitet haben, bietet dieses Unterkapitel eine kurze Einführung in die Installation aller notwendigen Programme.

Vorbedingungen

- Diese Anleitung wurde für Windows XP geschrieben, sollte aber für die meisten Windows-Plattformen anwendbar sein.
- Alle verwendeten Programme finden Sie auch auf der CD, die dieser Diplomarbeit beiliegt unter "<CD-LW>:\Programme\".

Java2 SDK (Service Development Kit) installieren:

- Das Java2 SDK 1.4.x von <http://java.sun.com/j2se/downloads.html> herunterladen, und gemäß den Installationsanleitungen installieren.
- Die aktuelle Eclipse-Version von <http://www.eclipse.org/downloads/> herunterladen.
- Das Eclipse-Tomcat-Plugin von <http://www.sysdeo.com/eclipse/tomcatPlugin.html> herunterladen.

Apache Tomcat installieren:

- Die aktuelle Apache Tomcat Version von <http://jakarta.apache.org/tomcat/> herunterladen, und gemäß den Installationsanleitungen installieren.

Eclipse installieren:

- Stellen Sie sicher, dass Sie das Java2 SDK installiert haben.
- Entzippen Sie die Eclipse-Zipdatei in das gewünschte Verzeichnis (z.B.: C:\Programme\Eclipse)
- Führen Sie die Datei `eclipse.exe` aus.

Das Tomcat-Plugin installieren:

- (Das Plugin enthält nicht den Apache Tomcat Server.)
- Die Datei tomcatPluginV2.zip von <http://www.sysdeo.com/eclipse/tomcatPlugin.html> herunterladen.
- In das Verzeichnis `<eclipse_home>/plugins` (z.B.: `C:\Programme\Eclipse\plugins`) entzippen.
- Um das Plugin zu aktivieren müssen Sie das Menü 'Window -> Customize Perspective...' wählen, 'Actions Sets' expandieren und 'Tomcat' auswählen.
- Um das Basisverzeichnis von Tomcat zu setzen müssen Sie das Menü 'Window -> Preferences' wählen und unter 'Tomcat' das Basisverzeichnis 'Tomcat Home' auf das Verzeichnis setzen, in das Sie Tomcat installiert haben (z.B.: `C:\Programme\Tomcat`).
- Das Plugin startet Tomcat mittels der Standard-JRE (Java Runtime Environment) von Eclipse.
- Um ein Java2 SDK als Standard-JRE für Eclipse zu definieren, wählen Sie das Menü 'Window -> Preferences' und dann 'Java -> Installed JREs'.
- Dieses JRE muss ein Java2 SDK sein (das ist eine Voraussetzung von Tomcat).
- Sie sollten Java2 SDK Version 1.4.x nutzen, da es das 'Hot Code Replacement' unterstützt (Übernahme von neuen Code ohne Neustart von Tomcat).
- Das Plugin setzt den Tomcat-Classpath (Klassenpfad) und -Bootclasspath. Unter 'Window -> Preferences' und dann 'Tomcat -> JVM Settings' können Sie bei Bedarf spezielle Einstellungen vornehmen.

Das anonyme CVS von Millstone konfigurieren (CVS: Concurrent Versions System – Versions Management System):

- Wählen Sie das Menü 'Window -> Show View -> Other...' und dann 'CVS -> CVS Repositories'.

- Öffnen Sie das Kontext-Menü in der CVS Explorer Ansicht und wählen Sie 'New -> Repository Location'.
- Geben Sie dann folgende Informationen ein:
 - Host: `cvs.millstone.sourceforge.net`
 - Repository path: `/cvsroot/millstone`
 - User: `anonymous`
 - Connection Type: `pserver`

und drücken Sie 'Finish'.

Den Millstone Base Quellcode über anonymes CVS installieren:

- Das anonyme CVS von Millstone, wie oben beschrieben, konfigurieren.
- Das 'base/src' Module als Java Projekt auschecken. Öffnen Sie den `src`-Ordner im CVS Repository Explorer und öffnen Sie das Kontext-Menü. Wählen Sie 'Check Out as', und dann 'Java -> Java Project'. Wählen Sie einen Projektnamen wie "Millstone Base Sources" und drücken Sie 'Finish'.
- TIPP: Um die CVS Quellen in einem anderen Projekt statt der jar-Datei der Millstone Base Library zu verwenden, öffnen Sie ihre Projekteigenschaften um dieses Projekt in den Build-Path zu integrieren. Die Projekteigenschaften erreichen Sie mittels des Kontextmenüs: Properties -> JavaBuild Path -> Projects und aktivieren Sie das Kästchen bei dem 'Millstone WA Sources' Projekt.

Den Millstone Web-Adapter Quellcode über anonymes CVS installieren:

- Das anonyme CVS von Millstone, wie oben beschrieben, konfigurieren.
- Das "webadapter/src" Module als Java Projekt auschecken. Öffnen Sie den 'src'-Ordner im CVS Repository Explorer und öffnen Sie das Kontext-Menü. Wählen Sie 'Check Out as', und dann 'Java -> Java Project'. Wählen Sie einen Projektnamen wie "Millstone WebAdapter Sources" und drücken Sie 'Finish'.

- Hinzufügen von 'Millstone Base' zum Build-Path. Öffnen Sie die Eigenschaften Ihres Projektes (Kontext-Menü -> Properties), wählen dann 'JavaBuild Path -> Projects' und selektieren das 'Millstone Base Sources'-Projekt.
- Hinzufügen der Tomcat-Bibliotheken zum Build-Path. Öffnen Sie die Package Explorer Ansicht in der Java Perspektive und wählen Sie die Projekteigenschaften im Kontextmenü. Wählen Sie: 'Java Build Path -> Libraries' und dann drücken Sie 'Add External JARs'. Wechseln Sie in das Tomcat-Installationsverzeichnis und wählen die Datei 'common/lib/servlet.jar'. (Im Tomcat-Installationsverzeichnis unter 'common\lib\servlet.jar')
- TIPP: Um die CVS Quellen in einem anderen Projekt statt der jar-Dateien der Millstone Bibliotheken zu verwenden, öffnen Sie ihre Projekteigenschaften um dieses Projekt in den Build-Path zu integrieren. Die Projekteigenschaften erreichen Sie mittels des Kontextmenüs: 'Properties -> JavaBuild Path -> Projects' und aktivieren Sie das Kästchen bei dem Millstone WA Sources Projekt.

Die Millstone Web-Themes über anonymes CVS installieren:

- Das anonyme CVS von Millstone, wie oben beschrieben, konfigurieren.
- Das 'base/src' Module als Java Projekt auschecken. Öffnen Sie den src-Ordner im CVS Repository Explorer und öffnen Sie das Kontext-Menü. Wählen Sie 'Check Out as', und dann 'Java -> Java Project'. Wählen Sie einen Projektnamen wie "Millstone Themes" und drücken Sie 'Finish'.
- Stellen Sie sicher, dass alle ihre Applikationen einen Parameter namens 'themsource' haben, der entweder in der Datei 'server.xml' oder 'web.xml' definiert wird. Für die Millstone Beispiele würde die Datei 'server.xml' wie folgt aussehen:

```
<Context path="/examples" [ ... ] >
    [ ... ]
    <!-- Default Theme Source -->
    <Parameter
        name="themsource"
        value="C:\Program
Files\eclipse\workspace\Millstone Themes"
        override="false"/>
```

```
[ ... ]  
</Context>
```

Millstone Beispiele per anonymen CVS installieren

- Als erstes müssen Sie den anonymen CVS-Zugang konfigurieren (Siehe obige Anleitung)
- Stellen Sie sicher, dass Sie das Tomcat-Plugin installiert haben.
- Auschecken des Beispiel-Moduls ('examples') als ein Tomcatprojekt: Öffnen Sie den `src`-Ordner im CVS Repository Explorer, wählen Sie 'Check Out as' im Kontextmenü und dann 'Java -> Tomcat Project'. Geben Sie als Projektnamen "Millstone Examples" an und drücken 'Finish'.
- Konfigurieren Sie den Build-Path, so dass er die Millstone Base Quellen enthält: Als Erstes stellen Sie sicher, dass Sie die Millstone Quellen vom anonymen CVS installiert haben (Siehe obige Anleitung). Öffnen Sie die Projekteigenschaften im Kontextmenü, wählen 'Java Build Path -> Projects' und aktivieren Sie das Projekt 'Millstone Base Sources'.
- Konfigurieren Sie den Build-Path um externe Bibliotheken einzubinden: Das StonePlayer-Beispiel benötigt eine externe MP3-Bibliothek, welche mit den Beispielen mitgeliefert wird. Der Build-Path muss diese Bibliothek enthalten. Öffnen Sie die Projekteigenschaften im Kontextmenü, wählen 'Java Build Path -> Libraries'. Fügen Sie mittels der 'Add JARs'-Schaltfläche die Bibliothek 'Millstone Examples -> WEB-INF -> lib -> j1011.jar' hinzu und drücken Sie 'OK'.

Das Beispielprojekt ausführen:

- Installieren Sie die Millstone Beispiele und die Millstone Quellen vom anonymen CVS, wie oben beschrieben.
- Konfigurieren Sie die Konfigurationsdatei 'server.xml' der Apache Tomcat. Fügen Sie den Beispiel-Kontext ihrer Server-Konfiguration hinzu. Die Konfigurationsdatei ist im Apache Tomcat Programmmenü verlinkt. Sie können zu Beginn auch folgende minimale `server.xml` Datei verwenden, anstatt der Standard Tomcat `server.xml`:

```
<!-- A Minimal conf/server.xml for Millstone Examples
-->
<Server port="8005" shutdown="SHUTDOWN" debug="0">
  <Service name="Tomcat-Standalone">
    <Connector
      className="org.apache.catalina.connector.http.HttpConn
ector" port="8080" />
    <Engine name="Standalone" defaultHost="localhost"
      debug="0">
      <Realm
        className="org.apache.catalina.realm.MemoryRealm" />
      <Host name="localhost" debug="0"
        appBase="webapps" unpackWARs="true">

        <Context path="/examples" docBase="C:\Program
Files\Eclipse\workspace\MillStone Examples">

          <!-- Default Theme Source -->
          <Parameter name="themesource"
            value="C:\Program Files\eclipse\workspace\Millstone
Themes" override="false"/>

          <!-- MP3 directory for StonePlayer -->
          <Parameter name="mp3path" value="C:\mp3"/>
        </Context>
      </Host>
    </Engine>
  </Service>
</Server>
```

HINWEIS: Die jeweiligen Pfade müssen natürlich noch an ihre Installation angepasst werden.

- Konfigurieren Sie den Tomcat Klassenpfad: Wählen Sie 'Window -> Preferences -> Tomcat' und stellen Sie sicher, dass die Projekte 'Millstone Sources' und 'Millstone Examples' aktiviert sind.
- Starten Sie Tomcat mittels der 'Start Tomcat'-Schaltfläche in der Werkzeugleiste.

Eclipse Konfigurationstipps:

- Setzen Sie die gewünschte Schrift-Kodierung für das Speichern: Unter 'Window -> Preferences -> Workbench -> Editors' (in der Regel ISO-8859-1 ie Latin1).
- Rechnerspezifische Applikationsparameter sollten in der Datei `server.xml` deklariert werden.
- Sie können Dateien von einem automatischen Hinzufügen ins CVS ausschließen, indem

Sie unter 'Window -> Preferences -> Team -> CVS -> Ignored Resources' die jeweiligen Ressourcen angeben.

- Sie können die Apache Tomcat als ein Projekt in Eclipse importieren. Dadurch können Sie einfach die Tomcat Konfigurationsdateien aus Eclipse aus anpassen.
- Um die automatische Anzeige der Java-Dokumentation (pop-up javadoc) zu aktivieren, stellen Sie sicher, dass unter 'Window -> Preferences -> Java -> Installed JRES' der JRE-Parameter 'Javadoc URL' auf die korrekte Java-Dokumentation verweist.

(Freie Übersetzung von <http://www.millstone.org/eclipse-install.html>)

6.2 Millstone verwenden (eine eigene Applikation schreiben)

Erste Schritte mit Millstone

Dies ist eine kurze Einführung wie Sie mit der Eclipse IDE eine einfache Millstone-Applikation entwickeln:

- Die Entwicklungsumgebung installieren
- Lesen Sie dafür das vorherige Unterkapitel "Millstone installieren".
- Ein neues Web-Applikation-Projekt anlegen:
 - Um eine neue Web-Applikation zu erstellen, müssen Sie ein neues Projekt in Eclipse anlegen:
 1. Öffnen Sie dafür 'File -> New -> Project' und wählen Sie 'Java -> Tomcat project'.
 2. Nenne Sie das Projekt "HelloWorld", aktivieren Sie 'can update server.xml' und drücken Sie 'Finish'.
 3. Wählen Sie 'Tomcat' in den Projekteigenschaften (erreichbar über das Kontextmenü) und setzen Sie den Kontextnamen auf "/tutorial".

4. Kopieren Sie die Bibliotheken 'millstone[-WeLearnEdition]-base-x.y.jar', 'millstone[-WeLearnEdition]-web-themes-x.y.jar' und 'millstone[-WeLearnEdition]-webadapter-x.y.jar' vom Archiv 'millstone[-WeLearnEdition]-x.y.zip' nach 'Eclipse/workspace/HelloWorld/WEB-INF/lib' (wobei x.y durch die aktuelle Versionsnummer zu ersetzen ist, und -WeLearnEdition weggelassen wird, wenn man mit den originalen Millstone-Bibliotheken arbeiten will).
5. Öffnen Sie die Projekteigenschaften im Kontextmenü und fügen Sie unter 'Java Build Path -> Libraries' die drei obigen externen Bibliotheken als externe JARs ein.

Die Applikation erstellen:

Nun sind Sie bereit, um ihre erste Millstone-Applikation zu erstellen. Sie werden ein klassisches 'Hallo Welt'-Beispiel erstellen, um das Millstone-Framework zu demonstrieren.

1. Wählen Sie 'New -> Class' im Kontextmenü des Projektes, geben Sie der Klasse den Namen "HelloWorld" und geben Sie als Superklasse "org.millstone.base.service.Application" an.
2. Fügen Sie folgendes Paket zur Klasse hinzu:

```
import org.millstone.base.ui.*;
```

3. Und dann weiters folgende drei Zeilen in die Methode `init()`:

```
Window main = new Window("Millstone Beispiel");  
setMainWindow(main);  
main.addComponent(new Label("Hallo Welt!"));
```

4. Das ganze Programm sieht dann aus wie folgt:

```
import org.millstone.base.service.Application;  
import org.millstone.base.ui.*;  
  
public class HelloWorld extends Application {  
  
    public void init() {  
        Window main = new Window("Millstone Beispiel");  
        setMainWindow(main);  
        main.addComponent(new Label("Hallo Welt!"));  
    }  
}
```

5. Speichern Sie die Klasse.

6. Öffnen Sie 'Window -> Preferences -> Tomcat' und stellen Sie sicher, dass das Beispielprojekt zum Tomcat-Klassenpfad hinzugefügt wird.

Die Applikation deployen (in Tomcat einfügen):

Der Servlet-Container benötigt etwas Konfiguration, bevor Sie ihre Applikation verwenden können. Diese Konfiguration wird in der Datei 'web.xml' vorgenommen, welche im 'WEB-INF'-Verzeichnis der Web-Applikation gespeichert wird.

Im folgenden finden Sie die Konfiguration für die obige Beispiel-Applikation. Öffnen Sie das Kontextmenü des 'WEB-INF'-Verzeichnisses ihres Projekts und Wählen Sie 'New -> File'. Nennen Sie die Datei 'web.xml' und kopieren Sie folgenden XML-Code in die Datei:

```
<?xml version="1.0" encoding="UTF-8"?>

<!DOCTYPE web-app PUBLIC "-//Sun Microsystems, Inc.//DTD
Web Application 2.2//EN"
"http://java.sun.com/j2ee/dtds/web-app_2_2.dtd">
<web-app>

    <servlet>
        <servlet-name>hello</servlet-name>
        <servlet-
class>org.millstone.webadapter.WebAdapterServlet</servlet-
class>
        <init-param>
            <param-name>application</param-name>
            <param-value>HelloWorld</param-value>
        </init-param>
    </servlet>

    <servlet-mapping>
        <servlet-name>hello</servlet-name>
        <url-pattern>/*</url-pattern>
    </servlet-mapping>

</web-app>
```

Ausführen des Beispiels:

Um die Beispiel-Applikation zu starten, muss der Servlet-Container laufen. Dafür haben Sie das Tomcat-Plugin installiert. Nachdem der der Servler-Container (Tomcat) läuft, können Sie ihre Applikation über die spezifizierte URL erreichen.

1. Starten Sie Tomcat mittels der 'Start Tomcat'-Schaltfläche in der Eclipse

Werkzeugleiste.

2. Öffnen Sie folgende URL in ihrem Web-Browser: <http://localhost:8080/tutorial>

(Freie Übersetzung von <http://www.millstone.org/getting-started.html>)

6.3 Millstone erweitern (ein bestehendes Theme erweitern)

Als Alternative zu einem völlig neuen Theme besteht auch die Möglichkeit ein vorhandenes Theme zu erweitern.

Dafür muss man unter dem 'WEB-INF/lib/' Verzeichnis des Projekts ein Verzeichnis 'themes' erstellen, in das alle Themes (neue bzw. erweiterbare) in eigene Unterverzeichnisse kommen. In einem solchen Theme-Verzeichnis (zum Beispiel 'WEB-INF/lib/themes/extended' für ein erweitertes Theme) muss sich eine XML-Datei mit dem Namen 'description.xml' befinden.

Der Inhalt dieser Datei wird mittels folgendem Muster erstellt:

----- Beginn description.xml -----

```
<?xml version="1.0" encoding="UTF-8"?>
<theme name="NameDesTheme">
```

Dieser Name muss der des Theme-Verzeichnisses sein (zum Beispiel 'extended').

```
<extends theme="default"/>
```

Bezeichnet den Namen des Theme, das erweitert werden soll.

```
<description>My Extended Theme</description>
```

Beschreibung für das Theme (beliebiger Text).

```
<author name="Your name" email="you@the.web" />
```

Autor des Theme (beliebiger Text).

Der folgende Abschnitt (von '<fileset>' bis '</fileset>') wird nur benötigt, falls im Theme XSL-Transformationsdateien vorkommen (sollte nur das CascadingStyleSheet angepasst werden kann dieser Abschnitt weggelassen werden). Werden XSL-

Transformationsdateien verwendet muss für jede Datei eine '<file name="Unterverzeichnis/Dateiname.xml" />'-Zeile eingetragen werden, wobei 'Dateiname.xml' natürlich mit dem Dateinamen der jeweiligen Datei zu ersetzen ist. 'Unterverzeichnis' ist mit der jeweiligen Unterverzeichnis-Struktur zu ersetzen (ohne führenden Schrägstrich). Man startet dabei im Verzeichnis des Theme.

Ist also das Theme unter 'WEB-INF/lib/themes/extended' zu finden und die XSL-Transformationsdateie 'MeineUIKomponente.xml' (für die Benutzerschnittstellen-Komponente 'MeineUIKomponente.java') unter 'WEB-INF/lib/themes/extended/xsl/basic/', dann lautet die Unterverzeichnis-Struktur 'xsl/basic/' und somit ist die Zeile '<file name="xsl/basic/MeineUIKomponente.xml" />' einzufügen.

```
<fileset>
  <file name="myOwnXsl_1.xml" />
  <file name="subdir/myOwnXsl_2.xml" />
</fileset>

</theme>
```

----- Ende description.xml -----

Will man nun beispielsweise das optische Erscheinungsbild der Benutzerschnittstellen-Komponenten des Standard Theme anpassen, so muss man nur das Theme 'default' – wie oben beschrieben – erweitern und man muss dann nur ein Unterverzeichnis 'css' und darin die Datei 'default.css' erstellen, in der man die gewünschten Änderungen vornimmt (am Einfachsten wird es wahrscheinlich sein, die original Datei zu Kopieren und dann zu Ändern, um keine Styles zu vergessen).

Um nun das erweiterte Theme zu benutzen, muss man in dem Applikationscode folgende Zeile einfügen:

```
setTheme ("NameDesTheme") ;
```

Bemerkung: 'getTheme()' gibt 'null' zurück, wenn das Standard Theme ('default') verwendet wird. Will man zum Standard Theme zurück wechseln, muss man 'setTheme ("default") ' verwenden und nicht 'setTheme (null) '.

7 Fazit

Das Model-View-Controller-Paradigma ist heute aus größeren Projekten kaum mehr wegzudenken, da die Applikationen eine Komplexität erreichen, die nur mehr im Team in vernünftigen Zeiträumen zu erledigen ist. Um hierbei das Konfliktpotential der einzelnen Teile möglichst gering zu halten, muss man nicht nur die Programmlogik in mehrere nur mittels definierter Schnittstellen kommunizierende Teile auftrennen, sondern auch die eigentliche Logik von der Eingabe und der Ausgabe, sowie diese beide untereinander zu trennen. Wie wir gesehen haben, hat diese Trennung noch weitere Vorteile: etwa die einfachere Wartbarkeit des übersichtlicheren Codes, die nicht (oder nur kaum) vorhandenen Einflüsse auf die anderen beiden Teile bei einer Änderung im dritten Teil oder die Austauschbarkeit der Ein- und Ausgabe um mit der gleichen Programmlogik unterschiedliche Darstellungsarten zu erreichen.

Wir haben auch gesehen, dass es mehrere Arten der Umsetzung der Model-View-Controller-Paradigma im Java-Umfeld gibt. Die Möglichkeiten gehen hier von Servlets über Java-Server-Pages und eXtensible-Server-Pages bis hin zu Frameworks wie Struts und Cocoon und schließlich hierarchischen Model-View-Controller-Frameworks wie Millstone. Während in dieser Reihenfolge die Komplexität der Implementierung größerer Applikationen immer mehr abnimmt wird ein direktes Eingreifen in das optische Ergebnis immer komplizierter. So ist es bei den hierarchischen Model-View-Controller-Frameworks nicht mehr so einfach möglich einem einzelnen Element in HTML mittels eines style-Attributs zum Beispiel eine leicht abweichende Position zuzuordnen, statt dessen muss man eine eigene CSS-Klasse definieren, die bis auf die paar abweichenden Werte der ursprünglichen Klasse entspricht und diese Klasse dann (falls überhaupt möglich) dem Objekt zuweisen. Andererseits benötigt man bei den hierarchischen Frameworks nur Kenntnisse in Java und nicht in HTML und CSS bzw. XML und XSLT. Es ist also wie so oft eine Frage der jeweiligen Anforderungen und Vorlieben zu welcher Umsetzung des Model-View-Controller-Paradigmas man greift.

In Unternehmen kommen noch weitere Punkte hinzu, die sich auf eine Entscheidung pro oder contra einer bestimmten Umsetzung des Model-View-Controller-Paradigmas auswirken. Da ist einerseits der Support, der bei kommerziellen Frameworks meist teurer aber dafür umfassender ist als bei Open-Source-Frameworks (wobei ich hier weniger an Handbücher als viel mehr an Schulungen und Unterstützung bei größeren Implementierungs-Problemen denke). Andererseits sind kommerzielle Frameworks meist in ihrer Anschaffung teurer und es gibt

normalerweise auch keine Zusicherung für den Preis von Folgeversionen. So kann man etwa bei den Open-Source-Frameworks Struts und Cocoon von der Apache Foundation davon ausgehen, dass auch zukünftige Versionen frei (und gratis) erhältlich sind. Bei kommerziellen Frameworks wie Millstone könnte es aber durchaus geschehen, dass die jeweilige Firma ihr Geschäftsmodell ändert und den Preis für das Framework anhebt, oder es überhaupt nur mehr in Kombination mit Entwicklung oder Support vertreibt. Für die WeLearn-Edition von Millstone geht hiervon aber keine Gefahr aus, da die zugrunde liegende Millstone-Version kostenlos als Open-Source zur Verfügung steht, und dies nur bei einer zukünftigen Folgeversion geändert werden könnte.

In dieser Arbeit wurde zuerst ein grundlegender Überblick über e-Learning und WeLearn, die Lernplattform in deren Rahmen diese Arbeit geschrieben wurde, gegeben. Danach wurde detailliert auf das Model-View-Controller-Paradigma, und auf das hierarchische Model-View-Controller-Framework Millstone, das für den praktischen Teil diese Arbeit die Ausgangsbasis bildete, eingegangen, um schließlich bei der Einführung in die Millstone-WeLearn-Edition zu enden.

Ziel dieser Arbeit war es eine theoretische Einführung für das Model-View-Controller-Paradigma zu geben und schließlich das hierarchische Model-View-Controller-Framework Millstone als Millstone-WeLearn-Edition an die Bedürfnisse von WeLearn anzupassen und um neue Komponenten zu erweitern. Die WeLearn-Edition ist nicht mehr notwendigerweise auf JavaScript angewiesen, bietet jedoch mit JavaScript etwas mehr Komfort. Natürlich ist dieses Framework nicht nur auf WeLearn im Speziellen oder e-Learning im Allgemeinen beschränkt, sondern kann für jegliche Art von Web-Applikation verwendet werden. Zur leichteren Verwendbarkeit findet sich in dieser Arbeit auch ein Benutzerhandbuch, das die wichtigsten Klassen und Methoden etwas ausführlicher als in der Millstone-API beschreibt, und zusätzlich mit einigen Beispielen sowohl den Einstieg in die Programmierung mit Millstone, als auch die Erweiterung von Millstone um eigene Komponenten erleichtert.

8 Appendices

8.1 Appendix A – *Benutzerhandbuch für das modifizierte Millstone*

Für die Millstone WeLearn-Edition.

Allgemeine Attribute aller Benutzerschnittstellenklassen (von AbstractComponent):

add-/removeListener(Class *eventType*, Object *object*, Method *method*): Fügt einen Listener der Klasse *eventType* zur Komponente hinzu, der auf die Methode *method* der Klasse *objekt* verweist, bzw. entfernt diesen Listener.

add-/removeListener(Class *eventType*, Object *object*, String *methodName*): Fügt einen Listener der Klasse *eventType* zur Komponente hinzu, der auf die Methode mit dem Namen *methodName* der Klasse *objekt* verweist, bzw. entfernt diesen Listener.

add-/removeListener(Component.Listener *listener*): Fügt den neuen Listener *listener* zur Komponente hinzu, bzw. entfernt diesen Listener.

add-/removeListener(Paintable.RepaintRequestListener *listener*): Fügt den neuen Listener *listener* zur Komponente hinzu, bzw. entfernt diesen Listener.

attach()/detach(): Benachrichtigt die Komponente, dass sie zu einer anderen Komponente hinzugefügt bzw. von einer anderen Komponente entfernt wurde.

changeVariables(Object *source*, Map *variables*): Diese Methode wird von Millstone aufgerufen, wenn sich der Zustand einer Variablen geändert hat. Die Quelle der Änderung ist dabei *source* und *variables* enthalten die Namen und die Werte der Variablen.

childRequestedRepaint(Collection *alreadyNotified*): Eine untergeordnete Komponente ersucht die Komponente um eine neu generierte Darstellung. Die Collection *alreadyNotified* enthält dabei alle bereits informierten Komponenten.

dependsOn/removeDirectDependency(VariableOwner *depended*): Definiert die Komponente als abhängig von *depended*, bzw. entfernt diese Abhängigkeit.

fireEvent(Component.Event *event*): Löst das Ereignis *event* aus.

getApplication(): Gibt die übergeordnete Applikation der Komponente zurück.

get-/setCaption([String caption]): Gibt Bezeichnung einer Komponente zurück, bzw. setzt diese auf *caption*. Bei Schaltflächen und Verweisen ist es beispielsweise der angezeigte Text, bei Textfeldern ist es die Beschriftung des Feldes (nicht dessen Inhalt) und bei komplexeren Komponenten wie zum Beispiel der Tabelle ist es die übergeordnete Beschriftung der Kindkomponenten.

get-/setComponentError([ErrorMessage componentError]): Gibt die Fehlermeldung einer Komponente zurück, bzw. setzt diese auf *componentError*. Diese Fehlermeldung einer Komponente, die in der JavaScript-Version per Pop-up mittels eines Verweises (standardmäßig ein weißes Ausrufezeichen im roten Kreis), ohne Javascript direkt in die Seite eingefügt wird. Diese Fehlermeldung dient der Übermittlung eines Fehlerberichts der Anwendung an den Benutzer (zum Beispiel eine Fehlermeldung, dass in dem betreffenden Feld ein Zahl erwartet wird, allerdings etwas anderes eingegeben wurde).

get-/setDescription(String description): Gibt die zusätzliche Beschreibung einer Komponente zurück, bzw. setzt diese auf *description*. Die Beschreibung wird in der JavaScript-Version per Pop-up mittels eines Verweises (standardmäßig ein weißes 'i' im blauen Kreis), ohne Javascript direkt in die Seite eingefügt. Zum Beispiel ein Hinweis, dass in dem betreffenden Feld ein Zahl eingegeben werden muss.

get-/setIcon(Resource icon): Gibt das Icon der Komponente zurück, bzw. setzt diese auf *icon*. Ein Icon ist ein kleines Bild, das vor der Komponente angezeigt wird.

get-/setLocale(Locale locale): Gibt die Klasse `java.util.Locale` der Komponente zurück, bzw. setzt diese auf *locale*. Diese Klasse wird verwendet um die Sprache der Komponente (deren Inhalt) zu definieren.

getParent(): Gibt die Elternkomponente zurück.

get-/setStyle(String style): Gibt den momentanen Style der Komponente zurück, bzw. setzt diese auf *style*. Ein Style definiert das optische Erscheinungsbild der Komponente. Einige Komponenten haben nur einen Standard-Style, bei den anderen sind die zusätzlichen Styles bei den speziellen Attributen aufgeführt.

getTag(): Gibt den UIDL-Tag der Komponente zurück, die verwendet wird um eine Komponente innerhalb der Millstone-Frameworks eindeutig zu identifizieren. Dieser Tag wird in der jeweiligen Klasse direkt angegeben und entspricht für gewöhnlich dem Klassennamen.

is-/setEnabled(boolean enabled): Gibt zurück ob die Komponente Änderungen des Wertes erlaubt, bzw. setzt diesen Zustand auf *enabled*. Bei Elementen die nicht 'enabled' sind werden keine Änderungen des Zustandes erlaubt. Standardmäßig sind alle Elementen 'enabled'.

is-/setImmediate(boolean immediate): Gibt zurück ob die Komponente auf Änderungen mit einem sofortigen Aufruf der Zielklasse reagiert, bzw. setzt diesen Zustand auf *immediate*. Gibt an, dass ein Element bei einer Zustandsänderung automatisch ein Ereignis auslöst. Dies (ist technologiebedingt nicht bei allen Elementen möglich - beispielsweise kann reiner Text im Gegensatz zu einem Verweis in einem Web-Browser keine Serverinteraktion auslösen (die Label-Klasse kann also bei einer HTML-Darstellung einem `setImmediate()` nicht entsprechen).

is-/setReadOnly(boolean readOnly): Gibt zurück ob die Komponente schreibgeschützt ist, bzw. setzt diesen Zustand auf *readOnly*.

is-/setVisible(boolean visible): Gibt zurück ob die Komponente sichtbar ist, bzw. setzt diesen Zustand auf *visible*. Ist eine Komponente nicht sichtbar, so wird sie in der Ausgabe nicht angezeigt.

paintContent(PaintTarget target): Initiiert eine erneute Darstellung aller benötigten, komponentenspezifischen Attribute, und ruft dann `paint()` auf.

paint(PaintTarget target): Initiiert eine erneute Darstellung aller komponentenübergreifenden Attribute

removeListener(Class eventType, Object target): Entfernt alle Listener des Typs *eventType*, die auf eine Methode der Klasse *target* verweisen.

requestRepaint(): Erbittet eine schnellst mögliche Neudarstellung der Komponente.

requestRepaintRequests(): Erbittet eine Versendung von Ereignissen an alle Komponenten mit sichtbaren Änderungen.

Allgemeine Attribute aller Benutzerschnittstellenklassen, die von `AbstractField` abstammen: (Button, DateField, Panel, Select, Table, TextField, Tree, Window, FrameWindow)

`add-/removeListener(Property.ReadOnlyChangeListener listener)`: Fügt den Listener *listener* hinzu, der auf Änderungen des Nur-Lesen-Status von Feldern reagiert, bzw. entfernt ihn.

`add-/removeListener(Property.ValueChangeListener listener)`: Fügt den Listener *listener* hinzu, der auf Änderungen des Wertes von Feldern reagiert, bzw. entfernt ihn.

`add-/removeValidator(Validator validator)`: Fügt einen den Validator *validator* für Feldwerte hinzu, bzw. entfernt diesen.

`commit()`: Überträgt alle neuen Änderungen an die Datenquelle

`constructField(Class propertyType)`: Erzeugt ein neues Element vom Typ *propertyType*.

`discard()`: Verwirft alle Änderungen seit dem letzten *commit()*.

`fireReadOnlyStatusChange()`: Erzeugt ein Ereignis, dass sich der Nur-Lesen-Status eines Feldes geändert hat.

`fireValueChange()`: Erzeugt ein Ereignis, dass sich der Wert eines Feldes geändert hat.

`focus()`: Versucht dem Feld im Browser den Fokus zu geben (funktioniert nur mit eingeschaltetem JavaScript und nicht bei allen Elementen).

`get-/setPropertyDataSource([Property newDataSource])`: Gibt die Datenquelle für das Feld zurück, bzw. setzt diese auf *newDataSource*.

`getType`: Gibt die Klasse des Elementes zurück.

`getValidators()`: Gibt eine Collection aller Validatoren zurück.

`get-/setValue([Object value])`: Gibt den Wert des Feldes zurück, bzw. setzt ihn auf *value*.

`is-/setInvalidAllowed([boolean invalidAllowed])`: Gibt an, ob dieses Feld ungültige Eingaben akzeptieren soll, bzw. setzt diesen Zustand auf *invalidAllowed*.

`is-/setInvalidCommitted([boolean isCommitted])`: Gibt an ob ungültige Eingaben in diesem Feld bei einem *commit()* in die Datenquelle geschrieben werden sollen, bzw. setzt diesen Zustand auf *isCommitted*.

isModified(): Gibt an, ob sich das Feld seit dem letzten Abgleich mit der Datenquelle verändert hat.

is-/setReadOnly([boolean readOnly]): Gibt an, ob das Feld nur lesbar ist, bzw. setzt diesen Zustand auf *readOnly*.

is-/setReadThrough([boolean readTrough]): Gibt an ob die Daten direkt aus der Datenquelle gelesen werden sollen, bzw. setzt diesen Zustand auf *readThrough*.

isValid(): Gibt an, ob die Eingaben in dem Feld gültig sind.

is-/setWriteThrough([boolean writeTrough]): Gibt an ob die Daten direkt in die Datenquelle geschrieben werden sollen, bzw. setzt diesen Zustand auf *writeThrough*.

validate(): Überprüft den Wert des Feldes auf seine Gültigkeit.

8.1.1 Label

Als erstes erstellen wir eine einfache “Hallo-Welt“ Applikation.

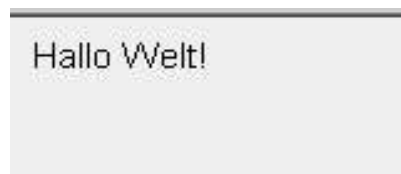


Abbildung 12: Beispiel eines Label

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_01_Label extends Application {

    public void init() {
        Window main = new Window("MillstoneDemo 1 - Label");
        setMainWindow(main);
        main.addComponent(new Label("Halo Welt!"));
    }
}
```

Allgemeine Attribute von:

AbstractComponent

Spezielle Attribute:

Label(): Erzeugt ein neues, leeres Label;

Label(Property contentSource[, int contentMode]): Erzeugt ein neues Label und liest den Inhalt aus *contentSource*. Der optionale *contentMode* gibt den Namensraum an.

Label(String content, int contentMode): Erzeugt ein neues Label mit dem Inhalt von *content*. Der optionale *contentMode* gibt den Namensraum an.

get-/setCaption: Die Label-Caption ist die optionale Überschrift des Label-Inhaltes.

get-/setValue: Der eigentliche Label-Inhalt

8.1.2 Layout

Um mehrere Elemente in einem Fenster verwenden zu können, muss man sie mittels eines Layouts gruppieren.

Hierfür gibt es das Ordered Layout (horizontale oder vertikale Reihung von Elementen) und das Grid Layout (tabellenförmiges Layout).

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_02_Layout extends Application {
    private GridLayout layout = new GridLayout(2, 1);

    public void init() {
        Window main = new Window("MillstoneDemo 02 - Layout", layout);
        setMainWindow(main);
        layout.addComponent(new Label("Hallo Welt!"), 0, 0);
        layout.addComponent(new Label("empty"), 1, 0);
    }
}
```

Allgemeine Attribute von:

AbstractComponent

Spezielle Attribute – GridLayout:

GridLayout(): Erzeugt ein neuer, leeres GridLayout.

GridLayout(int width, int heigh): Erzeugt ein *width* x *height* großes GridLayout.

addComponent(Component comp[, int x, int y]): Fügt die Komponente *comp* bei Breite *x* und Höhe *y* ein. Werden die Koordinaten nicht angegeben wird die momentane Position benutzt.

addComponent(Component component, int x1, int y1, int x2, int y2): Fügt die Komponente *component* in die Felder von Breite *x1* bis *x2* und Höhe *y1* bis *y2* ein.

getCursorX()/getCursorY(): gibt die momentane Position zurück.

get-/setHeight() / get-/setWidth(): gibt die Größe des GridLayouts zurück, bzw. setzt diese.

getComponentIterator(): gibt einen Iterator über die Komponenten zurück.

newLine(); Erzwingt die Positionierung der nächsten Komponente in einer neuen Zeile des GridLayouts.

removeComponent(Component component): entfernt die angegebene Komponente.

removeComponent(int x, int y): entfernt die Komponente bei *x*, *y*.

replaceComponent(Component oldComponent, Component newComponent): ersetzt die Komponente *oldComponent* mit der Komponente *newComponent*.

space(): rückt die momentane Position um eins weiter.

Spezielle Attribute – OrderedLayout:

OrderedLayout(): Erzeugt ein neuer, leeres OrderedLayout.

OrderedLayout(int orientation): Erzeugt ein OrderedLayout der Ausrichtung *orientation*.
(Wird gesetzt mittels OrderedLayout.ORTIENTATION_HORIZONTAL bzw. OrderedLayout.ORTIENTATION_VERTICAL).

addComponent(Component comp): Fügt die Komponente *comp* am Ende hinzu.

addComponent(Component component, int index): Fügt die Komponente *component* bei Position *index* ein.

addComponentAsFirst(Component comp): Fügt die Komponente *comp* am Anfang hinzu.

getComponentIterator(): gibt einen Iterator über die Komponenten zurück.

get-/setOrientation(): gibt die momentane Ausrichtung zurück, bzw. setzt diese.

removeComponent(Component component): entfernt die angegebene Komponente.

replaceComponent(Component oldComponent, Component newComponent): ersetzt die Komponente *oldComponent* mit der Komponente *newComponent*.

8.1.3 Button

Zur Interaktion mit der Applikation braucht man Buttons (bzw. Links als link-Style des Button, denn die Millstone Links sind nicht zur Interaktion mit der Applikation geeignet, nur zum Zugriff auf Ressourcen).



Abbildung 13: Beispiel eines Buttons mit Style Button (oben) und Link (unten)

Das Ziel eines Buttons kann entweder die Standard-Methode `buttonClick` sein (wofür die Applikation `Button.ClickListener` implementieren muss) oder eine beliebige Methode (ohne Parameter, bzw. mit Parameter `Button.ClickEvent`) sein.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_03_Button extends Application
    implements Button.ClickListener {
    private GridLayout layout = new GridLayout(2, 2);

    public void init() {
        Window main = new Window("MillstoneDemo 03 - Button", layout);
        setMainWindow(main);
        layout.addComponent(new Button
("myMethod", this, "myButtonClick"), 0, 0);
        layout.addComponent(new Button("stdMethod", this), 0, 1);
    }

    public void buttonClick(Button.ClickEvent event) {
        layout.removeComponent(1, 1);
        layout.addComponent(new Label("Button '" + ((Button)
event.getSource()).getCaption() + "' clicked."), 1, 1);
    }

    public void myButtonClick(Button.ClickEvent event) {
        layout.removeComponent(1, 0);
        layout.addComponent(new Label("myMethod called"), 1, 0);
    }
}
```

Allgemeine Attribute von:

`AbstractComponent`, `AbstractField`

Spezielle Attribute:

Button(String caption): Erzeugt einen neuen Button mit der Beschriftung *caption*.

Button(String caption, Button.ClickListener listener): Erzeugt einen neuen Button mit der

Beschriftung *caption* und verknüpft ihn mit der ClickListener-Klasse *listener* (und in dieser mit der Standardmethode *butonClick*).

Button(String caption, Object target, String methodName): Erzeugt einen neuen Button mit der Beschriftung *caption*. Und verknüpft ihn mit der Methode *methodName* der Klasse *target* (meist die eigene Klasse mittels *this*).

add-/removeListener(Button.ClickListener listener): Verknüpft den Button mit dem ClickListener, bzw. trennt diese Verknüpfung.

8.1.4 Textfield

Zur Texteingabe werden Textfelder benötigt.



Abbildung 14: Beispiel eines Eingabefeldes

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_04_TextField extends Application {
    private GridLayout layout = new GridLayout(2, 2);
    private TextField tf;

    public void init() {
        Window main = new Window("MillstoneDemo 04 - TextField",
layout);
        setMainWindow(main);
        tf = new TextField("Eingabe:");
        tf.setColumns(20);
        tf.setRows(3);
        layout.addComponent(tf, 0, 0);
        layout.addComponent(new Button("Submit", this, "processInput"),
0,1);
    }

    public void processInput(Button.ClickEvent event) {
        layout.removeComponent(1, 0);
        layout.addComponent(new Label("eingegebener Text: \n" +
tf.getValue().toString()), 1, 0);
    }
}
```

Allgemeine Attribute von:

AbstractComponent, AbstractField

Spezielle Attribute:

TextField(): Erzeugt ein neues, leeres TextField.

TextField(String caption): Erzeugt ein neues TextField mit der Überschrift *caption*.

TextField(Property dataSource): Erzeugt ein neues TextField ohne Überschrift und verknüpft es mit der DatenQuelle *dataSource*.

TextField(String caption, Property dataSource): Erzeugt ein neues TextField mit der Überschrift *caption* und verknüpft es mit der DatenQuelle *dataSource*.

TextField(String caption, String value): Erzeugt ein neuen TextField mit der Überschrift *caption* und dem Inhalt *value*.

get-/setColumns / get-/setRows([int size]): Liefert die Anzahl der Spalten bzw. Zeilen, bzw. setzt den Wert auf *size*.

get-/setNullRepresentation([String nullRepresentation]): Liefert den Wert, der an die Ausgabe geschickt wird, falls der Inhalt des TextField *null* ist, bzw. setzt diesen auf *nullRepresentation*.

is-/setNullSettingAllowed([boolean nullSettingAllowed]): Gibt an ob bei Inhalt des TextField von *null* statt dessen der Wert *nullRepresentation* an die Anzeige geschickt wird.

is-/setSecret([boolean secret]): Gibt an ob die eingegebenen Zeichen maskiert werden sollen, bzw setzt den Zustand auf *secret*. (Bei HTML-Ausgabe erhält man ein typisches Passwort-Feld).

is-/setWordwrap([boolean wordwrap]): Gibt an ob in einem (mehrzeiligen) TextField ein automatischer Zeilenumbruch vorgenommen werden soll, bzw setzt den Zustand auf *wordwrap*.

8.1.5 Datefield

Zur einfachen Datumseingabe gibt es das DateField-Element.



Abbildung 16: Beispiel eines Datumsfeldes (normal)



Abbildung 15: Beispiel eines Datumsfeldes (mit Style 'calendar')

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import java.util.Date;

public class MillstoneDemo_05_DateField extends Application {
    private GridLayout layout = new GridLayout(2, 2);
    private DateField df = null;

    public void init() {
        Window main = new Window("MillstoneDemo 05 - DateField",
        layout);
        setMainWindow(main);
        if(df == null || df.getValue() == null) df = new DateField
        ("Select Date:", new Date());
        layout.addComponent(df, 0, 0);
        layout.addComponent(new Button("Submit", this, "processInput"),
        0,1);
    }

    public void processInput(Button.ClickEvent event) {
        layout.removeComponent(1, 0);
        if(df.getValue() == null){
            int ef = df.getErrorField();
            int[] ev = df.getErrorValues();
            String[] fields = new String[]
            {"year", "month", "day", "hour", "minute", "second", "millisecond"};
            layout.addComponent(new Label("Datum: error: value '" + ev
            [ef] + "' does not exist for field '" + fields[ef] + "'."), 1, 0);
        } else layout.addComponent(new Label("Datum: " + df), 1, 0);
    }
}
```

Allgemeine Attribute von:

AbstractComponent, AbstractField

Spezielle Attribute:

DateField(): Erzeugt ein neues, leeres DateField.

DateField(String caption): Erzeugt ein neues DateField mit der Überschrift *caption*.

DateField(Property dataSource): Erzeugt ein neues DateField ohne Überschrift und verknüpft es mit der Datenquelle *dataSource*.

DateField(String caption, Date value): Erzeugt ein neues DateField mit der Überschrift *caption* und dem Datum *value*.

DateField(String caption, Property dataSource): Erzeugt ein neues DateField mit der Überschrift *caption* und verknüpft es mit der Datenquelle *dataSource*.

get-/setResolution([int resolution]): Liefert die Auflösung des DateField, bzw. setzt den Wert auf *resolution*. Die Auflösung des DateField gibt an bis zu welchem Element (Jahr, Monat, Tag, Stunde, Minute, Sekunde oder Millisekunde) das Datum in der Ausgabe dargestellt wird. (Dies geschieht mittels der Konstanten DateField.RESOLUTION_YEAR, DateField.RESOLUTION_MONTH, DateField.RESOLUTION_DAY, DateField.RESOLUTION_HOUR, DateField.RESOLUTION_MIN, DateField.RESOLUTION_SEC bzw. DateField.RESOLUTION_MSEC).

setPropertyDataSource(Property newDataSource): Verknüpft das DateField mit der neuen Datenquelle *newDataSource*.

8.1.6 Form

Dient als logischer Container für die verschiedenen Inputelemente, muss aber nicht verwendet werden. Man kann es als Buffer für die enthaltenen Inputelemente verwenden, oder um Bestätigungs- und Abbruchbuttons mit dem restlichen Formular zu verknüpfen.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_06_Form extends Application {
    private GridLayout layout = new GridLayout(1, 2);
    private TextField tf;
    private Form form;

    public void init() {
        Window main = new Window("MillstoneDemo 06 - Form", layout);
        setMainWindow(main);
        layout.addComponent(new Label("Form-Demo"), 0, 0);
        OrderedLayout ol = new OrderedLayout();
        form = new Form(ol);
        tf = new TextField("Eingabe:");
        tf.setColumns(20);
    }
}
```

```

        tf.setRows(3);
        ol.addComponent(tf);
        ol.addComponent(new Button("Submit", this, "processInput"));
        layout.addComponent(form, 0, 1);
    }

    public void processInput(Button.ClickEvent event) {
        layout.removeComponent(form);
        layout.addComponent(new Label("eingegebener Text: \n" +
tf.getValue().toString()), 0, 1);
    }
}

```

Allgemeine Attribute von:

AbstractComponent

Spezielle Attribute:

Form(): Erzeugt ein neues, leeres Form.

Form(Layout formLayout): Erzeugt ein neues Form mit dem Layout *formLayout*.

addField(Object propertyId, AbstractField field): Fügt das neue Feld *field* unter der ID *propertyId* ein.

addItemProperty(Object id, Property property): Fügt die neue Eigenschaft *property* unter der ID *Id* ein und versucht das zugehörige Feld automatisch zu erzeugen. Liefert *true* wenn das Feld erzeugt werden konnte, sonst *false*.

attach()/detach(): Benachrichtigt das Form, dass es mit einer Applikation verbunden bzw. von einer getrennt wurde.

commit(): Führt einen Aktualisierung der Änderungen seit dem letzten *commit* auf dem Datenmodell durch.

discard(): Verwirft alle Änderungen seit dem letzten *commit*.

getField(Object propertyId): Gibt das Feld als *AbstractField* zurück, das durch *propertyId* spezifiziert wird.

getItemDataSource(): Liefert die zugehörige Datenquelle als *Item* zurück.

getItemProperty(Object id): Liefert die zugehörige Eigenschaft des Feldes, das durch *id* spezifiziert wird.

getItemPropertyIds(): Liefert alle IDs als *Collection*.

getLayout: Liefert das Layout des Forms.

isModified(): Gibt an, ob seit dem letzten *commit()* eine Änderung vorgenommen wurde.

isReadThrough()/isWriteThrough(): Gibt an, ob direkt in die Datenquelle geschrieben, bzw. aus ihr gelesen werden soll.

removeAllProperties(): Entfernt alle Eigenschaften und Felder aus dem Form.

removeItemProperty(Object id): Entfernt die Eigenschaft und das Feld, das durch *id* spezifiziert wird aus dem Form..

replaceWithSelect(Object propertyId, Object[] values, Object[] descriptions): Ersetzt des durch *propertyId* spezifizierte Feld durch ein Select-Objekt mit den Beschreibungen *descriptions* und den Werten *values*. Dabei muss der momentane Wert des Feldes in *values* enthalten sein, *values* und *descriptions* müssen die gleiche Länge haben und dürfen nicht *null* enthalten.

setItemDataSource(Item newDataSource[, Collection propertyIds]): Setzt die Datenquelle auf *newDataSource* und limitiert die Inhalte des Forms auf durch *propertyIds* spezifizierte Eigenschaften.

setReadThrough(boolean readThrough): Gibt an ob die Daten direkt aus der Datenquelle gelesen werden sollen.

setWriteThrough(boolean writeThrough): Gibt an ob die Daten direkt in die Datenquelle geschrieben werden sollen.

8.1.7 Link

Millstone Links sind Verknüpfungen, die auf verschiedene Ressourcen zeigen. Externe Ressourcen sind alles außerhalb von Millstone mit direktem Zugriff, FileRessourcen sind für den Zugriff auf ein Dateisystem mittels Millstone, ClassRessourcen sind für den Zugriff auf Dateien der Millstone-Applikation und ThemeRessourcen für den Zugriff auf Dateien des Themes.



Abbildung 17: Beispiel für Links
(mit verschiedenen Zielen)

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.terminal.*;
import java.io.File;

public class MillstoneDemo_07_Link extends Application {
    private GridLayout layout = new GridLayout(1, 4);

    public void init() {
        Window main = new Window("MillstoneDemo 07 - Link", layout);
        setMainWindow(main);
        layout.addComponent(new Link("ClassResource", new ClassResource
("tree.jpg", this)), 0, 0);
        layout.addComponent(new Link("ThemeResource", new ThemeResource
("img/immediate.gif")), 0, 1);
        layout.addComponent(new Link("Fileresource", new FileResource
(new File("/test.txt"), this)), 0, 2);
        layout.addComponent(new Link("google.at", new ExternalResource
("http://google.at")), 0, 3);
    }
}
```

Allgemeine Attribute von:

AbstractComponent

Spezielle Attribute:

Link(): Erzeugt einen neuen, leeren Link.

Link(String caption, Resource resource): Erzeugt einen neuen Link mit der Bezeichnung *caption* und den Verweisziel *resource*.

Link(String caption, Resource resource, String targetName, int width, int height, int border): Erzeugt einen neuen Link mit der Bezeichnung *caption* und den Verweisziel *resource*. *targetName* bezeichnet das Browser-Fenster in dem der Link geöffnet werden soll (bei *null* oder einem leeren String wird das momentane Browser-Fenster verwendet). Die

Größe des Fensters wird von *height* und *width* bestimmt (wird unter Umständen vom Browser ignoriert). Der Rahmen des Fensters wird mittels *border* angegeben, wofür die Konstanten `TARGET_BORDER_DEFAULT`, `TARGET_BORDER_MINIMAL` und `TARGET_BORDER_NONE` verwendet werden sollten.

Link(Window window): Erzeugt einen neuen Link auf das Fenster *window*.

get-/setResource([Resource resource]): Gibt die Resource zurück, auf die der Link verweist, bzw. setzt sie auf *resource*.

get-/setTargetBorder([int targetBorder]): Gibt den Rahmen des Fensters zurück, bzw. setzt ihn auf *targetBorder*, wofür die Konstanten `TARGET_BORDER_DEFAULT`, `TARGET_BORDER_MINIMAL` und `TARGET_BORDER_NONE` verwendet werden sollten.

get-/setTargetHeight([int targetHeight]): Gibt die Fensterhöhe zurück, bzw. setzt sie auf *targetHeight*.

get-/setTargetName([int targetName]): Gibt den Namen des Browser-Fensters zurück, in dem der Link geöffnet werden soll (bei *null* oder einem leeren String wird das momentane Browser-Fenster verwendet), bzw. setzt ihn auf *targetName*.

get-/setTargetWidth([int targetWidth]): Gibt die Fensterbreite zurück, bzw. setzt sie auf *targetWidth*.

get-/setWindow(Window window): Liefert das Fenster zurück auf das der Link verweist bzw. setzt es auf *window*.

8.1.8 Panel

Ein Panel ist ein Container für andere Elemente und wird meist verwendet um einen Rahmen um diese Elemente zu ziehen.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.terminal.*;
import java.io.File;

public class MillstoneDemo_08_Panel extends Application {
    private GridLayout layout = new GridLayout(1, 4);

    public void init() {
```



Abbildung 18: Beispiel eines Panel

```

        Panel pan = new Panel("Resources", layout);
        Window main = new Window("MillstoneDemo 08 - Panel");
        setMainWindow(main);
        main.addComponent(pan);
        layout.addComponent(new Link("ClassResource", new ClassResource
("tree.jpg", this)), 0, 0);
        layout.addComponent(new Link("ThemeResource", new ThemeResource
("img/immediate.gif")), 0, 1);
        layout.addComponent(new Link("Fileresource", new FileResource
(new File("/test.txt"), this)), 0, 2);
        layout.addComponent(new Link("google.at", new ExternalResource
("http://google.at")), 0, 3);
    }
}

```

Allgemeine Attribute von:

AbstractComponent, AbstractField

Spezielle Attribute:

Panel(): Erzeugt ein neues, leeres Panel.

Panel(Layout layout): Erzeugt ein neues, leeres Panel mit dem Layout *layout*.

Panel(String caption): Erzeugt ein neues, leeres Panel mit der Überschrift *caption*.

Panel(String caption, Layout layout): Erzeugt ein neues, leeres Panel mit dem Layout *layout* und der Überschrift *caption*.

add-/removeComponent(Component component): Fügt die Komponente *componet* zum Panel hinzu bzw. entfernt sie.

getComponentIterator(): Gibt einen Iterator über die Komponenten des Panel zurück.

get-/setHeight([int height]): Gibt die Höhe des Panels zurück, bzw. setzt diese auf *height*.

get-/setHeightUnits([int units]): Gibt die Einheit an in der die Höhe definiert wird, bzw. setzt diese auf *units*. Dafür sollten die Konstanten UNIT_SYMBOLS, UNITS_CM, UNITS_EM, UNITS_EX, UNITS_INCH, UNITS_MM, UNITS_PERCENTAGE, UNITS_PICAS, UNITS_PIXELS bzw., UNITS_POINTS verwendet werden.

get-/setLayout([Layout layout]): Gibt das Layout des Panel zurück, bzw. setzt es auf *layout*.

get-/setWidth([int width]): Gibt die Breite des Panels zurück, bzw. setzt diese auf *width*.

get-/setWidthUnits([int units]): Gibt die Einheit an in der die Breite definiert wird, bzw.

setzt diese auf *units*. Dafür sollten die Konstanten `UNIT_SYMBOLS`, `UNITS_CM`, `UNITS_EM`, `UNITS_EX`, `UNITS_INCH`, `UNITS_MM`, `UNITS_PERCENTAGE`, `UNITS_PICAS`, `UNITS_PIXELS` bzw., `UNITS_POINTS` verwendet werden.

is-/setScrollable([boolean isScrollingEnabled]): Gibt an ob das Panel scrollbar ist, oder nicht, bzw. setzt den Wert auf *isScrollingEnabled*.

replaceComponent(Component oldComponent, Component newComponent): Ersetzt die Komponente *oldComponent* durch die Komponente *newKomponent*.

8.1.9 TabSheet

Tabsheets werden verwendet um



Abbildung 19: Beispiel eines Tabsheets

Applikationen zu untergliedern und übersichtlicher zu gestalten.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_09_TabSheet extends Application {

    public void init() {
        Window main = new Window("MillstoneDemo 09 - TabSheet");

        setMainWindow(main);
        TabSheet ts = new TabSheet();
        main.addComponent(ts);
        ts.addTab(new Label("Tab 1 Body"), "Tab 1 caption", null);
        ts.addTab(new Label("Tab 2 Body"), "Tab 2 caption", null);
        ts.addTab(new Label("Tab 3 Body"), "Tab 3 caption", null);
    }
}
```

Allgemeine Attribute von:

AbstractComponent

Spezielle Attribute:

TabSheet(): Erzeugt ein neues, leeres TabSheet.

add-/removeComponent(Component component): Fügt die Komponente *component* als neuen Tab zum TabSheet hinzu bzw. entfernt sie.

add-/removeListener(TabSheet.SelectedTabChangeListener listener): Fügt einen Listener für den Wechsel auf einen anderen Tab des TabSheets hinzu, bzw. entfernt ihn.

addTab(Component component, String caption, Resource icon): Fügt einen neuen Tab mit den Inhalt *component*, der Tab-Beschriftung *caption* und dem Icon *icon* hinzu.

areTabsHidden() / hideTabs(boolean tabsHidden): Gibt zurück ob die Zeile mit den Tabs angezeigt wird oder nicht, bzw. setzt diesen Zustand auf *tabsHidden*.

getComponentIterator(): Gibt einen Iterator über die Komponenten des TabSheet zurück.

getSelectedTab(): Gibt die Komponente des aktiven Tabs zurück.

get-/setTabCaption(Component component[, String caption]): Gibt die Tab-Beschriftung für die Komponente *component* zurück, bzw. setzt diesen Wert auf *caption*.

get-/setTabIcon(Component component[, Resource icon]): Gibt die Resource für das Tab-Icon der Komponente *component* zurück, bzw. setzt das Icon auf *icon*.

replaceComponent(Component oldComponent, Component newComponent): Ersetzt die Komponente *oldComponent* durch die Komponente *newKomponent*.

8.1.10 Select

Select wird für die verschiedenen Auswahlfelder verwendet. (Drop-Down, Checkboxes, Selectionlists, ...).

```

import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_10_Select extends
Application {
    private Select s1;
    private Select s2;
    private Select s3;
    private Select s4;
    private GridLayout layout = new GridLayout(2, 4);

    public void init() {
        Window main = new Window("MillstoneDemo 10
- Select", layout);
        setMainWindow(main);
        s1 = new Select("select 1");
        s1.addItem("item 1");
        s1.addItem("item 2");
        s1.addItem("item 3");
        s1.addItem("item 4");
        s2 = new Select("select 2");
        s2.addItem("item 1");
        s2.addItem("item 2");
        s2.addItem("item 3");
        s2.addItem("item 4");
        s2.setStyle("optiongroup");
        s3 = new Select("select 3");
        s3.addItem("item 1");
        s3.addItem("item 2");
        s3.addItem("item 3");
        s3.addItem("item 4");
        s3.setMultiSelect(true);
        s4 = new Select("select 4");
        s4.addItem("item 1");
        s4.addItem("item 2");
        s4.addItem("item 3");
        s4.addItem("item 4");
        s4.setMultiSelect(true);
        s4.setStyle("optiongroup");
        layout.addComponent(new Label("Selects:"), 0, 0);
        layout.addComponent(s1, 0, 1);
        layout.addComponent(s2, 1, 1);
        layout.addComponent(s3, 0, 2);
        layout.addComponent(s4, 1, 2);
        layout.addComponent(new Button("Submit", this,
"showSelected"));
    }

    public void showSelected() {
        OrderedLayout ol = new OrderedLayout();
        layout.removeComponent(0, 3);
        layout.addComponent(new Panel("Selected:", ol), 0, 3, 1, 3);
        Object[] c;
        c = s1.getItemIds().toArray();
        for(int i=0; i<c.length; i++)
            if(s1.isSelected(c[i]))
                ol.addComponent(new Label(s1.getCaption()+" / "+c
[i]));
        c = s2.getItemIds().toArray();
        for(int i=0; i<c.length; i++)
            if(s2.isSelected(c[i]))
                ol.addComponent(new Label(s2.getCaption()+" / "+c
[i]));
    }
}

```




Abbildung 20: Beispiel
verschiedener Selects

```

[i]));
    c = s3.getItemIds().toArray();
    for(int i=0; i<c.length; i++)
        if(s3.isSelected(c[i]))
            ol.addComponent(new Label(s3.getCaption()+" / "+c
[i]));
    c = s4.getItemIds().toArray();
    for(int i=0; i<c.length; i++)
        if(s4.isSelected(c[i]))
            ol.addComponent(new Label(s4.getCaption()+" / "+c
[i]));
    }
}

```

Allgemeine Attribute von:

AbstractComponent, AbstractField

Spezielle Attribute:

add-/removeContainerProperty(Object propertyId[, Class type, Object defaultValue]):

Fügt eine neue Eigenschaft der Klasse *type* und dem Standardwert *defaultValue* mit der ID *propertyId* zu allen Feldern hinzu.

addItem(): Fügt den Platz für ein neues Feld ein und gibt die zugehörige ID zurück.

add-/removeItem(Object itemId): Fügt ein neues Feld mit der ID *propertyId* hinzu und gibt dieses zurück, bzw. löscht das Feld.

add-/removeListener(Container.PropertySetChangeListener listener): Fügt einen neuen Listener hinzu, der auf eine Änderung einer Eigenschaft reagiert, bzw. löscht ihn.

add-/removeListener(Container.ItemSetChangeListener listener): Fügt einen neuen Listener hinzu, der auf eine Änderung eines Felds reagiert, bzw. löscht ihn.

containerItemSetChange(Container.ItemSetChangeEvent event): Informiert alle Listener über eine Änderung eines Feldes.

containerPropertySetChange(Container.PropertySetChangeEvent event): Informiert alle Listener über eine Änderung einer Eigenschaft.

containsId(Object itemId): Gibt zurück ob das Select ein Feld mit der ID *itemId* enthält.

fireItemSetChange(): Löst ein Ereignis aus, das über die Änderung eines Felds informiert.

firePropertySetChange(): Löst ein Ereignis aus, das über die Änderung einer Eigenschaft informiert.

get-/setContainerDataSource([Container newDataSource]): Gibt die Datenquelle zurück, bzw. setzt sie auf *newDataSource*.

getContainerProperty(Object itemId, Object propertyId): Gibt die Eigenschaft des Select zurück, die durch *itemId* und *propertyId* spezifiziert wird.

getContainerPropertyIds(): Gibt die IDs aller Eigenschaften des Select zurück.

getItem(Object itemId): Gibt das Feld (die Option) zurück, das durch *itemId* spezifiziert wird.

get-/setItemCaption(Object itemId[, String caption]): Gibt die Beschriftung der Feldes (der Option) zurück, das durch *itemId* spezifiziert wird, bzw. setzt den Wert auf *caption*.

get-/setItemCaptionMode([int mode]): Gibt die Art der Beschriftung zurück, bzw. setzt sie auf *mode*. (Dafür sollten folgende Konstanten verwendet werden:

ITEM_CAPTION_MODE_EXPLICIT,

ITEM_CAPTION_MODE_EXPLICIT_DEFAULTS_ID,

ITEM_CAPTION_MODE_ICON_ONLY, ITEM_CAPTION_MODE_ID,

ITEM_CAPTION_MODE_INDEX, ITEM_CAPTION_MODE_ITEM oder

ITEM_CAPTION_MODE_PROPERTY).

get-/setItemCaptionPropertyId([Object propertyId]): Gibt die ID der Eigenschaft zurück, die durch *propertyId* spezifiziert wird und für die Beschriftungen der Felder (Optionen) verwendet wird.

get-/setItemIcon(Object itemId[, Resource icon]): Gibt das Icon der Feldes (der Option) zurück, das durch *itemId* spezifiziert wird, bzw. setzt das Icon auf *icon*.

get-/setItemIconPropertyId([Object propertyId]): Gibt die ID der Eigenschaft zurück, die durch *propertyId* spezifiziert wird und für die Icons der Felder (Optionen) verwendet wird.

getItemIds(): Gibt die IDs aller Felder zurück.

get-/setNullSelectionItemId([Object nullSelectionItemId]): Gibt das Objekt zurück das für *null* steht (da *null* von der Datenschnittstelle Millstones nicht akzeptiert wird), bzw. setzt es auf *nullSelectionItemId*.

getType(Object propertyId): Gibt den Typ der Eigenschaft zurück, die durch *propertyId* spezifiziert wird.

Get-/setValue([Object newValue]): Gibt die ID des selektierten Felder (der selektierten Option) zurück, oder eine Liste von IDs wenn mehrere Felder selektiert wurden, bzw. setzt den Wert auf *newValue* (eine Collection im Fall von Mehrfachselektierbarkeit).

getVisibleItemIds(): Gibt eine Collection mit den IDs der sichtbaren Felder (Optionen) zurück.

is-/setMultiSelect([boolean multiSelect]): Gibt an ob mehrere Felder (Optionen) ausgewählt werden können, bzw. setzt diesen Zustand auf *multiSelect*.

is-/setNewItemAllowed([boolean allowNewOptions]): Gibt an ob das Hinzufügen neuer Felder (Optionen) erlaubt ist, bzw. setzt diesen Zustand auf *allowNewOptions*.

isSelected(Object itemId): Gibt zurück ob das Feld mit der ID *itemId* selektiert ist.

removeAllItems(): Entfernt alle Felder aus dem Select.

[un]select(Object itemId): (De)Selektiert das Feld mit der ID *itemId*.

size(): Gibt die Anzahl der Felder (Optionen) im Select zurück.

8.1.11 Table

Class	Category	Immediate
Label	Basic	no
Button	Basic	yes
Link	Basic	yes
Date Field	Basic	no
Text Field	Basic	no
Form	Basic	no
Image Map	Basic	yes
Select	Item Container	no
Tree	Item Container	yes
Table	Item Container	possible

Abbildung 22: Beispiel einer Tabelle

Zur tabellarischen
Darstellung von
Einträgen.
Optional
selektierbar und
sortierbar.

Class	Category	Immediate
Label	Basic	no
Button	Basic	yes
Link	Basic	yes
Date Field	Basic	no
Text Field	Basic	no
Form	Basic	no
Image Map	Basic	yes
Select	Item Container	no
Tree	Item Container	yes
Table	Item Container	possible

Abbildung 21: Beispiel einer Tabelle
(ohne JavaScript)

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.terminal.ThemeResource;

public class MillstoneDemo_11_Table extends Application {
    private Table tbl;
    private GridLayout layout = new GridLayout(1, 3);

    public void init() {
        Window main = new Window("MillstoneDemo 11 - Table", layout);
        setMainWindow(main);
        tbl = new Table("Table");
        tbl.setPageLength(10);
        tbl.addContainerProperty("first", String.class, "ERROR",
            "Class", new ThemeResource("icon/files/file.gif"), Table.ALIGN_LEFT);
        tbl.addContainerProperty("second", String.class, "ERROR",
            "Category", new ThemeResource("icon/files/folder.gif"), Table.ALIGN_LEFT);
        tbl.addContainerProperty("third", String.class, "ERROR",
            "Immediate", null, Table.ALIGN_CENTER);
        tbl.setColumnHeaderMode(1);
        tbl.setSelectable(true);
        tbl.setMultiSelect(true);
        tbl.setPlaceSelectAtEnd();
        tbl.setSortable(true);
        tbl.addItem(new Object[] { "Label", "Basic", "no", new Integer
(11));
        tbl.addItem(new Object[] { "Button", "Basic", "yes", new Integer
(12));
        tbl.addItem(new Object[] { "Link", "Basic", "yes", new Integer
(13));
        tbl.addItem(new Object[] { "Date Field", "Basic", "no", new
Integer(14));
        tbl.addItem(new Object[] { "Text Field", "Basic", "no", new
Integer(15));
        tbl.addItem(new Object[] { "Form", "Basic", "no", new Integer
(16));
        tbl.addItem(new Object[] { "Image Map", "Basic", "yes", new
Integer(17));
        tbl.addItem(new Object[] { "Select", "Item Container", "no", new
Integer(21));
        tbl.addItem(new Object[] { "Tree", "Item Container", "yes", new
Integer(22));
```

```

        tbl.addItem(new Object[] { "Table", "Item Container", "possible"},
new Integer(23));
        tbl.addItem(new Object[] { "Different Layouts", "Layouts", "no"},
new Integer(31));
        tbl.addItem(new Object[] { "Panel", "Layouts", "no"}, new Integer
(32));
        tbl.addItem(new Object[] { "Tab Sheet", "Layouts", "yes"}, new
Integer(33));
        tbl.addItem(new Object[] { "Window", "Layouts", "no"}, new Integer
(34));
        tbl.addItem(new Object[] { "Frame Window", "Layouts", "no"}, new
Integer(35));
        tbl.addItem(new Object[] { "Embedded Objects", "Data
Handling", "no"}, new Integer(41));
        tbl.addItem(new Object[] { "Upload", "Data Handling", "no"}, new
Integer(42));
        layout.addComponent(tbl, 0, 0);
        layout.addComponent(new Button("Submit", this, "showSelected"),
0, 1);
    }

    public void showSelected() {
        OrderedLayout ol = new OrderedLayout();
        layout.removeComponent(0, 2);
        layout.addComponent(new Panel("Selected:", ol), 0, 2);
        Object[] c = tbl.getItemIds().toArray();
        for(int i=0; i< c.length; i++){
            if(tbl.isSelected(c[i])){
                Object[] help = tbl.getItem(c[i]).getItemPropertyIds().
toArray();
                String output = new String();
                for(int j=0; j<help.length; j++){
                    output += tbl.getItem(c[i]).getItemProperty(help[j]).
getValue() + " / ";
                }
                output = output.substring(0, output.length() - 3);
                ol.addComponent(new Label("selected: "+output));
            }
        }
    }
}

```

Allgemeine Attribute von:

AbstractComponent, AbstractField, Select

Spezielle Attribute:

Table(): Erstellt eine neue, leere Tabelle.

Tree(java.lang.String caption): Erstellt eine neue, leere Tabelle mit der Überschrift *caption*.

Tree(java.lang.String caption, Container dataSource): Erstellt eine neue, leere Tabelle mit der Überschrift *caption* und verknüpft ihn mit der Datenquelle *dataSource*.

add-/removeActionHandler(Action.Handler actionHandler): Fügt den Action.Handler *actionHandler* zuTabelle hinzu, bzw. entfernt ihn. Der ActionHandler wird aufgerufen, wenn vom Benutzer eine Aktion auf der Tabelle durchgeführt wird.

add-/removeContainerProperty(Object propertyId[, Class type, Object defaultValue]): Fügt eine neue Spalte mit der ID *propertyId*, der Klasse *type* und dem Standardwert *defaultValue* zur Tabelle hinzu, bzw. entfernt die Spalte mit der ID *propertyId*.

addContainerProperty(Object propertyId, Class type, Object defaultValue, String columnHeader, Resource columnIcon, String columnAlignment): Fügt eine neue Spalte mit der ID *propertyId*, der Klasse *type* und dem Standardwert *defaultValue* zur Tabelle hinzu. Weiters hat die Spalte die Überschrift *columnHeader*, das Icon *columnIcon* und die Ausrichtung *columnAlignment* (Dafür sollte man die Konstanten ALIGN_CENTER, ALIGN_LEFT bzw. ALIGN_RIGHT verwenden).

add-/removeItem([Object[] cells,]Object itemId): Fügt eine neue Zeile mit den Werten *cells* unter der ID *itemId* zur Tabelle hinzu, bzw. löscht die Zeile.

containerItemSetChange(Container.ItemSetChangeEvent event): Benachrichtigt die Tabelle, dass sich die Datenquelle der Items geändert hat, und sie bei der nächsten Darstellung neu generiert werden muss.

containerPropertySetChange(Container.PropertySetChangeEvent event): Benachrichtigt die Tabelle, dass sich die Datenquelle der Eigenschaften geändert hat, und sie bei der nächsten Darstellung neu generiert werden muss.

focus(): Die Komponente Tabelle unterstützt keine Fokussierung.

get-/setColumnAlignments([String[] columnAlignments]): Gibt ein Array mit der Ausrichtung der Spalten zurück, bzw. setzt sie auf *columnAlignments*.

get-/setColumnHeaderMode([int columnHeaderMode]): Gibt die Darstellungsart der Spaltenüberschriften zurück, bzw. setzt diese auf *columnHeaderMode* (Dafür sollten die Konstanten COLUMN_HEADER_MODE_EXPLICIT, COLUMN_HEADER_MODE_EXPLICIT_DEFAULTS_ID, COLUMN_HEADER_MODE_HIDDEN bzw. COLUMN_HEADER_MODE_ID verwendet werden).

get-/setColumnHeaders([String[] columnHeader]): Gibt ein Array mit den Spaltenüberschriften zurück, bzw. setzt diese auf *columnHeaders*.

get-/setColumnIcons([Resource[] columnIcons]): Gibt ein Array mit den Spaltenicons zurück, bzw. setzt diese auf *columnIcons*.

get-/setCurrentPageFirstItemId([Object currentPageFirstItemId]): Gibt die ID der ersten Zeile auf der momentanen Seite der Tabelle zurück, bzw. setzt sie auf *currentPageFirstItemId*.

get-/setCurrentPageFirstItemIndex([int currentPageFirstItemIndex]): Gibt den Index der ersten Zeile auf der momentanen Seite der Tabelle zurück, bzw. setzt sie auf *currentPageFirstItemIndex*.

get-/setPageLength([int pageLength]): Gibt die Anzahl der Zeilen pro Seite zurück, bzw. setzt sie auf *pageLength*.

get-/setPlaceSelectAt([int position]): Gibt die Position der Selector-Spalte zurück, bzw. setzt sie auf *position*. Die Selector-Spalte wird nur bei selektierbarer Tabelle und fehlender JavaScript-Unterstützung angezeigt.

get-/setRowHeaderMode([int mode]): Gibt die Darstellungsart der Zeilenüberschriften (links neben der einzelnen Zeile) zurück, bzw. setzt diese auf *columnHeaderMode* (Dafür sollten die Konstanten ROW_HEADER_MODE_EXPLICIT, ROW_HEADER_MODE_EXPLICIT_DEFAULTS_ID, ROW_HEADER_MODE_HIDDEN, ROW_HEADER_MODE_ICON_ONLY, ROW_HEADER_MODE_ID, ROW_HEADER_MODE_INDEX, ROW_HEADER_MODE_ITEM bzw. ROW_HEADER_MODE_PROPERTY verwendet werden).

get-/setSortBy([int sortBy]): Gibt die Nummer der Spalte zurück nach der selektiert wird, bzw. setzt sie auf *sortBy*.

get-/setVisibleColumns([Object[] visibleColumns]): Gibt ein Array mit den sichtbaren Spalten zurück, bzw. setzt diese auf *visibleColumns*.

getVisibleItemIds(): Gibt eine Collection mit den IDs der sichtbaren Spalten zurück.

invertSelection(): Invertiert die momentane Auswahl der Tabellenzeilen.

is-/setPageBufferingEnabled([boolean pageBuffering]): Gibt zurück ob eine Seite bei jedem Aufruf neu generiert oder aus einem Zwischenspeicher genommen werden soll, bzw. setzt diesen Zustand auf *pageBuffering*.

is-/setSelectable([boolean selectable]): Gibt zurück ob die Tabelle selektierbar ist, bzw. setzt diesen Zustand auf *selectable*.

is-/setShowSelector([boolean showSelector]): Gibt zurück ob die Schaltflächen für die Selektion von allen oder keinen Zeilen, bzw. die Invertierung dieser Selektion angezeigt werden, bzw. setzt diesen Zustand auf *showSelector*.

is-/setSortable([boolean sortable]): Gibt zurück ob die Tabelle sortierbar ist, bzw. setzt diesen Zustand auf *sortable*.

refreshCurrentPage(): Initiiert eine Neugenerierung der aktuellen Seite der Tabelle. Dies ist nur notwendig wenn 'pageBuffering' aktiviert ist.

removeAllItems(): Entfernt alle Zeilen der Tabelle.

[un]selectAll(): Wählt alle bzw. keine der Tabellenzeilen aus.

setContainerDataSource(Container newDataSource): Setzt die Datenquelle der Tabelle auf *newDataSource*.

setNewItemAllowed(boolean allowNewOptions): Definiert ob neue Zeilen zur Tabelle hinzugefügt werden dürfen.

setPlaceSelectAtBegin(): Setzt die Position der Selektor-Spalte an den Beginn der Tabelle (links). Die Selektor-Spalte wird nur bei selektierbarer Tabelle und fehlender JavaScript-Unterstützung angezeigt.

setPlaceSelectAtEnd(): setPlaceSelectAtBegin(): Setzt die Position der Selektor-Spalte an das Ende der Tabelle (rechts). Die Selektor-Spalte wird nur bei selektierbarer Tabelle und fehlender JavaScript-Unterstützung angezeigt.

8.1.12 Tree

Bäume werden oft zur Realisation eines Menüs verwendet.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.data.*;
import org.millstone.base.data.Property.ValueChangeEvent;

public class MillstoneDemo_12_Tree extends
Application implements Property.ValueChangeListener{
    private Tree tree;
    private GridLayout layout = new GridLayout(2,
1);

    public void init() {
        Window main = new Window("MillstoneDemo
12 - Tree", layout);
        setMainWindow(main);
        tree = new Tree("Tree");
        tree.setStyle("link");
        tree.setSelectable(true);
        tree.setImmediate(true);
        tree.addItem("ti_1");
        tree.addItem("ti_11"); tree.setParent("ti_11", "ti_1");
        tree.addItem("ti_111"); tree.setParent("ti_111", "ti_11");
        tree.setChildrenAllowed("ti_111", false);
        tree.addItem("ti_112"); tree.setParent("ti_112", "ti_11");
        tree.addItem("ti_1121"); tree.setParent("ti_1121", "ti_112");
        tree.setChildrenAllowed("ti_1121", false);
        tree.addItem("ti_12"); tree.setParent("ti_12", "ti_1");
        tree.setChildrenAllowed("ti_12", false);
        tree.addItem("ti_2");
        tree.addItem("ti_21"); tree.setParent("ti_21", "ti_2");
        tree.setChildrenAllowed("ti_21", false);
        tree.addItem("ti_22"); tree.setParent("ti_22", "ti_2");
        tree.setChildrenAllowed("ti_22", false);
        tree.addItem("ti_3"); tree.setChildrenAllowed("ti_3", false);
        layout.addComponent(tree, 0, 0);
        tree.addListener(this);
    }

    public void valueChange(ValueChangeEvent event) {
        layout.removeComponent(1, 0);
        layout.addComponent(new Label("Show content for menu entry
"+event.getProperty()+"'."), 1, 0);
    }
}
```

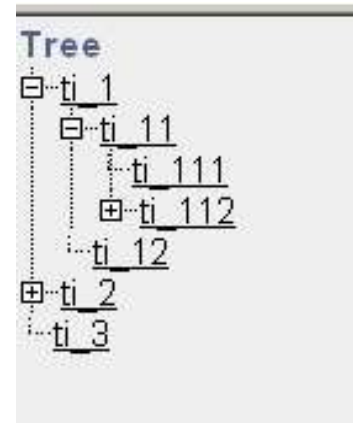


Abbildung 23: Beispiel eines Tree

Allgemeine Attribute von:

AbstractComponent, AbstractField, Select

Spezielle Attribute:

Tree(): Erstellt einen neuen, leeren Baum.

Tree(java.lang.String caption): Erstellt einen neuen, leeren Baum mit der Wurzelbezeichnung *caption*.

Tree(java.lang.String *caption*, Container *dataSource*): Erstellt einen neuen, leeren Baum mit der Wurzelbezeichnung *caption* und verknüpft ihn mit der Datenquelle *dataSource*.

add-/removeActionHandler(Action.Handler *actionHandler*): Fügt den Action.Handler *actionHandler* zum Baum hinzu, bzw. entfernt ihn. Der ActionHandler wird aufgerufen, wenn vom Benutzer eine Aktion auf dem Baum durchgeführt wird.

addListener(Tree.CollapseListener *listener*): Fügt den Listener *listener* zu dem Baum hinzu, bzw. entfernt ihn. Der Listener wird aufgerufen, wenn ein Baumknoten vom Benutzer zusammen geklappt wird.

addListener(Tree.ExpandListener *listener*): Fügt den Listener *listener* zu dem Baum hinzu, bzw. entfernt ihn. Der Listener wird aufgerufen, wenn ein Baumknoten vom Benutzer auseinander geklappt wird.

are-/setChildrenAllowed(Object *itemId*[, boolean *areChildrenAllowed*]): Gibt zurück ob für den durch *itemId* spezifizierten Knoten Kindelemente erlaubt sind, bzw. setzt diesen Zustand auf *areChildrenAllowed*.

collapse-/expandItem(Object *itemId*): Klappt den Knoten *itemId* zusammen bzw. auseinander.

collapse-/expandItemsRecursively(Object *itemId*): Klappt den Knoten *itemId* und alle Kindknoten zusammen bzw. auseinander.

fireCollapseEvent/fireExpandEvent(Object *itemId*): Informiert alle Listener, dass der Knoten *itemId* zusammen bzw. auseinander geklappt wurde.

focus(): Die Komponente Baum unterstützt keine Fokussierung.

getChildren(Object *itemId*): Gibt eine Collection aller Kindelemente des Knotens *itemId* zurück.

getParent(java.lang.Object *itemId*): Gibt den Elternknoten des Knotens *itemId* zurück.

getVisibleItemIds(): Gibt eine Collection aller sichtbaren Knoten zurück.

hasChildren(Object *itemId*): Gibt zurück ob der Knoten *itemId* Kindelemente hat.

isExpanded(Object itemId): Gibt zurück ob der Knoten *itemId* auseinander geklappt ist.

isRoot(Object itemId): Gibt zurück ob der Knoten *itemId* ein Wurzelknoten (also ein Knoten ohne Elternknoten) ist.

is-/setSelectable([boolean selectable]): Gibt zurück ob die Baumelemente selektiert werden können (eine Interaktionsmöglichkeit bieten), bzw. setzt diesen Zustand auf *selectable*.

rootItemIds(): Gibt eine Collection aller Wurzelknoten zurück.

setNewItemAllowed(boolean allowNewOptions): Setzt den Zustand ob neue Elemente zu dem Baum hinzugefügt werden können auf *allowedNewOptions*.

8.1.13 Window

Window wird verwendet um eine Applikation zu erzeugen, aber auch um weitere Fenster zu öffnen.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.terminal.*;

public class MillstoneDemo_13_Window
extends Application {

    public void init() {
        OrderedLayout ol = new
OrderedLayout();
        Window main = new Window
("MillstoneDemo 13 - (new) Window", ol);
        setMainWindow(main);
        ol.addComponent(new Button("Open
new window (with Google)", this,
"newWindow"));
    }

    public void newWindow() {
        Window newW = new Window("new Window: Google");
        newW.open(new ExternalResource("http://google.at"));
        this.addWindow(newW);
    }
}
```



Abbildung 24: Beispiel eines Window (Öffnen eines neuen Fensters)



Abbildung 25: Beispiel eines Window (Öffnen eines neuen Fensters ohne JavaScript)

Allgemeine Attribute von:

AbstractComponent, AbstractField, Panel

Spezielle Attribute:

Window(): Erzeugt ein neues, leeres Fenster.

Window(String caption): Erzeugt ein neues, leeres Fenster mit der Bezeichnung *caption*.

Window(String caption, Layout layout): Erzeugt ein neues, leeres Fenster mit der Bezeichnung *caption* und dem Layout *layout*.

add-/removeParameterHandler(ParameterHandler handler): Fügt den ParameterHandler *handler* hinzu, bzw. entfernt diesen.

add-/removeURIHandler(URIHandler handler): Fügt den URIHandler *handler* hinzu, bzw. entfernt diesen.

get-/setApplication([Application application]): Gibt die Millstone-Applikation zurück mit der dieses Fenster verbunden ist, bzw. verbindet dieses Fenster mit der Applikation *application*.

get-/setBorder([int border]): Gibt die Art des Rahmens des Fensters zurück, bzw. setzt diesen auf *border* (Dafür sollten die Konstanten BORDER_DEFAULT, BORDER_MINIMAL bzw. BORDER_NONE verwendet werden). Bei einem Web-Browser werden durch *border* die angezeigten Toolbars definiert.

getHeightUnits()/getWidthUnits(): Gibt die Einheit zurück in der Höhe bzw. Breite gemessen werden (Dafür sollten die Konstanten UNIT_SYMBOLS, UNITS_CM, UNITS_EM, UNITS_EX, UNITS_INCH, UNITS_MM, UNITS_PERCENTAGE, UNITS_PICAS, UNITS_PIXELS bzw. UNITS_POINTS verwendet werden).

get-/setName([String name]): Gibt den Namen des Fensters zurück, bzw. setzt ihn auf *name*.

get-/setParent([Component parent]): Gibt die übergeordnete Komponente des Fensters zurück, bzw. setzt diese auf *parent*.

get-/setTheme([String theme]): Gibt das aktuelle Theme für dieses Fenster zurück, bzw. setzt es auf *theme*.

handleParameters(Map parameters): Übergibt zu verarbeitende Parameter als eine Map. Die Namen sind alle vom Typ String und die Werte vom Typ String[].

handleURI(URL context, String relativeUri): Startet die rekursive Abarbeitung der URI.

open(Resource resource): Öffnet die Resource *resource* im aktuellen Fenster.

open(Resource resource, java.lang.String windowName): Öffnet die Resource *resource* im Fenster mit dem Namen *windowName*.

open(Resource resource, String windowName, int width, int height, int border): Öffnet die Resource *resource* im Fenster mit dem Namen *windowName* und versucht die Größe dieses Fensters auf eine Breite von *width* und eine Höhe von *height* einzustellen, mit einem Fensterrahmen von *border* (Dafür sollten die Konstanten BORDER_DEFAULT, BORDER_MINIMAL bzw. BORDER_NONE verwendet werden).

setHeightUnits()/setWidthUnits(int units): Setzt die Einheit in der Höhe bzw. Breite gemessen werden auf *units* (Dafür sollten die Konstanten UNIT_SYMBOLS, UNITS_CM, UNITS_EM, UNITS_EX, UNITS_INCH, UNITS_MM, UNITS_PERCENTAGE, UNITS_PICAS, UNITS_PIXELS bzw. UNITS_POINTS verwendet werden).

8.1.14 Embedded

Embedded Elemente werden verwendet um Bilder, Flash-Applikationen, Java-Applets, etc. In die Millstone-Applikation einzubinden.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.terminal.*;

public class MillstoneDemo_14_Embedded extends
Application {

    public void init() {
        OrderedLayout ol = new OrderedLayout();
        Window main = new Window("MillstoneDemo 14 - Embedded Objects",
ol);
        setMainWindow(main);
        ol.addComponent(new Embedded("Embedded", new ClassResource
("tree.jpg", this)));
    }
}
```

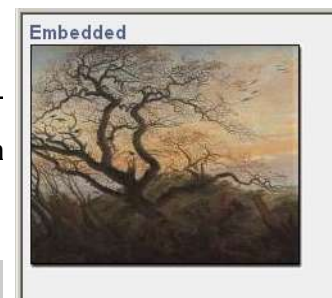


Abbildung 26: Beispiel eines Embedded-Objekt

Allgemeine Attribute von:

AbstractComponent

Spezielle Attribute:

Embedded(): Erzeugt ein neues, leeres Embedded-Objekt.

Embedded(String caption): Erzeugt ein neues, leeres Embedded-Objekt mit der Überschrift *caption*.

EmbeddedString caption, Resource source): Erzeugt ein neues Embedded-Objekt mit der Überschrift *caption*, und dem Inhalt *source*.

get-/setArchive([String archive]): Gibt das archiv-Attribut zurück, bzw. setzt es auf *archive*.

get-/setClassId([String classId]): Gibt das classId-Attribut zurück, bzw. setzt es auf *classId*.

get-/setCodebase([String codebase]): Gibt die Codebasis zurück, bzw. setzt diese auf *codebase*.

get-/setCodetype([String codetype]): Gibt den MIME-Typ des Codes zurück, bzw. setzt diesen auf *codetype*.

getHeight/-Width(): Gibt die Höhe bzw. Breite des Embedded-Objekts zurück.

getHeightUnits()/getWidthUnits(): Gibt die Einheit zurück in der Höhe bzw. Breite gemessen werden (Dafür sollten die Konstanten UNIT_SYMBOLS, UNITS_CM, UNITS_EM, UNITS_EX, UNITS_INCH, UNITS_MM, UNITS_PERCENTAGE, UNITS_PICAS, UNITS_PIXELS bzw. UNITS_POINTS verwendet werden).

get-/setMimeType([String mimetype]): Gibt den MIME-Typ des Objekts zurück, bzw. setzt diesen auf *mimetype*.

get-/setParameter(String name[, String value]): Gibt den Wert des Parameters *name* zurück, bzw. setzt diesen auf *value*.

getParameterNames(): Gibt einen Iterator über alle Parameter zurück.

get-/setSource([Resource source]): Gibt die Resource des Embedded-Objekts zurück, bzw. setzt diese auf *resource*.

get-/setStandby([String standby]): Gibt den Text zurück, der im Browser während des Ladens des Embedded-Objekts angezeigt wird, bzw. setzt diesen auf *standby*.

removeParameter(String name): Entfernt den Parameter *name*.

setHeight/-Width(int size): Setzt die Höhe bzw. die Breite des Embedded-Objekts auf size.

setHeightUnits()/setWidthUnits(int units): Setzt die Einheit in der Höhe bzw. Breite gemessen werden auf *units* (Dafür sollten die Konstanten UNIT_SYMBOLS, UNITS_CM, UNITS_EM, UNITS_EX, UNITS_INCH, UNITS_MM, UNITS_PERCENTAGE, UNITS_PICAS, UNITS_PIXELS bzw. UNITS_POINTS verwendet werden).

8.1.15 Upload

Die Uploadklasse dient zum Hinaufladen von Dateien..

```
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.OutputStream;
import org.millstone.base.terminal.StreamResource;
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.ui.Upload.FinishedEvent;

public class MillstoneDemo_15_Upload extends Application
    implements Upload.FinishedListener, Button.ClickListener {

    private Buffer buffer = new Buffer();
    private OrderedLayout layout;
    private Panel status = new Panel("Status: ");

    public void uploadFinished(FinishedEvent event) {
        status.removeAllComponents();
        if(buffer.getStream() == null)
            status.addComponent(
                new Label("Upload finished, but output buffer is
null!!"));
        else{
            status.addComponent(new Label("<b>Name:</b> " +
event.getFilename(),
                Label.CONTENT_XHTML));
            status.addComponent(new Label("<b>Mimetype:</b> " +
event.getMIMEType(),
                Label.CONTENT_XHTML));
            status.addComponent(new Label("<b>Size:</b> " +
event.getLength() + " bytes.",
                Label.CONTENT_XHTML));
            status.addComponent(new Link("Download " +
buffer.getFileName(),
                new StreamResource(buffer, buffer.getFileName(),
this)));
        }
    }
}
```



Abbildung 27: Beispiel eines Upload

```

    public void init() {
        layout = new OrderedLayout();
        Window main = new Window("MillstoneDemo 15 - Upload", layout);
        setMainWindow(main);
        Upload u = new Upload("Upload a file:", buffer);
        u.addListener(this);
        layout.addComponent(u);
        layout.addComponent(new Button("Upload", this));
        layout.addComponent(status);
    }

    public void buttonClick(Button.ClickEvent event) {
    }

    public class Buffer implements StreamResource.StreamSource,
Upload.Receiver {
        ByteArrayOutputStream outputBuffer = null;
        String mimeType;
        String fileName;

        public InputStream getStream() {
            if (outputBuffer == null) return null;
            return new ByteArrayInputStream(outputBuffer.toByteArray
());
        }

        public OutputStream receiveUpload(String filename, String
MIMEType) {
            fileName = filename;
            mimeType = MIMEType;
            outputBuffer = new ByteArrayOutputStream();
            return outputBuffer;
        }

        public String getFileName() {
            return fileName;
        }

        public String getMimeType() {
            return mimeType;
        }
    }
}

```

Allgemeine Attribute von:

AbstractComponent

Spezielle Attribute:

Upload(String caption, Upload.Receiver uploadReceiver): Erzeugt einen neuen Upload mit der Bezeichnung *caption* und dem Ziel *uploadReceiver*.

add-/removeListener(Upload.FailedListener listener): Fügt einen Listener hinzu der benachrichtigt wird wenn der Upload fehlschlägt, bzw. entfernt den Listener.

add-/removeListener(Upload.FinishedListener listener): Fügt einen Listener hinzu der benachrichtigt wird wenn der Upload entgegengenommen wurde, bzw. entfernt den Listener.

add-/removeListener(Upload.SucceededListener listener): Fügt einen Listener hinzu der benachrichtigt wird wenn der Upload erfolgreich beendet wurde, bzw. entfernt den Listener.

fireUploadInterrupted(String filename, String MIMEType, long length): Löst ein Ereignis aus, das mitteilt, dass ein Upload abgebrochen wurde.

fireUploadReceived(String filename, String MIMEType, long length): Löst ein Ereignis aus, das mitteilt, dass ein Upload entgegengenommen wurde.

fireUploadSuccess(String filename, String MIMEType, long length): Löst ein Ereignis aus, das mitteilt, dass ein Upload erfolgreich abgeschlossen wurde.

get-/setReceiver([Upload.Receiver receiver]): Gibt die Empfängerklasse der Uploads zurück, bzw. setzt sie auf *receiver*.

8.1.16 ImageMap

ImageMaps sind verweissensitive Grafiken, als Ziele können theoretisch alle Millstone-Ressourcen dienen.

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.ui.ImageMap.ClickEvent;
import org.millstone.base.terminal.*;

public class MillstoneDemo_16_ImageMap extends
Application implements ImageMap.ClickListener {
    private GridLayout layout;

    public void init() {
        layout = new GridLayout(1,2);
        Window main = new Window("MillstoneDemo 16 - ImageMap",
layout);
        setMainWindow(main);
        ImageMap im = new ImageMap("demo", new ClassResource("imagemap-
demo.jpg", this), 214,173, this);
        im.setCaption("ImageMap");
        ImageMap.Shape sh;
        sh = im.new Circle(im.new ImageMapResource("shape: green"),
"shape: green",
            im.new Point(72, 42), 30);
        im.addShape(sh);
        sh = im.new Rectangle(new ExternalResource("http://google.at"),
"Google", im.new Point(24, 107), im.new Point(100, 150));
        im.addShape(sh);
    }
}
```

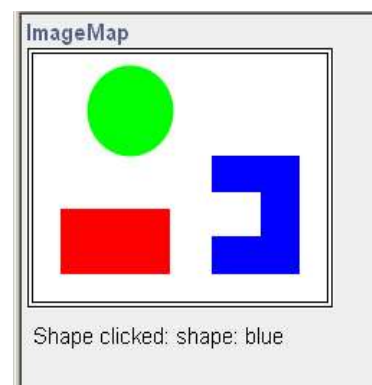


Abbildung 28: Beispiel einer
Imagemap

```

        sh = im.new Polygon(im.new ImageMapResource("shape: blue"),
"shape: blue");
        ((ImageMap.Polygon) sh).addPoint(im.new Point(129, 72));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(190, 72));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(190, 150));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(129, 150));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(129, 125));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(163, 125));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(163, 96));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(129, 96));
        im.addShape(sh);
        layout.addComponent(im, 0, 0);
    }

    public void imagemapClick(ClickEvent event) {
        layout.removeComponent(0, 1);
        layout.addComponent(new Label("Shape clicked: " +
((ImageMap.ImageMapResource)
        event.getImageMap().getFiredShape().getResource()).
getContent()), 0, 1);
    }
}

```

Allgemeine Attribute von:

AbstractComponent, AbstractField

Spezielle Attribute:

ImageMap(String mapName, Resource image, int width, int height): Erzeugt eine neue ImageMap mit dem Namen *mapName* und dem Bild *image*. Durch *width* und *height* werden die Ausmaße des Bildes angegeben.

ImageMap(String mapName, Resource image, int width, int height, ImageMap.ClickListener listener): Erzeugt eine neue ImageMap mit dem Namen *mapName*, dem Bild *image* und verknüpft sie mit der ClickListener-Klasse *listener* (und in dieser mit der Standardmethode *imagemapClick*). Durch *width* und *height* werden die Ausmaße des Bildes angegeben.

ImageMap(.String mapName, Resource image, int width, int height, Object target, methodName): Erzeugt eine neue ImageMap mit dem Namen *mapName*, dem Bild *image* und verknüpft sie mit der Methode *methodName* der Klasse *target* (meist die eigene Klasse mittels *this*). Durch *width* und *height* werden die Ausmaße des Bildes angegeben.

add-/removeListener(ImageMap.ClickListener listener): Fügt den neuen Listener *listener* hinzu bzw. entfernt diesen.

addShape(ImageMap.Shape shape): Definiert einen neuen verweissensitiven Bereich (Shape) in der ImageMap. Dies geschieht mittels von ImageMap.Shape abgeleiteten Klassen, die den drei Möglichkeiten in HTML entsprechen.

- **ImageMap.Circle(Resource target, String alternative, ImageMap.Point center, int radius):** Erstellt einen neuen verweissensitiven Bereich in Kreisform mit dem Ziel *target*, dem HTML-Alternativtext *alternative* dem Kreiszentrum bei Punkt *center* und dem Radius *radius*.
- **ImageMap.Rectangle(Resource target, String alternative, ImageMap.Point topLeft, ImageMap.Point bottomRight):** Erstellt einen neuen verweissensitiven Bereich in Rechtecksform mit dem Ziel *target*, dem HTML-Alternativtext *alternative* dem linken, oberen Punkt *topLeft* und dem rechten, unteren Punkt *bottomRight*.
- **ImageMap.Polygon**
 - **ImageMap.Polygon(Resource target, String alternative):** Erstellt einen neuen verweissensitiven Bereich in Polygonform mit dem Ziel *target*, dem HTML-Alternativtext *alternative* dem linken aber ohne Punkte.
 - **ImageMap.Polygon(Resource target, String alternative, ImageMap.Polygon.PolygonPoint head):** Erstellt einen neuen verweissensitiven Bereich in Polygonform mit dem Ziel *target*, dem HTML-Alternativtext *alternative* dem linken und dem Startpunkt *head* (mit dessen eventuell vorhandenen Folgepunkten).
 - **addPoint(ImageMap.Point p):** Fügt einen neuen Punkt *p* zur Liste der Polygonpunkten hinzu.

Um die verweissensitiven Bereiche zu definieren werden folgende Hilfsklassen verwendet:

- **ImageMap.Point**
 - **ImageMap.Point(int x, int y):** Erstellt einen neuen Punkt mit der horizontalen Position *x* und der vertikalen Position *y*.
- **ImageMap.Polygon.Polygon**
 - **PointImageMap.Polygon.PolygonPoint(ImageMap.Point p):** Erstellt einen neuen Polygonpunkt mittels der Koordinaten der Punktes *p*.
 - **ImageMap.Polygon.PolygonPoint(int x, int y):** Erstellt einen neuen Polygonpunkt mit der horizontalen Position *x* und der vertikalen Position *y*.
 - **next:** enthält den nächsten Polygonpunkt bzw. *null* am Ende der Liste der Punkte.

getFiredShape(): Gibt den Shape zurück, durch den das Ereignis ausgelöst wurde.

get-/setImageAlternativeText([String imageAlt]): Gibt den HTML-Alternativtext für die ImageMap zurück, bzw. setzt diesen auf *imageAlt*.

getShapes(): Gibt ein Array mit allen in der ImageMap enthalten Shapes zurück.

8.1.17 FrameWindow

FrameWindows können mehrere Windows parallel anzeigen, und miteinander verbinden. Sie sollten aber nur wenn unbedingt nötig eingesetzt werden, da sie denn Zugriff für Sehbehinderte ziemlich erschweren und ohne JavaScript nicht korrekt funktionieren.

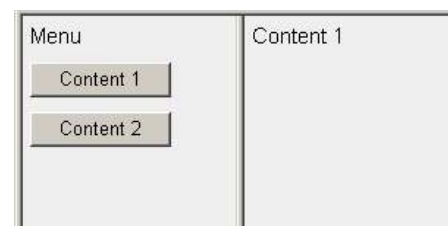


Abbildung 29: Beispiel eines FrameWindow

```
import org.millstone.base.Application;
import org.millstone.base.ui.*;

public class MillstoneDemo_17_FrameWindow extends Application implements
Button.ClickListener {
    private FrameWindow main;
    private Window menu, cont1, cont2;

    public void init() {
        main = new FrameWindow("MillstoneDemo 17 - Frame Window");
        setMainWindow(main);
        menu = new Window();
        menu.addComponent(new Label("Menu"));
        menu.addComponent(new Button("Content 1", this));
        menu.addComponent(new Button("Content 2", this));
        cont1 = new Window();
        cont1.addComponent(new Label("Content 1"));
        cont2 = new Window();
        cont2.addComponent(new Label("Content 2"));
        main.getFrameset().newFrame(menu);
        main.getFrameset().newFrame(cont1);
        main.getFrameset().getFrame(0).setRelativeSize(25);
    }

    public void buttonClick(Button.ClickEvent event) {
        if (event.getButton().getCaption().equals("Content 1")) {
            main.getFrameset().removeFrame(main.getFrameset().
getFrame(1));
            main.getFrameset().newFrame(cont1);
        }
        if (event.getButton().getCaption().equals("Content 2")) {
            main.getFrameset().removeFrame(main.getFrameset().
getFrame(1));
            main.getFrameset().newFrame(cont2);
        }
    }
}
```

```
    }  
}  
}
```

Allgemeine Attribute von:

AbstractComponent, AbstractField, Panel, Window

Spezielle Attribute:

addComponent(Component c): Wird nicht direkt unterstützt.

getFrameset(): Gibt die Frames als Instanz der Klasse `FrameWindow.Frameset` zurück.

is-/setScrollable([boolean isScrollingEnabled]): Wird nicht direkt unterstützt.

setApplication(Application application): Setzt die Applikation für alle enthaltenen Frames

setScrollOffsetX/-Y(int pixelsScrolled): Wird nicht direkt unterstützt.

Zum definieren von Frames werden folgende zwei Unterklassen verwendet:

- **FrameWindow.Frame**
 - **FrameWindow.Frame():** Wird nicht verwendet.
 - **getName():** Gibt den Namen des Frames zurück.
 - **getParentFrameset():** Gibt das übergeordnete `FrameWindow.Frameset` zurück
 - **getResource():** Gibt die dem `FrameWindow.Frameset` zugeordnete Resource zurück.
 - **getURL():** Gibt den Uniform Resource Locator zurück (als `java.net.URL`).
 - **getWindow():** Gibt das enthaltene Fenster zurück.
 - **setAbsoluteSize(int widthInPixels):** Setzt die Größe des Frames absolut auf *widthInPixel* Pixel (je nach Orientierung des Frames wird damit die Breite oder die Höhe angegeben).
 - **setFreeSize():** Entfernt vorherige Größeneingaben für dieses Frame.
 - **setRelativeSize(int widthInPercents):** Setzt die Größe des Frames relativ auf *widthInPercent* Prozent (je nach Orientierung des Frames wird damit die Breite oder die Höhe angegeben).
- **FrameWindow.Frameset**
 - **FrameWindow.Frameset():** Wird nicht verwendet.

- **getFrame(int index):** Gibt das Frame mit dem Index *index* zurück.
- **getFrame(String name):** Gibt das Frame mit dem Namen *name* zurück.
- **getFrames():** Gibt alle enthaltenen Frames als Liste zurück.
- **isVertical():** Gibt an, ob das Frameset horizontal oder vertikal orientiert ist.
- **newFrame(Resource resource, String name[, int index]):** Fügt ein neues Frame mir der Resource *resource* und dem Namen *name* hinzu (wird kein Index angegeben, wird das neue Frame am Ende hinzugefügt, ansonsten an Position *index* eingefügt).
- **newFrame(URL url, String name[, int index]):** Fügt ein neues Frame mir der Zieladresse *url* und dem Namen *name* hinzu (wird kein Index angegeben, wird das neue Frame am Ende hinzugefügt, ansonsten an Position *index* eingefügt).
- **newFrame(Window window[, int index]):** Fügt ein neues Frame mir dem Inhalt des Window *window* hinzu (wird kein Index angegeben, wird das neue Frame am Ende hinzugefügt, ansonsten an Position *index* eingefügt).
- **newFrameset(boolean isVertical, int index):** Fügt ein neues Frameset an der Position *index* ein.
- **removeAllFrames():** Entfernt alle Frames und Unter-Frameset
- **removeFrame(FrameWindow.Frame frame):** Entfernt das Frame *frame* aus dem Frameset.
- **setVertical(boolean isVertical):** Definiert, ob das FrameWindow.Frameset horizontal oder vertikal orientiert werden soll.
- **size():** Gibt die Anzahl der Frames im Frameset zurück.

8.1.18 Beispiel mit allen Elementen

```

import org.millstone.base.Application;
import org.millstone.base.ui.*;
import org.millstone.base.data.Property;
import org.millstone.base.terminal.*;
import org.millstone.base.ui.Upload.FinishedEvent;
import org.millstone.base.terminal.StreamResource;
import java.util.Date;
import java.io.File;
import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.InputStream;
import java.io.OutputStream;

public class MillstoneDemo_00_AllElements
extends Application implements
Button.ClickListener,
Property.ValueChangeListener,
Upload.FinishedListener, ImageMap.ClickListener {
    private FrameWindow frameWind;
    private GridLayout demo;
    private TextField tf;
    private DateField df;
    private Form form;
    private Select s1, s2, s3, s4;
    private Table tbl;
    private Buffer uploadBuf = new Buffer();
    private Panel upload = new Panel("Status: ");

    public void init() {
        Window main = new Window("MillstoneDemo 00 - All Elements");
        setMainWindow(main);
        /* menu */
        demo = new GridLayout(2,1);
        main.setLayout(demo);
        Tree tree = new Tree("Menü");
        tree.setStyle("link");
        tree.setSelectable(true);
        tree.setImmediate(true);
        tree.addItem("Basic");
        tree.addItem("Label"); tree.setParent("Label", "Basic");
        tree.setChildrenAllowed("Label", false);
        tree.addItem("Button"); tree.setParent("Button", "Basic");
        tree.setChildrenAllowed("Button", false);
        tree.addItem("Link"); tree.setParent("Link", "Basic");
        tree.setChildrenAllowed("Link", false);
        tree.addItem("Date Field"); tree.setParent("Date Field",
"Basic"); tree.setChildrenAllowed("Date Field", false);
        tree.addItem("Text Field"); tree.setParent("Text Field",
"Basic"); tree.setChildrenAllowed("Text Field", false);
        tree.addItem("Form"); tree.setParent("Form", "Basic");
        tree.setChildrenAllowed("Form", false);
        tree.addItem("Image Map"); tree.setParent("Image Map",
"Basic"); tree.setChildrenAllowed("Image Map", false);
        tree.addItem("Item Container");
        tree.addItem("Select"); tree.setParent("Select", "Item
Container"); tree.setChildrenAllowed("Select", false);

```

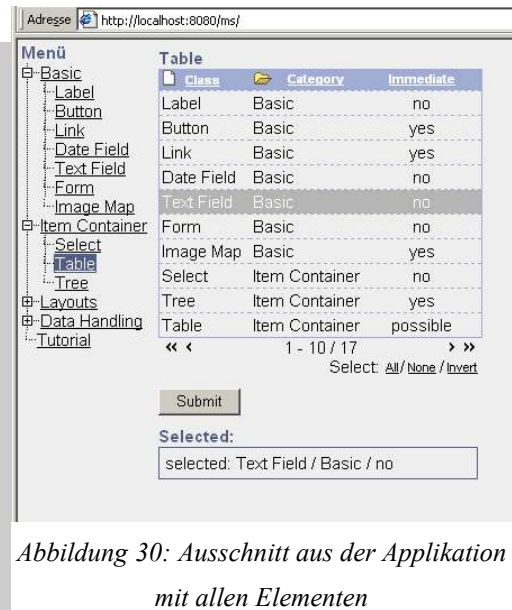


Abbildung 30: Ausschnitt aus der Applikation mit allen Elementen

```

        tree.addItem("Table"); tree.setParent("Table", "Item
Container"); tree.setChildrenAllowed("Table", false);
        tree.addItem("Tree"); tree.setParent("Tree", "Item Container");
tree.setChildrenAllowed("Tree", false);
        tree.addItem("Layouts");
        tree.addItem("Different Layouts"); tree.setParent("Different
Layouts", "Layouts"); tree.setChildrenAllowed("Different Layouts", false);
        tree.addItem("Panel"); tree.setParent("Panel", "Layouts");
tree.setChildrenAllowed("Panel", false);
        tree.addItem("Tab Sheet"); tree.setParent("Tab Sheet",
"Layouts"); tree.setChildrenAllowed("Tab Sheet", false);
        tree.addItem("Window"); tree.setParent("Window", "Layouts");
tree.setChildrenAllowed("Window", false);
        tree.addItem("Frame Window"); tree.setParent("Frame Window",
"Layouts"); tree.setChildrenAllowed("Frame Window", false);
        tree.addItem("Data Handling");
        tree.addItem("Embedded Objects"); tree.setParent("Embedded
Objects", "Data Handling"); tree.setChildrenAllowed("Embedded
Objects", false);
        tree.addItem("Upload"); tree.setParent("Upload", "Data
Handling"); tree.setChildrenAllowed("Upload", false);
        tree.addItem("Tutorial"); tree.setChildrenAllowed("Tutorial",
false);

        demo.addComponent(tree, 0, 0);
        tree.addListener(this);

        demo.addComponent(new Label("Select from the tree at the
left."), 1,0);
    }

    public void buttonClick(Button.ClickEvent event) {

    }

    public void valueChange(Property.ValueChangeEvent event) {
        String ep = event.getProperty().toString();
        demo.removeComponent(1,0);
        OrderedLayout cont = new OrderedLayout();
        demo.addComponent(cont, 1,0);
        if(ep.equals("Tutorial")){
            cont.addComponent(new Label("Tutorial should be opened in
a new window!"));
            Window newW = new Window("new Window: Google");
            newW.open(new ExternalResource
("http://uni.diquadrat.net/dipl/Millstone_mod-Di%b2/tutorial"));
            this.addWindow(newW);
        }else{
            if(ep.equals("Label")){
                cont.addComponent(new Label("Hallo Welt!"));
            }else if(ep.equals("Different Layouts")){
                Panel p1 = new Panel();
                OrderedLayout ol = new OrderedLayout();
                p1.setCaption("OrderedLayout");
                p1.setLayout(ol);
                ol.addComponent(new Label("1"));
                ol.addComponent(new Label("2"));
                ol.addComponent(new Label("3"));
                ol.addComponent(new Label("4"));
                ol.addComponent(new Label("5"));
                cont.addComponent(p1);
                Panel p2 = new Panel();
                GridLayout gl = new GridLayout(2, 3);
                p2.setCaption("GridLayout (2x3)");
            }
        }
    }

```

```

        p2.setLayout(gl);
        gl.addComponent(new Label("1"), 0, 0);
        gl.addComponent(new Label("2"), 0, 1);
        gl.addComponent(new Label("3"), 0, 2);
        gl.addComponent(new Label("4"), 1, 0);
        gl.addComponent(new Label("5"), 1, 1);
        cont.addComponent(p2);
    }else if(ep.equals("Button")){
        cont.addComponent(new Button("<-> Change Language",
this, "chngLang"));
    }else if(ep.equals("Text Field")){
        tf = new TextField("Eingabe:");
        tf.setColumns(20);
        tf.setRows(3);
        OrderedLayout ol = new OrderedLayout
(OrderedLayout.ORIENTATION_HORIZONTAL);
        ol.addComponent(tf);
        ol.addComponent(new Button("Submit", this,
"processTextField"));
        cont.addComponent(ol);
    }else if(ep.equals("Date Field")){
        if(df == null || df.getValue() == null) df = new
DateField("Select Date:", new Date());
        cont.addComponent(df);
        cont.addComponent(new Button("Submit", this,
"processDateField"));
    }else if(ep.equals("Form")){
        GridLayout gl = new GridLayout(1,2);
        gl.addComponent(new Label("Form-Demo"), 0, 0);
        OrderedLayout ol = new OrderedLayout();
        form = new Form(ol);
        tf = new TextField("Eingabe:");
        tf.setColumns(20);
        tf.setRows(3);
        ol.addComponent(tf);
        ol.addComponent(new Button("Submit", this,
"processForm"));
        gl.addComponent(form, 0, 1);
        cont.addComponent(gl);
    }else if(ep.equals("Link")){
        cont.addComponent(new Link("ClassResource", new
ClassResource("tree.jpg", this));
        cont.addComponent(new Link("ThemeResource", new
ThemeResource("img/immediate.gif"));
        cont.addComponent(new Link("Fileresource", new
FileResource(new File("/test.txt"), this));
        cont.addComponent(new Link("google.at", new
ExternalResource("http://google.at"));
    }else if(ep.equals("Panel")){
        Panel pan = new Panel("This is a Panel");
        pan.addComponent(new Label("with Content"));
        cont.addComponent(pan);
    }else if(ep.equals("Tab Sheet")){
        TabSheet ts = new TabSheet();
        Label help = new Label("Tab 1 Body");
        ts.addTab(help, "Tab 1 caption",null);
        ts.setTabIcon(help, new ThemeResource
("icon/plus.gif"));
        ts.addTab(new Label("Tab 2 Body"), "Tab 2
caption",null);
        help = new Label("Tab 3 Body");

```

```

        ts.addTab(help, "Tab 3 caption", null);
        ts.setTabIcon(help, new ThemeResource
("icon/plus.gif"));
        cont.addComponent(ts);
    } else if (ep.equals("Select")) {
        s1 = new Select("select 1");
        s1.addItem("item 1");
        s1.addItem("item 2");
        s1.addItem("item 3");
        s1.addItem("item 4");
        s2 = new Select("select 2");
        s2.addItem("item 1");
        s2.addItem("item 2");
        s2.addItem("item 3");
        s2.addItem("item 4");
        s2.setStyle("optiongroup");
        s3 = new Select("select 3");
        s3.addItem("item 1");
        s3.addItem("item 2");
        s3.addItem("item 3");
        s3.addItem("item 4");
        s3.setMultiSelect(true);
        s4 = new Select("select 4");
        s4.addItem("item 1");
        s4.addItem("item 2");
        s4.addItem("item 3");
        s4.addItem("item 4");
        s4.setMultiSelect(true);
        s4.setStyle("optiongroup");
        GridLayout gl = new GridLayout(2, 4);
        gl.addComponent(new Label("Selects:"), 0, 0);
        gl.addComponent(s1, 0, 1);
        gl.addComponent(s2, 1, 1);
        gl.addComponent(s3, 0, 2);
        gl.addComponent(s4, 1, 2);
        gl.addComponent(new Button("Submit", this,
"showSelectSelected"));
        cont.addComponent(gl);
    } else if (ep.equals("Table")) {
        tbl = new Table("Table");
        tbl.setPageLength(10);
        tbl.addContainerProperty("first", String.class,
"ERROR", "Class", new ThemeResource("icon/files/file.gif"),
Table.ALIGN_LEFT);
        tbl.addContainerProperty("second", String.class,
"ERROR", "Category", new ThemeResource("icon/files/folder.gif"),
Table.ALIGN_LEFT);
        tbl.addContainerProperty("third", String.class,
"ERROR", "Immediate", null, Table.ALIGN_CENTER);
        tbl.setColumnHeaderMode(1);
        tbl.setSelectable(true);
        tbl.setMultiSelect(true);
        tbl.setPlaceSelectAtEnd();
        tbl.setSortable(true);
        tbl.addItem(new Object[] {"Label", "Basic", "no"},
new Integer(11));
        tbl.addItem(new Object[] {"Button", "Basic",
"yes"}, new Integer(12));
        tbl.addItem(new Object[] {"Link", "Basic", "yes"},
new Integer(13));
        tbl.addItem(new Object[] {"Date Field", "Basic",
"no"}, new Integer(14));

```



```

tbl.addItem(new Object[] {"Text Field", "Basic",
"no"}, new Integer(15));
tbl.addItem(new Object[] {"Form", "Basic", "no"},
new Integer(16));
tbl.addItem(new Object[] {"Image Map", "Basic",
"yes"}, new Integer(17));
tbl.addItem(new Object[] {"Select", "Item
Container", "no"}, new Integer(21));
tbl.addItem(new Object[] {"Tree", "Item Container",
"yes"}, new Integer(22));
tbl.addItem(new Object[] {"Table", "Item
Container", "possible"}, new Integer(23));
tbl.addItem(new Object[] {"Different Layouts",
"Layouts", "no"}, new Integer(31));
tbl.addItem(new Object[] {"Panel", "Layouts",
"no"}, new Integer(32));
tbl.addItem(new Object[] {"Tab Sheet", "Layouts",
"yes"}, new Integer(33));
tbl.addItem(new Object[] {"Window", "Layouts",
"no"}, new Integer(34));
tbl.addItem(new Object[] {"Frame Window",
"Layouts", "no"}, new Integer(35));
tbl.addItem(new Object[] {"Embedded Objects", "Data
Handling", "no"}, new Integer(41));
tbl.addItem(new Object[] {"Upload", "Data
Handling", "no"}, new Integer(42));
cont.addComponent(tbl);
cont.addComponent(new Button("Submit", this,
"showTableSelected"));
} else if (ep.equals("Tree")) {
cont.addComponent(new Label("Look at the tree in
the left.));
} else if (ep.equals("Window")) {
cont.addComponent(new Button("Open new window (with
Google)", this, "newWindow"));
} else if (ep.equals("Embedded Objects")) {
cont.addComponent(new Embedded("Embedded", new
ClassResource("tree.jpg", this)));
} else if (ep.equals("Upload")) {
Upload u = new Upload("Upload a file:", uploadBuf);
u.addListener(this);
cont.addComponent(u);
cont.addComponent(new Button("Upload", this));
cont.addComponent(upload);
} else if (ep.equals("Image Map")) {
ImageMap im = new ImageMap("demo", new
ClassResource("imagemap-demo.jpg", this), 214, 173, this);
im.setCaption("ImageMap");
ImageMap.Shape sh;
sh = im.new Circle(im.new ImageMapResource("shape:
green"), "shape: green",
im.new Point(72, 42), 30);
im.addShape(sh);
sh = im.new Rectangle(new ExternalResource
("http://google.at"),
"Google", im.new Point(24, 107), im.new Point
(100, 150));
im.addShape(sh);
im.setIcon(new ThemeResource("icon/plus.gif"));
sh = im.new Polygon(im.new ImageMapResource("shape:
blue"), "shape: blue");

```

```

        ((ImageMap.Polygon) sh).addPoint(im.new Point(129,
72));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(190,
72));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(190,
150));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(129,
150));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(129,
125));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(163,
125));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(163,
96));
        ((ImageMap.Polygon) sh).addPoint(im.new Point(129,
96));
        im.addShape(sh);
        cont.addComponent(im);
    }else if(ep.equals("Frame Window")){
        cont.addComponent(new Label("Frame demo should be
opened in a new window!"));
        frameWind = new FrameWindow("Frame demo window");
        Window left = new Window("left");
        Window right = new Window("right page 1");
        left.addComponent(new Label("left"));
        left.addComponent(new Button("Set right frame to
page 1", this, "frameDemo"));
        left.addComponent(new Button("Set right frame to
page 2", this, "frameDemo"));
        right.addComponent(new Label("right page 1"));
        frameWind.getFrameset().newFrame(left);
        frameWind.getFrameset().newFrame(right);
        frameWind.getFrameset().getFrame(0).setRelativeSize
(33);

        this.addWindow(frameWind);
        // }else if(ep.equals("new_tree_item")){
        //     //for further Tree items
        // }else{
        cont.addComponent(new Label("Select from the tree
at the left.));
        }
    }

    public void chngLang(Button.ClickEvent event){
        if(event.getButton().getCaption().equals("<-> Change
Language")) event.getButton().setCaption("<-> Sprache Aendern");
        else event.getButton().setCaption("<-> Change Language");
        event.getButton().getParent().requestRepaint();
    }

    public void processTextField(Button.ClickEvent event){
        demo.removeComponent(1,0);
        Panel pan = new Panel("Result:");
        pan.addComponent(new Label(tf.getValue().toString()));
        demo.addComponent(pan, 1,0);
    }

    public void processDateField(Button.ClickEvent event){
        demo.removeComponent(1,0);
        Panel pan = new Panel("Result:");
        if(df.getValue() == null){

```

```

        int ef = df.getErrorField();
        int[] ev = df.getErrorValues();
        String[] fields = new String[]
{"year", "month", "day", "hour", "minute", "second", "millisecond"};
        pan.addComponent(new Label("Datum: error: value '"+ev[ef]
+ "' does not exist for field '"+fields[ef]+"'."););
        }else pan.addComponent(new Label("Datum: " + df));
        demo.addComponent(pan, 1,0);
    }

    public void processForm(Button.ClickEvent event){
        demo.removeComponent(1,0);
        GridLayout gl = new GridLayout();
        Component c = event.getButton().getParent().getParent().
getParent();
        if(c instanceof GridLayout){
            gl = (GridLayout) c;
            gl.removeComponent(form);
            Panel pan = new Panel("eingegebener Text:");
            pan.addComponent(new Label(tf.getValue().toString()));
            gl.addComponent(pan, 0, 1);
        }else{
            gl.addComponent(new Label("ERROR: GridLayout not
found!")););
            gl.addComponent(new Label("x Instead '" + c.getClass().
getName() + "' found.)););
        }
        demo.addComponent(gl, 1,0);
    }

    public void showSelectSelected(){
        demo.removeComponent(1,0);
        Panel pan = new Panel("Results:");
        Object[] c;
        c = s1.getItemIds().toArray();
        for(int i=0; i<c.length; i++)
            if(s1.isSelected(c[i]))
                pan.addComponent(new Label(s1.getCaption() + " / "
+ c[i]));
        c = s2.getItemIds().toArray();
        for(int i=0; i<c.length; i++)
            if(s2.isSelected(c[i]))
                pan.addComponent(new Label(s2.getCaption() + " / "
+ c[i]));
        c = s3.getItemIds().toArray();
        for(int i=0; i<c.length; i++)
            if(s3.isSelected(c[i]))
                pan.addComponent(new Label(s3.getCaption() + " / "
+ c[i]));
        c = s4.getItemIds().toArray();
        for(int i=0; i<c.length; i++)
            if(s4.isSelected(c[i]))
                pan.addComponent(new Label(s4.getCaption() + " / "
+ c[i]));
        GridLayout gl = new GridLayout(2,4);
        gl.addComponent(new Label("Selects:"), 0, 0);
        gl.addComponent(s1, 0, 1);
        gl.addComponent(s2, 1, 1);
        gl.addComponent(s3, 0, 2);
        gl.addComponent(s4, 1, 2);
        gl.addComponent(new Button("Submit", this,
"showSelectSelected")););
    }

```

```

        gl.addComponent(pan, 0, 3);
        demo.addComponent(gl, 1,0);
    }

    public void newWindow() {
        Window newW = new Window("new Window: Google");
        newW.open(new ExternalResource("http://google.at"));
        this.addWindow(newW);
    }

    public void showTableSelected(){
        demo.removeComponent(1,0);
        OrderedLayout ol = new OrderedLayout();
        OrderedLayout cont = new OrderedLayout();
        cont.addComponent(tbl);
        cont.addComponent(new Button("Submit", this,
"showTableSelected"));
        cont.addComponent(new Panel("Selected:", ol));
        Object[] c = tbl.getItemIds().toArray();
        for(int i=0; i< c.length; i++)
            if(tbl.isSelected(c[i])){
                Object[] help = tbl.getItem(c[i]).getItemPropertyIds().
toArray();
                String output = new String();
                for(int j=0; j<help.length;j++)
                    output += tbl.getItem(c[i]).getItemProperty(help[j]).
getValue() + " / ";
                output = output.substring(0, output.length() - 3);
                ol.addComponent(new Label("selected: "+output));
            }
        demo.addComponent(cont, 1,0);
    }

    public void uploadFinished(FinishedEvent event) {
        upload.removeAllComponents();
        if(uploadBuf.getStream() == null)
            upload.addComponent(
                new Label("Upload finished, but output buffer is
null!!"));
        else{
            upload.addComponent(new Label("<b>Name:</b> " +
event.getFilename(),
                Label.CONTENT_XHTML));
            upload.addComponent(new Label("<b>Mimetype:</b> " +
event.getMIMEType(),
                Label.CONTENT_XHTML));
            upload.addComponent(new Label("<b>Size:</b> " +
event.getLength() + " bytes.",
                Label.CONTENT_XHTML));
            upload.addComponent(new Link("Download " +
uploadBuf.getFileName(),
                new StreamResource(uploadBuf, uploadBuf.getFileName
(), this)));
        }
    }

    public class Buffer implements StreamResource.StreamSource,
Upload.Receiver {
        ByteArrayOutputStream outputBuffer = null;
        String mimeType;
        String fileName;
        public InputStream getStream() {

```

```

        if (outputBuffer == null) return null;
        return new ByteArrayInputStream(outputBuffer.toByteArray());
    }
    public OutputStream receiveUpload(String filename, String
MIMEType) {
        fileName = filename;
        mimeType = MIMEType;
        outputBuffer = new ByteArrayOutputStream();
        return outputBuffer;
    }
    public String getFileName() {
        return fileName;
    }
    public String getMimeType() {
        return mimeType;
    }
}

public void imagemapClick(ImageMap.ClickEvent event) {
    demo.removeComponent(1,0);
    OrderedLayout ol = new OrderedLayout();
    demo.addComponent(new Label("Shape clicked: " +
((ImageMap.ImageMapResource)
        event.getImageMap().getFiredShape().getResource()).
getContent()), 1,0);
}

public void frameDemo(Button.ClickEvent event){
    String newPage;
    if(event.getButton().getCaption().equals("Set right frame to
page 1")) newPage = "page 1";
    else if(event.getButton().getCaption().equals("Set right frame
to page 2")) newPage = "page 2";
    else newPage = "ERROR";
    if(frameWind.getFrameset().size() >= 2)
        frameWind.getFrameset().removeFrame(frameWind.getFrameset
().getFrame(1));
    Window newW = new Window("right "+newPage);
    newW.addComponent(new Label("right "+newPage));
    frameWind.getFrameset().newFrame(newW);
}
}

```

8.2 Appendix B – Codebeispiele

Als exemplarisches Codebeispiele habe ich die, von mir neu implementierte, ImageMap gewählt. (Ursprünglich war als weiteres Codebeispiel die Table vorgesehen, die allerdings mit 25 Seiten rein für den Java-Code den Umfang dieser Arbeit sprengen würde.)

Java-Code der ImageMap-Komponente:

```

/*
*****

                                Millstone(TM)
                        Open Sourced User Interface Library for
                        Internet Development with Java

                Millstone is a registered trademark of IT Mill Ltd
                Copyright (C) 2000,2001,2002 IT Mill Ltd

*****
*

This library is free software; you can redistribute it and/or
modify it under the terms of the GNU Lesser General Public
License as published by the Free Software Foundation; either
version 2.1 of the License, or (at your option) any later version.

This library is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU
Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General Public
License along with this library; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
USA
*****
*

This class is part of the modified millstone-library by Dietmar Stoiber
*****
*

For more information, contact:

Dietmar Stoiber                                email: uni@diquadrat.net

*****
*/

package org.millstone.base.ui;

import java.lang.Math;
import java.util.Map;
import java.util.Vector;
import java.lang.reflect.Method;

import org.millstone.base.terminal.PaintTarget;

```

```

import org.millstone.base.terminal.PaintException;
import org.millstone.base.terminal.ErrorMessage;
import org.millstone.base.terminal.SystemError;

import org.millstone.base.terminal.Resource;
import org.millstone.base.terminal.Sizeable;

/** A generic button component.
 *
 * @author Dietmar Stoiber (uni@DiQuadrat.net)
 * @version 3.0.3
 * @since millstone version 3.0.3
 */
public class ImageMap extends AbstractField {

    /* Final members ***** */

    /** strings to be caught at adapter (transformer) */
    private static final String IMAGEMAP_VAR_NAME = "imagemap";

    /* Private members ***** */

    /** Vector containing the different shapes of the image-map */
    private Vector shapes;

    /** Name of the ImageMap */
    private String mapName;

    /** Default alternative text for the Image (if set to null or blank)
 */
    private String defaultImageAlt = "Image-Map";

    /** Alternative text for the Image */
    private String imageAlt = defaultImageAlt;

    /** Source of the embedded object */
    private Resource image = null;

    /** Dimensions of the object. */
    private int width = -1;
    private int height = -1;

    /** contained the number of the last clicked Shape with an
    ImageMapResource. */
    private int lastClickedShape = -1;

    /** Creates a new ImageMap.
 *
 * @param mapName The name used to reference the imagemap in HTML -
    randomly generated if null.
 * @param image The image-resource for the image of the imagemap (eg.
    ClassResource, ExternalResource)
 * @param width The width of the image.
 * @param height The height of the image.
 */
    public ImageMap(String mapName, Resource image, int width, int height)
    {
        if (mapName == null)
            mapName = "imagemap_" + Math.random();
        this.mapName = mapName;
        this.image = image;
        this.width = width;
    }

```

```

        this.height = height;
        shapes = new Vector();
    }

    /** Creates a new ImageMap with click listener.
     *
     * @param mapName The name used to reference the imagemap in HTML -
     randomly generated if null.
     * @param image The image-resource for the image of the imagemap
     (eg. ClassResource, ExternalResource)
     * @param width The width of the image.
     * @param height The height of the image.
     * @param listener The ClickListener listening to a click-event.
     */
    public ImageMap(String mapName, Resource image, int width, int height,
ClickListener listener) {
        this(mapName, image, width, height);
        addListener(listener);
    }

    /** Creates a new ImageMap with a method listening button clicks.
     * The method must have either no parameters, or only one parameter of
     * ImageMap.ClickEvent type.
     *
     * @param mapName The name used to reference the imagemap in HTML -
     randomly generated if null.
     * @param image The image-resource for the image of the imagemap
     (eg. ClassResource, ExternalResource)
     * @param width The width of the image.
     * @param height The height of the image.
     * @param target Object having the method for listening button clicks.
     * @param methodName The name of the method in target object, that
     receives button click events.
     * (The method must have either no parameters, or only one parameter of
     ImageMap.ClickEvent type.)
     */
    public ImageMap(String mapName, Resource image, int width, int height,
Object target, String methodName) {
        this(mapName, image, width, height);
        addListener(ClickEvent.class, target, methodName);
    }

    /** Get component UIDL tag.
     * @return Component UIDL tag as string.
     */
    public String getTag() {
        return "imagemap";
    }

    /** Add an Shape-object to the ImageMap
     *
     * @param shape The Shape to add.
     * @throws IllegalArgumentException If the Shape has no alternative
     text.
     */
    public void addShape(Shape shape) throws IllegalArgumentException{
        if(shape != null){
            if(shape.alternative == null) throw new
IllegalArgumentException("No alternative text for ImageMap-Shape set!");
            shapes.add(shape);
        }
    }

```



```

    }

    /** Returns a Shape-array of all the ImageMap's Shapes
     *
     * @return Shape-array or null if no Shapes were add.
     */
    public Shape[] getShapes() {
        if(shapes.size() == 0) return null;
        return (Shape[]) shapes.toArray();
    }

    /** Returns the alternativ text of the image.
     * (Default is "Image-Map")
     *
     * @return The alternative text of the image.
     */
    public String getImageAlternativeText() {
        return imageAlt;
    }

    /** Sets the alternativ text of the image.
     * (Default is "Image-Map" used if set to null or blank)
     *
     * @param imageAlt The alternative text of the image.
     */
    public void setImageAlternativeText(String imageAlt) {
        if(imageAlt == null || imageAlt.trim() == "") this.imageAlt =
this.defaultImageAlt;
        this.imageAlt = imageAlt;
    }

    /** Returns the last clickes Shape with ImageMapResource.
     *
     * @return The last clicked Shape or null if no Shape was clicked
before.
     */
    public Shape getFiredShape() {
        if(lastClickedShape < 0) return null;
        return (Shape) shapes.elementAt(lastClickedShape);
    }

    /** Paint the content of this component.
     * @param event PaintEvent.
     * @throws IOException Passed from the UIDLStream.
     * @throws PaintException The paint operation failed.
     */
    public void paintContent(PaintTarget target) throws PaintException {
        super.paintContent(target);
        // image
        target.addAttribute("mapname", mapName);
        target.addAttribute("image", image);
        target.addAttribute("width", width + Sizeable.UNITS_PIXELS);
        target.addAttribute("height", height + Sizeable.UNITS_PIXELS);
        target.addAttribute("altText", imageAlt);
        // Map
        target.startTag("map");
        target.addAttribute("mapname", mapName);
        for (int i = 0; i < shapes.size(); i++) {
            target.startTag("shape");
            // type and other arguments
            Shape curSh = (Shape) shapes.elementAt(i);
            if(curSh.target instanceof ImageMapResource)

```

```

target.addAttribute("target", "this_ImageMap");
    else target.addAttribute("target", curSh.target);
    target.addAttribute("altText", curSh.alternative);
    if (curSh.getClass().isInstance(new Circle())) {
        target.addAttribute("shapeType", "circle");
        target.addAttribute("center_x", ((Circle) curSh).
center.x);
        target.addAttribute("center_y", ((Circle) curSh).
center.y);
        target.addAttribute("radius", ((Circle) curSh).
radius);
    } else if (curSh.getClass().isInstance(new Rectangle())) {
        target.addAttribute("shapeType", "rect");
        target.addAttribute("topLeft_x", ((Rectangle)
curSh).topLeft.x);
        target.addAttribute("topLeft_y", ((Rectangle)
curSh).topLeft.y);
        target.addAttribute("bottomRight_x", ((Rectangle)
curSh).bottomRight.x);
        target.addAttribute("bottomRight_y", ((Rectangle)
curSh).bottomRight.y);
    } else if (curSh.getClass().isInstance(new Polygon())) {
        target.addAttribute("shapeType", "poly");
        Polygon.PolygonPoint pp = ((Polygon) curSh).head;
        while (pp != null) {
            target.startTag("polygonpoint");
            target.addAttribute("poly_x", pp.x);
            target.addAttribute("poly_y", pp.y);
            target.endTag("polygonpoint");
            pp = pp.next;
        }
    } else {
        throw new PaintException("No handler for ImageMap-
Shape \"" + shapes.elementAt(i).getClass().getName() + "\" was found.");
    }
    target.endTag("shape");
}
target.endTag("map");
// Variables
target.addVariable(this, "imageMapClick", -1);

}

/** Invoked when the value of a variable has changed. Button
 * listeners are notified if the button is clicked.
 * @param event Variable change event.
 */
public void changeVariables(Object source, Map variables) {
    if (variables.containsKey("imageMapClick")) {
        try {
            Integer newValue = (Integer) variables.get
("imageMapClick");
            if (newValue != null) {
                lastClickedShape = newValue.intValue() - 1;
                fireClick();
            }
        } catch (Throwable e) {
            if (e instanceof ErrorMessage)
                setComponentError((ErrorMessage) e);
            else
                setComponentError(new SystemError(e));
        }
    }
}

```

```

    }
}

/** The type of the button as a property.
 * @see org.millstone.base.data.Property#getType()
 */
public Class getType() {
    return Object.class;
}

/* Nested Classes *****
*/

/** Shape defines a clickable area of an ImageMap.
 *
 * @author Dietmar Stoiber
 */
public class Shape{
    /** Holds the target Resource of the Shape */
    Resource target;
    /** Holds the alternative text of the Shape */
    String alternative;

    /** Private constructor only for instanceof etc.
     */
    private Shape() {}

    /** Private constructor for a shape - only used from the
subclasses.
     */
    * @param target The target of the Shape.
    * @param alternative The alternative text of the Shape.
    */
    private Shape(Resource target, String alternative){
        this.target = target;
        this.alternative = alternative;
    }

    /** Returns the target Resource of the Shape.
     */
    * @return Returns the target Resource.
    */
    public Resource getResource() {
        return target;
    }
}

/** A single point in the ImageMap.
 *
 * @author Dietmar Stoiber
 *
 * Contains x and y coordinates of the point.
 */
public class Point{
    /** Holds the x coordinate */
    int x;
    /** Holds the y coordinate */
    int y;

    /** Generates a new point with x and y as it's coordinates.
     */
    * @param x The x coordinate.

```

```

        * @param y The y coordinate.
        */
        public Point(int x, int y){
            this.x = x;
            this.y = y;
        }
    }

    /** The Circle shape.
    *
    * @author Dietmar Stoiber
    *
    * Contains a point (the center of the circle) and an integer (the
    radius).
    */
    public class Circle extends Shape{
        /** Holds the center point of the circle. */
        Point center;
        /** Holds the radius of the circle. */
        int radius;

        /** Private constructor only for instanceof etc.
        */
        private Circle(){}

        /** Generates a new Circle shape with the given parameters.
        *
        * @param target The target-resource of the shape.
        * @param alternative The alternative text of the shape.
        * @param center The center of the Circle.
        * @param radius The radius of the Circle.
        */
        public Circle(Resource target, String alternative, Point
center, int radius){
            super(target, alternative);
            this.center = center;
            this.radius = radius;
        }
    }

    /** The Rectangle shape.
    *
    * @author Dietmar Stoiber
    *
    * Contains two points - the top, left and the bottom, right one, to
    define the rectangle.
    */
    public class Rectangle extends Shape{
        /** Holds the coordinates of the upper left corner. */
        Point topLeft;
        /** the coordinates of the lower right corner. */
        Point bottomRight;

        /** Private constructor only for instanceof etc.
        */
        private Rectangle(){}

        /** Generates a new Rectangle shape with the given parameters.
        *
        * @param target The target-resource of the shape.
        * @param alternative The alternative text of the shape.
        * @param topLeft The Point with the coordinates of the upper

```

```

left corner.
    * @param bottomRight The Point with coordinates of the lower
right corner.
    */
    public Rectangle(Resource target, String alternative, Point
topLeft, Point bottomRight){
        super(target, alternative);
        this.topLeft = topLeft;
        this.bottomRight = bottomRight;
    }

}

/** The Polygon shape.
 *
 * @author Dietmar Stoiber
 *
 * Contains a list of Points, that defines the polygon.
 */
public class Polygon extends Shape{
    /** Holds the begin of the PolygonPoint list. */
    PolygonPoint head;
    /** Holds the end of the PolygonPoint list (for fast addPoint).
 */
    PolygonPoint tail;

    /** Private constructor only for instanceof etc.
    */
    private Polygon() {}

    /** Creates a new Polygon with the given parameters and a empty
PolygonPoint-list.
    *
    * @param target The target-resource of the shape.
    * @param alternative The alternative text of the shape.
    */
    public Polygon(Resource target, String alternative){
        super(target, alternative);
    }

    /** Creates a new Polygon with the given parameters.
    *
    * @param target The target-resource of the shape.
    * @param alternative The alternative text of the shape.
    * @param head The first PoligonPoint of a PolygonPoint-list.
    */
    public Polygon(Resource target, String alternative,
PolygonPoint head){
        super(target, alternative);
        this.head = head;
        tail = head;
        while(tail.next != null) tail = tail.next;
    }

    /** Add a Point to the end of the PolygonPoint-list.
    *
    * @param p The Point to add.
    */
    public void addPoint(Point p){
        if(head == null){
            head = new PolygonPoint(p);
            tail = head;

```

```

        }else{
            tail.next = new PolygonPoint(p);
            tail = tail.next;
        }
    }

    /** A single PolygonPoint holding the coordinates of itself and
    the next PolygonPoint.
    *
    * @author Dietmar Stoiber
    */
    public class PolygonPoint extends Point{
        /** The next PolygonPoint. */
        public PolygonPoint next;

        /** Creates a new PoligonPoint with the given
coordinates.
        *
        * @param x The x coordinate.
        * @param y The y coordinate.
        */
        public PolygonPoint(int x, int y){
            super(x, y);
            next = null;
        }

        /** Creates a new PoligonPoint with the given Point.
        *
        * @param p The Point, used to create a new PolygonPoint.
        */
        public PolygonPoint(Point p){
            super(p.x, p.y);
            next = null;
        }
    }
}

/** ImageMapResource defines that the target of the area is the
current Application.
*
* @author Dietmar Stoiber
*/
public class ImageMapResource implements Resource {
    /** Additional Object to be able to recognize the clicked Shape
with an id for example */
    private Object content;

    /** Creates a new ImageMapResource without a additional Object
(set to null).
    *
    * @param content The additional content of the
ImageMapResource.
    */
    public ImageMapResource(){
        this.content = null;
    }

    /** Creates a new ImageMapResource with the additional Object.
    *
    * @param content The additional content of the
ImageMapResource.
    */

```

```

    public ImageMapResource(Object content){
        this.content = content;
    }

    /** Returns the additional Object.
     *
     * @return The additional Object
     */
    public Object getContent(){
        return content;
    }

    /** Sets the additional Object to the given parameter.
     *
     * @param content The new addirtional Object.
     */
    public void setContent(Object content){
        this.content = content;
    }

    /** Returns the MIME-type of the Resource.
     *
     * @return The MIME-type of the Resource. (not needed in case
of this Resource).
     */
    public String getMimeType() {
        return "application/millstone-ImageMap";
    }
}

/* Click event ***** */

private static final Method IMAGEMAP_CLICK_METHOD;
static {
    try {
        IMAGEMAP_CLICK_METHOD =
            ClickListener.class.getDeclaredMethod(
                "imagemapClick",
                new Class[] { ClickEvent.class });
    } catch (java.lang.NoSuchMethodException e) {
        // This should never happen
        throw new java.lang.RuntimeException();
    }
}

/** Click event. This event is thrown, when the button is clicked.
 * @author Dietmar Stoiber
 * @version 3.0.3
 * @since millstone version 3.0.3
 */
public class ClickEvent extends Component.Event {

    /** New instance of text change event
     * @param source Source of the event.
     */
    public ClickEvent(Component source) {
        super(source);
    }

    /** ImageMap where the event occurred
     * @return Source of the event.
     */
}

```

```

        public ImageMap getImageMap() {
            return (ImageMap) getSource();
        }
    }

    /** ImageMap click listener
     * @author Dietmar Stoiber
     * @version 3.0.3
     * @since millstone version 3.0.3
     */
    public interface ClickListener {

        /** ImageMap has been pressed.
         * @param event ImageMap click event.
         */
        public void imagemapClick(ClickEvent event);
    }

    /** Add ImageMap click listener
     * @param listener Listener to be added.
     */
    public void addListener(ClickListener listener) {
        addListener(ClickEvent.class, listener, IMAGE_MAP_CLICK_METHOD);
    }

    /** Remove ImageMap click listener
     * @param listener Listener to be removed.
     */
    public void removeListener(ClickListener listener) {
        removeListener(ClickEvent.class, listener, IMAGE_MAP_CLICK_METHOD);
    }

    /** Emit options change event. */
    protected void fireClick() {
        fireEvent(new ImageMap.ClickEvent(this));
    }
}

```

XSLT-Datei zur Transformation der XML-Ausgabe der ImageMap-Komponente in HTML:

```

<?xml version="1.0" encoding="ISO-8859-1"?>

<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

<xsl:template match="imagemap" mode="core">
    <MAP>
        <xsl:attribute name="NAME"><xsl:value-of
select="@mapname"/></xsl:attribute>
        <xsl:for-each select="map/shape">
            <xsl:call-template name="map-shape">
                <xsl:with-param name="imageMapId"><xsl:value-of
select="../../integer[@name='imageMapClick']/@id"/></xsl:with-param>
                <xsl:with-param name="shapeNr"><xsl:value-of

```



```

select="position()"/></xsl:with-param>
    </xsl:call-template>
  </xsl:for-each>
</MAP>
<xsl:if test="@icon"><IMG SRC="{@icon}"/></xsl:if>
<xsl:if test="@caption"><nobr CLASS="caption"><xsl:value-of
select="@caption"/></nobr><br /></xsl:if>
  <IMG BORDER="0">
    <xsl:attribute name="SRC"><xsl:value-of
select="@image"/></xsl:attribute>
    <xsl:attribute name="WIDTH"><xsl:value-of
select="@width"/></xsl:attribute>
    <xsl:attribute name="HEIGHT"><xsl:value-of
select="@height"/></xsl:attribute>
    <xsl:attribute name="USEMAP">#<xsl:value-of
select="@mapname"/></xsl:attribute>
    <xsl:attribute name="ALT"><xsl:value-of
select="@altText"/></xsl:attribute>
  </IMG>
  <!-- Set focus to field -->
  <xsl:if test="@focus='true' and $dhtml">
    <SCRIPT>document.getElementById('<xsl:value-of
select="."/boolean/@id"/>').focus()</SCRIPT>
  </xsl:if>
</xsl:template>

<xsl:template name="map-shape">
  <xsl:param name="imageMapId" />
  <xsl:param name="shapeNr" />
  <AREA>
    <xsl:attribute name="SHAPE"><xsl:value-of
select="@shapeType"/></xsl:attribute>
    <xsl:choose>
      <xsl:when test="@target='this_ImageMap'">
        <xsl:attribute name="HREF">?<xsl:value-of
select="$imageMapId"/>=<xsl:value-of select="$shapeNr"/></xsl:attribute>
      </xsl:when>
      <xsl:otherwise>
        <xsl:attribute name="HREF"><xsl:value-of
select="@target"/></xsl:attribute>
      </xsl:otherwise>
    </xsl:choose>
    <xsl:attribute name="ALT"><xsl:value-of
select="@altText"/></xsl:attribute>
    <xsl:choose>

      <!-- Shape: Circle -->
      <xsl:when test="@shapeType='circle'">
        <xsl:attribute name="COORDS"><xsl:value-of
select="@center_x"/>,<xsl:value-of select="@center_y"/>,<xsl:value-of
select="@radius"/></xsl:attribute>
      </xsl:when>

      <!-- Shape: Rectangle -->
      <xsl:when test="@shapeType='rect'">
        <xsl:attribute name="COORDS"><xsl:value-of
select="@topLeft_x"/>,<xsl:value-of select="@topLeft_y"/>,<xsl:value-of
select="@bottomRight_x"/>,<xsl:value-of
select="@bottomRight_y"/></xsl:attribute>
      </xsl:when>

      <!-- Shape: Polygon -->
      <xsl:when test="@shapeType='poly'">

```

```

        <xsl:variable name="count"><xsl:value-of select="count
(polygonpoint)"/></xsl:variable>
        <xsl:attribute name="COORDS"><xsl:for-each
select="polygonpoint"><xsl:variable name="curPos"><xsl:value-of
select="position()"/></xsl:variable><xsl:value-of
select="@poly_x"/>,<xsl:value-of select="@poly_y"/><xsl:if
test="$curPos!=$count">,</xsl:if></xsl:for-each></xsl:attribute>
        </xsl:when>

    </xsl:choose>
</AREA>
</xsl:template>

<xsl:template match="imagemap">
    <!-- Core ImageMap -->
    <xsl:apply-templates select="." mode="core"/>
    <!-- descriptions and errors -->
    <xsl:choose>
        <xsl:when test="$dhtml">
            <xsl:for-each select="./error"><xsl:apply-templates select="."
mode="dhtml"/></xsl:for-each>
            <xsl:for-each select="./description"><xsl:apply-templates
select="." mode="dhtml"/></xsl:for-each>
            </xsl:when>
            <xsl:otherwise>
                <xsl:if test="./error"><BR /><xsl:apply-templates
select="./error" mode="inline"/></xsl:if>
                <xsl:if test="./description"><BR /><xsl:apply-templates
select="./description" mode="inline"/></xsl:if>
            </xsl:otherwise>
        </xsl:choose>
    </xsl:template>
</xsl:stylesheet>

```

9 Quellverzeichnis

9.1 Literaturverzeichnis

9.1.1 Basisliteratur

- [Tie01] Ernst Tiemeyer, Karl Wilbers; e-Learning - Neue Möglichkeiten für die berufliche Bildung; 2001
http://www.anuba-online.de/extdoc/mv_info/gemeinsam/Anuba_MVI_E-Learning_tiemeyer_wilbers2001.pdf (14.11.2003)
- [Gol01] E-Learning und Fernunterricht sollen weiter zusammengehen
<http://www.golem.de/0211/22861.html> (14.11.2003)
- [Bau] Peter Baumgartner, Kornelia Häferle, Hartmut Häferle; Was ist E-Learning - Eine Definition
<http://www.hsw.fhso.ch/e-learning/definition.htm> (14.11.2003)
- [OFT] e-Learning - Neue multimediale Lernformen
http://onforte.de/freie_seite.php3?hauptkategorie=elearning (14.11.2003)
- [Ast] Hermann Astleitner; Prinzipien guten Unterrichts
http://www.qis.at/material/astleitner_unterrichtsqualität.pdf (14.11.2003)
- [WeLearn] WeLearn
<http://www.fim.uni-linz.ac.at/research/WeLearn/index.htm> (14.11.2003)
- [Div02] Roman Divotkey, Jörg R. Mühlbacher, Susanne Reisinger, Doris Remplbauer; The WeLearn Distance Teaching Framework; 2002
http://www.fim.uni-linz.ac.at/research/DistanceEducation/publications/The_WeLearn_Distance_Teaching_Framework.pdf (14.11.2003)
- [Son] Michael Sonntag; Legal Engineering - Introducing legal thoughts to the design of an online learning platform
http://www.fim.uni-linz.ac.at/Publications/Sonntag/Legal_Engineering.pdf (14.11.2003)

- [WeLearnTA] Technische Aspekte von WeLearn
<http://content.bg-bab.ac.at/download/WeLearn%20Offline%20Konverter.pdf> (14.11.2003)
- [IMS] IMS Global Learning Consortium
<http://www.imsglobal.org/aboutims.cfm> (14.11.2003)
- [Gam96] Erich Gamma, Richard Helm, Ralph. Johnson, John M. Vlissides;
Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software;
Addison-Wesley 1996, ISBN 3893199500
- [Lan02] James Landay; User Interface Design, Prototyping, & Evaluation: 2002
<http://bmrc.berkeley.edu/courseware/cs160/fall00/Lectures/model-view-controller/model-view-controller.pdf> (03.12.2003)
- [Ben02] Dieter Bender; Model View Controller in Servlets; 2003
<http://www.common-d.de/pdf/JaMVC.pdf> (03.12.2003)
- [Bar99] Christobal Baray; The model-view-controller (MVC) design pattern; 1999
<http://www.cs.indiana.edu/~cbaray/projects/mvc.html> (03.12.2003)
- [eNode] Model-View-Controller Pattern: 2002
<http://www.enode.com/x/markup/tutorial/mvc.html> (03.12.2003)
- [Kna] Christian Knauer; Das Model/View/Controller Paradigma
<http://www.inf.fu-berlin.de/lehre/SS03/aws/mvc.pdf> (03.12.2003)
- [Bar02] Klaus G. Barthelmann; Objektorientierte Programmierung in Java; 2002
<http://www.informatik.uni-mainz.de/~barthel/OOPJ/Lektionen1/11/Lektion11.html> (03.12.2003)
- [Sun02] J2EE Architecture Approaches; 2002
http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/app-arch/app-arch2.html (03.12.2003)
- [Dea00] John Deacon; Model-View-Controller (MVC) Architecture; 2000
<http://www.jdl.co.uk/briefings/MVC.pdf> (03.12.2003)
- [NetChem] Model View Controller (MVC)
http://www.netchemistry.com/pdf/white_papers/NCModelViewController.p

df (03.12.2003)

[Oracle] Design

http://www.oracle.com/technology/sample_code/tutorials/fbs/over/design.htm
(03.12.2003)

[ScreenReader] Behindertengerechtes Internet

<http://www.tecchannel.de/internet/764/1.html> ff (03.12.2003)

[UTA] MVC Architecture

<http://www.uta.fi/~jl/pguibook/mvc.html> ff (03.12.2003)

[Wiki/4] Wikipedia: Model-view-controller

<http://en.wikipedia.org/wiki/Model-view-controller> (03.12.2004)

9.1.2 Referenzen

[Apache] Apache Software Foundation

<http://www.apache.org/> (03.12.2003)

[Cocoon] The Apache Cocoon Project

<http://cocoon.apache.org/> (03.12.2003)

[FIM] Institut für Informationsverarbeitung und Mikroprozessortechnik

<http://www.fim.uni-linz.ac.at/> (14.11.2003)

[IMS] IMS Global Learning Consortium

<http://www.imsglobal.org/> (14.11.2003)

[ITMill] IT Mill Ltd

<http://www.itmill.com/> (03.12.2003)

[JKU] Johannes Kepler Universität Linz

<http://www.jku.at/> (14.11.2003)

[Mensa] Die häufigsten Fragen zum IQ

<http://at.mensa.org/iq.htm> (14.11.2003)

[Millstone] Millstone.org - Free Open Source Web UI Library and Reusable Components for J2EE and Java

<http://www.millstone.org/> ff (03.12.2003)

- [MSapi] Millstone API-Dokumentation
<http://www.millstone.org/documentation/apidocs/index.html> (03.12.2003)
- [MSsrc] Millstone Source-Dateien
<http://www.millstone.org/downloads/index.html> (03.12.2003)
- [Mue02a] Mühlbacher, J.R., Mühlbacher, S.C., Reisinger, S.: Learning Arrangements and Settings for Distance Teaching/Coaching/Learning: Best practice report. In: Hofer Christian, Chroust Gerhard (Ed.): IDIMT-2002. 10th Interdisciplinary Information Management Talks. Linz: Universitätsverlag Rudolf Trauner 2002
- [Mue02b] Mühlbacher, S.C.: Distance Learning, Ablauf und Evaluierung einer Lehrveranstaltung aus Informatik. Diplomarbeit, Institut für Psychologie, University of Vienna, Austria, 2002
- [OIO] Vergleich von Servlets, JSP, XSP, MVC und HMVC
<http://www.oio.de/public/jsp-xsp-mvc-vergleich.htm> (03.12.2003)
- [Qis] Hermann Astleitner, Prinzipien guten Unterrichts
http://www.qis.at/material/astleitner_unterrichtsqualität.pdf (14.11.2003)
- [Rec02/1, S. 525 ff] Rechenberg, Pomberger: Informatik-Handbuch, Hanser 2002, ISBN 3-446-21842-4, Seite 525 ff
- [Rec02/2, S. 529 ff] Rechenberg, Pomberger: Informatik-Handbuch, Hanser 2002, ISBN 3-446-21842-4, Seite 529 ff
- [Rec02/3, S. 802 ff] Rechenberg, Pomberger: Informatik-Handbuch, Hanser 2002, ISBN 3-446-21842-4, Seite 802 ff
- [Ree] Trygve M. H. Reenskaug; The original MVC
<http://heim.ifi.uio.no/~trygver/themes/mvc/mvc-index.html> (03.12.2003)
- [Rei02] Susanne Reisinger, Mobile intelligente Agenten als Wegweiser im Distance Teaching/Coaching/Learning, 2002
http://www.fim.unilinz.ac.at/Diplomarbeiten/dissertation_reisinger/Inhalt/dissertation_reisinger.zip (29.10.2003)
- [ScreenReader] Behindertengerechtes Internet

- <http://www.tecchannel.de/internet/764/1.html> ff (03.12.2003)
- [Struts] The Apache Struts Web Application Framework
<http://struts.apache.org/> (03.12.2003)
- [Swing] Java Foundation Classes
<http://java.sun.com/products/jfc/index.jsp>,
<http://java.sun.com/j2se/1.4.2/docs/api/javawx/swing/package-summary.html>
(03.12.2003)
- [SWT] SWT: The Standard Widget Toolkit
<http://www.eclipse.org/articles/Article-SWT-Design-1/SWT-Design-1.html>
(03.12.2003)
- [Wiki/2] Wikipedia: Learning by doing
http://de.wikipedia.org/wiki/Learning_by_doing (14.11.2003)
- [Wiki/3] Wikipedia: Spaghetti code
http://en.wikipedia.org/wiki/Spaghetti_code (03.12.2003)

9.1.3 Zitate

- [DfES] Department for Education and Skills, e-learning strategy
<http://www.dfes.gov.uk/elearningstrategy/elearning.stm> (14.11.2003)
- [DELG02] Distributed and Electronic Learning Group for the Learning and Skills Council (2002),
http://www.deni.gov.uk/inspection_services/publications/IQRS%20Further%20Ed%20Supp.pdf (14.11.2003)
- [Cis01] E-Learning Trends: iQ Leaders 2001
http://business.cisco.com/prod/tree.taf?asset_id=74398&public_view=true&kbns=1.html (14.11.2003)
- [Wiki/1] Wikipedia: E-learning
<http://en.wikipedia.org/wiki/E-learning> (14.11.2003)

9.2 Abbildungsverzeichnis

(Hier nicht aufgeführte Abbildungen wurden von mir selbst erstellt.)

[Abb_Lernmodel] Aus [Diss_Loidl] Kapitel 6.2.3 Lernarrangements - Settings

[Abb_MilstJEEA] <http://millstone.org/documentation/architecture/overview.html>

[Abb_MilstAppstr] <http://millstone.org/documentation/architecture/overview.html>

[Abb_MVCBeisp] <http://www.informatik.uni-mainz.de/~barthel/OOPJ/Lektionen1/11/Lektion11.html>

Eidesstattliche Erklärung

Ich erkläre an Eides statt, dass ich die vorliegende Diplom- bzw. Magisterarbeit selbstständig und ohne fremde Hilfe verfasst, andere als die angegebenen Quellen und Hilfsmittel nicht benutzt bzw. die wörtlich oder sinngemäß entnommenen Stellen als solche kenntlich gemacht habe.

Marchtrenk, 16 Februar 2005

Lebenslauf



Persönliche Information:	Name	Dietmar Stoiber
	Wohnhaft:	Robert-Stolz-Strasse 8 A-4614 Marchtrenk
	Geboren:	26.11.1978, Wels
	Staatsbürgerschaft:	Österreich
	E-Mail:	Dietmar.Stoiber@students.jku.at
Ausbildung:	1985 - 1989:	Besuch der Volksschule VS1-Marchtrenk
	1989 - 1998:	Besuch des Welser BRG Anton Bruckner Strasse (Schulversuchszweig mit naturwissenschaftlichem Schwerpunkt)
	1998:	Ablegung der Reifeprüfung
	1998 - 1999:	Absolvieren der Wehrdienstes
	Okt. 1999 - [März 2004]:	Informatikstudium an der Universität Linz
Berufstätigkeit:	Juli/August 2000:	Ferialjob: Sport Eybl
	SS01, SS02, WS02, SS03:	Tutor für 'Embedded Systems' und 'Telekooperation' am Institut für Praktische Informatik, Gruppe Software
	März 2003 - Oktober 2003:	Praktischer Teil der Diplomarbeit (Mitarbeit am WeLearn-Projekt) am Institut für Informationsverarbeitung und Mikroprozessortechnik auf Werkvertragsbasis
	März 2003 – Dezember 2003	Mitarbeit am Mobilearn-Projekts am Institut für Praktische Informatik, Gruppe Software auf Werkvertragsbasis
	Jänner 2004 -	Softwareentwicklung bei forms2web communications GmbH
Sonstiges:	Mai 2003 – Jänner 2004	Mitglied der Berufungskommission für die Vorziehprofessur 'Computational Perception'
Fähigkeiten:	Java(J2EE, Servlets, JSP, Swing, AWT, SWT, ...), PHP, Perl, JavaScript, C++, C# XML, XSL, (X)HTML, CSS, SQL (MySQL, Oracle, Torque), WAI, UML, Use Cases, LaTeX Netzwerkadministration, Netzwerkplanung, IT-Security, Mensch-Maschine-Interfaces, Testen, Server (Apache, MySQL, Tomcat, Struts, ..), Eclipse, OpenOffice Windows (NT-Reihe), GNU/Linux Deutsch, Englisch	