Technische Universität Ilmenau

Fakultät für Informatik und Automatisierung

Institut für Theoretische und Technische Informatik

Fachgebiet Prozessinformatik

Studienarbeit

„Erstellung und Anbindung intuitiver Frontends an eine Anwendung zur Verwaltung administrativer Personendaten unter Beachtung von Aspekten der Internationalisierung.“

Bearbeiter:	**Kumanan Kanagasabapathy**
Matrikelnummer:	**26880**
Matrikel:	**M 97**
Studiengang:	**Informatik**
Wissenschaftlicher Betreuer:	**Dipl. Ing. Christian Heller**

INHALTSVERZEICHNISS

ABBILDUNGSVERZEICHNIS

1. Einleitung

Die Informationstechnologie hat durch den stetigen technischen Fortschritt und der zunehmenden Globalisierung der Märkte für die Unternehmen an Bedeutung gewonnen. Diese Entwicklung wird durch die Verbreitung des Internets mitgetragen. Durch den expandierenden Weltmarkt ist es für Unternehmen sehr wichtig, ihre Softwareprodukte auf die Gegebenheiten der Lokaleneigenschaften angepasst anzubieten, um nicht aus dem Markt verdrängt zu werden.

Lange bevor Softwareprodukte international eingesetzt werden sollten, entwickelte schon die Industrie Produkte für globale Märkte. Beispiele hierfür sind die Automobilindustrie und die Textilindustrie, die ihre Produkte zugeschnitten auf den lokalen Märkten anbieten. So kann eine deutsche Automobilindustrie in einen amerikanischen Markt nur eindringen, wenn sie ihr Fahrzeugtacho auf die Maßeinheit Meilen/Stunde umstellt. In England fahren die Fahrzeuge auf der linken Seite, dem zufolge sind die Lenkräder auf der rechten Seite einzubauen. Bei der Textilindustrie wären die Beschriftungen auf den Textilien zu nennen, die in die jeweilige Landessprache übersetzt werden müssen.

In dieser Arbeit werden die Probleme der Internationalisierung von Software behandelt. Um diese verschiedenen Probleme verstehen zu können ist es notwendig, zuerst auf die unterschiedlichen Aspekte der Sprachen, Schriften und Kulturen der vielfältigen Regionen einzugehen. Aus diesem Grund wird für dieses Thema ein eigenes Kapitel eingeräumt. Der zweite Teil befasst sich mit den technischen Umsetzungsmöglichkeiten im Speziellen unter der Programmiersprache Java. Parallel zu dieser Arbeit wurde ein Modul als Beispiel Programm ResAdmin unter ResMediceane erstellt. In diesem Modul lassen sich Patienten-Daten in drei verschiedenen Sprachen bedienen. Im dritten Teil dieser Arbeit wird auf das Modul eingegangen.

2. Die Probleme der Internationalisierung von Software

Die Softwareindustrie hat ihre ursprüngliche Entwicklung in den Vereinigten Staaten. Demzufolge waren Englischkenntnisse für die Nutzung der Software unumgänglich. Erst seit den 90ern Jahren bieten namenhafte Softwarefirmen ihre Produkte in fast allen europäischen Landessprachen an. In den asiatischen Ländern findet erst seit den letzten Jahren die Informationstechnologie mehr und mehr Einzug. In diesen neuen Regionen Fuß zu fassen bedarf nicht nur der reinen Übersetzung der Software, sondern viel mehr stehen die kulturellen und sprachlichen Aspekte im Gegensatz zu den europäischen Ländern im Vordergrund. Viele Aspekte sind auf den ersten Blick vielleicht nicht so offensichtlich und werden darum oft vernachlässigt. Aus den zahlreichen Problemen der Internationalisierung von Software werden in diesem Kapitel nur die in erster Linie zu beachtenden kulturellen Unterschiede und Probleme aufgezeichnet.
In diesem Zusammenhang werden in der Softwareindustrie häufig die Abkürzungen l10n und i18n verwendet, die auf den englischen Bezeichnungen „*Localization*“ und „*Internationalization*“ basieren. Sie bestehen aus den Anfangs- und Endbuchstaben der Begriffe sowie der Anzahl dazwischenlegender Buchstaben. Die verallgemeinerte Bedeutung dieser Begriffe wird nachfolgend aufgelistet.

Localization (l10n): *“Localization is the process of adapting software for a specific region or language by adding locale-specific components and translating text. The term localization is often abbreviated as l10n, because there are 10 letters between the "l" and the "n." Usually, the most time-consuming portion of the localization phase is the translation of text. Other types of data, such as sounds and images, may require localization if they are culturally sensitive. Localizers also verify that the formatting of dates, numbers, and currencies conforms to local requirements.”*
Internationalization (i18n): “*Internationalization is the process of designing an application so that it can be adapted to various languages and regions without engineering changes. Sometimes the term internationalization is abbreviated as i18n, because there are 18 letters between the first "i" and the last "n.".”*[1]

---

[1] Zit. [JavaTutorial]

2.1 Die Sprache

Die Sprache ist das wichtigste Kommunikationsmittel des Menschen, um Informationen auszutauschen. Sie ist gekennzeichnet durch die Verwendung willkürlicher gesprochener oder geschriebener Symbole mit festgelegter Bedeutung. Die Schätzungen der existierenden Sprachen auf der Welt liegen zwischen 3000 und 4000 Einzelsprachen. Die Einzelsprachen lassen sich in vielen Fällen nicht eindeutig zu einem Land oder einer Region zuordnen. Vielmehr werden neben regional abhängigen sprachlichen Dialekten auch völlig unterschiedliche Fremdsprachen gesprochen. Ein Beispiel hierfür ist Indien. Hier haben sich vielfältige Sprachen herausgebildet, deren genaue Anzahl nicht bekannt ist. Offizielle Schätzungen liegen zwischen 845 bis 1652 Sprachen und Dialekte.

Die Probleme bei der heutigen Software sind nicht nur die neuen Sprachen, sondern dessen Vielfalt an unterschiedlichen Dialekten und kulturellen Ausprägungen. Diesen wichtigen Aspekt lässt sich anhand der uns bekannten Sprache Englisch verdeutlichen. Englisch wird in den USA, England und in Australien gesprochen, jedoch gibt es in allen drei Ländern unterschiedliche Dialekte. Die Amerikaner bezeichnen den Abfalleimer als „*trashcan*“, die Engländer „*wastebasket*“ und die Australier nennen es „*rubbish*“. In der Software werden inzwischen amerikanisches Englisch und britisches Englisch unterschieden, australisches Englisch liegt bis jetzt noch außen vor.[2]

Für die Identifikation der Länder besteht ein Standard unter ISO 3166 als Buchstabenkürzel, die auch im Internet als Top-Level Domain vergeben werden. Die Sprachen sind unter ISO 639 als Kürzel definiert. Jedoch gibt es keinen Standard oder eine Definition zum Dialekt.[3]

---

[2] Siehe weitere Beispiele unter : [DeiCza2001] , S. 6

[3] Siehe Anhang ISO 639 u. ISO 3166

2.2 Schreibsysteme

Sprachwissenschaftlichen Ermittlungen zufolge ist das erste Schreibsystem vor ca. 1500 Jahren vor Chr. in Sumeria (heute ein Teil des Iraks) entstanden. Die Sumerianer versuchten durch die Abbildung von Gegenständen und Tieren ihre alltäglichen Dinge zu beschreiben. Man nennt diese Art von Schreibsystemen in der Fachsprache Piktorgraphen. Erst um 600 Jahren vor Chr. repräsentierten die Symbole als Klänge, wodurch sich Verbindungen zu einem Wort formen ließen. Dieses Schreibsystem wurde damals von unterschiedlichen Sprachbevölkerungsgruppen wie die Akkadiian, die Assyrian, die Babylonians und die Persians übernommen.

Im Laufe der Jahre haben sich die Schreibsysteme in unterschiedlichen Regionen zu vielfältigen Formen weiterentwickelt. Aus der Vielfalt der existierenden Schreibsysteme und dessen Untergruppen wird an dieser Stelle nur auf die chinesischen, die indischen, und die arabischen Schreibsysteme eingegangen, da diese in großen Teilen der Welt Anwendung finden.

2.2.1 Chinesisches Schriftsystem

Die ältesten chinesischen Texte stammen aus dem frühen 14. Jahrhundert v. Chr. Sie wurden eingeritzt in Schildkrötenpanzer oder Schulterblätter von Rindern gefunden. Das Schreibsystem besitzt keine alphabetische Ordnung, sondern ist eine Symbolschrift oder auch in der Fachsprache ideographische Schrift. Dies bedeutet, dass zu jedem Wort ein Schriftzeichen zugeordnet wird. Insgesamt gibt es über 40 000 Zeichen. Im Laufe der Entwicklung hat sich das Schriftsystem standardisiert und verändert, trotzdem sind seine Grundprinzipien und viele der Symbole erhalten geblieben. Im Gegensatz zu anderen Schriften enthält ein chinesisches Schriftzeichen Hinweise auf seinen Bedeutungsbereich und zu seiner phonetischen Realisierung. In der Abbildung 1 werden einige Beispiele für die Entwicklung der chinesischen Zeichen und ihre Herkunft verdeutlicht. Die chinesischen Schriftzeichen existieren unabhängig von der Sprache. Sie werden in verschiedenen Regionen anders ausgesprochen, jedoch haben sie überall die gleiche Bedeutung.

Abbildung 1: Entwicklung der chinesischen Zeichen

Die Zeichen setzten sich aus genormten Strichen zusammen, die bis zu 33 Striche enthalten können. Die Reihenfolge der Striche ist durch Regeln festgelegt, dadurch lassen sich die Zeichen flüssiger schreiben. Alle Striche haben ihren eigenen Namen. Hier einige Beispiele wie die Striche aussehen.[4]

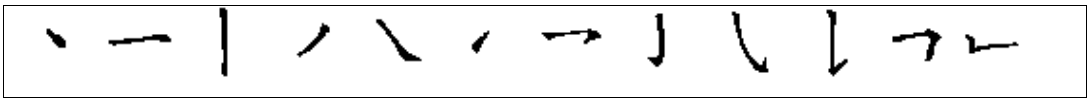


Abbildung 2: Chinesische Striche

Aus diesen Strichen werden Strichgruppen gebildet. Die Strichgruppen werden auch als Radikale genannt, die im chinesischen Lexikon je nach System als 186 oder 227 Strichgruppen aufgefasst werden. Die Strichgruppen sind mit lateinischen Buchstaben vergleichbar. In der Druckschrift bauen sich die Zeichen aus stets gedachten kleinen Quadraten. In diese Quadrate werden die Strichgruppen eingepasst. Die meisten Schriftzeichen bestehen aus eins bis vier Strichgruppen. Die Abbildung 3 zeigt oberhalb das jeweilige Schriftzeichen und darunter die darin enthaltenen Strichgruppen.

Abbildung 3: Chinesische Schriftzeichen

[4] Vgl. [SchChina]

2.2.2 Indisches Schriftsystem

Die Ursprünge der indischen Schriften sind ungeklärt. Die ältesten Inschriften stammen von Kaiser Asoka (270 - 232 v. Chr.). Sie wurden in Nordindien gefunden und sind in Kharosthi und Brahmi geschrieben. Von der Brahmin Schrift leiten sich über 200 verschiedene moderne indische Schriften ab. Die darunter meist verbreiteten Schriften sind Devanagari, Gujarati, Punjabi, Bengali, Oriya, Telungu und Tamil[5]. Die Schrift Devanagari wurde für die heilige Sprache Sanskrit benutzt und findet jetzt Anwendung in der Sprache Hindi. Die indischen Schriften werden als komplexes Schreibsystem betrachtet, weil es, wie das arabische Schreibsystem, eine komplexe Bindung der Zeichen beinhaltet. Hier ein Beispiel für das Brahmi-Zeichen „Na“ in den verschiedenen modernen indischen Schriften.

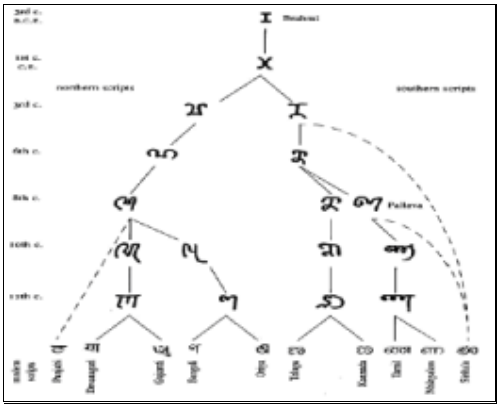


Abbildung 4: Entwicklung der Brahmi-Zeichen „Na“

Aus der Vielfalt der existierenden indischen Schriften wird hier nur das tamilische Schreibsystem vorrangig vorgestellt um die Komplexität zu verdeutlichen. Obwohl der Ursprung der meisten modernen indischen Schriften aus der Brahmi Schrift hervorgehoben wird, sind die Schriften in ihre Form und Schreibstile völlig unterschiedlich. Ein Beispiel hier für sind die Schriftsysteme Devanagari und Tamil.

Im Devanagari wird manchmal die Zeichenreihenfolge in den Wörtern anders dargestellt als sie eingegeben wurde. Ein Beispiel hierfür ist in der nachfolgenden Abbildung dargestellt. Auf der linken Seite wird das Wort „Hindi“ in Devanagari eingegeben und auf der rechten Seite sieht man, wie das Wort dargestellt wird.

---

[5] Eine Einführung in die indischen Schriften ist unter [SchIndien] zu finden.

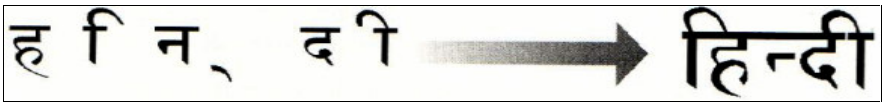


Abbildung 5: Das Wort „Hindi“[6]

Im Gegensatz zu Devanagari ist die tamilische Schrift völlig anders geformt und die Darstellungsreihenfolge entspricht der lateinischen Schrift. Die tamilischen Schriftzeichen bilden sich aus der Kombination von Konsonanten und Vokale. Heraus die Gesamtzahl der Zeichen beträgt 247 und setzt sich aus der Kombination von 18 Konsonanten und 13 Vokalen zusammen. Obwohl die Anzahl der tamilischen Buchstaben ganz beträchtlich scheinen, setzen sie sich aus immer wiederkehrenden Hilfssymbolen zusammen. Die Vokale können im Wort einzeln vorkommen oder sie verschmelzen mit den Konsonanten und bilden ein neues Zeichen mit dem jeweiligen Laut. Die verschiedenen Laute des Konsonanten werden meisten mit den Hilfssymbolen kombiniert. Alle Stammkonsonanten haben den Vokal „a“ in sich, so dass zum Beispiel der erste Konsonant „k“ als ka ausgesprochen wird. Die kompletten Zeichen der tamilischen Schrift sind im Anhang zu finden.[7]

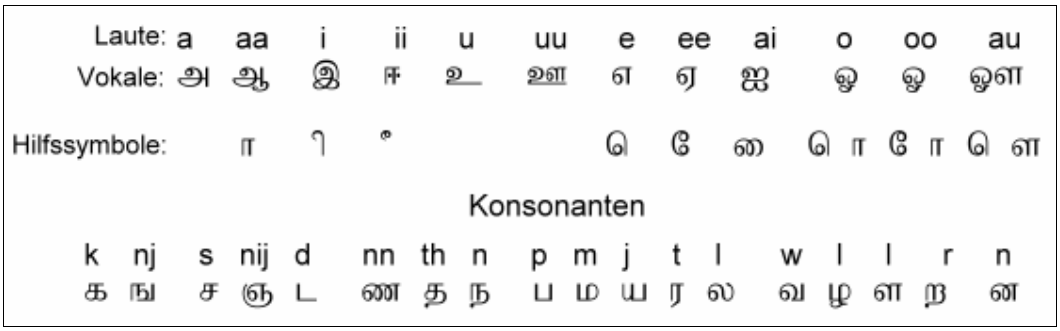


Abbildung 6: Tamilische Schriftzeichen[8]

Das Wort „Tamil“ wird in tamilisch als „thamill“ ausgesprochen und setzt sich aus den folgenden drei Zeichen zusammen.

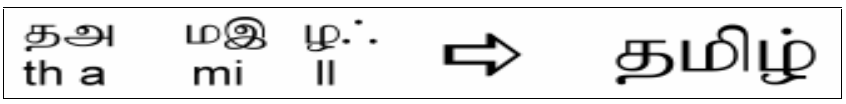


Abbildung 7: Das Wort „Tamil“

[6] Quelle: [DeiCza2001]

[7] Vgl. [TamZei]

[8] Im Anhang kann die komplette Tamilzeichen-Matrix eingesehen werden.

2.2.3 Arabisches Schriftsystem

Der Vorläufer der hebräischen und arabischen Schrift war das Aramäische Alphabet, welches ungefähr 800 v.Chr im heutigen Syrien praktiziert wurde. Die arabische Schrift hatte größtenteils durch den islamischen Propheten Mohammed um ca. 500 n.Chr ihre Verbreitung gefunden. Seit 1947 ist die Schrift offiziell anerkannt und wird von 22 arabischen Staaten in Asien und Afrika verwendet.

Das arabische Schriftsystem umfasst 26 Konsonanten und 3 Vokale, von denen zwei auch konsonantischen Wert haben können. Die Unterscheidung zwischen Großbuchstaben gibt es nicht, aber die Buchstaben haben unterschiedliche Höhen und Breiten. Das Alphabet ist eine reine Konsonantenschrift, die von rechts nach links in Kursivstil geschrieben und gelesen wird. Die Buchstaben werden bis auf sechs nach beiden Seiten verbunden geschrieben. Arabische Buchstaben verändern ihre Form in Abhängigkeit von ihrer Stellung im Wort. Vier Varianten werden unterschieden, je nachdem, ob der Buchstabe in der Anfangs-, Mittel-, bzw. Endposition steht oder aber isoliert betrachtet wird. Welche Form ein Buchstabe schließlich annimmt, hängt von seinem rechten bzw. linken Nachbarn ab. Eine weitere Besonderheit des Arabischen besteht in der Existenz von Ligaturen, dass heißt der Verbindung von verschiedenen Buchstaben zu einer Drucktype. Die folgende Abbildung zeigt auf der linken Seite den Text: „Das ist ein neues Buch.“ in Arabisch und dessen Einzelbuchstaben auf der rechten Seite.

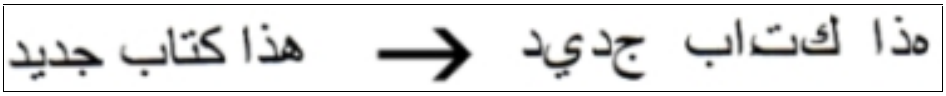


Abbildung 8: In Arabisch „Das ist ein neues Buch“

Während die arabischen Buchstaben von rechts nach links geschrieben werden, schreibt man die Zahlen von links nach rechts, wodurch ein Wechsel in der Schreibrichtung erfolgt. Gleiches geschieht auch beim Einfügen von Wörtern aus lateinischen Buchstaben, die ja ebenfalls von links nach rechts geschrieben werden müssen. Stehen arabische Wörter gemeinsam mit solchen aus lateinischen Buchstaben oder auch mit Zahlen auf einer Zeile, so spricht man von bidirektionalem Text, da die Schreibrichtung wechselt. Die hier nur kurz geschilderten Probleme stellen besondere Anforderungen an die Entwicklung von Software für den arabischen Raum dar.

2.3 Kulturelle Unterschiede

Die kulturellen Unterschiede wie Farben, Zahlen, Symbole und unterschiedliche Formate werden oft in der Softwareentwicklung unterschätzt, da sie vielleicht weniger offensichtlich sind als verschiedene Sprachen und Schriften. Nicht nur die kulturellen Unterschiede spielen auf dem Globalmarkt eine starke Rolle in der Entwicklung von Software, sondern auch die Wertvorstellungen in vielen Ländern und Regionen sind unterschiedlich geprägt. Wertvorstellungen sind zum Beispiel: Gleichberechtigung der Geschlechter und Rassen, Sexualität, Abtreibungen, Tierschutz, Steuern, Krankheiten, Familienwerte usw. Die wesentlichen kulturellen Unterschiede bezüglich der Softwareentwicklung werden hier kurz erläutert.

2.3.1 Farben

Farben haben einen wesentlichen Einfluss auf den ästhetischen Eindruck einer Anwendung. Farben haben in unterschiedlichen Kulturen eine unterschiedliche Bedeutung. Während in westlichen Ländern Schwarz als Trauerfarbe gilt und Weiß zur Hochzeit getragen wird, symbolisiert in Südost Asien die Farbe Weiß den Tod. Im Hinduismus tragen die Witwen in der Öffentlichkeit weiße Kleider. Die Farbe Rot hat in den asiatischen Gebieten vielerlei Bedeutung. In China gilt rot als festliche Stimmung und in Indien tragen verheiratete Frauen einen roten Punkt auf ihrer Stirn. Die untere Tabelle zeigt einige Beispiele für kulturspezifische Farbbedeutungen.

	Westeuropa	Japan	China	Arab. Länder
Rot	Gefahr	Gefahr, Aggression	Freude	
Gelb	Vorsicht, Feigheit	Würde, Adel		Glück, Wohlstand
Grün	Sicher	Zukunft, Jugend, Energie		Fruchtbarkeit, Stärke
Blau	Männlichkeit, Kälte, Autorität			Tugend, Vertrauen Wahrheit
Weiß	Reinheit, Tugend	Tod, Trauer	Tod, Trauer	
Schwarz	Tod, Trauer			

Abbildung 9: Kulturspezifische Farbbedeutungen[9]

[9] Weiterführende Informationen unter [Farben]

2.3.2 Weitere Faktoren

Nicht nur auf Farben, sondern auch weitere kulturspezifische Merkmale wie Grafiken, Symbole, Zahlen und Wertvorstellungen müssen bei der Internationalisierung von Software beachtet werden.
Für die optische Darstellung von Informationen werden häufig Symbole oder Grafiken als Repräsentanten in den Programmen verwendet. Symbole sind in ihrer Interpretation besonders kulturabhängig. Sie können zu massiven Verständnisschwierigkeiten bis hin zur totalen Ablehnung führen. Ein Haken steht in den meisten westlichen Ländern für richtig oder gut, aber in anderen Kulturen kann er genau das Gegenteil bedeuten, nämlich „falsch“. Weiterhin gibt es noch kulturell verschiedene Vereinbarungen, um z.B. graphische Elemente in Tabellen, in Diagrammen oder Koordinatensystemen darzustellen.

Auch Zahlen haben Bedeutungen welche nicht unterschätzt werden sollte. So steht die Zahl 13 in westlichen Ländern für Pech, ist das die Zahl 4 in den östlichen Ländern für Tod. So z.B. sucht man in Hongkong vergeblich nach dem vierten Stock in einem Gebäude, während 8 als Glückzahl betrachtet wird.

2.4 Unterschiedliche Formate

Nicht nur die kulturellen Unterschiede bereiten Probleme bei der Softwareentwicklung, sondern auch die Darstellung von Informationen. Neben unterschiedlichen Textlängen je Sprache sind die kultursensitiven Textformate wie Datum und Zeit, Zahlen, Kalender usw. zu beachten, die vor der Darstellung nach lokalen Vorschriften richtig formatiert werden müssen. Hier einige weitere Beispiele, die nicht fest in Programmen eingebunden werden sollten.

- Datum, Zeit, Kalender, Zeitzone
- Maße/Gewichte/Normen
- Gesetzliche Vorschriften
- Zahlen, Währung
- Papierformate
- Adressen-Labels

Aus den genannten Beispielen werden die wichtigsten Formatierungen von Datum und Zeit, Kalender und Zeitzone näher betrachtet.

2.4.1 Datum und Uhrzeit

Die Darstellungsformen von Datum und Uhrzeit variieren nach den lokalen Konventionen. Ein Beispiel hierfür ist die Interpretation des Datums 03/10/1999 in Amerika. Hierzulande würden wir das Datum als 3. Oktober 1999 verstehen, hingegen wird in den USA darunter der 10. März 1999 verstanden. Die Chinesen und Japaner schreiben üblicherweise das Datum in umgekehrter Reihenfolge und zwar als erstes das Jahr und dann den Monat und zum Schluss den Tag. In Hebräisch gibt es kein Zahlensystem in unserem Sinne, sondern jeder Buchstabe entspricht einem Zahlenwert. So möchte natürlich jeder seine Datumsanzeige in seinem gewohnten Format angezeigt bekommen, mit sämtlichen Besonderheiten der Zeichen. In vielen Gebieten werden auch mehrere verschiedene Formate akzeptiert.

Das folgende Beispiel zeigt nicht nur die Unterschiede der sprachlichen Groß- und Kleinschreibung bei der Darstellung des Datums in den USA und Frankreich, sondern auch die Trennzeichen zwischen den Formaten.

USA	Frankreich
10/04/99	04/10/99
October 4th, 1999	4 octobre 1999
Monday, October 04, 1999	lundi 4 octobre 1999
Mon, Oct 4, 1999	lun. 4 octobre 1999
4-OCT-99	4-oct.-99

Die Zeitformate unterscheiden sich wesentlich durch zwei Formen. In den europäischen Ländern ist die Darstellung der Zeit am 24-Stunden-Rhythmus orientiert, im Gegensatz zu Amerika und einigen asiatischen Ländern in denen der 12-Stunden-Rhythmus herrscht. In Amerika wird von Mitternacht bis Mittag der Begriff a.m. hinten angestellt, für die Zeit zwischen Mittag und Mitternacht p.m. Eine Zeitangabe wie „open 7 to 11“ kann sowohl 7.00 bis 11.00 Uhr, 19.00 bis 23.00 Uhr oder aber 7.00 bis 23.00 Uhr bedeuten.

2.4.2 Kalender

In verschiedenen Kulturen kommen unterschiedliche Kalendersysteme zur Geltung. Der Westen orientiert sich am gregorianischen Kalender, der von Papst Gregor XIII im Jahre 1582 eingeführt wurde, um den julianischen Kalender zu ersetzen. Um die Länge des astronomischen Jahres (365.2422 Tage) besser anzugleichen, legt dieser Kalender fest, dass es sich bei Jahren, die durch 100 geteilt werden können, nur dann um Schaltjahre handelt, wenn sie auch durch 400 geteilt werden können. Daher war das Jahr 2000 ein Schaltjahr, während 1900 keines war.

Der Orient orientiert sich am islamischen Kalender. Entscheidend für die westliche Zeitrechung ist ein Sonnenjahr, für die islamische ein Mondjahr. Die meisten semitischen Völker orientieren sich am Mondjahr. Dieses besteht aus zwölf Monaten, die abwechselnd 29 oder 30 Tage enthalten. Ein Jahr enthält damit - im Gegensatz zum Sonnenjahr, nur 354 Tage und acht Stunden. Deswegen wird alle zwei Jahre ein Schalttag hinzugerechnet. Die Differenz zum Sonnenjahr beträgt damit zehn bzw. elf Tage. Jedes Jahr verschieben sich somit auch die Feiertage.

So wie der islamischen Kalender existieren noch weitere Kalendersysteme in unterschiedlichen Kulturen. In Indien wird im Alltag ein astrologischer Kalender verwendet, der auch als vedischer Kalender[10] oder als „Panchangam“ bezeichnet wird. Der vedische Kalender ist weit komplizierter als der einfache gregorianische Kalender. Er liefert Informationen über die Konstellation der Planeten und der Sterne und welche Auswirkungen sie auf die menschliche Psyche besitzen. Das Wort Panchangam bedeutet in Sanskrit „fünf Glieder“, und steht für die fünf grundlegenden Elemente: Name des Tages, Mondphase, Mondhaus, Winkel zwischen Sonne und Mond, und Halbtag. Der Kalender variiert je nach Standort.

[10] Eine ausführliche Beschreibung des vedischen Kalenders befindet sich unter [Siva97]

2.4.3 Zeitzonen

Wegen der Erddrehung ist die Zeit an unterschiedlichen Orten nicht genau gleich. Die Sonne geht z.B. in Berlin früher auf als in München. Man hat festgestellt, dass pro 15° Längenunterschied die Ortszeit um genau 1 Stunde abweicht. Daher wurde die Erde auf der Weltkarte in 24 Stundenzonen (24*15=360) unterteilt, die wir Zeitzonen nennen. Innerhalb einer Zeitzone wurde die gleiche Uhrzeit festgelegt. Bezugspunkt für die Zeitzoneneinteilung ist der Null Meridian, der 0.Längengrad der durch Greenwich bei London führt. Diese Zeitzone ist die so genannte Weltzeit, GMT oder Greenwich-Time. Vom Nullmeridian nach Westen und nach Osten gehend sind die Meridiane jeweils aufsteigend bis zum 180°-Längenkreis nummeriert. Dieser liegt genau gegenüber dem Nullmeridian. Der 180. Längengrad trennt die beiden Tage voneinander, weshalb er Datumsgrenze genannt wird.

Für den Übergang von einer Zeitzone westlich von Greenwich zu einer anderen wird jeweils 1 Stunde abgezogen, für die Zeitzonen östlich von Greenwich dazugezählt. Die Differenz zur Zeitzone beträgt eine Stunde, d.h. wenn es in London z.B. 4 Uhr ist, so ist es in München 5 Uhr. Praktisch folgt die Zeitzoneneinteilung nicht genau dem theoretischen Modell, bei dem jeweils nach 15 Längengraden eine Zeitzone beginnt, sondern sie fallen mit den Landesgrenzen zusammen. In welcher Zeitzone ein Land liegt und ob es mehrere Zeitzonen beinhaltet, ist eine politische Entscheidung des jeweiligen Landes. Während beispielsweise in Russland mehrere Zeitzonen gelten, verwendet China nur eine einzige Zeitzone, obwohl sich das Land über etwa 60 Längengrade erstreckt.

Das zweite Problem ist die Zeitumstellung in einigen Zeitzonen. Um mehr Tageslicht zu erhalten wird im Winter die Zeit um eine Stunde zurückgestellt und im Frühling wieder auf die ursprüngliche Zeit umgestellt. In den Staaten Arizona, Hawaii, Teile von Indiana, Puerto Rico, Virgin Islands und American Samoa der USA findet eine Zeitumstellung im Gegensatz zu anderen Staaten nicht statt. [11]

---

[11] Vgl. [DeiCza2001]

3. Praktische Lösungen zur Internationalisierung von Software

In diesem Kapitel werden mögliche Lösungsansätze zu den in vorherigem Kapitel aufgelisteten Problematiken der Internationalisierung von Software behandelt. Dabei werden die Lösungen anhand von kleineren Beispielen erläutert. Diese Beispiele orientieren sich an die Programmiersprache Java, welches auch für die Implementierung des praktischen Teils verwendet wurde. Java ist die einzige Sprache die von Beginn an auf die Unterstützung der Internationalisierung von Anwendungen ausgelegt war.

Neben den Standard-Java-Klassen existiert von der Firma IBM eine erweiterte Klassenbibliothek für die spezielle Unterstützung der Internationalisierung. Die Bezeichnung dafür lautet „International Components for Unicode“[12] und wird als „ICU“ abgekürzt. Die ICU ist sowohl für Java mit der Bezeichnung ICU4J als auch für C mit ICU4C verfügbar. Die Bibliothek liegt unter XLicense[13], welches mit der GPL-Lizenz[14] kompatibel ist.

Sowohl im praktischen als auch im schriftlichen Teil dieser Arbeit wurde mit den Standard Klassenbibliotheken der Java 2 Standard Edition (J2SE) v. 1.4.1 gearbeitet.

---

[12] Nähres zu ICU unter [ICU]

[13] Siehe Anhang ICU License – ICU 1.8.1 and later

[14] Siehe Anhang GPL-Compatible, Free Software Licenses

4.1 Zeichensätze

Das Grundproblem des existierenden Zeichensatzes für Schriften war lange Zeit ein Hindernis für die Verbreitung von Software auf dem Globalmarkt. Seit der Standardisierung der Zeichensätze durch den Unicode konnte dieses Hindernis bewältigt werden. Bevor ich auf den Standard Unicode eingehe, werde ich hier kurz die historisch entwickelten Zeichensatzkodierungen behandeln und die daraus entstehenden Probleme erläutern.

Der Standard-Zeichensatz ist ASCII, der 1965 vom ANSI festgelegt wurde. Er enthält die lateinischen Grundbuchstaben, Zahlen und einige sonstige Zeichen, aber keine durch Sonderzeichen erweiterten Buchstaben wie Umlaute oder Akzentbuchstaben. Ein Zeichen bestand aus 7 Bit und umfasste 128 Zeichen. Anfang der 70er-Jahre wurden die 8-Bit-Zeichensätze mit jeweils 256 Codeplätzen entwickelt. Die ersten 128 Zeichen sind identisch mit dem ASCII-Zeichensatz. Die restlichen 128 Plätze enthalten je nach Sprache und Computersystem verschiedene Sonderzeichen. Da die Codierung für die zweite Hälfte der Zeichen von Computer zu Computer unterschiedlich war, wurde der Informationsaustausch sehr schwierig. Die Internationale Organisation für Normung ISO begegnete diesem Problem in den 80er-Jahren mit der Serie der ISO-8859-Standards. Der Zeichensatz für westeuropäische Sprachen ist unter ISO-8859 festgehalten. Der Zeichensatz ISO-8859-5 für kyrillische Schriftzeichen, ISO-8859-6 für arabische etc. Neben den ISO-8859-Standards gibt es etliche andere, häufig landesspezifische Zeichensätze.

Das Problem dieser Zeichensätze ist, dass es keine einheitliche Norm gibt, d. h. dass die Zeichen in den verschiedenen Code Pages durch unterschiedliche Nummern repräsentiert werden und umgekehrt derselben Codenummer verschiedene Zeichen zugeordnet sind. Ein besonderes Problem bildet die Darstellung asiatischer Schriftsysteme. Sprachen wie Chinesisch, Japanisch und Koreanisch haben Zehntausende von Schriftzeichen. Eine 8-Bit-Codierung mit 256 Zeichen reicht hier nicht aus. Deshalb verwendet man in asiatischen Code Pages eine 16-Bit-Codierung.

Ein weiteres Problem dieser Kodierungssysteme ist die Unverträglichkeit untereinander. Somit muss jeder Rechner viele verschiedene Kodierungssysteme unterstützen und der Austausch von Texten unterschiedlicher Kodierungssysteme erschwert sich zwischen den Rechnersystemen.

4.2 Unicode

Der Unicode beseitigt das Problem der verschiedenen Kodierungen der Zeichensätze in den unterschiedlichen Ländern. Unicode[15] (dt. universeller Schlüssel) strebt die möglichst vollständige Erfassung aller bekannten Zeichen aus gegenwärtigen und vergangenen Schriftsystemen an. Sie gibt für jedes Zeichen unabhängig von Sprache, Plattform und Programm eine eindeutige Identifikationsnummer. Für die Erfassung und Erweiterung der Zeichensätze in Unicode ist das Unicode-Konsortium verantwortlich. Das Unicode-Konsortium ist eine gemeinnützige Organisation, die aus Linguisten und anderen Fachleuten im Jahre 1991 gegründet wurde. Mit der Version 2.0 wurde das Unicode-System in den ISO-Standard aufgenommen und wird von vielen namenhaften Firmen wie Apple, HP, IBM, Microsoft, SAP, Sun etc. eingesetzt.

Das gesamte Unicode-System ist in Zahlenbereiche aufgeteilt, welches jeweils ein bestimmtes Schriftsystem oder ein Set von Sonderzeichen repräsentiert. Jedes Zeichen erhält einen Zeichenwert. Die Zahlen selbst werden in der Form U+XXXX notiert. Das U steht für Unicode, und die X für je eine hexadezimale Ziffer. Ebenso wie bei den herkömmlichen Zeichensätzen sind auch im Unicode-System die ersten 128 Codeplätze mit den ASCII-Zeichen belegt. Das Unicode-Konsortium entscheidet darüber, ob ein Zeichen neu aufgenommen wird und ordnet diesem einen Zeichenwert zu. Das hat den Vorteil, dass es im gesamten System für jedes Zeichen nur einen Zeichenwert gibt und keinem Zeichenwert mehrere Zeichen zugeordnet sind.

---

[15] Siehe [Unicode]

Im Gegensatz zu anderen Zeichensätzen ist in Unicode für jedes Zeichen ein Set von Eigenschaften definiert. Zu den Eigenschaften eines Zeichens gehört zum Beispiel die Schreibrichtung. Im Arabischen und im Hebräischen etwa ist die Schreibrichtung von rechts nach links. Die Komplexität bei asiatischen Schriften lassen sich durch dynamische Kombinationen von Zeichen verringern. So umfasst zum Beispiel das in meinem Programm verwendete Schriftsystem Tamil 247 Zeichen, aber viele dieser Zeichen lassen sich aus eine Kombination von Zeichen und Elemente vereinfachen. Ein weiteres Beispiel ist das deutsches „ä“, welches sich auch aus „a“ und dem Element für Doppelpunkt über dem Zeichen erzeugen lässt. Insgesamt stecken hinter dem Unicode-System unzählige Forschungsergebnisse der weltweiten Sprachwissenschaft.

4.3 Isolierung von übersetzbaren Texten

Um eine Software in verschiedenen Sprachen anzubieten, bedarf es schon vor der Implementierungsphase, dass alle sämtlichen vorkommenden statischen Texte sowie alle übrigen sprachabhängigen Inhalte vom Programmcode getrennt werden. Wesentliche in Programmen vorkommende Texte sind Statusmeldungen, Fehlermeldungen und Beschriftungen der GUI Komponenten. Für diesen Zweck bietet Java in den Paketen *java.text* und *java.util* leistungsfähige Klassen an, welche dem Entwickler die Hauptarbeit bei der Anpassung von Sprachen in der Software abnehmen. Die hierfür in Java zuständigen Hauptkassen *Locale* und *ResourceBundle* sowie Propertiedatei werden im Folgenden näher erläutert.

4.3.1 Indentifikation von Locale

Die Verknüpfung von Internationalisierung und Lokalisierung ist das *Locale*, welches einen „Platz“ definiert. Ein Platz kann die Kombination von Land, Sprache und ein selbst definiertes Merkmal haben. Zur Laufzeit bestimmt das Programm die aktuelle *Locale*. Dies kann auch durch statischen Aufruf bei der Programmausführung oder

dynamisch durch die Anforderung des Benutzers erfolgen. Jedes Programm besitzt ein Standard *Locale*, eine bevorzugte Einstellung. In Java sind die dazugehörigen Einstellungen in der Klasse *Locale* im *Package java.util* zusammengefasst.

Das Objekt *Locale* dient für die Identifikation von *Resourcebundles* und hat drei Parameter. Das erste Argument ist der Sprachcode mit zwei Kleinbuchstaben nach dem ISO-639 Standart. Das zweite Argument des *Locale Konstruktors* ist der Ländercode. Dieser besteht aus zwei Großbuchstaben und entspricht der ISO-3166. Falls eine weitere Unterscheidung der *Locale* Objekte benötigt wird, kann man ein selbst definiertes drittes Merkmal aufführen. Eine typische Unterscheidung ist die Plattform, auf der das System läuft. Für mein Programm sieht das *Locale* Objekt für die Sprachen Englisch, Deutsch und Tamilisch wie folgt aus.

Public Locale(String language, String country, String variant);

deLocale = new Locale(„de“, „DE“); // für deutsch und Deutschland

enLocale = new Locale (“en”); // für englisch

taLocale = new Locale(“ta”); // für tamilisch

Neben diesen Grundeinstellungen bietet die Klasse *Locale* weitere Funktionen für die Identifikation der Kulturgesellschaft an.

4.3.2 Trennen der locale - spezifischen Daten

Nach der Identifizierung des Locales wird sich nachfolgend mit der Trennung der Benutzersprache und der Region befasst. Hierfür bietet Java die Klasse *ResourceBundle* in *java.text package* an. Bevor man jedoch mit der Erzeugung von *ResourceBundle* Objekt beginnen kann muss genau geplant werden, welche Objekte in welcher Art und Weise von den lokalen Anpassungen betroffen sind. Die gewünschten Anpassungen an die Objekte sind in den Objektkategorien einzuteilen, um eine übersichtliche Struktur zu schaffen. Die einzelnen Kategorien werden dann auf die *ResourceBundle* abgebildet.

Die abstrakte Klasse *ResourceBundle* in Java bietet zwei Unterklassen: *PropertyResourceBundle* und *ListResoureceBundle*.

Ein *PropertyResourceBundle* wird durch eine Propertiesdatei unterstützt. Eine Propertiesdatei ist eine reine Textdatei, welche den übersetzbaren Text enthält. Propertiesdateien sind Teil des Java Quellcodes und enthalten Werte für Zeichenkettenobjekte. Falls andere Objekttypen abgespeichert werden müssen, kann auf die *ListResourceBundle* Klasse zugegriffen werden. Dadurch lassen sich beliebige Objekte implementieren. Die Objekte werden in diesem Fall in Form einer Liste angeordnet.
Konzeptionell ist jedes *ResourceBundle* ein Set von zusammenhängenden Unterklassen, welche denselben Basisnamen haben.

Die folgende Abbildung 10 zeigt einige Möglichkeiten der Kombinationen von Sprachcode, Ländercode und einen weiteren Unterscheidungscode für ein ResourceBundle.

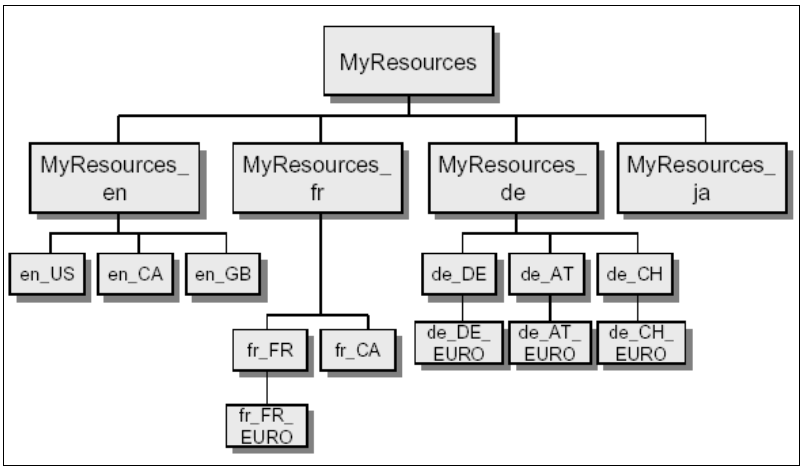


Abbildung 10: MyResources Struktur

Der passende ResourceBundle wird mit der *ResourceBundle.getBundle()* Methode ausgewählt. Im folgenden Beispiel selektiert das *PatientLabel-ResourceBundle* die Locale Objekte, welche tamilisch in Indian und Windows unterstützen.

Locale currentLocale = new Locale(“ta”,“IN“,“Windows“);

ResourceBundle patientLabels = ResourceBundle.getBundle(“PatientLabel”, currentLocale);

Falls das *ResourceBundle* Objekt für das betreffende *Locale* Objekt nicht gefunden wird, versucht die Methode das nächste Objekt zu finden, jenes *ResourceBundle* Objekt, welches möglichst viele der Auswahlkriterien erfüllt. Die Reihenfolge für das obige Beispiel, mit der der Mechanismus eine Alternative sucht, sieht in diesem Fall folgendermaßen aus:

PatientLabel_ta_IN_Windows

PatientLabel_ta_IN

PatientLabel_ta

PatientLabel_StandadSprache_StandardLand

PatientLabel

Die eigentlichen Übersetzungsinformationen sind in den Propertiesdateien gespeichert. Eine Propertiesdatei ist eine reine Textdatei, welches Schlüsseln mit den dazugehörigen Werten enthält. Es lässt sich mit jedem Texteditor kreieren. In meinem Programm sehen die PatientLabel-Propertiesdateien für die drei verschiedenen Sprachen wie folgt aus.

PatientLabel.properties	PatientLabel_en.properties
# PatientLabel - Standard	# PatientLabel - englisch
Lastname = Lastname	Lastname = Lastname
Firstname = Firstname	Firstname = Firstname
City = City	City = City

Die Probertiesdateien enthalten <Schlüssel, Wert> Paare. Die Werte bestehen aus dem übersetzten Text, den das Programm für die ausgewählte Sprache anzeigen soll. Um den jeweiligen Text aus dem *ResourceBundle* zu lesen, benötigt man den Schlüssel und die *getString()* Methode der *ResourceBundle* Klasse.

```
// PatientLabels
JLabel firstnameLabel = new JLabel();
firstnameLabel.setText(patientLabels.getString("Firstname"));
JLabel lastnameLabel = new JLabel();
lastnameLabel.setText(patientLabels.getString("Lastname"));
JLabel cityLabel = new JLabel();
cityLabel.setText(patientLabels.getString("City"));
```

In meinem Programm wurden alle visuellen Texte, Meldungen, Icons, sowie Schriftarten und Größen aus dem Quellcode isoliert in Propertiedateien aufgefasst.

4.4 Formatierungen

In diesem Kapitel werden die verschieden Lösungsmöglichkeiten für die Formatierung von Zahlen, Datum, Uhrzeit, Kalender und Zeitzonen in Java vorgestellt. Im Programm werden die Daten in einer locale - unabhängigen Darstellung gespeichert. Sie müssen also vor der Ausgabe auf dem Bildschirm oder dem Drucker die Daten passend für den entsprechenden locale formatieren.

Mit Hilfe der Number-Format-Klasse können die numerischen Daten wie Zahlen, Währungen und Prozentdarstellungen gemäß den Locale-Objekten formatiert werden. Weiterhin stehen Anpassungsmöglichkeiten sowie eigene Formatierungsmuster zur Verfügung. An dieser Stelle einige Beispiele mit der Klasse.

```
NumberFormat numberForamt, currencyFormat, percentFormat;
String mengeOut, betragOut, prozentOut;
Integer menge = new Integer(555545);
Double betrag  = new Double(436342.323);
Double prozent = new Double(0.25);

numberFormat = NumberFormat.getNumberInsatance(currentLocale);
mengeOut  = numberFormat.format(menge);
currencyFormat = Numberforamt.getCurrencyInstance(currentLocale);
betragOut = currencyForamt.format(betrag);
percentFormat = NumberFormat.getPercentInstance(currentLocale)
prozentOut = percentFormat.format(prozent);
System.out.println("Menge: " + betragOut);
System.out.println("Währung: " + betragOut);
System.out.println("Prozent: " + prozentOut);
```

Locale_de_DE	Locale_en_US	Locale_fr_FR
Menge: *555.545* Währung: *4.363.423,23DM* Prozent: *25%*	Menge: *555,545* Währung: $*4,363,423.23* Prozent: *25%*	Menge: *555 545* Währung: *4 363 423,23 F* Prozent: *25%*

4.4.1 Datum- und Zeitformatierung

Auch die Ausgabe eines Datums oder einer Zeit ist von Region zu Region unterschiedlich. Java bietet zu verschiedenen Datum- und Zeitformatierungen die Klasse *java.text.Dateformat* bzw. *java.text.SimpleDateFormat* an.
Datenobjekte präsentieren die Zeit und das Datum. Es ist nicht möglich ein Datenobjekt ohne irgendeine Konvertierung durchzuführen. Die Formatierung des Datums ist sehr stark an Locale gebunden. Neben frei wählbaren Mustern, mit denen die Ausgabe formatiert werden kann, existieren auch vordefinierte Stiles für Datum und Uhrzeit: Default, SHORT, MEDIUM, LONG, FULL.

```
Date today = new Date();
DateFormat df_de = DateFormat.getDateInstance(DateFormat.[STILE],
Locale.GERMANY);
DateFormat df_us = DateFormat.getDateInstance(DateFormat. [STILE],
Locale.US);
DateFormat df_fr = DateFormat.getDateInstance(DateFormat. [STILE],
Locale.FRANCE);
System.err.println(df_de.format(today));
System.err.println(df_us.format(today));
System.err.println(df_fr.format(today));
```

Stile	GERMANY	US	FRANCE
SHORT	22.04.03	4/22/03	22/04/03
MEDIUM	22.04.2003	Apr 22, 2003	22 avr. 2003
LONG	22. April 2003	April 22, 2003	22 avril 2003
FULL	Dienstag, 22. April 2003	Tuesday, April 22, 2003	mardi 22 avril 2003
SHORT	21:57	9:57 PM	21:57
MEDIUM	22:04:19	10:04:19 PM	22:04:19

Die Uhrzeit lässt sich analog zum Datum aus einem Date Objekt erzeugen. Der Unterschied besteht in der Methode *getTimeInstance(),* die die Uhrzeit liefert.

4.4.2 Kalender

In Java existiert eine abstrakte Calendar Klasse, die zur Konvertierung zwischen verschiedenem Datum - Formaten eingesetzt wird. Über Methoden von Calendar ist es möglich, Datum und Uhrzeit in den einzelnen Komponenten wie Jahr, Monat, Tag, Minute, Sekunde zu verändern. Da Calendar eine abstrakte Basisklasse ist, können erst Unterklassen die Attribute und Methoden richtig benutzen. Unter dem Java SDK ist bisher nur die Unterklasse GregorianCalendar implementiert, deren Exemplare Daten und Zeitpunkte gemäß dem gregorianischen Kalender verkörpern. In der Bibliothek icu4j von IBM finden sich im Paket *com.ibm.util* auch die Klassen für weitere Kalendersysteme. In der aktuellen Version 2.6 befinden sich ChineseCalendar, GregorianCalendar, BuddhistCalendar, JapaneseCalendar, HebrewCalendar, und IslamicCalendar Kalendersysteme.

Die Klasse Calendar besitzt nur wenige statische Funktionen, die sofort genutzt werden können. Eine Klassen- und Fabrik-Methode ist *getInstance(),* um ein benutzbares Objekt zu bekommen. Die Methode gibt ein Objekt vom Typ GregorianCalendar zurück und die Werte sind mit der aktuellen Zeit gesetzt. Neben der parameterlosen Variante von *getInstance()* bestehen drei weitere Varianten, denen ein TimeZone-Objekt und Locale-Objekt mit übergeben werden kann. Damit kann dann der Kalender auf eine spezielle Zeitzone und einen Landstrich zugeschnitten werden. Da ein Kalender unter verschiedenen Orten installiert sein kann, gibt *getAvailableLocales()* eine Aufzählung von Locale-Objekten zurück.

4.5 Eingabe Methode

Die westlich orientierten Schreibsysteme sind meistens aus einer relativ kleinen Anzahl von Zeichen zusammengesetzt, so dass sie leicht auf den Standardtastaturen angezeigt werden können. Jedoch besitzen asiatische Schreibsysteme wie chinesisch oder die indischen Schriften eine Vielzahl von individuellen Zeichen, die sich auf den Standardtastaturen nicht einfach unterbringen lassen. Um dieses Problem zu lösen wurde in Java 2 Standard Edition Version 1.3 die input Method Editor (IME) vorgestellt, welches im nachfolgenden Abschnitt kurz dargestellt wird. Anschließendem Beispiel für die tamilische Schrift.

Input Methods ist Applikation oder eine Software Komponente, die die vom Benutzer eingegebenen Tasten in Symbole, Zeichen oder Wörter umwandelt. Die IME versucht für jedes getippte Zeichen die eigentlichen Symbole, die der Benutzer haben will, auf dem Bildschirm darzustellen. Die Java Input Method Framework stellt dafür zwei separate Schnittstellen zu Verfügung. Die erste stellt die Clientseitige Unterstützung dar, welches dem Programm als Schnittstelle für die Unterstützung von Eingabe-Methode ermöglicht. Die zweite ist die Input Method Engine Service Provider Interface (SPI). Dieses Interface dient für die Entwicklung von eigenen Eingabe-Methoden. Nachfolgend werden die Hauptmerkmale des Input-Method Engine SPI sowie die Erstellung von einem neuen Input-Method mit dessen Einbindung im Programm vorgestellt. Die SPI besitzt folgende Eigenschaften:

- Die Input-Methods sind als Java Package zusammengetragen und können von jeder Java Applikation genutzt werden.
- Es bedarf keine Neukomplilierung bei der Erweiterung von Input-Methods.
- Input-Methods sind plattformunabhängig. Das heißt, dass sie das Model „Write Once, Run Anywhere“ befolgen.
- Die Input-Methods wird überwiegend für asiatische Schriften verwendet, trotzdem besitzt die SPI die Flexibilität zur Entwicklung von Input-Methods für alle Schriften.
- Die SPI ist von den Eingabegräten unabhängig.

Um die Input-Method SPI zu verdeutlichen wird nachstehend die Entwicklung einer Input-Method für die Indische Schrift „Tamil“ vorgestellt. Dabei kann an diese Stelle nur auf die wesentlichen Funktionen der Klassen Bezug genommen werden. Die zu entwickelnde Input-Methode soll aus der deutschen Tastatur eingegebene Zeichen in tamilische Zeichen umwandeln.

Um eine neue Input-Method zu entwickeln müssen zwei Klassen erzeugt werden, die in der *java.awt.im.spi package* enthaltene Schnittstellen in sich implementieren. Die zwei Schnittstellen in diesem Package sind einmal *java.awt.im.spi.InputMethod* und *java.awt.im.spi.InputMethodDescriptor*. Die beiden Klassen werden später in einer speziellen Datei zusammengefasst, so dass der Input - Method - Framework es finden kann.

4.5.1 Beispiel für TamilInputMethodDescriptor

```
package com.ora.intl.ime;
import java.awt.im.spi.InputMethod;
import java.awt.im.spi.InputMethodDescriptor;
import java.util.Locale;
public class TamilInputMethodDescriptor implements InputMethodDescriptor {
private static Locale TAMIL = new Locale("ta", "IN");
public TamilInputMethodDescriptor() { }
public InputMethod createInputMethod() throws Exception { return new TamilInputMethod();   }
  // This input method supports only one locale.
public Locale[] getAvailableLocales() { Locale[] locales = { TAMIL };     return locales;  }
public boolean hasDynamicLocaleList()   {     return true; }

public String getInputMethodDisplayName(Locale inputLocale, Locale displayLanguage)
 {  return "Tamil Input Method";  }
}
```

Die Abbildung von einzelnen lateinischen Buchstaben in tamilische Zeichen wird durch ein langes Switch Statement in der Methode *handleCharacter(char ch)* in die TamilInputMethode Klasse realisiert.

5. Das ResAdmin - Modul

In diesem Kapitel wird das ResAdmin-Modul, welches im Rahmen dieser Arbeit als Prototyp zur Verwaltung von Patientendaten konzipiert ist, vorgestellt. Dieses Modul ist Teil eines größeren Projektes namens ResMedicinae[16] und gliedert sich in mehrere voneinander unabhängige Softwaremodule. Die einzelnen Module decken autonome Aufgabenbereiche der Anwendung ab. Wie beispielsweise die entwickelten Module Record[17] und Reform[18]. Das Record-Modul dient zur medizinischen Befund-Dokumentation und das ReForm-Modul für das Ausdrucken von Formularen.

Bevor ich auf das ResAdmin-Modul näher eingehe, soll im Folgenden kurz das Rahmenprojekt ResMedicinae und dessen Implementierungskonzept erläutert werden. „Der Kerngedanke bzw. die Intention des Projektes Res Medicinae besteht darin, ein stabiles, erweiterbares und plattformunabhängiges, klinisches Informationssystem auf der Basis von Open Source zu entwickeln. Res Medicinae hat sich zum Ziel gesetzt, der Kommerzialisierung heutiger Softwaresysteme sowie der damit verbundenen Abhängigkeit von einem bestimmten Hersteller entgegenzuwirken.“[19]

Zur Konstruktion von Benutzer-Schnittstellen wurde das Model-View-Controller-Paradigma angewandt, welches aus den drei Klassen Model, View und Controller besteht.
Das Model-Objekt stellt die Kernfunktionalität und das Anwendungsobjekt dar, das View-Objekt seine Bildschirmpräsentation, und das Controller-Objekt bestimmt die Möglichkeiten, mit denen die Benutzer-Schnittstelle auf Eingaben des Benutzers reagieren kann. Das MVC-Konzept erlaubt somit eine klare Schichtentrennung zwischen den Anwendungsdaten, den Sichten auf die Anwendungsdaten und der Benutzer-Schnittstelle.

---

[16] Nähre Informationen unter [Res03]

[17] Siehe [Bohl03]

[18] Siehe [Kunze03]

[19] Zit. [Bohl03]

5.1 Funktionalitäten

Die Aufgabe des Moduls bestand in erste Linie in der Erweiterung der Patientendaten mit Zusatzinformationen wie Kontaktdaten sowie demografischen Daten. Zur Verwaltung dieser Daten wurden weitere Funktionalitäten wie z.B. neue Datenanlagen, Editieren, Suchen, Drucken und Löschen implementiert[20]. Zurzeit lassen sich die unterschiedlich strukturierten Patientendaten von ResForm und ResRecord in ResAdmin verändern sowie mit weiteren Zusatzdaten versehen in XML – Dateien speichern. Als zweite Hauptaufgabe stand die Internationalisierung des ResAdmin Modul im Vordergrund. Auf den folgenden Seiten werden die einzelnen Funktionalitäten vom ResAdmin –Modul[21] mit den Screenshots erläutert.

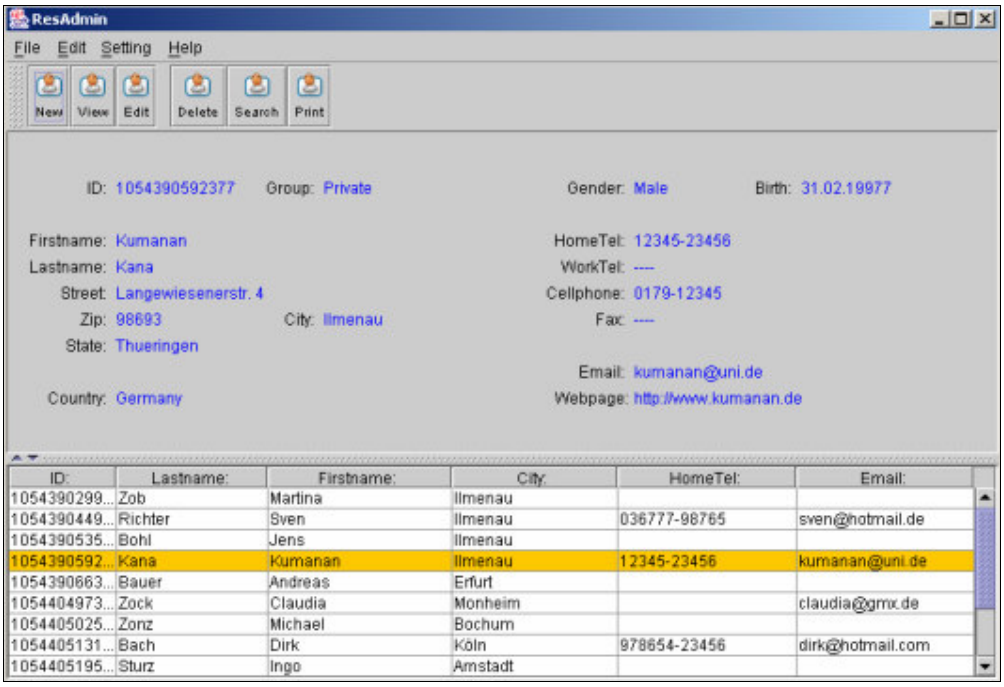


Abbildung 11: Hauptfenster

Das Hauptfenster ist in vier Bereiche (Menü, Toolbar, Datenfelder, Tabelle mit Daten) aufgeteilt. Im Menübereich befinden sich unter „File“ Möglichkeiten zur Datenauswahl,

[20] Siehe Anhang Edit, Delete, Print

[21] Siehe Anhang MainWindow

zum Drucken und zum Verlassen des Programms. Die Sprachauswahl[22] und die Speicherstruktur der XML - Daten befinden sich unter dem Menüpunkt „Setting“.

Die Verwaltung der Daten lässt sich leicht durch das Anklicken der Toolbar - Icons kontrollieren. Je nach Auswahl der einzelnen Icons wechselt die Sicht der Datenfelder für die entsprechende Verarbeitung. Für ein erfolgreiches Speichern der XML - Datei müssen die als rot gekennzeichneten Felder zwingend ausgefüllt werden. Die im unteren Teil des Hauptfensters befindliche Tabelle zeigt die geladenen XML - Daten mit ID, Lastname, Firstname, City, HomeTel und Email-id an. Die Navigation in der Tabelle lässt sich durch den Cursor oder die Maus bedienen. Entsprechend der ausgewählten Datenzeile, die sich durch eine gelbliche Hintergrundfarbe hervorhebt, werden in den oberen Datenfeldern die kompletten Daten angezeigt. Die Abbildung 11 zeigt ein Beispiel hierfür. Für das Löschen der XML- Daten aus der Tabelle ist auch eine mehrfache Auswahl der Datenzeilen möglich.

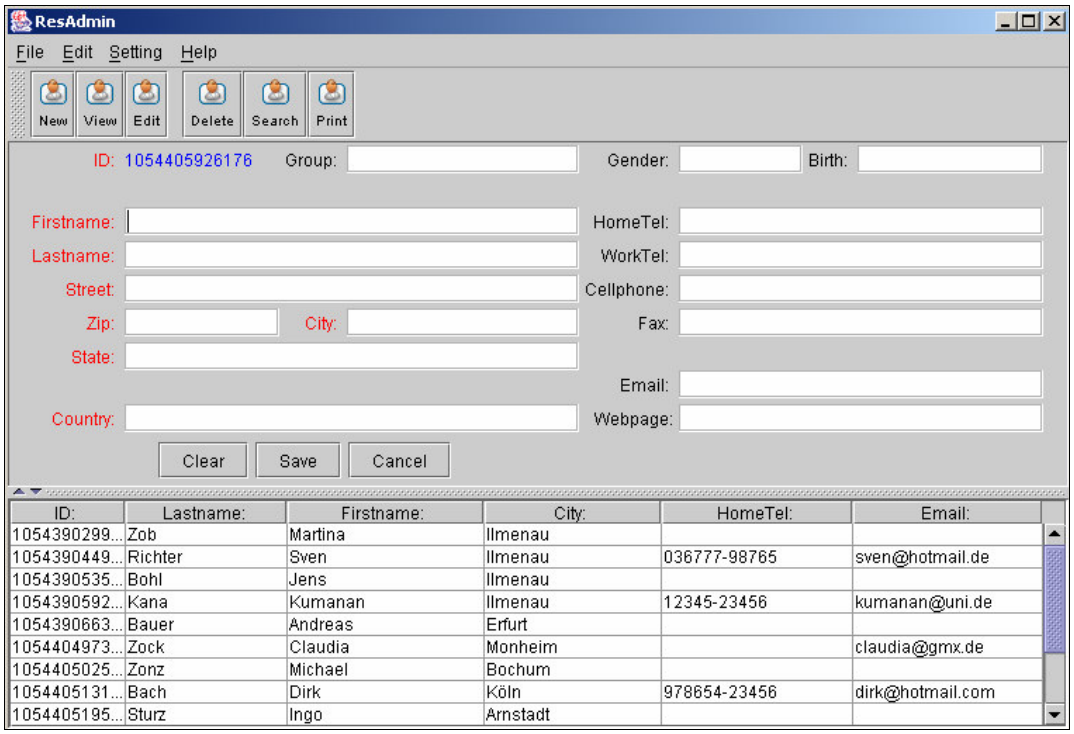


Abbildung 12: Datenstruktur anlegen

[22] Siehe Anhang SelectLanguage

Bei jeder neu erstellten Datenstruktur wird automatisch eine Identifikationsnummer auf Basis der aktuellen Zeit generiert. Diese Identifikationsnummer dient als Schlüssel für die Datenstruktur und wird als Dateiname für die Speicherung in die XML – Datei benutzt. Aus diesen Gründen ist die Veränderung des ID – Feldes nicht möglich. Die aus anderen Modulen von ResMedicinae erstellen XML – Dateien werden nach der Verarbeitung in ResAdmin mit ihren eigenen Schlüsseln gespeichert. Die Abbildung 12 gibt die Darstellung bei der Erstellung neuer Datenstrukturen wider.

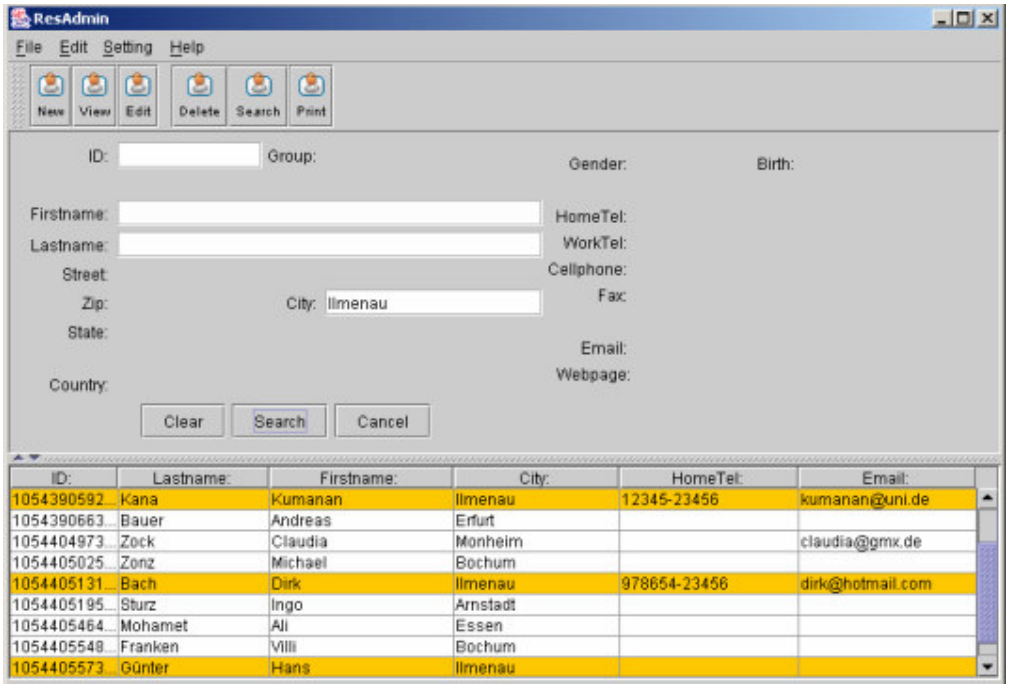


Abbildung 13: Datenstruktur suchen

Die Suchmöglichkeiten der Datenstruktur innerhalb der geladenen Daten der Tabelle beschränken sich auf ID, Vorname, Nachname und Ort. Die Abbildung 13 zeigt ein Beispiel für das Suchkriterium „Ilmenau“ als Ort. Falls mehrere Zeilen betroffen sind, werden sie wie auch in unserem Beispiel durch Markierungen hervorgehoben. Es ist auch möglich die Suchkriterien kombiniert einzusetzen.

In den verschiedenen ResMedicenae Modulen werden unterschiedliche XML – Tagsbeschriftungen zur Identifikation der jeweiligen Daten verwendet. Ein Beispiel hierfür ist die Bezeichnung „<City>“ oder „<Town>“ für das Tag Ort. Weiteres Beispiel ist die Root-Tag Bezeichnung „<Person>“ oder „<Patient>“. Aus diesen Gründen wurde die XML - Tags - Struktur in ResAdmin so implementiert, dass sie während der Laufzeit des Moduls je nach Wunsch verändern werden kann. Die Abbildung 10 zeigt die voreingestellte Struktur des XML - Tags.

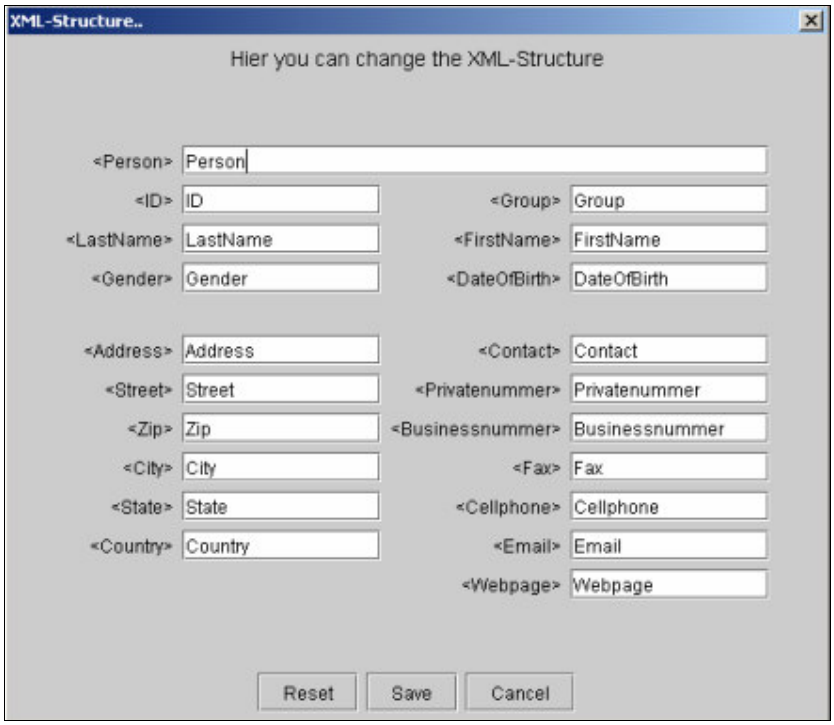


Abbildung 14: XML - Struktur

5.2 Internationalisierung

Die zweite Hauptaufgabe bestand in der Nutzung von ResAdmin in verschiedenen Sprachen. Die hier zur Auswahl stehenden Sprachen Englisch, Deutsch und Tamilisch können beim Programmstart statisch ausgewählt werden oder sie lassen sich während der Laufzeit des Programms dynamisch verändern. Bei der Umschaltung in die jeweilige Sprache werden nicht nur die Texte sondern auch die Bilder, in diesem Fall die Icons, verändert. Die Informationen über die dargestellten Texte, die Schriftart, die Schriftgröße und die Bilder sind in dem entsprechenden Sprach-ResourceBundles zusammengetragen. Diese Daten können unabhängig vom Quellcode mit einem Texteditor verändert werden. In der folgenden Abbildung 15 wird das Modul in der asiatischen Sprache Tamilisch dargestellt. Für die Darstellung und die Eingabe wurde die unicodebasierte Schriftart „Latha“ verwendet[23].

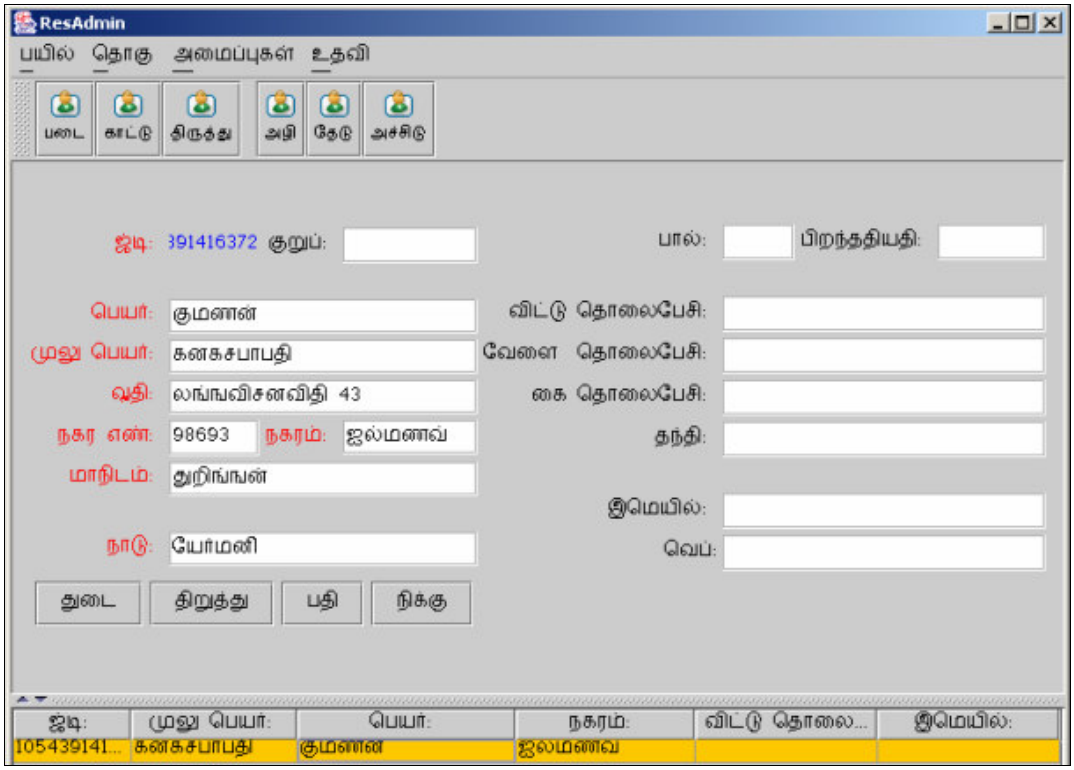


Abbildung 15: Modul in Tamilisch

[23] Im Anhang befinden sich die Screenshots für das Auswählen von Eingabemethoden unter SelectInputMethod, TamilInputMethod.

6. Zusammenfassung und Ausblick

Nach einer ausführlichen Erläuterung der Probleme der Internationalisierung von Software, widmete sich diese Arbeit im zweiten Kapitel den Realisierungsmöglichkeiten mit kleineren Beispielen. Der Schwerpunkt im praktischen Teil bildete neben der Verwaltung von Patientendaten die Internationalisierung des Moduls, welches im Rahmen des Res Medicinae Projektes durchgeführt wurde. Das Res Medicinae Projekt baut auf neueste Technologien auf und ist plattformunabhängig. Durch die Weiterentwicklung und die Freiverfügbarkeit des Quellcodes, ist es auch vorstellbar, dass dieses Produkt in der Zukunft auch in anderen Ländern zum Einsatz kommen kann.

Es wird heutzutage meistens vorausgesetzt, dass jegliche Produkte in der Landesprache erhältlich sind. Trotzdem werden von den Herstellern die globalen Aspekte der Internationalisierung vielfach vernachlässigt. Es werden Programmierer mit der Übersetzung beauftragt, die sich den oben aufgeführten Problemen meist nicht bewusst sind oder denen die Kenntnisse der Regionalaspekte fehlen. Wenn man diesen Problemen nach der Entwicklung aus dem Weg gehen will, sollte man vor der Implementierung die *locale-sensitiven Daten* identifizieren und von der eigentlichen Funktionalität getrennt ablegen. Da dieser Punkt bei den jetzigen Res Medicinae Modulen kaum Beachtung geschenkt wurde, empfiehlt sich diese Studienjahresarbeit als Grundlage (der Internationalisierung von Software) für neue Module.

Die Programmiersprache Java ist auf die Internationalisierung ausgerichtet und unterstützt zahlreiche Möglichkeiten, Programme an landes- und sprachspezifische Gegebenheiten anzupassen. Die Ausgabe eines Programms kann sprachneutral programmiert und dann mittels ResourceBundles an die Zielsprache adaptiert werden. Die Pakete *java.text* und *java.util* stellen darüber hinaus umfangreiche Funktionen für die Formatierungen bereit. Für das Suchen und Vergleichen von Strings in verschiedenen Sprachen bietet Java ebenfalls umfangreiche Mechanismen.

Quellenverzeichnis

[DeiCza2001] Andrew Deitsch and David Czarnecki
Java internationalization
Beijing: O'Reilly, 2001

[Fischer01] Paul Fischer
Grafik-Programmierung mit Java-Swing
München: Addison-Wesley, 2001

[Ullenboom02] Christian Ullenboom
Java ist auch eine Insel
Bonn : Galileo Press, 2002

[JavaTutorial] Java Sun Tutorial
Internationalization
http://java.sun.com/docs/books/tutorial/i18n/intro/index.html

[Krüger00] Krüger, Guido:
GoTo Java 2, Handbuch der Java Programmierung
Addison - Wesley, 2000.

[SchChina] Exotische Schriften lesen - leicht gemacht
Chinesisch lesen - leicht gemacht
http://www.schriften-lernen.de/Schrift/China/Ch18.htm

[SchIndien] Exotische Schriften lesen - leicht gemacht
Einführung in die indische Schrift
http://www.schriften-lernen.de/Schrift/Indien/Indien1.htm

[Farben] Farbanmutungen nach Nationen - kulturspezifische Farbbedeutungen
http://www.farbenundleben.de/kulturspezifische_Farbbeutungen.htm

[Siva97] Satguru Sivaya Subramuniyaswami
VedicCalendar
Hawaii: Himalayan Academy Kapaa, 1997
http://gurudeva.dynip.com/~htoday/pub/panchangam/pancha_intro.PDF

[TamZei] Dr. Kodumudi Shanmugam
Tamil Script and Students
Chennai (India): kodumudi.pdf

[ICU] IBM
International Components for Unicode
http://oss.software.ibm.com/icu4j/

[Unicode] Unicode
http://www.unicode.org/

[Res03] Res Medicinae:
The Res Medicinae Homepage
http://www.resmedicinae.org

[Kunze03] Kunze, Torsten:
Untersuchung zur Realisierbarkeit einer technologieneutralen Mapping - Schicht für den Datenaustausch am Beispiel einer Anwendung zum medizinischen Formulardruck als integrativer Bestandteil eines Electronic Health Record (EHR)
Diplomarbeit, TU Ilmenau, 2003.

[Bohl03] Bohl, Jens:
Möglichkeiten der Gestaltung flexibler Software - Architekturen für Präsentationsschichten, dargestellt anhand episodenbasierter, medizinischer Dokumentationen unter Einbeziehung topologischer Befundung
Diplomarbeit, TU Ilmenau, 2003.

Weiterführende Informationsquellen im Internet:

KDE Internationalization Home
http://i18n.kde.org/

OpenOffice Source Project
http://l10n.openoffice.org/

AncientScripts
http://www.ancientscripts.com

Indic Scripts
http://www.w3.org/2002/Talks/09-ri-indic/indic-paper.html

OpenI18N Initiative
http://www.openi18n.org/

IBM: Open Source
http://oss.software.ibm.com/developerworks/oss/

IBM: Globalize your e-business
http://www.ibm.com/software/globalization/index.html

Sun: Java i18n forum
http://forum.java.sun.com/forum.jsp?forum=16

Anhang

ISO 3166: http://www.unicode.org/onlinedat/countries.html

ISO 639: http://www.unicode.org/onlinedat/languages.html

Tamilzeichen Matrix:

ஃ	அ	ஆ	இ	ஈ	உ	ஊ	எ	ஏ	ஐ	ஒ	ஓ	ஔ
க்	க	கா	கி	கீ	கு	கூ	கெ	கே	கை	கொ	கோ	கெ(ள
ங்	ங	ஙா	ஙி	ஙீ	ஙு	ஙூ	ஙெ	ஙே	ஙை	ஙொ	ஙோ	ஙௌ
ச்	ச	சா	சி	சீ	சு	சூ	செ	சே	சை	சொ	சோ	சௌ
ஞ்	ஞ	ஞா	ஞி	ஞீ	ஞு	ஞூ	ஞெ	ஞே	ஞை	ஞொ	ஞோ	ஞௌ
ட்	ட	டா	டி	டீ	டு	டூ	டெ	டே	டை	டொ	டோ	டௌ
ண்	ண	ணா	ணி	ணீ	ணு	ணூ	ணெ	ணே	ணை	ணொ	ணோ	ணௌ
த்	த	தா	தி	தீ	து	தூ	தெ	தே	தை	தொ	தோ	தௌ
ந்	ந	நா	நி	நீ	நு	நூ	நெ	நே	நை	நொ	நோ	நௌ
ப்	ப	பா	பி	பீ	பு	பூ	பெ	பே	பை	பொ	போ	பௌ
ம்	ம	மா	மி	மீ	மு	மூ	மெ	மே	மை	மொ	மோ	மௌ
ய்	ய	யா	யி	யீ	யு	யூ	யெ	யே	யை	யொ	யோ	யௌ
ர்	ர	ரா	ரி	ரீ	ரு	ரூ	ரெ	ரே	ரை	ரொ	ரோ	ரௌ
ல்	ல	லா	லி	லீ	லு	லூ	லெ	லே	லை	லொ	லோ	லௌ
வ்	வ	வா	வி	வீ	வு	வூ	வெ	வே	வை	வொ	வோ	வௌ
ழ்	ழ	ழா	ழி	ழீ	ழு	ழூ	ழெ	ழே	ழை	ழொ	ழோ	ழௌ
ள்	ள	ளா	ளி	ளீ	ளு	ளூ	ளெ	ளே	ளை	ளொ	ளோ	ளௌ
ற்	ற	றா	றி	றீ	று	றூ	றெ	றே	றை	றொ	றோ	றௌ
ன்	ன	னா	னி	னீ	னு	னூ	னெ	னே	னை	னொ	னோ	னௌ

Quelle: [TamZei]

GPL-Compatible, Free Software Licenses:

http://www.gnu.org/philosophy/license-list.html#GPLCompatibleLicenses

ICU License - ICU 1.8.1 and later

COPYRIGHT AND PERMISSION NOTICE

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, provided that the above copyright notice(s) and this permission notice appear in all copies of the Software and that both the above copyright notice(s) and this permission notice appear in supporting documentation.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN NO EVENT SHALL THE COPYRIGHT HOLDER OR HOLDERS INCLUDED IN THIS NOTICE BE LIABLE FOR ANY CLAIM, OR ANY SPECIAL INDIRECT OR CONSEQUENTIAL DAMAGES, OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.

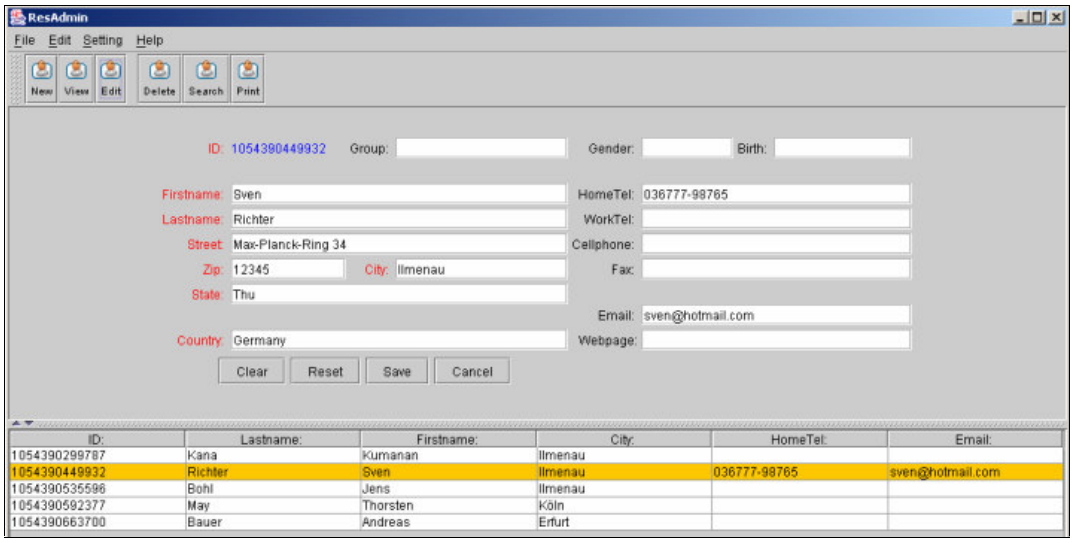
Except as contained in this notice, the name of a copyright holder shall not be used in advertising or otherwise to promote the sale, use or other dealings in this Software without prior written authorization of the copyright holder.

Quelle: http://oss.software.ibm.com/cvs/icu/~checkout~/icu/license.html

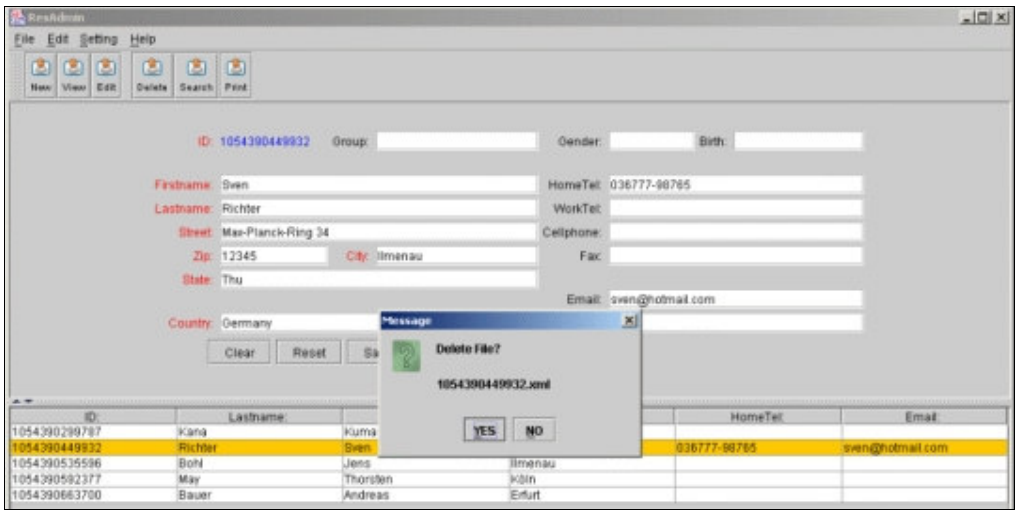
ResAdmin – ScreenShots

MainWindow

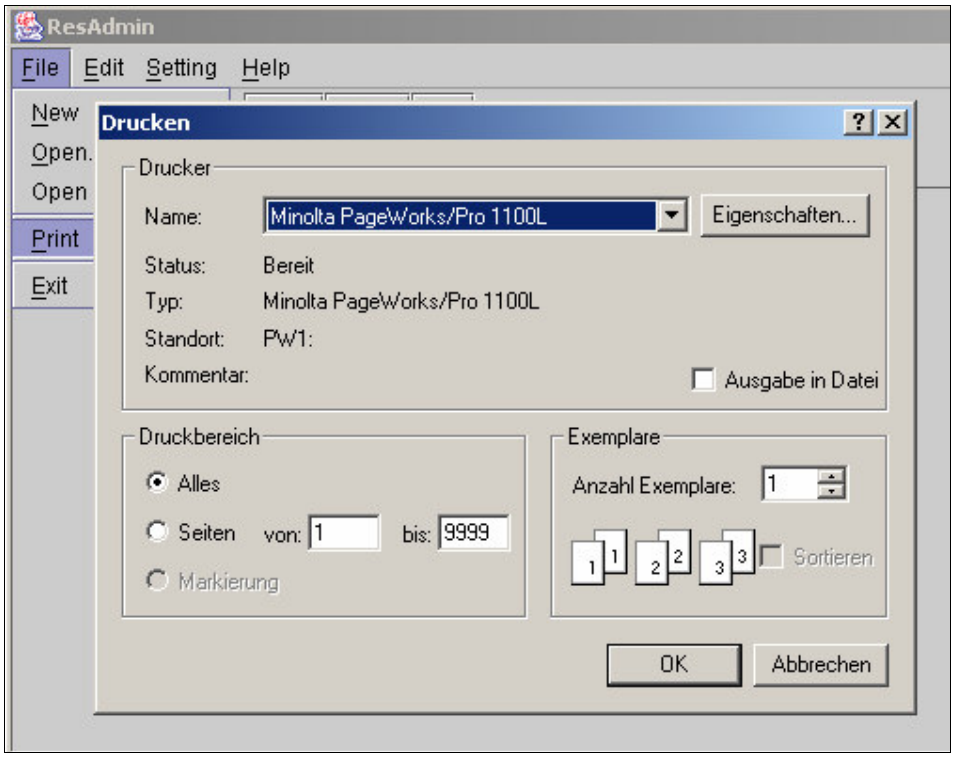
Edit



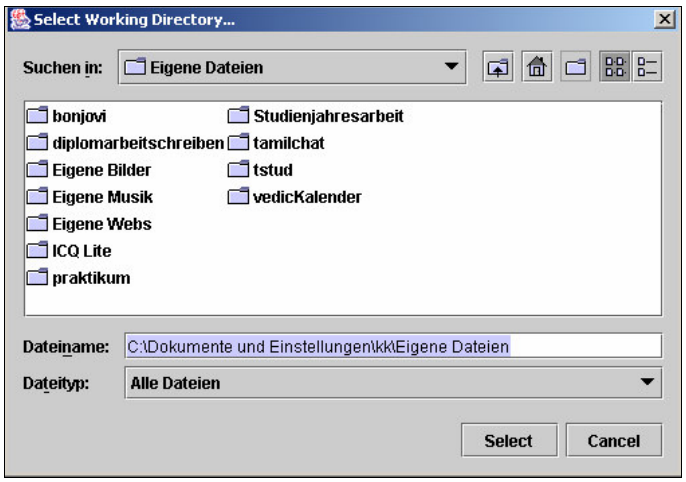
Delete



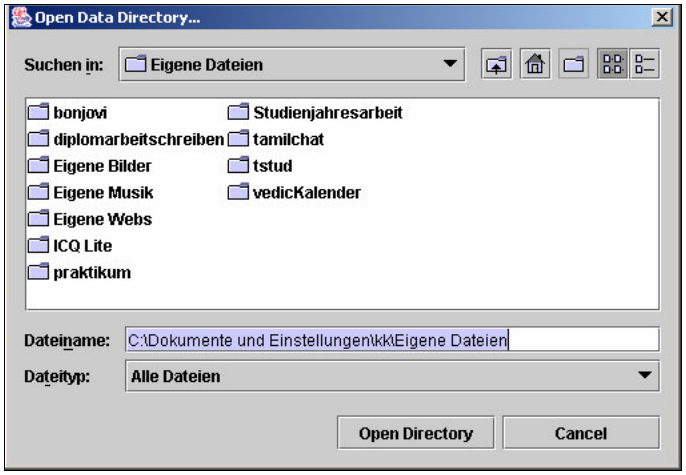
Print



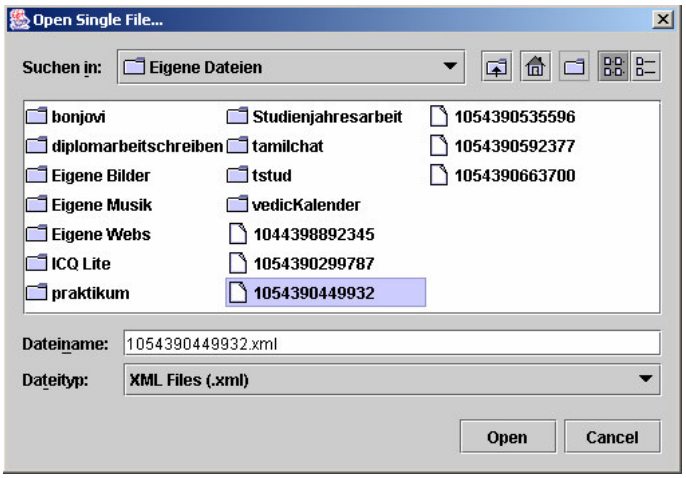
SelectWorkingDirectory



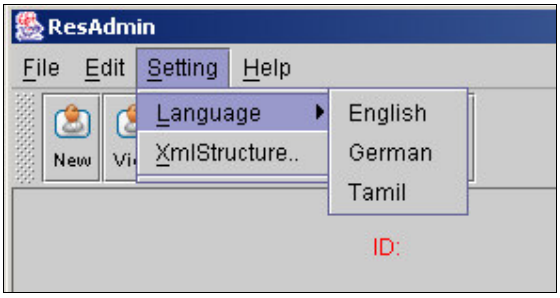
OpenDataDirectory



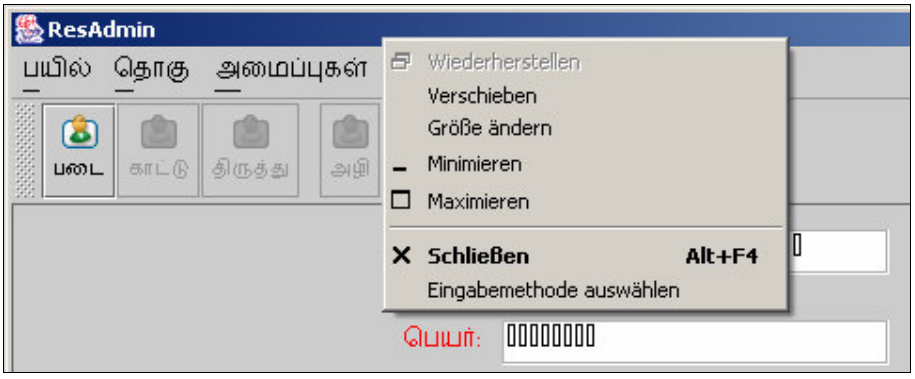
OpenSingleFile



SelectLanguage



SelectInputMethod



TamilInputMethod