

Fachhochschule Wiesbaden  
Fachbereich Design Informatik Medien  
Studiengang Allgemeine Informatik

Diplomarbeit  
zur Erlangung des akademischen Grades  
Diplom-Informatiker (FH)

# **Entwicklung eines benutzerfreundlichen Editors zur Erfassung von standardisierten Personenprofilen mit XML-Generierung**

Vorgelegt von:  
am:

Patric Eid  
28.01.2008

Referent:  
Korreferent:

Prof. Dr. Grit Behrens  
Dipl.-Inf. Martin Klossek



## Erklärungen

Erklärung gemäß Prüfungsordnung – Teil A - §6.4.2

Ich versichere, dass ich die Diplomarbeit selbstständig verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe.

Ort, Datum

Unterschrift Diplomand

Hiermit erkläre ich mein Einverständnis mit den im Folgenden aufgeführten Verbreitungsformen dieser Diplomarbeit:

Verbreitungsform	ja	nein
Einstellung der Arbeit in die Bibliothek der FHW	X	
Veröffentlichung des Titels der Arbeit im Internet	X	
Veröffentlichung der Arbeit im Internet		X

Ort, Datum

Unterschrift Diplomand



Erstens: Alles ist, wie es war.  
Zweitens: Nichts ist wie zuvor.  
*Thomas Lehr [LEH05]*



---

## Inhaltsverzeichnis

<b>1 Einleitung</b>	<b>1</b>
1.1 Gliederung	1
1.2 Problemstellung	2
1.3 Ziele der Diplomarbeit	2
1.4 Arbeitsumgebung	3
<b>2 Grundlagen</b>	<b>5</b>
2.1 Extensible Markup Language (XML)	5
2.1.1 Der Aufbau	5
2.1.2 Syntaxregeln von XML	9
2.1.3 Namensräume	9
2.1.4 XML-Schema	10
2.1.5 XSL	13
2.1.6 Fazit	15
2.2 .NET Framework	16
2.2.1 Einstieg in C#	17
2.2.2 C# Datentypen	18
2.2.3 Garbage Collection	19
2.2.4 Begriffe aus dem .NET-Umfeld	19
2.2.4.1 Namensräume und Klassen	19
2.2.4.2 Objekte	22
2.2.4.3 Exceptions	22
2.2.4.4 Events	25
2.2.5 C# und XML	25
<b>3 Konzept</b>	<b>31</b>
3.1 Analyse	31
3.1.1 Evaluierung über Bestandteile eines Lebenslaufes	32
3.1.1.1 Persönliche Angaben	32
3.1.1.2 Kontaktdaten	33
3.1.1.3 Schulbildung	33
3.1.1.4 Wehr- oder Zivildienst / Freiwilliges soziales Jahr	34
3.1.1.5 Studium	34
3.1.1.6 Praktische Erfahrungen	35
3.1.1.7 Sprachkenntnisse	35
3.1.1.8 IT-Kenntnisse	35
3.1.1.9 Weiterbildung	36
3.1.1.10 Hobbies und Interessen	36
3.1.1.11 Gesellschaftliches Engagement / Ehrenamtliche Aktivitäten	36
3.1.1.12 Publikationen	36

3.1.2 Allgemeines Gleichbehandlungsgesetz	37
3.1.3 Untersuchung von möglichen Standards zur Datenhaltung	38
3.1.3.1 HR-XML	38
3.1.3.2 HR-BA-XML	43
3.1.3.3 GSCV	47
3.1.3.4 iProfile	51
3.1.3.5 Fazit	52
3.1.4 Evaluierung von vergleichbaren Softwareprogrammen	54
3.1.4.1 Resume Builder	55
3.1.4.2 Easy Resume Creator	59
3.2 Lösungsansatz	61
3.3 Design der Benutzerschnittstelle	64
3.4 Architektur des Programms	67
3.4.1 Anwendungsfallmodell	67
3.4.1.1 Akteure	68
3.4.1.2 Anwendungsfälle	70
3.4.1.3 Anwendungsfalldiagramm	75
3.4.1.4 Aktivitätsdiagramm	77
3.4.2 Klassendiagramm	79
3.4.2.1 Klassendiagramm Model-View-Controller	79
3.4.2.2 Klassendiagramm Model-Konzept	81
3.4.3 Pluginkonzept	83
3.4.4 Sequenzdiagramm	84
<b>4 Implementierung</b>	<b>87</b>
4.1 Projekte und Unterprojekte	87
4.1.1 Projekt Utils	88
4.1.2 Projekt Plugin	89
4.1.3 Projekt Data	90
4.2 Aufbau der Projektumgebung	91
4.3 Aufbau der Klassen	91
4.3.1 Klasse ApplicationStarter	92
4.3.2 Klasse MenuAction	93
4.3.3 Klasse CountryHandler	94
4.3.4 Klasse ProfileManager	96
4.3.5 Klasse ObjectSerializer	96
4.4 Bedienung des Programms	99
4.4.1 GSCV (HR-XML)-Export	105
4.4.2 HTML-Export	107
4.4.3 MS-Word-Export	108
4.5 Zukünftige Erweiterungen	109
4.5.1 Auswahl der Formulare	109

---

4.5.2	Verbesserung der Export-Funktionen	109
4.5.3	Import-Funktion für HR-XML-Daten	109
4.5.4	Hilfetexte für die Formulare erstellen	110
4.5.5	Ausgefüllten Lebenslauf als Beispiel anbieten	110
4.5.6	Löschen des Bildes aus dem Profil	110
4.5.7	Export von hResume-Microformate	110
4.5.8	Import von hResume-Microformate	111
<b>5</b>	<b>Evaluierung</b>	<b>113</b>
<hr/>		
5.1	Erstellen eines Testbogens	113
5.2	Testdurchführung	114
5.3	Auswertung der Testergebnisse	114
5.3.1	Persönliche Angaben	114
5.3.2	Optische Wahrnehmung	116
5.3.3	Systemleistung	116
5.3.4	Dateneingabe	117
5.3.5	Terminologie	117
5.3.6	Fazit	118
5.4	Zusammenfassung	120
<b>6</b>	<b>Fazit</b>	<b>121</b>
<hr/>		
6.1	Technologie	121
6.2	Programmierung	122
6.3	Ergebnisse	123
<b>7</b>	<b>Anhang</b>	<b>125</b>
<hr/>		
A	Glossar	126
B	Abbildungsverzeichnis	128
C	Tabellenverzeichnis	129
D	Quellcodeverzeichnis	130
E	Literaturverzeichnis	131
F	Spezifikationen	133
G	Firmen und Organisationen	134
H	Umfragebogen zur Evaluierung	135



# 1 EINLEITUNG

## 1.1 Gliederung

Diese Diplomarbeit umfasst die Analyse, die Konzeption, die Entwicklung sowie den Test und die Evaluierung einer Software zur komfortablen Eingabe von Personenprofilen auf der Basis von standardisierten XML-Daten.

Das erste Kapitel beschreibt den Umfang der Diplomarbeit sowie dessen Problemstellung. Die Arbeitsumgebung wird in einem Unterkapitel vorgestellt. Das darauf folgende Kapitel weist auf eingesetzte Technologien hin, die die Grundlage für den Verlauf der Diplomarbeit bilden. Es folgen die Analyse der Problemstellung und die Definition der Anforderungen. Im Analyse-Teil des Kapitels werden Evaluierungen über mögliche Standards und über bereits verfügbare vergleichbare Softwareprodukte durchgeführt. Unter Bezugnahme auf die aus den Evaluierungen erlangten Erkenntnisse wird ein Lösungsansatz für die zu entwickelnde Software definiert. Anhand des Lösungsansatzes werden das Design und die Architektur erstellt.

Nach dem Konzeptkapitel folgt im vierten Kapitel die Implementierung. Es wird anhand von Diagrammen der Aufbau der innerhalb der Diplomarbeit zu entwickelten Software gezeigt. Screenshots veranschaulichen die Bedienoberfläche sowie die möglichen Schritte in denen der Benutzer vorgehen kann. Im fünften Kapitel wird die Testphase ausführlich beschrieben. Dabei werden Ergebnisse einer Umfrage zu dem entwickelten Programm präsentiert und ausgewertet. Der dabei von mehreren Testpersonen benutzte Testkatalog befindet sich im Anhang dieser Diplomarbeit.

Am Ende dieser Ausarbeitung wird im Rahmen des Fazits auf die Umsetzbarkeit der Diplomarbeit aufgabe und die dabei aufgetretenen Probleme und Fragestellungen eingegangen. Es wird ein Resümee über die verwendeten Technologien, über die geleisteten Programmierungen und die erzielten Ergebnisse gezogen.

Im Anhang dieser Diplomarbeit befinden sich, neben dem Testkatalog, ein Glossar, das Tabellenverzeichnis, das Literaturverzeichnis mit allen Quellenangaben, eine Auflistung von Spezifikationen, Unternehmen und Organisationen sowie ein Quellcodeverzeichnis.

## 1.2 Problemstellung

Große Unternehmen verlangen immer häufiger einen Lebenslauf, der direkt auf der Firmenwebseite eingegeben werden muss. Dies garantiert dem Unternehmen, dass in den erfassten Lebensläufen nur die Daten stehen, die für das Unternehmen relevant sind. Außerdem liegt die Bewerbung so in digitaler Form vor und kann einfach und einheitlich weiterverarbeitet und vorsortiert werden. Allerdings führt dies zu einem großen Mehraufwand für einen Bewerber, da heutzutage viele Bewerbungen bei mehreren Unternehmen üblich sind. Muss jetzt für jede Bewerbung der Lebenslauf erneut erstellt werden, so ist für eine immer wiederkehrende Arbeit ein hoher Zeitaufwand notwendig.

Neben den Firmen, die einen eigenen Lebenslauf verlangen, gibt es auch eine Vielzahl an Jobportalen, bei denen ein Arbeitssuchender ebenfalls seine Lebenslaufdaten hinterlegen kann. Für den Arbeitssuchenden hat dies zwar den Vorteil, dass die Firmen, die über diese Jobportal-Webseiten angeschrieben werden, die Lebenslaufdaten besitzen, doch ein derartiger Lebenslauf kann nicht weiterverwendet werden. Er ist somit nur auf dem jeweiligen Jobportal zugänglich. Meldet sich ein Bewerber bei einem weiteren Jobportal an, muss der Lebenslauf erneut eingegeben werden.

Es hat sich bisher noch kein Standard auf dem deutschen Markt etabliert, der dieses Problem angeht und den Bewerbungsvorgang vereinfacht und standardisiert.

## 1.3 Ziele der Diplomarbeit

Ziel der Diplomarbeit ist es, eine Lösung für die Problemstellung zu finden, mit der ein Bewerber einen Lebenslauf erstellen und ihn mehrfach verwenden kann. Dafür muss ein Standard zur Haltung der Daten verwendet werden, mit dem die Unternehmen und Jobportale arbeiten können.

Ein möglicher Lösungsansatz ist hierbei die Speicherung der Daten in einem standardisiertem XML-Format. Es gibt bereits verschiedene Standards, um Daten zu speichern. Einer dieser Standards ist der HR-XML-Standard des HR-XML-Konsortiums [*HR-XML*].

Im Rahmen der Diplomarbeit müssen folgende Punkte untersucht werden:

- Ist der genannte HR-XML-Standard ausreichend definiert, um als Datenhaltungssprache eingesetzt werden zu können.

- Gegebenfalls muss nach alternativen Standards zur Speicherung der Daten gesucht werden.
- Gibt es bereits Programme zur komfortablen Eingabe von Personenprofilen und welche Stärken und Schwächen weisen sie aus.
- Welche Standards werden durch diese Programme unterstützt.

Abhängig von den Ergebnissen der Untersuchungen muss ein Konzept für ein benutzerfreundliches Programm entworfen werden, das es ermöglicht, auf einer komfortablen Art den Lebenslauf in einem standardisierten Format zu erstellen und zu verarbeiten. Dabei soll berücksichtigt werden, dass die zu entwickelnde Software für eine weit gefasste Endbenutzergemeinde geeignet sein muss.

Durch die Software soll der Bewerbungsablauf für den Benutzer erheblich vereinfacht werden. Für die Unternehmen soll es eine Möglichkeit darstellen, standardisierte Lebensläufe zu erhalten, die leicht zu verarbeiten sind und bei denen ein Bewerber keinen Mehraufwand durch mehrmaliges Erfassen der Daten hat. Gleichzeitig behält der Benutzer die Datenhoheit über seinen Lebenslauf, da er die Daten unabhängig von Internetportalen auf seinem eigenen Computer erfassen kann.

#### **1.4 Arbeitsumgebung**

Die Diplomarbeit wurde in Zusammenarbeit mit der Firma eWorks GmbH [EWORKS] konzeptioniert und erstellt. Während der Entwicklung wurde an zwei verschiedenen Computersystemen mit unterschiedlichen Konfigurationen gearbeitet, um verschiedene Betriebssysteme leichter berücksichtigen zu können und mobil in der Arbeit zu sein.

Büro-Umgebung:

- Betriebssystem: Microsoft Windows Vista
- Entwicklungsumgebung: Visual Studio 2005 Professional Edition
- Office: Microsoft Office 2007 und Microsoft Office 2003

Laptop-Umgebung:

- Betriebssystem: Microsoft Windows XP SP 2
- Entwicklungsumgebung: Visual Studio 2005 Express Edition
- Office: Open Office 2.2.1 und Microsoft Office 2000

Auf beiden Systemen wurde für die Versionskontrolle CVS benutzt, damit zum Einen immer mit der aktuellen Quellcodeversion gearbeitet und zum Anderen ein Backup der Daten verfügbar war. Für die Zeiterfassung wurde TimeEngine<sup>1</sup>, eine von der eWorks GmbH entwickelte Software, benutzt. Anhand der damit gesammelten Daten kann ein Überblick gegeben werden, für welche Teile der Diplomarbeit wie viel Zeit benötigt wurde. In der folgenden Abbildung ist die benötigte Zeit prozentual auf verschiedene Projektabschnitte aufgeteilt zu sehen:

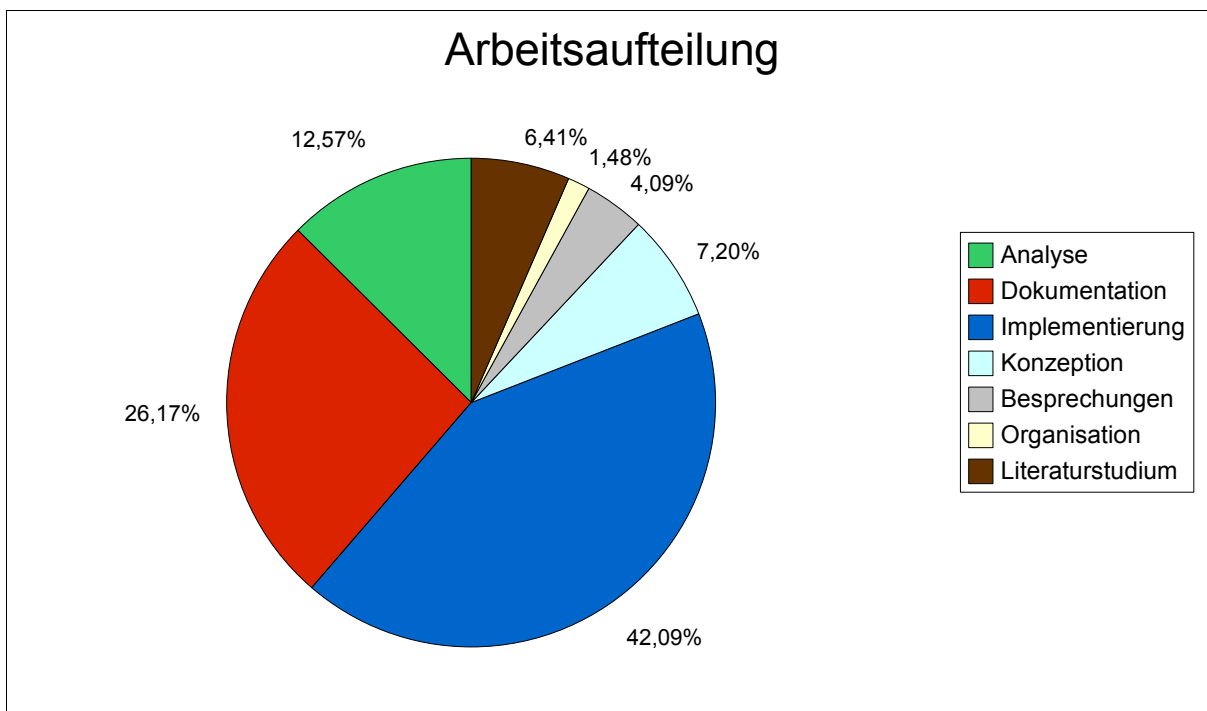


Abbildung 1: Aufteilung der benötigten Zeit

Anhand der Abbildung ist zu sehen, dass für die Implementierung der zu entwickelten Anwendung die meiste Zeit (42,09 %) benötigt wurde. Den zweit größten Wert erforderte die Dokumentation. Danach folgen die Analyse, die Konzeption und das Literaturstudium. Besprechungen und Organisation haben zusammen ca. 5,5 % der Zeit in Anspruch genommen.

<sup>1</sup> <http://www.timeengine.de/> (Stand: 22.01.2008)

## 2 GRUNDLAGEN

In diesem Kapitel sollen die Grundlagen zu der Beschreibungssprache XML und dem .NET Frameworks dargelegt werden, damit darauf im weiteren Verlauf der Diplomarbeit Bezug genommen werden kann.

### 2.1 Extensible Markup Language (XML)

Die „Extensible Markup Language“ (kurz: XML) ist eine Beschreibungssprache für Dokumente und Daten [BPS06]. Die Sprache kann unabhängig von der Betriebssystemplattform und in verschiedenen Anwendungen eingesetzt werden. XML ist eine sogenannte Auszeichnungssprache (englisch: Markup Language) und wurde speziell für den Zweck entwickelt, Dokumente zwischen Plattformen oder Anwendungen austauschen zu können. Die Dokumente werden als Textdateien gespeichert und können somit von jedem Texteditor geöffnet und bearbeitet werden [BOR05].

Im Folgenden wird XML in einer kurzen Einführung näher beschrieben. Da das Konzept von XML sehr umfangreich ist, werden nur grundlegende Eigenschaften von XML betrachtet. Dazu gehören der Aufbau einer XML-Datei, wichtige Syntaxregeln sowie Grundlagen zu XML-Namensräumen. Zudem wird der zur Formatdefinition relevante XML-Schema-Standard (kurz: XSD) in seinen Grundlagen beschrieben.

#### 2.1.1 Der Aufbau

Ein XML-Dokument besteht aus einem Prolog und Daten. Der Prolog besteht aus einer Kopfzeile und der Document Type Definition (DTD) oder einem Verweis auf diese DTD. Der Verweis zur DTD ist nicht zwingend erforderlich. Um das XML-Dokument zu vereinfachen, kann der Verweis zu der DTD auch weggelassen werden. Auf bestimmte Funktionen muss dann aber verzichtet werden, zum Beispiel die Überprüfung des Dokumentes. Allerdings kann dies durch die Verwendung eines speziellen XML-Parsers umgangen werden. Dafür muss intern festgelegt werden, welche Elemente zu erwarten sind.

Die Kopfzeile enthält den Header des XML-Dokuments mit der XML-Deklaration. Es handelt sich dabei um die in dem XML-Dokument verwendete XML-Version. Dadurch kann ein Parser beim zeilenweisen Einlesen des Dokuments den Text auf Übereinstimmung mit den XML-Spezifikationen überprüfen.

Außerdem kann durch die Angabe des Headers erkannt werden, dass es sich bei dem Dokument nicht um eine einfache Textdatei, sondern um ein XML-Dokument handelt.

Die Kopfzeile muss mit einem Kleiner-Zeichen, einem Fragezeichen und dem Präfix `xml` beginnen (`<?xml`). Es folgt die Versionsnummer des XML-Dokuments. Die Zeile muss mit einem Fragezeichen und einem Größer-Zeichen (`?>`) abschließen.

Nach der Versionsnummer kann die Kodierung des Dokuments angegeben werden. Es gibt eine Vielzahl von verschiedenen Zeichensätzen, die in der folgenden Tabelle dargestellt werden. Der Zeichensatz kann durch die Angabe von `encoding` angegeben werden:

---

```
<?xml version="1.1" encoding="ISO-8859-1" ?>
```

---

In der nachfolgenden Tabelle sind die gültigen Zeichenkodierungen nach ISO-8859 angegeben:

Kodierung	Bezeichnung
UTF-8	Internationaler Zeichensatz (8 Bit)
UTF-16	Internationaler Zeichensatz (16 Bit)
ISO-8859-1	Westeuropa
ISO-8859-2	Osteuropa
ISO-8859-3	Südeuropa
ISO-8859-4	Nordeuropa
ISO-8859-5	Kyrillisch
ISO-8859-6	Arabisch
ISO-8859-7	Griechisch
ISO-8859-8	Hebräisch
ISO-8859-9	Türkisch
ISO-8859-10	Nordisch

**Tabelle 1: Zeichenkodierung nach ISO-8859** <sup>1</sup>

---

<sup>1</sup> Quelle: <http://alis.isoc.org/codage/iso8859/jeuxiso.de.htm> (Stand: 22.01.2008)

Die Daten des Dokuments werden in einem Dokumenteninhalte gehalten, wobei sie auch verschachtelt dargestellt werden können. Der Dokumenteninhalte besteht aus 1 bis n Elementen. Ein Element wiederum besteht aus einem Namen, der in spitzen Klammern eingeschlossen ist, was einen sogenannten Tag darstellt. Jeder einleitende Tag muss einen zugehörigen abschließenden Tag auf der gleichen Hierarchieebene und Reihenfolge besitzen. Der abschließende Tag wird mit einem Schrägstrich (/) gekennzeichnet. Der Wert des XML-Elementes steht dabei zwischen dem einleitenden und dem abschließenden Tag.

---

```
(1) <?xml version="1.1" ?>
(2) <browser>
(3)   <name>Internet Explorer</name>
(4)   <name>Firefox</name>
(5) </browser>
```

---

#### ***Quelltext 1: XML-Beispiel***

In dem Quellcodebeispiel 1 werden zwei Web-Browser angegeben. Die Elemente stehen alle innerhalb eines Wurzelementes (hier: `<browser>`). Jeder Web-Browser hat einen Namen. Es wäre möglich, die Daten zu schachteln und noch weitere Informationen mit aufzunehmen, wie zum Beispiel den Namen des Herstellers.

---

```
(1) <?xml version="1.1" ?>
(2) <browser>
(3)   <browser-typ>
(4)     <name>Internet Explorer</name>
(5)     <hersteller>Microsoft</hersteller>
(6)   </browser-typ>
(7)   <browser-typ>
(8)     <name>Firefox</name>
(9)     <hersteller>Mozilla</hersteller>
(10)  </browser-typ>
(11) </browser>
```

---

#### ***Quelltext 2: XML-Beispiel erweitert***

In Quellcodebeispiel 2 ist ein Unterelement (`<browser-typ>`) zum Wurzelement (`<browser>`) hinzugekommen. Somit ist es möglich, neben dem Namen auch noch den Hersteller innerhalb des Dokuments zu speichern. Auch weitere Angaben wie Versionsnummer, Plattform etc. sind denkbar.

Innerhalb eines XML-Dokuments können auch leere Tags definiert werden. Dazu wird der Bereich zwischen einleitendem und abschließendem Tag leer gelassen. Durch eine vereinfachte Notation kann ein leerer Tag verkürzt dargestellt werden. Aus `<leer></leer>`, wird dann `<leer />`. Dem einleitenden Tag wird ein Schrägstrich (/) hinzugefügt. Dadurch erkennt der XML-Parser, dass der Tag geschlossen ist und keine weiteren Daten beinhaltet.

Eine weitere Möglichkeit zur Aufnahme von Daten, sind die so genannten Attribute. Die Attribute beschreiben die Eigenschaften des XML-Elements. Innerhalb der DTD kann definiert werden, ob ein Element kein, ein oder mehrere Attribute besitzen kann.

---

```
(1) <?xml version="1.1" ?>
(2) <browser>
(3)   <!-- Kommentar: Auflistung der Browser -->
(4)   <browser-typ name="Internet Explorer" hersteller="Microsoft />
(5)   <browser-typ hersteller="Mozilla">Firefox</browser-typ>
(6) </browser>
```

---

### *Quelltext 3: XML-Beispiel mit Attributen*

Das Quellcodebeispiel 3 zeigt eine einfache XML-Datei, in der zwei Web-Browser beschrieben werden. Beim ersten Web-Browser werden der Name und der Hersteller innerhalb von `browser-typ` als Attribut angegeben. Der zweite Web-Browser gibt nur den Hersteller mit dem entsprechenden Attribut an, der Name des Web-Browsers steht zwischen dem einleitenden und dem abschließenden Tag. Wie zu sehen ist, sind je nach Definition innerhalb der DTD auch verschiedene Schreibweisen zulässig.

Weiterhin ist im Quellcodebeispiel 3 auch ein Kommentar enthalten. Kommentare werden in XML durch die Zeichenfolge `<!--` eingeleitet und mit `-->` wieder geschlossen.

Die Wurzelemente und Unterelemente ergeben eine Hierarchie in Form eines Baums.

### 2.1.2 Syntaxregeln von XML

- Jeder einleitende Tag muss auch einen abschließenden Tag besitzen.
- Leere Elemente benötigen entweder einen End-Tag oder müssen mit einem Schrägstrich beendet werden.
- Attribut-Werte innerhalb von Tags müssen in Anführungszeichen (") oder Hochkommata (') gesetzt werden.
- Jedes Element muss mit einem Buchstaben oder einem Unterstrich beginnen, Sonderzeichen dürfen nicht verwendet werden.
- Da XML zwischen Groß- und Kleinschreibung unterscheidet (engl: case-sensitive), muss auf gleiche Schreibweise für einleitende und abschließende Tags geachtet werden.

Wenn alle Syntaxregeln eingehalten werden, spricht man von einem wohlgeformten XML-Dokument. Folgt das XML-Dokument einer Grammatik, so nennt man es typgerecht oder gültig (valide) [GER00].

### 2.1.3 Namensräume

Durch Einführung von Namensräumen (engl.: „namespaces“) wird die Möglichkeit gegeben, Elemente und Attribute auch in größeren Dokumenten eindeutig zu benennen. Wenn Dokumente zum Beispiel verschiedene externe und interne Inhalte verwenden, können Elemente oder Attribute nicht mehr eindeutig sein. Namensräume ermöglichen es, ein Element mit einem Kontext zu versehen, so dass es innerhalb eines Dokumentes in verschiedenen Zusammenhängen verwendet werden kann.

Um innerhalb eines Dokumentes einen Namensraum nutzen zu können, muss er in folgender Syntax angegeben werden:

---

Syntax: Präfix:Elementname

---

Der Präfix besteht aus dem Schlüsselwort `xmlns` (steht für XML Namespace) [BHL06]. Innerhalb eines XML-Dokuments können mehrere Namensräume angegeben werden. Dem Schlüsselwort folgt der Name des zu deklarierenden Namensraumes. Über diesen Namen kann der Namensraum benutzt werden.

Dem Elementnamen muss auch noch der Ort der Deklaration anhand einer URI (Uniform Resource Identifier) [RFC3986] zugewiesen werden.

---

Syntax: xmlns:Elementname=URI

Für ein XML-Dokument kann auch ein Standard-Namensraum vergeben werden. Dabei muss der Elementname weggelassen werden, die Syntax lautet dann wie folgt:

---

Syntax: xmlns=URI

Das folgende Quellcodebeispiel 4 zeigt, wie ein Namensraum innerhalb eines XML-Dokumentes verwendet werden kann [SKO01]:

---

```
(1) <?xml version="1.1" encoding="ISO-8859-1" ?>
(2) <students xmlns:s="http://www.develop.com/student">
(3)   <s:student>
(4)     <s:id>321456</s:id>
(5)     <s:name>Hans Mustermann</s:name>
(6)   </s:student>
(7) </students>
```

---

**Quelltext 4: Namensräume in einem XML-Dokument verwenden**

In der ersten Zeile steht zunächst die XML-Deklaration, in der zweiten Zeile wird dann der verwendete Namensraum eingebunden. Danach wird ein Student anhand seiner Matrikelnummer und seines Namens beschrieben. Die einzelnen Elemente eines Studenten werden durch den Elementnamen `s` angegeben. Dieser muss vor `student`, `id` und `name` stehen. Da hier nur ein Namensraum angewendet wird, könnte bei dem Beispiel auch ein Standard-Namensraum verwendet werden. Auf den Elementnamen `s` könnte dadurch verzichtet werden.

### 2.1.4 XML-Schema

XML ermöglicht es, beliebige Elemente und Attribute innerhalb eines XML-Dokuments zu verwenden. Dafür werden die bereits erwähnten DTDs oder die neueren und mächtigeren XML-Schemata verwendet. XML-Schema ist selbst eine XML-Anwendung und bietet somit mehr Möglichkeiten als eine DTD. Aus diesem Grund und da in der vorliegenden Diplomarbeit nur XML-Schemata eingesetzt werden, wird auf DTD nicht weiter eingegangen und dafür eine Grundlage für den Umgang mit XML-Schemata geschaffen. Den Aufbau einer einfachen XML-Schema-Datei zeigt das folgende Quellcodebeispiel:

---

```

(1) <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" >
(2)   <xs:element name="tools">
(3)     <xs:complexType>
(4)       <xs:sequence>
(5)         <xs:element name="browser">
(6)           <xs:complexType>
(7)             <xs:sequence>
(8)               <xs:element name="browser-typ"
(9)                 maxOccurs="unbounded" minOccurs="1">
(10)                <xs:complexType>
(11)                  <xs:attribute name="name" type="xs:string"
(12)                    use="required"/>
(13)                  <xs:attribute name="hersteller"
(14)                    type="xs:string" default="null" use="optional"/>
(15)                </xs:complexType>
(16)              </xs:element>
(17)            </xs:sequence>
(18)          <xs:attribute name="version" type="xs:string"
(19)            default="null" use="optional"/>
(20)        </xs:complexType>
(21)      </xs:element>
(22)    </xs:sequence>
(23)  </xs:complexType>
(24) </xs:element>
(25) </xs:schema>

```

---

#### Quelltext 5: XML-Schema-Beispiel

In Quellcodebeispiel 5 ist ein XML-Schema gegeben, mit dem ein Datensatz von Web-Browsern validiert werden kann, wie es bereits im vorherigen Abschnitt beschrieben wurde. In der ersten Zeile wird der Namensraum des XML-Schemata-Formats angegeben. Danach erfolgt die Definition des XML-Schemas. Die Endung einer XML-Schema-Datei lautet XSD (XML-Schema-Definition).

Der angegebene Namensraum enthält die XML-Schema-Elemente, die im Beispiel verwendet werden (`<schema>`, `<element>`, `<complexType>` usw.).

Mit dem `<schema>`-Tag wird das XML-Schema eingeleitet. Danach folgt ein Element mit dem Namen `tools`. Dieses Element muss mindestens einmal im zu validierenden XML-Dokument vorhanden sein. Das Element beinhaltet einen `<complexType>`, der wiederum eine `<sequence>` besitzt. Das Element `<tools>` beinhaltet ein oder mehrere `<browser>`-Elemente.

Auch das Element `<browser>` hat wieder einen `<complexType>` und eine `<sequence>`. Innerhalb dieser Sequenz kann ein `<browser-typ>` angegeben werden. In dem Element `<browser>` muss es mindestens einen `<browser-typ>` geben (`minOccurs="1"`). Die Anzahl der Browser-Typen ist unbeschränkt (`maxOccurs="unbounded"`).

Jeder `<browser-typ>` besitzt einen Namen und einen Hersteller. Beides wird jeweils als Attribut angegeben. Die Angabe eines Herstellers ist optional (`use="optional"`), die des Namens hingegen zwingend erforderlich (`use="required"`).

Auch das Browser-Element besitzt ein Attribut:

---

```
<xs:attribute name="version" type="xs:string" default="null"
use="optional"/>
```

---

Die Angabe dieses Elements ist wieder optional. Für jedes Element kann auch ein Datentyp angegeben werden. In dem vorliegenden Beispiel wird aber nur `String` als Datentyp verwendet. Es könnten auch weitere Datentypen eingesetzt werden, wie zum Beispiel `Integer`, `Double` oder `Boolean`.

---

```
(1) <?xml version="1.0" encoding="UTF-8"?>
(2) <tools xmlns:x="http://www.w3.org/2001/XMLSchema-instance"
(3)       x:noNamespaceSchemaLocation=".\\schema.xsd">
(4)   <browser version="">
(5)     <browser-typ name="Firefox" />
(6)     <browser-typ name="Internet Explorer" hersteller="Microsoft" />
(7)   </browser>
(8) </tools>
```

---

*Quelltext 6: Ein dem XML-Schema konformes XML-Beispiel*

In Quellcodebeispiel 6 wird gezeigt, wie eine gültige XML-Datei zu dem Schema in Quellcodebeispiel 5 aussehen kann. Das Wurzelement `<tools>` beinhaltet einen Web-Browser. Innerhalb von `<browser>` gibt es zwei Web-Browser-Typen (Firefox-Browser und Internet Explorer).

Nicht zulässig wären zum Beispiel folgende Elemente:

---

```
<browser-typ hersteller="Mozilla" />
```

---

Das erforderliche Attribut `name` ist hier nicht vorhanden.

---

```
<tools...>
  <browser>
    ...
  </browser>
  <browser>
    ...
  </browser>
</tools>
```

---

Das Element `<browser>` darf innerhalb des Wurzelementes `<tools>` nur einmal vorkommen.

---

```
<tools ...>
  <browser></browser>
</tools>
```

---

Das Element `<browser>` muss mindestens ein `<browser-ty>`-Element enthalten.

### 2.1.5 XSL

Die Extensible Stylesheet Language (kurz: XSL) bietet ein mächtiges Werkzeug mit dem u.a. XML-Dokumente in neue XML-Dokumente mit einem anderen Aufbau oder in HTML-Dokumente transformiert werden können. XSL besitzt eine XML-Syntax und bietet zahlreiche Elemente zum Arbeiten mit XML-Daten. Im Folgenden werden einige dieser Elemente vorgestellt. Ein komplettes Quellcodebeispiel einer XSL-Datei befindet sich in dem Kapitel „C# und XML“. Zahlreiche Beispiele und Erklärungen zum Thema XSL sind in dem Buch „XSLT“ von Daniel Koch [KOC07] zu finden.

Innerhalb des einleitenden Tags wird ein XSL-Namensraum angegeben:

---

```
<xsl:stylesheet ... xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
```

---

Mit Hilfe dieses Namensraumes stehen die XSL-Elemente bereit. In dem obigen Beispiel deuten die drei Punkte an, dass zwischen `xsl:stylesheet` und der Definition des Namensraumes noch weitere Definitionen stehen, wie unter anderem die Version des vorliegenden Stylesheets.

Als Stylesheet werden die XSL-Dateien umgangssprachlich benannt. Danach wird mitgeteilt, welche Befehle das Stylesheet genau ausführen soll. Dafür ist folgende Zeile zuständig:

---

```
<xsl:output indent="yes" method="xml"/>
```

---

In dem Quellcodeausschnitt wird angegeben, dass ein Output geschehen und die resultierende Datei eine XML-Datei sein soll. Hier könnte anstatt `xml` auch `html` stehen. Ist der Output ausgeführt kann die XML-Datei, die mit diesem Stylesheet transformiert werden soll, ausgelesen werden. Ein komplettes Erfassen der Datei ist mit Angabe eines „/“ zu erreichen:

---

```
<xsl:template match="/">
```

---

Danach existiert die XML-Datei als eine interne Baumstruktur und die XSL-Datei kann auf einzelne XML-Elemente zugreifen.

Für die Arbeit mit der eingelesenen XML-Datei bietet XSL Kontrollstrukturen und Schleifen an. Folgender Ausdruck holt aus dem Root-Element `tools` alle `browser`:

---

```
<xsl:for-each select="//tools/browser">
...
</xsl:for-each>
```

---

Zwischen dem einleitendem und dem schließendem `for-each`-Tag kann mit den Daten gearbeitet werden. Die `for-each`-Schleife arbeitet dabei wie die aus Hochsprachen bekannten `foreach`-, `while`- oder `for`-Schleifen. Neben Schleifen gibt es auch eine Äquivalente zu einer `switch-case`-Anweisung. Innerhalb von XSL heißt diese `choose`.

---

```
<xsl:choose>
  <xsl:when test="browser-typ/@name='Opera'">Opera</xsl:when>
  <xsl:when test="browser-typ/@name='Firefox'">Firefox</xsl:when>
  <xsl:otherwise>Internet Explorer</xsl:otherwise>
</xsl:choose>
```

---

Zwischen dem einleitendem und dem schließendem Tag stehen die Anweisungen. In diesem Fall wird innerhalb einer `for-each`-Schleife auf die einzelnen Browser zugegriffen und untersucht, ob es sich dabei um den Opera oder den Firefox Browser handelt. Trifft keines von beiden zu wird angenommen, dass es sich um den Internet Explorer handelt. Neben der erwähnten `switch-case`-Variante `choose` gibt es auch die aus vielen Hochsprachen bekannte `if`-Anweisung:

---

```
<xsl:if test="browser-typ/@name='Opera'">
...
</xsl:if>
```

---

Auch hier wird wieder auf das `name`-Attribut des Elements `browser-typ` zugegriffen und verglichen, ob es sich um Opera handelt. Wie bereits gesehen, werden so Daten innerhalb der XML-Elemente ausgelesen. Die Daten können aber auch direkt ausgelesen und in das neue Dokument geschrieben werden. Dafür gibt es den XSL-Tag `value-of`:

---

```
<xsl:value-of select="browser-typ/@name" />
```

---

Dieser Tag veranlasst, dass das Attribut `name` von `browser-typ` ausgegeben wird. Bei dem Beispiel wird angenommen, dass es nur einen `browser-typ` gibt. Es können aber nicht nur XML-Elemente ausgelesen, sondern auch geschrieben werden.

Im Folgenden wird gezeigt, wie ein XML-Element mit einem Attribut erstellt werden kann:

```
<xsl:element name="browser">
  <xsl:attribute name="title">
    Firefox
  </xsl:attribute>
</xsl:element>
```

Die Ausgabe davon würde so aussehen:

```
<browser title="Firefox" />
```

Anhand der wenigen Beispielen wurde versucht zu demonstrieren, was mit einer XSL-Transformation (kurz: XSLT) möglich ist. Das Thema XSL soll an dieser Stelle allerdings nicht weiter vertieft werden. XSL wird im weiteren Verlauf der Diplomarbeit noch eine Rolle spielen. Dieses Kapitel soll dazu dienen, einen kurzen Überblick über die Möglichkeiten von XSL zu liefern, damit der Begriff XSL verwendet werden kann, ohne auf dessen Eigenschaften im weiteren Verlauf eingehen zu müssen.

### 2.1.6 Fazit

Für die Diplomarbeit und die Entwicklung des angestrebten Programms eignet sich XML am besten zum Speichern der Personendaten, da sich XML als Standard für den Datenaustausch etabliert hat und weit verbreitet ist. Darüber hinaus ist XML plattformunabhängig, was ein weiteres wichtiges Kriterium bei der Frage der Datenhaltung ist. Hinzu kommt, auch oder gerade wegen der weiten Verbreitung von XML, dass es bereits viele hilfreiche Programme und Bibliotheken zum Erstellen und Arbeiten mit XML-Daten gibt, wodurch die in XML gespeicherten Daten von beliebigen Programmen verarbeitet werden können.

Dazu kommt noch die Möglichkeit, die Korrektheit eines XML-Dokumentes gegen ein XML-Schema überprüfen zu können, wodurch einem Entwickler viel Arbeit im Bezug auf die Validierung abgenommen wird. Darüber hinaus kann mit Hilfe von XSL ein XML-Dokument in ein anderes XML-Dokument transformiert werden, ohne dass das Dokument selbst verarbeitet werden muss. Dies könnte dazu verwendet werden, zwischen verschiedenen XML-Standards zu transformieren, oder aus einer XML-Datei heraus eine HTML-Datei zu generieren.

Oben genannte Punkte spielen eine wichtige Rolle bei der Entscheidung, XML als Datenhaltungssprache für das Projekt innerhalb der Diplomarbeit einzusetzen.

## 2.2 .NET Framework

Das .NET Framework besteht aus Laufzeitbibliotheken, der sogenannten Common Language Runtime (CLR) und aus Klassenbibliotheken, der Framework Class Library (FCL). Das Framework wurde von der Firma Microsoft [MSOFT] entwickelt und soll dabei helfen, die Entwicklung von Diensten und Anwendungen zu vereinfachen [RIC02].

Die CLR ist die virtuelle Maschine (VM) des .NET Frameworks und stellt somit die Laufzeitumgebung für verschiedene, an .NET angepasste, Hochsprachen dar. Über die CLR können in unterschiedlichen Sprachen entwickelte Programme, bzw. Programmteile, auf gemeinsame Ressourcen zugreifen. Dies funktioniert, in dem der Compiler aus der verwendeten Programmiersprache zunächst eine plattformunabhängige Sprache generiert, die sogenannte Common Intermediate Language (kurz: CIL). Aus der CIL wird die CLR kompiliert, welche dann in Maschinencode umgewandelt wird. Anhand der folgenden Grafik wird dieser Vorgang noch einmal veranschaulicht [KUE06].

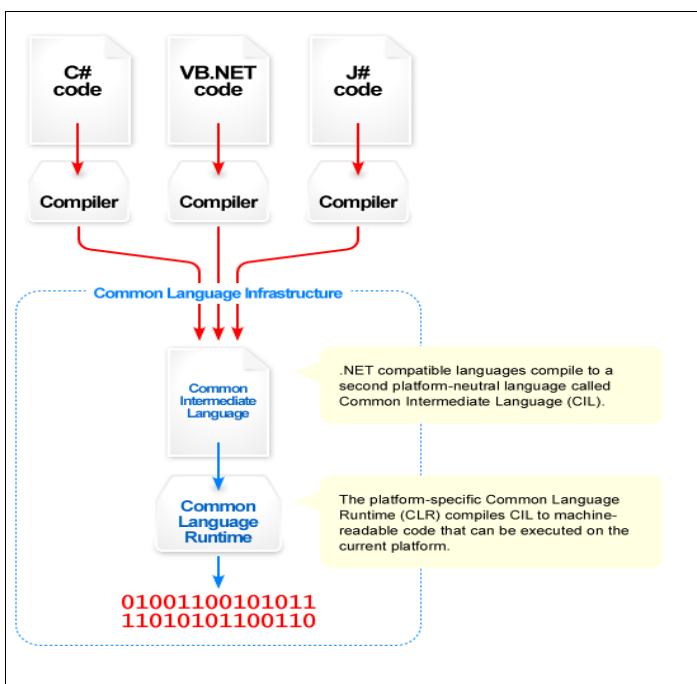


Abbildung 2: .NET Framework Infrastruktur<sup>1</sup>

<sup>1</sup> Quelle: <http://www.answers.com/topic/net-framework?cat=technology> (Stand: 22.01.2008)

Die Klassenbibliothek FCL umfasst mehrere tausend Klassen, die in so genannten Namensräumen (engl.: „namespaces“) unterteilt sind. Die Klassen stellen Funktionen bereit, wie zum Beispiel zum Arbeiten mit XML-Dateien, Formatieren von Texten oder zum Versenden von E-Mails. Aus Gründen der Übersicht wurden diese Klassen in entsprechende Namensräume zusammengefasst.

Zu den besonderen Stärken des Frameworks zählen neben der Verwaltung von Ressourcen die Wiederverwendbarkeit und das individuelle Anpassen von Code. Im weiteren Verlauf dieses Kapitels wird eine Grundlage für den Umgang mit der .NET-Programmiersprache C# geschaffen.

### 2.2.1 Einstieg in C#

C# (sprich „C sharp“) ist eine objektorientierte Programmiersprache [BUC01], die definiert wurde, um .NET mit dessen Möglichkeiten und Einschränkungen vollständig abbilden zu können. Die Syntax der Sprache ist eng mit C, C++ und Java verwandt.

---

```
(1) using System;
(2) class MyHelloClass
(3) {
(4)     public static void Main()
(5)     {
(6)         Console.WriteLine("Hello C# World");
(7)     }
(8) } //end of class
```

---

#### *Quelltext 7: „Hello World“-Programm mit C#*

Quellcodebeispiel 7 zeigt den Quelltext eines einfachen Programms, das einen Text auf der Konsole ausgibt. Dies wird durch das Objekt `Console` und dessen Methode `WriteLine` erledigt. Beides steht durch Einbinden des Namensraumes `System` mittels der `using`-Direktive zur Verfügung.

Die meisten Anweisungen und Schlüsselwörter sind aus der Programmiersprache C++ übernommen worden. C# ist allerdings typsicher, was bedeutet, dass alle Typen durch Klassen beschrieben werden, die alle von der Klasse `Object` abgeleitet werden.

## 2.2.2 C# Datentypen

Nachfolgend werden die in C# verfügbaren Datentypen anhand einer Tabelle dargestellt:

Name	Beschreibung
sbyte	Integerwert zwischen -128 und +127
byte	Integerwert zwischen 0 und +255
short	Integerwert zwischen -32768 und +32767
ushort	Integerwert zwischen 0 und +65535
int	Integerwert zwischen -2 <sup>31</sup> +2 <sup>31</sup> -1
uint	Integerwert zwischen 0 und +2 <sup>32</sup> -1
long	Integerwert zwischen -2 <sup>63</sup> und +2 <sup>63</sup> -1
ulong	Integerwert zwischen 0 und +2 <sup>64</sup> -1
bool	Boolescher Wert (true oder false)
char	Ein einzelnes Unicode-Zeichen
string	Unicode-Zeichenkette
float	32-Bit-Fließkommazahl mit 7 Stellen Genauigkeit
double	64-Bit-Fließkommazahl mit 15 Stellen Genauigkeit
decimal	128-Bit-Wert mit 28 bis 29 Stellen Genauigkeit im Dezimalsystem
object	Ein Objekt

**Tabelle 2: Basistypen in C#<sup>1</sup>**

<sup>1</sup> Quellen: <http://www.galileocomputing.de/openbook/csharp/kap03.htm> (Stand: 22.01.2008) und „C# universell programmieren von Anfang an“ von Helmut Erlenkötter [ERL02]

### 2.2.3 Garbage Collection

Der Garbage Collector der .NET Runtime (kurz: GC) sorgt dafür, dass belegter Speicher von nicht mehr benötigten Objekten der Anwendung auf dem Heap wieder freigegeben wird. Dies hat den Vorteil, dass bei der Entwicklung kein Code mehr geschrieben werden muss, der die Lebensdauer von Objekten überwacht. Da das Ausführen des Garbage Collectors sehr speicherintensiv ist, werden die Objekte in verschiedene Generationen eingeteilt. Wird der GC ausgeführt, so untersucht er immer nur die Objekte der Generation 0. Ausgeführt wird der GC sobald die Generation 0 voll, das heißt, alle der Generation 0 zur Verfügung stehenden Speicherplätze belegt sind. Die Generation 1 wird anschließend zur neuen Generation 0 usw.

Der GC verhindert Ressourcenlecks, da alle Objekte, auf die nicht mehr zugegriffen werden kann, früher oder später beseitigt werden. Auf ein Objekt, das bereits freigegeben wurde, kann nicht zugegriffen werden, da alle noch zugreifbaren Objekte nicht freigegeben werden. Der GC ist ein effektives Mittel zur Beschleunigung der Applikationsentwicklung, der in vielen modernen Programmiersprachen zum Einsatz kommt.

### 2.2.4 Begriffe aus dem .NET-Umfeld

Im Folgenden sollen einige Begriffe aus dem .NET-Umfeld erklärt werden, die bei der späteren Konzeption und Implementierung verwendet werden. Die Begriffe werden anhand von Beispielen erklärt und sollen einen groben Einblick in die Materie schaffen, ohne sie weiter zu vertiefen. Die Beispiele schaffen eine Grundlage worauf bei der späteren Konzeption und Implementierung aufgebaut werden kann .

#### 2.2.4.1 Namensräume und Klassen

Durch Namensräume können Klassen in verschiedene Gruppen unterteilt werden. Sie eignen sich zum Sortieren und Kategorisieren der Klassen. Neben den Standard-Namensräumen der Bibliotheken können auch eigene Namensräume entworfen werden. Im folgenden Beispiel wird das vorhergehende „Hello World“-Beispiel in einem Namensraum eingebunden:

---

```
(1) using System;
(2) using System.Collections.Generic;
(3) using System.Text;
(4)
(5) namespace MyHelloWorld.Utils
(6) {
(7)     class HelloWorld
(8)     {
(9)     }
(10) }
```

---

#### **Quelltext 8: "Hello World" innerhalb eines Namensraumes**

Die Klasse `HelloWorld` befindet sich innerhalb des Namensraumes `MyHelloWorld.Utils`. `Utils` stellt eine Untermenge des Namensraumes `MyHelloWorld` dar. Hier könnten sich auch noch weitere Klassen befinden. Das Quellcodebeispiel 8 wird jetzt noch um eine Methode zur Ausgabe eines Textes erweitert:

---

```
(1) using System;
(2) using System.Collections.Generic;
(3) using System.Text;
(4)
(5) namespace MyHelloWorld.Utils
(6) {
(7)     class HelloWorld
(8)     {
(9)         public void PrintMessage(String msg)
(10)        {
(11)            Console.WriteLine("Ausgabe: " + msg);
(12)        }
(13)    }
(14) }
```

---

#### **Quelltext 9: "Hello World"-Beispiel mit einer Methode**

Die Klasse wurde um die `public` Methode `PrintMessage` erweitert. Als Parameter kann dieser Methode ein `String` mit dem Namen `msg` übergeben werden. Innerhalb der Methode (Zeile 11) wird die übergebene Nachricht zusammen mit einem Text auf der Konsole ausgegeben. Die Methode `WriteLine` des Objekts `Console` bewirkt, dass am Ende des Textes ein Zeilenumbruch erfolgt.

---

```
(1) using System;
(2) using System.Collections.Generic;
(3) using System.Windows.Forms;
(4) using MyHelloWorld.Utills;
(5)
(6) namespace MyHelloWorld
(7) {
(8)     /// <summary>
(9)     /// Diese Klasse wird von der Anwendung als erstes geladen
(10)    /// </summary>
(11)    static class ApplicationStarter
(12)    {
(13)        /// <summary>
(14)        /// Der Haupteinstiegspunkt für die Anwendung.
(15)        /// </summary>
(16)        [STAThread]
(17)        static void Main()
(18)        {
(19)            HelloWorld hw = new HelloWorld();
(20)            hw.PrintMessage("Hello World");
(21)        }
(22)    }
(23) }
```

---

**Quelltext 10: Aufruf der Methode `PrintMessage` aus einer anderen Klasse heraus**

In Quellcodebeispiel 10 wird eine Klasse gezeigt, die als Haupteinstiegspunkt der Anwendung dient. Der Name der Klasse lautet `ApplicationStarter`. Innerhalb der Klasse gibt es eine `static` Methode mit dem Namen `Main`, die in .NET-Applikationen als Einstiegspunkt definiert werden kann. Die Klasse befindet sich in dem Namensraum `MyHelloWorld`.

Um die in dem vorhergehenden Beispiel gezeigte `PrintMessage`-Methode aufrufen zu können, muss zunächst der entsprechende Namensraum eingebunden werden. Dies geschieht durch das Schlüsselwort `using`. In Zeile 4 wird der Namensraum eingebunden, der für die `HelloWorld`-Klasse benötigt wird. Danach muss ein `HelloWorld`-Objekt erzeugt werden. Dies geschieht im Beispiel in Zeile 19 durch das Schlüsselwort `new`. Danach kann die Methode `PrintMessage` des erzeugten `HelloWorld`-Objektes `hw` aufgerufen werden. Da die Methode einen `String` als Parameter erwartet, muss in den Klammern ein `String` (hier "Hello World") angegeben werden. Die Ausgabe auf der Konsole würde beim Start der Anwendung folgendermaßen aussehen:

---

Ausgabe: Hello World

---

### 2.2.4.2 Objekte

Die Sprache C# ist eine objektorientierte Sprache und basiert vollständig auf Klassen. Der Unterschied zwischen einer Klasse und einem Objekt liegt darin, dass eine Klasse nur die Definition eines Objektes ist und ein Objekt anhand der Definition erzeugt wird. Von einer Klasse können beliebig viele Objekte erzeugt werden. In diesem Zusammenhang spricht man auch von Instanzieren.

Ein Objekt wird durch eine Klasse beschrieben und verfügt über Datenfelder und Methoden. Die Datenfelder (engl.: „properties“) halten die Daten des Objektes, mit den Methoden kann mit dem Objekt und dessen Daten gearbeitet werden.

In C# ist praktisch alles als ein Objekt definiert. Im folgenden Codeausschnitt wird ein `String` mit dem Namen `myString` erstellt:

---

```
String myString = "Ich bin eine Instanz von String";
```

---

Für die Variable `myString` wird eine Instanz der Klasse `String` erzeugt und der Text "Ich bin eine Instanz von String" zugewiesen.

Vor der Instanzierung hat die Variable den Wert `null`. Wird vor einer Initialisierung auf das `String`-Objekt zugegriffen, so erfolgt eine Fehlermeldung des Compilers, da das Objekt noch `null` ist.

### 2.2.4.3 Exceptions

Eine Exception ist ein Fehlerobjekt, bzw. eine Fehlerklasse. Exceptions werden in C# benutzt, um Fehler zu werfen, d.h. Fehler der Anwendung bekannt zu machen und abzufangen. Mit einem `try-catch`-Block können Exceptions abgefangen werden, was bedeutet, es kann direkt auf einen Fehler reagiert werden. Dadurch wird ein kontrollierter Programmablauf ermöglicht.

Es kann viele Gründe geben, warum eine Exception geworfen wird. Beispielsweise gibt es einen Fehler, wenn auf ein leeres Objekt zugegriffen wird, oder wenn eine Division durch 0 versucht wird. Letzteres wird im folgenden Beispiel veranschaulicht:

---

```
(1) using System;
(2) using System.Collections.Generic;
(3) using System.Windows.Forms;
(4)
(5) namespace MyHelloWorld
(6) {
(7)     /// <summary>
(8)     /// Diese Klasse wird von der Anwendung als erstes geladen
(9)     /// </summary>
(10)    static class ApplicationStarter
(11)    {
(12)        /// <summary>
(13)        /// Der Haupteinstiegspunkt für die Anwendung.
(14)        /// </summary>
(15)        [STAThread]
(16)        static void Main()
(17)        {
(18)            String myString = "0";
(19)            int i = 4;
(20)            i = i / Convert.ToInt32(myString);
(21)            Console.WriteLine(i.ToString());
(22)        }
(23)    }
(24) }
```

---

#### **Quelltext 11: Exception-Beispiel mit Division durch 0**

In Quellcodebeispiel 11 wird in Zeile 18 ein `String` definiert und bekommt eine 0 als Zeichenkette zugewiesen. In der nächsten Zeile wird ein `Integer` `i` mit dem Wert 4 definiert. In Zeile 20 wird der Wert von `i` geteilt durch den Wert von `myString`, wobei der `String` erst in eine `Integer`-Zahl konvertiert wird. Das Ergebnis wird dem `Integer` `i` wieder zugewiesen. Der Compiler weiß an dieser Stelle nicht, dass in dem `String`-Objekt eine 0 steht und gibt keine Fehlermeldung aus. Allerdings kann diese Operation während der Laufzeit nicht ausgeführt werden und es wird eine `DivideByZeroException` geworfen. Zu einer Bearbeitung des restlichen Codes, hier wäre es nur die Ausgabe des Ergebnisses, kommt es nicht mehr.

Mit einem `try-catch`-Block könnte diese Fehlermeldung nun abgefangen werden, was anhand des Beispiels auf der nächsten Seite gezeigt wird. [HUT07]

---

```
(1) using System;
(2) using System.Collections.Generic;
(3) using System.Windows.Forms;
(4)
(5) namespace MyHelloWorld
(6) {
(7)     /// <summary>
(8)     /// Diese Klasse wird von der Anwendung als erstes geladen
(9)     /// </summary>
(10)    static class ApplicationStarter
(11)    {
(12)        /// <summary>
(13)        /// Der Haupteinstiegspunkt für die Anwendung.
(14)        /// </summary>
(15)        [STAThread]
(16)        static void Main()
(17)        {
(18)            String myString = "0";
(19)            int i = 4;
(20)            try
(21)            {
(22)                i = i / Convert.ToInt32(myString);
(23)            }
(24)            catch (DivideByZeroException e)
(25)            {
(26)                i = 2;
(27)            }
(28)            Console.WriteLine(i.ToString());
(29)        }
(30)    }
(31) }
```

---

#### *Quelltext 12: Exception abgefangen durch try-catch*

In dem Quellcodebeispiel 12 wird ab Zeile 20 ein `try-catch`-Block eingefügt, der bei der Division eine mögliche `DivideByZeroException` abfängt und in dem `catch`-Block, der zur Behandlung des Fehlers dient, der Variablen `i` den Wert 2 zuweist. Danach wird der Code weiterhin normal durchlaufen und der Wert von `i` (= 2) wird auf der Konsole ausgegeben.

Anhand dieses einfachen Beispiels wurde der typische Umgang mit potentiellen Fehlern gezeigt. Es wurde ein Weg angegeben, wie Fehler mit Exceptions behandelt werden können. Das Abfangen von Fehlern und damit verbunden, die Vermeidung von nicht aussagekräftigen Fehlermeldungen an den Benutzern, ist ein wesentlicher Bestandteil beim Arbeiten mit .NET und wird auch im weiteren Verlauf dieser Diplomarbeit einen sehr wichtigen Platz einnehmen.

#### **2.2.4.4 Events**

Ein Event stellt ein Ereignis innerhalb der Anwendung dar. Dabei wird eine fremde Methode gekapselt und aufgerufen. Events treten beispielsweise auf, wenn ein Fenster geladen oder gezeichnet, die Maus bewegt oder eine Maus- bzw. Tastaturtaste betätigt wird. Es gibt eine Vielzahl von vordefinierten Events, aber auch eigene Events können definiert werden. Events werden in der späteren Entwicklungsphase häufiger auftreten.

#### **2.2.5 C# und XML**

Das .NET Framework liefert zahlreiche Klassen zum Arbeiten mit XML-Dateien. Im Folgenden wird anhand einfacher Beispiele gezeigt, wie eine XML-Datei erstellt, gegen eine XSD-Datei validiert und schließlich mit Hilfe einer XSL-Datei transformiert werden kann [BOR07]. Auf der nächsten Seite wird eine C#-Datei vorgestellt, die eine XML-Datei lädt, validiert und dann als HTML-Datei mit Hilfe einer XSL-Datei transformiert.

---

```
(1) using System;
(2) using System;
(3) using System.Windows.Forms;
(4) using System.Xml;
(5) using System.Xml.Xsl;
(6) using System.Xml.XPath;
(7) namespace XmlExample
(8) {
(9)     class Program
(10)    {
(11)        private static String _validatingError;
(12)        static void Main(string[] args)
(13)        {
(14)            String xmlFile = @"C:\literatur.xml", xsdFile =
(15)                @"C:\example.xsd", xslFile = @"C:\stylesheet.xsl",
(16)            htmlFile = @"C:\output.html";
(17)
(18)            //XML-Datei erzeugen
(19)            CreateXmlDoc(xmlFile);
(20)
(21)            //XML-Datei validieren
(22)            if (IsXmlValidate(xsdFile, xmlFile) == false)
(23)            {
(24)                MessageBox.Show("Die Datei ist nicht gültig" +
(25)                    "\n\nFehlermeldung: " + _validatingError);
(26)            }
(27)            else
(28)            {
(29)                //HTML-Datei erzeugen
(30)                Transform(xmlFile, xslFile, htmlFile);
(31)            }
(32)        } //end of Main
(33)    } //end of class
(34) } //end of namespace
```

---

### *Quelltext 13: Namensräume und Aufbau der Main-Methode*

Im Quellcodebeispiel 13 sind alle für die Arbeit mit XML-Daten benötigten Namensräume und einige Methodenaufrufe zu sehen. Die Methoden befinden sich ebenfalls in dieser Datei und werden nach und nach vorgestellt. Aus dem Beispiel wurden diese erstmal entfernt, damit es übersichtlicher bleibt. In Zeile 19 wird die Methode `CreateXmlDoc` aufgerufen, die das Erstellen der XML-Datei übernimmt. Nachdem die Datei erstellt wurde, wird sie gegen ein XML-Schema validiert (Zeile 22). Ist die Validierung erfolgreich, so erfolgt eine XSL-Transformation, die entsprechende Methode wird in Zeile 30 aufgerufen. Sollte die Validierung der XML-Datei fehlschlagen, wird in Zeile 24 eine entsprechende Meldung ausgegeben.

---

```

(1) private static void CreateXmlDoc(String fileName)
(2) {
(3)     XmlDocument doc = new XmlDocument();
(4)     //XML-Deklaration erstellen
(5)     XmlNode node = doc.CreateNode(XmlNodeType.XmlDeclaration, "", "");
(6)     doc.AppendChild(node);
(7)     //Root-Element erstellen
(8)     XmlElement rootElement = doc.CreateElement("Literatur");
(9)     doc.AppendChild(rootElement);
(10)    //erstes Element erzeugen
(11)    XmlElement nodeElement = doc.CreateElement("Buch");
(12)    nodeElement.SetAttribute("Titel", "User Interface Design");
(13)    nodeElement.SetAttribute("Autor", "Ben Shneiderman");
(14)    XmlText xmlText = doc.CreateTextNode(
(15)        "Effektive Interaktion zwischen Mensch und Maschine");
(16)    nodeElement.AppendChild(xmlText);
(17)    rootElement.AppendChild(nodeElement);
(18)    nodeElement = doc.CreateElement("Buch"); //zweites Element erzeugen
(19)    nodeElement.SetAttribute("Titel", "C#");
(20)    nodeElement.SetAttribute("Autor", "Helmut Erlenkötter");
(21)    xmlText = doc.CreateTextNode("Universell programmieren " +
(22)        "von Anfang an");
(23)    nodeElement.AppendChild(xmlText);
(24)    rootElement.AppendChild(nodeElement);
(25)    try {
(26)        doc.Save(fileName); //Xml-Datei speichern
(27)    }
(28)    catch (Exception e) {
(29)        MessageBox.Show("Fehler beim erstellen der XML-Datei." +
(30)            "\n\nFehlermeldung: " + e.Message, "XML-Beispiel")
(31)    }
(32) } //end of CreateXmlDoc

```

---

**Quelltext 14: Methode CreateXmlDoc zum Erstellen einer XML-Datei mit C#**

Das Quellcodebeispiel 14 zeigt, wie eine einfache XML-Datei mit Hilfe von C# erstellt werden kann. Das Root-Element ist in dem Beispiel Literatur (Zeile 8) und besitzt 2 Elemente (Zeile 11 und 18). Jedes Element wiederum besitzt zwei Attribute und einen Text. Die erzeugte XML-Datei sieht wie folgt aus:

---

```

(1) <?xml version="1.0"?>
(2) <Literatur>
(3)     <Buch Titel="User Interface Design" Autor="Ben Shneiderman">
(4)         Effektive Interaktion zwischen Mensch und Maschine
(5)     </Buch>
(6)     <Buch Titel="C#" Autor="Helmut Erlenkötter">
(7)         Universell programmieren von Anfang an
(8)     </Buch>
(9) </Literatur>

```

---

**Quelltext 15: Erstellte XML-Datei literatur.xml**

Das XML-Schema, gegen das die XML-Datei aus Quellcodebeispiel 15 validiert werden soll, wird nachfolgend definiert:

---

```
(1) <?xml version="1.0" encoding="UTF-8"?>
(2) <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
(3)   <xs:element name="Literatur">
(4)     <xs:complexType>
(5)       <xs:sequence>
(6)         <xs:element name="Buch" maxOccurs="unbounded">
(7)           <xs:complexType>
(8)             <xs:simpleContent>
(9)               <xs:extension base="xs:string">
(10)                <xs:attribute name="Titel" use="required"/>
(11)                <xs:attribute name="Autor" use="required"/>
(12)              </xs:extension>
(13)            </xs:simpleContent>
(14)          </xs:complexType>
(15)        </xs:element>
(16)      </xs:sequence>
(17)    </xs:complexType>
(18)  </xs:element>
(19) </xs:schema>
```

---

#### *Quelltext 16: XML-Schema example.xsd*

Die Validierung der XML-Datei gegen das XML-Schema aus Quellcodebeispiel 16 erfolgt durch die Methode `IsXmlValidate`. Sofern die XML-Datei gültig ist, liefert die Methode `true` zurück. Ansonsten wird `false` zurückgegeben und die Fehlermeldung in die Variable `_validatingError` geschrieben.

---

```
(1) private static Boolean IsXmlValidate(String xsdFile, String xmlFile)
(2) {
(3)   XmlDocument doc = null;
(4)   try {
(5)     doc = new XmlDocument();
(6)     doc.Schemas.Add(null, xsdFile); //Schema hinzufügen
(7)     doc.Load(xmlFile); //XML-Datei laden
(8)     doc.Validate(null); //XML-Datei validieren
(9)   }
(10)  catch (Exception e) {
(11)    _validatingError += e.Message;
(12)    return false;
(13)  }
(14)  return true;
(15) } //end of IsXmlValidate
```

---

#### *Quelltext 17: Methode IsXmlValidate zum validieren einer XML-Datei*

In dem Quellcodebeispiel 17 wird anhand eines kurzen Beispiels demonstriert, wie eine XML-Datei gegen ein XML-Schema mit Hilfe von C# validiert werden kann. Der entscheidende Aufruf passiert dabei in Zeile 8: das erstellte XML-Dokument wird gegen die angegebenen XML-Schemata validiert. Bei einem Fehler wird eine `Exception`-

tion geworfen, die in Zeile 10 abgefangen und dessen Inhalt in Zeile 11 der Variablen `_validatingError` zugewiesen wird. Nachdem die XML-Datei auf ihre Gültigkeit überprüft wurde, kann sie transformiert werden. Dafür ist die folgende XSL-Datei zuständig:

---

```

(1) <?xml version="1.0" encoding="UTF-8" ?>
(2) <xsl:stylesheet exclude-result-prefixes="xs xdt err fn"
(3)     version="2.0" xmlns:err="http://www.w3.org/2005/xqt-errors"
(4)     xmlns:fn="http://www.w3.org/2005/xpath-functions"
(5)     xmlns:xdt="http://www.w3.org/2005/xpath-datatypes"
(6)     xmlns:xs="http://www.w3.org/2001/XMLSchema"
(7)     xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
(8) <xsl:output indent="yes" method="html"/>
(9) <xsl:template match="/">
(10)   <html>
(11)     <head>
(12)       <meta http-equiv="Content-Type"
(13)         content="text/html; charset=UTF-8" />
(14)       <title>XML-Beispiel</title>
(15)     </head>
(16)     <body>
(17)       <center>
(18)         <table width="50%" border="1" cellpadding="3"
(19)           cellspacing="3">
(20)           <tbody>
(21)             <tr bgcolor="gray"><th>Buchname</th>
(22)               <th>Autor</th><th>Beschreibung</th>
(23)             </tr>
(24)             <xsl:for-each select="//Literatur/Buch">
(25)               <tr>
(26)                 <td><xsl:value-of select="@Titel"/></td>
(27)                 <td><xsl:value-of select="@Autor"/></td>
(28)                 <td><xsl:value-of select="." /></td>
(29)               </tr>
(30)             </xsl:for-each>
(31)           </tbody>
(32)         </table>
(33)       </center>
(34)     </body>
(35)   </html>
(36) </xsl:template>
(37) </xsl:stylesheet>

```

---

#### **Quelltext 18: XSL-Datei *stylesheet.xsl***

Anhand der XSL-Datei wird eine HTML-Datei generiert. Die dazu benötigte Methode heißt `Transform` und wird anhand eines weiteren Quellcodebeispiels dargestellt:

```

(1) public static void Transform(String xmlPath, String xslPath,
(2)                               String target)
(3) {
(4)     XmlTextWriter writer = null;
(5)     try {
(6)         XslCompiledTransform xslTransform = new XslCompiledTransform();
(7)         //XSL-Datei laden
(8)         xslTransform.Load(xslPath);
(9)         //Output-Stream erstellen
(10)        writer = new XmlTextWriter(target, null);
(11)        XPathDocument doc = new XPathDocument(xmlPath);
(12)        //Transformieren
(13)        xslTransform.Transform(doc, null, writer);
(14)        writer.Close();
(15)    }
(16)    catch (Exception e) {
(17)        MessageBox.Show("Bei der Transformierung ist ein " +
(18)                        "Fehler aufgetreten.\n\nFehlermeldung: " +
(19)                        e.Message, "XML-Beispiel");
(20)    }
(21) } //end of Transform

```

### Quelltext 19: Methode Transform zum Anwenden der XSL-Datei

Der Methode in dem Quellcodebeispiel 19 werden der Pfad zu der vorhandenen XML-Datei und der XSL-Datei übergeben. In Zeile 13 kommt die XSL-Datei zum Einsatz und die HTML-Datei wird erzeugt und an der in Zeile 10 angegebenen Stelle (durch die Variable `target`) gespeichert. Nach Beendigung der Transformation kann die HTML-Datei in einem Webbrowser angesehen werden, was in folgender Abbildung zu sehen ist:

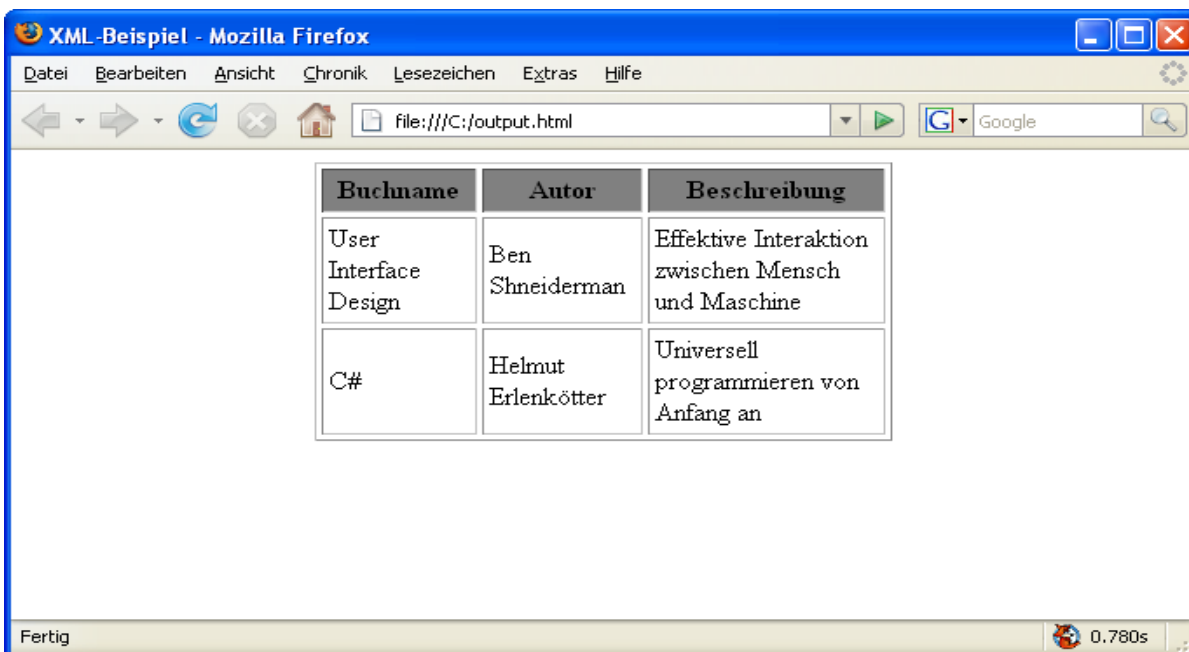


Abbildung 3: Ausgabe nach XSL-Transformation

### **3 KONZEPT**

In diesem Kapitel wird das Konzept für die zu entwickelnde Anwendung aufgestellt. Das Konzept beginnt mit einer Analyse, in der eine Evaluation über die typischen Bestandteile eines Lebenslaufes vollzogen wird. Ferner werden mögliche Standards zur Speicherung der ermittelten Daten untersucht werden. Im Anschluss d wird ein Lösungsansatz präsentiert, mit dem die genannten Ziele der Diplomarbeit erreicht werden können. Aufbauend darauf wird ein Design für die Benutzerschnittstelle vorgestellt und die Architektur des Programms schrittweise definiert und erläutert.

#### **3.1 Analyse**

Die Analyse umfasst eine Evaluierung der Bestandteile eines Lebenslaufes [BEW07] und eine Untersuchung von möglichen und geeigneten Standards zur Speicherung der Personendaten. Danach werden bereits existierende Softwarelösungen zum Erstellen von Lebensläufen dahingehend untersucht, ob sie die Bestandteile eines Lebenslaufes abdecken, einen Standard zur Speicherung unterstützen und ob sie komfortabel zu bedienen sind.

Die Evaluierung der Bestandteile eines Lebenslaufes soll zeigen, auf was beim Erstellen der Personendaten geachtet werden muss. Es wird auf das am 18. August 2006 in Kraft getretene Gesetz zur allgemeinen Gleichbehandlung eingegangen und erwähnt, welche Angaben in einem Lebenslauf dadurch nur als optional angesehen werden dürfen. Zu manchen Punkten innerhalb des Lebenslaufes gibt es bereits Anmerkungen, die bei der späteren Entwicklung berücksichtigt werden müssen. Darauf aufbauend werden die gängigsten Standards zur Datenhaltung auf deren Abdeckung der Lebenslaufdaten hin untersucht und es wird der für die Umsetzung der Ziele am besten geeignetste Standard ausgewählt, den das zu entwickelnde Programm später unterstützen soll.

### 3.1.1 Evaluierung über Bestandteile eines Lebenslaufes

Es gibt relevante und weniger relevante Angaben, die ein Bewerber machen kann oder sollte [UNI07]. Diese Angaben können in verschiedenen Kategorien eingeteilt werden. Im Folgenden werden alle in Frage kommenden Elemente aufgezählt, wobei zu beachten ist, dass einige Elemente als optional angesehen werden können. Diese Elemente sind mit einem O für optional gekennzeichnet [STE07]. Ferner sind die Elemente des Lebenslaufes mit abstrakten Datentypen gekennzeichnet. Mögliche Typen sind hierbei Text, Datum und Auswahlmenü.

Zu dem Typ Datum ist zu beachten, dass dem Benutzer der zu entwickelnden Software bei der Eingabe von Zeitangaben ein Kalender zur Verfügung gestellt werden sollte, damit das Datum komfortabel ausgewählt werden kann.

#### 3.1.1.1 Persönliche Angaben

- Vorname : Text
- Name : Text
- Künstlername (O) : Text
- Titel / Akademischer Grad (O) : Auswahlmenü und Text (beispielsweise Dr.)
- Geschlecht (O) : Auswahlmenü (männlich, weiblich)
- Geburtsdatum : Datum
- Geburtsort : Text
- Geburtsland : Auswahlmenü
- Nationalität (O) : Auswahlmenü
- Familienstand (O) : Auswahlmenü (ledig, verheiratet)
- Religion (O) : Text
- Führerscheinklassen (O) : Text

Beim optionalen Feld „Familienstand“ sollte dem Benutzer eine Auswahl an möglichen Eingaben angeboten werden. Das Feld „Nationalität“ sollte auch aus einem Auswahlmenü bestehen. Da es durchaus stetige Veränderungen in den Strukturen der

Nationen gibt, sollte diese Liste daher dynamisch geladen werden. Zu untersuchen ist hier, welche Möglichkeiten dabei die Globalisierungsfunktionen innerhalb des verwendeten .NET Frameworks bieten und ob alle Länder dort bereits eingetragen sind und für die Liste verwendet werden können.

### 3.1.1.2 Kontaktdaten

- Telefon- und Faxnummern (O) : Text
- Postleitzahl (O) : Text
- Ort (O) : Text
- Strasse und Hausnummer (O) : Text
- Postfach (O) : Text
- Land (O) : Auswahlmenü
- E-Mail-Adressen (Privat, Geschäftlich, beides O) : Text
- URL einer privaten Homepage (O) : Text

„E-Mail-Adressen“ und „URL einer privaten Homepage“ sollten auf Gültigkeit hin überprüft werden, sofern sie eingetragen wurden.

### 3.1.1.3 Schulbildung

- Name der Schule : Text
- Postleitzahl (O) : Text
- Ort : Text
- Land : Auswahlmenü
- Zeitraum (von und bis) : Datum
- Schulart : Auswahlmenü und Text
- Abschluss : Auswahlmenü und Text (Realschulabschluss, Abitur)
- Abschlussnote : Text
- Notenskala : Auswahlmenü (sehr gut, gut)

Unter dem Punkt „Schulart“ kann die Art der Schule angegeben werden, beispielsweise Grundschule, Realschule, Berufsschule etc.

Die Abschlussnote sollte frei von Konventionen sein, da es viele Unterschiede in der Art der Benotung gibt. So ist in Deutschland die beste Note eine 1, in anderen Ländern allerdings eine 6 oder gar etwas Alphanumerisches. Um dennoch automatisiert Auswertungen durchführen zu können, kann neben der Note noch eine Bewertung auf einer vordefinierten „Notenskala“ angegeben werden. Die Skala reicht von „sehr gut“ bis „ausreichend“.

#### **3.1.1.4 Wehr- oder Zivildienst / Freiwilliges soziales Jahr**

- Art der Tätigkeit : Auswahlmenü (Wehrdienst, Zivildienst)
- Organisation / Firma / Dienst Einheit : Text
- Postleitzahl : Text
- Ort der Tätigkeit : Text
- Beschreibung : Text
- Zeitraum (von und bis) : Datum

Es ist zu untersuchen, ob eine Tätigkeit in einem freiwilligen sozialen Jahr vielleicht doch als praktische Erfahrung erfasst werden sollte.

#### **3.1.1.5 Studium**

- Hochschulname : Text
- Postleitzahl : Text
- Ort : Text
- Land : Auswahlmenü
- Studiengang : Auswahlmenü (Uni, FH, TU)
- Abschluss : Auswahlmenü (Diplom, Master, Dr.)
- Thema der Abschlussarbeit : Text
- Note der Abschlussarbeit (O, nur bei einem Abschluss) : Text und Notenskala
- Abschlussdatum : Datum
- Abschlussnote : Text und Notenskala
- Studienschwerpunkte : Text
- Zeitraum (von und bis) : Datum
- Engagement im Studenumfeld : Text

Für ein Studium können zwei Noten angegeben werden, zum einen die Benotung der Abschlussarbeit und zum anderen die Abschlussnote. Studienschwerpunkte können angegeben und beschrieben werden.

#### **3.1.1.6 Praktische Erfahrungen**

- Firma : Text
- Postleitzahl : Text
- Ort : Text
- Land : Auswahlmenü
- Branche : Text
- Arbeitsschwerpunkte : Text
- Zeitraum (von und bis) : Datum
- Berufsart : Auswahlmenü

Zu der „Berufsart“ zählen Ausbildung, Praktika, Studienbegleitende Tätigkeit, Festanstellung, Freiberufliche Tätigkeit und sonstige Arten.

#### **3.1.1.7 Sprachkenntnisse**

- Sprache : Auswahlmenü
- Einschätzung : Auswahlmenü

Die Einschätzung der Sprache sollte in Form eines Auswahlfeldes geschehen, zum Beispiel mit einem Bereich von „Grundkenntnisse“ bis „Muttersprache“.

#### **3.1.1.8 IT-Kenntnisse**

- IT-Bereich : Auswahlmenü
- Einschätzung : Auswahlmenü
- Beschreibung : Text

Für einen Benutzer mit nur wenigen IT-Kenntnissen könnte eine Auflistung mit allen möglichen IT-Kenntnissen sehr unüberschaubar und kompliziert wirken. Von daher sollte hier nur eine Teilmenge berücksichtigt und benutzerfreundlich dargestellt werden. Die verfügbaren IT-Kenntnisse sind über das Auswahlmenü „IT-Bereich“ auszuwählen.

Zu den IT-Bereichen gehören:

Office-Anwendungen (Textverarbeitungssoftware, Tabellenkalkulationssoftware, E-Mail-Clients, Web-Browser, etc.), Programmiersprachen, Entwicklungsumgebungen, Betriebssysteme, Grafik-Software, betriebswirtschaftliche Anwendungen, Konstruktions-Software und sonstige IT-Kenntnisse. Zu den ausgewählten Kenntnissen sind jeweils noch eine Einschätzung und eine Beschreibung der genauen Kenntnisse anzugeben.

### **3.1.1.9 Weiterbildung**

- Art der Weiterbildung : Auswahlmenü
- Organisation : Text
- Postleitzahl : Text
- Ort : Text
- erreichte Zertifikate / Urkunden / Abschlüsse (O) : Text
- Zeitraum : Datum

### **3.1.1.10 Hobbies und Interessen**

- Art : Auswahlmenü
- Beschreibung : Text

Hobbies und Interessen sollten nur angegeben werden, wenn diese für die Stelle relevant sind.

### **3.1.1.11 Gesellschaftliches Engagement / Ehrenamtliche Aktivitäten**

- Art : Auswahlmenü
- Titel : Text
- Beschreibung : Text

### **3.1.1.12 Publikationen**

- Art der Publikation (Buch, Artikel, Vortrag, Poster) : Auswahlmenü
- Titel : Text
- Jahr : Datum
- Verlag (O, sofern vorhanden) : Text
- Weitere Autoren (O, nur wenn es sich um mehrere Autoren handelt) : Text

- ISBN (O, sofern vorhanden) : Text
- Beschreibung : Text

### **3.1.2 Allgemeines Gleichbehandlungsgesetz**

Bei der Verarbeitung bewerbungsrelevanter Informationen in einem Lebenslauf ist darauf zu achten, dass das allgemeine Gleichbehandlungsgesetz [AGG06] berücksichtigt wird. Das Gesetz soll verhindern, dass Benachteiligungen aus Gründen der Rasse, der ethnischen Herkunft, des Geschlechts, der Religion, der Weltanschauung, einer Behinderung, des Alters oder der sexuellen Identität entstehen. Das Gesetz ist am 18. August 2006 in Kraft getreten und seitdem gültig.

Bei der Erstellung eines Lebenslaufes ist daher darauf zu achten, dass die betroffenen Felder nur optional sind und bei Bedarf auch leer gelassen werden können. Der Benutzer muss eine Wahlmöglichkeit haben, welche Informationen er freigibt. Zu diesen Daten zählen das Alter, die Nationalität, die Religion und der Familienstand. Außerdem ist es seit in Kraft treten dieses Gesetzes dem Bewerber überlassen, ob dem Lebenslauf ein Passfoto hinzugefügt wird oder nicht. Benachteiligungen gegenüber des Bewerbers dürfen davon nicht ausgehen.

### 3.1.3 Untersuchung von möglichen Standards zur Datenhaltung

Hinsichtlich der Abdeckung der zu einem Lebenslauf gehörenden Daten, werden in dieser Untersuchung einige Standards zur Speicherung von Daten genauer betrachtet.

Ein bereits weit verbreitetes Format zur Speicherung von Personendaten ist das HR-XML-Format. Es soll ein Überblick über dieses Format und über Formate, die auf dem HR-XML-Format aufbauen, gegeben werden. Ein solches auf HR-XML aufbauende Format wurde von der Bundesagentur für Arbeit<sup>1</sup> entwickelt [*HR-BA-XML*]. Ein weiteres Format ist das German Standard Curriculum Vitae (GSCV), welches ebenfalls auf HR-XML aufbaut und als Standard für den deutschen Markt gedacht ist. GSCV hat sich als Ziel gesetzt, den Prozess der Onlinebewerbung für den Bewerber zu vereinfachen, damit sich ein Bewerber auf die wesentlichen Elemente einer Bewerbung konzentrieren kann.

Außerhalb von Deutschland ist iProfile beispielsweise anzutreffen. In Großbritannien hat sich iProfile etabliert und wird bereits von vielen recruiting Agencies, aber auch von Unternehmen genutzt. Auch iProfile nutzt als Grundlage HR-XML.

Im weiteren Verlauf wird auf jedes dieser XML-Formate näher eingegangen.

#### 3.1.3.1 HR-XML

Das XR-XML-Konsortium ist eine unabhängige, nicht profitorientierte Organisation, die Standards, basierend auf XML, für offenen Datenaustausch für den Bereich Personalwesen (engl.: „Human Resources“, kurz: HR) entwickelt und veröffentlicht. In Europa wird HR-XML durch die Schwestergesellschaft HR-XML Consortium Europe<sup>2</sup> repräsentiert.

Das HR-XML-Konsortium verfolgt das Ziel, Unternehmen die Mittel zur Verfügung zu stellen, um einen einheitlichen Austausch für HR-relevante Daten zu ermöglichen. Viele bekannte Unternehmen sind bereits Mitglied des HR-XML-Konsortiums und helfen dabei, den Standard weiter zu entwickeln. Zu den über achtzig Mitgliedern gehören u. a. die Bundesagentur für Arbeit, Cisco Systems, EDS, IBM, Microsoft, milch & zucker AG, Monster.com, Oracle und SAP.

<sup>1</sup> <http://www.arbeitsagentur.de> (Stand: 22.01.2008)

<sup>2</sup> <http://eu.hr-xml.org> (Stand: 22.01.2008)

Das HR-XML-Konsortium bietet eine Vielzahl von XML-Dateien und XML-Schemata an, die für die Erstellung von Lebensläufen eingesetzt werden können. Die einzelnen XML-Schemata treten dabei als Module auf, wodurch es ermöglicht wird, ein spezielles XML-Dokument aus validierten HR-XML-Modulen zusammenzusetzen. Dies kann zum Beispiel für die Erstellung eines Lebenslaufes eingesetzt werden. Neben Lebensläufen sind auch die Erstellung von Rechnungen, Speicherung von Anstellungsdaten und Bezügen, Human Resource Informationen wie Kompetenzen, Verfügbarkeiten und individuelle Erfahrungen und Stellenausschreibungen mit HR-XML möglich.

Auf der Internetseite von HR-XML können die XML-Schema-Dateien und -Beispiele dazu als ZIP-Archiv<sup>1</sup> [MAT07] heruntergeladen werden. Beim Auspacken des Archivs werden die dort enthaltenen Dateien in den Ordner „\HRXML200704“ kopiert. In dem Ordner befindet sich der Unterordner „\HR-XML-2\_5“, welcher die XML-Schema-Dateien und die Beispiele zur Anwendung der Dateien beinhaltet.

Das folgende Quellcodebeispiel 20 stammt aus der HR-XML-Spezifikation und zeigt, wie ein Lebenslauf mit Einsatz von HR-XML aufgebaut sein könnte. Es handelt sich dabei um die Datei ResumeExample.xml aus dem Ordner „\HRXML200704\HR-XML-2\_5\SEP“.

SEP steht dabei für „Staffing Exchange Protocol“ und ist eine Ansammlung von XML-Spezifikationen, mit denen Anwerbe- und Einstellverfahren (engl.: „staffing“) ermöglicht werden. In dem folgenden Quellcodebeispiel wird der Aufbau eines Lebenslaufes in einer HR-XML konformen XML-Datei gezeigt:

---

<sup>1</sup> Aktuelle Version ist HR-XML 2.5 vom 15.04.2007 (Stand: 22.01.2008)

---

```

(1) <?xml version="1.0" encoding="UTF-8"?>
(2) <Resume xml:lang="EN" xmlns="http://ns.hr-xml.org/2007-04-15"
(3)     xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)     xsi:schemaLocation="http://ns.hr-xml.org/2007-04-15
(5)     Resume.xsd">
(6)   <ResumeId><IdValue>JA2284</IdValue></ResumeId>
(7)   <StructuredXMLResume>
(8)     <ContactInfo>
(9)       <PersonName>
(10)        <FormattedName>John A. Example</FormattedName>
(11)      </PersonName>
(12)      <ContactMethod>
(13)        <Telephone>
(14)          <FormattedNumber>+1 404 122 1234</FormattedNumber>
(15)        </Telephone>
(16)        <Fax><FormattedNumber>+1 404 123 1234</FormattedNumber>
(17)        </Fax>
(18)        <InternetEmailAddress>john@example.com</InternetEmailAddress>
(19)        <InternetWebAddress>http://www.example.com
(20)        </InternetWebAddress>
(21)        <PostalAddress>
(22)          <CountryCode>US</CountryCode>
(23)          <PostalCode>12345-1234</PostalCode>
(24)          <Region>GA</Region>
(25)          <Municipality>Atlanta</Municipality>
(26)          <Recipient>
(27)            <OrganizationName>College of Computing, Georgia
(28)              Institute of Technology</OrganizationName>
(29)          </Recipient>
(30)        </PostalAddress>
(31)      </ContactMethod>
(32)    </ContactInfo>
(33)    <EmploymentHistory>
(34)      <EmployerOrg>
(35)        <EmployerOrgName>IBM</EmployerOrgName>
(36)        <EmployerContactInfo>
(37)          <LocationSummary>
(38)            <Municipality>Yorktown Heights</Municipality>
(39)            <Region>NY</Region>
(40)          </LocationSummary>
(41)        </EmployerContactInfo>
(42)        <PositionHistory positionType="internship">
(43)          <OrgName>
(44)            <OrganizationName>T.J. Watson Research Center
(45)            </OrganizationName>
(46)          </OrgName>
(47)          <Description>Job - Discription</Description>
(48)          <StartDate><YearMonth>1998-06</YearMonth>
(49)          </StartDate>
(50)          <EndDate>
(51)            <YearMonth>1998-08</YearMonth>
(52)          </EndDate>
(53)        </PositionHistory>
(54)      </EmployerOrg>
(55)    </EmploymentHistory>

```

---

---

```
(56) <!-- An dieser Stelle würden noch Schulausbildung, weitere Berufs-
(57) erfahrungen usw stehen. Aus Platzgründen ist die Datei hier gekürzt
(58) -->
(59) </StructuredXMLResume>
(60) </Resume>
```

---

### *Quelltext 20: Beispiellebenslauf als HR-XML-Datei<sup>1</sup>*

Im obigen Beispiel werden Daten zu einem Lebenslauf innerhalb einer XML-Datei gespeichert. Die Datei wird anhand eines XML-Schematas validiert. Die dafür benötigte XML-Schema-Datei heißt „Resume.xsd“ und wird im späteren Verlauf noch genauer vorgestellt.

Damit sich das Beispiel nicht über mehrere Seiten erstreckt, wurden die Schulbildung, Berufserfahrung, Sprachkenntnisse und sonstige Angaben heraus genommen. Die prinzipielle Einsatzmöglichkeit von HR-XML als Standard für Lebensläufe kann mit dem vorliegenden Beispiel dennoch gut demonstriert werden. Zu sehen sind in dem Beispiel die persönlichen Daten wie Name, Anschrift, Telefonnummer, Internetadresse, E-Mail-Adresse usw. Der Name wird in Zeile 10 als zusammenhängender Name angegeben. Es gibt auch ein XML-Schemata, mit dem Vorname und Nachname getrennt angegeben werden kann. Die Telefonnummer steht in Zeile 14, in Zeile 18 wird die E-Mail-Adresse angegeben. Mit dem Tag `EmploymentHistory` in Zeile 33 wird die Berufserfahrung eingeleitet und umfasst eine Anstellung bei IBM inkl. des Zeitraums der Beschäftigung, einer Stellenbeschreibung und weiteren, für die Berufserfahrung relevanten Daten.

In dem Ordner `SEP` gibt es auch noch die XML-Schema-Datei mit dem Namen „Candidate.xsd“. Innerhalb dieser Datei werden verschiedene XML-Schemata eingebunden, unter anderem die bereits erwähnte Resume.xsd. Somit kann anhand von verschiedenen Modulen, wobei die Resume.xsd hierbei ein Modul darstellt, eine komplexe Struktur, zum Beispiel zum Erstellen von Lebensläufen, geschaffen werden. Das Einbinden von verschiedenen XML-Schemata erfolgt durch folgenden Aufruf:

---

```
<xsd:include schemaLocation="../../../SEP/Resume.xsd"/>
```

---

Durch den `Include`-Tag kann ein XML-Schema geladen und in einem XML-Schema verwendet werden.

---

<sup>1</sup> Quelle: [http://ns.hr-xml.org/2\\_5/HR-XML-2\\_5/SEP/ResumeExample.xml](http://ns.hr-xml.org/2_5/HR-XML-2_5/SEP/ResumeExample.xml) (Stand: 22.01.2008)

---

```

(1) <?xml version="1.0" encoding="UTF-8"?>
(2) <xsd:schema xmlns="http://ns.hr-xml.org/2007-04-15"
(3)   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
(4)   targetNamespace="http://ns.hr-xml.org/2007-04-15"
(5)   elementFormDefault="qualified" version="2007-04-15">
(6)   <xsd:annotation>
(7)     <xsd:documentation>...</xsd:documentation>
(8)   </xsd:annotation>
(9)   <xsd:include schemaLocation="../CPO/ContactMethod.xsd"/>
(10)  <xsd:include schemaLocation="../CPO/SharedStaffingModules.xsd"/>
(11)  <xsd:include schemaLocation="../CPO/MilitaryHistory.xsd"/>
(12)  <xsd:include schemaLocation="../CPO/EducationHistory.xsd"/>
(13)  <xsd:include schemaLocation="../SEP/Resume.xsd"/>
(14)  <xsd:include schemaLocation="../CPO/EmploymentHistory.xsd"/>
(15)  <xsd:include schemaLocation="../CPO/OnlineAddress.xsd"/>
(16)  <xsd:include schemaLocation="../CPO/DateTimeDataTypes.xsd"/>
(17)  <xsd:include schemaLocation="../CPO/IdentifierTypes.xsd"/>
(18)  <xsd:include schemaLocation="../CPO/PersonDescriptors.xsd"/>
(19)  <xsd:include schemaLocation="../CPO/Organization.xsd"/>
(20)  <xsd:include schemaLocation="../SEP/MatchingTypes.xsd"/>
(21)  <xsd:include schemaLocation="../CPO/UserArea.xsd"/>
(22)  <xsd:include schemaLocation="../SEP/SearchTypes.xsd"/>
(23)  <xsd:import namespace="http://www.w3.org/XML/1998/namespace"
(24)    schemaLocation="../../W3C/xml.xsd"/>
(25)  <!-- Profile Type -->
(26)  ...
(27) </xsd:schema>

```

---

#### **Quelltext 21: Quelltext von XML-Schema Candidate.xsd<sup>1</sup>**

Das Quellcodebeispiel 21 zeigt die Datei Candidate.xsd. Aus Gründen der Übersicht ist diese Datei wieder etwas gekürzt. Es ist dennoch sehr gut zu sehen, wie mehrere XML-Schemata innerhalb eines XML-Schemas mit Hilfe des `include`-Tags eingebunden werden können.

Das HR-XML-Format bietet viele XML-Schema-Dateien an und da alle Dateien miteinander kombiniert werden können, stellt HR-XML ein sehr mächtiges, standardisiertes und modularisiertes XML-Format dar. Mit HR-XML ist es möglich, alle für einen Lebenslauf relevanten Daten innerhalb einer HR-XML validierten Datei zu speichern. Allerdings werden einige Bereiche nicht optimal abgedeckt. So gibt es zum Beispiel keine XML-Elemente, um Daten für einen geleisteten Zivildienst zu speichern. Dies könnte umgangen werden, wenn Zivildienst als Militärausbildung gespeichert werden würde.

---

<sup>1</sup> Quelle: [http://ns.hr-xml.org/2\\_5/HR-XML-2\\_5/SEP/Candidate.xsd](http://ns.hr-xml.org/2_5/HR-XML-2_5/SEP/Candidate.xsd) (Stand: 22.01.2008)

HR-XML bietet kein XML-Element an, mit dem binäre Daten, wie zum Beispiel ein Bewerberfoto, innerhalb einer XML-Datei gespeichert werden könnten. Ein Bild müsste also separat übermittelt werden, wenn HR-XML als Standard für einen Lebenslauf benutzt wird.

### 3.1.3.2 HR-BA-XML

Die Bundesagentur für Arbeit [BAA] benötigte einen Standard, um mit Arbeitgebern, Jobbörsen und anderen Arbeitsvermittlungsbeteiligten kommunizieren zu können. Aus diesem Grund hat sich die Bundesagentur für Arbeit dafür entschieden, ein auf HR-XML aufbauendes XML-Format zu entwerfen, das für die Stellen- und Bewerberinformationen eingesetzt werden kann. Mit diesem Format kann eine Kommunikation zwischen zwei Softwaresystemen stattfinden [BUN07].

Durch den dazu entworfenen HR-BA-XML Standard erfolgt der Austausch von Job- und/oder Bewerberdaten zwischen der Bundesagentur für Arbeit und externen Kooperationspartnern. Die Bundesagentur für Arbeit strebt dabei eine enge Kooperation mit den Herstellern von HR-Software an, damit die Verbreitung und Akzeptanz des Standards weiter unterstützt wird. Von Seiten der Bundesagentur für Arbeit gibt es auch Pläne, den HR-BA-XML Standard mittelfristig in den HR-XML Standard aufnehmen zu lassen.

HR-BA-XML gliedert sich in drei Grundelemente:

- den „Job Position Seeker“ (JPS),
- dem „Job Position Posting“ (JPP) und
- das Element „Errorinformation“.

In diesen Elementen spiegeln sich die funktionalen und technischen Anforderungen der Bundesagentur für Arbeit wieder. Zu dem JPS gehören Angaben zu persönlichen und demographischen Daten eines Bewerbers, Informationen zu Lebenslauf, Werdegang und Qualifikationen, sowie Angaben zum Stellengesuch und den gewünschten Arbeitsbedingungen.

Das Element JPP wird von Unternehmen benutzt, die über die Bundesagentur für Arbeit eine Stelle anbieten wollen. Es können Informationen und Details zum Unternehmen, ausführliche Informationen zum Stellenangebot und detaillierte Angaben zu dem gewünschten Bewerberprofil angegeben werden.

Das Grundelement Errorinformationen wird genutzt, um auf Fehler bei der Übermittlung der Daten an die zu reagieren. Es beinhaltet Antwortdatensätze für eventuelle Fehlerinformationen und eine Auflistung der Fehlercodes und die dazugehörigen Beschreibungen.

Das Staffing Exchange Protocol (SEP), ein Unterprotokoll des HR-XML Standards, wurde um einige Punkte erweitert, um die Grundelemente umsetzen zu können. Nachfolgend werden die für einen Lebenslauf notwendigen Ergänzungen gegenüber dem SEP (SEP = Staffing Exchange Protocol, siehe Kapitel 3.1.3.1) vorgestellt:

- Angaben zur Mobilität (Führerschein, Fahrzeug)
- Informationen über besuchte Unterrichtsfächer und dabei erzielten Noten
- Angaben zu Wehr- und Zivildienst
- Erweiterung der persönlichen Angaben, z.B. Mobilfunknummer
- Informationen zum Familienstand
- Angaben zu Kündigungsfristen und dem Status der Beschäftigung
- Angaben zu berufsvorbereitenden Bildungsmaßnahmen und durchgeführten Praktika

HR-BA-XML soll ständig an die funktionalen und technischen Anforderungen des Arbeitsmarktes angepasst. Dabei kommt eine flexible und weitestgehend automatisierte Verfahrensweise bei der Anbindung von externen Partnern zum Einsatz.

Das HR-BA-XML-Format besitzt als Wurzelement das Element `HRBAXML`. Als Unter-elemente gibt es einen `Header`- und einen `Data`-Teil. `Data` unterteilt sich in `JPS` und `JPP`. Das folgende Bild zeigt die Struktur von HR-BA-XML und stammt aus einem Dokument, das von der Bundesagentur für Arbeit zur Verfügung gestellt wurde [BUN07]:

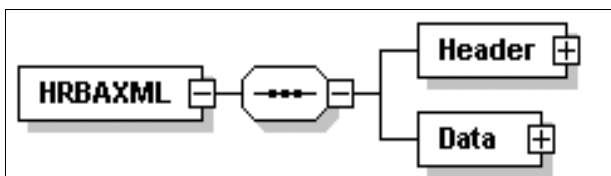


Abbildung 4: HR-BA-XML Struktur

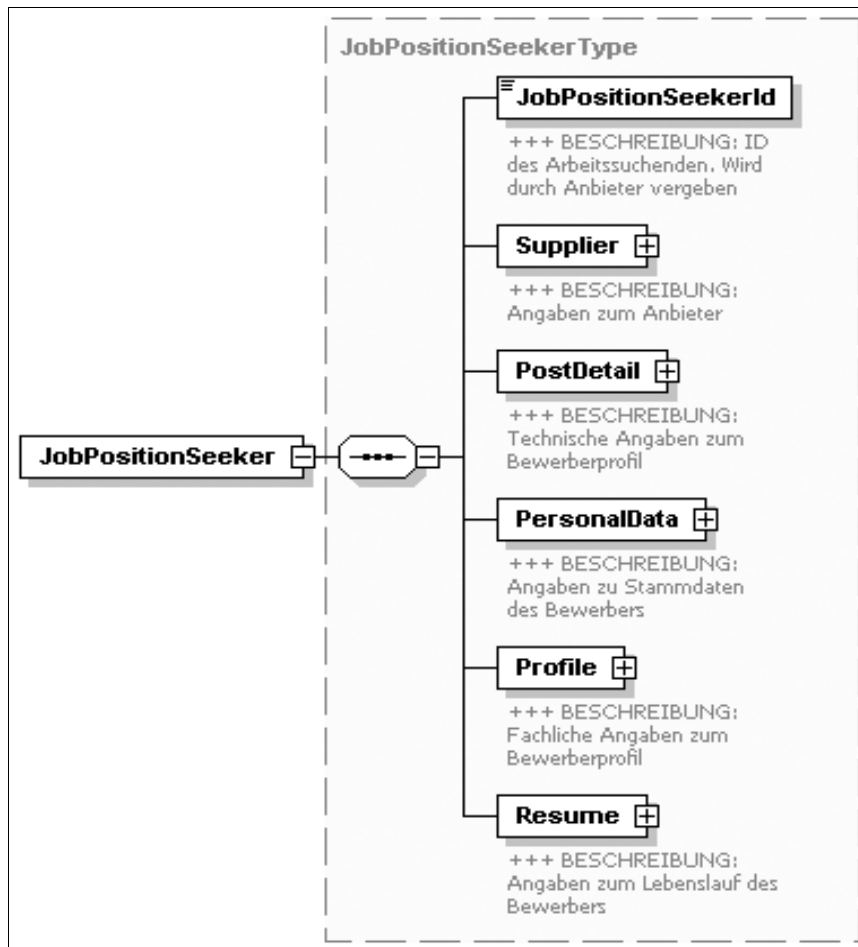


Abbildung 5: Elemente von `JobPositionSeekerType`

Abbildung 5 stammt erneut aus dem von der Bundesagentur für Arbeit zur Verfügung gestellten Dokument [BUN07] und zeigt den `JobPositionSeekerType` innerhalb des `Data`-Elements. Dieser Teil ist der für einen Lebenslauf relevante Teil des XML-Schemas. Innerhalb des Elements `HRBAXML` kann das Element `JobPositionSeekerType` beliebig oft vorkommen.

---

```

(1) <JobPositionSeeker />
(2) <PersonalData>
(3)   <Salutation>Herr</Salutation>
(4)   <Title/>
(5)   <GivenName>John A.</GivenName>
(6)   <NamePrefix/>
(7)   <FamilyName>Example</FamilyName>
(8)   <PostalAddress>
(9)   <CountryCode>US</CountryCode>
(10)  <PostalCode>12345-1234</PostalCode>
(11)  <Region>GA</Region>
(12)  <Municipality>Atlanta</Municipality>
(13) </PostalAddress>
(14)  <VoiceNumber>
(15)  <IntlCode>+1</IntlCode>
(16)  <AreaCode>404</AreaCode>
(17)  <TelNumber>122 1234</TelNumber>
(18) </VoiceNumber>
(19)  <Email>john@example.com</Email>
(20)  <Website>http://www.example.com</Website>
(21)  <DemographicDetail>
(22)  <DateOfBirth/>
(23)  <Nationality/>
(24) </DemographicDetail>
(25) </PersonalData>
(26) <Resume>
(27)  <StructuredResume>
(28)  <EmploymentExperience/>
(29)  <EmploymentHistory>
(30)  <Position>
(31)  <EmployerOrIndustry>IBM</EmployerOrIndustry>
(32)  <Description>Job - Description</Description>
(33)  <StartDate>1998-06</StartDate>
(34)  <EndDate>1998-08</EndDate>
(35)  </Position>
(36) </EmploymentHistory>
(37)  <EducationQualifs>
(38)  <EduDegree>Dipl-Inf</EduDegree>
(39)  <TypeOfSchool>Uni</TypeOfSchool>
(40)  <BranchOfStudy>Computer Science</BranchOfStudy>
(41) </EducationQualifs>
(42)  <SkillQualifs>
(43)  <Skill>
(44)  <SkillName>English</SkillName>
(45)  <SkillLevel>Native Language</SkillLevel>
(46)  </Skill>
(47) </SkillQualifs>
(48) </StructuredResume>
(49) </Resume>
(50) </JobPositionSeeker>

```

---

**Quelltext 22: Lebenslauf als HR-BA-XML-Datei<sup>1</sup>**

---

<sup>1</sup> Quelle: White Paper HR-BA-XML Standard [BUN07]

Das Quellcodebeispiel 22 zeigt eine XML-Datei, die als Grundlage die XML-Schema-Datei von HR-BA-XML heranzieht. Die XML-Datei beinhaltet Informationen über persönliche und demographische Bewerberdaten und Informationen zu Lebenslauf und Qualifikationen. Darüber hinaus sind auch noch weitere Angaben zu dem Stellensuch des Bewerbers vorhanden, was für einen Lebenslauf allerdings nicht relevant ist. Dieser Teil müsste von der zu entwickelnde Software nicht weiter beachtet werden.

Auf dem Internetportal der Bundesagentur für Arbeit stehen die verwendeten XML-Schema-Dateien nicht zum freien Herunterladen bereit. Doch in der Informationsbroschüre<sup>1</sup> ist eine E-Mail-Adresse enthalten, an die sich Interessenten der XML-Schema-Definition wenden können. Über diese E-Mail-Adresse können die XML-Schema-Dateien und eine ausführliche Dokumentation darüber angefordert werden. Dadurch war es möglich, das Format HR-BA-XML im Rahmen der Diplomarbeit zu untersuchen.

Wie auch bei HR-XML ist es bei HR-BA-XML nicht vorgesehen, binäre Dateien innerhalb einer XML-Datei zu speichern. Dadurch muss auch bei Verwendung von HR-BA-XML ein Bewerbungsfoto ggf. separat gespeichert und verarbeitet werden.

### **3.1.3.3 GSCV**

Das German Standard Curriculum Vitae (GSCV) ist ein Service der „milch & zucker – the marketing & software AG“. Es handelt sich dabei um eine von mehreren Unternehmen ins Leben gerufene Initiative, die sich als Ziel gesetzt hat, einen einheitlichen Standard für Lebensläufe in Deutschland zu etablieren. Unternehmen wie Lufthansa, Bosch, Commerzbank, Audi und Siemens haben gemeinsam eine auf HR-XML aufbauende Definition erarbeitet, die für den beruflichen Werdegang und weiteren Lebenslaufdaten die Abspeicherung der Daten standardisiert [GSCV].

Bei der Entwicklung von GSCV wurde darauf geachtet, auf bestehende und etablierte Standards und Spezifikationen zurückzugreifen. Im Gegensatz zum zugrunde liegenden HR-XML wurden nicht nur Vorgaben für die Strukturierung der Inhalte, sondern auch über die Inhalte selbst gemacht. Daher unterstützt GSCV Standards

---

<sup>1</sup> Ausgabe August 2007

und Spezifikationen für Datumsformate [ISO8601], Sprachen [RFC3066], Ländercodes [ISO3166], Branchencodes [NACE], Bildungsrichtungen und Abschlüsse [EURYDICE]. Für die Datenstruktur wird das HR-XML SEP 2.4 candidate ohne Erweiterungen verwendet.

Über das Internetportal des GSCV kann eine Präsentation über den Stand der Dinge<sup>1</sup> [TEE07] heruntergeladen werden. Laut dieser Präsentation wurde bereits ein Standard entwickelt und von verschiedenen Unternehmen in Pilotprojekten eingesetzt. Außerdem sollen bereits Gespräche mit weiteren Unternehmen geführt werden, wozu die Social Networking-Internetplattform XING<sup>2</sup> und die Stellenbörse Monster<sup>3</sup> gehören, um diese von GSCV zu überzeugen.

Auf dem Internetportal steht eine Implementierungsrichtlinie für den GSCV bereit [GER07]. Diese beschreibt ausführlich, wie eine XML-Datei aufgebaut sein muss, damit sie GSCV genügt. Im Wesentlichen wird dazu eine XML-Schema-Datei des HR-XML-Konsortiums (candidate.xsd) verwendet. Die Datei wird lediglich um Angaben von verwendeten Standards und Spezifikationen ergänzt und es wird ein von GSCV entworfener XML-Stylesheet für die Formatierung des Lebenslaufes eingesetzt. Das in der Implementierungsrichtlinie enthaltene Beispiel kann auf dem Internetportal des GSCV heruntergeladen werden.

Im Folgenden wird anhand eines Beispiels das Eintragen von Bewerberdaten in einem GSCV konformem Format gezeigt. Es sind dabei noch weitere Ergänzungen möglich, zum Beispiel die Angabe einer Militärausbildung, Publikationen usw.

---

1 Vom 03.05.2007

2 XING AG, <http://www.xing.com> (Stand: 22.01.2008)

3 Monster AG, <http://www.monster.de> (Stand: 22.01.2008)

---

```

(1) <?xml version="1.0" encoding="UTF-8"?>
(2) <Candidate xml:lang="de-DE" xmlns="http://ns.hr-xml.org/2006-02-28"
(3)   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
(4)   xsi:schemaLocation="http://ns.hr-xml.org/2006-02-28
(5)   http://ns.hr-xml.org/2_4/HR-XML-2_4/SEP/Candidate.xsd">
(6) <CandidateProfile xml:lang="de-DE">
(7) <!-- Hier wird die Sprache des Profils (CV) nach RFC 3066 angegeben.
(8) Default de-DE für deutsch. Sprachen-Codes (de) kommen aus dem
(9) ISO 639 Standard. Länder Codes (DE) aus ISO 3166. In der ersten
(10) Ausbaustufe des GSCV wird nur de-DE unterstützt-->
(11) <ProfileName><![CDATA[German Standard CV von Klaus Müller]]>
(12) </ProfileName>
(13) <PersonalData>
(14)   <PersonName>
(15)     <GivenName><![CDATA[Klaus]]></GivenName>
(16)     <!-- Vorname des Bewerbers -->
(17)     <FamilyName><![CDATA[Müller]]></FamilyName>
(18)     <!-- Nachname des Bewerbers -->
(19)   </PersonName>
(20)   <ContactMethod>
(21)     <Use>personal</Use><!-- Deafault Wert konstant personal -->
(22)     <Telephone>
(23)       <FormattedNumber>0049 6032 934037</FormattedNumber>
(24)     </Telephone>
(25)     <InternetEmailAddress>me@example.com</InternetEmailAddress>
(26)     <PostalAddress>
(27)       <CountryCode>DE</CountryCode>
(28)       <!-- Nach ISO 3166 -->
(29)       <PostalCode>61209</PostalCode>
(30)       <Region>Hessen</Region>
(31)       <!-- wichtig ist hier die korrekte deutsche Schreibweise mit
(32)         Umlauten -->
(33)       <Municipality>Echzell</Municipality>
(34)       <DeliveryAddress>
(35)         <StreetName>Schöne Aussicht</StreetName>
(36)         <BuildingNumber>30 a</BuildingNumber>
(37)       </DeliveryAddress>
(38)     </PostalAddress>
(39)   </ContactMethod>
(40) </PersonalData>
(41) <EmploymentHistory>
(42) <!-- hier steht die Historie des Berufsweges des Bewerbers.
(43)   EmployerOrg kann dabei mehrfach wiederholt werden. -->
(44)   <EmployerOrg>
(45)     <EmployerOrgName><![CDATA[Universität Karlsruhe]]>
(46)   </EmployerOrgName>
(47)     <EmployerContactInfo>
(48)       <LocationSummary>
(49)         <Municipality>Karlsruhe</Municipality>
(50)         <CountryCode>DE</CountryCode>
(51)         <!-- Land nach ISO 3166 -->
(52)       </LocationSummary>
(53)     </EmployerContactInfo>
(54)   <PositionHistory positionType="internship">
(55)   <!-- als PositionType sind folgenden Werte erlaubt : contract,

```

---

---

```

(56)      contractToHire, direct Hire, internship, temp, volunteer-->
(57)      <Title>Wissenschaftliche Hilfskraft</Title>
(58)      <OrgName>
(59)      <OrganizationName>Forschungszentrums Informatik an der
(60) Universität Karlsruhe</OrganizationName> <!-- hier kann, wenn
(61) gewünscht, die nächst tiefere Organisationseinheit eingetragen werden.
(62) Ansonsten muss hier der gleiche Name wie unter EmployerOrgName
(63) eingetragen werden. -->
(64)      </OrgName>
(65)      <OrgIndustry primaryIndicator="true">
(66)      <IndustryCode classificationName="NACE 1.1">73</IndustryCode>
(67)      <!-- Branche nach NACE 1.1 level 3 -->
(68)      </OrgIndustry>
(69)      <Description>Beschreibung der Tätigkeit</Description>
(70)      <StartDate>
(71)      <AnyDate>1995-02-01</AnyDate>
(72)      <!-- Datum nach ISO 8601 -->
(73)      </StartDate>
(74)      <EndDate>
(75)      <AnyDate>1995-09-01</AnyDate>
(76)      <!-- Datum nach ISO 8601 -->
(77)      </EndDate>
(78)      </PositionHistory>
(79)      </EmployerOrg>
(80) </EmploymentHistory>
(81) <EducationHistory>
(82) <!-- SchoolOrInstitution kann mehrfach gefüllt werden. -->
(83) <SchoolOrInstitution schoolType="x:Universität (Hochschule)">
(84) <!-- Werte für den Schooltype aus dem Eurydice Katalog. Wichtig x:
(85) muss voran gestellt werden, um hr-xml kompatibel zu bleiben -->
(86) <School>
(87) <SchoolName>Universitaet Karlsruhe</SchoolName>
(88) </School>
(89) <PostalAddress>
(90) <CountryCode>DE</CountryCode> <!-- Nach ISO 3166 -->
(91) <Municipality>Karlsruhe</Municipality>
(92) </PostalAddress>
(93) <Degree degreeType="x:Diplom">
(94) <!-- degreeType nach Eurydice x: aus Kompatibilitätsgründen. -->
(95) <DegreeName>Diplom-Ingenieur Maschinenbau</DegreeName>
(96) <DegreeDate>
(97) <AnyDate>1994-07-01</AnyDate> <!-- Datum nach ISO 8601 -->
(98) </DegreeDate>
(99) <DegreeMajor>
(100) <Name>Motortechnik</Name>
(101) <!-- genaue Spezifizierung des Abschlusses oder Schwerpunktes -->
(102) </DegreeMajor>
(103) <DegreeMeasure>
(104) <EducationalMeasure>
(105) <MeasureValue>
(106) <StringValue>91%</StringValue> <!-- Note -->
(107) </MeasureValue>
(108) <LowestPossibleValue>
(109) <StringValue>0%</StringValue>
(110) <!-- schlechtest mögliche Note -->

```

---

---

```

(111)         </LowestPossibleValue>
(112)         <HighestPossibleValue>
(113)           <StringValue>100%</StringValue> <!-- Beste Note -->
(114)         </HighestPossibleValue>
(115)       </EducationalMeasure>
(116)     </DegreeMeasure>
(117)     <Comments><![CDATA[Titel oder Thema der Abschlussarbeit]]>
(118)     </Comments>
(119)   </Degree>
(120) </EducationHistory>
(121) </CandidateProfile>
(122) </Candidate>

```

---

#### **Quelltext 23: Lebenslauf als GSCV-XML-Datei<sup>1</sup>**

Das Quellcodebeispiel 23 zeigt einen GSCV-konformen Lebenslauf. Das hier verwendete XML-Schema ist „Candidate.xsd“ des HR-XML-Konsortiums. Die Datei ist auch HR-XML konform, da lediglich Standards und Spezifikationen zum Füllen der XML-Elemente verwendet werden. Beispielsweise werden in Zeile 27 der ISO-3166 Standard für die Eingabe des Ländercodes und in Zeile 97 der ISO-8601 Standard für die Datumsformatierung verwendet. Da GSCV komplett auf HR-XML basiert und nur Standards und Spezifikationen zum Befüllen der XML-Elemente einsetzt, würde sich dieses Format ebenfalls als Standard für Lebensläufe eignen. Hinter GSCV steht eine Reihe von großen Unternehmen, die diesen Standard unterstützen, wodurch eine weite Verbreitung des Standards möglich ist. Das Ziel hinter GSCV war außerdem, einen Standard zu entwerfen, der die Onlinebewerbung vereinfachen soll. Dies deckt sich mit einigen Zielen dieser Diplomarbeit. Allerdings ist es auch bei diesem Format nicht vorgesehen, binäre Dateien innerhalb einer XML-Datei zu speichern.

#### **3.1.3.4 iProfile**

Das „iProfile“ ist ein industrielles Standardformat, um Lebensläufe online zu speichern *[IPROFILE]* und wird in Großbritannien von vielen Unternehmen eingesetzt. Das Profil kann online von Benutzern angelegt werden. Bei einer eingehenden Bewerbung kann das Unternehmen die Daten abrufen, sofern es sich für diesen Dienst registriert hat. Der Bewerber hingegen hat die Möglichkeit Zeit zu sparen, da die Lebenslaufdaten nicht bei jeder Bewerbung neu eingegeben werden müssen.

---

<sup>1</sup> Quelle: [http://german-standard-cv.de/candidate\\_mueller\\_comment.xml](http://german-standard-cv.de/candidate_mueller_comment.xml) (Stand: 22.01.2008)

Das iProfile kann nur von Unternehmen angesehen werden, die der Bewerber freigeschaltet hat. Damit ist sichergestellt, dass die Daten geheim gehalten bleiben und nicht von einem Außenstehenden betrachtet werden können.

Das Format ist allerdings nur online verfügbar und die XML-Schemata stehen nicht zum Herunterladen bereit, eine tiefer gehende Untersuchung war nicht möglich. Dieses Format kann daher nicht für die zu entwickelnde Software eingesetzt werden.

### **3.1.3.5 Fazit**

Das HR-XML-Konsortium stellt ein sehr mächtiges, frei verfügbares Format bereit. Die einzelnen Komponenten sind gut dokumentiert und es gibt viele Beispiele für die Anwendung der XML-Schemata.

HR-BA-XML baut auf dem HR-XML-Format auf und ist auf die Anforderungen des deutschen Arbeitsmarktes abgestimmt und optimiert. Es ist in Planung, das HR-BA-XML-Format mit in das HR-XML-Format eingliedern zu lassen. Von der Bundesagentur für Arbeit wird das Format bereits eingesetzt und auch viele Unternehmen stellen Ihre Stellenausschreibungen bereits in diesem Format der zur Verfügung. Die Akzeptanz für in diesem Format bereitgestellte Bewerbungen, bzw. Lebensläufe, ist sicherlich vorhanden. Es existiert eine umfassende Formatbeschreibung, die die Nutzung sehr genau beschreibt. Nachteil beim Einsatz dieses Formates ist allerdings, dass es zwar auf HR-XML basiert, doch auch eigene, spezielle XML-Elemente enthält. Bei Verwendung dieses Formates muss also die zu entwickelnde Software ggf. angepasst werden, wenn sich etwas in der HR-XML-Definition und/oder in der HR-BA-XML-Definition ändert. Es müssen also immer Änderungen von zwei verschiedenen Organisationen berücksichtigt werden.

Das Format GSCV entstand aus einer Initiative von mehreren Unternehmen und könnte zu einem weit verbreitenden Standard zu werden. GSCV beruht komplett auf HR-XML und benutzt nur Standards und Spezifikationen zum einheitlichen Eintragen der Daten innerhalb der XML-Elemente, wie zum Beispiel ein einheitliches Datumsformat oder einheitliche Sprachdefinitionen nach ISO-Norm. Welche Standards und Spezifikationen eingesetzt werden sollen, wurde durch Absprache der an GSCV beteiligten Unternehmen entschieden. Es muss dabei nur bei Änderungen der HR-XML Definition eine Anpassung vorgenommen werden.

Das Internetportal iProfile ist eher für den englischsprachigen Markt geeignet und bietet auch keine frei zugängliche Möglichkeit an, die benutzten XML-Schema-Dateien herunterzuladen und anzuwenden.

Feature	HR-XML	HR-BA-XML	GSCV	iProfile
XML-Schema verfügbar	Ja	Ja	Ja <sup>1</sup>	Nein
Anwendung der XML-Dateien dokumentiert	Ja	Ja	Ja	Nein
Informationen auf der Internetseite	Ja	Nein	Nein <sup>2</sup>	Ja
Für den deutschen Markt geeignet	Ja	Ja	Ja	-
Binäre Daten speichern	Nein	Nein	Nein	-

**Tabelle 3: Entscheidungsmatrix zum Einsatz eines XML Standards**

Anhand der Entscheidungsmatrix würden sich die XML-Spezifikationen von HR-XML, HR-BA-XML und GSCV für den Einsatz innerhalb des Diplomprojekts eignen. Alle drei Standards bieten vergleichbare Features.

Für HR-BA-XML spricht, dass es von der Bundesagentur für Arbeit eingesetzt wird und die Verbreitung des Standards dadurch staatlich gefördert wird.

Der GSCV hingegen wird von marktführenden Unternehmen in Deutschland eingesetzt und verspricht dadurch, zu einem wichtigen Standard auf dem deutschen Markt zu werden. Außerdem basiert der GSCV auf HR-XML und ist komplett mit HR-XML kompatibel. Ein in GSCV geschriebener Lebenslauf kann also auch im Ausland verwendet werden, wenn ein dortiges Unternehmen HR-XML oder einen darauf aufbauenden Standard unterstützt.

Alle Standards haben gewichtige Gründe, warum sie als Standard für die Speicherung eingesetzt werden könnten. Da GSCV eng mit HR-XML verbunden ist, sollte dieser Standard für die Speicherung der Daten im Rahmen dieser Diplomarbeit

<sup>1</sup> Benutzung von HR-XML

<sup>2</sup> Seite befindet sich noch im Aufbau (Stand: 22.01.2008)

bevorzugt werden. Damit die Daten auch in einem anderen Format gespeichert werden können, zum Beispiel als HR-BA-XML-Datei, ist bei der Entwicklung des Programms darauf zu achten, dass eine spätere Weiterentwicklung und das Hinzufügen von weiteren Standards ermöglicht wird. Dies kann durch ein Pluginkonzept realisiert werden.

### 3.1.4 Evaluierung von vergleichbaren Softwareprogrammen

Bevor eine eigene Software entwickelt wird, soll untersucht werden, ob bereits vergleichbare Programme zur Erstellung von Lebensläufen auf dem Markt sind. Dabei muss herausgestellt werden, welche Standards sie einsetzen und welche Features sie dem Benutzer bieten.

Um an geeignete Software zu kommen, wurde auf den Internetseiten von CHIP Online<sup>1</sup>, ZDNet<sup>2</sup> und Tucows<sup>3</sup> nach kommerziellen und freien Programmen gesucht. Auch wurde Google<sup>4</sup> mit in die Suche einbezogen. Zu den dabei verwendeten Suchbegriffen gehörten: „resume“, „Lebenslauf“, „Lebenslauf Software“, „Lebenslauf Software Tool“, „Lebenslauf exportieren“, „resume export“, „hr-xml export“, „hr-xml tool“, „currucilum vitae“, „currucilum vitae tool“, „Software Lebenslauf erstellen“, „Software resume“, „Professional Resumes“, „Professional Resumes quick easy“ und „resume-maker“.

Das Ergebnis der Evaluierung ist, dass es viele Produkte auf dem Markt gibt, mit denen ein Lebenslauf erstellt werden kann. Doch nur wenige dieser Produkte bieten auch einen Export der Lebenslaufdaten in einem standardisierten XML-Format an. Im weiteren Verlauf dieser Diplomarbeit werden zwei Produkte näher beschrieben, die den Anforderungen gerecht geworden sind und als Referenzprodukt für die Entwicklung eines Personenprofil-Editors angesehen werden können.

---

1 Chip Xonio Online GmbH, <http://www.chip.de/> (Stand: 22.01.2008)

2 CNET Networks Deutschland GmbH, <http://www.zdnet.de/> (Stand: 22.01.2008)

3 Tucows.Com Inc., <http://tucows.net/> (Stand 22.01.2008)

4 Google Inc., <http://www.google.de/> (Stand 22.01.2008)

### 3.1.4.1 Resume Builder

Der Resume Builder ist ein Produkt der Firma Sarm Software [SARMS]. Die Software hilft beim Erstellen eines Lebenslaufes und bietet sehr viele Features. Auf der Internetseite von Sarm Software wird eine Probeversion des Resume Builders<sup>1</sup> zum Herunterladen und Testen angeboten.

Nach dem Herunterladen einer ca. 12 MB großen Installationsdatei kann diese ausgeführt und der Resume Builder installiert werden. Danach steht das Programm als Testversion zur Verfügung. Die Vollversion der Software kostet \$24,95 und kann über die Internetseite des Anbieters bezogen werden. Sie kann durch Eingabe eines Registrierungsschlüssels freigeschaltet werden. Der dazu benötigte Schlüssel wird nach erfolgreicher Überweisung per E-Mail an den Benutzer verschickt. In der Testversion werden fast alle Features angeboten, aber der Resume Builder kann nur für eine bestimmte Anzahl von Tagen frei benutzt werden. Zu den Features, die in der Testversion nicht angeboten werden, gehört u.a. ein Export des erstellten Lebenslaufes als MS-Word-Datei. Nach der ersten Installation stehen 9 Tage zum Testen zur Verfügung, danach erscheint permanent ein Fenster mit der Aufforderung, die Software zu erwerben. Nach Ablauf der Testphase sind einige Funktionen nicht mehr verfügbar, zum Beispiel das Exportieren als HR-XML-Datei.

Der erste Eindruck des Resume Builders ist sehr positiv. Eine große Anzahl Features wurde umgesetzt, das Design ist übersichtlich und leicht verständlich. Auf der rechten Seite der Benutzeroberfläche werden Tipps zur Gestaltung einer Bewerbung angezeigt und es kann zwischen einer Vielzahl verschiedener Sprachen gewählt werden.

Bereits während der Installation auffällt fällt auf, dass die Software schlecht in die deutsche Sprache übersetzt wurde. Bei vielen Auswahlmöglichkeiten ist deshalb nur zu erraten, was damit gemeint ist. Die folgende Abbildung zeigt den letzten Schritt während des Installationsvorganges. Anhand der Abbildung auf der nächsten Seite ist deutlich zu sehen, mit welchen Schwierigkeiten ein Benutzer durch die schlechte Übersetzung konfrontiert wird.

---

<sup>1</sup> Auch unter dem Namen OpenCV erhältlich: <http://opencv.com/en/index.htm> (Stand: 22.01.2008)

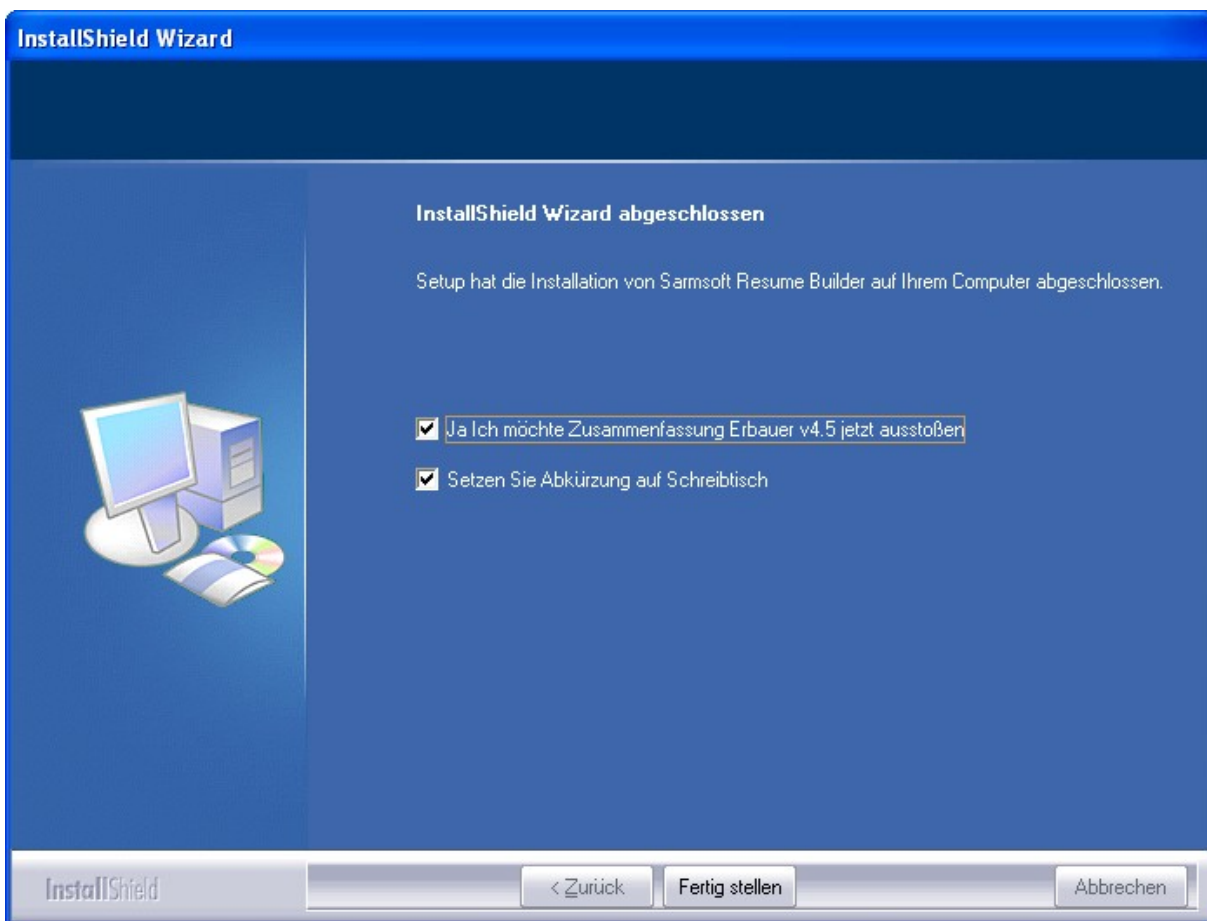


Abbildung 6: Resume Builder schlecht Übersetzt

Ist die sprachliche Hürde nach der Installation überwunden, kann ein erster Lebenslauf erstellt werden. Ein Assistent hilft beim Eingeben der personenbezogenen Daten wie Name, Vorname, Anschrift usw.

Nach Eingabe dieser Daten ist eine erste Version des Lebenslaufes erstellt. Der Lebenslauf kann mit weiteren Angaben zur Ausbildung, Berufserfahrung und weiteren Daten angereichert werden. Nach jeder Änderung sieht ein Benutzer immer das aktuelle Erscheinungsbild des Lebenslaufes. In der folgenden Abbildung ist die Benutzeroberfläche des Resume Builders während der Erfassung eines Lebenslaufes zu sehen:

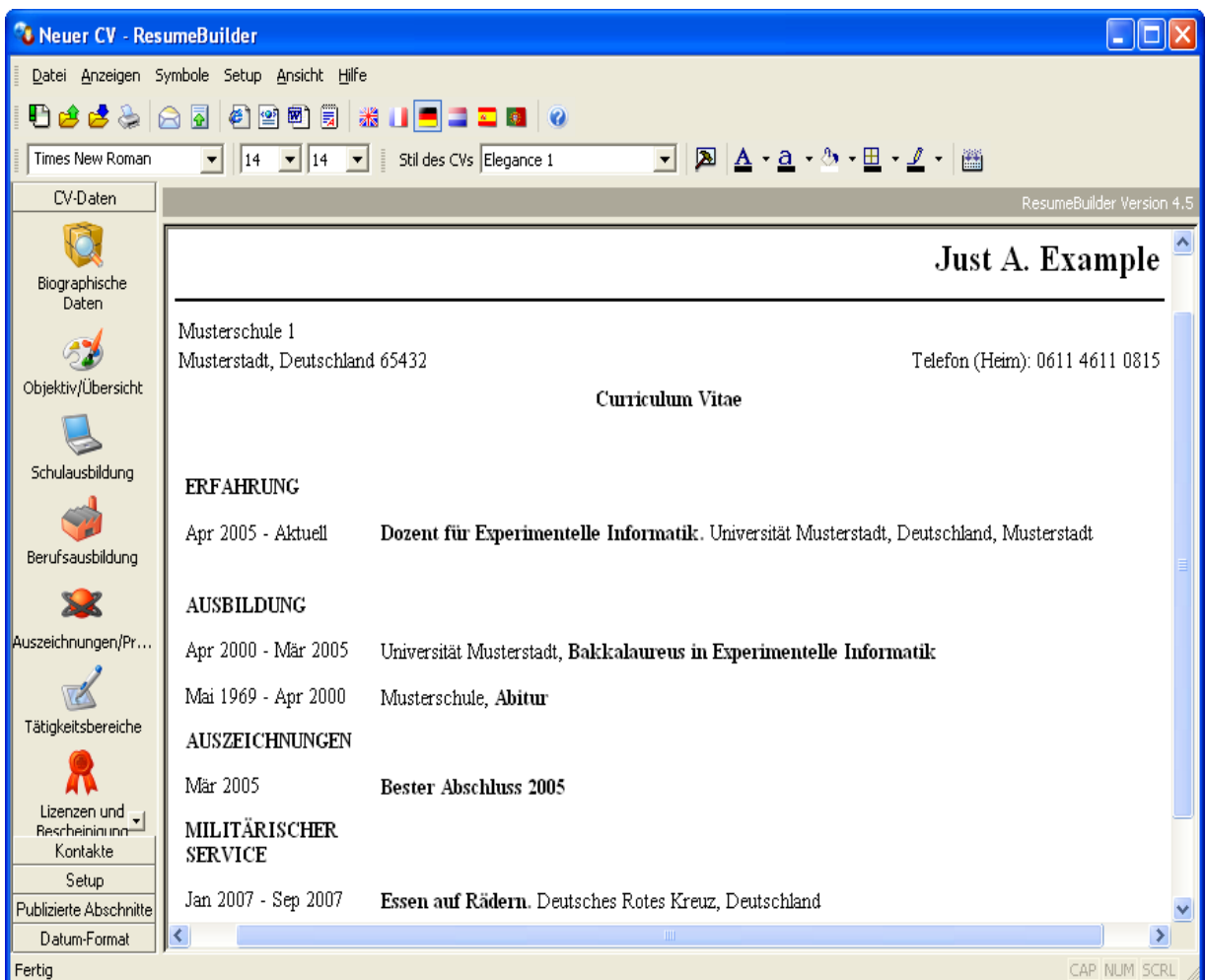


Abbildung 7: Resume Builder CV - Übersicht

Das Hauptfenster ist in verschiedene Bereiche unterteilt. Es gibt eine Menüleiste, über die alle Funktionen des Programms aufgerufen werden können. Darunter befindet sich eine Toolbar, mit den wichtigsten Features, damit diese schneller gefunden werden können. Auf der linken Seite befindet sich ein Menü mit verschiedenen Untermenüpunkten. Die bisherigen Eingaben werden in Form eines Lebenslaufes in der Mitte des Fensters dargestellt.

Wird eines der Elemente im linken Menü ausgewählt, so erscheint ein weiteres Fenster, in dem die Angaben zu dem Bereich eingegeben werden. Diese Fenster rufen vereinzelt ihrerseits wiederum Fenster auf, so dass es schnell für einen Benutzer mit wenig IT-Erfahrung unübersichtlich werden kann.

Ansonsten ist die Menüführung optisch sehr ansprechend gestaltet. Die Reihenfolge der Elemente eines Lebenslaufes kann frei gewählt werden. Es kann eingestellt werden in welchem Format die Datumsangaben angezeigt werden sollen und einzelne Bereiche können frei formatiert werden. Außerdem stehen verschiedene Templates zur Verfügung, mit denen das Design eines Lebenslaufes schnell veränderbar ist.

Unter dem Menüpunkt Setup kann ein Lebenslauf in verschiedenen Formaten auf einen FTP-Server geladen oder per E-Mail verschickt werden. Bei dem Versenden per E-Mail kann entweder ein Mailserver angegeben oder der in Microsoft Outlook eingetragene Mailserver verwendet werden.

Ein Export der Daten ist als Text-, HTML-, Microsoft Word- oder HR-XML-Datei möglich.

Vorteile	Nachteile
Es werden mehrere Sprachen angeboten	Die deutsche Übersetzung ist eher mangelhaft
Formatierung des Aussehens eines Lebenslaufes ist durch direktes Eingreifen oder Templates (Skins) möglich	Nicht erfahrene Nutzer verlieren bei vielen geöffneten Fenstern leicht den Überblick
Integration eines Fotos in den Lebenslauf	Beim Schulabschluss werden verschiedene Abschlussarten nicht angeboten (z.B. Diplom)
Rechtschreibprüfung für verschiedene Sprachen	Zivildienst kann nur als Militärausbildung angegeben werden
Versenden von E-Mails über einen Mailserver oder durch Microsoft Outlook	Lebenslauf ist auf den amerikanischen Markt zugeschnitten (siehe Punkt Zivildienst)
Export in viele gängige Dateiformate	
Die Reihenfolge der Angaben und damit das Aussehen des Lebenslaufes können frei gewählt werden	

**Tabelle 4: Resume Builder, Vor- und Nachteile**

Der Resume Builder bringt sehr viele Features mit, die für das im Rahmen der Diplomarbeit zu entwickelnde System auch interessant wären. Alle Features in die Anforderungsdefinition mit aufzunehmen, würde aber den Rahmen der Diplomarbeit sprengen, da es sich um sehr viele handelt, die in der gegebenen Zeit nicht alle umgesetzt werden könnten.

Die beiden schwerwiegendsten Nachteile sind die schlechte Übersetzung in die deutsche Sprache und die unübersichtliche Programmführung für unerfahrene Benutzer. Vor allem auf die Vermeidung des letzten Punktes sollte in dem Diplomprojekt sehr geachtet werden.

### 3.1.4.2 Easy Resume Creator

Der Easy Resume Creator ist ebenfalls von der Firma Sarm Software und kann von der Internetseite <http://www.winresume.com> (Stand: 22.01.2008) heruntergeladen werden. Ähnlich wie der Resume Builder kann vom Easy Resume Creator eine Testversion mit eingeschränkten Features installiert werden. Die Software ist nur in englischer Sprache verfügbar und bietet ansonsten alle aus dem Resume Builder bereits bekannten Features. Das Erscheinungsbild ist jedoch ein wenig verändert. Die Vollversion kann für \$34,95 erworben werden.

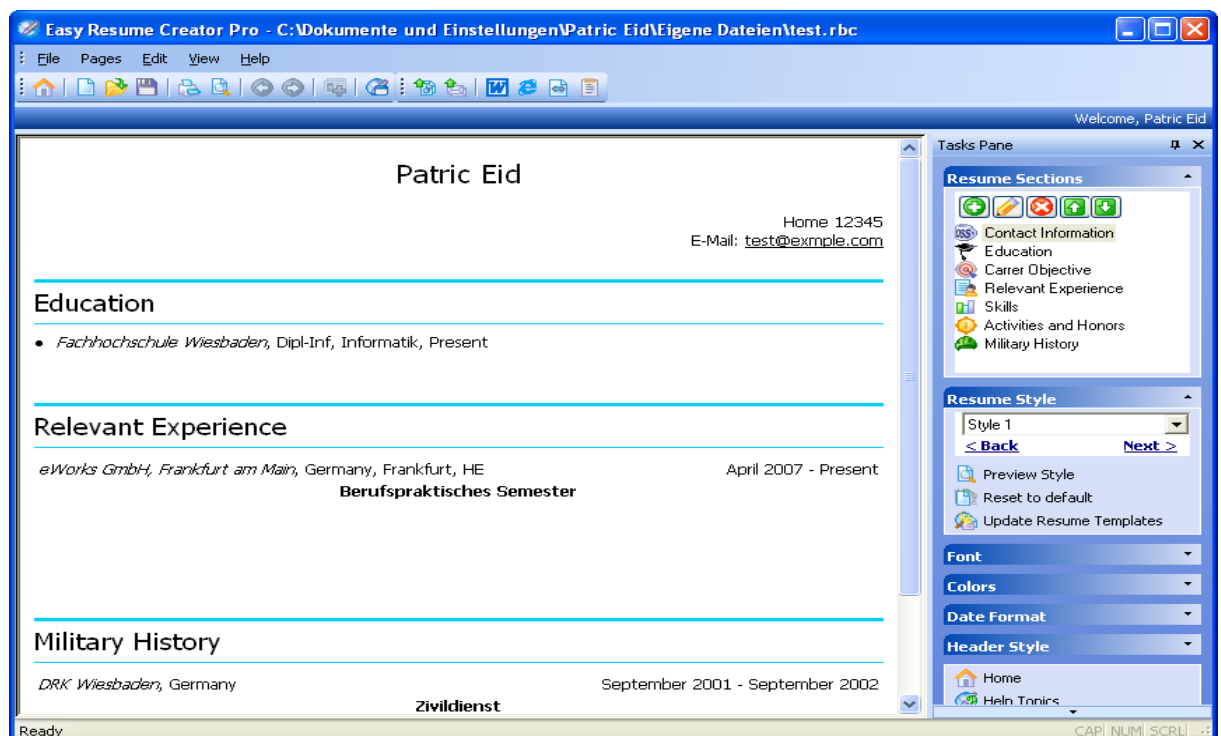


Abbildung 8: Easy Resume Creator Hauptfenster

Das Hauptfenster ist ähnlich wie beim Resume Builder unterteilt. Eine Ausnahme stellt das Menü dar. Es ist von der linken auf die rechte Seite verschoben. Die Buttons und Menüs sehen überarbeitet und neuer aus.

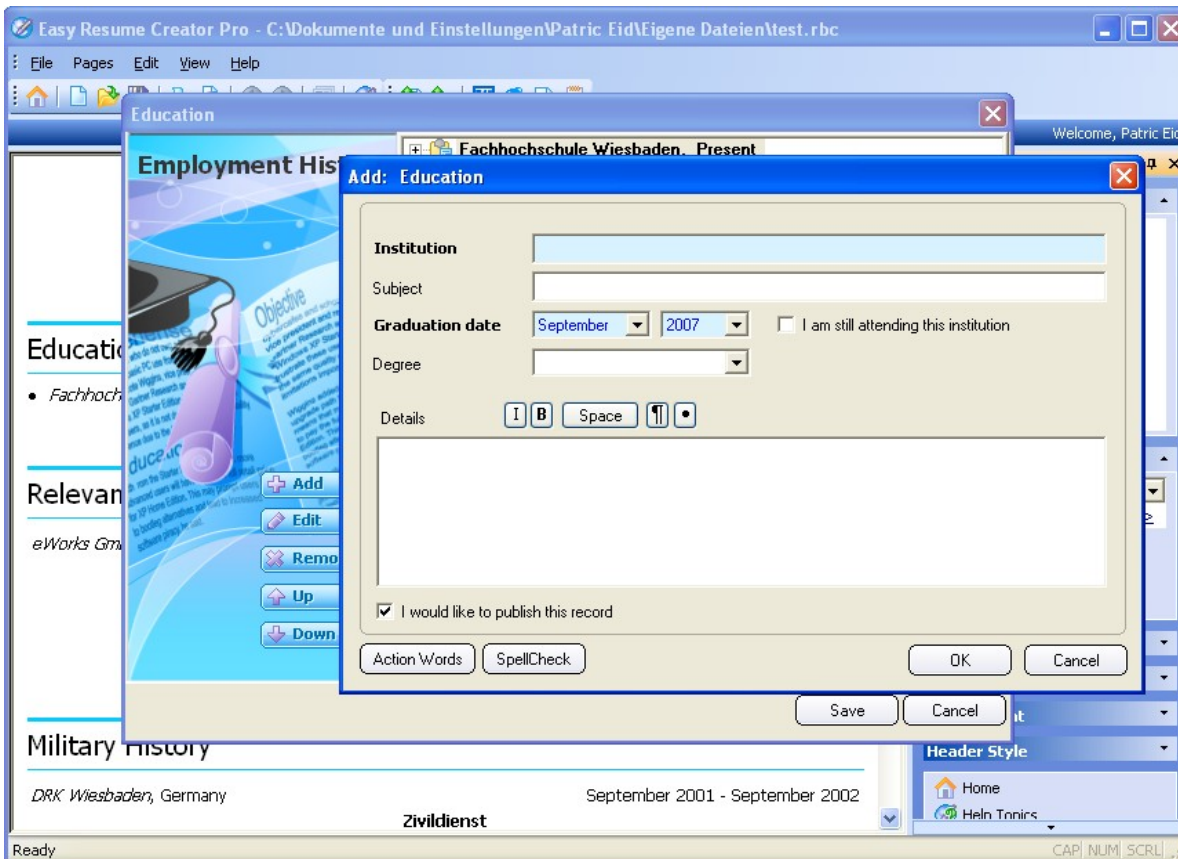


Abbildung 9: Easy Resume Creator hilft beim Anlegen der Daten

Wie auch beim Resume Builder öffnen sich beim Anlegen von Lebenslaufdaten neue Fenster. Auch hier hat ein Benutzer das Problem, dass sich schnell viele Fenster geöffnet haben. Das Erscheinungsbild der Fenster wurde offensichtlich ebenfalls verbessert und anschaulicher gestaltet.

Grundlegende funktionale Unterschiede sind in der Software allerdings nicht festzustellen. Auch hier ist ein Export als HR-XML-Datei möglich. Auch wird das Versenden der Daten per E-Mail unterstützt, genauso wie ein FTP-Upload.

## 3.2 Lösungsansatz

Da die zu entwickelnde Software gewisse Ziele verfolgt, wird in diesem Abschnitt ein Lösungsansatz präsentiert, welcher Bezug auf die Evaluierung der Standards zur Personenprofil-Speicherung und den Erkenntnissen aus der Evaluierung von vergleichbaren Produkten nimmt. Der Lösungsansatz berücksichtigt die Ziele der Diplomarbeit und des Diplomprojektes. Aus den Zielen geht hervor, dass das Programm für eine weit gefasste Endbenutzergemeinde konzeptioniert werden muss und für alle Berufe geeignet sein soll. Folgende Punkte ergeben sich aus dieser Zielsetzung:

- Die Bedienoberfläche soll verständlich und übersichtlich aufgebaut sein. Hierbei ist zu untersuchen, ob die Eingabe der Daten als eine Art Interview vollzogen werden kann.
- Einhaltung der acht goldenen Grundregeln der Schnittstellendefinition [SCH02]:

### 1. Konsistenz

Bei Eingabeaufforderungen und Menüs sollte eine identische Terminologie bestehen. Das Design, also Farbauswahl, Layout, Großschreibung usw. sollte durchgehend konsistent sein.

### 2. Tastaturkürzel

Für regelmäßige Benutzer sollte es durch Tastaturkürzel (engl.: „shortcuts“) ermöglicht werden, die Anzahl der Interaktionen zu verringern und dadurch das Tempo der Bearbeitung zu steigern.

### 3. Informatives Feedback

Für regelmäßige und weniger wichtige Handlungen kann die Systemantwort knapp ausfallen, bei weniger häufigen und wichtigen Aktionen sollten sie ausführlich und verständlich sein.

### 4. Geschlossene Dialoge

Handlungssequenzen sollten in Gruppen mit Anfang, Mitte und Ende organisiert werden. Nach Beendigung einer Folge von Aktionen sollte dem Benutzer ein Feedback gegeben werden.

### 5. Fehlervermeidung und einfaches Umgehen von Fehlern

Fehlereingaben sollten frühzeitig durch geeignete Auswahlmenüs vermieden werden. Das System soll Fehler erkennen und einfache, konstruktive und spezifische Instruktionen zur Korrektur anbieten.

### 6. Umkehr von Aktionen

Aktionen sollten wenn möglich umkehrbar sein. Dies sollte sich auf einzelne Aktionen oder Dateneingaben beziehen.

### 7. Kontrollbedürfnis unterstützen

Überraschende Aktionen des Systems sollen vermieden werden. Die Dateneingabe sollte nicht in ermüdenden Sequenzen erfolgen.

### 8. Reduzierung der Belastung des Kurzzeitgedächtnisses

Die Anzeige sollte einfach gehalten, multiple Anzeigeseiten zusammengelegt und die Frequenz der Fenster-Bewegung reduziert werden.

- Flexibler Umgang mit den verwendeten Standards zur Speicherung der Personenprofildaten. Dies könnte durch eine Export-Funktion erledigt werden, die verschiedene Formate zulässt. Wobei hier ein Pluginkonzept anzustreben ist. Das Pluginkonzept müsste im Rahmen der Diplomarbeit noch nicht ganz umgesetzt werden, doch die Infrastruktur dafür sollte zumindest geschaffen sein.
- Export der Daten als gut lesbare MS-Word- und HTML-Datei ermöglichen.
- Interne Speicherung der Daten kann in einem allgemeinen XML-Format erfolgen. XML bietet sich hierbei an, da die Daten so komfortabel in ein anderes XML-Format, zum Beispiel HR-XML, transformiert werden können.
- Laden von erstellten Profilen und füllen der Formulare mit den Profildaten.
- Einlesen der unterstützten XML-Formate. Hierbei ist allerdings zu beachten, dass der Export der Daten (in ein unterstütztes XML-Format) eine höhere Priorität hat. Im Rahmen der Diplomarbeit sollte aber zumindest die Infrastruktur für das Einlesen geschaffen werden.

- Eingabeformulare für die Bestandteile eines Lebenslaufes:
  - Persönliche Angaben
  - Kontaktdaten
  - Schulbildung
  - Wehr- oder Zivildienst / Freiwilliges soziales Jahr
  - Studium
  - Praktische Erfahrungen
  - Sprachkenntnisse
  - IT-Kenntnisse
  - Weiterbildung
  - Hobbies und Interessen
  - Gesellschaftliches Engagement / Ehrenamtliche Aktivitäten
  - Publikationen

### 3.3 Design der Benutzerschnittstelle

In dem kommenden Abschnitt wird anhand eines Prototyps das Design der Anwendung beschrieben. Das entworfene Design sollte dabei nicht als endgültige Version, sondern als Richtlinie und Einstieg für die spätere Entwicklung des Programms angesehen werden. In der Entwicklungsphase könnten also noch Veränderungen am Design vorgenommen werden, falls sich eine der hier vorgenommenen Entscheidungen als nicht praktikabel erweisen sollte.

Die Benutzeroberfläche des Editors unterteilt sich in zwei Hauptbereiche. Auf der linken Seite befinden sich ein Menü, über das der Benutzer die einzelnen Elemente des Lebenslaufes auswählen kann. In der Abbildung 10 ist dieses Menü als Navigationsmenü beschriftet und wird im weiteren Verlauf nur noch Navigation genannt. Auf der rechten Seite der Benutzeroberfläche wird für die ausgewählten Lebenslaufdaten ein entsprechendes Formular geladen. Dieser Bereich wird in Abbildung 10 als Content-Bereich (im weiteren Verlauf nur noch Content genannt) bezeichnet. Der obere Rand enthält eine weitere Menüleiste. Die Menüleiste soll dazu dienen, wichtige Funktionen wie Datei speichern, öffnen oder schließen bereitzustellen. In der folgenden Abbildung ist die Aufteilung der Bereiche der Benutzeroberfläche zu sehen:

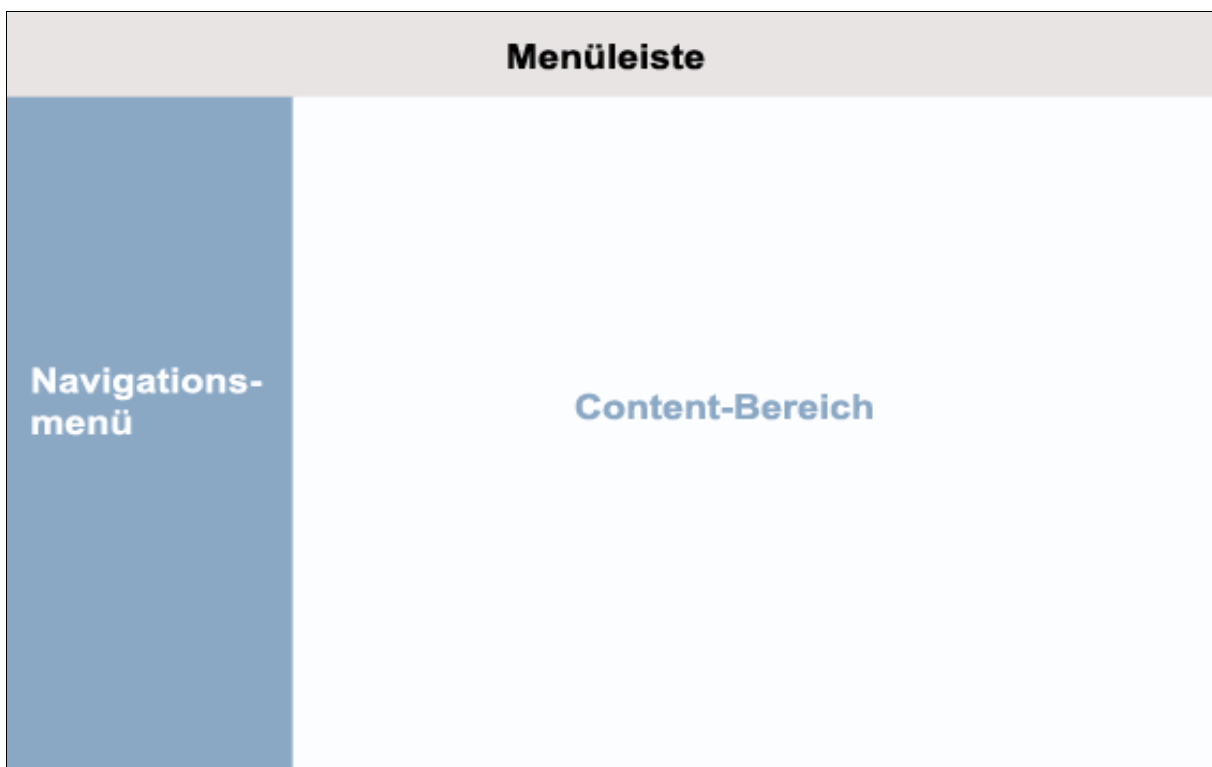


Abbildung 10: Design der Benutzeroberfläche nach Programmstart

Die Einträge in der Navigation stellen jeweils eine Seite mit Informationen dar. Wird ein solcher Eintrag angeklickt, so öffnet sich ein separates Benutzersteuerelement (engl.: „user control“). Dieses Element enthält ein Formular, über welches die Daten zu dem ausgewählten Lebenslaufabschnitt eingegeben werden können. Für eine benutzerfreundliche Steuerung durch die einzelnen Bestandteile sollten die Elemente allerdings in Untermenüs eingeteilt werden. Dabei sollte eine aus Windows XP oder Microsoft Outlook bekannte Optik angestrebt werden. Durch Pfeile soll dem Benutzer kenntlich gemacht werden, dass die Menüs auf- und zuklappbar sind. Kann ein Menü aufgeklappt werden, so sollte der Pfeil nach unten zeigen. Um zu signalisieren, dass ein Menü zugeklappt werden kann, sollte der Pfeil nach oben zeigen.

Neben der Möglichkeit die einzelnen Elemente über die Navigation aufzurufen, sollte der Benutzer auch durch den Erstellungsvorgang geführt werden. Dabei sollen über den Formularen Vor- und Zurück-Buttons angebracht sein. Der aktuelle Status der Eingabe soll durch eine Statusanzeige jederzeit angezeigt werden. Die Statusanzeige ist in dem Programm oberhalb der Navigation zu finden. Die folgende Abbildung stellt einen ersten Ansatz einer Benutzeroberfläche dar, eine Verfeinerung findet im weiteren Verlauf dieser Diplomarbeit statt.

The screenshot shows a window titled "Skill-Editor" with a blue title bar. On the left is a vertical navigation menu with a blue background and white text. The menu items are: Menü, Status, Persönliche Angaben (highlighted), Kontaktdaten, Schulbildung, Studium, Wehr- und Zivildienst, Praktische Erfahrung, Sprachen, IT-Kenntnisse, Weiterbildung / Zertifikate, Publikationen, and Sonstige Angaben. The main content area is light gray and contains a form titled "Persönliche Angaben". The form fields are: Name: (text input), Vorname: (text input), Namenszusatz: (text input), Künstlername: (text input), Geschlecht: (dropdown menu with "Bitte auswählen;" selected), Geburtsdatum: (calendar dropdown menu showing "Sonntag , 21. September 1980"), and Geburtsort: (text input). A "Weiter >>" link is located in the top right corner of the form area.

Abbildung 11: Design mit geöffnetem Formular

In Abbildung 11 ist ein geöffnetes Untermenü mit Einträgen zu „Persönliche Angaben“ geöffnet. Durch Klick auf den Button „Weiter >>“ kann ein Benutzer auf die nächste Seite gelangen, in diesem Fall „Kontaktdaten“. Dies kann aber auch über den entsprechenden Eintrag in der Navigation erfolgen. Auf der letzten zu bearbeitenden Seite soll dieser Link nicht erscheinen oder deaktiviert sein. Wenn alle erforderlichen Daten eingegeben wurden, so wird die Statusanzeige oben links im Bild um den entsprechenden Wert erweitert. Das soll dem Benutzer vermitteln, wie lange es noch in etwa dauert, bis alle relevante Daten eingegeben wurden. Die Prozentanzahl soll darüber hinaus noch zusätzlich über der Statusanzeige in Form einer Zahl angegeben werden.

Zusätzliche Features des Programms, wie etwa der Export der Daten als MS-Word-Datei, sollen in der Menüleiste untergebracht werden.

### **3.4 Architektur des Programms**

In diesem Kapitel wird die Architektur der Anwendung beschrieben. Zuerst erfolgt dazu eine Übersicht Anwendungsmöglichkeiten aus Sicht eines Benutzers. Die Fülle der Anwendungen werden in einem Anwendungsfallmodell bildlich dargestellt und beschrieben. Die darin vorkommenden Akteure werden genannt und ihr Aufgabengebiet erläutert.

Da die Eingabe und der Export der Lebenslaufdaten ein wichtiger Bestandteil des Programms sind, werden beide Schritte anhand eines Aktivitätsdiagramms visualisiert und im weiteren Verlauf näher betrachtet.

In dem Klassendiagrammen wird der strukturelle Aufbau der Anwendung veranschaulicht. Es wird ein Diagramm gezeigt, aus dem ersichtlich wird, wie die Benutzeroberfläche mit dem restlichen Programm kommuniziert. In einem separaten Diagramm werden die Zusammenhänge der Datenschicht erläutert. Das Pluginkonzept wird in einem weiteren Klassendiagramm beschrieben.

Aufbauend auf den Klassendiagrammen folgt im letzten Unterkapitel ein Sequenzdiagramm, welches die Ablaufreihenfolge beim Export von Lebenslaufdaten in die Exportformate GSCV (HR-XML), HTML und Microsoft Word zeigt. Anhand dieses Diagramms wird ersichtlich, welche Klassen benötigt werden, um Daten zu exportieren.

#### **3.4.1 Anwendungsfallmodell**

Innerhalb eines Anwendungsfallmodells gibt es Akteure, die eine bestimmte Rolle einnehmen. Ein Akteur ist dabei allerdings keine konkrete Person. Vielmehr kann innerhalb einer Anwendung kann eine Person mehrere Rollen einnehmen und somit als verschiedene Akteure auftreten. Im Folgenden werden alle im zu entwickelnden Programm vorkommenden Rollen beschrieben. Danach wird gezeigt, welche Anwendungsfälle es bei den Akteuren geben kann. Die Fälle werden beschrieben und ihre Abhängigkeiten untereinander in einem Anwendungsfalldiagramm gezeigt.

### 3.4.1.1 Akteure

#### **Benutzer**

In der Anwendung gibt es einen primären Akteur Benutzer. Ein Benutzer kommt dabei aus einer weit gefassten Endbenutzergemeinde, wodurch es sich um einen häufigen bis hin zu einem gelegentlichen Benutzer handeln kann. Die IT-Kenntnisse des Benutzers sind nicht bekannt und sollen für das System auch nicht relevant sein. Die Anwendung muss deshalb einfach gestaltet werden, sollte jedoch auch einen erfahrenen Benutzer ansprechen. Das Dialogsystem darf nicht zu kompliziert aufgebaut werden und bei der Eingabe der Daten sollte dem Benutzer ausreichend Hilfe angeboten werden, beispielsweise ein Kalender für Datumsangaben.

In der folgenden Tabelle werden die fachlichen Ziele des Akteurs Benutzer dargestellt, die implementiert werden sollen. Aus jedem Ziel wird ein Anwendungsfall abgeleitet, inklusive Beschreibung und Priorität. Letztere besagt, in welcher Reihenfolge die Ziele während der Entwicklung verwirklicht werden sollen.

Beschreibung	Priorität (1 = sehr hoch, 4 = niedrig)
Der Benutzer soll seine Lebenslaufdaten eingeben können	1
Es soll ein neuer Lebenslauf angelegt, ein bestehender Lebenslauf geöffnet und ein bearbeiteter Lebenslauf gespeichert werden können	2
Die Lebenslaufdaten sollen in einem leicht lesbarem Format exportiert werden können	3

**Tabelle 5: Die Ziele des Akteurs Benutzer**

Hinter jedem Ziel verbergen sich zu bewältigende Aufgaben, die in der nächsten Tabelle abgebildet werden. Die Aufgaben werden beschrieben und erhalten Informationen, in wie weit dabei auf Systeminformationen zurückgegriffen werden muss. Außerdem wird die Art des Zugriffs angegeben.

Beschreibung	Systeminformationen	Systemzugriff
Die Lebenslaufdaten müssen vom Benutzer eingegeben werden. Dabei werden die Daten bereits validiert, damit der Benutzer direkt auf Fehler reagieren kann	keine	-
Der Benutzer bekommt die Möglichkeit, einen neuen Lebenslauf anzulegen. Dabei kann ein Zielspeicherort und ein Profilname angegeben werden. Ein bereits angelegter Lebenslauf soll geöffnet werden können. Nach der Eingabe oder Änderung von Daten soll der Lebenslauf gespeichert werden können	Der Benutzer benötigt Zugriff auf ein Datenverzeichnis und muss dort Ordner und Dateien anlegen oder öffnen können	Lese- und Schreibrechte
Die Lebenslaufdaten sollen als eine standardisierte XML-Datei, eine HTML- oder MS-Word-Datei exportiert werden können	Der Lebenslauf wird in einem ausgewählten Format in einem Datenverzeichnis erstellt	Schreibrechte

**Tabelle 6: Hauptaufgaben des Akteurs Benutzer**

Der Benutzer soll darüber informiert werden, wenn bei der Validierung der Lebenslaufdaten ein Fehler aufgetreten ist. Überdies sollen mögliche Fehler beim Speichern, Öffnen, Anlegen oder Exportieren eines Lebenslaufes angezeigt werden.

### 3.4.1.2 Anwendungsfälle

Ein Anwendungsfall (engl.: „use case“) dient dazu, das Verhalten einer Software zu spezifizieren. Nachfolgend werden die vorkommenden Anwendungsfälle näher beschrieben. Zu jedem Anwendungsfall, im weiteren Verlauf UseCase genannt, gibt es eine Beschreibung, den dazugehörigen Akteur, Vor- und Nachbedingungen sowie ein Szenario, in dem der Ablauf des UseCases dokumentiert wird.

#### 3.4.1.2.1 Datei bearbeiten

- Beschreibung: Es sollen Profildateien bearbeitet werden können. Dieser UseCase ist abstrakt, die Behandlung der Dateien findet in abhängigen UseCases statt.
- Akteur: Dieser UseCase wird vom Akteur Benutzer ausgeführt.
- Vorbedingungen: Der Akteur muss das Programm gestartet haben. Danach können die verschiedenen Funktionen ausgeführt werden.
- Nachbedingungen: Es wird der UseCase „Neuen Lebenslauf anlegen“, „Vorhandener Lebenslauf öffnen“ oder „Lebenslauf speichern“ aufgerufen.

#### 3.4.1.2.2 Neuen Lebenslauf anlegen

- Beschreibung: Durch dieses UseCase erhält der Akteur die Möglichkeit, einen neuen Lebenslauf anzulegen. Dafür muss der Speicherort und der Name des Profils angegeben werden.
- Akteur: Benutzer.
- Vorbedingungen: Der Benutzer muss das Programm gestartet haben.
- Nachbedingungen: Es wird eine Profildatei angelegt und anschließend geöffnet.
- Szenario:
  - Der Akteur betätigt einen Button zum Anlegen eines neuen Lebenslaufes.
  - Das System öffnet einen Dialog, in dem der Zielspeicherort und der Name des Profils eingegeben werden muss.
  - Der Akteur wählt einen Namen und ein Zielspeicherort aus und bestätigt seine Eingaben durch Anklicken eines OK-Button.
  - Das System legt in dem angegebenen Ordner eine Datei mit dem eingegebenen Namen an und öffnet sie zum Eintragen der Lebenslaufdaten.

#### 3.4.1.2.3 Vorhandenen Lebenslauf öffnen

- Beschreibung: Eine bereits angelegte Profildatei kann geöffnet und der darin enthaltene Lebenslauf weiterverarbeitet werden.
- Akteur: Benutzer.
- Vorbedingungen: Es muss bereits eine Profildatei angelegt worden sein.
- Nachbedingungen: Der geänderte Lebenslauf kann gespeichert werden.
- Szenario:
  - Der Akteur muss den Button zum Öffnen einer Datei betätigen.
  - Das System öffnet ein Dialog, in dem die zu öffnende Profildatei ausgewählt werden muss.
  - Der Akteur wählt eine Profildatei aus und bestätigt durch Klick auf einen OK-Button.
  - Das System versucht die Profildatei zu laden und zeigt den Lebenslauf an. Bei einem auftretendem Fehler wird der Akteur darüber informiert und das Laden abgebrochen.

#### 3.4.1.2.4 Lebenslauf speichern

- Beschreibung: Ein eingegebener Lebenslauf kann gespeichert werden.
- Akteur: Benutzer.
- Vorbedingungen: Es muss bereits eine Profildatei angelegt oder eine bestehende geladen worden sein.
- Szenario:
  - Der Akteur klickt auf den Speichern-Button.
  - Das System speichert die Datei in dem Verzeichnis, in dem sich die Profildatei befindet und informiert den Akteur über den Erfolg der Speicherung.

#### 3.4.1.2.5 Lebenslaufdaten eingeben

- Beschreibung: Der Akteur soll durch Formulare die Möglichkeit erhalten, die Lebenslaufdaten einzugeben.
- Akteur: Benutzer.
- Vorbedingungen: Es muss bereits eine Profildatei angelegt und geöffnet sein.
- Nachbedingungen: Die Eingabe wird validiert.

- Szenario:
  - Der Akteur gibt in vordefinierten Feldern die Lebenslaufdaten ein.
  - Das System validiert die Eingaben und zeigt dem Akteur mögliche Fehlerquellen an.

#### **3.4.1.2.6 Validierung der Eingaben**

- Beschreibung: Die eingegebenen Daten werden auf ihre Korrektheit hin überprüft. Ferner wird überprüft, ob alle Pflichtfelder ausgefüllt worden sind.
- Akteur: Benutzer.
- Vorbedingungen: Es müssen Daten im aktuell geöffnetem Lebenslauf eingegeben werden.
- Nachbedingungen: Bei eventuellen Fehlern sollen diese dem Akteur durch geeignete Warnhinweise kenntlich gemacht werden.

#### **3.4.1.2.7 Anzeige der Fehlerquellen**

- Beschreibung: Fehler bei der Eingabe werden sofort angezeigt, damit ein Akteur später nicht zur Fehlerquelle springen muss, sondern gleich darauf reagieren kann.
- Akteur: Benutzer.
- Vorbedingungen: Bei der Eingabe kann ein Feld nicht validiert werden, oder ein Pflichtfeld wurde ausgelassen.
- Nachbedingungen: Dem Benutzer wird in dem Formular der Fehler angezeigt.

#### **3.4.1.2.8 Lebenslaufdaten exportieren**

- Beschreibung: Ein Lebenslauf soll in ein gängiges Format exportiert werden können.
- Akteur: Benutzer.
- Vorbedingungen: Es muss bereits ein ausgefüllter und validierter Lebenslauf zum Exportieren vorliegen.
- Nachbedingungen: Es wird eine spezielle Funktion zum Exportieren aufgerufen. (Siehe folgender UseCase)

#### 3.4.1.2.9 Export-Funktion

- Beschreibung: Der UseCase wird angewandt, wenn ein Lebenslauf exportiert werden soll und eine entsprechende Exportmöglichkeit ausgewählt wurde. Dieser UseCase ist abstrakt und ist mit den UseCases „MS-Word-Datei exportieren“, „HTML-Datei exportieren“ und „XML-Datei exportieren“ verbunden.
- Akteur: Benutzer.
- Vorbedingungen: Es muss bereits ein Lebenslauf vorliegen und der Akteur muss auf den Export-Button geklickt haben.
- Nachbedingungen: Die ausgewählte Exportfunktion wird ausgeführt.

#### 3.4.1.2.10 MS-Word-Datei exportieren

- Beschreibung: Der UseCase soll es ermöglichen, einen Lebenslauf in eine MS-Word-Datei zu exportieren.
- Akteur: Benutzer.
- Vorbedingungen: Es muss bereits ein Lebenslauf erstellt worden sein und der Akteur muss die entsprechende Funktion ausgewählt haben.
- Nachbedingungen: Der Lebenslauf wird als MS-Word-Datei exportiert.
- Szenario:
  - Der Akteur hat einen fertigen Lebenslauf und klickt auf die MS-Word-Export-Funktion. Anschließend kann der Benutzer den Lebenslauf in MS-Word weiterverarbeiten oder drucken.
  - Das System konvertiert den Lebenslauf als MS-Word-Datei und speichert sie auf der Festplatte. Mögliche Fehler werden dem Akteur mitgeteilt.

#### 3.4.1.2.11 HTML-Datei exportieren

- Beschreibung: Der UseCase stellt den Export eines Lebenslaufes als HTML-Datei dar.
- Akteur: Benutzer.
- Vorbedingungen: Der Akteur muss einen Lebenslauf erstellt und die entsprechende Funktion angeklickt haben.
- Nachbedingungen: Der Lebenslauf wird als eine HTML-Datei konvertiert und auf der Festplatte gespeichert.

- Szenario:
  - Der Akteur hat einen fertig erstellten Lebenslauf und klickt auf die HTML-Export-Funktion. Anschließend kann der Lebenslauf im Internet veröffentlicht oder per E-Mail versendet werden.
  - Das System konvertiert den Lebenslauf als HTML-Datei und speichert sie auf der Festplatte. Der Akteur wird dabei über mögliche Fehler informiert.

#### **3.4.1.2.12 XML-Datei exportieren**

- Beschreibung: Dieser UseCase behandelt das Exportieren eines Lebenslaufes als eine standardisierte XML-Datei.
- Akteur: Benutzer.
- Vorbedingungen: Der Akteur muss einen Lebenslauf erstellt und auf den entsprechenden Button zum Export geklickt haben.
- Nachbedingungen: Der Lebenslauf wird als die angeklickte XML-Datei konvertiert und auf der Festplatte gespeichert.
- Szenario:
  - Der Akteur erstellt einen Lebenslauf und klickt anschließend auf den entsprechenden Button.
  - Das System konvertiert den Lebenslauf als XML-Datei und speichert sie auf der Festplatte. Bei einem möglichen Fehler wird der Akteur darüber informiert.

### 3.4.1.3 Anwendungsfalldiagramm

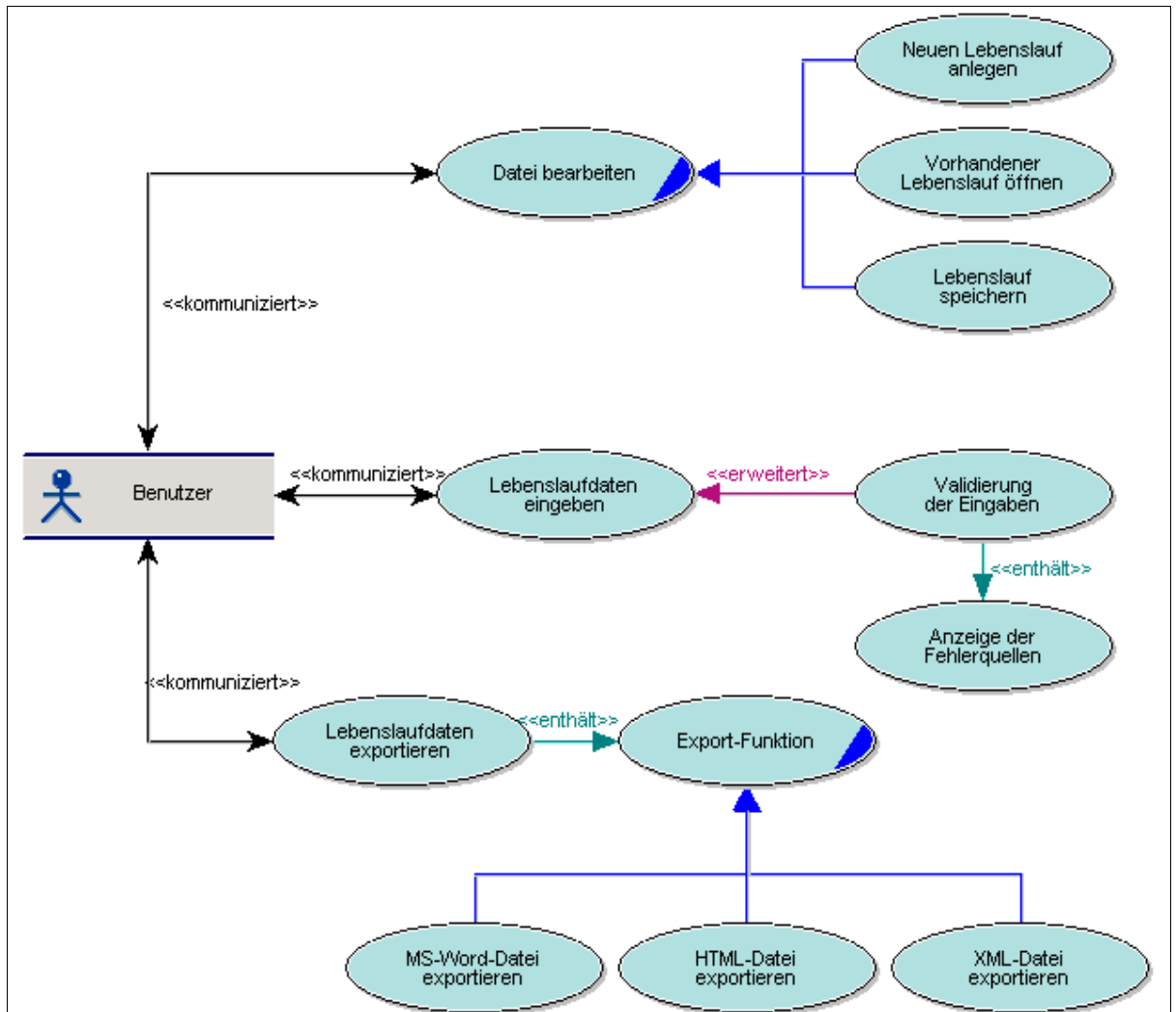


Abbildung 12: Anwendungsfalldiagramm

Abbildung 12 zeigt ein Anwendungsfalldiagramm für die Bearbeitung eines Lebenslaufes. Es gibt nur einen Akteur, dieser ist in der Rolle eines Benutzers im linken Teil des Bildes dargestellt. Der Akteur kann drei UseCases auswählen: „Datei bearbeiten“, „Lebenslaufdaten eingeben“ und „Lebenslaufdaten exportieren“. Ein UseCase wird dabei durch ein ovales Element symbolisiert.

„Datei bearbeiten“ stellt eine Generalisierung der UseCases „Neuen Lebenslauf anlegen“, „Vorhandener Lebenslauf öffnen“ und „Lebenslauf speichern“ dar. Der UseCase ist deshalb abstrakt und wird durch eine dunkelblaue Markierung in dem ovalen Element gekennzeichnet.

Der UseCase „Lebenslaufdaten eingeben“ wird durch den UseCase „Validierung der Eingaben“ erweitert (engl.: „extend“). Die Validierung der Daten ist bedingungsabhängig und wird nur aufgerufen, wenn Daten eingegeben wurden. Je nachdem, ob eine Validierung fehlschlägt oder ein Pflichtfeld nicht ausgefüllt wurde, wird der Akteur darüber in Kenntnis gesetzt. Dabei wird der UseCase „Anzeige der Fehlerquellen“ benutzt. Die Anzeige der Fehler kann dabei als eigenes Modul angesehen werden, da sie auch von anderen UseCases eingesetzt werden könnte. Aus diesem Grund besteht zwischen den beiden UseCases eine „enthält-Beziehung“ (engl.: „include“).

Ferner hat der Akteur noch die Möglichkeit Lebenslaufdaten zu exportieren. Das dazu benötigte UseCase heißt „Lebenslaufdaten exportieren“, es enthält das UseCase „Export-Funktion“. Diese Funktion ist wieder als eigenes Modul anzusehen, da es von verschiedenen UseCases aufgerufen werden könnte. Aus diesem Grund besteht hier wieder eine „enthält-Beziehung“. Das UseCase „Export-Funktion“ ist abstrakt und generalisiert die einzelnen Funktionen zum Exportieren des Lebenslaufes. Die UseCases „MS-Word-Datei exportieren“, „HTML-Datei exportieren“ und „XML-Datei exportieren“ sind mit dem UseCase „Export-Funktion“ verbunden und stellen jeweils eine eigene Art zum Export der Daten dar.

### 3.4.1.4 Aktivitätsdiagramm

Die folgenden zwei Aktivitätsdiagramme zeigen die Reihenfolge bei der Eingabe der Profildaten, sowie den Export der Daten in ein anderes Format.

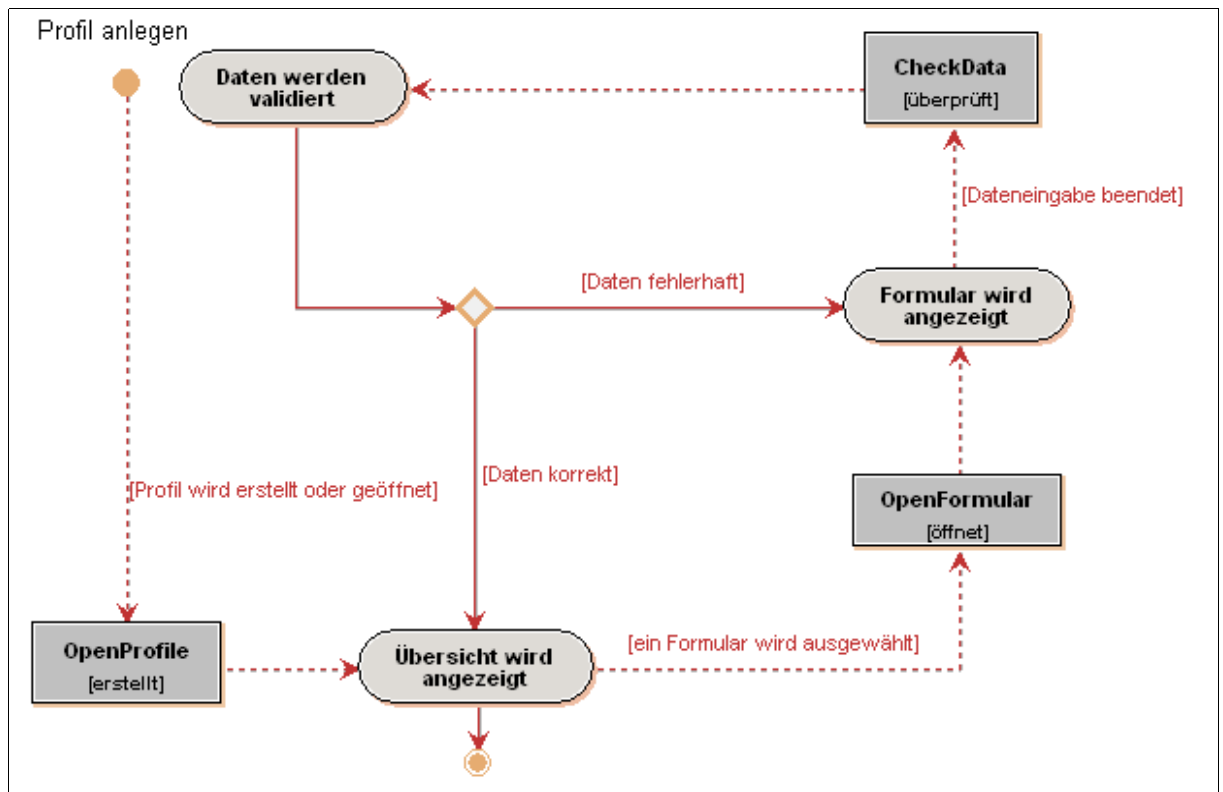


Abbildung 13: Aktivitätsdiagramm Profil anlegen oder bearbeiten

Der Startpunkt in der Abbildung wird durch einen ausgefüllten Kreis dargestellt. Dieser befindet sich in der linken oberen Ecke des Bildes. Von dem Startpunkt aus geht ein Objektfluss zu der Aktion „Übersicht wird angezeigt“. Damit dieser Objektfluss durchlaufen wird, muss zunächst eine Profildatei angelegt oder geöffnet werden. Das Objekt „OpenProfile“ mit dem Zustand „erstellt“ wird angelegt und sorgt dafür, dass das Hauptfenster erzeugt und die Profildatei geladen wird. In dem Hauptfenster wird eine Übersicht mit den Bestandteilen eines Lebenslaufes angezeigt.

Aktionen stellen die Basiselemente eines Aktivitätsdiagramms dar und zeigen, dass eine Verarbeitung stattfindet. Im obigen Aktivitätsdiagramm sind die Aktionen als ovale Elemente symbolisiert.

Von der Übersichtsseite aus können die einzelnen Formulare zum Eintragen der Lebenslaufdaten geladen werden, wobei es für jede Gruppe von Lebenslaufdaten ein separates Formular zum Eintragen gibt. Das Öffnen eines Formulars wird durch das

Objekt „OpenFormular“ dargestellt. Es folgt die Anzeige des entsprechenden Formulars. Die eingegebenen Daten werden nach der Erfassung validiert, wofür das Objekt „CheckData“ zuständig ist. Die Aktion „Daten werden validiert“ führt die Validierung der Daten durch. Von dieser Aktion geht ein Entscheidungsknoten aus. Konnten die Daten validiert werden, so ist alles in Ordnung und der Benutzer gelangt auf die Übersichtsseite zurück. Wenn ein Fehler in den Daten festgestellt wurde, so wird das entsprechende Formular erneut geladen und die Fehlerquelle markiert.

Nachdem die Daten für das ausgewählte Formular alle korrekt sind, ist dieser Teil des Lebenslaufes beendet und es kann ein weiteres Formular geöffnet und bearbeitet werden. Ein Endzustand ist erreicht, wenn ein Benutzer keine weiteren Daten mehr eingeben möchte. Der Endzustand kann von der Aktion „Übersicht wird angezeigt“ erreicht werden.

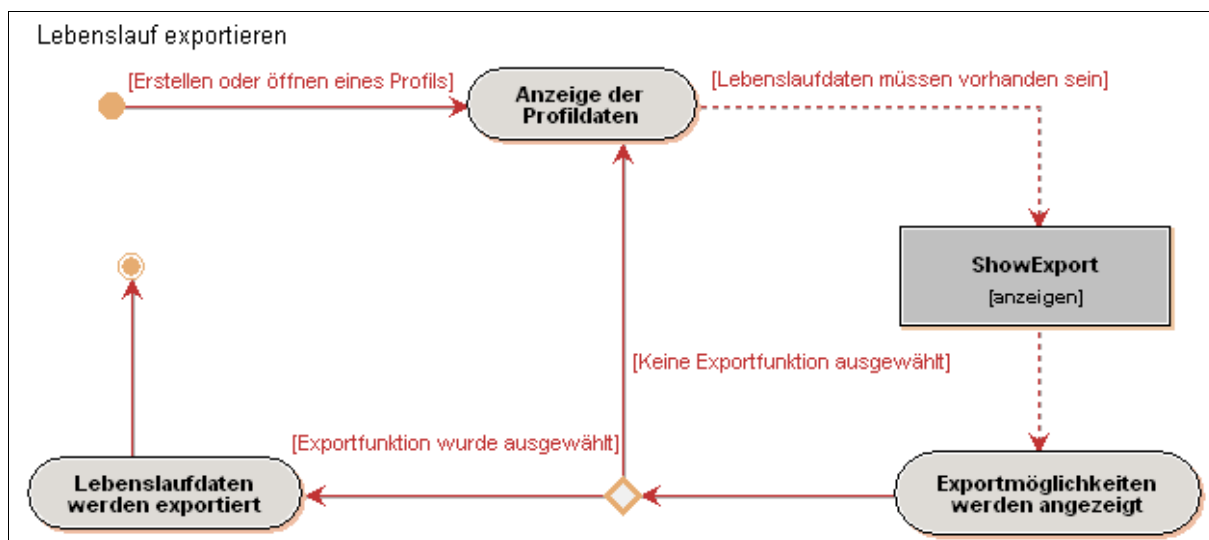


Abbildung 14: Aktivitätsdiagramm Lebenslauf exportieren

In der obigen Abbildung wird das Vorgehen beim Export von Daten dargestellt. Die Aktion „Anzeige der Profildaten“ beinhaltet die aus der vorherigen Abbildung bekannte Übersicht, sowie die einzelnen Formulare. Nachdem ein Profil erstellt oder ein bereits existierendes geöffnet wurde, können die Daten in den entsprechenden Formularen angezeigt werden. In dem zu entwickelnden Programm werden dem Benutzer verschiedene Exportformate zur Verfügung stehen. Vor einem Export der Profildaten werden die zur Verfügung stehenden Formate angezeigt, was in dem Diagramm durch das Objekt „ShowExport“ und der Aktion „Exportmöglichkeiten werden angezeigt“ symbolisiert wird.

Wird keine Exportfunktion ausgewählt, so gelangt der Benutzer zurück auf eine Übersichtsseite und die Profildaten werden wieder angezeigt. Wurde allerdings ein Format ausgewählt, so werden die Profildaten in dem ausgewählten Format konvertiert und gespeichert. Danach ist ein Endzustand erreicht.

### 3.4.2 Klassendiagramm

Im folgenden Verlauf wird anhand von zwei Klassendiagrammen der Aufbau der Anwendung beschrieben. In dem ersten Diagramm ist die Verbindung zwischen dem Model und der View, also der Datenhaltung und der Bedienoberfläche, visualisiert. Sogenannte Controller helfen bei der Verbindung zwischen Model und View. In dem zweiten Diagramm ist das Model-Konzept noch einmal detaillierter abgebildet. Für eine bessere Übersicht in den Diagrammen werden die Verbindungen innerhalb der Datenklassen nur für die Klassen der persönlichen Angaben betrachtet.

#### 3.4.2.1 Klassendiagramm Model-View-Controller

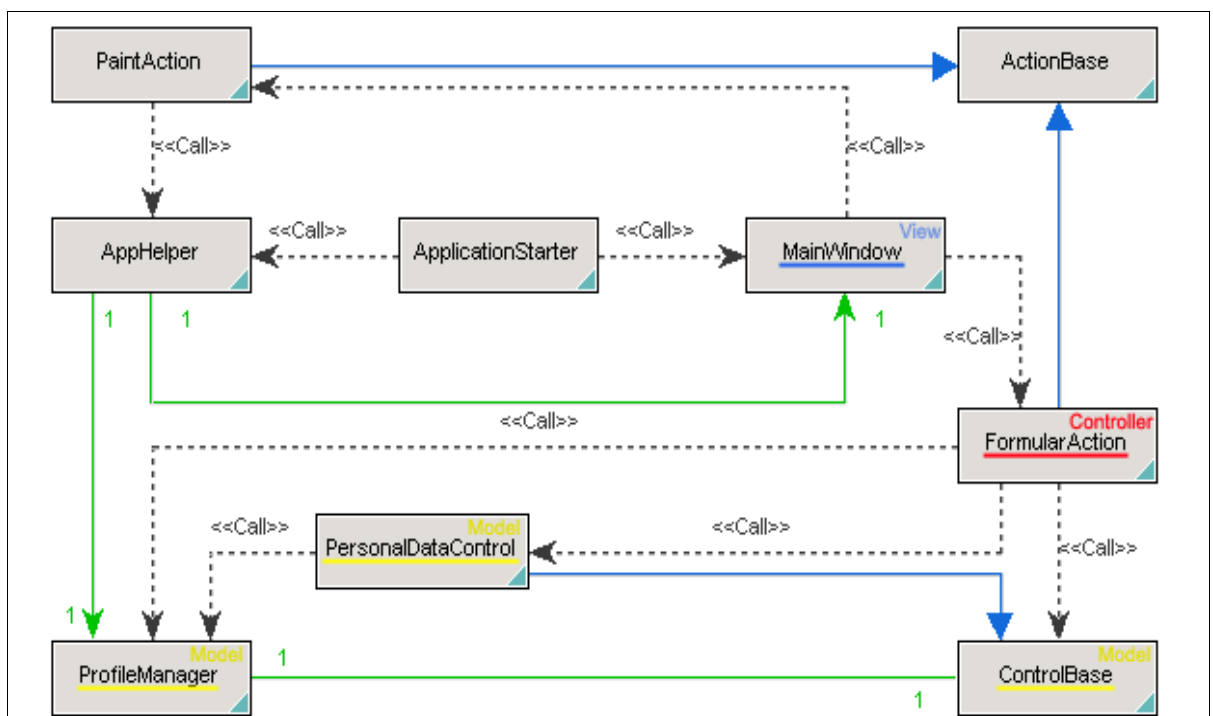


Abbildung 15: Klassendiagramm Model-View-Controller

Abbildung 15 beinhaltet ein Klassendiagramm, das die Verbindung zwischen der Benutzerschicht und der Datenhaltung zeigt. Die elementare Klasse für die Benutzerschicht heißt `MainWindow` und besteht aus einem `WindowsForm`. Das `MainWindow` ist in

der Abbildung als View markiert und blau unterstrichen. Es stellt die Benutzeroberfläche der Anwendung dar und wird beim Start von der Klasse `ApplicationStarter` erzeugt. Das erzeugte Objekt wird an die Klasse `AppHelper` übergeben und anschließend aufgerufen. Danach bleibt das `MainWindow` solange bestehen, bis der Benutzer die Anwendung wieder beendet. Die Verbindungen zwischen Klassen werden in dem Diagramm durch gepunktete Linien dargestellt und mit `call` beschriftet. Der Pfeil zeigt dabei zu der aufgerufenen Klasse.

In der Abbildung ist zu sehen, dass das `MainWindow` die Klassen `FormularAction` und `PaintAction` aufruft. Beide Klassen dienen dazu, Events zu behandeln, die in dem `MainWindow` auftreten, sowie weitere Methoden für den Umgang mit den Formulardaten bereitzustellen. Diese Klassen, vor allem die Klasse `FormularAction`, die in dem Diagramm rot unterstrichen und als Controller gekennzeichnet wurde, dienen als Controller innerhalb der Anwendung. Sie sorgen dafür, dass die Benutzeroberfläche keinen eigenen Code mehr ausführen muss und alles an zuständige Klassen weiter delegiert werden kann.

Diese Action-Klassen leiten alle von der Basisklasse `ActionBase` ab. Die Klasse `PaintAction` stellt Methoden zum Zeichnen der Hintergrundfarben bereit, die beim Zeichnen des `MainWindows` aufgerufen werden. Die Klasse `FormularAction` wird eingesetzt, um die ausgewählten Formulare zu laden. Ein Formular, beispielsweise zum Eintragen der persönlichen Angaben, wird als `UserControl` bereitgestellt und dem `MainWindow` über die Klasse `AppHelper` hinzugefügt.

`FormularAction` legt ein eigenes `ProfileManager`-Objekt an und weist diesem einen `ProfileManager` zu, der im `AppHelper` gespeichert wird. Die Aufgaben des `ProfileMangers` werden im weiteren Verlauf noch genauer beschrieben. Als weitere Verbindung ruft `FormularAction` noch die Klasse `ControlBase` auf. Dies geschieht beim Speichern der Daten, da über `ControlBase` auf die Formulare zugegriffen werden kann. Wird ein neues Formular geladen, so wird es durch `FormularAction` erzeugt und dem `MainWindow`-Objekt in `AppHelper` hinzugefügt. Dies wird durch die Verbindung mit dem `PersonalDataControl` deutlich gemacht. Alle Formulare, in dem Diagramm ist als Beispiel das bereits erwähnte `PersonalDataControl` zu sehen, leiten

von der Basisklasse `ControlBase` ab. Diese Klassen, sowie der `ProfileManager`, gehören zu dem Namensraum `Data` und werden in einem separaten Diagramm noch einmal genauer dargestellt. Sie dienen innerhalb der Anwendung als Datenhaltungs-klassen und werden dementsprechend in dem Diagramm als Model-Klassen gekennzeichnet und gelb unterstrichen.

### 3.4.2.2 Klassendiagramm Model-Konzept

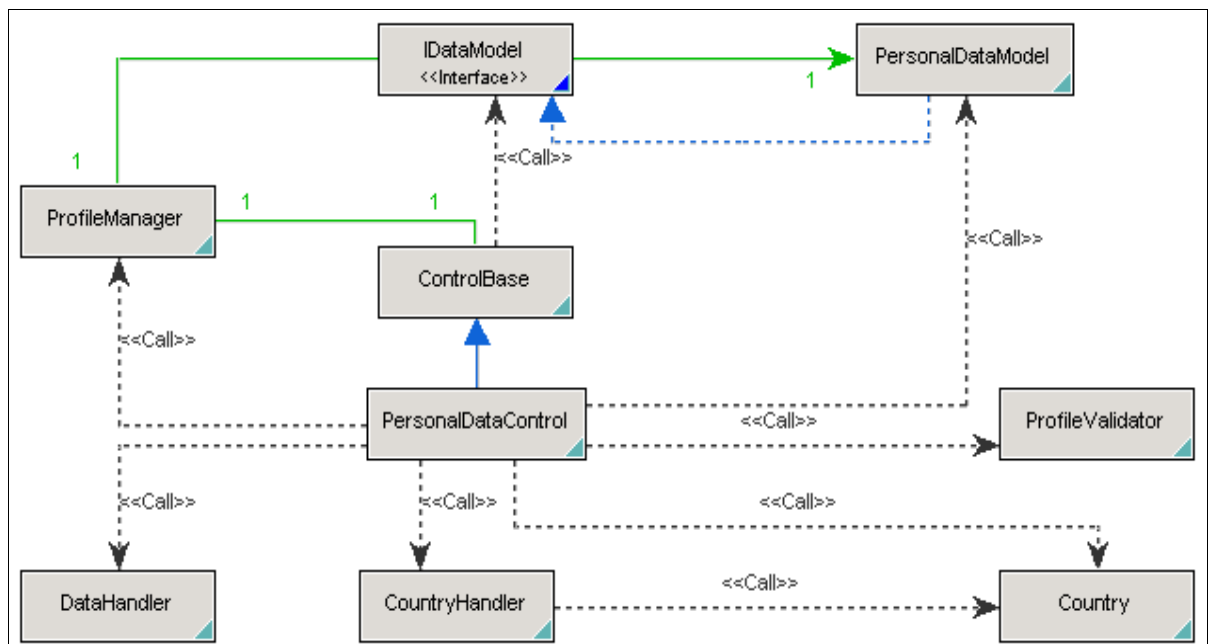


Abbildung 16: Klassendiagramm Model-Konzept

In diesem Diagramm sind die Klassen aus Abbildung 15, `ControlBase` und `ProfileManager`, wieder zu finden. `ControlBase` stellt eine Basisklasse dar, von der alle Formulare abgeleitet werden. Obligatorisch für die Formulare ist in Abbildung 16 die Klasse `PersonalDataControl` zu sehen. Für jede Kategorie der Lebenslaufdaten gibt es eine einzelne Control- und Datenklasse. Aus Platzgründen sind in dem Diagramm lediglich die Klassen für die persönlichen Angaben anhand der Klasse `PersonalDataModel` und dem UserControl `PersonalDataControl` dargestellt. Die Datenhaltungs-klasse, `PersonalDataModel`, implementiert das Interface `IDataModel`, welches von den übrigen Datenklassen ebenfalls benutzt wird. Das Interface sorgt für ein einheitliches Erscheinungsbild der Datenklassen.

Zum Überprüfen der Eingaben und Speichern der Daten wird aus dem `FormularAction` eine `CheckData`-Methode von `ControlBase` aufgerufen. Dieser Methode wird eine Referenz zu einem `ProfileManager` übergeben, wodurch die Verbindung zwischen `ControlBase` und `ProfileManager` entsteht. Die Klasse `ControlBase` gibt diesen `ProfileManager` allerdings nur an die zuständige Klasse weiter. Die abgeleiteten `Control`-Klassen, zum Beispiel `PersonalDataControl`, benutzen das `ProfileManager`-Objekt und füllen es mit den in den Formularfeldern eingetragenen Daten. Die Felder werden zuvor durch die Klasse `ProfileValidator` auf ihre Gültigkeit hin überprüft. `PersonalDataControl` hat Verbindungen zu den Klassen `CountryHandler` und `Country`. Beide Klassen werden für die Anzeige der Länder benötigt. Anhand von den in .NET verfügbaren Ländernamen und Ländercodes werden entsprechende Listen im `CountryHandler` zum Befüllen von Auswahlmenüs bereitgestellt. Die Länder werden dabei als Objekte der Klasse `Country` angelegt. Somit können Informationen zu dem Ländernamen, dem ISO-Ländercode und der Sprachenbezeichnung in den Objekten innerhalb der Auswahlmenüs gespeichert werden.

Da `CountryHandler` und die Länderlisten auch in weiteren `Controls`, zum Beispiel bei der Angabe der Sprachen, verwendet werden, sind die Listen als statische Variablen gespeichert und werden bei Programmstart durch die Klasse `ApplicationStarter` angelegt. Die Klasse `DataHandler` wird von `PersonalDataControl` aufgerufen, um die Auswahlmenüs für das Geschlecht und den Familienstand zu füllen. Beide Menüs werden dynamisch von der Klasse `DataHandler` aus eine XML-Datei herausgeladen. Da die bereits erwähnte `CheckData`-Methode aus `ControlBase` überschrieben ist, wird ein Aufruf dieser Methode an `PersonalDataControl` weitergegeben. Innerhalb der überschriebenen Methode wird ein `PersonalDataModel`-Objekt angelegt und mit den Daten aus den Formularfeldern gefüllt. Dabei wird, wie bereits beschrieben, `ProfileValidator` zum Validieren aufgerufen. Danach werden die Daten dem übergebenen `ProfileManager` hinzugefügt. Da nur eine Referenz des Objektes übergeben wird, sind die eingetragenen Daten sofort in der Anwendung, bzw. in dem von dem `AppHelper` gehaltenen `ProfileManager`-Objekt, verfügbar.

### 3.4.3 Pluginkonzept

Damit die Anwendung im Anschluss an die Diplomarbeit weiterentwickelt und weitere Exportformate hinzugefügt werden können, ist dieser Teil als ein Pluginkonzept aufgebaut. Jede Exportfunktion besteht dabei aus einer eigenen Assembly-Datei. Eine Assembly ist eine Sammlung von mehreren Dateien und wird während der Laufzeit dynamisch eingebunden. Sie beinhaltet eine Klasse, die von dem System aufgerufen wird. Diese Klasse wird als Schnittstelle von der Anwendung bereitgestellt und jedes Plugin muss von dieser Schnittstelle abgeleitet werden.

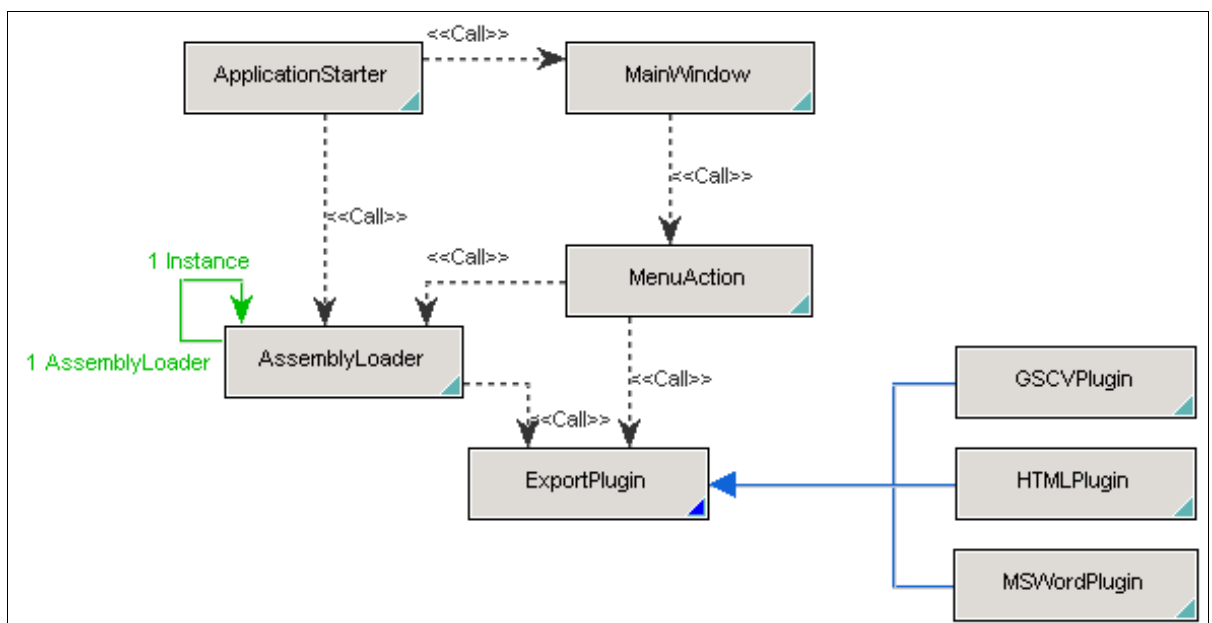


Abbildung 17: Klassendiagramm Pluginkonzept

In der Abbildung ist das Prinzip des Aufrufes eines Plugins zu sehen. Die aus den vorangehenden Diagrammen bekannten Klassen `ApplicationStarter` und `Main-Window` spielen dabei eine wesentliche Rolle. Bei Programmstart wird in der Klasse `ApplicationStarter` wieder ein `MainWindow`-Objekt erzeugt und das Fenster geöffnet. Außerdem besitzt die Klasse nun noch eine Verbindung zu der Singleton-Klasse<sup>1</sup> `AssemblyLoader`. Während der Initialisierung der Anwendung wird eine Methode aus der Klasse `AssemblyLoader` aufgerufen und es wird nach vorhandenen Plugins gesucht. Die gefundenen DLL-Dateien (jede Assembly ist eine DLL-Datei) müssen dabei vom Typ `ExportPlugin` sein. Ist dies der Fall, wird die Assembly geladen und in einem `Dictionary` gespeichert. Als Schlüssel dient dabei der Dateiname der

<sup>1</sup> Singleton ist ein Design-Pattern und stellt sicher, dass von dem Objekt nur eine Instanz in der Anwendung existiert

Assembly. Ein Export der Daten erfolgt über das `MainWindow`. Wird ein entsprechender Aufruf in dem Hauptfenster getätigt, wird eine Methode in der Klasse `MenuItem` aufgerufen. Diese Methode holt aus dem `AssemblyLoader` die entsprechende Assembly und ruft die darin enthaltene `ExportPlugin`-Klasse auf. Diese Klasse dient als Schnittstelle zwischen der Anwendung und den Plugins. Ein Plugin muss von dieser Klasse ableiten, um eine Export-Klasse darzustellen. In dem Diagramm sind die Klassen `MSWordPlugin`, `HTMLPlugin` und `GSCVPlugin` als Plugins zu sehen.

### 3.4.4 Sequenzdiagramm

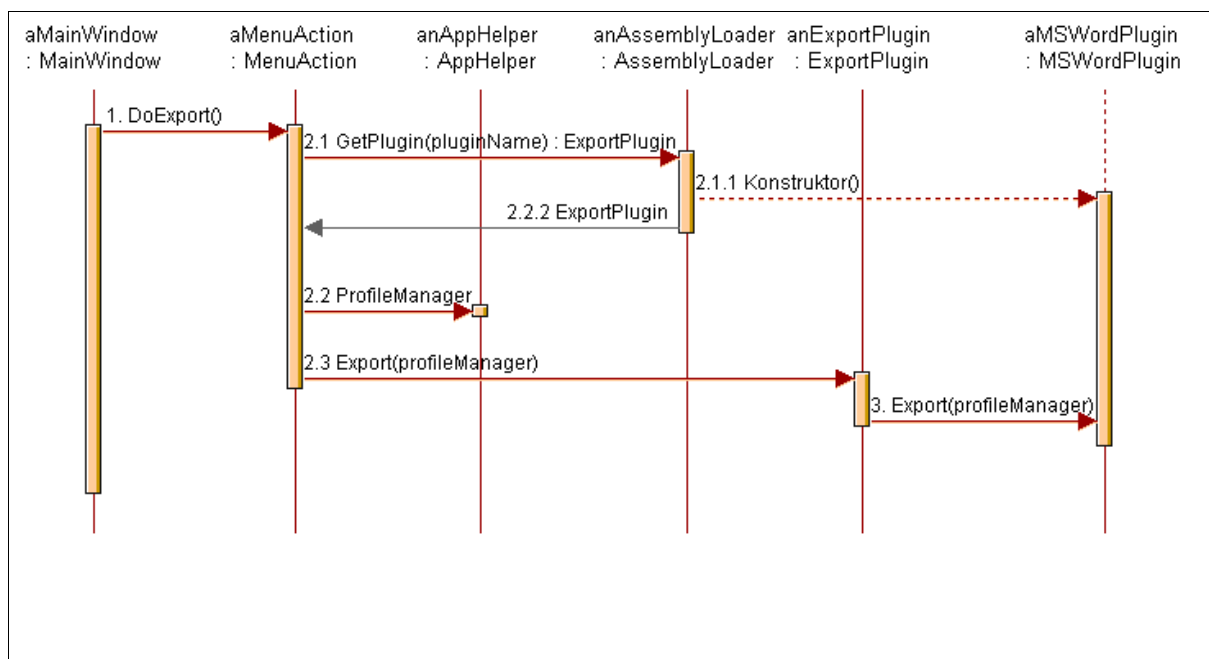


Abbildung 18: Sequenzdiagramm Plugin-Aufruf

In dem vorangehenden Sequenzdiagramm ist die Ablauffolge beim Aufruf eines Plugins dargestellt. Bei dem Diagramm wird angenommen, dass ein Profil bereits angelegt und mit Daten gefüllt wurde, damit diese exportiert werden können. Das Diagramm zeigt einen zeitlichen Ablauf beim Export dieser Daten als eine MS-Word-Datei.

Der Anfang des Verlaufes wird aus der Klasse `MainWindow` heraus fabriziert. Von hier können die Daten exportiert werden, indem die Methode `DoExport` aufgerufen wird.

`DoExport` ruft die Methode `GetPlugin` aus dem `AssemblyLoader` auf und übergibt den Namen der Datei, welcher in dem entsprechenden Menüeintrag hinterlegt ist. Anhand des Namens wird in dem `AssemblyLoader` aus einem, durch die Klasse `ApplicationStarter` bei der Initialisierung der Anwendung heraus generierten, `Dictionary` die entsprechende Assembly geladen und erzeugt. Dazu wird in Punkt 2.1.1 ein Konstruktor aufgerufen, der ein `MsWordPlugin`-Objekt erzeugt. Dieses Objekt wird als ein `ExportPlugin`-Objekt zurück an `MenuItem` geliefert. Das `ExportPlugin`-Objekt besitzt eine Methode `Export`, welcher ein `ProfileManager`-Objekt übergeben wird. Das `ProfileManager`-Objekt wird aus dem `AppHelper` geholt und enthält die gespeicherten Profildaten. In Punkt 3 wird letztendlich die Methode `Export` in der Klasse `MSWordPlugin` aufgerufen und ein MS-Word-Dokument mit den Lebenslaufdaten erstellt.



## 4 IMPLEMENTIERUNG

Die Implementierung der Anwendung erfolgte mit Microsoft Visual Studio 2005 und der Programmiersprache C#. Der Quellcode wurde auf verschiedene „Unterprojekte“ verteilt, damit die Anwendung anpassbar an neue oder geänderte Anforderungen bleibt. Jedes „Unterprojekt“ stellt dabei eine eigene Assembly dar, die zur Laufzeit geladen wird. Im weiteren Verlauf erfolgen eine Beschreibung der an der Anwendung beteiligten Projekte und deren genauen Aufgaben. Danach wird die Projektstruktur beschrieben, bevor ein Einblick in verschiedene Klassen anhand von Quellcodebeispielen gegeben wird. Dabei werden elementare Methoden der Anwendung erläutert, wie zum Beispiel die beteiligten Klassen während eines Speichervorgangs.

Es folgt eine Anleitung zur Bedienung des entwickelten Programms. Der Startvorgang und die ersten Schritte werden mit Hilfe von Abbildungen dargestellt und erläutert. Die implementierten Export-Funktionen werden vorgestellt und beschrieben. Zum Schluss dieses Kapitels erfolgt ein Ausblick auf zukünftige Erweiterungen, die im Rahmen der Diplomarbeit nicht mehr umgesetzt werden konnten oder alternativ noch hinzugefügt werden könnten.

### 4.1 Projekte und Unterprojekte

Es gibt ein Hauptprojekt, in dem das Fenster der Anwendung (Klasse `MainWindow`) und dessen Events definiert sind. Daneben befindet sich in dem Hauptprojekt noch eine Klasse Namens `AssemblyLoader`, die das Laden der Plugins übernimmt. Die Klasse `ApplicationStarter` steht für den Einstiegspunkt der Anwendung, sie besitzt eine `Main`-Methode, in der das `MainWindow` initiiert und aufgerufen wird. Die Klasse `MainWindow` stellt die Benutzeroberfläche des Programms dar, von der alle Benutzeraktionen ausgehen.

Zu den „Unterprojekten“ gehören die Projekte `Components`, `Utils`, `Plugin` und `Data`. Jedes dieser Projekte kann, wenn es in einem neuen Projekt als Verweis eingebunden wird, durch den entsprechenden Namensraum benutzt werden. Für das Projekt `Utils` wäre das der Namensraum `Skill_Editor.Utils`. Das Projekt `Components` enthält eigens für die Anwendung kreierte `UserControls`.

### 4.1.1 Projekt Utils

Das Projekt `Utils` beinhaltet eine Vielfalt von Hilfsklassen für die Anwendung. So stellt es eine Klasse `Exceptions` bereit, in der eigene `Exception`-Klassen für das Projekt definiert sind. Innerhalb von `Utils` gibt es den Namensraum `Meanwhile`, in dem sich eine `Meanwhile`-Klasse befindet. Diese Klasse stellt ein `WindowsForm` bereit, mit dessen Hilfe der aktuelle Statuszustand der Anwendung angezeigt werden kann. Unter anderem wird diese Klasse beim Start der Anwendung benutzt, um dem Benutzer anzuzeigen, was gerade passiert, bzw. wie weit der Fortschritt des Ladevorganges ist. Dies ist in der folgenden Abbildung zu sehen:

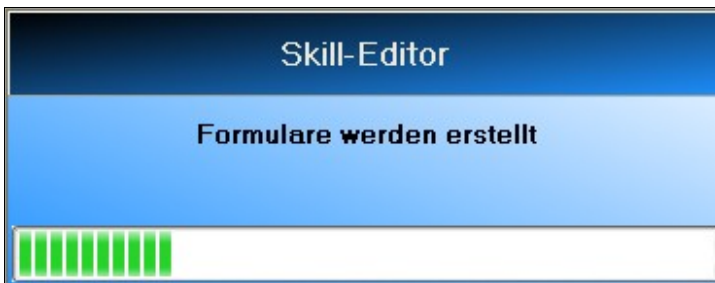


Abbildung 19: Anzeige eines `Meanwhile`-Forms

Die Klasse `Meanwhile` ist als eine Singleton-Klasse implementiert, wodurch gegeben ist, dass es von dieser Klasse nur eine Instanz in der Anwendung geben kann. Folgender Aufruf würde bei obiger Abbildung bewirken, dass die Statusanzeige einen Schritt weiterläuft und „Hello World“ als Text angezeigt wird:

```
Meanwhile.Instance.SetStatus("Hello World");
```

In dem Projekt `Utils` befinden sich noch weitere Hilfsklassen, wie zum Beispiel ein `Logger`. Mit Hilfe dieses `Loggers` können Warnungen, Infos, Debug-Ausgaben oder Fehler in eine Textdatei geschrieben werden, was bei der Suche nach möglichen Fehlerursachen behilflich sein kann. In einem Namespace `XML` werden verschiedene Klassen zum Arbeiten mit XML-Daten bereitgestellt. So gibt es einen `XmlValidator` und einen `XmlTransformer`. Der `XmlValidator` wird benutzt, um eine XML-Datei gegen ein XML-Schema zu validieren. Der `XmlTransformer` kommt zum Einsatz, wenn eine XML-Datei mit Hilfe eines XML-Stylesheets transformiert werden soll.

In dem Namensraum `Messages` werden die Klassen `Messages`, `HelpMessages` und `DataMessages` bereitgestellt. Diese Klassen beinhalten statische `Strings`, die verwendet werden, um Texte an den Benutzer auszugeben. Die `Strings` werden an

zentraler Stelle definiert, damit eine spätere Globalisierung der Anwendung durchführbar ist. Für die Globalisierung bietet `Utils` auch noch einen Namensraum `Globalization` an, in dem sich Klassen für das Arbeiten mit Ländern und Sprachen befinden. Es gibt eine Klasse `Language`, eine Klasse `Country` und einen `CountryHandler`. Der `CountryHandler` lädt während des Programmstarts die im .NET Framework verfügbaren Ländernamen, Ländercodes und Sprachen und speichert diese in `Dictionaries` und `Collections` ab. Diese sind von den Klassen der Anwendung während der Laufzeit zugänglich und Länder, bzw. Sprachen können auf dem Bildschirm ausgegeben werden. Dieses Verfahren hat folgende Vorteile:

- Der Benutzer bekommt alle Sprachen und Länder angezeigt, die auch unter Windows verfügbar sind.
- Die auf dem Computersystem des Benutzers eingestellte Sprache wird angezeigt, eine Übersetzung der Ländernamen und Sprachen in eine andere Sprache entfällt.

Die Globalisierung wird an dieser Stelle also von dem .NET Framework übernommen. Die Länderlisten kommen zum Beispiel bei der Angabe von Nationalität oder Geburtsland, die Sprachen bei den Sprach-Kenntnissen zum Einsatz.

#### 4.1.2 Projekt Plugin

Das Projekt `Plugin` stellt einen Namensraum `ExportPlugin` bereit, in dem sich die Klasse `ExportPlugin` befindet. Jedes zur Laufzeit geladene Plugin muss von der `ExportPlugin`-Klasse erben. Die Klasse besitzt `Properties` und `Methoden`, mit denen die Kommunikation mit der Anwendung durchgeführt wird. Geladen werden die Plugins durch die bereits erwähnte Klasse `AssemblyLoader`. Während des Programmstarts werden alle Assemblies geladen und in einer statischen `Collection` bereitgestellt. Klickt ein Benutzer eine Export-Funktion an, so wird die entsprechende Assembly aus der `Collection` geholt und die Export-Methode des Plugins aufgerufen.

### 4.1.3 Projekt Data

In dem Projekt `Data` sind die Datenklassen beschrieben. Es beinhaltet eine Model-Schicht, die sich in dem Namensraum `Model` befindet. Innerhalb dieses Namensraumes gibt es für jeden Lebenslaufabschnitt einen weiteren Namensraum. So sind die Model-Klassen für die persönlichen Angaben im folgendem Namensraum enthalten:

---

```
Skill_Editor.Data.Model.PersonalData;
```

---

Die Anwendung lädt den ausgewählten Abschnitt in ein in `Data` definiertes `UserControl` mit dem Formular zum Eintragen der Daten. Folgender Namensraum beinhaltet diese Formulare:

---

```
Skill_Editor.Data.DataControls;
```

---

Das Formular für die persönlichen Angaben heißt `PersonalDataControl` und ist, wie alle Daten-Controls, von der Basisklasse `ControlBase` abgeleitet.

Innerhalb des Projektes `Data` befinden sich noch die Klassen `DataHandler`, `ProfileManager` und `Profile`, wobei `DataHandler` als Hilfsklasse für die Daten-Controls dient. In dieser Klasse werden zum Beispiel Methoden zum Laden der Schulabschlüsse bereitgestellt.

`ProfileManager` hingegen wird auch von der Hauptanwendung verwendet. Über diese Klasse laufen alle Aktionen mit den Lebenslaufdaten, wie zum Beispiel das Speichern einer Datei oder das Erstellen der einzelnen Daten-Controls. Der `ProfileManager` kann somit als eine Art Controller zwischen View und Model gesehen werden. Die Klasse `Profile` stellt eine interne Speicherung der Lebenslaufdaten dar. Diese Klasse ist serialisierbar, das bedeutet, dass alle Daten die sich in der Klasseninstanz befinden, automatisch durch das .NET Framework als XML-Datei auf der Festplatte gespeichert werden, sobald der Speichervorgang eingeleitet wird. Es ist kein weiterer Quellcode für die Erstellung der XML-Datei oder das Füllen der XML-Datei mit Daten notwendig. Auch das Laden eines Profils wird durch das .NET Framework übernommen, wodurch die geladenen Daten lediglich aus dem erstellten Profil in die entsprechenden Daten-Controls übergeben werden müssen.

## 4.2 Aufbau der Projektumgebung

Es gibt einen Ordner „\source“, in dem sich alle Projekte befinden. Der Ordner enthält einen Ordner „Skill-Editor“ mit dem Hauptprojekt der Anwendung. Alle anderen „Unterprojekte“ sind unterhalb des Ordners „\source“ angesiedelt. Die einzelnen Plugins befinden sich in dem Ordner „\source\PluginModule“.

Die Solution-Datei befindet sich ebenfalls in dem Ordner „\source“. Diese Datei enthält alle Informationen über das Projekt und deren Verweise. Wird diese Datei in Visual Studio geladen, werden auch automatisch alle darin enthaltenen Projekte geladen.

Bei der Kompilierung der Anwendung werden die benötigten DLL-Dateien in das „bin“-Verzeichnis der Anwendung kopiert. Für jedes „Unterprojekt“ wird eine separate DLL-Datei erzeugt. Nur die Hauptanwendung erstellt eine EXE-Datei, mit der die Anwendung gestartet werden kann. Neben den DLL-Dateien der Projekte werden auch die DLL-Dateien der Fremdkomponenten benötigt. Innerhalb des Projektes kommen Infragistics *[INFRA]* Komponenten zum Einsatz, die eine kommerzielle Erweiterung der .NET `UserControls` darstellen. Diese Komponenten, lizenziert auf die eWorks GmbH, werden in den `Data-Controls` eingesetzt.

## 4.3 Aufbau der Klassen

Die Anwendung besteht aus einer Vielzahl von Klassen, von denen einige im weiteren Verlauf vorgestellt werden. Es handelt sich dabei um elementare Klassen, die eine wesentliche Rolle in der Anwendung spielen. Es folgen Beispiele zum Ladevorgang der Plugins, zum Aufruf einer Export-Funktion, zum Laden von Ländern und das Speichern einer Profildatei. Es werden dabei allerdings nicht die gesamten Klassen vorgestellt, sondern nur vereinzelt Methoden daraus. Eine Ausnahme stellt dabei die Klasse `ObjectSerializer` dar, die im Kapitel 4.3.5 komplett dargestellt wird.

### 4.3.1 Klasse ApplicationStarter

Als erstes wird eine Methode der Klasse `ApplicationStarter` vorgestellt. Die `Main`-Methode der Klasse wird beim Start der Anwendung aufgerufen. Innerhalb der `Main`-Methode der Klasse `ApplicationStarter` werden weitere Methoden aufgerufen, die unterschiedliche Aufgaben während des Startvorgangs erledigen. Eine der aufgerufenen Methoden ist die `LoadPlugins`-Methode, die anhand eines Quellcodebeispiels genauer betrachtet wird:

---

```
(1) private static void LoadPlugins()
(2) {
(3)     //Plugins laden
(4)     AssemblyLoader.Instance.LoadPlugins();
(5)
(6)     //Menüeinträge generieren
(7)     foreach (ExportPlugin plugin in
(8)         AssemblyLoader.Instance.ExportPluginCollection)
(9)     {
(10)        if (plugin != null)
(11)        {
(12)            Meanwhile.Instance.SetStatus("Menüeintrag für " +
(13)                plugin.PluginName + " erstellen", 1);
(14)            Logger.Write(plugin.PluginName);
(15)            ToolStripMenuItem item = null;
(16)
(17)            //Menüeintrag mit oder ohne Bild erstellen
(18)            if (plugin.PluginMenuItemImage == null)
(19)                item = new ToolStripMenuItem(plugin.PluginName);
(20)            else
(21)                item = new ToolStripMenuItem(plugin.PluginName,
(22)                    plugin.PluginMenuItemImage);
(23)            item.Name = plugin.PluginName;
(24)
(25)            //Event registrieren
(26)            item.Click += new EventHandler(
(27)                new MenuAction().ExportPluginClicked);
(28)
(29)            //Menüeintrag hinzufügen
(30)            AppHelper.MainWindow.Menu_Export.DropDownItems.Add(item);
(31)        }
(32)    }
(33) }
```

---

#### *Quelltext 24: Die LoadPlugins-Methode des ApplicationStartes*

`LoadPlugins` wird aus der Methode `Initialize` heraus aufgerufen. Das eigentliche Laden der Assemblies erfolgt nicht in dieser Methode, sondern in der Klasse `AssemblyLoader`. In Zeile 4 wird die entsprechende Methode aus dieser Klasse aufgerufen. Der `AssemblyLoader` speichert alle geladenen Assemblies in einer `Collection`, die in Zeile 7, bzw. Zeile 8 in einer `foreach`-Schleife durchlaufen wird. In der Schleife wer-

den alle Export-Plugins untersucht und als Menüeintrag angelegt. In Zeile 12 bekommt der Benutzer via einem `Meanwhile` mitgeteilt, welches Plugin gerade geladen wird. In den Zeilen 18 bis 23 wird ein Menüeintrag erstellt, wobei in Zeile 19 ein Menüeintrag ohne Bild und in Zeile 21 ein Menüeintrag mit Bild erzeugt wird. Das Bild erscheint dabei links neben dem Bezeichner des Menüeintrags in der Menüleiste. In Zeile 23 wird der Name des Menüeintrags mit dem Namen des Plugins gleichgesetzt, damit bei einem Klick auf den Eintrag die entsprechende Assembly aufgerufen werden kann. Anschließend wird für den Menüeintrag ein `MouseClicked`-Event registriert, was in Zeile 26 und Zeile 27 zu sehen ist. Bei einem Klick auf den Menüeintrag wird die Methode `ExportPluginClicked` aus der Klasse `MenuItem` aufgerufen. In Zeile 30 wird der Menüeintrag zu dem Export-Menü hinzugefügt.

### 4.3.2 Klasse MenuItem

Die Klasse `MenuItem` beinhaltet eine Vielzahl von Methoden für die Abarbeitung von Events aus der Klasse `MainWindow` heraus. Unter anderem werden über diese Klasse die Aufrufe zum Speichern oder Export einer Profildatei bearbeitet. Der Codeausschnitt der folgenden Methode `ExportPluginClicked` zeigt die Abarbeitung beim Klick auf einen Plugin-Menüeintrag:

---

```
(1) public void ExportPluginClicked(object sender, EventArgs e)
(2) {
(3)     SaveFile();
(4)     foreach (ExportPlugin plugin in
(5)         AssemblyLoader.Instance.ExportPluginCollection)
(6)     {
(7)         if (plugin.PluginName.Equals(((ToolStripDropDownItem) sender).Name))
(8)         {
(9)             plugin.Export(ProfileManager.Instance.Profile);
(10)            break;
(11)        }
(12)    }
(13) }
```

---

#### *Quelltext 25: ExportPluginClicked aus Klasse MenuItem*

In dem Quellcodebeispiel 25 ist die Methode `ExportPluginClicked` verkürzt dargestellt. In der dritten Zeile wird die Methode `SaveFile` aufgerufen, die einen „Datei-öffnen“-Dialog öffnet, in dem der Zielspeicherort und der Dateiname angegeben werden kann, falls dies vorher noch nicht geschehen ist. Ist bereits ein Dateiname und -pfad vorhanden, wird die Profildatei zunächst gespeichert. Danach durchläuft eine

`foreach`-Schleife alle geladenen Plugins und untersucht den Namen eines jeden Plugins mit dem Namen des Menüeintrags. In dem vorherigen Beispiel war zu sehen, dass ein Menüeintrag den Namen eines Plugins erhält. Wird der Eintrag in Zeile 7 gefunden, so kann die Methode `Export` des ausgewählten Plugins in Zeile 9 aufgerufen und das Profil-Objekt, also die interne Speicherung des Profils, an das Plugin übergeben werden. Die Anwendung muss an dieser Stelle nicht wissen, um welches Plugin es sich handelt und was das Plugin genau mit den übergebenen Profildaten macht. Nachdem das Plugin beendet wurde kehrt die Anwendung in die `ExportPluginClicked`-Methode zurück, in der durch die `break`-Anweisung in Zeile 10 die `foreach`-Schleife verlassen und der Export-Vorgang somit beendet wird.

### 4.3.3 Klasse CountryHandler

Die Klasse `CountryHandler` gehört zu dem Projekt `Utils` und ist in dem Namensraum `Utils.globalization` untergebracht. Die Klasse hält verschiedene Methoden und Properties für eine Globalisierung der Anwendung bereit. So kann mit Hilfe dieser Klasse eine Liste mit Ländercodes, eine Liste mit Ländernamen oder aber eine Liste mit Sprachen inkl. Sprachencodes geholt werden. Wird eine dieser Listen zum ersten Mal aufgerufen, so ruft die Klasse `CountryHandler` die private Methode `FillCountryLists` auf. Diese Methode sorgt dafür, dass die in dem .NET Framework vorhandenen Sprachen geladen und in den Listen bereitgestellt werden. Im folgenden Quellcodebeispiel ist die Methode `FillCountryLists` verkürzt abgebildet:

---

```
(1) private static void FillCountryLists()
(2) {
(3)     SortedList<String, Country> tmpList =
(4)         new SortedList<string, Country>();
(5)     SortedList<String, String> tmpLanguageList =
(6)         new SortedList<string, string>();
(7)     foreach (CultureInfo i in
(8)         CultureInfo.GetCultures(CultureTypes.AllCultures &
(9)             ~CultureTypes.NeutralCultures))
(10)    {
(11)        try
(12)        {
(13)            RegionInfo region = new RegionInfo(i.LCID);
(14)
(15)            if (!_countryCodes.ContainsKey(region.DisplayName))
(16)            {
(17)                _countryCodes.Add(region.DisplayName,
(18)                    region.TwoLetterISORegionName);
(19)            }
(20)        } catch { }
(21)    }
(22) }
```

---

**Quelltext 26: Methode FillCountryLists aus Klasse CountryHandler**

Die Methode `FillCountryLists` holt in den Zeilen 7 bis 9 aus der `CultureInfo` alle Kulturen und durchläuft sie in einer `foreach`-Schleife. In Zeile 13 wird die Region anhand der `LCID`, der Bezeichnung der aktuellen Kultur, geladen. In Zeile 15 wird untersucht, ob der Name der Region bereits in dem `_countryCodes`-Dictionary vorhanden ist und wenn nicht, dann wird der Name der Region und der ISO-Bezeichner für den Ländercode in Zeile 17 und 18 in dem `Dictionary` aufgenommen.

### 4.3.4 Klasse ProfileManager

Die Klasse `ProfileManager` dient als Verbindung zwischen der Hauptanwendung, dem Projekt `Data`. Die Klasse beinhaltet viele Methoden für die Kommunikation zwischen Hauptanwendung und Profildaten und Profilformularen. Zum Beispiel kann hierüber die Profildatei gespeichert oder geladen werden. Zum Speichern wird die Methode `SaveProfile` aufgerufen, die wiederum die private Methode `Serialize` aufruft, die im folgenden Quellcodebeispiel dargestellt wird:

---

```
(1) private void Serialize()
(2) {
(3)     try
(4)     {
(5)         ObjectSerializer<Profile>.Save(this._profile,
(6)                                     this._profile.ProfilePath);
(7)     }
(8)     catch (Exception e)
(9)     {
(10)        throw (new ProfileException(DataMessages.ProfileNotSavedException
(11)                                    + "\n" + e.Message));
(12)    }
(13)    finally
(14)    {
(15)        Meanwhile.Instance.CloseMeanwhile();
(16)    }
(17) }
```

---

#### *Quelltext 27: Die Methode Serialize aus der Klasse ProfileManager*

Die eigentliche Aufgabe der Methode ist das Aufrufen der Methode `Save` aus der Klasse `ObjectSerializer` in Zeile 5. Beim Aufruf der Methode wird `ObjectSerializer` vom Typen `Profile` typisiert und bekommt die zu serialisierende Klasse und den gewünschten Pfad übergeben. Ansonsten wird nur noch Fehlerbehandlung innerhalb dieser Methode betrieben.

### 4.3.5 Klasse ObjectSerializer

Wie bereits in der Klasse `ProfileManager` zu sehen, wird `ObjectSerializer` beim Speichern der Profildaten, aber auch beim Laden der Profildaten verwendet. `ObjectSerializer` ist Bestandteil des Projekts `Utils` und befindet sich in dem Namensraum `Utils.Serializer`. Im folgenden Quellcodebeispiel wird die Klasse `ObjectSerializer` mit den beiden Methoden `Save` und `Load` dargestellt:

---

```
(1) public static class ObjectSerializer<T> where T
(2)           : class //T muss eine Klasse sein
(3) {
(4)     public static void Save(T serializeObject, String path)
(5)     {
(6)         XmlSerializer serializer = null;
(7)         StreamWriter writer = null;
(8)         try
(9)         {
(10)            serializer = new XmlSerializer(typeof(T));
(11)            writer = File.CreateText(path);
(12)            serializer.Serialize(writer, serializeObject);
(13)        }
(14)        catch (Exception e)
(15)        {
(16)            Logger.Write("Speichern fehlgeschlagen: " +
(17)                e.Message + " Source: " + e.Source, LogMode.Error);
(18)            throw e;
(19)        }
(20)        finally
(21)        {
(22)            if (writer != null)
(23)            {
(24)                writer.Close();
(25)                writer = null;
(26)            }
(27)            serializer = null;
(28)        }
(29)    }
(30)
(31)    public static T Load(String path)
(32)    {
(33)        T loadObject = null;
(34)        //mache nur was, wenn was übergeben wurde und die Datei
(35)        //auch existiert
(36)        if (!String.IsNullOrEmpty(path) && File.Exists(path))
(37)        {
(38)            XmlSerializer serializer = null;
(39)            StreamReader reader = null;
(40)
(41)            try
(42)            {
(43)                serializer = new XmlSerializer(typeof(T));
(44)                reader = File.OpenText(path);
(45)                loadObject = serializer.Deserialize(reader) as T;
(46)            }
(47)            catch (Exception e)
(48)            {
(49)                Logger.Write("Laden fehlgeschlagen: " + e.Message +
(50)                    " Source: " + e.Source, LogMode.Error);
(51)                throw e;
(52)            }

```

---

---

```
(53)     finally
(54)     {
(55)         if (reader != null)
(56)         {
(57)             reader.Close();
(58)             reader = null;
(59)         }
(60)         serializer = null;
(61)     }
(62) }
(63) else Logger.Write("Datei '" + path + "' existiert nicht.");
(64)
(65) return loadObject;
(66) }
(67) }
```

---

#### Quelltext 28: Die Klasse ObjectSerializer

In Zeile 1 und 2 ist zu sehen, dass nur ein typisierter Aufruf der Klasse möglich ist, wobei als `T` eine Klasse angegeben werden muss. Wie ein solcher Aufruf aussieht, ist in dem Quellcodebeispiel 27 zu sehen. In Zeile 4 des Quellcodebeispiels 28 beginnt die `Save`-Methode, die aus dem `ProfileManager` heraus aufgerufen wird, um eine Profildatei zu speichern. Die Methode bekommt ein Objekt vom Typen `T` übergeben und einen Pfad, an dem das Objekt gespeichert werden soll. In Zeile 10 wird ein `XmlSerializer` vom Typen `T` erzeugt, danach wird in Zeile 11 die Zieldatei erstellt und anschließend wird die Serialisierung in Zeile 12 durchgeführt. Für die Serialisierung wird der `XmlSerializer` aus dem Namensraum

---

```
System.Xml.Serialization;
```

---

verwendet, der durch das .NET Framework zur Verfügung gestellt wird. Dieser `XmlSerializer` führt das eigentliche Speichern der Daten durch.

Auch ein Laden der Profildatei ist durch den `ObjectSerializer` möglich. Dazu muss die Methode `Load` aufgerufen und der Pfad zu der zu deserialisierenden Datei übergeben werden. Der Aufruf der Methode aus dem `ProfileManger` geschieht wie folgt:

---

```
this._profile = ObjectSerializer<Profile>.Load(path);
```

---

Für die Variable `path` wird der Pfad zu der Profildatei an die Methode übergeben. Die `Load`-Methode liefert nach erfolgreicher Deserialisierung ein Objekt vom Typen `Profile` zurück, das in der Membervariablen `_profile` gespeichert wird. In dem Quellco-

debeispiel 28 beginnt die Load-Methode in Zeile 31. Auch hier kommt wieder der `XmlSerializer` zum Einsatz. Allerdings wird in Zeile 44 die übergebene Datei geöffnet und in Zeile 45 deserialisiert. In Zeile 65 wird schließlich das geladene Objekt zurückgeliefert.

#### 4.4 Bedienung des Programms

Das Programm kann durch Öffnen der Datei „Skill-Editor.cmd“ oder durch die EXE-Datei „\bin\Skill-Editor.exe“ gestartet werden.

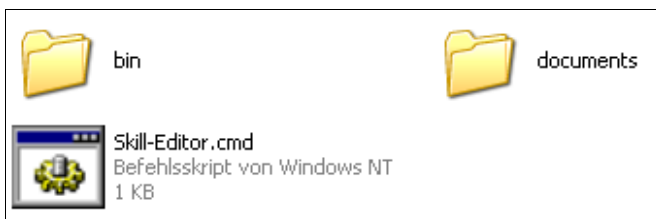


Abbildung 20: Starten durch Skill-Editor.cmd

Bei der Datei „Skill-Editor.cmd“ handelt es sich um eine Batch-Datei, die das Starten der Anwendung vereinfacht. Im Folgenden ist der Quellcode der Datei zu sehen:

```
(1) @echo off
(2)
(3) echo Skill-Editor laden
(4)
(5) if exist %bin\Skill-Editor.exe goto LOAD
(6) echo Datei nicht gefunden!
(7) goto EOF
(8)
(9) :LOAD
(10) start bin\Skill-Editor.exe
(11)
(12) :EOF
```

Quelltext 29: Code innerhalb der Batch-Datei Skill-Editor.cmd

In Zeile 5 wird untersucht, ob die Datei „\bin\Skill-Editor.exe“ existiert. Ist dies der Fall, springt das Skript zu Zeile 9 und lädt in Zeile 10 die EXE-Datei der Anwendung. Sollte die Datei nicht gefunden werden, wird eine entsprechende Meldung auf der Konsole ausgegeben und die Batch-Datei beendet.

Beim ersten Programmstart erscheint nach dem Starten der EXE-Datei ein Fenster, in dem die gewünschte Sprache ausgewählt werden muss. Die ausgewählte Sprache wird in einer sogenannten Config-Datei gespeichert und bei jedem Programmstart

eingelassen. Nach Auswahl der Sprache erscheint ein weiteres Fenster, das sogenannte „Start-Screen“ der Anwendung. In diesem Fenster kann der Benutzer auswählen, ob ein neuer Lebenslauf angelegt oder ein bereits erstellter Lebenslauf geladen werden soll.

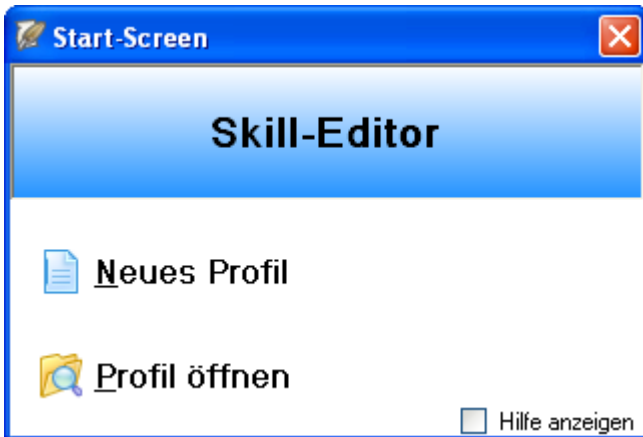


Abbildung 21: Start-Screen

Durch Aktivierung eines Häkchens kann für dieses Fenster ein kleiner Hilfetext angezeigt werden. Dieser Text erscheint direkt über der entsprechenden `CheckBox`. Das Fenster kann durch Klick auf das X auch geschlossen werden, wobei dann eine leere Profildatei angelegt wird. Wählt der Benutzer „Profil öffnen“ aus, so erscheint ein Dialog, in dem eine bereits angelegte Profildatei ausgewählt werden kann. Durch Klick auf „Neues Profil“ erscheinen in dem Fenster zwei Textboxen zum Eintragen von Vorname und Name, wie in der nächsten Abbildung zu sehen ist:

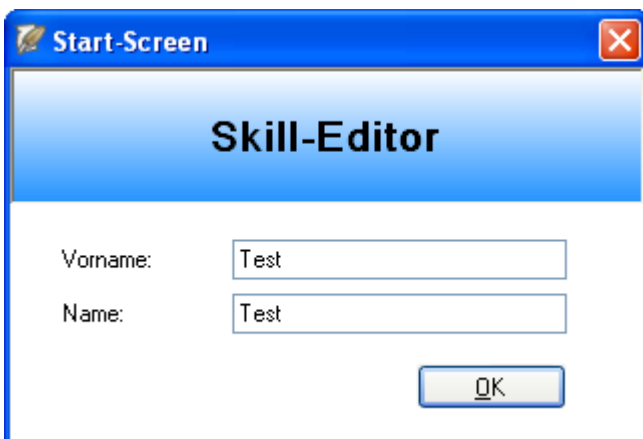
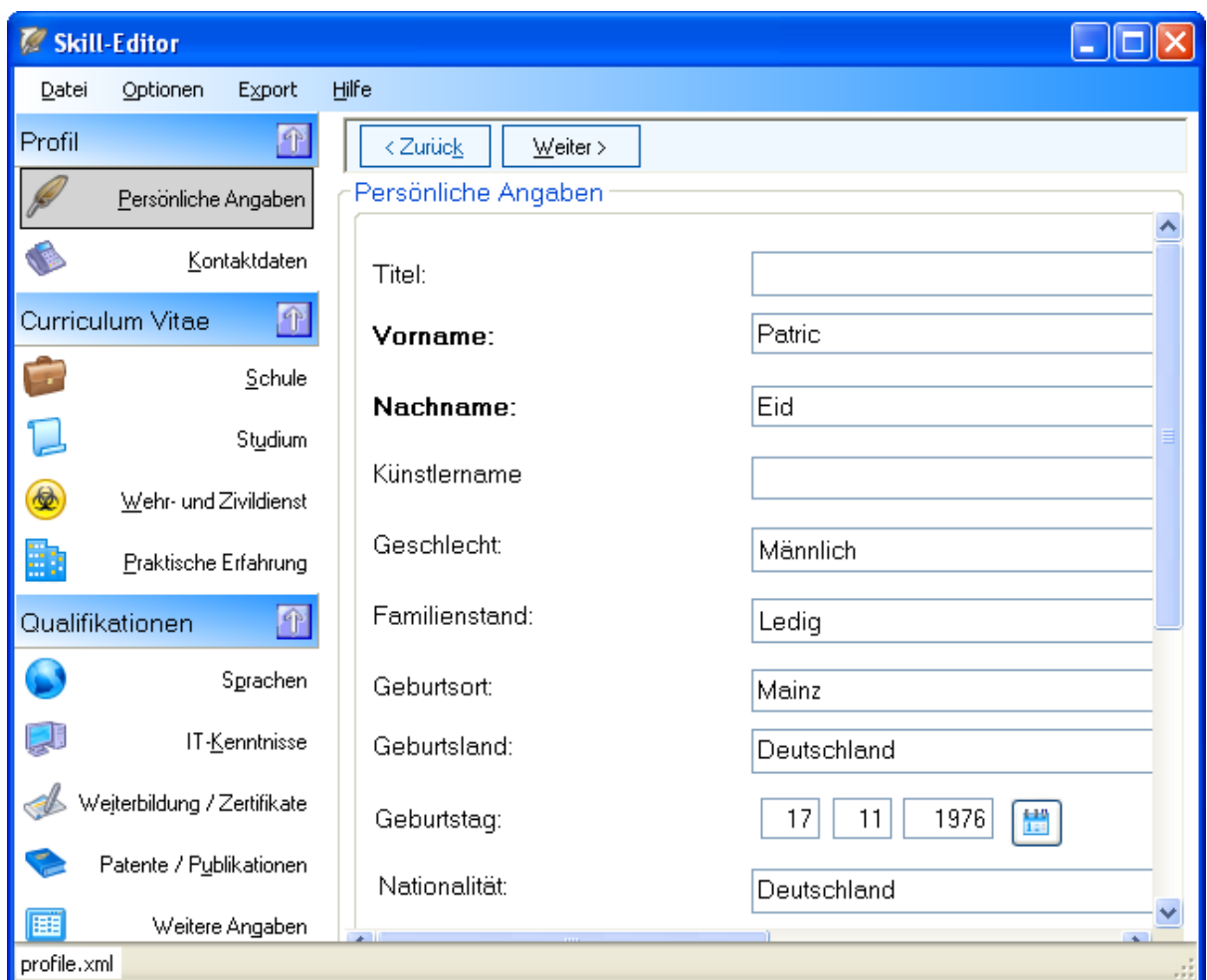


Abbildung 22: Neues Profil anlegen

Nachdem jeweils in beiden Feldern etwas geschrieben wurde, ist der „OK“-Button aktiviert und das Profil kann durch Klicken auf diesen Button oder betätigen der „Enter“-Taste, angelegt werden. Anschließend wird die Anwendung geladen und der Benutzer gelangt auf die erste Profilsseite.

Wie bereits aus dem Design-Kapitel bekannt, ist das Hauptfenster in drei wesentliche Bereiche unterteilt: Menü, Navigation und Inhalt. Das Menü besteht aus einer Menüleiste, die wichtige Funktionen der Anwendung enthält. Über die Navigation kann der Benutzer zwischen den verschiedenen Formularen, die zum Eintragen der Lebenslaufdaten benötigt werden, wählen. Im Inhalt, dem sogenannten Content-Bereich, werden die Eingabelemente des jeweils ausgewählten Formulars angezeigt. Der Content-Bereich beansprucht den größten Teil der Benutzeroberfläche. Die folgende Abbildung stellt das Formular zum Eintragen der persönlichen Angaben dar:



The screenshot shows the 'Skill-Editor' application window. The title bar reads 'Skill-Editor' and includes standard window controls. The menu bar contains 'Datei', 'Optionen', 'Export', and 'Hilfe'. The left sidebar is divided into sections: 'Profil' (with an up arrow), 'Curriculum Vitae' (with an up arrow), and 'Qualifikationen' (with an up arrow). Under 'Profil', there are icons for 'Persönliche Angaben' (selected), 'Kontaktdaten', 'Schule', 'Studium', 'Wehr- und Zivildienst', and 'Praktische Erfahrung'. Under 'Qualifikationen', there are icons for 'Sprachen', 'IT-Kenntnisse', 'Weiterbildung / Zertifikate', 'Patente / Publikationen', and 'Weitere Angaben'. The main content area shows the 'Persönliche Angaben' form with the following fields and values:

Field	Value
Titel:	
<b>Vorname:</b>	Patric
<b>Nachname:</b>	Eid
Künstlername	
Geschlecht:	Männlich
Familienstand:	Ledig
Geburtsort:	Mainz
Geburtsland:	Deutschland
Geburtsdag:	17 11 1976
Nationalität:	Deutschland

Navigation buttons '< Zurück' and 'Weiter >' are located above the form. A status bar at the bottom left shows 'profile.xml'.

Abbildung 23: Formular zum Eintragen der persönlichen Angaben

Der Benutzer hat die Möglichkeit mit Hilfe der „Weiter“- und „Zurück“-Buttons durch die einzelnen Profelseiten geführt zu werden. Es ist aber auch eine direkte Auswahl einer Profelseite durch Klicken auf den entsprechenden Eintrag in der Navigation möglich. Die aktuelle Position wird in der Navigation durch einen umrandeten und grau hinterlegten Eintrag dargestellt.

Die Abbildung 23 zeigt die Ansicht, wenn eine bereits vorhandene Profildatei geladen wurde. In diesem Fall ist keine Statusanzeige zu sehen, da nur schwer entschieden werden kann, wie weit der Bearbeitungsvorgang fortgeschritten ist. Anders sieht es aus, wenn ein neues Profil angelegt wird. Dann erscheint eine Statusanzeige, die den ungefähren Stand der Bearbeitung mitteilt. In der nächsten Abbildung ist eine neue Profildatei angelegt worden und die Statusanzeige befindet sich in dem oberen rechten Bereich der Benutzeroberfläche:

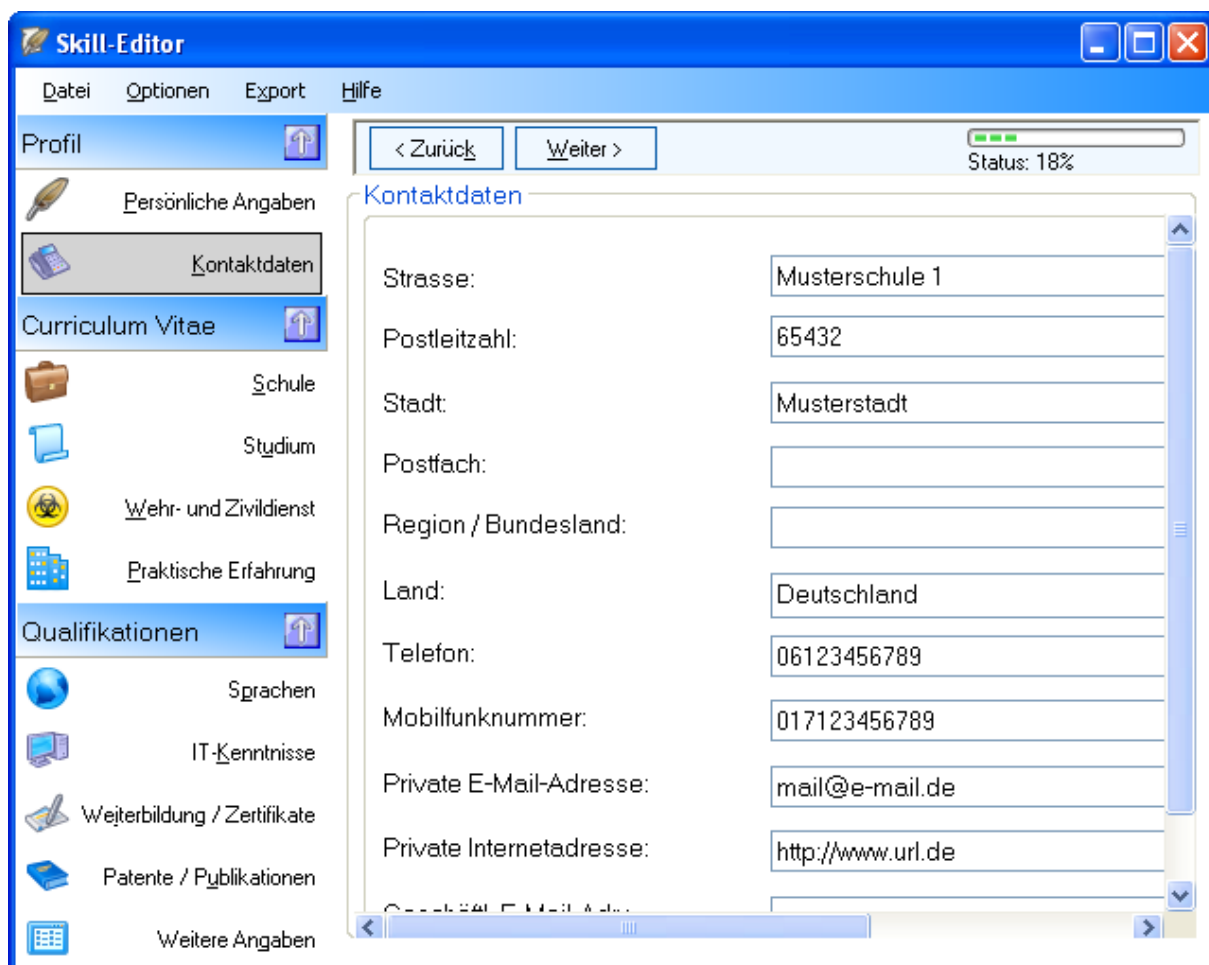


Abbildung 24: Ein neues Profil mit Anzeige des Eingabestatus

Der Bearbeitungsstatus wird anhand der aufgerufenen Formulare ermittelt. Der Bearbeitungsstatus von 100 % ist erreicht, sobald der Benutzer jedes der 11 verfügbaren Formulare mindestens einmal geöffnet hat. Die Statusanzeige zählt dabei die Seiten als bearbeitet, auch wenn keine Daten eingegeben worden sind. Dies geschieht, da nicht angenommen werden kann, welche der Formulare ein Benutzer ausfüllen möchte und wie viel Zeit für welches Formular dabei benötigt wird. Von daher gibt die Statusanzeige nur einen ungefähren Wert wieder und soll dem Benutzer lediglich vermitteln, dass die Eingabe in absehbarer Zeit ein Ende finden wird. In einer späteren Weiterentwicklung der Anwendung könnte es dem Benutzer freigestellt werden, welche Formulare bearbeitet und welche ausgelassen werden sollen. Die Statusanzeige könnte sich dann an den tatsächlich bearbeiteten Formularen richten und würde den aktuellen Verarbeitungsstand genauer anzeigen.

Der Menüeintrag „Datei“ stellt verschiedene Funktionen zum Arbeiten mit den Profildateien bereit, wie zum Beispiel die aus anderen Programmen ebenfalls bekannten Funktionen „Speichern“, „Öffnen“, „Speichern unter“ oder „Neu“. Der Menüeintrag „Hilfe“ enthält eine Info-Box, die Informationen zu der Anwendung enthält. Der Menüeintrag „Optionen“ beinhaltet verschiedene Einstellungen, die ein Benutzer vornehmen kann, wie zum Beispiel die Auswahl der Sprache oder das Anzeigen von Hilfetexten zu dem aktuell geöffnetem Formular. Wie eine solche Hilfeseite, in dem Fall für die Schulausbildung, aussieht, demonstriert die folgende Abbildung:

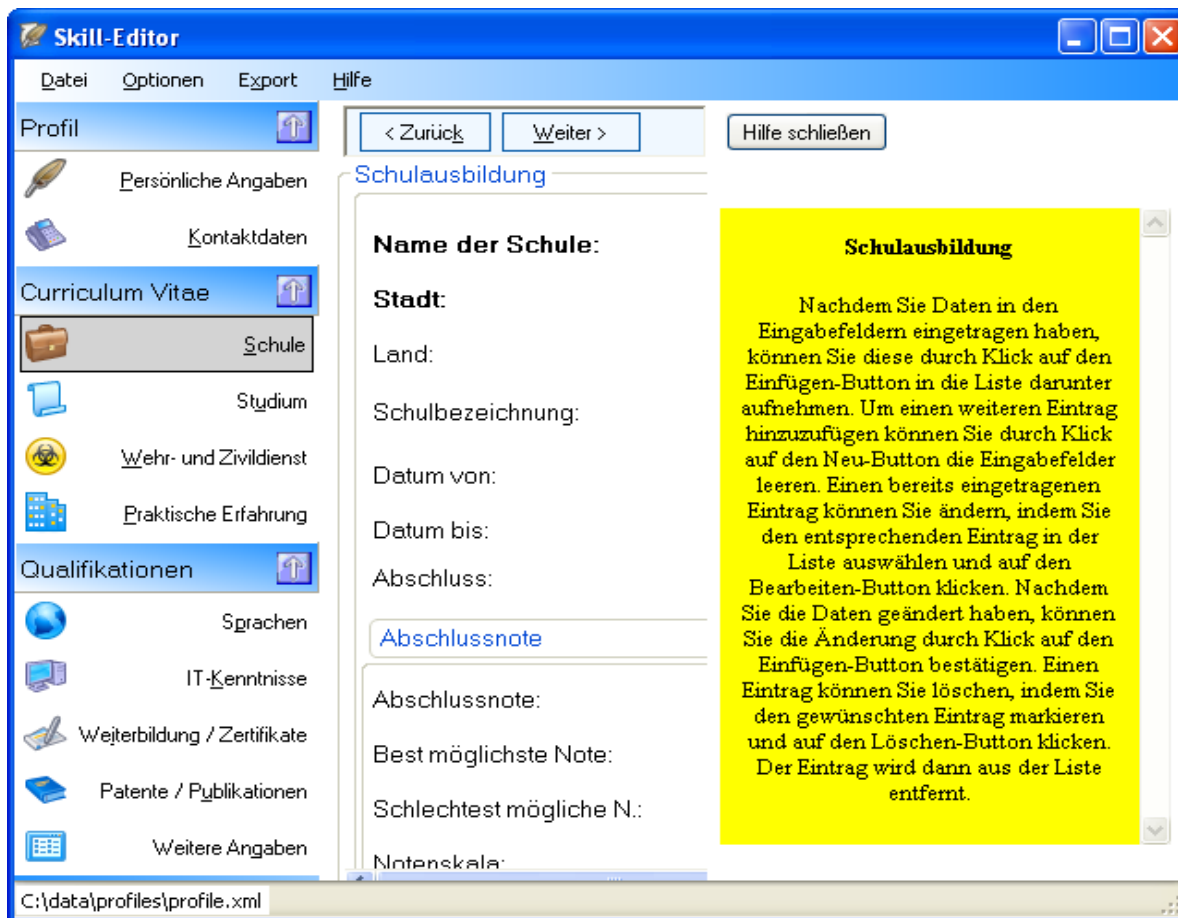


Abbildung 25: Anzeige einer Hilfeseite für die Schulbildung

Zu jedem Formular kann es einen entsprechenden Hilfetext geben, der in dem Hilfe-fenster angezeigt werden kann. Im aktuellen Stadium der Implementierung sind allerdings nur beispielhafte Hilfetexte hinterlegt. Der Menüeintrag „Export“ zeigt alle verfügbaren Export-Funktionen an. Diese können benutzt werden, sobald ein Profil angelegt und gespeichert wurde. Wird eine Export-Funktion vor dem erstmaligen Speichern der Profildatei benutzt, wird der Benutzer aufgefordert, die Profildatei zunächst zu speichern. Ansonsten erfolgt vor jedem Export eine automatische Speicherung der Profildatei. Im Rahmen der Diplomarbeit wurden die Exportfunktionen GSCV (HR-XML), HTML und MS-Word implementiert, was auch in der folgenden Abbildung ersichtlich ist:

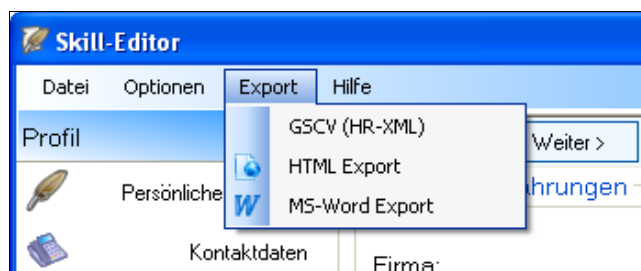


Abbildung 26: Export-Funktionen

Die Export-Funktionen „GSCV (HR-XML)“ und „HTML Export“ benutzen jeweils ein XML-Stylesheet für die Transformation der Profildaten. Das Plugin „MS-Word Export“ benutzt die interne Speicherung der Profildatei und generiert mit Hilfe eines Templates ein MS-Word-Dokument. Im weiteren Verlauf werden die einzelnen Export-Funktionen nochmals genauer dargestellt.

#### 4.4.1 GSCV (HR-XML)-Export

Beim Exportieren in ein GSCV-Format wird anhand der Profildatei eine XML-Datei erstellt, wozu ein XML-Stylesheet verwendet wird. Dieser XML-Stylesheet holt aus der gespeicherten Profildatei die relevanten Daten und exportiert diese in eine neue XML-Datei. Die Datei wird aus dem Namen der Profildatei und der Endung „\_gscv.xml“ zusammengesetzt. Damit die exportierten Daten in einem Web-Browser angesehen werden können, bietet GSCV ein weiteres XML-Stylesheet an, mit dem bestimmte Daten wie Ländercodes oder Sprachkenntnisse formatiert als HTML dargestellt werden können. Dieser XML-Stylesheet, der bei den GSCV-Dateien verwendet wird, benutzt allerdings eine veraltete HR-XML Version. Von daher wird beim Export nicht die aktuellste HR-XML Version<sup>1</sup>, sondern die von GSCV unterstützte Version<sup>2</sup> angegeben. Dieses XML-Stylesheet wird ebenfalls in das Verzeichnis kopiert, in dem die Export-Datei erstellt wird. In der folgenden Abbildung ist die Auswirkung des XML-Stylesheets ersichtlich, wenn der Lebenslauf in einem Web-Browser geöffnet wird:

<sup>1</sup> Aktuelle HR-XML Version: 15.04.2007 (Stand: 22.01.2008)

<sup>2</sup> Von GSCV unterstützte HR-XML Version: 28.02.2006 (Stand: 22.01.2008)



Abbildung 27: Anzeigen der exportierten Datei im Web-Browser

Die Abbildung zeigt die Darstellung eines exportierten Profils im GSCV-Format im Internet Explorer. Auf der rechten Seite werden die Kontaktdaten des Bewerbers aufgelistet, auf der linken Seite die Profildaten. Es fängt dabei mit persönlichen Angaben an, gefolgt von beruflichen Erfahrungen, Ausbildung und Sprach-Kenntnissen. Die Abbildung zeigt allerdings nur die Daten, die das GSCV-Stylesheet als HTML-Datei transformiert. Weitere Daten wie IT-Kenntnisse oder Publikationen sind in der exportierten XML-Datei gespeichert und sind ersichtlich, wenn die XML-Datei in einem Editor geöffnet wird.

## 4.4.2 HTML-Export

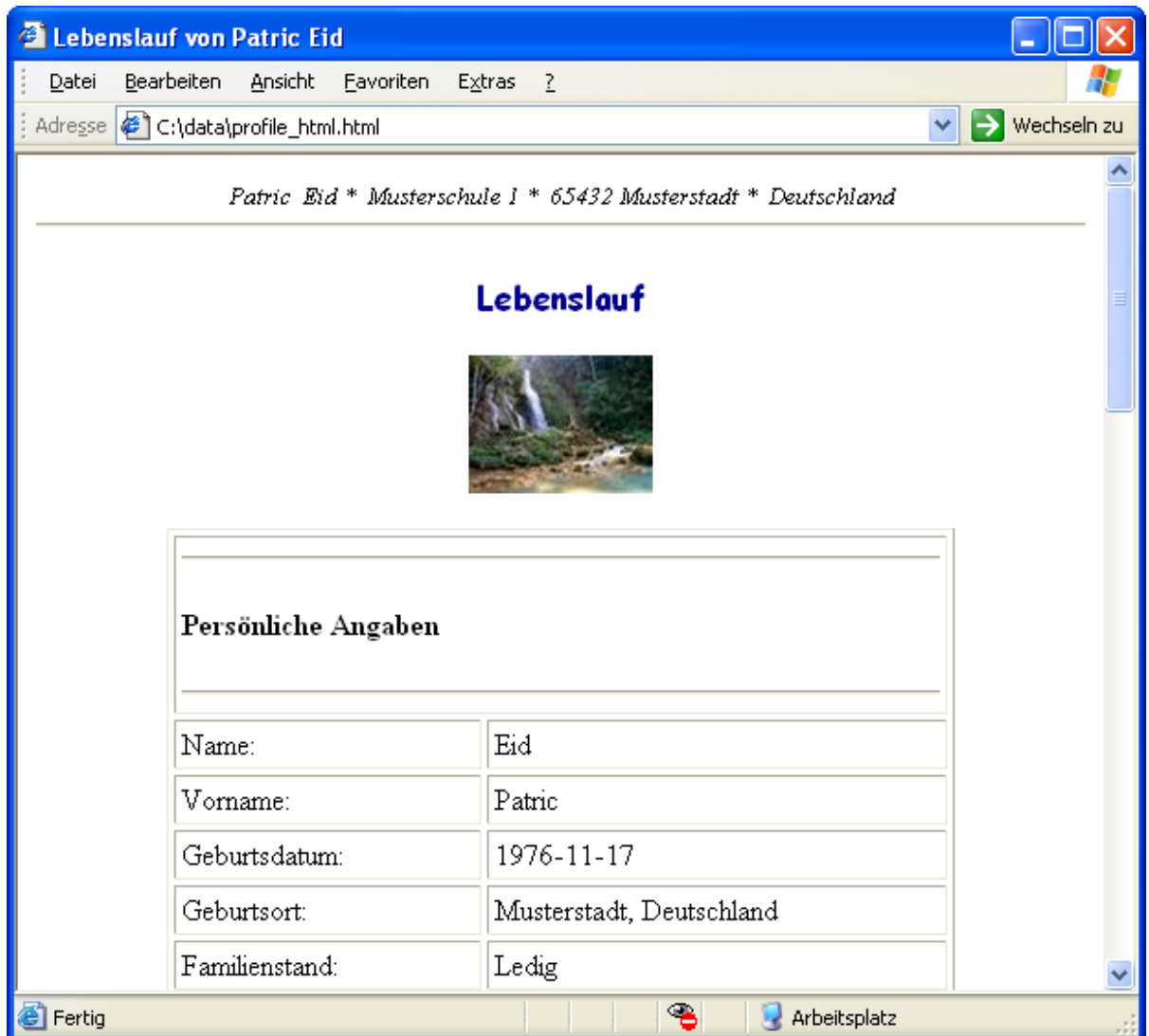


Abbildung 28: Beispiel eines HTML-Exports

Der Export als HTML-Datei wird ebenfalls mittels eines XML-Stylesheets durchgeführt. Dabei werden die eingegebenen Profildaten in Form von Tabellen in eine HTML-Datei geschrieben. Nachdem das Exportieren der Daten beendet ist, wird die lokal erstellte HTML-Datei in dem Standard Web-Browser geöffnet. Sofern der Benutzer ein Bild in seinem Profil hinzugefügt und markiert hat, dass dieses angezeigt werden soll, wird das Bild als eine JPG-Datei in dem Ordner gespeichert, in dem sich auch der exportierte Lebenslauf befindet. Die Bilddatei erhält dabei immer den Namen „picture.jpg“.

### 4.4.3 MS-Word-Export

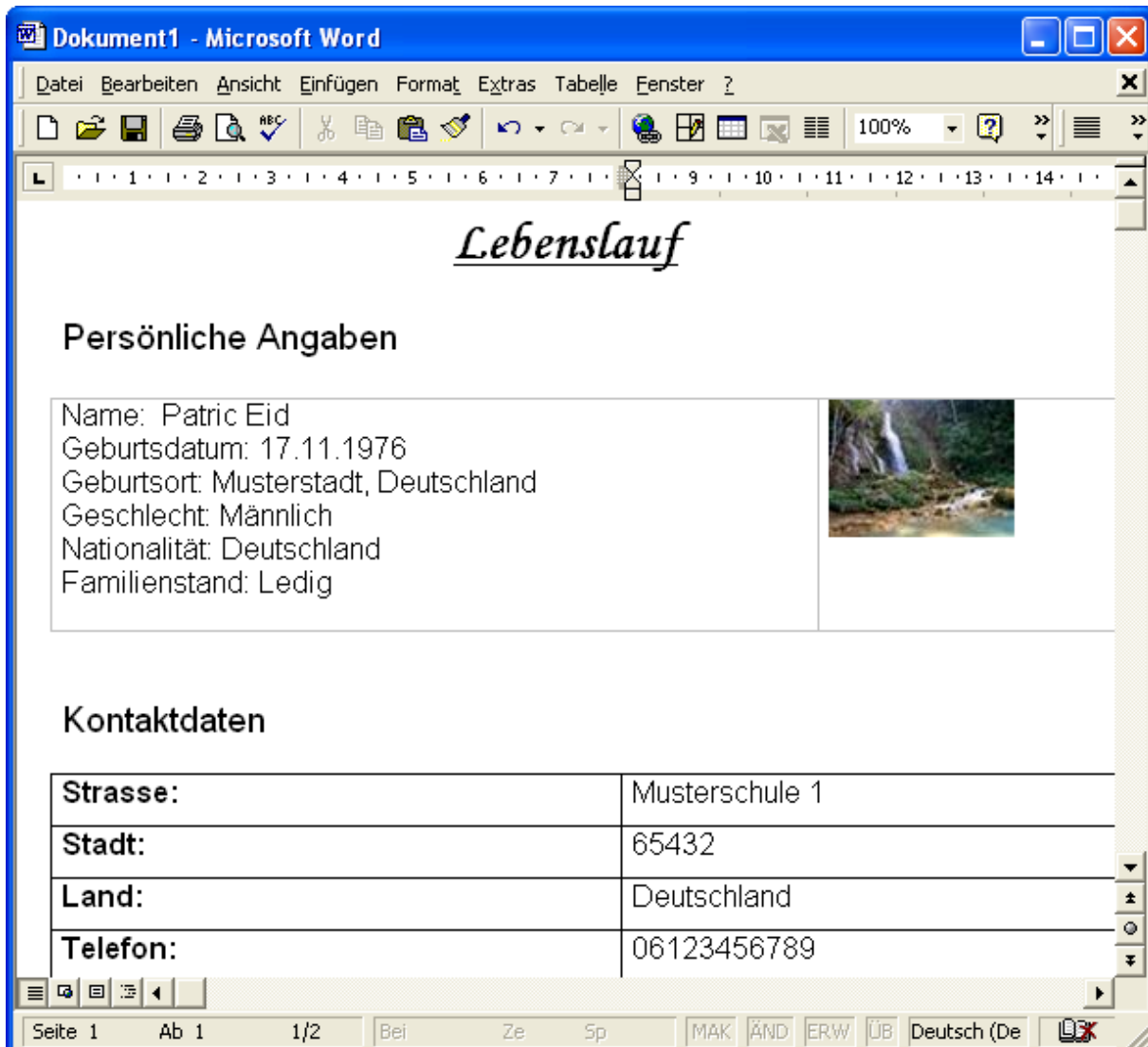


Abbildung 29: Beispiel eines MS-Word-Exports

Beim Exportieren der Daten als eine MS-Word-Datei werden die Profildaten ebenfalls in Tabellenform gespeichert. Dazu wird Microsoft Word geöffnet und eine neue Datei anhand eines Word-Templates erstellt. Das Profil kann dann mit Microsoft Word weiterverarbeitet werden. Die Überschriften „Persönliche Angaben“ oder „Kontaktdaten“ sind genauso wie die Position der Tabellen bereits als Word-Bookmarks in dem Word-Template definiert.

## 4.5 Zukünftige Erweiterungen

Im Rahmen der Diplomarbeit sollte lediglich ein Prototyp entwickelt werden, sodass es an einigen Stellen noch Verbesserungen oder Erweiterungen geben kann. Einige dieser möglichen Erweiterungen sind im weiteren Verlauf zusammengefasst.

### 4.5.1 Auswahl der Formulare

Ein Benutzer sollte auswählen können, welche Formulare ausgefüllt werden sollen. Es könnte Benutzer verunsichern, wenn sie zum Beispiel zu dem Punkt „Studium“ kommen, aber dort nichts eintragen wollen. Die Möglichkeit zur Auswahl der Formulare könnte entweder auf den jeweiligen Formularen in Form einer zusätzlichen `CheckBox` am Kopfe des Formulars oder anhand einer generellen Auswahlliste erfolgen. In der Auswahlliste wären alle zur Verfügung stehenden Formulare, die entweder aktiviert oder deaktiviert werden könnten. Die Statusanzeige sollte sich dann nach den ausgewählten Formularen richten.

### 4.5.2 Verbesserung der Export-Funktionen

Zur Zeit erfolgen nur wenige Kontrollen bei dem Export der Daten. Es werden verschiedene Dinge bereits überprüft, zum Beispiel ob Eingaben angezeigt werden oder verborgen bleiben sollen. Doch wenn zum Beispiel bei der Schulbildung keine Daten angegeben wurden und es erfolgt ein Export als MS-Word-Datei, so wird dennoch die Überschrift Schulausbildung in das Dokument geschrieben. Der Bereich darunter bleibt dann allerdings leer. Um dieses Problem zu umgehen müsste das verwendete Word-Template noch angepasst werden. Zur Zeit ist es so, dass dort auch schon die Überschriften aufgeführt stehen. Doch auch diese sollten durch das Export-Plugin in das Dokument geschrieben werden. Wenn keine Daten vorhanden sein sollten, könnte dann auch die Überschrift ausgelassen werden.

### 4.5.3 Import-Funktion für HR-XML-Daten

Da HR-XML ein weit verbreitetes Format ist, sollte die Anwendung einen Import von HR-XML konformen Daten ermöglichen. Dazu könnte wie auch bei dem Export als GSCV-Format ein XML-Stylesheet verwendet werden. Allerdings sollte dieser eine XML-Datei erstellen, die der internen Speicherung der Profildaten entspricht. Diese Datei sollte dann durch die Anwendung geladen werden.

#### 4.5.4 Hilfetexte für die Formulare erstellen

Die Hilfefunktion ist in dem Prototyp nur ansatzweise für alle Formulare implementiert. Jedes Formular sollte noch einen eigenen Hilfetext mit Angaben zum Ausfüllen bekommen. Die Infrastruktur dafür ist gegeben und auch eine mehrsprachige Übersetzung ist berücksichtigt.

#### 4.5.5 Ausgefüllten Lebenslauf als Beispiel anbieten

Es sollte ein Beispiel eines Lebenslaufes erstellt werden, das geladen werden kann. Dieses Profil sollte dafür komplett mit Daten und einem Bild gefüllt sein. Weiter sollte zu dem Lebenslauf eine Anleitung beiliegen, die Schritt für Schritt das Erstellen eines Lebenslaufes anhand des Beispiels erklärt.

#### 4.5.6 Löschen des Bildes aus dem Profil

Zur Zeit hat ein Benutzer noch nicht die Möglichkeit, ein einmal in ein Profil geladenes Bild wieder zu entfernen. Hierfür sollte es noch einen weiteren Button zum Löschen des geladenen Bildes geben. Bisher kann nur ein neues Bild geladen werden, um das vorherige Bild zu überschreiben.

#### 4.5.7 Export von hResume-Microformate

Mit Hilfe des Microformat Standards hResume [*HRESUME*] können Profildaten und Lebensläufe in HTML, XHTML oder RSS eingebunden werden. Unter anderem wird dieses Microformat bereits von der Social-Network-Plattform LinkedIn<sup>1</sup> eingesetzt, indem öffentliche Profile als hResume generiert werden. Für die Erstellung eines Profils im hResume-Format ist eine XML-ähnliche Syntax notwendig. Die Profildaten werden direkt in die HTML-Seite geschrieben, in der sie angezeigt werden sollen. Ein Export in ein solches Format wäre also durchaus mit dem vorhandenen Pluginkonzept denkbar.

---

<sup>1</sup> <http://www.linkedin.com/> (Stand: 22.01.2008)

### **4.5.8 Import von hResume-Microformate**

Für den Import der Daten in einem hResume-Format müsste die Anwendung noch um eine weitere Funktion ergänzt werden. Da auch ein Import von HR-XML angestrebt werden könnte, würde es sich anbieten, das Pluginkonzept um die Möglichkeit eines Imports zu erweitern. Durch einen Import von hResume wird ermöglicht, dass zum Beispiel die in LinkedIn hinterlegten Daten in das entwickelte Programm eingebunden und weiterverarbeitet werden können. Ein in LinkedIn eingetragener Lebenslauf muss somit nicht erneut erfasst werden.



## 5 EVALUIERUNG

Im Rahmen der Diplomarbeit hat eine Evaluierung über den Stand der entwickelten Anwendung stattgefunden. Ziel der Evaluierung war es mögliche Schwachstellen aufzuzeigen und Verbesserungsvorschläge einzuholen, damit die Anwendung im Anschluss an die Diplomarbeit noch weiter entwickelt und somit für eine weit gefasste Endbenutzergemeinde zugänglich gemacht werden kann. Die Auswahl der Testkandidaten setzte sich aus Verwandten, Freunden und Arbeitskollegen zusammen. Die PC-Kenntnisse waren dabei sehr unterschiedlich, da es sich sowohl um Leute handelte, die in der IT-Branche tätig sind, als auch um Personen, die wenig Zeit mit Computersystemen verbringen und diese auch nicht beruflich einsetzen. Die Teilnahme an der Evaluierung ist anonym mit Hilfe einer speziellen Internetseite<sup>1</sup> erfolgt. Auf dieser Seite war eine Umfrage mit sechs Unterkapiteln angelegt, wobei jedes Unterkapitel zwischen zwei und sechs Fragen enthielt. Zur Teilnahme an der Umfrage haben die Teilnehmer einen eindeutigen Zugangsschlüssel erhalten, mit dem an der Umfrage teilgenommen werden konnte. Nachdem ein Teilnehmer die Umfrage durchgeführt hatte, wurden dessen Daten aus dem System gelöscht. Die Antworten allerdings wurden beibehalten und nach Ablauf der Umfragefrist ausgewertet. Die Teilnahme an der Umfrage war für jeden eingeladenen Testkandidaten freiwillig. Im weiteren Verlauf wird der Umfang der Evaluierung beschrieben und die erzielten Ergebnisse zu den einzelnen Fragen angegeben.

### 5.1 Erstellen eines Testbogens

Die Anforderung des Testbogens war, dass er allgemeine Fragen über den Testkandidaten und Fragen zu dem entwickelten System erhalten sollte. Bei der Zusammenstellung der Fragen wurde das Buch „User Interface Design“ von Ben Shneiderman [SCH02] zu Rate gezogen, in dem ein Kapitel von Evaluierungen handelt. Für die Beantwortung der Fragen sollten einem Testkandidaten zwei unterschiedliche Varianten angeboten werden. Je nach Art der Frage sollte es möglich sein entweder einen Freitext oder eine Benotung anhand einer Notenskala anzugeben. Die Notens-

---

<sup>1</sup> <http://www.diplombewertung.tospiti2.de/survey.php> (Stand: 22.01.2008)

kala beinhaltete die Zahlen 0 bis 10, wobei 0 für eine neutrale Bewertung, 1 für die schlechteste und 10 für die best mögliche Bewertung steht. Im nächsten Kapitel wird die genaue Durchführung des Tests beschrieben, anschließend werden die Ergebnisse zu den Fragen präsentiert.

## **5.2 Testdurchführung**

Zur Durchführung des Tests wurde für die Testkandidaten ein Dokument erstellt, in dem Hilfen zur Installation der Software und den benötigten Komponenten, sowie eine Erklärung des Testbogens erhalten war. Dieses Dokument wurde zusammen mit den Zugangsdaten zur elektronischen Umfrageteilnahme an die jeweiligen Testkandidaten per E-Mail geschickt. Nach Ablauf der Teilnahmefrist wurden die Ergebnisse ausgewertet. In Anlage H ist der verwendete Testbogen zu finden. Auf die einzelnen Fragen wird später auch bei der Auswertung der Ergebnisse eingegangen.

## **5.3 Auswertung der Testergebnisse**

In diesem Kapitel werden die in der Umfrage erzielten Ergebnisse präsentiert und ausgewertet. Von den 30 eingeladenen Personen haben 12 an der Umfrage teilgenommen, was einer Umfragenbeteiligung von 40 % entspricht. Im weiteren Verlauf werden die Ergebnisse anhand der Umfragekapitel gegliedert dargestellt. Bei den Fragen mit Angabe einer Note wird die durchschnittlich erzielte Note angegeben. Die Antworten der Fragen mit Freitext werden zusammengefasst. Zu jedem Kapitel konnten auch Kommentare abgegeben werden, die ebenfalls in diesem Kapitel angegeben werden. Die Kommentare werden dabei in Hochkomma gestellt und mit einem Komma separiert, wenn mehrere Kommentare verfügbar sind.

### **5.3.1 Persönliche Angaben**

- Durchschnittliches Alter der Testkandidaten: 35,42 Jahre
- Geschlechtsverteilung:  
Männlich: 58,33 % Weiblich: 41,67 %
- Durchschnittliche Arbeitszeit mit Computersystemen in der Woche:  
41,33 Stunden oder 5,17 Arbeitstage
- Verteilung der verwendeten Betriebssystemen:

Windows XP: 91,67 % Windows Vista: 41,67 % Unix-Systeme: 25,00%

- Eigene Computererfahrung (1=keine Erfahrung ... 10=sehr erfahren): 7,33

Anhand des durchschnittlichen Alters und der durchschnittlichen Einschätzung der Erfahrung mit Computersystemen der Testkandidaten ist zu sehen, dass das Feld der Testkandidaten einen großen Bereich von Anwendern abdeckt. Die durchschnittliche Arbeitszeit mit Computersystemen ist jedoch sehr hoch. Dies lässt darauf schließen, dass es sich um Anwender handelt, die sehr viel mit Computersystemen zu tun haben. Allerdings muss bei diesem Wert auch beachtet werden, dass Testkandidaten dabei waren, die bei der Arbeitszeit 50 oder mehr Stunden in der Woche angegeben haben. In der folgenden Abbildung ist nochmals die genaue Verteilung der Computererfahrung zu sehen:

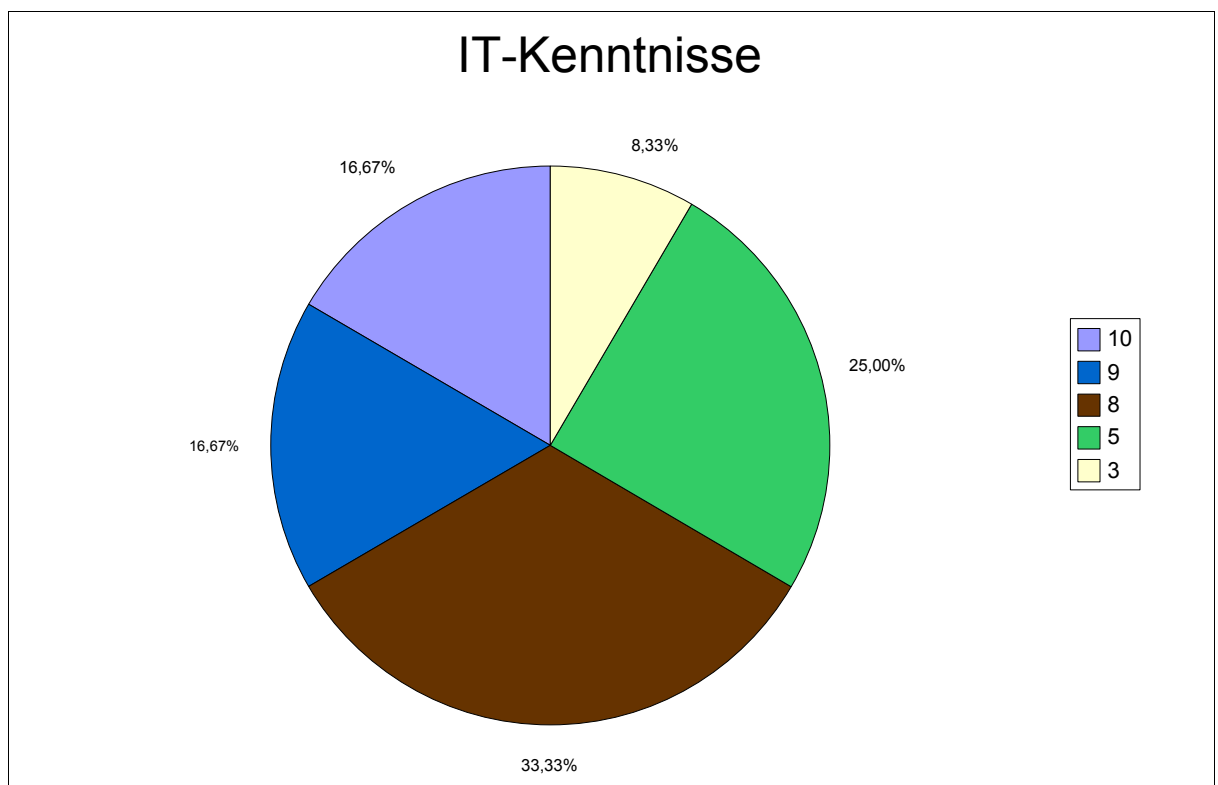


Abbildung 30: Einschätzung der IT-Kenntnisse

Anhand der Abbildung ist zu sehen, dass rund 33 % der Testkandidaten die eigene Erfahrung mit Computersystemen mit der Note 8 und rund 33 % die Computererfahrung mit der Note 9 oder 10 angegeben haben. Auf der rechten Seite der Abbildung sind in einer Legende die abgegebenen Einschätzungen aufgezeigt. Die prozentuale Verteilung ist in dem Bild jeweils neben dem farblich markierten Bereich zu finden.

### 5.3.2 Optische Wahrnehmung

- Zeichen auf dem Bildschirm (1=schwer zu lesen ... 10=leicht zu lesen): 8,67
- Hilfreiches Layout (1=nie ... 10=immer): 8,75
- Übersichtliche Anzahl von Informationen (1=zu viele Informationen ... 10=Menge der Informationen ist optimal): 8,92
- Abfolge der Screens (1=verwirrend ... 10=Nachvollziehbar): 9,5
- Kommentare: „Gut und übersichtlich“, „Zielführend“, „Navigationsmenü grafisch ansprechend, Eingabemasken sind eher funktional“

Die in diesem Kapitel erzielten Ergebnisse spiegeln einen ziemlich guten Eindruck des Systems wieder. Die optische Erscheinung der Anwendung scheint bei allen Testkandidaten gut angekommen zu sein.

### 5.3.3 Systemleistung

- Geschwindigkeit während der Bedienung (1=zu langsam ... 10=schnell genug): 9,5
- Antwortzeit bei den meisten Operationen (1=zu langsam ... 10=schnell genug): 9,33
- Anzeige von Informationen (1=zu langsam ... 10=schnell genug): 8,83
- System arbeitet verlässlich (1=nie ... 10=immer): 8,91
- Kommentare: „Gut“, „Angemessen“, „Für eine Diplomarbeit schon fast zu zuverlässig“

Auch die Antworten bei der Systemleistung zeigen, dass die Software den Anforderungen der Testkandidaten gerecht geworden ist und keine großen Mängel bei der Bedienung entstanden sind.

### 5.3.4 Dateneingabe

- Eingabe der Profildaten (1=umständlich ... 10=leicht): 8,5
- Bedienung der Export-Funktion (1=unklar ... 10=klar): 8,5
- Kommentare: „Einfach und übersichtlich“, „Englisch-sprachige Auswahlfelder nicht immer wirklich hilfreich“, „Berufsausbildung kann nicht eindeutig angegeben werden. Einfügefunktion nicht sofort erkennbar“, „Geht schnell“, „Ordnerauswahl beim Export“, „Inhalt Drop-Down Felder werden nicht voll angezeigt, bei Schulausbildung langt das Format MM/YYYY“

Die Bewertung bei der Dateneingabe ist gut ausgefallen, allerdings wurden viele Optimierungen vorgeschlagen, die an dem System noch durchgeführt werden könnten. Neben den hier erwähnten Zusammenfassungen gab es zu diesem Punkt auch weitere Zusammenfassungen von Testkandidaten, was noch verbessert werden könnte. Einige Punkte wurden im Rahmen der Diplomarbeit auch noch umgesetzt, andere können als Ausblick für eine weitere Entwicklung festgehalten werden.

### 5.3.5 Terminologie

- Terminologie innerhalb des Systems (1=inkonsistent ... 10=konsistent): 9,17
- Position von Botschaften auf dem Bildschirm (1=inkonsistent ... 10=konsistent): 9
- Auf dem Bildschirm erscheinende Botschaften sind (1=unklar und undeutlich ... 10=klar und deutlich): 8,33
- Vorhersagbare Ergebnisse werden (1=nie erzielt ... 10=immer erzielt): 9
- Fehlermeldungen sind (1=nicht hilfreich ... 10=hilfreich): 8,4
- Fehlermeldungen klären das Problem (1=nie ... 10=immer): 9,89
- Kommentare: „Keine Aussage zu Fehlermeldungen, da keine produziert“

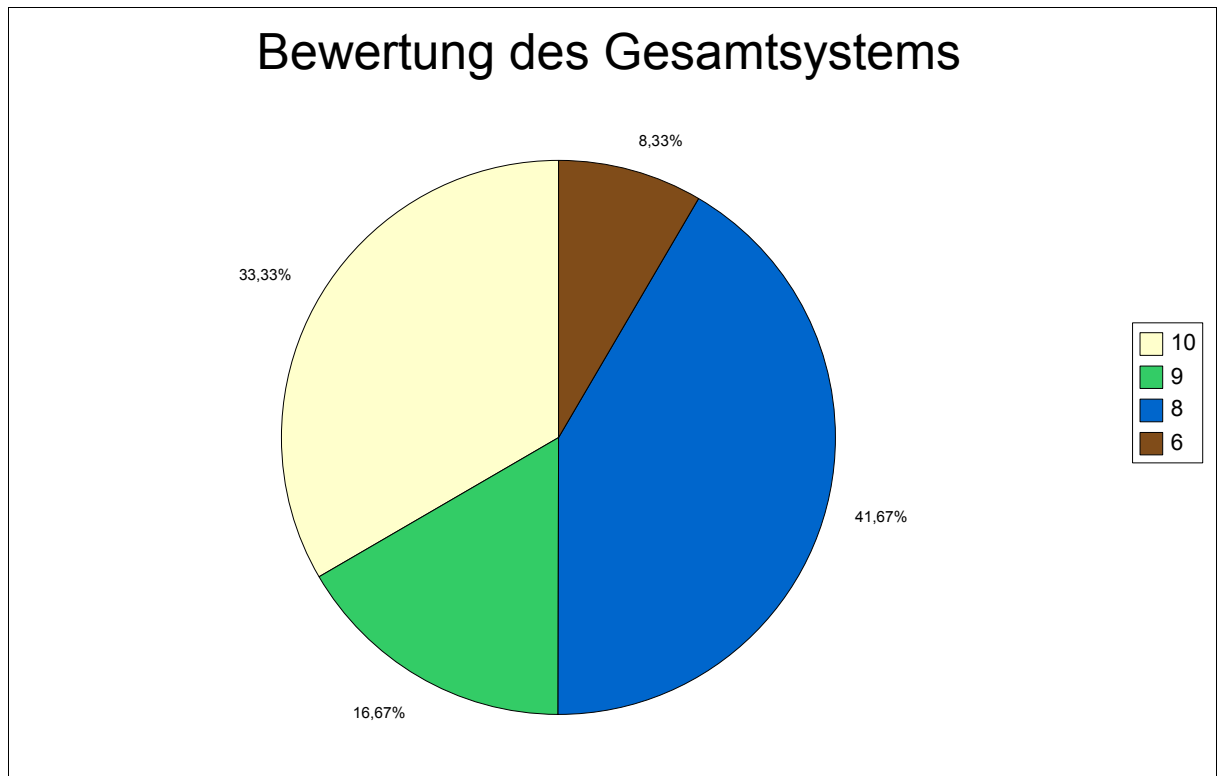
Die Terminologie des Systems hat sehr gute Ergebnisse erzielt. Fehler wurden nur in seltenen Fällen festgestellt. Allerdings sind bei vereinzelt Anwendern Probleme beim Export der Daten als MS-Word-Datei aufgetreten, die allerdings nicht reproduzierbar waren. Außerdem haben verschiedene Testkandidaten davon berichtet, dass die Anwendung während des Programmstarts versuchte auf das Internet zuzugrei-

fen. Das ist sehr ungewöhnlich, da eine solche Funktion in dem Skill-Editor nicht eingebaut ist. Doch da dies von mehreren Testkandidaten unabhängig von einander berichtet wurde, muss dieses Verhalten bei einer weiteren Entwicklung unbedingt genauer untersucht werden.

### 5.3.6 Fazit

- Erlangen eines Überblicks über die Grundfunktionen ist (1=schwierig ... 10=leicht): 9,25
- Fortgeschrittene Features können leicht erlernt werden (1=niemals ... 10=immer): 9,09
- Das Erforschen von Features durch Versuch und Irrtum ist (1=Entmutigend ... 10=Ermutigend): 9,09
- Der Gesamteindruck der Software ist (1=sehr schlecht ... 10=sehr gut): 8,67
- Kommentare: „Hilfreich und arbeitserleichternd“, „Gut“, „Bis auf die Schwierigkeiten des Export in Word (kann am Home-System liegen) keine Probleme. Allerdings finde ich es als Anwender einmal verwirrend, wenn ich solche Systemvoraussetzungen schaffen muss“, „Wenn man die Einfügefunktion verstanden hat, ist es möglich in kürzester Zeit einen Lebenslauf zu erstellen. Aufbau ist übersichtlich und verständlich.“, „Ist schon sehr, sehr gut. Aber für PC-Unerfahrene vermutlich nicht so leicht bedienbar. Jedoch würde das den Rahmen der DA sprengen“, „Funktional mit einem Ergebnis, das weiterverarbeitet werden kann“

Anhand der abgegebenen Bewertungen und den Kommentaren im Punkt Fazit ist davon auszugehen, dass die entwickelte Software bei den Testkandidaten einen sehr guten Eindruck hinterlassen hat. In der folgenden Grafik ist noch einmal die prozentuale Verteilung der Bewertung des Gesamtergebnisses zu sehen:



*Abbildung 31: Bewertung des Gesamtsystems*

Anhand der Abbildung ist zu sehen, dass die Hälfte der Testkandidaten die Software mit der Note 9 oder 10 bewertet haben. Rund 41 % gaben der Software eine 8, lediglich 8 % eine 6. Die abgegebenen Bewertungen sind in der Legende auf der rechten Seite der Abbildung zu sehen, die prozentuale Verteilung der jeweiligen Note ist in dem Diagramm ersichtlich.

## 5.4 Zusammenfassung

In den Kapiteln optische Wahrnehmung, Systemleistung, Dateneingabe, Terminologie und Fazit waren insgesamt 20 Fragen, bei denen eine Benotung angegeben werden konnte. Bei diesen Fragen konnte während der Evaluierung ein Notendurchschnitt von 8,96 erzielt werden, was einem sehr guten Schnitt für den jetzigen Stand des entwickelten Programms entspricht. Es ist zu beachten, dass es sich während der Testphase nicht um ein fertiges Produkt, sondern lediglich um einen Prototypen handelte. Das durchschnittlich erzielte Gesamtergebnis von 8,96 liegt relativ nahe an der durchschnittlich abgegebenen Bewertung des Gesamtsystems von 8,67. Dies zeigt, dass bei der Benotung des Gesamtsystems auch die anderen Punkte eine Rolle spielten, wohl jedoch in einer anderen Gewichtung als bei der Berechnung des Mittelwerts.

Mit den aus der Umfrage erzielten Ergebnissen und abgegebenen Verbesserungsvorschlägen, kann das im Rahmen der Diplomarbeit entwickelte Programm nach Beendigung der Diplomarbeit optimal auf die Anwendergruppe zugeschnitten und verbessert werden.

## 6 FAZIT

Damit das Fazit übersichtlicher gestaltet werden kann, wird es in diesem Kapitel in die Unterkapitel Technologie, Programmierung und Ergebnisse unterteilt. In dem Abschnitt zur Technologie wird auf die Verwendung des .NET Frameworks, der Programmiersprache C# und XML eingegangen. Der Punkt Technologie wird darauf von dem Abschnitt Programmierung erneut aufgegriffen, in dem die Vorteile von C# für dieses Projekt erläutert werden. Abschließend erfolgt ein Fazit über die erzielten Ergebnisse und was daraus geschlossen werden kann.

### 6.1 Technologie

Das .NET Framework einzusetzen ermöglicht ein „leichtes“ Erstellen von Formularen, da in der Klassenbibliothek bereits sehr viele `Controls` verfügbar sind. Mit Hilfe von Visual Studio können die entworfenen `UserControls` optimal an die Bedürfnisse angepasst werden. Unterstützt werden die .NET `Controls` im Rahmen der Diplomarbeit von Infragistics Komponenten, die eine Fülle von weiteren `Controls` bereitstellen. Dadurch kann die Eigenständigkeit des Erscheinungsbildes der Anwendung weiter erhöht werden. Ein weiterer Pluspunkt ist die Unterstützung der Globalisierung durch das .NET Framework, mit dem die Anwendung komfortabel auf mehrere Sprachen ausgerichtet werden kann. Die Vorgangsweise dazu ist, dass ein `UserControl` für eine Standard Sprache erstellt und dann die Eigenschaft `Localizable` auf den booleschen Wert `true` gesetzt wird. Unter der Eigenschaft `Language` wird dann die Sprache eingestellt, in die das `UserControl` übersetzt werden soll. Visual Studio legt dann für diese Sprache eine eigene Ressourcen-Datei an, in dem für alle in dem `UserControl` beinhalteten `Controls`, wie zum Beispiel die `Labels`, eine Übersetzung angegeben werden kann. Beim Aufrufen des `UserControls` in der Anwendung wird dann automatisch anhand der eingestellten Sprache die entsprechende Ressourcen-Datei geladen. Ist für eine Sprache keine Ressourcen-Datei verfügbar, wird die Standard Ressourcen-Datei geladen.

Da in den Formularen zum Eintragen der Lebenslaufdaten allerdings auch dynamische Werte aus XML-Dateien geladen werden, musste die Globalisierung dahingehend noch erweitert werden. Dazu gibt es für jede unterstützte Sprache eine eigene

XML-Datei mit allen Elementen, die innerhalb der in der Anwendung enthaltenen Auswahlmenüs gebraucht werden, wie zum Beispiel die Liste der Abschlüsse oder die verschiedenen Schulformen. Entsprechend der eingestellten Sprache wird eine XML-Datei Namens „controls.xml“ geladen, in der für das jeweilige Auswahlmenü die Werte erhalten sind. Für die Standard Sprache Deutsch sieht der Pfad, ausgehend von der Datei „Skill-Editor.exe“, zu dieser XML-Datei folgendermaßen aus:

---

```
\bin\data\lang\de\controls.xml
```

---

Die englische Version dieser Datei ist in dem Unterordner

---

```
\bin\data\lang\en\
```

---

enthalten. Jede weitere Sprache würde einen eigenen Unterordner bekommen. Der Dateiname wäre allerdings jedesmal gleich. Da also für jede Sprache eine eigene XML-Datei existiert, können somit auch die Abschlüsse, Schulformen etc. in verschiedenen Sprachen dargestellt werden.

Da weder das .NET Framework noch die Infragistics Komponenten ein `Label` mit einem Farbverlauf im Hintergrund anbieten, musste solch ein `Label` mit Farbverlauf selbst entwickelt werden. Zum Einsatz kommt dieses `Label` bei diversen Fenstern und in dem Navigationsmenü. Die Elemente des Navigationsmenüs mussten ebenfalls selbst kreiert werden, um die Menüs auf bzw. zu klappbar zu gestalten. Ansonsten waren keine weiteren eigenen Ergänzungen der `Controls` notwendig.

## 6.2 Programmierung

Es hat sich gezeigt, dass sich das Konzept während der Programmierung bewährt hat. Veränderungen mussten daran nicht mehr vorgenommen werden. Lediglich ein paar Ergänzungen waren notwendig, was allerdings abzusehen war, da während der Konzeptphase noch nicht an alle benötigten Klassen gedacht wurde. Unter anderem war während der Konzeption noch nicht klar, ob die interne Speicherung in einem Binärformat oder per XML geschieht. Aufgrund der Möglichkeit, die Profildaten mittels eines XML-Stylesheets in ein anderes XML- oder HTML-Dokument transformieren zu können, ist die Wahl des Speicherformats dann auf die XML-Speicherung gefallen. Die Klassen, die für die Speicherung und das Laden der Profildatei verantwortlich sind, gehören zu den Klassen, die während der Konzeption noch nicht berücksichtigt wurden.

Durch eine angestrebte Globalisierung sind während der Programmierung einige Probleme im Bezug auf die korrekte Übersetzung von Schulabschlüssen und Schulformen aufgetreten. Es hat sich herausgestellt, dass viele Abschlüsse und Schulformen aus dem deutschen Sprachgebrauch nicht in das Englische übersetzt werden können, da es diese Abschlüsse und Schulformen in dieser Form in Großbritannien (oder den Vereinigten Staaten) nicht gibt. Weiter hat es sich als kompliziert erwiesen, für diese Fälle äquivalente Bezeichnungen zu finden, die innerhalb des Programms und in der „controls.xml“ verwendet werden konnten. In der Implementierungsphase traten neben der schwierigen Übersetzbarkeit keine weiteren Probleme mehr auf.

Durch Nutzung des .NET Frameworks war die Umsetzung einer Globalisierung gut durchführbar. Ebenfalls hilfreich war das .NET Framework bei der Gestaltung der Formulare. Diese wurden komplett von der Datenschicht getrennt, damit ein leichtes Austauschen oder Ändern des Erscheinungsbildes möglich ist. Die Formulare sind zusammen mit der Datenschicht als eine eigenständige Assembly-Datei implementiert worden und werden als separate DLL-Datei zu dem Hauptprojekt hinzugefügt.

### **6.3 Ergebnisse**

Anhand der Evaluation aus Kapitel 5 ist zu erkennen, dass die für die Diplomarbeit gesteckten Ziele erfüllt werden konnten. Das entwickelte Programm ist benutzerfreundlich und verständlich, eine leichte und intuitive Bedienung wird dem Benutzer geboten. Die Profildaten können in leicht lesbare, sowie weiterzuverarbeitende Formate exportiert werden, was ein ganz wichtiges Ziel dieser Diplomarbeit war. Der Export in eine GSCV-Datei verläuft problemlos und kann von außen her angepasst werden, ohne dass die Anwendung neu kompiliert werden muss. Dies garantiert eine schnelle Anpassung des exportierten Formats, sollte es Änderungen in der HR-XML-Struktur geben. Besonders bei Bewerbungen in Unternehmen, die den GSCV- oder HR-XML-Standard verwenden, stellt dies eine ansehnliche Arbeitserleichterung dar. Auch für die Unternehmen bedeutet diese Form der digitalen Bewerbung eine Erleichterung, da die Profildaten in einem standardisierten Format vorliegen und so leicht verarbeitet werden können. Die Möglichkeiten für ein Unternehmen gehen hierbei von einer automatischen Vorsortierung der Bewerber, bis hin zu einer Katalogisierung der Bewerberdaten. Die mit dem entwickelten Programm erzielten

Ergebnisse decken sich mit den Erwartungen an das Programm. Anhand der Evaluation ist ersichtlich, dass die Mehrheit der Anwender leicht mit der Anwendung arbeiten kann und die Anwendung als eine Erleichterung bei der Erfassung eines Profils gesehen wird. Der Anwender wird bei der Bearbeitung durch die einzelnen Formulare geführt und die jeweilige Position wird angezeigt. Damit wird stets das Gefühl vermittelt, dass der Anwender die Kontrolle über den Ablauf der Eingabe hat und sich jederzeit einen Überblick über den restlichen Aufwand bis zur vollständigen Erfassung der Profildaten verschaffen kann. Bei der Umsetzung der Konzeption wurde darauf geachtet, dass der Lösungsansatz aus Kapitel 3.2 vollständig auf Hinblick auf die acht goldenen Grundregeln der Schnittstellendefinition [SCH02] implementiert wurde, um dem Anwender einen möglichst großen Komfort bei der Erstellung des Profils zu bieten. Auch die übrigen Punkte aus Kapitel 3.2 wurden berücksichtigt, was sich in den Ergebnissen der Evaluation aus Kapitel 5 widerspiegelt

Bei der Umsetzung der Konzeption und der Programmierung des zu entwickelten Programms hat es sich als sehr hilfreich erwiesen, dass das Konzept so ausführlich dargestellt wurde. Dadurch konnte viel Arbeit gespart und vielen Problemen während der Implementierung aus dem Weg gegangen werden. Besonders die Überlegungen im Vorfeld im Bezug auf das Pluginkonzept waren für die Implementierung einer leistungsfähigen Pluginmodularisierung sehr hilfreich. Das Pluginkonzept und die daraus erfolgte Implementierung stellt einen wichtigen Baustein der Anwendung dar. Es ist somit möglich, durch Erstellen von autogenen Plugins die Anwendung beim Programmstart zu erweitern, ohne dass das eigentliche Programm von den Änderungen erfahren muss. Damit ist eine hohe Flexibilität bei der Bereitstellung von Exportfunktionen gegeben. Die Anwendung kann auf neue Anforderung schnell und sicher zugeschnitten werden, da alle erdenklichen Standards von der Exportfunktion unterstützt werden können.

## **7 ANHANG**

Der Anhang dieser Diplomarbeit beinhaltet folgende Informationen:

- A – Glossar, in dem einige in dieser Ausarbeitung genannten Begriffe erklärt werden
- B – Abbildungsverzeichnis mit den in diesem Dokument eingefügten Abbildungen
- C – Tabellenverzeichnis
- D – Quellcodeverzeichnis die Seiten gelistet, in denen ein Quellcode abgebildet wird
- E – Literaturverzeichnis mit einer Übersicht über die für diese Diplomarbeit verwendete Literatur
- F – Verwendete Spezifikationen
- G – Verzeichnis der genannten Firmen und Organisationen
- H – der in der Evaluierung verwendete Umfragebogen

## A Glossar

Begriff	Beschreibung
Assembly	Jede Softwarekomponente im .NET Umfeld heißt Assembly. Eine .NET-Anwendung ist eine Assembly, also eine Funktionseinheit mit mindestens einem Modul und beliebigen weiteren Dateien
Batch-Datei	Eine Batch-Datei kann eine Reihe von Befehlen enthalten, die Schritt für Schritt abgearbeitet werden. In der Anwendung wird eine Batch-Datei zum Starten der Applikation verwendet
DLL	DLL steht für Dynamic Link Library und stellt eine dynamische Bibliothek dar. Die in der Anwendung verwendeten Assemblies befinden sich jeweils in einer eigenen DLL-Datei
C#	C# ist die systemeigene Sprache für die .NET Common Language Runtime. Sie wurde entworfen, um sich nahtlos in die .NET-Laufzeitumgebung anzupassen
CVS	CVS ist die Abkürzung für Concurrent Versions Systems und bezeichnet ein Softwaresystem zur Verwaltung von Quellcode und anderen Dateien
EXE	EXE steht für executable und heißt ausführbar. Es handelt sich dabei um eine Dateierweiterung für ausführbare Dateien
HTML	HTML, oder Hypertext Markup Language, ist eine textbasierte Auszeichnungssprache zur Darstellung von Inhalten wie Texten, Bildern und Hyperlinks in Dokumenten. Diese Sprache wird verwendet, um die Inhalte in einem Browser darstellen zu können
JPG	Es handelt sich bei JPG oder auch JPEG um eine Familie von Komprimierungsalgorithmen, die nach der Entwicklergruppe, der Joint Photographics Experts Group, benannt wurde. Mit Hilfe von JPG kann eine verlustbehaftete Komprimierung einer Bilddatei durchgeführt werden
MS-Word	MS-Word, oder auch Microsoft Word, ist ein Textverarbeitungsprogramm von der Firma Microsoft, das in dem Microsoft Office-Paket enthalten ist
.NET	Dies ist eine von Microsoft entwickelte Softwareplattform. Sie umfasst Laufzeitanwendungen, eine Sammlung von Bibliotheken und angeschlossenen Dienstprogrammen
Plugin	Export-Plugins können nach der Kompilation eingebunden werden. Sie ermöglichen der Applikation somit, Export-Funktionen dynamisch zu laden und somit von außen veränderbar zu sein
Solution-Datei	Eine Solution-Datei stellt eine Projektmappe dar, die mehrere Projekte enthalten kann. Innerhalb der Projektmappe werden die Beziehungen und Bedingungen der enthaltenen Projekte gespeichert
UserControl	UserControls, oder auch Benutzersteuerelemente genannt, sind selbst entwickelte Steuerelemente für Windows Forms
Visual Studio	Von Microsoft entworfene Entwicklungsumgebung. Mit diesem Programm kann in mehreren Hochsprachen entwickelt werden, u.a. mit C#

Begriff	Beschreibung
Windows Forms	Stellt die GUI-Schnittstelle einer .NET-Anwendung dar
Windows XP / Vista	Microsoft Windows XP und Microsoft Windows Vista sind Betriebssysteme der Firma Microsoft
Word-Bookmark	Mit Hilfe eines Bookmarks in einem Word-Dokument kann ein Lesezeichen mit einem eindeutigen Namen gesetzt werden. Dieser Bookmark kann an Daten gebunden werden
Word-Template	Mit Hilfe eines Templates kann für Word eine Vorlage erstellt werden. Bei dem entwickelten Programm wird solch eine Vorlage zum Erstellen eines Lebenslaufes als MS-Word-Datei verwendet. Innerhalb dieser Vorlage können Bookmarks angegeben oder auch Definitionen der Schriftart, Schriftgröße usw. hinterlegt werden
XML	Die Extensible Markup Language ist eine Auszeichnungssprache zur Darstellung von hierarchisch strukturierten Daten in Form von Textdateien
XSD	XSD ist die Abkürzung von XML-Schema und dient zur Definition von XML-Dokumentenstrukturen in Form einer XML-Datei
XSL	Die Extensible Stylesheet Language dient zur Erzeugung von Layouts für XML-Dokumente
ZIP	Mit dem ZIP-Dateiformat können Dateien komprimiert archiviert werden

**B Abbildungsverzeichnis**

Abbildung 1: Aufteilung der benötigten Zeit	4
Abbildung 2: .NET Framework Infrastruktur	16
Abbildung 3: Ausgabe nach XSL-Transformation	30
Abbildung 4: HR-BA-XML Struktur	44
Abbildung 5: Elemente von JobPositionSeekerType	45
Abbildung 6: Resume Builder schlecht Übersetzt	56
Abbildung 7: Resume Builder CV - Übersicht	57
Abbildung 8: Easy Resume Creator Hauptfenster	59
Abbildung 9: Easy Resume Creator hilft beim Anlegen der Daten	60
Abbildung 10: Design der Benutzeroberfläche nach Programmstart	64
Abbildung 11: Design mit geöffnetem Formular	66
Abbildung 12: Anwendungsfalldiagramm	75
Abbildung 13: Aktivitätsdiagramm Profil anlegen oder bearbeiten	77
Abbildung 14: Aktivitätsdiagramm Lebenslauf exportieren	78
Abbildung 15: Klassendiagramm Model-View-Controller	79
Abbildung 16: Klassendiagramm Model-Konzept	81
Abbildung 17: Klassendiagramm Pluginkonzept	83
Abbildung 18: Sequenzdiagramm Plugin-Aufruf	84
Abbildung 19: Anzeige eines Meanwhile-Forms	88
Abbildung 20: Starten durch Skill-Editor.cmd	99
Abbildung 21: Start-Screen	100
Abbildung 22: Neues Profil anlegen	100
Abbildung 23: Formular zum Eintragen der persönlichen Angaben	101
Abbildung 24: Ein neues Profil mit Anzeige des Eingabestatus	102
Abbildung 25: Anzeige einer Hilfeseite für die Schulausbildung	104
Abbildung 26: Export-Funktionen	105
Abbildung 27: Anzeigen der exportierten Datei im Web-Browser	106
Abbildung 28: Beispiel eines HTML-Exports	107
Abbildung 29: Beispiel eines MS-Word-Exports	108
Abbildung 30: Einschätzung der IT-Kenntnisse	115
Abbildung 31: Bewertung des Gesamtsystems	119

**C Tabellenverzeichnis**

Tabelle 1: Zeichenkodierung nach ISO-8859	6
Tabelle 2: Basistypen in C#	18
Tabelle 3: Entscheidungsmatrix zum Einsatz eines XML Standards	53
Tabelle 4: Resume Builder, Vor- und Nachteile	58
Tabelle 5: Die Ziele des Akteurs Benutzer	68
Tabelle 6: Hauptaufgaben des Akteurs Benutzer	69

**D Quellcodeverzeichnis**

Quelltext 1: XML-Beispiel	7
Quelltext 2: XML-Beispiel erweitert	7
Quelltext 3: XML-Beispiel mit Attributen	8
Quelltext 4: Namensräume in einem XML-Dokument verwenden	10
Quelltext 5: XML-Schema-Beispiel	11
Quelltext 6: Ein dem XML-Schema konformes XML-Beispiel	12
Quelltext 7: „Hello World“-Programm mit C#	17
Quelltext 8: "Hello World" innerhalb eines Namensraumes	20
Quelltext 9: "Hello World"-Beispiel mit einer Methode	20
Quelltext 10: Aufruf der Methode PrintMessage aus einer anderen Klasse heraus	21
Quelltext 11: Exception-Beispiel mit Division durch 0	23
Quelltext 12: Exception abgefangen durch try-catch	24
Quelltext 13: Namensräume und Aufbau der Main-Methode	26
Quelltext 14: Methode CreateXmlDoc zum Erstellen einer XML-Datei mit C#	27
Quelltext 15: Erstellte XML-Datei literatur.xml	27
Quelltext 16: XML-Schema example.xsd	28
Quelltext 17: Methode IsXmlValidate zum validieren einer XML-Datei	28
Quelltext 18: XSL-Datei stylesheet.xsl	29
Quelltext 19: Methode Transform zum Anwenden der XSL-Datei	30
Quelltext 20: Beispiellebenslauf als HR-XML-Datei	41
Quelltext 21: Quelltext von XML-Schema Candidate.xsd	42
Quelltext 22: Lebenslauf als HR-BA-XML-Datei	46
Quelltext 23: Lebenslauf als GSCV-XML-Datei	51
Quelltext 24: Die LoadPlugins-Methode des ApplicationStartes	92
Quelltext 25: ExportPluginClicked aus Klasse MenuAction	93
Quelltext 26: Methode FillCountryLists aus Klasse CountryHandler	95
Quelltext 27: Die Methode Serialize aus der Klasse ProfileManager	96
Quelltext 28: Die Klasse ObjectSerializer	98
Quelltext 29: Code innerhalb der Batch-Datei Skill-Editor.cmd	99

**E Literaturverzeichnis**

- [AGG06] Allgemeines Gleichbehandlungsgesetz (AGG), bund.de, 2006, [http://www.bund.de/nn\\_175498/DE/BuB/A-Z/A-wie-Ausbildung/Allgemeines-Gleichbehandlungsgesetz/Allgemeines-Gleichbehandlungsgesetz-knoten.html\\_\\_nnn=true](http://www.bund.de/nn_175498/DE/BuB/A-Z/A-wie-Ausbildung/Allgemeines-Gleichbehandlungsgesetz/Allgemeines-Gleichbehandlungsgesetz-knoten.html__nnn=true) (Stand: 22.01.2008)
- [BEW07] Der Lebenslauf ist der wichtigste Teil Ihrer Bewerbung, Bewerbungsservice Spezial, <http://www.bewerbungsservice-spezial.de/lebenslauf.htm> (Stand: 22.01.2008)
- [BHL06] Tim Bray, Dave Hollander, Andrew Layman, Richard Tobin. Namespaces in XML 1.0, W3C Recommendation, 2006, <http://www.w3.org/TR/REC-xml-names/> (Stand: 22.01.2008)
- [BOR05] Günter Born. Jetzt lerne ich XML, Markt+Technik, 2005, ISBN: 3-8272-6854-0
- [BOR07] Günter Born, Benjamin Born. Visual C# 2005 Programmierhandbuch, Software & Support Verlag GmbH, 2007, ISBN: 978-3-939084-40-2
- [BPS06] Tim Bray, Jean Paoli, C. M. Sperberg-McQueen, Eve Maler, François Yergeau. Extensible Markup Language (XML) 1.0 (Fourth Edition), W3C Recommendation, 2006, <http://www.w3.org/TR/REC-xml/> (Stand: 22.01.2008)
- [BUC01] Marcellus Buchheit. Die Sprache C# im Detail, 2001, <http://www.microsoft.com/germany/msdn/library/net/csharp/DieSpracheCSharpImDetail.aspx> (Stand: 22.01.2008)
- [BUN07] White Paper zum HR-BA-XML Standard, Bundesagentur für Arbeit, <http://www.arbeitsagentur.de/zentraler-Content/A04-Vermittlung/A045-Dritte/Publikation/White-paper.pdf> (Stand: 22.01.2008)
- [ERL02] Helmut Erlenkötter. C# universell programmieren von Anfang an, rororo, 2002, ISBN: 3-499-60074-9
- [GER00] Christian Gerstberger. XML-Technologie, TU München, Wintersemester 2000/2001, <http://wwwweickel.in.tum.de/lehre/Seminare/Hauptseminar/WS00/XML/xml.pdf> (Stand: 22.01.2008)
- [GER07] Implementierungsrichtlinien, German Standard CV, <http://www.german-standard-cv.de/GSCV-Impl.pdf> (Stand: 22.01.2008)
- [HUT07] Rudolf Huttary. Visual C# 2005, Software & Support Verlag GmbH, 2007, ISBN: 978-3-935082-16-7
- [KOC07] Daniel Koch. XSLT, entwickler.press, 2007, ISBN: 978-3-939084-32-7
- [KUE06] Andreas Kühnel. Visual C# 2005 Das umfassende Handbuch, Galileo Computing, 2006, ISBN: 978-3-89842-586-5

- [LEH05] Thomas Lehr. 42, Aufbau-Verlag, 2005, ISBN: 3-351-03042-8
- [MAT07] Michael Matzer, Hartwig Lohse. Dateiformate, Software & Support Verlag GmbH, 2007, ISBN: 978-3-939084-34-1
- [RIC02] Jeffrey Richter. Microsoft .NET Framework Programmierung, Microsoft Press, 2002, ISBN: 3-86063-650-2
- [SCH02] Ben Shneiderman. User Interface Design, mitp-Verlag Bonn, 2002, ISBN: 3-8266-0753-8
- [SKO01] Aaron Skonnard. Understanding XML Namespaces, MSDN Magazin, 2001, <http://msdn.microsoft.com/msdnmag/issues/01/07/xml/> (Stand: 22.01.2008)
- [STE07] Lebenslauf, StepStone, [http://www.stepstone.de/home\\_fs.cfm?contentpage=http%3A//www.stepstone.de/tips/content/stepstone/bewerbung/schriftliche\\_bewerbung.cfm%3Fpreferer%3D](http://www.stepstone.de/home_fs.cfm?contentpage=http%3A//www.stepstone.de/tips/content/stepstone/bewerbung/schriftliche_bewerbung.cfm%3Fpreferer%3D) (Stand: 22.01.2008)
- [TEE07] Ingolf Teetz. Making online applications more easy for the candidate, German Standard CV, 2007, [http://www.german-standard-cv.de/GSCV\\_03.05.2007.pdf](http://www.german-standard-cv.de/GSCV_03.05.2007.pdf) (Stand: 22.01.2008)
- [UNI07] Der Lebenslauf, Uni Münster, <http://www.uni-muenster.de/CareerService/praktika/bewerbung/lebenslauf.html> (Stand: 22.01.2008)

**F Spezifikationen**

- [ECMA] European Computer Manufactures Association,  
<http://www.ecma-international.org> (Stand: 22.01.2008)
- [HR-BA-XML] Ein von der Bundesagentur für Arbeit entwickelter Standard zur Speicherung von Bewerberdaten und Jobinformationen
- [HRESUME] Ein Mikroformat um zum Beispiel Lebensläufe innerhalb einer HTML-Seite zu integrieren. <http://www.hresume.org>  
(Stand: 22.01.2008)
- [ISO3166] Ein Standard der ISO, zur Kodierung von geographischen Einheiten,  
[http://www.iso.org/iso/country\\_codes/iso\\_3166\\_code\\_lists.htm](http://www.iso.org/iso/country_codes/iso_3166_code_lists.htm)  
(Stand: 22.01.2008)
- [ISO8601] Ein Standard der ISO, um Zeit- und Datumsangaben zu beschreiben,  
[http://isotc.iso.org/livelink/livelink/4021199/ISO\\_8601\\_2004\\_E.zip?func=doc.Fetch&nodeid=4021199](http://isotc.iso.org/livelink/livelink/4021199/ISO_8601_2004_E.zip?func=doc.Fetch&nodeid=4021199) (Stand: 22.01.2008)
- [NACE] Nace 1.1 Level 3, Nomenclature générale des activités économiques dans les Communautés Européennes (NACE), ist ein System zur Klassifizierung von Wirtschaftszweigen,  
[http://www.ihk-bonn.de/service/service\\_96.php](http://www.ihk-bonn.de/service/service_96.php) (Stand: 22.01.2008)
- [RFC3066] Tags for the Identification of Languages,  
<http://tools.ietf.org/html/rfc3066> (Stand: 22.01.2008)
- [RFC3986] Uniform Resource Identifier, <http://tools.ietf.org/html/rfc3986>  
(Stand: 22.01.2008)

**G Firmen und Organisationen**

- [BAA] Bundesagentur für Arbeit, Regensburger Straße 104,  
90478 Nürnberg, <http://www.arbeitsagentur.de> (Stand: 22.01.2008)
- [EURIDYCE] Das Informationsnetz zum Bildungswesen in Europa,  
EU-Bureau of the Federal Ministry of Education and Research,  
Königswinterer Strasse 522-524, 53227 Bonn,  
<http://www.eurydice.org> (Stand: 22.01.2008)
- [EWORKS] eWorks GmbH, Hebelstr. 11, 60318 Frankfurt am Main,  
<http://www.eworks.de> (Stand: 22.01.2008)
- [GSCV] German Standard CV ist ein Service der milch & zucker – THE  
MARKETING & SOFTWARE COMPANY AG, Küchlerstr. 1, 61231  
Bad Nauheim, <http://www.german-standard-cv.de>  
(Stand: 22.01.2008)
- [HR-XML] HR-XML Consortium, Inc., 7474 Creedmoor Rd. #221, Raleigh,  
NC 27613, USA, <http://www.hr-xml.org> (Stand: 22.01.2008)
- [INFRA] Infragistics, Inc., Windsor Corporate Park, 50 Millstone Road,  
Princeton, NJ 08520, USW, <http://www.infragistics.com>  
(Stand: 22.01.2008)
- [IPROFILE] iProfileCentral, <http://www.iprofilecentral.com> (Stand: 22.01.2008)
- [MSOFT] Microsoft Corporation, 1 Microsoft Way, Redmond, WA 98052, USA,  
<http://www.microsoft.com> (Stand: 22.01.2008)
- [SARMS] Sarm Software, <http://www.sarmsoft.com> (Stand: 22.01.2008)

## **H Umfragebogen zur Evaluierung**

### ***Persönliche Angaben***

1. Ihr Alter:
2. Ihr Geschlecht:
3. Wie lange arbeiten Sie durchschnittlich mit Computersystemen in der Woche?
4. Welche Betriebssysteme (z.B. Windows XP, Windows Vista) benutzen Sie?
5. Bitte schätzen Sie Ihre eigene Computererfahrung ein (1=keine Erfahrung ... 10=sehr erfahren):

### ***Optische Wahrnehmung***

1. Anzeige von Zeichen auf dem Bildschirm (1=schwer zu lesen ... 10=leicht zu lesen):
2. Das Layout ist hilfreich (1=nie ... 10=immer):
3. Auf dem Bildschirm befindet sich eine übersichtliche Anzahl von Informationen (1=es sind zu viele Informationen zu sehen... 10=die Menge der Informationen ist optimal):
4. Die Abfolge der einzelnen Screens (oder Bildschirme, „Seiten“) ist (1=verwirrend 10=Nachvollziehbar):
5. Kommentar zur optischen Wahrnehmung:

### ***Systemleistung***

1. Geschwindigkeit während der Bedienung (1=zu langsam ... 10=schnell genug):
2. Antwortzeit bei den meisten Operationen (1=zu langsam ... 10=schnell genug):
3. Anzeige von Informationen (1=zu langsam ... 10=schnell genug):
4. System arbeitet verlässlich (1=nie ... 10=immer):
5. Kommentar zur Systemleistung:

### ***Dateneingabe***

1. Eingabe der Profildaten (1=umständlich ... 10=leicht):
2. Bedienung der Export-Funktion (1=unklar ... 10=klar):
3. Kommentar zur Dateneingabe:

**Terminologie**

1. Terminologie innerhalb des Systems (1=inkonsistent ... 10=konsistent):
2. Position von Botschaften auf dem Bildschirm (1=inkonsistent ... 10=konsistent):
3. Auf dem Bildschirm erscheinende Botschaften sind (1=unklar und undeutlich ... 10=klar und deutlich):
4. Vorhersagbare Ergebnisse werden (1=nie erzielt ... 10=immer erzielt):
5. Fehlermeldungen sind (1=nicht hilfreich ... 10=hilfreich):
6. Fehlermeldungen klären das Problem (1=nie ... 10=immer):
7. Kommentar zur Terminologie:

**Fazit**

1. Erlangen eines Überblicks über die Grundfunktionen ist (1=schwierig ... 10=leicht):
2. Fortgeschrittene Features können leicht erlernt werden (1=niemals ... 10=immer):
3. Das Erforschen von Features durch Versuch und Irrtum ist (1=Entmutigend ... 10=Ermutigend):
4. Der Gesamteindruck der Software ist (1=sehr schlecht ... 10=sehr gut):
5. Kommentar zum Gesamtsystem: