FAKULTÄT FÜR !INFORMATIK

Internationalisierung von Webapplikationen

mit den Möglichkeiten aktueller Webentwicklungsframeworks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Diplomstudium Informatik

eingereicht von

Harald Bamberger

Matrikelnr.: 9726020

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung:
Betreuerin: Assoc. Prof.^{in} Dipl. Ing.^{in} Dr.^{in} Hilda Tellioğlu

Wien, 28. Oktober 2008 _____ _____

(Unterschrift Verfasser) (Unterschrift Betreuerin)

Abstract

The increasing number of internet users worldwide bears many opportunities for internationally acting companies and organisations. The single users are very different concerning their cultur or social background. So it is very important to attract users to stay with the company and use the product more often. This can be accomplished by having a user interface that uses the language and cultural values like currency or date format of the specific user. For most of the users this makes them feel more comfortable while using an application. It is the focus of internationalisation to provide methods and concepts, so that an application has not to be developed for every supported cultur - which would be a form of internationalisation but a bad one - but can be adapted for different cultures a more elegant and efficient way not even requiring to recompile it.

In the first part of this thesis, starting with an overview of the problems which are in the focus of internationalising an application, methods and concepts to deal with them are presented. In the second part a few up-to-date Webdevelopmentframeworks (Tapestry/JAVA, ASP.Net/C# and Zend Framework/PHP) are compared regarding their possibilities for internationalisation by implementing a simple representative web application.

Kurzfassung

Die steigende Zahl an InternetnutzerInnen weltweit bringt für international agierende Unternehmen bzw. Organisationen ein immer größeres Zielpublikum. Die einzelnen NutzerInnen kommen allerdings aus den unterschiedlichsten Kulturkreisen und sozialen Schichten. Um die BenutzerInnen bzw. KundInnen anzusprechen und eventuell zu binden, ist es notwendig Webauftritte oder Webapplikationen so zu gestalten, dass sich die BenutzerInnen wohl fühlen. Dies kann z.B. durch Verwendung der Sprache bzw. Schrift der BenutzerInnen, deren gewohntes Datumsformat oder Währung erreicht werden. Das Gebiet der Internationalisierung soll hierfür Methoden bereitstellen, um eine Anwendung nicht für jeden Kulturkreis extra zu entwickeln - was zwar auch eine Form der Internationalisierung wäre - sondern auf elegante und effiziente Weise nur jene Teile anzupassen, welche von Kultur zu Kultur verschieden sind, und dies tunlichst ohne einer neuerlichen Kompilierung der Applikation.

Im ersten Teil der Arbeit werden ausgehend von einer Übersicht der einzelnen Probleme, die im Fokus der Internationalisierung liegen, Methoden und Konzepte zu deren Lösung vorgestellt. Im zweiten Teil werden anhand einer einfachen repräsentativen Webapplikation die Möglichkeiten zur Internationalisierung, die aktuelle Webentwicklungsframeworks (Tapestry/JAVA, ASP.Net/C# und Zend Framework/PHP) bieten, evaluiert und verglichen.

Danksagung

An dieser Stelle möchte ich all jenen danken, die durch ihre Unterstützung, sei es in fachlicher oder persönlicher Hinsicht, zum Gelingen dieser Arbeit beigetragen haben.

Zunächst gilt mein Dank Frau Assoc. Prof.^{in} Dipl. Ing.^{in} Dr.^{in} Hilda Tellioğlu für die Betreuung der Arbeit und die zahlreichen Ratschläge, welche zur Verbesserung der Arbeit beigetragen haben.

Mein besonderer Dank gilt meinen Eltern, Ingrid und Anton Bamberger, die mir das Studium erst ermöglicht, und mich über die gesamte Dauer immer unterstützt haben.

Inhaltsverzeichnis

1 Einleitung

Die Anzahl der Internet nutzenden Menschen weltweit steigt stetig an. Die stärksten Zuwachsraten findet man außerhalb Europas, Japans und den USA. Demzufolge nimmt auch der Anteil an nicht Englisch sprechenden Personen bzw. derer die Englisch zumindest nicht als Muttersprache sprechen immer mehr zu. Auch ist es naheliegend, dass ein Großteil der BenutzerInnen es als angenehmer empfindet, Software in der jeweiligen Muttersprache zu verwenden (dies belegen auch Umfragen zu diesem Thema).

Weltweit oder international agierende Unternehmen und Konzerne sowie Organisationen oder auch reine Internetunternehmen haben somit zwar ein großes und weiter wachsendes potentielles Zielpublikum, welches aber sehr heterogen ist betreffend ihrer Sprache, Schrift und generellen Kultur. In unser schnelllebigen Zeit mit all ihrem Wettbewerb und übermäßigen Angebot an Konkurrenz in vielen Produktsparten ist es wichtig Kunden bzw. NutzerInnen zu binden also zu wiederholter Nutzung des angebotenen Produkts zu bewegen. Dies kann unter anderem dadurch erreicht werden, wenn eine an die kulturellen Bedürfnisse einzelner Zielgruppen angepasste Internetpräsenz bzw. Version der Webapplikation angeboten wird.

Der Großteil der Softwareentwicklung erfolgt in englischer Sprache – also auch Werkzeuge und Programmbibliotheken sowie Frameworks zur Programmierung bzw. Webprogrammierung werden meistens zunächst in Englisch erstellt. Mit der Aufgabe, Software so zu entwickeln, dass sie mit möglichst geringem Aufwand in andere Sprachen übertragbar oder allgemeiner für andere Kulturen adaptierbar ist, beschäftigt sich das Themenfeld der Internationalisierung.

Zunächst sind einige grundlegende Begriffe genauer zu bestimmen und die möglichen Organisationsmodelle der Internationalisierung festzulegen. Dies hat Abschnitt 2.1 *Grundlagen* zum Inhalt. Danach ist zu klären, welche Bereiche einer Applikation von kulturellen und regionalen Unterschieden betroffen sind.

Der augenscheinlichste Bereich, der im Fokus der Internationalisierung liegt, ist sicher die Sprache und in weiterer Folge deren Darstellung, die Schrift. Eine internationalisierte Applikation muss also in der Lage sein, sowohl Ausgaben in der jeweiligen Sprache bzw. Schrift der BenutzerInnen darzustellen, als auch Eingaben in dieser zu verarbeiten. Dabei stellen nicht nur die vielen verschiedenen Schriften eine Herausforderung der Internationalisierung dar, sondern auch Unterschiede hinsichtlich der Reihenfolge bei Sortierungs- oder Suchvorgängen bzw. das Auffinden von Buchstaben-, Wort- oder Satzgrenzen. Die Annahme, dass in jedem Schriftsystem Wörter durch Leerzeichen getrennt sind oder aber

die Sortierreihenfolge gleich ist, weil in unterschiedlichen Ländern dasselbe Schriftsystem verwendet wird, stellt sich schnell als Irrtum heraus. Auch ist es keinesfalls so, dass ein Buchstabe immer einem Byte entspricht. Daraus resultieren natürlich einige Probleme bei der elektronischen Verarbeitung. Diese werden im Abschnitt 2.2 *Schrift* ausführlich behandelt.

Hinsichtlich der verwendeten Formate, wie etwa Zahlen oder Datumsangaben dargestellt werden, gibt es viele Varianten je nach lokalen Gewohnheiten. Die Datumsangabe 11/04/2008 würde ein Amerikaner als 04. November 2008, ein Brite dagegen als 11. April 2008 interpretieren, wobei immerhin in beiden Ländern zumindestens der / als Trennzeichen gebräuchlich ist und dasselbe Kalenderformat nämlich der Gregorianische Kalender verwendet wird. Man sieht also, schon bei so etwas "banalen" wie einer Datumsangabe gibt es eine Vielzahl an regionalen und kulturellen Unterschieden zu berücksichtigen. Dazu kommen noch diverse unterschiedliche Zahlendarstellungen mit dazugehörigen Trennsymbolen, Maßeinheiten, Währungen etc. Der Abschnitt 2.3 *Formate, Datumsangaben, Einheiten, Systemnachrichten* etc. beschäftigten sich mit diesem Aspekt der Internationalisierung.

Ein weiterer Punkt betrifft die Verwendung von Farben und Symbolen, da deren Bedeutung kulturell sehr unterschiedlich ist. Im westeuropäischen Kulturkreis ist schwarz etwa die Farbe des Todes im asiatischen ist dies weiß. Auch beim Umgang mit Symbolen gilt es Unterschiede zu berücksichtigen, das bei uns weit verbreitete ? als Symbol für die Hilfefunktion in einem Programm, wird nicht überall so verstanden, da nicht in allen Schriftsystemen Fragen mit ? gekennzeichnet werden. Auch kann es nötig bzw. von Vorteil sein, das Layout oder auch den Funktionsumfang an kulturelle Gegebenheiten anzupassen. In [SinPer05] werden Kulturen bzw. Länder nach mehreren Gesichtspunkten klassifiziert und daraus resultierend Empfehlungen erarbeitet, welche Charakteristiken Webpräsenzen je nach Zielkultur aufweisen sollten. Mit dieser Problematik beschäftigen sich der Abschnitt 2.4 *Farbe, Layout, Kultur.*

Der Abschnitt 2.5 *Stufen und Konzepte der I18n* beschäftigt sich mit den Ausbaustufen der Internationalisierung und welche Eigenschaften Anwendungen in der jeweiligen Ausbaustufe aufweisen. Auf Auswirkungen der einzelnen Ausbaustufen auf den Entwicklungsprozess wird ebenfalls kurz eingegangen. Weiters werden Konzepte beschrieben, die zur Bewältigung der Aufgabenstellungen bei der Internationalisierung von Anwendungen bzw. Webanwendungen Verwendung finden.

Im letzten Teil des theoretischen Teils werden Voraussetzungen an die technische Um-

gebung, in der die Webapplikation abläuft, und einige exemplarische Anwendungsfälle behandelt. Ein Überblick über das Thema Übersetzung und Testen von internationalisierten Webapplikationen bildet den Abschluss des Theorieteiles.

Auf Basis der Erkenntnisse des Theorieteils wird im praktischen Teil der Arbeit ein Kriterienkatalog erstellt, der es ermöglichen soll, Webentwicklungsframeworks bezüglich ihrer Internationalisierungsfähigkeiten zu vergleichen. Diese Kriterien gliedern sich in allgemeine Informationen über das jeweilige Framework, Eigenschaften beim Umgang mit Unicode kodiertem Text/Sprache, Formatierungen von Datums- und Zeitangaben und Zahlen, das Externalisieren von Systemnachrichten und die Internationalisierung von Benuterschnittstelle und Layout.

Um die erstellten Kriterien anwenden zu können, werden mehrere Beispielaufgaben festgelegt, welche mit drei aktuellen Frameworks (Tapestry/Java, ASP.Net/C# und Zend Framework/PHP) umgesetzt werden. Die praktischen Erkenntnisse bei der Implementierung werden dann in der Analyse bzw. dem Vergleich dieser Frameworks zusammengefasst. Zusätzlich wird eine tabellarische Übersicht der Ergebnisse nach den festgelegten Kriterien ausgeführt. Zusammenfassung und Auslick bilden den Abschluss dieser Arbeit.

2 Begriffliche und technische Grundlagen der Internationalisierung

In diesem Abschnitt der Arbeit werden, beginnend mit der Bestimmung einiger grundlegender Begriffe, die Problembereiche im Fokus der Internationalisierung betrachtet. Ziel ist es einen umfassenden Überblick über diese, sei es nun der Umgang mit verschiedenen Schriftsystemen oder die unterschiedliche Formatierung von Datumsangaben oder Währungen in den verschiedenen Kulturen und Regionen, zu erhalten. Weiters werden zu jedem Problembereich gängige Techniken beschrieben, um die auftretenden Anforderungen handhaben zu können. Daran anschließend werden aus diesen Lösungsansätzen grundlegende Konzepte abgeleitet, welche bei der Internationalisierung zum Einsatz kommen. Weiters wird eine Klassifizierung eingeführt, die zur Unterscheidung der Ausbaustufen von Webapplikationen hinsichtlich der Internationalisierung dient. Abschließend wird auf die Voraussetzungen für die Internationalisierung von Webapplikationen seitens der Infrastruktur und auch deren Auswirkungen auf die Entwicklung einer solchen Webapplikation eingegangen.

2.1 Grundlagen

Beschäftigt man sich etwas mit dem Themenkreis der Anpassung von Software an die Bedürfnisse unterschiedlicher Nationalitäten oder Kulturen, stößt man schnell auf Begriffe wie Globalisierung, Internationalisierung, Lokalisierung oder Locale. Diese Begriffe werden nachfolgend definiert und erläutert. Danach folgt eine Auflistung möglicher Entwicklungsansätze bzw. eine historische Reihung von Ausformungen der Internationalisierung von Anwendungen.

2.1.1 Begriffe

Locale: Ein Locale bestimmt ein Gebietschema und in weiterer Folge dazugehörige Parameter. Bei Linux-Betriebsystemen besteht der Bezeichner für ein Locale aus einem Sprachcode und einem Ländercode (z.B. de_AT für Deutsch/Österreich oder en_GB für Englisch/Großbritannien). Die Bezeichnung in der Programmiersprache JAVA folgt dem selben Schema. Mit einem Locale assoziiert sind für gewöhnlich Vorgaben für die verwendeten Datums- und Zahlenformate aber auch Währung und Sortierung. Das Locale ist ein wichtiges Konzept der Internationalisierung.

G11n - Globalization[1]: Die Localization Industry Standards Association (LISA) definiert auf ihrer Webseite Globalization wie folgt:

> „The process of making all the necessary technical, financial, managerial, personnel, marketing and other enterprise decisions necessary to facilitate international business.
>
> (Source: LISA Glossary)“ [LISA08]

Globalization stellt demnach einen Überbegriff dar, welcher alle nötigen Handlungen, um ein Softwareprodukt für internationales Zielpublikum zu entwickeln, abdeckt. Darunter fallen unter anderem auch die technischen Lösungsansätze, die hier behandelt werden sollen.

I18n - Internationalization[2]: LISA definiert auf ihrer Webseite Internationalization wie folgt:

> „1. The process of enabling a product at a technical level for localization.
>
> 2. The process of designing an application so that the feature design and code design do not make assumptions that are based on a single language or locale. Internationalization simplifies the creation of different language editions of a program. One goal of internationalization is to ensure that international conventions (including rules for sorting strings and for formatting dates, times, numbers, and currencies) are supported. Another goal is to design the product in such a way that users will experience consistent appearance and functionality across different language editions of a product. Internationalization is often abbreviated as I18N. The abbreviation is formed using the first and last letters of the word (I, N) and the number 18, which specifies the number of letters between the I and the N.
>
> 3. The process of ensuring at a technical/design level that a product can be easily localized.
>
> (Source: LISA Glossary)“ [LISA08]

Internationalization beschäftigt sich mit der Problematik, Software technisch so zu entwickeln, dass eine Localization für eine andere Zielgruppe leicht bzw. leichter möglich ist.

[1] G11n: Globalization {lobalizatio}=11 Buchstaben
[2] I18n: Internationalization {nternationalizatio}=18 Buchstaben

Dies deckt sowohl das konsequente entwickeln nach Gesichtspunkten der Internationalisierung einer Applikation, aber auch das Re-Engineering einer nicht internationalisierten Anwendung ab.

In [Kub06] wird ein I18N Internationalization Model so definiert, dass mehrere Sprachen unterstützt werden aber nur 2 gleichzeitig - nämlich Englisch also ASCII Code und eine weitere, welche spezifiziert werden muss (z.B. über Umgebungsvariablen).

L10n - Localization[3]: LISA definiert auf ihrer Webseite Localization wie folgt:

> „1. The process of modifying products or services to account for differences in distinct markets.
>
> 2. the process of adapting software for a particular geographical region (locale). Translation of the user interface, system messages, and documentation is a large part (but not all) of the localization process. Localization is often abbreviated as L10N. This abbreviation is formed using the first and last letters of the word (L, N) and the number 10, which specifies the number of letters between the L and the N.
>
> 3. The process of modifying products or services to account for differences in distinct markets.
>
> (Source: LISA Glossary)“ [LISA08]

Localization ist demnach das Adaptieren eines bereits internationalisierten Softwareproduktes an eine bestimmte Sprache/Region.

In [Kub06] wird L10N als Localization Model beschrieben, bei dem zum einen Englisch in Form des ASCII Codes und eine bestimmte andere Sprache unterstützt wird. In diesem Fall erfolgt nur eine Adaptierung an eine bestimmte Sprache und keine generelle Anpassung, um das Adaptieren in andere Sprachen zu erleichtern oder zu ermöglichen.

M17n - Multilingualization[4]: Unter Multilingualization versteht man die Unterstützung mehrerer Sprachen gleichzeitig. Dies bezieht sich allerdings auf die Möglichkeit mehrere verschiedene Sprachen einzugeben und zu verarbeiten, nicht aber auf die gleichzeitige mehrsprachige Ausgabe von Fehlermeldungen oder Beschriftungen des User Interfaces. In [Kub06] wird die Multilingualization als ein mögliches Entwicklungsmodell

[3] L10n: Lokalization {okalizatio}=10 Buchstaben
[4] M17n: Multilingualization {ultilingualizatio}=17 Buchstaben

angeführt und als Beispiel ein Texteditor genannt, welcher dazu verwendet werden kann, ein Dokument über die Unterschiede von Koreanisch und Chinesisch in Finnisch zu erstellen.

2.1.2 Entwicklungsansätze

In [Kub06] werden zwei Entwicklungsansätze vorgestellt und deren Vor- und Nachteile kurz erläutert.

Zum einen Implementation ohne Kenntnisse jeder der zu unterstützenden Sprachen. Bei diesem Ansatz müssen demnach soweit als möglich standardisierte Methoden verwendet werden. Als Beispiele solcher Methoden sind etwa die Locale Technologie oder die GNU gettext Bibliothek, welche für einige Programmiersprachen verfügbar ist, zu nennen. Die Vorteile dieses Ansatzes sind:

- Bei einem Update der verwendeten Bibliothek unterstützt auch das Programm neu hinzugefügte Sprachen.
- Die ProgrammiererInnen müssen nicht alle Zielsprachen beherrschen.
- Die BenutzerInnen können über standardisierte Methoden das Verhalten der Software beeinflussen.

Der Nachteil dieses Ansatzes ist, dass nicht für alle Probleme standardisierte Methoden vorhanden sind, z.B. Silbentrennung.

Beim zweiten möglichen Ansatz besitzt der ProgrammiererInnen Kenntnisse der zu unterstützenden Sprachen und kann deshalb Informationen über diese Sprachen direkt implementieren. Das L10n Modell nach [Kub06] wird fast immer nach diesem Ansatz implementiert - im Gegensatz zum I18n Modell, welches dem erstgenannten Ansatz folgt. Der Vorteil dieses Ansatzes liegt darin, dass Probleme, die sich bei manchen Sprachen ergeben, von standardisierten Methoden aber nicht oder nicht in ausreichendem Maße behandelt werden, explizit gelöst werden können. Die Nachteile sind:

- Die Anzahl der unterstützten Sprachen hängt vom Können der ProgrammiererInnen ab.
- Probleme, die bei standardisierten Methoden quasi zentral gelöst werden, müssen bei diesem Ansatz explizit erfolgen.

- Die BenutzerInnen sind bei verschiedenen Programmen mit unterschiedlichen Konfigurationsvarianten konfrontiert.

In [HeBus02] werden 7 Ansätze zur Internationalisierung bzw. Lokalisierung beschrieben, die je nach der Phase ihres Auftretens (design time, compile time, link time, run time) klassifiziert und historisch geordnet werden.

1. *Run-time Localisation*: In diesem Fall steht nur das ausführbare Programm zur Verfügung und die ProgrammiererInnen müssen zu lokalisierende Elemente finden und ersetzen.

2. *Compile-time Localisation*: Hier steht der Source Code zur Verfügung und wird nach den Vorgaben lokalisiert und neu kompiliert.

3. *Compile-time Internationalisation and Compile-time Localisation*: Bei diesem Ansatz werden die zu lokalisierenden Elemente des Programms für alle geforderten Locales in den Source Code aufgenommen, um dann für verschiedene Locales adaptierte Versionen zu kompilieren und zu linken.

4. *Compile-time Internationalisation and Link-time Localisation*: Das Prinzip hinter diesem Ansatz ist das Aufteilen in Module, die vom jeweiligen Locale abhängig sind, und solche, die es nicht sind. Die jeweils adaptierten Locale abhängigen Module können dann mit den nicht locale-abhängigen Modulen verlinkt werden, um an verschiedenen Locales angepasste Versionen zu generieren.

5. *Compile-time Internationalisation and Run-time Localisation*: Hier wird die Anwendung so internationalisiert, dass die zu lokalisierenden Elemente ohne neuerliches Kompilieren oder Linken verwendet werden können.

6. *Design-time Internationalisation and Link-time Localisation* bzw.

7. *Design-time Internationalisation and Run-time Localisation*: Bei diesen beiden Ansätzen wird nicht mit bestehenden Anwendungen bzw. Source Code gearbeitet, sondern die Internationalisierung und Lokalisierung sind fixer Bestandteil des Entwicklungsprozesses. Sie werden also, wie der Name vermuten lässt, schon während der Design Phase berücksichtigt.

Man sieht also, dass die Bestrebungen lokalisierte Anwendungen zu erzeugen mit dem mühsamen Ersetzen von Strings im Binärcode mittels Hex-Editor begonnen haben (Run-time Localisation) und sich kontinuierlich zu einer Internationalisierung des Source Codes und Einbinden separater lokalisierter Resourcen (Compile-time Internationalisation and Run-time Localisation) weiter entwickelt haben. Da die Internationalisierung und in weiterer Folge die Lokalisierung für verschiedene Region einen nicht zu unterschätzenden Aufwand bedeuten sollte sie möglichst früh, also bereits in der Design Phase, in den Entwicklungsprozess eingebunden werden.

2.1.3 Webapplikation

Eine Webapplikation ist ein Programm, welches auf einem Webserver abläuft und als Benutzerschnittstelle einen Web-Browser als Client verwendet. Die Steuerung erfolgt dabei durch Anfragen, die von Client gestellt werden (HTTP-Request) und vom Server beantwortet werden (HTTP-Response). Zur Kommunikation zwischen Webserver und dahinter liegender Webapplikation wurde lange Zeit das Common Interface Protokoll (CGI) verwendet. Da die CGI Schnittstelle relativ langsam ist, werden heute meist andere Lösungen bevorzugt. So zum Beispiel existieren für den Apache Webserver Module für PHP oder Perl, welche direkt zur Interpretation des Codes dienen. Für Java Servlets bzw. ServerPages kann zum Beispiel Apache Tomcat verwendet werden, welcher eine Referenzimplementierung der Servlet Spezifikation von Sun Microsystems darstellt. Da Tomcat einen HTTP Server mitbringt, kann er für Entwicklungszwecke stand-alone verwendet werden - in Produktionsumgebungen wird zumeist ein Apache Server mit geeignetem Modul zur Kommunikation mit Tomcat vorgeschaltet.

Das im ersten Absatz beschriebene Szenario war in den Anfängen des World Wide Web bestenfalls Zukunftsmusik, wenn nicht undenkbar. Als Tim Berners-Lee im Jahre 1990 den ersten HTTP-Server in Betrieb nahm, war sein Traum einen gemeinsamen Informationsraum zu schaffen, um eben diese Informationen austauschen zu können [Ber98]. Diese Konzeption als Informationsmedium sah statische Dokument vor, welche durch Hyperlinks miteinander verknüpft waren. Anfang der Neunziger Jahre wurden zuerst Hochschulen und dann die Industrie auf das Konzept aufmerksam und es begann eine immer schneller werdende Verbreitung, welche, begünstigt durch die Integration der notwendigen Technologien (TCP/IP und Internet Explorer) in das Mainstream Betriebssystem Windows95, auch den privaten Bereich erfasste.

In weiterer Folge fand eine Entwicklung vom Informationsmedium zum Anwendungs-

medium statt. Webanwendungen sind weit verbreitet und stellen vollwertige und komplexe Softwaresysteme dar [Kapp04]. Da Webanwendungen in vielen Bereichen Einzug halten, bilden sich auch unterschiedliche Kategorien von Webanwendungen. In [Kapp04] wird folgende Unterscheidung von Webanwendungen getroffen (siehe Tabelle 1). Die Kategorien sind historisch aufsteigend sortiert. Auch ist zu beachten, dass die Komplexität der Anwendungstypen kontinuierlich ansteigt.

Kategorie	**Beispiele**
Dokumentenzentriert	Statische Homepage Web-Radio Firmenpräsentation
Interaktiv	Virtuelle Ausstellung News-Site Fahrplanauskunft
Transaktional	Onlinebanking E-Shopping Reservierungssystem
Workflowbasiert	E-Government B2B-Lösung Patienten-Workflow
Kollaborativ	Chatroom E-Learning Plattform Virtueller gemeinsamer Arbeitsraum
Portalorientiert	Community-Portal Online Shopping Mall Unternehmensportal
Ubiquitär	Personalisierte Dienste Ortsabhängige Dienste Multi-Platform Delivery
Semantisches Web	Recommender System Syndication Wissensmanagement

Tabelle 1: Kategorien von Webanwendungen nach [Kapp04]

Dokumentzentrierte Webseiten finden sich in dieser Auflistung eher aus historischen Gründen, da sie keine Webanwendungen im eigentlichen Sinn darstellen. Charakteristisch für diese Art ist die manuelle Aktualisierung der Inhalte mit geeigneten Werkzeugen und Editoren, welche aber auch meist zu veralteten Informationen führt, aber auch die Einfachheit, Stabilität und geringe Antwortzeiten aufgrund der statischen Dokumente.

Interaktive Webanwendungen verwenden grundlegende Interaktionstechniken wie die dynamische Generierung von Inhalten und Links über das oben bereits erwähnte Common Gateway Interface (CGI) und HTML-Formulare für Benutzereingaben, zur Interaktion mit den AnwenderInnen.

Transaktionale Webanwendungen bieten in der Regel durch Einsatz eines geeigneten Datenbanksystems weiterführende Interaktionsmöglichkeiten. Mit solchen Systemen können Abläufe wie Überweisungen bei Onlinebanking abgewickelt werden.

Zentraler Aspekt bei *Workflowbasierten Webanwendungen* ist die Abwicklung von Geschäftsprozessen. Die BenutzerInnen können dabei sowohl innerbetrieblich als auch betriebsübergreifend oder auch aus dem Behördlichen oder privaten Bereich stammen. Die Herausforderung hierbei ist unter Gewährleistung der Autonomie der einzelnen Teilnehmer eine Integration der jeweiligen Dienste zu erreichen, um eine Zusammenarbeit zu ermöglichen. Solche Anwendungen finden sich im Business-to-Business (B2B) Bereich zur Abwicklung von E-Commerce oder auch im Bereich des E-Government.

Das Einsatzgebiet von *Kollaborativen Webanwendungen* ist ähnlich dem von Workflowbasierten Anwendungen, allerdings geht es hierbei um die Abwicklung unstrukturierter Abläufe, wie sie bei Groupware oder E-Learning Plattformen auftreten.

Portalorientierte Webanwendungen stellen einen Zugangspunkt zum Web dar, über welchen sehr unterschiedliche Informationsquellen und Dienste zentral erreicht werden können. Solche Portale können sehr allgemein gehalten sein und ein großes Spektrum verschiedenster Themenbereiche abdecken, wie z.B. das Portal des Suchmaschinenanbieters Yahoo, oder aber sehr spezifisch für eine Branche. Eine weiter möglich Variante wären Intranet- oder Extranetportale, um Informationen für die MitarbeiterInnen eines Unternehmens oder für Partnerunternehmen bereit zu stellen.

Ubiquitäre Webanwendungen stellen personalisierte oder ortsabhängige Dienste für verschiedenste Endgeräte dar. Aufgrund der immer zahlreicher werdenden vor allem mobilen Geräte, welche in der Lage sind, eine Internetverbindung herzustellen, wird diese Kategorie von Anwendungen in Zukunft immer größere Bedeutung gewinnen. Die Voraussetzung zur Nutzung solcher Anwendungen sind allerdings Informationen, in welchem Kontext die Anwendung gerade benutzt wird. Ein ortsabhängiger Dienst, welcher etwa die Apotheken in der Nähe der BenutzerInnen anzeigen soll, muss relativ genaue Informationen über deren Standort und eventuell über deren Möglichkeiten, die Apotheke zu erreichen, zur Verfügung haben, um hilfreiche Informationen liefern zu können.

Ziel des *Semantischen Webs* ist es, die Bedeutung von Informationen so aufzubereiten,

dass sie maschinell verarbeitbar wird. Daraus würden sich Möglichkeiten zur Vernetzung und Wiederverwendung von Wissen ergeben. Man wird sehen, was dieses Gebiet in der Zukunft an Anwendungen hervorbringt.

Aufgrund der steigenden Komplexität von Webanwendungen ist ein entscheidender Aspekt bei deren Entwicklung die zugrunde liegende Architektur. Diese wird von verschiedenen Faktoren beeinflusst. Einerseits sind dies natürlich Anforderungen der AuftraggeberInnen und der AnwenderInnen an das Endprodukt aber auch Vorgaben die Performance, Skalierbarkeit usw. betreffend. Andererseits spielen Erfahrungen mit bestehenden Architekturen und Architekturmustern und technischen Randbedingungen, die sich durch die Verwendung bestimmter Middleware oder der Integration von Legacy Systemen ergeben eine Rolle [Kapp04].

Durch die Funktionsweise des Web - also dem Beantworten einer Anfrage (Request) durch den Client mit einer Antwort (Response) des Servers - kann die Architektur in Schichten betrachtet werden [Kapp04]. Die 2-Schichten Architektur definiert dabei eine Client- und eine Serverschicht. Die Serverschicht kann dabei statische HTML-Seiten oder den Zugriff über die Anwendungslogik des Webservers auf Datenbankinhalte und die anschließende Generierung der HTML-Seiten bestehen. Für einfach Webanwendungen ist so eine Architektur ausreichend, allerdings empfiehlt sich für komplexere Anwendungen eine mehrschichtige Architektur.

Bei der N-Schichten Architekturen können beliebig viele Schichten eingeführt werden [Kapp04]. Dies betrifft meist den Einsatz eines Applikationsserver, welcher Dienste zur Verfügung stellt und Datenbanken und Business-to-Business Systeme in das System integriert und durch Verteilungs- und Lastenausgleichsmechanismen zur Verbesserung der Performance der Webanwendung beiträgt.

Aspekte der Internationalisierung können bei jedem Typ von Webapplikation zur Anwendung kommen und müssen auf allen Schichten berücksichtigt werden. Das Ausmaß der zu berücksichtigenden Aspekte ist dabei von Schicht zu Schicht unterschiedlich und weist eine Konzentration auf Ebene der Präsentationslogik auf, wo eben die unterschiedlichen Erwartungen der BenutzerInnen an regionale und kulturelle Konventionen erfüllt werden sollten. Die darunterliegenden Schichten müssen dies allerdings vor allem im Hinblick auf die Verarbeitung von verschiedensprachigen Daten unterstützen.

2.1.4 (Webapplication) Framework

Ein Framework bildet ein Programmiergerüst für eine bestimmte Art von Anwendung [GaHeJoVl95]. Das Framework stellt dabei eine Ansammlung an Klassen (konkret und abstrakt) zur Verfügung und gibt in der Regel eine Applikationsstruktur vor. Frameworks werden aufgrund der immer komplexer werdenden Applikationen häufig eingesetzt. Auch die Kombination verschiedenster Frameworks ist keine Seltenheit z.B. die Integration eines Frameworks zur Anbindung einer Datenbank mit einem zur Erstellung der GUI der Applikation. Einerseits hat die Verwendung von Frameworks den Vorteil, dass gewisse Grundfunktionalitäten wiederverwendet werden können und erprobte Muster in die Anwendung übernommen werden. Als Nachteil ist ein hoher Einarbeitungsaufwand und eine gewissen Abhängigkeit vom Hersteller des Frameworks zu nennen [Kapp04].

Webapplication Frameworks stellen bestimmte Funktionen zur Verfügung, die bei der Entwicklung von Webapplikationen benötigt werden. Diese umfassen meist Möglichkeiten zur Validierung von Benutzereingaben, die Anbindung von Datenbanken und das Verwalten von BenutzerInnen und deren Sitzungsdaten. Die meisten Webapplication Frameworks folgen dem Model-View-Controller Muster und erzwingen daher die Trennung von Präsentationslogik, Ablaufsteuerung und Anwendungslogik. Das Model repräsentiert dabei Daten und Operationen auf diesen Daten. Der Zustand des Models kann über den View dargestellt werden. Der Controller stellt die Verbindung zwischen Model und View dar, indem er z.B. aufgrund von Benutzereingaben das Model dementsprechend aktualisiert [Shi04] [Moeh08].

Da die AJAX & Web 2.0 Technologien zunehmend Verbreitung finden und damit Benutzerschnittstellen in Webapplikationen Features wie Drag-and-Drop, welche bislang nur in Desktopapplikationen möglich waren, und das dynamische Nachladen von Daten Änderungen am Seiteninhalt ohne eines neuerlichen expliziten Request durch die BenutzerInnen, ermöglichen, bieten viele Frameworks auch Module zu deren Einbindung bzw. Verwendung.

Viele Webapplication Frameworks bringen auch Fähigkeiten zur Behandlung von I18n-aufgaben mit. Der Umfang dieser Fähigkeiten hängt dabei zum einen davon ab, wie stark I18N Ansätze in der zugrundeliegenden Programmiersprache implementiert sind, aber auch ob und in welchen Ausmaß diese beim Design und der Implementierung des Frameworks berücksichtigt wurden. Ein Vergleich zwischen drei auf unterschiedlichen Programmiersprachen basierenden Frameworks wird im praktischen Teil dieser Arbeit durchgeführt.

2.1.5 Web-Services

Abschließend soll noch ein Thema angesprochen werden, das in den letzten Jahren immer mehr an Bedeutung gewonnen hat - Web-Services. Diese sind an sich nach der oben eingeführten Definition zwar keine Webapplikationen, da sie nicht Web-Browser als Clients verwenden. Sie können allerdings von Webapplikationen zur Bereitstellung von Funktionen verwendet werden.

In [Dus03] werden Web-Services als im Internet nutzbar gemachte Softwarekomponenten, die eine Softwareoperation zur Verfügung stellen, auf die über eine Schnittstelle mittels Nachrichten-basiertem XML Datentransfer zugegriffen werden kann, beschrieben. Webservices sind ein Beispiel für eine service-orientierte Architektur (SOA). In dieser existieren verschiedene Rollen. Der Service-Provider stellt ein Web-Service zur Verfügung und veröffentlicht dieses über einen Service-Broker, über welchen ein Service Requestor ein Web-Service finden kann und dann an den Service Provider gebunden wird. Für die einzelnen Aufgaben zwischen Service-Provider, Broker und Requestor werden Web-Service Protokolle verwendet.

Zum Austausch von Nachrichten auf Basis von XML wird das Simple Object Access Protocoll (SOAP) verwendet [Dus03]. Mittels SOAP lassen sich neben dem Einbinden von Daten auch entfernte Prozeduraufrufe realisieren und erlauben das Einbinden binärer Daten mittels MIME (Multipurpose Internet Mail Extension) [Kapp04]. SOAP spezifiziert nur den Aufbau der Nachrichten nicht aber den Transport. Hierfür ist kein spezielles Protokoll festgelegt, in der Regel wird aber HTTP verwendet. Die grundlegende Struktur einer SOAP Nachricht besteht aus einem SOAP-Envelope, welches optional einen SOAP-Header und zwingend einen SOAP-Body enthält.

Zur Beschreibung der Schnittstelle des Web-Services wird die Web Service Description Language (WSDL) verwendet [Dus03]. Mit Hilfe von WSDL wird die Art der Funktionsaufrufe durch den Nutzer festgelegt, aber auch die, wie Nachrichten, die von Web-Service angenommen werden, hinsichtlich Struktur und Inhalt aufgebaut sein müssen. Weiters wird die Struktur der zu erwartende Antwort spezifiziert. WSDL ist ebenfalls XML-basiert.

Zur Veröffentlichung bzw. dem Suchen und Auffinden von Webservices dient UDDI (Universal Description, Discovery and Integration) [Dus03]. UDDI spezifiziert ein Framework für das Beschreiben und Auffinden von Webservices aber auch die Integration von Webservices zu neuen Webservices. Es setzt dabei auf SOAP auf und ist ebenfalls in XML spezifiziert.

Internationalisierung von Web-Services

Da viele Anwendungen, die Web-Services einbinden, internationalisiert sind und Web-Services Ergebnisse liefern können, die locale-abhängig sind, ist auch die Internationalisierung von Web-Services ein Thema. Da Web-Services in nahezu beliebigen Programmiersprachen implementiert werden können und nur nach außen, wie weiter oben beschrieben, den diversen XML basierten Standards genügen müssen, können diese natürlich theoretisch internationalisiert werden. Der Punkt ist nun festzulegen, wie diese Tatsache nach außen also zum Nutzer eines Web-Services kommuniziert wird. Dabei können nach Einschätzung des W3C folgende vier Fälle auftreten [W3Cws08]:

1. *Locale Neutral*: In diesem Fall ist das angebotene Service nicht Locale behaftet, z.B. die Addition von zwei Integer-Werten würde als locale neutral eingestuft.

2. *Data Driven*: Die gelieferten Daten bestimmen das Ausmaß der Localeabhängigkeit, z.B. werden Daten aus einer Datenbank als Antwort geliefert, so können diese eventuell eine locale-spezifische Sortierung aufweisen.

3. *Service Determined*: Das Service verwendet ein bestimmtes Locale. Dies kann entweder fix vorgegeben sein oder vom Host-Computer des Sevices abhängig sein.

4. *Client Influenced*: Das Service kann so aufgebaut sein, dass ein vom Client angefordertes Locale verwendet werden soll. Es wurde die Bezeichnung influenced bewusst gewählt, da der Client theoretisch zwar ein beliebiges Locale anfordern könnte, dieses aber nicht notwendigerweise von Service unterstützt werden muss.

In [W3Cws08] wird daher ein Satz von XML Elementen, die in SOAP Nachrichten zur Spezifizierung von Internationalisierungsinformationen verwendet werden können, festgelegt. In diesen Tags können z.B. das Locale, Zeitzoneninformation und sonstige Voreinstellungen in einer festgelegten Syntax definiert werden, welche teilweise der Locale Data Markup Language (LDML) entspricht, die im Unicode Common Locale Data Repository (CLDR) Anwendung findet. Weiters wird ein zusätzliches XML Element beschrieben, welches in WSDL Beschreibungen die Möglichkeit der Verwendung der Erweiterungen für SOAP Nachrichten anzeigt.

2.2 Schrift

Einer der augenscheinlichsten Punkte der Internationalisierung ist die Sprache bzw. deren Darstellung in Form von Schrift. Weltweit gibt es nicht nur ein Schriftsystem sondern viele verschiedene und auch Unterschiede bei Verwendung desselben Schriftsystems für verschiedene Sprachen. Daher sollte man sich einen Überblick über einige der Schriftsysteme verschaffen, um deren Unterschiede und damit auftretende Probleme bei der automatisierten Verarbeitung zu erkennen. Hier wurde auch schon ein wesentlicher Unterschied genannt: Ein Schriftsystem ist keine Sprache. So wird zum Beispiel das Lateinische Schriftsystem sowohl für Deutsch als auch Englisch und einige andere Sprachen verwendet - meist mit Anpassungen an die jeweilige Sprache z.B. im Form von zusätzlichen Zeichen wie die Umlaute im Deutschen.

Historisch gesehen war die sogenannte Keilschrift das erste bekannte Schriftsystem, welches in Mesopotamien (im heutigen Irak gelegen) gegen Ende des 4. Jahrtausends vor unserer Zeit entwickelt wurde [DeiCza01]. Der Name Keilschrift kommt von der keilförmigen Einprägungen die durch den verwendeten Schreibgriffel entstanden. Zuerst würde vor allem Symbole für Dinge des täglichen Lebens entwickelt, später aber auch abstraktere Symboliken. Etwas später entwickelten die Ägypter die Hieroglyphen, durch welche sowohl Objekte als auch Töne ausgedrückt werden konnten.

Die Phönizier entwickelten Mitte des 2. Jahrtausends vor unserer Zeit das erste Alphabet [DeiCza01]. Die phönizischen Zeichen entsprachen also Klängen, die in der Sprache verwendet wurden. Das Alphabet bestand aus 22 Konsonanten aber keinen Vokalen. Die Sprache selbst enthielt natürlich Vokale allerdings war es mit den Konsonanten möglich Worte zu schreiben, die aus dem Kontext heraus verständlich waren. Das phönizische Alphabet bildet die Grundlage für fast alle alphabetischen Schirftsysteme auch des Lateinischen.

Man sieht also das anhand dessen, was die Schriftzeichen ausdrücken, seien es nun Symbole für ganze Wörter oder so wie beim Alphabet Laute, dass man anhand dieser Unterschiede Schriftsysteme klassifizieren kann. Der nächste Abschnitt stellt einige Unterteilungen der Schriftsysteme vor und in weiterer Folge wird jeweils ein Vertreter jeder Kategorie genauer behandelt.

2.2.1 Klassifizierung von Schriftsystemen

In [DeiCza01] werden Schriftsysteme nach folgenden Kategorien unterschieden:

Schrifttyp

- Alphabetschrift: Diese Schriftsysteme bestehen aus Zeichen für Konsonanten und Vokale (nicht bei allen Schriftsystemen). Vertreter sind folgende Schriftsysteme: Arabisch, Lateinisch, Hebräisch, Kyrillisch.
- Silbenschrift: Die einzelnen Schriftzeichen stehen für ganze Silben. Vertreter sind folgende Schriftsysteme: Japanisch und Koreanisch.
- Ideogrammschrift: Bei diesen Schriftsystemen werden durch die Symbole ganze Wörter bzw. Begriffe repräsentiert. Vertreter ist das Chinesische Schriftsystem.

Kontextabhängige Zeichendarstellung

- Position: je nach Position des Zeichens innerhalb eines Wortes wird es verschieden dargestellt. Dies tritt z.B. im arabischen, griechischen und hebräischen Schriftsystem auf.
- Ligaturen: Folgen bestimmte Zeichen aufeinander, werden diese als Ganzes anders dargestellt. Im arabischen Schriftsystem sind Ligaturen zwingend erforderlich, aber auch im Lateinischen (z.B. `fi` als fi) kommen sie vor, vor allem um ein schöneres Schriftbild zu erreichen.
- Kursiv: Manche Schriftsysteme, wie zum Beispiel das Arabische, erfordern eine kursive Darstellung.

Ausrichtung

- Links nach Rechts: Text verläuft wie zum Beispiel im lateinischen Schriftsystem üblich von links nach rechts.
- Bidirektional: Im arabischen Schriftsystem verläuft Text von rechts nach links. Zahlen oder lateinischer Text innerhalb von arabischem Text werden allerdings von links nach rechts geschrieben.
- Vertikal: Der Text verläuft von oben nach unten in Spalten, welche von rechts nach links geschrieben werden. Diese Ausrichtung kann zum Beispiel im chinesischen und japanischen Schriftsystem verwendet werden, ist aber nicht zwingend vorgeschrieben. Alternativ kann Chinesisch und Japansich auch horizontal von links nach rechts geschrieben werden.

Sonstige Charakteristika

- Diakritika: Manche Schriftsysteme bzw. Sprachen erfordern spezielle Zeichen, die als Zusatz zu einem Buchstaben spezielle Betonungen anzeigen. Französisch im lateinischen Schriftsystem erfordert Diakritika aber auch diverse osteuropäische Sprachen. In manchen Schriftsystemen können sich Diakritika auch über mehrere Zeichen erstrecken.
- Worttrennung: Die meisten Schriftsysteme verwenden Leerzeichen, um Wörter voneinander zu trennen. Für Chinesisch oder Japanisch gilt dies nicht.
- Satzzeichen: Die verwendeten Satzzeichen sind sehr unterschiedlich sowohl von einem Schriftsystem zum anderen als auch für unterschiedliche Sprache innerhalb desselben Schriftsystems.

2.2.2 Schriftsysteme

Um auf die unterschiedlichen Charakteristika auch in Form eines Beispiels einzugehen, werden im folgenden einige repräsentative Schriftsysteme genauer betrachtet.

Lateinisches Schriftsystem

Die Römer entwickelten das lateinische Schriftsystem als Adaptierung des griechischen Alphabets. In seiner jetzt gebräuchlichen Form besteht es aus 26 Zeichen, für die sowohl eine Darstellung als Großbuchstaben und auch als Kleinbuchstaben existiert. Das Schriftsystem wird heutzutage von vielen Kulturen verwendet.

Um die englische Sprache darzustellen, findet man mit den 26 Buchstaben das Auslangen, aber dies ist in vielen anderen Sprachen, die das lateinische System verwenden, nicht der Fall. So kommen zum Beispiel im Deutschen die Umlaute (Ä, Ü und Ö) hinzu, welche nach dem jeweiligen Basislaut (also Ä folgt auf A, Ü auf U und Ö auf O) in das Alphabet einsortiert werden. In den skandinavischen Ländern Finnland und Schweden (Å, Ä und Ö) bzw. Dänemark und Norwegen (Æ, Ø und Å) kommen ebenfalls jeweils 3 Buchstaben hinzu, welche allerdings nach dem Z einsortiert werden.

Das lateinische Schriftsystem wurde allerdings durch christliche Missionare und Kolonialisierung auch in anderen Teilen der Welt verbreitet und wird daher etwa in Südamerika, Australien und Indonesien, Teilen Afrikas und auch in der Türkei verwendet, wo es (erweitert auf 29 Zeichen) 1928 das bis dahin verwendete arabische Schriftsystem ersetzte.

Arabisches Schriftsystem

Durch die Verbreitung des Islams fand auch das arabische Schriftsystem weithin Verwendung. Heutzutage wird es in Nordafrika, der saudiarabischen Halbinsel entlang des persischen Golfes bis in Teile Zentralasiens verwendet. Es besteht aus 28 Buchstaben, welche, da es eine semitische Schrift ist, vorwiegend Konsonanten repräsentieren, nur 3 Zeichen können zur Darstellung von Vokalen verwendet werden [DeiCza01].

Im arabischen Schriftsystem wird Text von rechts nach links geschrieben, Zahlen und Einschübe in lateinischer Schrift aber von links nach rechts. Daher zählt das arabische so wie zum Beispiel das hebräische Schriftsystem zu den sogenannten bidirektionalen Schriftsystemen.

Eine weitere Besonderheit ist die Veränderung der Darstellungsform der Buchstaben je nach Position innerhalb des Wortes. Für viele Buchstaben gibt es daher bis zu vier Darstellungsformen je nachdem, ob der Buchstabe am Anfang, in der Mitte, am Ende des Wortes oder isoliert vorkommt (siehe Abbildung 1).

Ein weiterer integraler Bestandteil des Schriftsystems sind Ligaturen, d.h. wenn zwei bestimmte Buchstaben aufeinanderfolgend im Text vorkommen, verändert sich ihre Darstellungsform zu einer neuen gemeinsamen Darstellung. Dies ist deshalb von großer Bedeutung, da Worte die aus bloßer Aneinanderreihung von Buchstaben bestehen, nicht als korrekt angesehen werden.

Chinesisches Schriftsystem

Das chinesische Schriftsystem wurde in der ersten Hälfte des 3. Jahrtausends vor unserer Zeit entwickelt. Zuerst wurden Piktogramme verwendet, die konkrete Objekte repräsentierten. Die Schrift wurde dann um einfache und immer komplexere Ideogramme, welche zur Darstellung abstrakter Begriffe dienen, ergänzt. Basierend auf dieser Unterscheidung werden in [Lun02] die chinesischen Zeichen in 4 Kategorien unterteilt:

- Piktogramme stellen konkrete Objekte wie Sonne, Mond oder Baum dar.
- Einfache Ideogramme werden zu Darstellung von abstrakten Begriffen wie Richtungsangaben, Positionsangaben oder Zahlen verwendet.
- Zusammengesetzte Ideogramme werden aus Piktogrammen und einfachen Ideogrammen gebildet, um z.B. aus der Kombination des Zeichens für Baum die Begriffe Wald oder Wälder zu bilden.

ARABISCHE NESKHI.

Name	Ende	Mitte	Anfang	Isolirt	Wert	Name	Ende	Mitte	Anfang	Isolirt	Wert
Elif	ا			ا	'a	*Ta*	ط	ط	ط	ط	d̲
Be	ب	بـ	بـ	ب	b	*Tza*	ظ	ظ	ظ	ظ	ẓ
Te	ت	تـ	تـ	ت	t	*'Ain*	ع	عـ	عـ	ع	ʒ
θe	ث	ثـ	ثـ	ث	θ	*Γain*	غ	غـ	غـ	غ	γ
Džim	ج	جـ	جـ	ج	dž	*Fe*	ف	فـ	فـ	ف	f
Hha	ح	حـ	حـ	ح	h'	*Qaf*	ق	قـ	قـ	ق	q
Kha	خ	خـ	خـ	خ	χ	*Kef*	ك	كـ	كـ	ك	k
Dal	د			د	d	*Lam*	ل	لـ	لـ	ل	l
Dzal	ذ			ذ	ð	*Mim*	م	مـ	مـ	م	m
Re	ر			ر	r	*Nun*	ن	نـ	نـ	ن	n
Ze	ز			ز	z	*He*	ه	هـ	هـ	ه	h
Sin	س	سـ	سـ	س	s		ة			ة	t
Šin	ش	شـ	شـ	ش	š	*Wav*	و			و	w
Sad	ص	صـ	صـ	ص	ṣ	*Ye*	ى ي	يـ	يـ	ى	y
Zad	ض	ضـ	ضـ	ض	ẓ	*Lam-elif*	لا			لا	la

Ligaturen.

Zeichen	Wert	Zeichen	Wert	Zeichen	Wert	Zeichen	Wert	Zeichen	Wert
بم بم	bm	تھ تھ	th	ثی ثی	θy	كا كا	ka	مجـ مجـ	mdž
تم تم	tm	ثھ ثھ	θh	نی نی	ny	كل كل	kl	مم مم	mm
ثم ثم	θm	نھ نھ	nh	يي يي	yy	كم كم	km	نجـ نجـ	nmdž
نم نم	nm	يھ يھ	yh	فی	fy	لجـ لجـ	ldž	سجـ	sdž
يم يم يم	ym	بی بی	by	قی	qy	لم لم	lm	سم سم	sm
بھ بھ	bh	تی تی	ty	قم قم	qm	لھ لھ	lh	سھ سھ	sh

Abbildung 1: Arabische Schriftzeichen und Ligaturen [Fau1880]

- Phonetische Ideogramme bestehen zumindest aus zwei Komponenten, eine, die auf die Lesart bzw. Sprechweise hinweist, und eine bedeutungsgebende Komponente.

Die Schriftzeichen bestehen aus sogenannten Radikalen und radikal-ähnlichen Elementen, welche sich wiederum aus Strichen zusammensetzen. Ein chinesisches Zeichen besteht immer aus mindestens einem Radikal. Das Hauptradikal wird zur Ordnung der Zeichen verwendet. Zusätzlich werden die Anzahl und Reihenfolge der Striche zur Sortierung herangezogen.

Chinesische Schriftzeichen können zu einem Verbund zusammengefasst werden. Jedes Schriftzeichen muss in ein imaginäres Quadrat derselben Größe passen - auch ein Zeichenverbund. Die einzelnen Teilelemente (Zeichen) werden nach Bedarf gestaucht oder gestreckt.

Wörter werden in der chinesischen Schrift nicht durch Leerzeichen getrennt, jedes Zeichen hat denselben Abstand zum nächsten. Bei der Textausrichtung gibt es 3 Varianten: Die traditionelle vertikal in Spalten von rechts nach links, welche in Taiwan vorherrscht, horizontal von links nach rechts, welche in der Volksrepublik China vorherrscht oder aber horizontal von rechts nach links, welche nur bei sehr kurzen Texten (weniger als 1 Zeile) Anwendung findet.

Japanisches Schriftsystem

Im japanischen Schriftsystem werden 4 Schriften kombiniert. Zum einen bilden vom chinesischen Schriftsystem übernommene Zeichen, die als *kanji* bezeichnet werden, einen Großteil des Zeichenvorrats. Von der japanischen Regierung wurden 1945 Zeichen festgelegt, die für den täglichen Gebrauch und die Verwendung in Zeitungen und Publikationen bestimmt sind. Hinzu kommen *hiragana* und *katakana*, welche gemeinsam als *kana* bezeichnet werden. Beide Schriften stellen Silben dar. Sie beinhalten Zeichen, die die Vokale a, e, i, o, und u und das n darstellen, und solche, die eine Kombination aus einem Konsonanten und einem der Vokale darstellen. Hiragana wird vor allem für grammatikalische Prä- und Suffixe verwendet, obwohl japanisch komplett in Hiragana geschrieben werden kann und japanische Kinder dies vor dem Erlernen des kanji auch tun.

In Katakana werden vor allem Begriffe geschrieben, die in die japanische Sprache übernommen wurden.

Die vierte verwendete Schrift ist das lateinische Alphabet und die arabischen Ziffern. Diese wird als *romaji* bezeichnet und wird für westliche Namen, Titel usw. verwendet.

		K	S	T	N	H	M	Y	R	W	G	Z	D	B	P
A	あ	か	さ	た	な	は	ま	や	ら	わ	が	ざ	だ	ば	ぱ
I	い	き	し	ち	に	ひ	み		り	ゐ	ぎ	じ	ぢ	び	ぴ
U	う	く	す	つ	ぬ	ふ	む	ゆ	る		ぐ	ず	づ	ぶ	ぷ
E	え	け	せ	て	ね	へ	め		れ	ゑ	げ	ぜ	で	べ	ぺ
O	お	こ	そ	と	の	ほ	も	よ	ろ	を	ご	ぞ	ど	ぼ	ぽ
N	ん														

Tabelle 2: Hiragana Silbenzeichen [Lun02]

		K	S	T	N	H	M	Y	R	W	G	Z	D	B	P
A	ア	カ	サ	タ	ナ	ハ	マ	ヤ	ラ	ワ	ガ	ザ	ダ	バ	パ
I	イ	キ	シ	チ	ニ	ヒ	ミ		リ	ヰ	ギ	ジ	ヂ	ビ	ピ
U	ウ	ク	ス	ツ	ヌ	フ	ム	ユ	ル		グ	ズ	ヅ	ブ	プ
E	エ	ケ	セ	テ	ネ	ヘ	メ		レ	ヱ	ゲ	ゼ	デ	ベ	ペ
O	オ	コ	ソ	ト	ノ	ホ	モ	ヨ	ロ	ヲ	ゴ	ゾ	ド	ボ	ポ
N	ン														

Tabelle 3: Katakana Silbenzeichen [Lun02]

Es ist grundsätzlich zulässig, dass ein japanischer Text Elemente jeder der vier Schriftsysteme (*kanji*, *hiragana*, *katakana* und *romaji*) enthält. Weiters ist als Textausrichtung sowohl die traditonelle Art (vertikale in Spalten von rechts nach links) als auch die horizontale von links nach rechts zulässig.

2.2.3 Zeichensätze und Zeichenkodierung

Eine Menge von Zeichen, die zum Ausdrücken bestimmter Inhalte verwendet werden, wird als Zeichensatz bezeichnet [DeiCza01]. Man kann sogenannte nichtkodierte Zeichensätze und kodierte Zeichensätze unterscheiden. Nichtkodierte Zeichensätze werden nicht für die Verarbeitung mit einem Computer definiert, sondern zum Beispiel aus pädagogischen Gründen, um bei umfangreichen Schriftsystemen eine Teilmenge an Zeichen zu definieren, die zuerst erlernt werden müssen, oder um zur Darstellung von Formel einen festgelegten Satz an Symbolen zu definieren. Kodierte Zeichensätze werden im Hinblick auf die Verarbeitung mit dem Computer definiert und basieren auf der Abbildung der einzelnen Zeichen auf Nummern (Integer-Werte), die dann im Computer verarbeitet werden können. Zu jedem kodiertem Zeichensatz gehört mindestens eine Zeichenkodierung, welche festlegt, wie die einzelnen Zeichen im Computer dargestellt werden.

Ein-Byte Zeichensätze

Bei Zeichensätzen mit weniger als 256 Zeichen erfolgt die Kodierung normalerweise durch die binäre Darstellung des zugewiesenen Zahlenwerts der einzelnen Zeichen. Als Beispiel seien hier der ASCII-Code[5] oder die ISO-8859 Serie erwähnt. Der ASCII Zeichensatz umfasst 128 Zeichen, darunter die Buchstaben a-z in Groß- und Kleinschreibung einige Satz- und Sonderzeichen. Zur Kodierung dieser 128 Zeichen benötigt man 7 Bit.

Da in anderen Sprachen wie z.B. Deutsch zusätzliche Zeichen benötigt werden, wurden die ISO-8859 Zeichensätze entworfen. Die Idee dabei war, um zu ASCII kompatibel zu sein, die ersten 128 Zeichen quasi von ASCII zu übernehmen und danach, unter Ausnützung der Tatsachen, dass mit einem Byte = 8 Bit weitere 128 Zeichen kodiert werden können, weitere Zeichen einzufügen. ISO-8859 gibt es in 15 Teilnormen (Tabelle 4), die unterschiedliche Sprachen abdecken.

Zeichensatznorm	[Kürzel], Sprache
ISO-8859-1	Latin-1, Westeuropäisch
ISO-8859-2	Latin-2, Mitteleuropäisch
ISO-8859-3	Latin-3, Südeuropäisch
ISO-8859-4	Latin-4, Baltisch
ISO-8859-5	Kyrillisch
ISO-8859-6	Arabisch
ISO-8859-7	Griechisch
ISO-8859-8	Hebräisch
ISO-8859-9	Latin-5, Türkisch
ISO-8859-10	Latin-6, Nordisch
ISO-8859-11	Thai
ISO-8859-13	Latin-7, Baltisch
ISO-8859-14	Latin-8, Keltisch
ISO-8859-15	Latin-9, Westeuropäisch
ISO-8859-16	Latin-10, Südosteuropäisch

Tabelle 4: ISO-8859-x Zeichensatznormen [Har07]

Multi-Byte Zeichensätze

Anders stellt sich die Situation bei Schriften dar, die nicht mit 256 Zeichen auskommen. Das chinesische Schriftsystem enthält etwa einige zehntausend Zeichen. Zur Kodierung solcher Mengen von Zeichen ist ein Byte pro Zeichen nicht ausreichend, daher ist man gezwungen mehrere Bytes zu verwenden. Aus deren Verwendung ergeben sich eine Reihe

[5] American Standard Code for Information Interchange

von Problemen. So variieren viele Kodierungsmethoden von Multi-Byte Zeichensätzen die Anzahl der Bytes je nach Zeichenbereich, um Speicherplatz zu sparen. Dies erschwert den Umgang mit den kodierten Zeichen. Es muss daher z.B. vor dem Löschen eines Zeichens festgestellt werden, wieviele Bytes dieses Zeichen tatsächlich belegt. Bei den Kodierungsmethoden kann zwischen 3 Kategorien unterschieden werden [Lun02]:

- Kodierung mit fester Länge verwendet für jedes Zeichen dieselbe Anzahl an Bits. Dies vereinfacht die Verarbeitung kann allerdings unnötig viel Speicherplatz verbrauchen.
- Modale Kodierungen wechseln zwischen einzelnen Teilbereichen des Zeichensatzes. Dazu werden sog. Escape-Sequenzen innerhalb des kodierten Textes benötigt, um feststellen zu können, welcher Teilbereich gemeint ist, und mit wievielen Bits nachfolgende Zeichen kodiert sind.
- Nicht modale Kodierungen benötigen keine speziellen Escape-Sequenzen sondern definieren bestimmte Intervalle für die einzelnen Teilbereiche des Zeichensatzes. Daran ist zu erkennen, wieviele Bytes das kodierte Zeichen benötigt.

2.2.4 Unicode

Der Anspruch des Unicode Standards ist es, einen Zeichensatz für geschriebene Zeichen und Text bereit zu stellen, welcher, mittlerweile in Version 5.1 verfügbar, neben den heutzutage weltweit verwendeten Schriftsystemen auch historische Schriftsysteme, Satzzeichen, mathematische und technische Symbole, geometrische Formen und typographische Symbole enthält. Die aktuelle Version 5.1 enthält über 100.000 Zeichen.

Grundsätzlich definiert der Unicode Standard für jedes enthaltene Zeichen einen sog. Code Point, welcher ein 16 Bit Wert (0x0000 bis 0xFFFF) ist. Dadurch ließen sich allerdings nur 65536 Code Points definieren. Deshalb wurde das Konzept der sog. Planes (Ebenen) eingeführt. Es gibt 17 Planes (0x00 bis 0x10), welche jeweils 65536 Code Points enthalten und deren Nummer als Präfix des Code Points verwendet wird. Daraus folgt eine theoretische Anzahl an kodierbaren Zeichen von 1.114.112 Zeichen (0x0000 bis 0x10FFFF). Theoretisch deshalb, weil für die Enkodierungsform UTF-16 (siehe unten) 2048 (0xD800 bis 0xDFFF) Code Points der Plane 0 für sogenannte surrogate pairs verwendet werden und nicht mit einem Zeichen belegt sind. Ein Surrogate Pair besteht aus einem High-Surrogate (0xD800 bis 0xDBFF) und einem Low-Surrogate (0xDC00 bis 0xDFFF). Daher ergibt sich die Anzahl an tatsächlich kodierbarer Zeichen mit 1.112.064.

Die einzelnen Planes/Ebenen sind in Blocks/Blöcke unterteilt, die zusammengehörende Zeichen (meist eines Schriftsystems) enthalten. Die wichtigste Plane/Ebene ist die Basic Multilingual Plane (BMP bzw. Plane 0), die aktuell gebräuchliche Schriftzeichen und Symbole sowie Satzzeichen enthält. Weitere Planes/Ebenen sind: die Supplementary Multilingual Plane (SMP bzw Plane 1), die zum Beispiel historische Schriftzeichen wie Gotisch und spezielle Symbole enthält - die Supplementary Ideographic Plane (SIP bzw. Plane 2) enthält CJK Zeichen, die entweder selten verwendet werden, oder für die in der BMP kein Platz mehr war.

Dem Unicode Standard liegen 10 Design Prinzipien zugrunde. Wobei anzumerken ist, dass nicht alle Prinzipien gleichzeitig erfüllt werden können. Es existiert ein Spannungsfeld zwischen Konsistenz, Einfachheit und Effizienz aber auch Kompatibilität. Diese Prinzipien sind nach [Uni06]:

- Universalität
- Effizienz
- Zeichen, keine Glyphen
- Semantik
- Klartext
- logische Anordnung
- Unifikation
- dynamische Komposition
- Stabilität
- Konvertierbarkeit

Universalität bezieht sich dabei auf das Bereitstellen eines einzigen sehr großen Zeichensatzes. Ziel ist es, um den Anforderungen verschiedenster BenutzerInnen zu genügen, alle weltweit relevanten und in gewissen Masse standardisierten Schriftsysteme (sowohl aktuelle als auch historische) abzudecken.

Effizienz wird dadurch erlangt, dass alle Zeichen gleich behandelt werden und bei der Enkodierung keine Escape-Sequenzen erforderlich sind, was das Parsen von Unicode

erleichtert. Weiters sind zu einem Schriftsystem gehörige Zeichen soweit als möglich gruppiert.

Das Prinzip *Zeichen, keine Glyphen* bedeutet, dass ausschließlich abstrakte Zeichen einen Code Point in Unicode haben und keine konkrete Ausformung von Zeichen (Glyphen). Glyphen werden erst durch Unicode kompatible Schriftarten bei der Darstellung eines Code Points verwendet und haben nichts mit dem Unicode Standard an sich zu tun. Ähnlich verhält es sich beim Prinzip Klartext. Der Unicode Standard enkodiert Klartext, also eine Aneinanderreihung von Code Points. Werden so wie z.B. bei HTML mittels Tags, die wiederum als Zeichen kodiert sind, Formatierungsrichtlinien hinterlegt, so werden diese vom Unicode Standard nicht gesondert behandelt.

Zu jedem Unicode Zeichen können, nach dem Prinzip *Semantik*, in der Unicode Character Database eine Vielzahl von Character Properties gespeichert sein, die z.B. beschreiben, zu welchem Schriftsystem das Zeichen gehört, ob es ein Klein- oder Großbuchstabe ist oder auch welche äquivalenten Kodierungen des Zeichens existieren. Die logische Anordnung bezieht sich auf die Reihenfolge der Code Points in einem Unicode enkodierten Text gegenüber der schlussendlichen Darstellung nach der Ausrichtung des jeweiligen Schriftsystem. Grundsätzlich werden die Code Points nach Eingabereihenfolge kodiert. Für die Ausgabe wird die in den Character Properties definierte Orientierung herangezogen.

Unifikation dient zu Vermeidung mehrfach enkodierter gleicher Zeichen, soweit dies möglich ist, da hierbei auch auf bestehende Kodierungen Rücksicht genommen werden muss.

Die *dynamische Komposition* ermöglicht das Versehen von Zeichen mit Akzent- und Tremazeichen. Der Unicode Standard sieht hierfür eine Menge an Zeichen, die keinen Platz verbrauchen, sondern auf das vor ihnen enkodierte Zeichen gestapelt werden.

Stabilität bezieht sich hauptsächlich auf die Tatsache, das ein einmalig zugewiesener Code Point nicht verändert oder gelöscht wird, um zu garantieren, dass bestehende Enkodierung sich nicht verändern.

Enkodierungen

Der Unicode Standard definiert 3 Enkodierungsformen: UTF-32, UTF-16 und UTF-8 [Uni06]. UTF steht dabei für *Unicode bzw. UCS*[6] *Transformation Format*. Alle 3 Enkodierungen decken dabei den gesamten Bereich der Code Points ab und können bei Bedarf verlustlos ineinander umgewandelt werden.

[6] Universal Character Set nach ISO/IEC 10646 definiert und entspricht de facto dem Unicode Standard

UTF-32 ist die einfachste Enkodierung der 3 Varianten. Der Wert des Code Points wird durch eine Kodierungseinheit von 32 Bit dargestellt, d.h. es gibt eine 1:1 Relation zwischen Code Point und kodiertem Wert. Für UTF-32 werden keine Surrogate Pairs benötigt, daher ist der deren Wertebereich (0xD800 bis 0xDFFF) und Werte oberhalb von 0x10FFFF (obere Grenze der Plane 16) als ill-formed definiert. UTF-32 stellt eine Enkodierung mit fixer Länge dar, somit wird jedes Zeichen mit gleich vielen Bits kodiert. Dies vereinfacht einerseits die Verarbeitung von Text, da a priori genau bekannt ist, wie lange ein Zeichen in Bits ist, verbraucht allerdings in der Regel unnötig viel Speicherplatz. Da die meistgenutzten Zeichen in der BMP liegen und deren Code Points mit 16 Bit auskommen, wird bei UTF-32 für diese Zeichen doppelt soviel Platz als nötig verwendet.

UTF-16 verwendet Kodierungseinheiten mit 16 Bit. Zeichen in der BMP (0x0000 bis 0xFFFF) können somit 1:1 abgebildet und mit einer 16 Bit Kodierungseinheit dargestellt werden. Für die Kodierung von Zeichen in den anderen Planes werden die schon oben erwähnten Surrogate Pairs verwendet und die Zeichen somit mit zwei 16 Bit Kodierungseinheiten dargestellt. Die Darstellung mit Hilfe der Surrogate Pairs betrifft Werte ab 0x10000. Dabei wird so vorgegangen, dass von dem zu kodierendem Wert 0x10000 abgezogen wird und das Resultat in zwei 10 Bit Blöcke geteilt und 0b110110 bzw. 0b110111 als Präfix vorangestellt wird. Somit kann alles außerhalb der BMP auf den Bereich der Surrogates abgebildet werden. UTF-16 ist eine variable nicht modale Enkodierung.

Skalar	**UTF-16**
xxxx xxxx xxxx xxxx	xxxx xxxx xxxx xxxx
000u uuuu xxxx xxxx xxxx xxxx	1101 10ww wwxx xxxx 1101 11xx xxxx xxxx

Anm.: wwww = uuuuu - 1

Tabelle 5: UTF-16 Bit Verteilung [Uni06]

UTF-8 ist eine variable nicht modale Enkodierungsmethode, welche bevorzugt für HTML und XML aber auch in anderen Bereichen Verwendung findet. Das Verfahren basiert auf 8 Bit Kodierungseinheiten, von welchen 1 bis 4 benötigt werden, um den gesamten Unicode-Bereich abzubilden. Im Präfix des ersten Byte (0b0, 0b110, 0b1110, 0b11110) wird dabei angezeigt, wieviele Folgebytes (Präfix 0b10) noch zu diesem Zeichen gehören. Die Kodierung von Werten unter 128 durch ein Byte entspricht exakt der AS-

CII Kodierung. Somit kann ASCII Code leicht als UTF-8 kodiert übernommen werden. Diese Eigenschaft wird als ASCII-Transparenz bezeichnet [Uni06].

Skalar	**1. Byte**	**2. Byte**	**3.Byte**	**4.Byte**
0000 0000 0xxx xxxx	0xxx xxxx			
0000 0yyy yyxx xxxx	110y yyyy	10xx xxxx		
zzzz yyyy yyxx xxxx	1110 zzzz	10yy yyyy	10xx xxxx	
000u uuuu zzzz yyyy yyxx xxxx	1111 0uuu	10uu zzzz	10yy yyyy	10xx xxxx

Tabelle 6: UTF-8 Bit Verteilung [Uni06]

2.2.5 Textvergleich in Unicode

Ein elementares Feld der Informatik ist die Verarbeitung von Text in Form von Strings bestehend aus Characters. Sei es das Sortieren einer Menge von Wörtern oder das Auffinden also Suchen bestimmter Elemente eines Textes. Beim Erlernen klassischer Algorithmen für diesen Zweck wird normalerweise von ASCII kodiertem Text ausgegangen. Aufgrund dieser Annahme und der Tatsache, dass nach der ASCII Kodierung die Großbuchstaben `A-Z` im Bereich von 0x41 bis 0x5A und die Kleinbuchstaben von 0x61 bis 0x7A jeweils aufsteigend kodiert sind, kann ein Vergleich des ASCII Wertes verwendet werden. Der reine Vergleich von Zeichencodes stößt allerdings schon auf seine Grenzen, wenn eine case-insensitive Sortierung gefordert ist.

Weitere Punkte ergeben sich, sobald regionale und kulturelle Unterschiede zu berücksichtigen sind. Die Regeln für Textvergleiche differieren nicht nur zwischen unterschiedlichen Schriftsystemen, sondern auch innerhalb einzelner Schriftsysteme je nach Region. Das lateinische Schriftsystem wird sowohl im Deutschen als auch im Schwedischen mit fast identen Zeichen verwendet. Beide kennen das Zeichen Ä, welches im Deutschen quasi erweitert zu AE nach dem A gereiht wird, im Schwedischen aber nach dem Z. Die Erweiterung eines Zeichens zu 2 für Vergleichszwecke wird als Expansion bezeichnet [DeiCza01]. Im Deutschen gibt es neben den Umlauten (ä, ö, ü), welche zu ae, oe, ue erweitert werden, auch das scharfe s (ß), welches zu ss erweitert wird. Das Gegenteil der Expansion ist die Kontraktion, welche 2 Zeichen als eines behandelt. Ein Beispiel hierfür ist ch im Tschechischen, welches zwischen h und i einsortiert wird. Die Behandlung von Akzentzeichen ist ebenfalls nicht einheitlich. In der Regel werden Unterschiede von Anfang zum Ende des Strings betrachtet. Die französische Sortierung sieht dies bei den Akzentzeichen allerdings umgekehrt vor [UniTS08].

normale Akzent Ordnung	`cote < coté < côte < côté`
französische Akzent Ordnung	`cote < côte < coté < côté`

Tabelle 7: Auswirkung von Akzentzeichen auf die Sortierung [UniTS08]

Durch die Verwendung von Unicode, der eine wichtige Basis der Internationalisierung ist, ergeben sich ebenfalls Umstände, die beim Textvergleich zu berücksichtigen sind. Durch die weiter oben erwähnte Möglichkeit der dynamischen Komposition ergibt sich die Situation, dass ein Zeichen auf mehrfache Weise in Unicode dargestellt werden kann. Um ein einheitliches Ergebnis zu erreichen, ist es in der Regel erwünscht, diese als gleich zu behandeln. Im Unicode Standard ist deshalb auch der Unicode Collation Algorithm (UCA) spezifiziert, der von Unicode kompatibler Software implementiert werden sollte. Die Spezifikation ist als logische Spezifikation ausgelegt, daher können konkrete Implementierungen davon abweichen, solange sie dieselben Ergebnisse produzieren [UniTS08].

Sortieren und Suchen

Aufgrund der Komplexität eines regions- und sprachspezifischen Textvergleichsverfahren wird im Unicode Collation Algorithm ein Ansatz spezifiziert, der mehrere Ebenen des Vergleichs vorsieht [UniTS08]. Die erste Ebene bezieht sich auf Unterschiede der Basiszeichen, Ebene 2 auf die Verwendung von diakritischen Zeichen also z.B. Akzentzeichen, Ebene 3 Unterschiede bei Groß- und Kleinschreibung und Ebene 4 Zeichensetzung. Prinzipiell ist es möglich, beliebige weitere Ebenen zu definieren, bis hin zur Einbeziehung von Zeichen, die normalerweise ignoriert werden (siehe Tabelle 8).

Ebene	**Beschreibung**	**Beispiel**
L1	Basiszeichen	role < roles < rule
L2	Akzentzeichen	role < rôle < roles
L3	Groß-Kleinschreibung	role < Role < rôle
L4	Zeichensetzung	role < ”role” < Role
Ln	normalerweise ignorierte Zeichen	role < ro□le < ”role”

Tabelle 8: UCA Ebenen des Textvergleichs [UniTS08]

Auf jeder Ebene kommen neue Unterscheidungskriterien dazu, wobei allerdings die Kriterien der darunterliegenden Ebenen erhalten bleiben. Die Konzeption sieht vor, dass die erste Ebene eine Basissortierung vorgibt und auf den drauffolgenden Ebenen sprach- und regionspezifische Unterschiede behandelt werden können. Um dies zu realisieren, sieht der UCA den sog. Default Unicode Collation Element Table vor. Diese Tabelle

ordnet jedem Zeichen ein Collation Element zu, welches für jede Ebene des Vergleich eine Gewichtung des Zeichens in Form eines 4stelligen Hex-Wertes festlegt.

Zeichen	Collation Element	Unicode Name
0300 "‘"	[0000.0021.0002]	COMBINING GRAVE ACCENT
0061 "a"	[06D9.0020.0002]	LATIN SMALL LETTER A
0062 "b"	[06EE.0020.0002]	LATIN SMALL LETTER B
0063 "c"	[0706.0020.0002]	LATIN SMALL LETTER C
0043 "C"	[0706.0020.0008]	LATIN CAPITAL LETTER C
0064 "d"	[0712.0020.0002]	LATIN SMALL LETTER D

Tabelle 9: Beispiel einer Collation Element Tabelle [UniTS08]

Für die in manchen Sprachen auftretende Expansion eines Zeichens (z.B. im Deutschen `ä => ae`) bzw. Kontraktion also das Behandeln mehrerer Zeichen als eines (z.B. ch im Tschechischen) ist folgende Lösung vorgesehen. Im Falle der Expansion eines Zeichens wird diesem als Collation Element eine Aneinanderreihung der Collation Elemente, in welche es expandiert wird, zugewiesen. Bei der Kontraktion wird einer Folge von einzelnen Zeichen ein (theoretisch auch mehrere) Collation Element zugewiesen. Solche Eigenheiten können nicht in der Default Unicode Collation Element Table (DUCET) berücksichtigt werden, da diese eine möglichst grundlegende Sortierung bieten soll. Der Unicode Standard sieht dafür ein Feature vor, dass sich "Tailoring" also frei übersetzt Maßschneidern nennt. Grundsätzlich versteht man darunter das Überführen des DUCET in einen anderen wohlgeformten Unicode Collation Element Table, der dann für eine gewünschte Art des Sortierens angepasst ist. Das Unicode Common Locale Data Repository, welches eine Vielzahl an Locale-Daten beinhaltet, stellt unter anderem auch locale-spezifische Regeln bereit, die für das Tailoring des UCA verwendet werden können.

Ein Voraussetzung für die Anwendung des UCA ist das Vorliegen eines normalisierten Eingabe-Strings. Durch die Existenz von zusammengesetzten Zeichen (nach dem Unicode Prinzip dynamische Komposition) und sogenannter Compatibility Characters (z.B. half-width und full-width Zeichen siehe Tabelle 10), die aus Kompatibilitätsgründen in Unicode enthalten sind, kann ein Unicode-String in 2 Formen vorliegen. Precomposed bedeutet, dass Zeichen enthalten sind, z.B. in Verbindung mit einem Akzent, für die ein eigener Code Point existiert, etwa `á` Code Point `U+00E1`. In der Decomposed Form werden diese Zeichen nach dem Prinzip dynamische Komposition mittels Basiszeichen und einem oder mehreren Kombinationszeichen, die keinen eigenen Platz verbrauchen, zusammengesetzt. Für das obige Beispiel würde dies bedeuten: `á` Code Point `U+0061 U+0300`.

Die Normalisierung kann theoretisch in beide Richtungen erfolgen, für den UCA ist allerdings die Decomposed Form gefordert. Mit Hilfe der Unicode Canonical Algorithm [Uni06] kann dies erreicht werden. Darin werden einerseits unter anderem zwei Arten von Dekomposition beschrieben und die dazu notwendigen Mapping-Tabellen festgelegt. Die Canonical Decomposition verwendet die Unicode canonical Decomposition Mappings, um rekursiv zusammengesetzte Zeichen zu ersetzen. Die Compatibility decomposition verwendet zusätzlich die Unicode compatibility decomposition Mappings.

	Half-Width	**Full-Width**
Latin	ABCDEF	Ａ Ｂ Ｃ Ｄ Ｅ Ｆ
Katakana	ｱｲｳｴｵｶ	ア イ ウ エ オ カ

Tabelle 10: Half-Width und Full-Width Zeichen (Beispiel) [DeiCza01]

Für die normalisierten Strings kann dann mittels des DUCET ein Sort Key bestehend aus den einzelnen Collation Elementen generiert werden, welcher dann für binäre Vergleiche herangezogen wird. Für den Aufbau des Sort Keys werden für jede zu betrachtende Ebene die Gewichte der jeweiligen Zeichen aneinandergereiht und zwischen den einzelnen Ebenen ein level separator (0x0000) eingefügt. Tabelle 11 zeigt ein Beispiel der Sortierung mit Hilfe der generierten Sort Keys.

String	**Sort Key**
`cab`	**0706** 06D9 06EE 0000 0020 0020 **0020** 0000 **0002** 0002 0002
`Cab`	**0706** 06D9 06EE 0000 0020 0020 **0020** 0000 **0008** 0002 0002
`cáb`	**0706** 06D9 06EE 0000 0020 0020 **0021** 0020 0000 0002 0002 0002 0002
`dab`	**0712** 06D9 06EE 0000 0020 0020 0020 0000 0002 0002 0002

Tabelle 11: Beispiel für die Sortierung mittels UCA [UniTS08]

Das Suchen in Unicode Strings ist sehr ähnlich dem Sortieren, nur das es dabei nicht darauf ankommt, eine Reihenfolge festzulegen, sondern ob ein String gleich dem anderen oder einem Teil des anderen Strings ist [UniTS08]. Das Prinzip der mehreren Ebenen (siehe oben) wird auch bei der Suche beibehalten. Sucht man also nur nach Unterschieden der Ebene 2, werden Unterschiede der Ebene 3 (im Normalfall Groß/Kleinschreibung) nicht berücksichtigt und man erhält in diesem Fall eine case-insensitive Suche. Die Suche kann grundsätzlich mit der schon bei der Sortierung angewandten Methode mittels Sort-Key durchgeführt werden. Diverse regionale und sprachspezifische Eigenheiten können und müssen wiederum durch die Methode des Tailoring auf eine per Locale Basis

angepasst werden. Dies betrifft zum Beispiel für Deutsch die Behandlung der Expansion von Umlauten und dem scharfen s.

Auffinden von Textbegrenzungen

Eine häufige Aufgabe ist das Auffinden Begrenzungen in einem Text. Gemeint sind hier Begrenzungen, wie sie die BenutzerInnen verstehen, also natürlichsprachliche Konzepte wie Zeichen, Wörter, Sätze oder mögliche Positionen für ein Zeilenende. Dies ist keine einfache Aufgabe, da z.B. möglicherweise etwas das für die BenutzerInnen ein Element darstellt auf der Ebene eines Unicode kodierten Strings nicht als solches identifiziert werden kann. Es könne vier Arten von Textbegrenzungen unterschieden werden [Uni06]:

- Zeichenbegrenzungen
- Wortbegrenzungen
- Zeilenbegrenzungen
- Satzbegrenzungen

Der Unicode Standard beschreibt Spezifikationen zu diesen vier Arten der Textbegrenzungen und enthält Default-Regeln, um diese in einem Unicode enkodierten Text aufzufinden. Diese Regeln können bei konkreten Implementierungen verwendet werden. In der Spezifikation ist auch so wie schon beim Unicode Collation Algorithmus die Möglichkeit des Taylorings, also der Anpassung an bestimmte Gegebenheiten vorgesehen.

Zeichenbegrenzungen

Die Regel ein Unicode Code Point entspricht dem, was BenutzerInnen als Zeichen wahrnehmen, kann nicht gelten, da durch die dynamische Komposition von Zeichen, also das Kombinieren von Base Charakters und Combining Marks viele Zeichen, die von BenutzerInnen als ein Zeichen interpretiert werden, aus zwei oder mehr Code Points bestehen. In manchen Sprachen, wie z.B. Koreanisch, stellen Silben die kleinste Einheit von Text dar. Diese Silben werden allerdings aus sog. Jamo, die jeweils einem Unicode Code Point entsprechen, zusammengesetzt [DeiCza01]. Teilweise ist aber auch für Silben zusätzlich ein eigener Code Point vorhanden. Die primäre Anwendung ist im Bereich von Editoren, um etwa den Cursor korrekt zu bewegen bzw. Zeichen nach der Sichtweise des Users zu löschen.

Wortbegrenzungen

Das Auffinden von Wortbegrenzungen scheint durch das in vielen Schriftsystemen verbreitete Konzept von sog. Whitespace (Leerzeichen, Tabulatoren, Umbrüche etc.) erleichtert zu werden. In Schriften wie Thai, Lao, Khmer und anderen ist es nicht gebräuchlich, Whitespace zu verwenden. In diesen Fällen sind zum Auffinden von Wortbegrenzungen zusätzliche Techniken notwendig, z.B. ein Vergleich mit einem Wörterbuch. Die Unicode Default Regeln für Wortgrenzen liefern zwar eine abschätzbare Basis, decken aber nicht alle Eigenheiten von Schriften ab, weshalb bei der Implementierung zusätzliche Anpassungen notwendig sein können. Anwendungen für das Auffinden von Wortgrenzen sind z.B. intelligentes Copy & Paste, welches beim Ausschneiden eines Wortes die Leerzeichen an den Grenzen zu einem zusammenfasst, oder Suchfunktionen die als Option nur Ergebnisse liefern, wo komplette Wörter übereinstimmen oder sich mehrere Suchbegriffe innerhalb einer definierbaren Wortanzahl befinden [Uni06].

Zeilenbegrenzungen

In diesem Fall ist das Ziel das Auffinden von möglichen Positionen im Text, um bei der Darstellung bzw. Ausgabe umzubrechen. Die dazu existierenden Regel in verschiedenen Schrift- und Sprachsystemen sind sehr unterschiedlich und durchaus komplex. In [Uni06] werden drei grundlegende Stile unterschieden. Im westlichen Stil werden Leerzeichen und Trennzeichen zur Festlegung herangezogen. Der ost-asiatische Stil erlaubt Umbrüche überall, es sei denn sie sind explizit verboten und im südost-asiatischen Stil sind morphologische Analysen notwendig. Der letztgenannte Stil wird vom vorgeschlagenen Unicode Algorithmus allerdings nicht abgedeckt.

Satzbegrenzungen

Das Aufteilen von Text in diejenigen Teile die von BenutzerInnen als Satz interpretiert werden, ist für manche Fälle ohne Kenntnis des Sinns schwer zu erreichen. Mit den Unicode default Regeln können passable Ergebnisse erzielt werden, die auch einige Sonderfälle beinhalten. Der Punkt dient in vielen Sprachen als Satzbegrenzung, kann aber auch für Abkürzungen oder als Trennzeichen vor Nachkommastellen verwendet werden. Anwendungen sind z.B. triple-click Aktionen zum einfachen Markieren ganzer Sätze [Uni06].

Reguläre Ausdrücke

Reguläre Ausdrücke stellen ein mächtiges Werkzeug zur Bearbeitung von Texten dar.

Grundsätzlich können mit Hilfe von regulären Ausdrücken Muster beschrieben werden, welche dann von einer RegEx Maschine auf einen Text angewendet werden können. Es werden verschieden Metazeichen bei der Beschreibung der Muster verwendet, die elementarsten sind in Tabelle 12 aufgelistet.

Metazeichen	**Bedeutung**
`.`	ein beliebiges Zeichen
`[...]`	eines der in den Klammern aufgeführten Zeichen
`x?`	ein oder kein Vorkommen von x
`x*`	kein oder mehrere Vorkommen von x
`x+`	mindestens ein oder mehrere Vorkommen von x

Tabelle 12: Elementare RegEx Metazeichen [Frie08]

Das Metazeichen `[..]` legt eine Zeichenklasse fest und wird z.B. als `[a-zA-Z0-9]` verwendet, um ein alphanumerische ASCII Zeichen zu beschreiben. Dies funktioniert für den begrenzten Zeichenvorrat des ASCII Codes sehr gut, ist aber für die Menge an Zeichen in anderen Schriftsystemen nicht anwendbar. Der Unicode Standard weist mit den schon mehrfach erwähnten kombinierenden Zeichen und dem Vorhandensein von mehreren Kodierungen für dasselbe Zeichen aus Kompatibilitätsgründen einige Eigenheiten auf, die bei konkreten Implementierungen von RegEx Maschinen, die auch Unicode verarbeiten sollen, beachtet werden müssen.

In [UniTS05] werden 3 Stufen der Unicode-Unterstützung durch RegEx Maschinen beschrieben. Eine Regex Maschine die *Basic Unicode Support* konform ist muss, demnach unter anderem folgende Eigenschaften aufweisen: Um alle Unicode Zeichen abzudecken, muss es möglich sein, Zeichen als Hex-Werte anzugeben. Die Unicode Character Properties, welche z.B. angeben ob ein Buchstabe ein Groß- oder Kleinbuchstabe ist, müssen zur Festlegung von Zeichenklassen verwendet werden können. Weiters wird die Unterstützung von einfachen Wortgrenzen und von Zeilenbegrenzungen verlangt.

Die 2te Stufe *Extended Unicode Support* verlangt Unterstützung der bei den Textbegrenzungen erwähnten Unicode default Regeln für Zeichen- und Wort-Begrenzungen. Es muss auch möglich sein, Zeichen mittels ihres Unicode Namens im regulären Ausdruck anzusprechen und kanonische Äquivalenzen durch das Vorhandensein mehrerer Kodierungen desselben Zeichens müssen von der RegEx Maschine korrekt behandelt werden, d.h. es muss für beide Kodierungsvarianten ein Match erfolgen, ohne das diese explizit im Ausdruck aufgeführt sind.

Die Anforderungen der 3ten Stufe *Tailored Support* setzt, wie der Name vermuten lässt, Möglichkeiten für das bereits mehrmals erwähnten Tailoring voraus. Erst RegEx Maschinen die diese Voraussetzungen erfüllen, lassen sich locale-abhängig z.B. Punktsetzung, Wortgrenzen oder Zeichengrenzen (z.B. das Behandeln einer Kombination von Zeichen als ein einziges in manchen Sprachen) anpassen. Da die Unterstützung der für die Stufe 3 geforderten Eigenschaften Auswirkungen auf die Performance haben kann, ist eine Möglichkeit zur Deaktivierung dieser ebenfalls Bestandteil der Voraussetzungen einer Stufe 3 konformen RegEx Implementation.

Die Verwendung von regulären Ausdrücken ist weit verbreitet und kann sowohl mittels Kommandozeilentools wie grep oder egrep erfolgen, als auch über in vielen Programmiersprachen vorhandene RegEx Implementationen in beliebigen Anwendungsprogrammen. Reguläre Ausdrücke können zum Beispiel bei der Validierung von Eingabedaten auch bei Webapplikationen Verwendung finden. Vor der Verwendung sollte man sich in die konkret verwendete Implementierung einlesen, da es große Unterschiede bezüglich Funktionsumfang und Syntax gibt [Frie08].

2.3 Formate, Datumsangaben, Einheiten, Systemnachrichten

Einen Bereich großer regionaler und kultureller Unterschiede stellt die Formatierung von Datumsangaben, Zeitangaben, Zahlen, Währungen etc. dar. Probleme ergeben sich dabei nicht nur bei der Ausgabe dieser Werte, sondern auch bei der Eingabe und vor allem der korrekten Validierung solcher Eingaben. Ebenfalls problematisch ist das Ausgeben von Systemnachrichten. Diese sind manchmal statisch, viel öfter enthalten sie allerdings dynamische Elemente, die z. B. die natürlichsprachliche Grammatik der Ausgabe beeinflussen und dem entsprechend berücksichtigt werden müssen. Bei Datumsangaben gibt es nicht nur zahlreiche unterschiedliche Formatierungen, sondern auch verschiedene Kalendersysteme. Dasselbe trifft auf die Formatierung von Zeitangaben zu, wobei die jeweilige Zeitzone zu berücksichtigen ist. Der erste Unterabschnitt Datums- und Zeitangaben geht auf diese Problematik ein. Danach wird die Formatierung von Zahlen und Währungen bzw. Einheiten allgemein behandelt. Abschließend findet die bereits erwähnte Ausgabe von Systemnachrichten genauere Betrachtung.

2.3.1 Datums- und Zeitangaben

Das Format von Datums- und Zeitangaben ist von Region zu Region unterschiedlich. Ein Amerikaner interpretiert die Angabe 10/05/2008 als 5. Oktober 2008, wohingegen ein Brite 10. Mai 2008 annimmt. Ein Österreicher dagegen würde aber 10.05.2008 schreiben, also andere Trennzeichen verwenden. Immerhin gilt in allen drei genannten Nationen der gregorianische Kalender, welcher zwar eine große Verbreitung weltweit hat, aber bei weitem nicht überall Anwendung findet. Weitere Unterschiede ergeben sich, wenn Datumsangaben nicht in ausschließlich nummerischen Format, sondern in erweiterten Formatierungen mit abgekürzten oder ausgeschriebenen Tages- und Monatsnamen auftreten. Diese unterscheiden sich nicht nur von Sprache zu Sprache, sondern manchmal auch innerhalb derselben Sprache. So ist in Deutschland Januar in Österreich aber Jänner die Bezeichnung für den 1. Monat des Jahres. Daher ergeben sich folgende Unterschiede bei Datums- und Zeitangaben.

- Trennzeichen
- Anordnung der einzelnen Elemente (Tag, Monat, Jahr)
- Textelemente bei erweiterten Formatierungen
- Kalendersysteme

- Zeitzonen

Kalendersysteme

Weltweit ist das Gregorianische Kalendersystem weit verbreitet. Benannt wurde es nach Papst Gregor XIII, welcher dessen Entwicklung initiierte, um die Abweichungen, die durch das damals verwendete Julianische Kalendersystem im Laufe der Zeit entstanden sind, zu korrigieren. Der Julianische Kalender, von Julius Cäsar 45 vor unserer Zeit eingeführt, sah alle 4 Jahre ein Schaltjahr mit einem zusätzlichen Tag vor. Da diese Regelung eine Abweichung zur tatsächlichen Dauer einer Sonnenumrundung ergab, die sich so auswirkte, dass sich in etwa alle 128 Jahre das Julianische Kalenderjahr gegenüber dem tropischen Jahr um einen Tag verschob. Der Jesuit Christophorus Clavius, Mathematiker und Astronom, schaffte es die Umlaufzeit der Erde um die Sonne exakt zu berechnen und danach die Regeln für das Auftreten von Schaltjahren anzupassen. So wie im Julianischen Kalender ist jedes Jahr, dass sich ohne Rest durch 4 teilen lässt, ein Schaltjahr: ausgenommen sind Jahre, die sich durch 100 ohne Rest teilen lassen, es sei denn, sie sind durch 400 ohne Rest teilbar. Eingeführt wurde der Gregorianische Kalender nach Donnerstag dem 4. Oktober 1582, dem letzten Tag nach dem Julianischen Kalender, mit Freitag dem 15. Oktober 1582. Der reformierte Kalender wurde nicht überall zur selben Zeit übernommen. Vor allem protestantische und orthodoxe Kirchen bzw. Länder übernahmen den neuen Kalender nur sehr zögerlich (z.B. England im Jahr 1752 oder Russland 1918).

Neben dem gregorianischen Kalender existieren noch einige andere Kalendersysteme weltweit [DeiCza01]. Es gibt nicht nur Kalender, die auf dem Umlauf der Erde um die Sonne basieren (Solarkalender), sondern auch solche, die den Umlauf des Mondes um die Erde als Grundlage verwendet (Lunakalender) aber auch Mischformen (Lunisolarkalender). Der islamische Kalender ist ein Lunakalender, basiert also auf dem Mondumlauf. Der hebräische Kalender ist ein Lunisolarkalender, bei dem das Mondjahr durch ein 13 monatiges Jahr alle 2 oder 3 Jahre an das Sonnenjahr angepasst wird. Der 13. Monat erhält dabei den Namen “Adar II“, da es den Monat Adar in jedem regulären jüdischen Jahr ebenfalls gibt. Die Zählung der Jahre ist natürlich auch je nach Kalender verschieden. Nach dem islamischen Kalender schreiben wir das Jahr 1429, nach dem jüdischen das Jahr 5768 bzw. 5769, da das jüdische Neujahr Ende September stattfindet.

Dem japanischen Kalendersystem liegt für die Unterteilung des Jahres das gregorianische System zu Grunde, wobei die Tage und Monate natürlich eigene Bezeichnungen haben. Allerdings wird für die Zählung der Jahre ein System von Ären der regierenden

Kaiser verwendet. Das Jahr, in dem ein Kaiser wechselt, wird zu beiden Ären gerechnet. In jüngerer Vergangenheit gab es folgende Ären:

Name der Ära	Periode
Meiji	von 8. September 1868 (Meiji 1) bis 29 Juli 1912 (Meiji 45)
Taisho	von 30. Juli 1912 (Taisho 1) bis 24 Dezember 1926 (Taisho 15)
Showa	von 25. Dezember 1926 (Showa 1) bis 7. Jänner 1989 (Showa 64)
Heisei	seit 8. Jänner 1989 (Heisei 1)

Tabelle 13: Ären im Japanischen Kalender seit 1868 [DeiCza01]

Zeitzonen

Bei Zeitangaben spielen einerseits, wie bei den Datumsangaben, unterschiedlichste Formatierungen, wie die verwendeten Trennzeichen, aber auch bei 12 Stundenangaben die Zusätze für vormittags und nachmittags eine entscheidende Rolle für die Ein- und Ausgabe. Andererseits muss auch die Unterteilung in Zeitzonen berücksichtigt werden. Dabei wird die Welt in 24 Zeitzonen unterteilt, die in etwa 15° breit sind. Vom Null Meridian, der durch Greenwich/London läuft, ausgehend wird in östlicher Richtung addiert und in westlicher Richtung subtrahiert. Der Zeitzone des Nullmeridians ist die UTC (Universal Time Coordinated) zugeordnet, die die GMT (Greenwich Mean Time) in ihrer Funktion als Weltzeit ersetzt. Die Abweichung von der UTC sind in der Regel in Stundenschritten. In einigen Ländern werden auf Grund ihrer großen Ost-West Ausdehnung mehrere Zeitzonen verwendet. Die strikte Einteilung in 15° Zonen existiert in der Praxis so nicht, da aus politischen bzw. geographischen Gründe gewisse Abweichung von dieser Regelung eingeführt wurden. Auch wird nicht in allen Ländern um gerade Stundenanzahlen vor- oder zurückgestellt, sondern es existieren einige halb stündliche Abweichungen (z.B. Iran UTC+3,5 oder Venezuela UTC-4,5). Darüber hinaus haben viele Staaten die Umstellung auf Sommerzeit (engl. daylight saving time) eingeführt, welche wegen der längeren Tage im Sommer mehr Tageslichtstunden bringt.

2.3.2 Zahlen und Einheiten

Ähnlich wie bei den Datums- und Zeitangaben gibt es auch bei der Formatierung von Zahlen und Währungen bzw. Einheiten im Allgemeinen regionale und kulturelle Unterschiede. Diese lassen sich wie folgt gliedern:

- Gruppierungs- und Dezimaltrennzeichen

- konkrete Darstellung der Ziffern
- Darstellung negativer Zahlen
- Währungsformatierung und Prozentdarstellung
- Maßeinheiten

Bei der Formatierung von Zahlenwerte gibt es verschiedene Varianten, wie bei großen Zahlen zur besseren Lesbarkeit Zifferngruppen voneinander getrennt werden, aber auch welches Zeichen zur Abgrenzung der Kommastellen verwendet wird. In Österreich ist der Punkt [.] zur Gruppierung und das Komma [,] als Dezimaltrennzeichen gebräuchlich. Im Anglo-amerikanischen Raum ist dies genau umgekehrt. In Russland und Frankreich wird zwar das Komma zur Trennung der Dezimalstellen verwendet als Gruppierungszeichen fungiert dort aber das Leerzeichen.

Beim Anzeigen einer Zahl wird die binär gespeicherte Zahl in einen String umgewandelt und nach dem jeweiligen lokalen Konventionen formatiert. Für die korrekte Ausgabe ist allerdings auch die konkrete Ausformung der Ziffern wichtig. Die in Europa verbreiteten "Arabischen Ziffern" wurden von den Arabern übernommen und in ihrer Form angepasst. Die Araber haben sie aber ihrerseits von den Indern übernommen, welche das Dezimalsystem entwickelt haben. Auch bei dieser Übernahme wurden die Glyphen verändert.

Auch die Darstellung negativer Zahlen, Prozentangaben und Währungen erfolgt sehr unterschiedlich. Bei der Formatierung von negativen Zahlen oder Prozentangaben variiert die Symbolik, mit der solche Werte ausgezeichnet werden, aber auch die Positionierung dieser Symbole zur Kennzeichnung. Negative Zahlen werden hauptsächlich durch vorangestelltes oder folgendes Minus möglichweise aber auch durch eine Klammerung gekennzeichnet. Prozentangaben können ebenfalls durch ein Prozentsymbol, welches kulturell nicht eindeutig ist und links oder rechts der Zahl autreten kann, markiert werden. Die Problematik der Währungsformatierung beinhaltet den Umgang mit und die richtige Platzierung von verschiedenen Währungskürzel und -symbolen, die nicht notwendigerweise nur aus einem Zeichen bestehen müssen. Abschließend sollten auch noch Unterschiede in den verwendeten Maßeinheiten Erwähnung finden, etwa bei Längenmaßen das europäische metrische System gegenüber dem anglo-amerikanischen Inchsystem. Bezüglich der Behandlung von Maßeinheiten und auch Währungen geht es einerseits um

europäisch	arabisch	arabisch-indisch	indisch Devanagari	Thai
0	٠	۰	०	๐
1	١	۱	१	๑
2	٢	۲	२	๒
3	٣	۳	३	๓
4	٤	۴	४	๔
5	٥	۵	५	๕
6	٦	۶	६	๖
7	٧	۷	७	๗
8	٨	۸	८	๘
9	٩	۹	९	๙

Tabelle 14: verschieden Ziffernzeichen [DeiCza01]

die korrekte Formatierung bei Ausgabe andererseits könnte auch eine Umrechnung gefordert sein, welche bei Währungen aufgrund der sich verändernden Umrechnungsfaktor eine verfügbare und aktuelle Datenquelle voraussetzt.

2.3.3 Ausgabe und Eingabe

Eine geeignete Technik, um mit den oben beschriebenen Unterschieden umzugehen, stellen locale-sensitive Objekte und Funktionen dar. Der Grundansatz ist es Objekte und Funktionen so zu implementieren, dass sie je nach gewähltem Locale ihre Funktionalität modifizieren. Dies kann in zwei Arten erfolgen:

- Es wird für die gesamte Applikation ein Locale gesetzt und alle locale-sensitiven Funktionen verwenden dieses.
- Jeder Funktion kann beim Aufruf ein Locale übergeben werden, was eine flexiblere Verwendung möglich macht. Wird kein Locale übergeben, verwendet die Funktion das default Locale der Systemumgebung.

Um dieses Verhalten zu ermöglichen, müssen einerseits zu allen Locales, die eine Applikation oder eine Programmierumgebung bzw. ein Framework unterstützt, für die einzelnen

Formatierungs- bzw. Parsingaufgaben die notwendigen Informationen vorhanden sein. Auf der anderen Seite müssen die locale-sensitiven Funktionen so implementiert sein, dass diese Informationen zur Abänderung des Verhaltens angewendet werden.

Eine wichtige Aufgabe ist die richtige Formatierung bei der Ausgabe sprich der Generierung des Dokuments, welches als Response an den Browser des Clients gesendet wird. Dabei muss ein mehr oder weniger komplexes Objekt (z.B. ein Datumsobjekt) der jeweils verwendeten Programmiersprache in einen String konvertiert werden. Dieser String muss nach den Richtlinien des gewählten bzw. aktuellen Locale formatiert werden.

Die Benutzerinteraktion einer Webapplikation umfasst in der Regel die Eingabe von Daten seitens der BenutzerInnen. Diese Daten können aus sämtlichen bis jetzt behandelten Kategorien stammen, angefangen von Text in allen Ausprägungen über Datums- und Zeitangaben, Zahlen und Währungen etc. Diese Eingabedaten müssen in den meisten Fällen gewisse Anforderungen erfüllen, dazu werden sie nach festgelegten Vorgaben geprüft. Diese können sich allerdings je nach Zielland bzw. Kultur verändern, nimmt man als Beispiel die Registrierung eines neuen Benutzerkontos für einen Onlineshop, werden unter anderem einige persönliche Daten benötigt:

- Name
  - Titel
  - Vorname
  - Nachname
- Adresse
  - Straße
  - Hausnummer
  - Postleitzahl
  - Bundesland
  - Land
- Geburtsdatum
- Email-Adresse
- Telefonnummer
- und einiges mehr

Typischerweise erfolgt in Webapplikation die Dateneingabe per HTML-Formular. Bei Aufbau des Formulars im Hinblick auf die Internationalisierung ergeben sich zwei grundlegende Probleme. Zum einen der Aufbau und die Beschriftung der Eingabeelemente. Mit der reinen Übersetzung der Eingabefelder ist es in vielen Fällen nicht getan. Es können verschiedenste Eigenheiten auftreten. Zum Beispiel gibt es nicht in jedem Land Bundesländer. Deshalb kann es erforderlich sein entweder Felder, die nur für manche Länder Gültigkeit haben als nicht erforderlich zu deklarieren oder das Formular länderspezifisch anders aufzubauen [Yun02]. Auch das Format von Postleitzahlen und Telefonnummern ist sehr unterschiedlich und muss dementsprechend berücksichtigt werden.

Die Validierung von Eingabedaten muss also bei einer internationalisierten Anwendung so flexibel ausgelegt sein, dass der zur Validierung aufgerufene Programmcode locale-abhängig unterschiedliche Vorgaben prüfen kann. Wird z.B. bei Namen auf [A-Z][a-z] also lateinische Buchstaben überprüft, werden sich viele internationale BenutzerInnen nicht registrieren können. Es geht allerdings nicht nur um die zulässigen Zeichen sondern auch um etwaige mindestens oder maximal zulässigen Längen der Eingabe. Diese sind natürlich in der Regel durch die zugrundeliegende Datenbank bzw. das erstellte Datenbankschema festgelegt und demnach schon bei dessen Erstellung zu berücksichtigen.

Grundsätzlich kann die Validierung sowohl am Server als auch am Client erfolgen. Clientseitig wird dazu Javascript verwendet. Dies hat den Vorteil, dass für die Validierung kein neuerlicher Request an den Server gestellt wird, sondern falls Bedingungen nicht erfüllt sind, ohne Neuladen der Seite Fehlermeldungen ausgegeben werden können. Es existieren auch Javascript Frameworks, die Internationalisierung und Lokalisierung unterstützen, allerdings nicht oder noch nicht in dem Ausmaß wie viele serverseitige Programmiersprachen und Frameworks. Javascript weist vor allem im Umgang mit Unicode (vor allem mit kombinierten Zeichen) und durch Funktionen die ausschließlich das Locale des Client Betriebsystems verwenden, einige Nachteile auf [Dojo08].

2.3.4 Systemnachrichten

Während der Interaktion mit den BenutzerInnen werden von einem System im Allgemeinen Nachrichten ausgegeben. Diese können einen Status beinhalten oder auf einen aufgetretenen Fehler hinweisen, zusätzliche Eingaben fordern oder Ähnliches. Rein statische Nachrichten können durch Externalisierung (z.B. Auslagern in sprachspezifische Dateien) und dynamisches Nachladen je nach gefordertem Locale behandelt werden.

Meist beinhalten Systemnachrichten allerdings dynamische Inhalte wie z.B. Datums- und Zeitangaben oder Anzahlen, die sich zur Laufzeit ergeben, und in die Nachricht sinngemäß integriert werden müssen. Dabei ergeben sich diverse Unterschiede bei der Übertragung einzelner Meldungen in andere Sprachen. Folgende Probleme ergeben sich bei der Internationalisierung von Systemnachrichten.

- Reihenfolge von dynamischen Elemente in einer Nachricht
- Pluralformen bei Anzahlen

Beim Programmieren ohne Internationalisierung im Hinterkopf werden gerne Meldungen durch das Aneinanderhängen von Strings zusammengebaut, z.B. so:

```
print("Das Verzeichnis" + Verzeichnis + "enhält" +
       Anz_Dateien + "Dateien.");
```

Das einfache Beispiel in Tabelle 15 verdeutlicht, dass bei der Übersetzung in andere Sprachen eine Änderung der Reihenfolge von dynamischen Elementen erforderlich sein kann.

Deutsch	Englisch
Das Verzeichnis /**home/e9726020** enthält **10** Dateien.	There are **10** files in directory /**home/e9726020**.

Tabelle 15: Reihenfolge von dynamischen Elementen (fett gedruckt)

Ein weiteres Problem stellen grammatikalische Unterschiede von Sprachen dar. Die englische Sprache, die meist den Ausgangspunkt bei einer Internationalisierung darstellt, kennt z.B. kein Konzept Geschlecht für Nomen und Verben. Andere Sprachen wie das Deutsche haben nicht nur männliche und weibliche sondern auch sächliche Wörter. In [DeiCza01] wird folgendes Beispiel zur Verdeutlichung diese Umstandes angeführt:

```
The printer is enabled.
The modem is enabled.
The network is enabled.
```

Man könnte nun versuchen, da das Wort enabled in allen drei Meldungen vorkommt dieses zu externalisieren und mehrfach zu verwenden. Ein Übersetzer, der die externalisierte Datei bekommt, hat dann keinen Anhaltspunkt zum Kontext, in dem das

übersetzte Wort verwendet wird. Zumeist kommt es dann dabei zu gänzlich unpassenden oder zumindest als "hölzern" zu bezeichnende Übersetzungen. Die Übersetzung der drei obigen Meldungen ins Französische verdeutlicht die Problematik.

```
L'imprimante activée.
Le modem activé.
Le réseau activé.
```

Es ist daher ratsam, Meldungen immer so anzulegen, dass sie komplett externalisiert werden. Somit kann ein Übersetzer ohne Kenntnisse der Programmiersprache und Vorliegen des Source Codes adäquate Übersetzungen erstellen. In seltenen Fällen wenn ein Wort immer im selben Kontext verwendet wird, z.B. als Beschriftung eines Abbrechen Buttons, kann man es mehrfach verwenden. Bei Meldungen ist es im Zweifelsfall immer besser sie eigens zu externalisieren.

Wenn man sich das obige Beispiel "Das Verzeichnis /home/e972620 enthält 10 Dateien." ansieht, steckt darin das nächste Problem bei der Internationalisierung von Meldungen. Wird nämlich die Meldung so wie oben mit den zwei dynamischen Elementen, nämlich dem Verzeichnisname und der Anzahl der gefundenen Dateien erstellt, ergibt sich im Fall, dass nur eine Datei gefunden wird, eine grammatikalisch nicht korrekte Ausgabe, die normalerweise von BenutzerInnen als unpassend empfunden wird. In so manchem Source Code findet man daher oft folgende Konstrukte, um mit Singular und Plural umzugehen:

```
print("Das Verzeichnis" + Verzeichnis + "enthält" +
       Anz_Dateien + "Datei(en).");
```

oder

```
if(Anz_Dateien == 1) {
   print("Das Verzeichnis" + Verzeichnis + "enthält" +
          Anz_Dateien + "Datei(en).");
} else {
   print("Das Verzeichnis" + Verzeichnis + "enthält" +
          Anz_Dateien + "Datei(en).");
}
```

Dies mag für Deutsch und Englisch, die nur eine Pluralform kennen, mehr oder weniger funktionieren. Bei Sprachen wie dem Polnischen gibt es allerdings mehrere Pluralformen [Drea07].

```
Datei = Plik

01    Plik
02-04 Pliki
05-21 Plików
22-24 Pliki
25-31 Plików
```

Da es viele Sprachen gibt, die mehrere Pluralformen nach unterschiedlichen Regeln verwenden, wird man dazu tendieren, die Behandlung dieses Problems wiederverwendbar auszulagern oder aber bestehende Internationalisierungsbibliotheken zu nutzen. So eine Bibliothek stellt GNU-gettext dar.

GNU-Gettext

GNU-Gettext ist eine Ansammlung verschiedener Komponenten mit dem Ziel, die Internationalisierung von Software zu erleichtern und zu unterstützen. Im Paket enthalten sind zum einen die Runtime Library für den Zugriff auf ausgelagerte Übersetzungen, einige Tools, um zu übersetzenden Text aus dem Source Code zu extrahieren, und in von Gettext verwendete Formate umzuwandeln. Ein Ziel von Gettext ist es, die Auswirkungen bei Verwendung der Bibliothek auf den zugrundeliegenden Programmcode möglichst gering zu halten. Die Autoren hoffen, dass dadurch die Akzeptanz der Bibliothek und von Internationalisierung allgemein erhöht wird [Drea07]. Gettext ist für die Verwendung mit C, C++ ausgelegt. Mittlerweile existieren aber Adaptionen für viele andere Programmier- bzw. Script-sprachen, so z.B Java, Python, Ruby, Pascal, PHP, Perl, Bash und sh Skripte, usw. Die Anwendung ist dabei weitgehend ähnlich der Anwendung unter C. Gettext findet breite Anwendung unter Linux, es sind allerdings auch Versionen für Mac-OS und Windows verfügbar.

Anwendung im Source Code

Um einen String mittels gettext zu lokalisieren, muss die gettext Funktion für diesen aufgerufen werden. Bei den meisten Programmiersprachen funktioniert dies auf 2 Arten:

```
printf (gettext ("String ‘%s’ has %d characters\n"), s, strlen (s));
```

In vielen Programmiersprachen ist es möglich, ein Makro zu definieren und somit eine kurze Schreibweise der Form `_()` für `gettext()` zu ermöglichen.

```
printf ( _("String ‘%s’ has %d characters\n"), s, strlen (s));
```

Der von der gettext Funktion eingeschlossene String fungiert dabei einerseits als Standardausgabe an dieser Stelle und andererseits als Identifier für die Übersetzung. Es sollte daher darauf geachtet werden, dass dieser String folgende Kriterien erfüllt[Drea07]. Er sollte zum einen in gutem Englisch und als ganzer Satz formuliert sein, da er Grundlage für eine spätere Übersetzung ist. Weiters sollte eine Ausgabe nicht auf mehrere Ausgabefunktionen verteilt werden, da dies für die Übersetzer nicht nachvollziehbar ist. Wie weiter oben schon erwähnt, sollte statt dem Aneinanderhängen von Strings mit Formatstrings gearbeitet werden. Formatstrings sind in vielen Programmiersprachen vorhanden, bei C können Formatierungsangaben direkt in einen mittels `printf()` auszugeben String geschrieben werden, in Java ist hierfür eine eigene Klasse MessageFormat vorhanden [Drea07] [DeiCza01]. Um Pluralformen zu handhaben, bietet gettext die Funktion `ngettext()`, an die beim Aufruf sowohl die Singularform als auch die Pluralform und die Variable für die Anzahl übergeben wird.

```
printf( ngettext("One File was deleted.",
                 "%d files were deleted",
                 filecount), filecount);
```

Das Verhalten bei unterschiedlichen Pluralregelungen kann über eine Regel, die in die Übersetzungsdatei aufzunehmen ist, gesteuert werden. Diese hat für Deutsch und Englisch, welche zur Gruppe 1 Pluralform und 1 Singularform ausschließlich für die Anzahl 1 gehört, folgende Form [Drea07].

```
Plural-Forms: nplurals=2; plural=n != 1;
```

Erstellen der Messagekataloge

Nachdem die zu übersetzenden Texte durch das Einbetten in den `gettext()` oder `ngettext()` Aufruf im Source Code ausgezeichnet sind, kann mit dem im Gettext Paket enthaltenen Tool `xgettext` ein sog. Portable Object Template (.pot) erstellt werden. Vom Übersetzer wird mit dem Tool `msginit` durch Angabe der gewünschten Sprache

ein Portable Object (.po), welches mittels Text Editor um die geforderten Übersetzungen erweitert wird. Zum Editieren der Portable Object Dateien gibt es auch ein Vielzahl an speziell angepassten Editor z.B. Poedit oder KBabel, die vor allem durch eine übersichtliche Darstellung die Übersetzung erleichtern [Doel08]. Falls es durch Umstellung oder Weiterentwicklung eine neue Version des Portable Object Template gibt, kann diese mittels des Programms `msgmerge` mit der bereits bestehenden Portable Object Datei abgeglichen werden, um nicht Übersetzungen erneut erstellen zu müssen. Damit die Übersetzung schlussendlich von gettext zur Programmlaufzeit verwendet werden kann, erstellt man mittels `msgfmt` ein Machine Object (.mo). Diese binären Machine Objects sind ein wesentlicher Vorteil gegenüber der Auslagerung in reine Textdateien, da sie zu einer Verbesserung der Performance führen [Drea07].

2.4 Farben, Layout, Kultur

Ein Teilbereich der Internationalisierung betrifft das Design von Benutzerschnittstellen und Interaktionen. Beginnend bei der Farbgebung über die Ausrichtung einzelner Komponenten bis hin zur Struktur der Benutzeroberfläche ergeben sich kulturelle und regionale Unterschiede, deren Berücksichtigung die Akzeptanz des Produktes durch die BenutzerInnen beeinflussen kann.

2.4.1 Farben und Symbole

Farben spielen im User Interface Design seit langem eine wichtige Rolle. Sie vereinfachen, richtig eingesetzt, die Bedienung des Systems durch die BenutzerInnen, deren Assoziationen mit gewissen Farben unterstützend ausgenützt werden. So zum Beispiel ist bei Mobiltelefonen in der Regel die Taste zum Beginnen oder Annehmen eines Anrufes grün und jene zum Auflegen oder Beenden rot eingefärbt oder markiert. Auf dem Gebiet der Mensch-Computer-Interaktion bzw. dem Interaktionsdesign wurden einige Gestaltungskriterien und Empfehlungen für die Anwendung von Farben publiziert. Zum Beispiel sollte bedacht werden, da ca. 10% (8-9% Männer und 1% Frauen) der Menschheit eine Farbfehlsichtigkeit aufweisen, dass nicht Rot-Grün oder Blau-Gelb zur farblichen Kodierung verwendet werden, da diese Farben von den Betroffenen nicht unterschieden werden können [Hein04]. Auch sollte eine Kodierung nicht allein auf Farbe beruhen, sondern immer ergänzend zur Kodierung durch Anordnung (z.B. Ampel) oder geometrische Formen. Desweiteren sind der Einsatz von zu vielen verschiedenen oder übermäßig gesättigten Farben kontraproduktiv [Coop07]. Bei der Internationalisierung ist zusätzlich die unterschiedliche Bedeutung bestimmter Farben in verschiedenen Kulturen problematisch und stellt einen Punkt dar, der beim Entwurf von Benutzerschnittstellen berücksichtigt werden muss. Es geht allerdings nicht nur um die Bedeutung von Farben, die kulturell sehr unterschiedlich ausfallen, sondern auch um die Wahrnehmung derselben.

Wahrnehmung von Farben

Farben werden nicht in jeder Kultur gleich wahrgenommen. Es gibt Hypothesen, dass die Farbwahrnehmung von den jeweiligen Umweltgegebenheiten und der in der jeweiligen Sprache vorhandenen Wörter für einzelne Farben beeinflusst wird [SinPer05]. In manchen Kulturen gibt es für Farben, die in anderen Kulturen selbstverständlich unterschieden werden, nur ein Wort. Nicht jede Kultur teilt das Farbspektrum gleichermaßen auf. Die Inuit etwa unterscheiden sehr viele Farbtöne von Schnee und Eis. Japaner erkennen die

Farbe “aoi”, welche als Grünschattierung bzw. grün und blau beschrieben werden kann. Solche Unterschiede sollten in die Überlegungen mit einfließen, wenn es z.B. darum geht, ein Farbkodierungschema für Teilbereiche der Applikation festzulegen, da diese dann auch für die BenutzerInnen unterscheidbar sein sollen.

Bedeutung von Farben

Die Bedeutung der Farben in den jeweiligen Zielkulturen muss ebenfalls bekannt sein und entsprechend Berücksichtigung finden, um bei unpassender Verwendung das Zielpublikum nicht abzuschrecken oder zu verärgern. Etwa steht die Farbe Weiß, die in Österreich und vielen anderen westlichen Kulturen für Reinheit und die Farbe der Braut, in vielen asiatischen Kulturen symbolisiert sie Tod und wird von Witwen getragen. Ähnliches gilt für Grün, welches in unserem Kulturkreis Sicherheit symbolisiert, bei einem Malayen allerdings Assoziationen mit dem Dschungel seinen Gefahren und Krankheiten auslöst [SinPer05].

	China	**Japan**	**Ägypten**	**Frankreich**	**USA**
Rot	Fröhlichkeit	Wut, Gefahr	Tod	Aristokratie	Gefahr, Stop
Blau	Himmel, Wolken	Schurkerei	Tugend, Vertrauen, Wahrheit	Freiheit, Frieden	Männlichkeit
Grün	Ming Dynastie, Himmel	Zukunft, Energie, Jugend	Fruchtbarkeit, Stärke	Kriminalität	Sicherheit, Gehen
Gelb	Geburt, Macht, Reichtum	Anmut, Adel	Fröhlichkeit, Wohlstand	Vergänglichkeit	Feigheit, Vergänglichkeit
Weiß	Tod, Reinheit	Tod	Freude	Neutralität	Reinheit

Tabelle 16: Bedeutung von Farben in verschiedenen Kulturen [BarBad98]

Vorsicht ist auch bei der Verwendung von Symbolen und Icons geboten. Einerseits muss für jedes Symbol geklärt werden, ob es für das beabsichtigte Ziellocale auch “funktioniert”. Das Fragezeichen als Symbol für die Hilfe ist nicht so allgemeingültig, wie man denkt. Im Griechischen wird jedenfalls nicht [?] zur Kennzeichnung einer Frage verwendet sondern der Strichpunkt [;]. Auch Icons, die Fingergesten enthalten, sind gefährlich, da sich für fast jede Fingergeste ein Land findet, in welchem diese als Beleidigung betrachtet würde. Icons die nicht verstanden werden, können zu Fehlbedienungen führen.

Das “trash can” Icon der Apple Macintosh verwirrte Britische BenutzerInnen, da es wie ein Britischer Briefkasten aussah und dazu führte das Mail darin landete [SinPer05] [Coop07].

2.4.2 Layout

Der Aufbau der Benutzerschnittstelle, welcher von BenutzerInnen aus verschiedenen Kulturkreisen als angenehm empfunden wird, unterscheidet sich ebenfalls stark. Die Textausrichtung des in der Zielkultur verwendeten Schriftsystems spielt eine wesentliche Rolle, wie das gesamte User Interface ausgerichtet sein sollte. Verläuft die Schrift von links nach rechts, wird auch eine linksbündige Ausrichtung der Elemente bevorzugt. Menüs befinden sich links oben, Navigation am linken und eventuell rechten Rand darin eingebettet nach rechts verlaufend der Anzeigebereich für die Daten (siehe Abb.2). In so einer Anordnung wird sich jemand, dessen Schriftsystem eine von rechts nach links verlaufende Textausrichtung vorsieht, nicht unbedingt wohl fühlen. Dieser würde ein rechtsbündiges User Interface bevorzugen - also Menüs von rechts oben beginnend, Navigation am rechten und eventuell linken Rand und Anzeigebereich der Daten nach links verlaufend (siehe Abb.3).

Diese Ausrichtung betrifft auch einzelne Komponenten wie zum Beispiel Combo-Boxes deren Ausklappbutton je nach Region entweder links oder rechts platziert sein sollte, aber auch diverse andere GUI Elemente. Vergleicht man z.B. die Anordnung der Suchkomponente in den beiden Varianten der Webseite der Stadt Jerusalem stellt man fest, dass in Abbildung 2 das Eingabefeld links und der Button zum Starten der Suchen rechts daneben angeordnet ist. In der hebräischen Version (siehe Abbildung 3) deren gesamtes Layout von rechts nach links angeordnet ist, befindet sich auch der Button links neben dem Eingabefeld. Dasselbe gilt auch für Radiogroups und sonstige Formularelemente.

Beim Design für eine mehrsprachige Applikation muss auch beachtet werden, dass sich bei Übersetzungen der Platzbedarf des Textes verändert. Dieses Phänomen wird als Text Expansion bzw. im umgekehrten Fall als Text Contraction bezeichnet [Yun02]. Die Übersetzung von Englisch in Deutsch benötigt durchschnittlich etwa 35% mehr Platz, manche Einzelworte auch wesentlich mehr, die Übersetzung in Spanisch durchschnittlich 15% [Yun02].

Abbildung 2: Webseite der Stadt Jerusalem - Englische Sprachversion

Abbildung 3: Webseite der Stadt Jerusalem - Hebräische Sprachversion

2.4.3 Templates und Cascading Stylesheets

Für die Behandlung der oben beschriebenen Probleme können und werden 2 Technologien herangezogen, die breite Anwendung in der Entwicklung von Web Applikationen finden. Zum einen Cascading Stylesheets (CSS), eine Empfehlung des W3C[7], welche eine Trennung von Inhalt und Formatierungsangaben ermöglicht. Über Stylesheets kann eine Vielzahl an Formatierungen zentral definiert werden und dann auf ein XHTML/HTML Dokument angewendet werden. Die gebotenen Möglichkeiten sind sehr weitreichend. Beginnend bei der Farbgestaltung, Hintergrundbilder über Positionierung und Sichtbarkeit von Elementen hin zur Gestaltung von Tabellen kann per CSS beeinflusst und vorgegeben werden.

Die zweite Technologie die zur Anwendung kommt sind Templates. Templates stellen dabei Vorlagen dar, die im Falle von Webapplikationen die statischen Elemente des auszugebenden Formates (z.B. HTML, XML, WML) ergänzt durch Platzhalter für dynamische Inhalte enthalten. Bei der Verarbeitung werden die Templates von der jeweiligen Template Engine eingelesen, geparst und die Platzhalter durch den vorgesehenen Inhalt ersetzt, um schlussendlich ein Dokument zu generieren, welches an den Client übermittelt wird. Viele Webapplication Frameworks bringen entweder eine Template Engine mit oder sehen die Verwendung einer bestehenden externen Lösung vor.

Die beiden Technologien können in Kombination für Zwecke der Internationalisierung und Lokalisierung eingesetzt werden. Zum einen können für unterschiedliche Locales eigene Stylesheets definiert werden und über die Template Engine bei der Generierung des Ausgabedokuments dynamisch nach gewähltem Locale eingesetzt werden. Dabei ist es theoretisch auch möglich, Stylesheets in einer Kaskade einzusetzen, konkurrierende Einstellungen werden vom Browser nach Gewichtung und Reihenfolge ersetzt. Dieser Ansatz wird auch in [Stea04] beschrieben. Weiters besteht bei vielen Frameworks die Möglichkeit, lokalisierte Templates zu definieren, die sich in ihrem Layout auch komplett unterscheiden können. Die Auswahl des zu verwendenden Templates erfolgt dabei meist über eine Erweiterung des Templatenamens um den Locale- oder Regionsbezeichner [Shi04] [WirBau08]. Die lokalisierten Templates sind dabei meist hierarchisch geordnet und werden nach dem Prinzip best-match zurückgeliefert, d.h. wird ein Locale de_AT angefordert, wird zuerst nach einem Template mit der Erweiterung de_AT gesucht, sollte es dieses nicht geben, wird als nächstes nach der Erweiterung de gesucht, ehe das root template verwendet wird.

[7] Das WorldWideWeb Konsortium siehe http://www.w3c.org

Mittels CSS lassen sich auch flexible Layouts gestalten, die sich an den veränderten Platzbedarf bei übersetztem Text anpassen. In [W3Ccss08] wird eine Technik beschrieben, die zum Erstellen einer Anzeigebox mit fester Breite und besonderer Gestaltung für die Titelzeile verwendet werden kann. Dabei wird die Eigenschaft von CSS ausgenützt, dass vom Hintergrundbild eines Elements nur der Teil zu sehen ist, der der Platzbedarf des Elements ist. Hat man nun eine oben genannte Anzeigenbox mit festgelegter Breite, kann man das Hintergrundbild der Titelzeile höher erstellen und somit wird beim Umbrechen von längerem Text einfach etwas mehr vom Hintergrundbild sichtbar und das grundlegende Design bleibt erhalten. Ein ähnliches Vorgehen, dass als "Sliding Doors of CSS" von Douglas Bowman [Bow03] veröffentlicht wurde, beruht auf demselben Prinzip, wird in diesem Fall aber dazu verwendet, Menüleisten im Karteireiterstil flexibler zu gestalten. Von der Grafik, die die Darstellung des Karteireiters enthält, wird links ein schmaler Streifen abgetrennt. Der Menüpunkt muss so angelegt werden, dass er aus zwei HTML Elementen besteht, da auf einem Menüpunkt eigentlich immer ein Hyperlink definiert ist, ist das kein Problem. Beim äußeren Element wird nun der schmale Streifen als Hintergrund verwendet. Das innere Element erhält die restliche Grafik als Hintergrund. Wird nun der Platzbedarf des Menüpunktes größer, wird auch mehr vom Rest der Grafik sichtbar und der Karteireiter wird einfach breiter. Die Grafik muss mindestens so breit wie die längste Übersetzung sein, wenn man die Möglichkeit berücksichtigt, dass die BenutzerInnen im Web-Browser die Möglichkeit haben, Schriftgröße oder Zoom der Seite zu erhöhen, sollte auch dies einkalkuliert werden.

2.4.4 Kultur

Ein wichtiger Faktor bei der Internationalisierung von Software ist die Kultur bzw. die höchst unterschiedlichen Kulturen der jeweiligen Zielgruppen weltweit. Kulturelle und regionale Eigenheiten spielen allerdings nicht nur bei den bisher besprochenen Teilaspekten in Form von unterschiedlicher Sprache und Schrift, Formatierungskonventionen, Maßeinheiten, Farbgestaltung und Layout der Benutzerschnittstelle eine Rolle, sondern auch in weniger formal greifbaren Anforderungen, wie der erwarteten Information oder erwünschten Funktionen. Zunächst stellt sich natürlich die Frage: was ist Kultur? und was alles umfasst der Begriff Kultur? In [Hofs05] wird Kultur als ein kollektives Phänomen, welches erlernt und nicht angeboren ist, bestehend aus Werten, Ritualen, Helden und Symbolen beschrieben. Den Kern bilden hierbei die Werte. Sie werden in den ersten Lebensjahren erlernt und betreffen grundlegende Tendenzen, z.B. was wird als gut oder böse, moralisch oder unmoralisch, verboten oder erlaubt, etc. eingestuft. Rituale, Hel-

den und Symbole werden unter dem Begriff Praktiken zusammengefasst und sind nach außen hin sichtbar, obwohl deren kulturelle Bedeutung meist nur Eingeweihten wirklich bekannt ist. In den ersten Jahren dominiert das Erlernen von Werten, mit zunehmendem Alter überwiegt dann allerdings das Aneignen von Praktiken.

Aufgrund der Zugehörigkeit jedes Einzelnen zu mehreren Gruppen ergeben sich verschiedene Ebenen der Kultur, deren Werte durchaus miteinander in Konflikt stehen können [Hofs05].

- Nationale Ebene durch die Zugehörigkeit zu einem Land bzw. einer Nation
- Regionale/ethnische/religiöse/sprachbedingte Ebene durch Zugehörigkeit zu ebensolchen Gruppen
- Geschlechtsspezifische Ebene durch die Geburt als Bub oder Mädchen
- Generationsebene durch Zugehörigkeit zur Gruppe der Jugendlichen, Eltern oder Großeltern
- Soziale Ebene bedingt durch Bildungsniveau und Beschäftigung
- Ebene der Arbeitsorganisation durch Zugehörigkeit zu einer Abteilung oder Firma

Durch den technologischen Fortschritt verändert sich unsere Welt. Die Anwendung neuer Technologien im Alltag verändert auch die oben bereits erwähnten Praktiken, also die nach außen sichtbare Repräsentation der Kultur. Der Kern der Kultur also die Werte verändert sich allerdings nur langsam. Dies wird in [Hofs05] damit begründet, dass diese Werte in der Kindheit erlernt werden und zwar von den Eltern, die diese Werte wiederum als Kinder erlernt haben. Aufgrund der Stabilität der kulturellen Werte bieten sich diese als Ansatzpunkt an, wenn man einen quantitativen Vergleich von Kulturen durchführen möchte. Geert Hofstede hat bei seiner Studie von Umfragedaten von IBM MitarbeiterInnen in 53 Ländern bezüglich ihrer Werte zuerst 4 kulturelle Dimensionen identifiziert. Später wurde durch Vergleich mit einer anderen Studie, die ebenfalls 4 Dimensionen enthielt von denen allerdings nur 3 mit den von Hofstede gefundenen Kriterien korrelierten eine 5te Dimension eingeführt.

Kulturelle Dimensionen

Die 53 in der IBM Studie betrachteten Länder wurden auf einer Skala von 1 bis 100 nach den 5 Dimensionen - Power-Distance, Collectivism vs. Individualism, Masculinity vs. Femininity, Uncertainty Avoidance und Short-term vs. Long-term Orientation bewertet [Hofs05]. In der Folge werden diese Dimensionen kurz erläutert und einige exemplarische Charakteristika angeführt. Um eine bessere Orientierung zu bekommen, wird zu jeder Dimension auszugsweise einige Länder und deren Werte wie sie in [Hofs05] publiziert wurden angeführt.

Power-Distance

Diese Dimension betrifft den Umgang mit der ungleichen Verteilung von Macht. Der Wert ist ein Indikator, inwiefern wenig mächtige Mitglieder der betrachteten Kultur es erwarten und akzeptieren, dass Macht ungleich verteilt ist. Hofstede beschreibt, dass in Gesellschaften mit hoher Power-Distance Ungleichheiten zwischen Personen erwartet und erwünscht werden, wohingegen bei geringer Power-Distance Ungleichheiten möglichst minimiert werden sollen. Respekt gegenüber Älteren und Höherstehenden Personen ist in Gesellschaften mit hoher Power-Distance sehr wichtig. Auch organisatorisch sind Unterschiede feststellbar, während auf der eine Seite hierarchische Strukturen und Zentralisierung bevorzugt werden (hohe Power-Distance), sind dies auf der anderen Seite flachere Strukturen und Dezentralisierung (geringe Power-Distance). Generell kann gesagt werden, dass einige Eigenschaften von Ländern mit dem Power-Distance Index korrelieren, so z.B. hängt die geographische Breite (höhere geographische Breite - höhere Power-Distance), die Bevölkungsgröße (größere Bevölkerung - höhere Power-Distance) und der Wohlstand (höherer Wohlstand - geringer Power-Distance) zusammen.

Land	**Ergebnis**	**Platzierung**	**Land**	**Ergebnis**	**Platzierung**
Malaysien	104	1-2	Spanien	57	45-46
Philippinen	94	5	USA	40	57-59
Frankreich	70	22-25	Österreich	11	74

Anm.: ein hoher Wert bedeutet hohe Power-Distance

Tabelle 17: Power-Distance Beispiele [Hofs05]

Collectivism/Individualism

Hier wird das Spannungsfeld zwischen Interessen und Bedürfnissen des Einzelnen und

denen der Gruppe behandelt. Eine kollektivistische Gesellschaft ist charakterisiert durch ein hohes Maß an Gruppenzusammengehörigkeit und hoher Wertschätzung von Loyalität. Weitere Merkmale sind z.B. größere und weitreichendere Familien, vorhandene Ressourcen werden mit Verwandten geteilt, Informationen werden vorwiegend aus dem sozialen Netzwerk bezogen oder kollektive Interessen überwiegen gegenüber individuellen Interessen. Bei individualistischen Gesellschaften dominiert die Kleinfamilie (oft nur Eltern und 1 Kind), Ressourcen werden individuell besessen (schon im Kindesalter), die Hauptinformationsquelle sind Medien und individuelle Interessen überwiegen kollektive Interessen. Die Positionierung eines Landes kann wiederum aus gewissen Eigenschaften relativ genau vorhergesagt werden, zum einen aufgrund des Wohlstands (höherer Wohlstand höhere Individualität) und aufgrund der geographischen Breite (in der Nähe des Äquator geringer Individualität).

Land	**Ergebnis**	**Platzierung**	**Land**	**Ergebnis**	**Platzierung**
USA	91	1	Japan	46	33-35
Niederlande	80	4-6	Südkorea	18	63
Österreich	55	27	Guatemala	6	74

Anm.: ein hoher Wert bedeutet Individualism

Tabelle 18: Collectivism/Individualism Beispiele [Hofs05]

Masculinity/Femininity

In maskulinen Kulturen spielen Werte wie Durchsetzungsfähigkeit, Konkurrenz und Erfolg eine große Rolle, wohingegen in femininen Kultur das Sorgen für Andere, Lebensqualität und das Fördern und Pflegen eher im Vordergrund stehen. Die Rollen von Eltern sind in maskulinen Gesellschaften aufgeteilt, sodass Väter für Fakten und Mütter für Gefühle zuständig sind. Allerdings erstreckt sich die Rollenunterscheidung auch über viele andere Bereiche z.B. Bildung oder das Einkaufsverhalten. Viele Lehrkräfte in maskulinen Gesellschaften im Pflichtschulbereich sind paradoxerweise weiblich, haben aber meist einen geringeren Stellenwert und sind deshalb kaum Vorbilder. Auch bei der Wahl des Studiums gibt er meist “typische” Männer und “typische” Frauenstudien. Diese Aufteilung setzt sich beim Einkaufsverhalten fort und kann als „Männer kaufen Autos Frauen kaufen Lebensmittel“ zusammengefasst werden. Diese Aufteilungen gibt es in femininen Gesellschaften nicht oder nicht in dieser Ausprägung. Weiters gibt es eine Tendenz zur Selbstüberschätzung in Maskulinen und Selbstunterschätzung in femininen Kulturen.

Das Besondere dieser Dimension ist, dass hierfür in der IBM Studie Männer und Frauen konsistent unterschiedliche Werte erreichten mit der Ausnahme, dass dies für Länder, die als extrem feminin bewertet wurden, nicht zutrifft. Eine weitere Besonderheit ist, dass sich die Ausrichtung auf dieser Dimension mit dem Alter verändert. Erzielen jüngere Menschen noch Ergebnisse, die auf der maskulinen Hälfte der Skala liegen so verändert sich dieser Wert mit zunehmenden Alter in die feminine Hälfte und auch der Unterschied zwischen Männern und Frauen ist altersabhängig und besteht nur bis zu einem Alter von etwa 50.

Land	**Ergebnis**	**Platzierung**	**Land**	**Ergebnis**	**Platzierung**
Japan	95	2	Frankreich	43	47-50
Österreich	79	4	Uruguay	38	60
USA	62	19	Schweden	5	74

Anm.: ein hoher Wert bedeutet Masculinity

Tabelle 19: Masculinity/Femininity Beispiele [Hofs05]

Uncertainty Avoidance

Obwohl Unsicherheit und Mehrdeutigkeiten ohne Zweifel Bestandteil der menschlichen Existenz sind, gibt es doch unterschiedliche Arten, wie Kulturen damit umgehen. Dieser Umgang wird bewertet und ergibt eine weitere kulturelle Dimension. Kulturen, die einen hohen Uncertainty Avoidance Index (UAI) haben, sind bestrebt, eine gewisse Vorhersagbarkeit zu erreichen, und tendieren in die Richtung Recht und Ordnung. Charakteristisch ist hoher Stress auch im Familienleben und strenge Regeln etwa für das Verhalten von Kindern aber auch eine Tendenz zu Formalismen und Präzision. Ein niedriger UAI zeigt eine höhere Bereitschaft zum Eingehen von Risiken und eine höhere Akzeptanz von existierender Ungewissheit. Dies äußert sich z.B. in der Art und der Menge an Gesetzen, die in einem Land existieren. In Ländern mit niedrigem UAI sind diese meist sehr allgemein gehalten und in geringerer Zahl vorhanden. Auch der Umgang mit anderen ethnischen Gruppen schlägt sich in dieser Dimension nieder, auf Seiten eines hohen UAI mit verstärkten Vorurteilen auf der anderen Seite durch erhöhte Toleranz.

Short-term/Long-term Orientation

Long-term Orientation bedeutet, dass in der Regel Tugenden wie Beharrlichkeit und

Land	Ergebnis	Platzierung	Land	Ergebnis	Platzierung
Griechenland	112	1	Niederlande	53	53
Japan	92	11-13	USA	46	62
Österreich	70	35-38	Singapur	8	74

Anm.: ein hoher Wert bedeutet starke Uncertainty Avoidence

Tabelle 20: Uncertainty Avoidence Beispiele [Hofs05]

Wirtschaftlichkeit hohen Stellenwert genießen und mit der Ausrichtung auf zukünftigen Honorierung einhergehen. Auf der anderen Seite steht Short-term Orientation für eine Ausrichtung auf die Vergangenheit und die Gegenwart. Bevorzugte Tugenden sind der Respekt für Tradition und die Erfüllung sozialer Pflichten. In wirtschaftliche Sinne zählt in short-term orientierten Gesellschaften der Profit dieses Jahres in long-term orientierten der Profit in 10 Jahren.

Land	Ergebnis	Platzierung	Land	Ergebnis	Platzierung
China	118	1	Deutschland	31	25-27
Japan	80	4-5	USA	29	31
Niederlande	44	13-14	Pakistan	0	39

Anm.: ein hoher Wert bedeutet long-term orientation

Tabelle 21: Short-term/Long-Term Orientation Beispiele [Hofs05]

Die Arbeit von Geert Hofstede ist die Basis für viele weitere Forschungen und Arbeiten, einige davon beschäftigen sich auch mit dem Einfluss von kulturellen Unterschieden auf User-Interface Design und auch Web Design. In [SinPer05] wird basierend auf 4 der von Geert Hofstede erforschten Dimensionen (Power-Distance, Uncertainty Avoidance, Collectivism/Idividualism und Masculinity/Femininity) und einer zusätzlichen Dimension Low/High Context, welcher auf der Arbeit von E.T. Hall begründet ist, Richtlinien für kulturell angepasste Webseiten zu erstellen. Die zusätzlich verwendete Dimension Low-High Context beschreibt wie die Kommunikation zwischen den Angehörigen der jeweiligen Kultur beschaffen ist. High-Context Kulturen weisen ein hohes Maß an nonverbaler und durch Symbole geprägten Kommunikation auf, deren Bedeutung sich aus dem jeweiligen Kontext ergibt, den Angehöriger einer Gruppe verstehen. In Low-Context Kulturen herrscht verbale Kommunikation vor, die sehr rational und explizit ist.

Der in [SinPer05] verfolgte Ansatz war, auf Basis vorangegangener Arbeiten, Features

von Webseiten den kulturellen Dimensionen zuzuordnen und anschließend zu verifizieren, ob in 4 ausgewählten Regionen (China, Japan, Indien und USA) erfolgreiche Webseiten diese Merkmale aufweisen. Dies wurde in der durchgeführten Studie auch belegt, wobei angemerkt wurde, dass die alleinige Existenz der Kriterien nicht für den Erfolg garantiert. Im folgenden werden die in [SinPer05] dargelegten Richtlinien kurz zusammengefasst:

Um etwa *Collectivism* zu betonen, könnten unter anderem folgende Elemente forciert bzw. angeboten werden. Ein Newsletter, Chat Rooms wo einerseits BenutzerInnen untereinander oder eventuell auch mit Angestellten des Unternehmens Kontakt aufnehmen können. Symbole und Bilder, die die nationale Identität ansprechen, wie die Landesflagge oder Bilder von nationalen Wahrzeichen. Auch Links zu lokalen Partner oder anderen lokalen Webseiten könnten präsentiert werden. Das Thema Familie zu unterstreichen durch Darstellung der Kunden als eine Familie.

Stehen *individualistische Zielgruppen* im Vordergrund sollte eine gute Datenschutzerklärung und Features zur Personalisierung angeboten werden. Das Unterstreichen der Einzigartigkeit des Produktes und das Herstellen eines Bezuges zu Unabhängigkeit durch Bilder und Themen kann ebenfalls zu besserer Akzeptanz beitragen.

Soll einer hohen *Uncertainty Avoidence* begegnet werden, kann dies mit Kundenservice in Form von FAQs, Kundeservice Kontaktdaten durch herunterladbare Testversion oder kostenlosen Telefonsupport erfolgen. Weiters kann die Verwendung von lokale Terminologien oder das Erwähnen von lokalen Geschäften oder Niederlassungen sich positiv auswirken.

Zeichnet sich die Zielgruppe durch hohe *Power-Distance* aus, so kann die Präsentation der Unternehmenshierarchie oder Bilder der Führungspersönlichkeiten das Mittel der Wahl sein. Auch der Hinweis auf Auszeichnungen oder Testberichte wäre möglich.

Maskulinität wird durch Betonung von Realismus durch das Design oder der Effektivität des Produktes angesprochen. Auch das Anbieten von Quiz oder anderen Spielen ist möglich. Ebenfalls wichtig ist die Betonung von geschlechterspezifischen Rollenbildern.

Die Maßnahmen, um *feminine Kulturen* anzusprechen, entsprechen denen, die für *High-Context* Kulturen verwendbar sind. Diese sind Ästhetik im Design eine unaufdringliche Verkaufsstrategie und ein höflicher und indirekter Stil.

Im Hinblick auf *Low-Context* Kulturen ist eine aggressive Verkaufsstrategie und die Verwendung von Superlativen zu empfehlen. Weiters sollte der Rang und das Prestige des Unternehmens betont werden und Bedingungen für den Kauf auffällig platziert werden.

Die Empfehlungen sind sicher nicht eins-zu-eins auf jeden Typ von Webapplikation übertragbar, veranschaulichen allerdings, dass sich die Erwartungen an Produkte zwischen einzelnen Kulturen stark unterscheiden. Das Miteinbeziehen kultureller Überlegungen abseits von Formatrichtlinien und Ähnlichem in den Entwicklungs- bzw. Designprozess ist sicherlich überlegenswert.

2.5 Stufen und Konzepte der Internationalisierung

Nachdem in den vorigen Abschnitten großteils versucht wurde, die auftretenden Probleme bei der Entwicklung von Software für einen internationalen Einsatz darzustellen, sollen nun die grundlegenden Konzepte vorgestellt werden, die dazu benutzt werden, mit diesen Schwierigkeiten besser umzugehen.

2.5.1 Konzepte I18n

Aus den zu den bisher beschriebenen Lösungsansätzen zu den einzelnen Internationalisierungsaufgaben lassen sich die folgenden drei grundlegenden Konzepte ableiten. Man könnte eventuell auch einen allgemeinen Zeichensatz, wie Unicode, als ein solches Konzept aufführen, da dies zum einen eine Voraussetzung für die gleichzeitige Darstellung mehrsprachiger Inhalte in Webapplikationen darstellt und zum anderen bei der internen Verarbeitung von Texten in unterschiedlichen Sprachen bzw. Schriftsystemen eine wesentliche Vereinfachung der Verarbeitung und auch Speicherung bedeutet.

- Locale
- Externalisierung - Trennung von zu lokalisierenden und nicht zu lokalisierenden Elementen
- locale-sensitive Funktionen und Objekte

Locale

Der Begriff des Locales wurde schon am Anfang dieser Arbeit definiert. Das Locale stellt allerdings ein sehr wichtiges und zentrales Konzept der Internationalisierung dar. Ein Locale definiert eine geographische, politische oder kulturelle Region. In der Regel besteht ein Locale aus einem Sprachcode und einem Ländercode. Für die Identifikation der Sprachen werden zweistellige Sprachcodes, wie sie die ISO-639 Norm definiert, und für die Ländercodes zwei- oder dreistellige Codes der ISO-3166 Norm verwendet.

Sprache	Land	Sprachcode	Ländercode	ISO3-Code	Locale Bezeichner
deutsch	Österreich	de	AT	deu AUT	de_AT

Tabelle 22: Localebezeichner mit Sprach- und Ländercode (exemplarisch)

Vorgesehen ist auch noch die weitere Unterscheidung durch sogenannte Variants, welche allerdings nicht näher bestimmt sind und daher quasi frei verwendbar sind. Denkbar wäre z.B. eine Unterscheidung nach Bundesstaaten, ethnischer Minderheiten in der

definierten Region oder auch installiertem Betriebssystem. Daraus ergibt sich die Anwendungen des Locales als Identifier der auszuführenden Applikationsvariante. Diese Funktion des Locales wird auch in den beiden nachfolgenden Konzepten verwendet.

Da ein Locale an sich nur eine Kombination aus Sprachcode und Ländercode ist, benötigt man, um die regionalen Unterschiede auf der technischen Ebene zu behandeln, Informationen. Diese werden Locale-Daten genannt und stellen eine Sammlung von locale-spezifischen Informationen dar [RhDaLo05]. Für jedes unterstützte Locale existiert eine Menge an Informationen über die definierte Region. Dies betrifft einerseits Angaben zur Formatierung von Datums- und Zeitangaben, Zahlen und Währung aber auch Übersetzungen von Ländernamen, Wochentagen, Monatsnamen und Währungsbezeichnungen in die jeweilige Sprache des Locales. Auch Vorgaben zur Sortierung sind enthalten. Diese Informationen werden von locale-sensitiven Funktionen und Objekten benützt, um deren Verhalten an das jeweils gewählte Locale anzupassen.

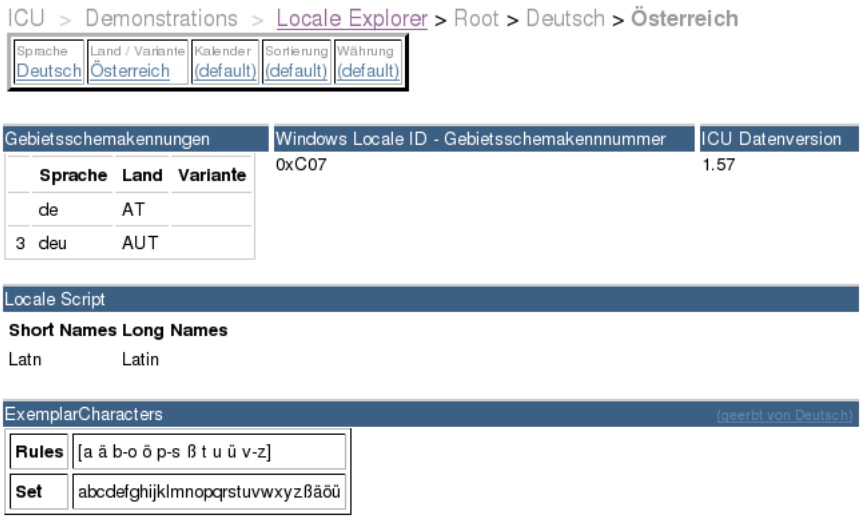
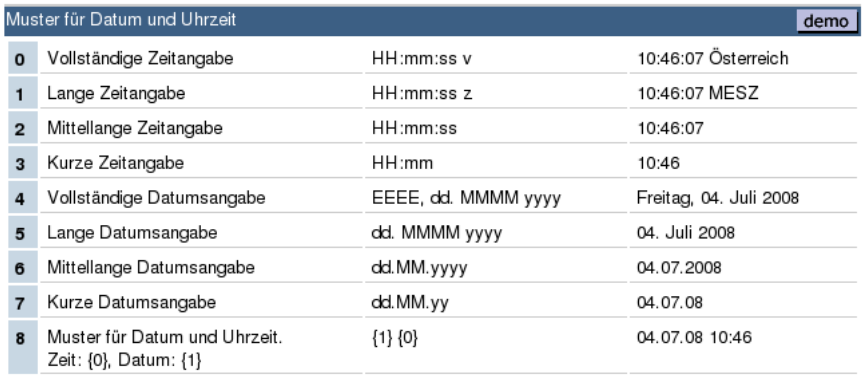


Abbildung 4: de AT Locale Bezeichnung und Grundinformationen [ICU08]

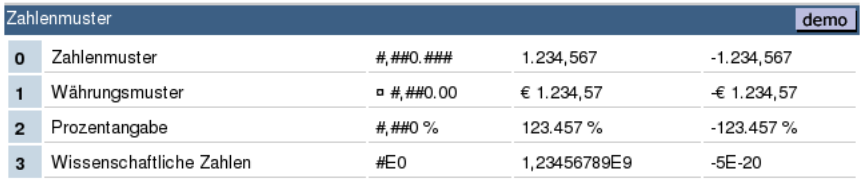
Die Abbildungen 4, 5 und 6 zeigen jene Locale-Daten für das Locale de_AT (also Deutsch für Österreich), die vom ICU Projekt [ICU08] für deren Internationalisierungsbibliothek Verwendung finden. Einige dieser Locale-Daten des ICU Projektes stammen aus einem von Unicode Consortium betreuten Projekt dem Common Locale Data Repository [CLDR08]. Das Ziel dieses Projektes ist es, Locale-Daten zu allen Weltsprachen zu sammeln und frei verfügbar zu machen. Dazu verwendet das Projekt ein XML Format, die LDML (Locale Data Markup Language). Die enthaltenen Daten umfassen Formatierungsangaben für Datum und Zeit, Zahlen und Währungen. Weiters Maßein-



Muster für Datum und Uhrzeit — demo

0	Vollständige Zeitangabe	HH:mm:ss v	10:46:07 Österreich
1	Lange Zeitangabe	HH:mm:ss z	10:46:07 MESZ
2	Mittellange Zeitangabe	HH:mm:ss	10:46:07
3	Kurze Zeitangabe	HH:mm	10:46
4	Vollständige Datumsangabe	EEEE, dd. MMMM yyyy	Freitag, 04. Juli 2008
5	Lange Datumsangabe	dd. MMMM yyyy	04. Juli 2008
6	Mittellange Datumsangabe	dd.MM.yyyy	04.07.2008
7	Kurze Datumsangabe	dd.MM.yy	04.07.08
8	Muster für Datum und Uhrzeit. Zeit: {0}, Datum: {1}	{1} {0}	04.07.08 10:46

Abbildung 5: de AT Locale Datumsformate [ICU08]



Zahlenmuster — demo

0	Zahlenmuster	#,##0.###	1.234,567	-1.234,567
1	Währungsmuster	¤ #,##0.00	€ 1.234,57	-€ 1.234,57
2	Prozentangabe	#,##0 %	123.457 %	-123.457 %
3	Wissenschaftliche Zahlen	#E0	1,23456789E9	-5E-20

Abbildung 6: de AT Locale Zahlenformate [ICU08]

ICU > Demonstrations > Locale Explorer > Root > Englisch > **Vereinigte Staaten**

Sprache: Englisch | Land / Variante: Vereinigte Staaten | Kalender: (default) | Sortierung: (default) | Währung: (default)

Speziellere Gebietsschemata:

Vereinigte Staaten (Posix)

Vereinigte Staaten under other languages: Spanisch, Hawaiianisch

Gebietsschemakennungen

	Sprache	Land	Variante
	en	US	
3	eng	USA	

Windows Locale ID - Gebietsschemakennnummer: 0x409

ICU Datenversion: 1.51

Locale Script

Short Names	Long Names
Latn	Latin

ExemplarCharacters (geerbt von Englisch)

Rules	[a-z]
Set	abcdefghijklmnopqrstuvwxyz

Abbildung 7: en US Locale Bezeichnung und Grundinformationen [ICU08]

Muster für Datum und Uhrzeit (geerbt von Englisch) demo

0	Vollständige Zeitangabe	h:mm:ss a v	11:35:53 AM Austria Time
1	Lange Zeitangabe	h:mm:ss a z	11:35:53 AM GMT+02:00
2	Mittellange Zeitangabe	h:mm:ss a	11:35:53 AM
3	Kurze Zeitangabe	h:mm a	11:35 AM
4	Vollständige Datumsangabe	EEEE, MMMM d, yyyy	Friday, July 4, 2008
5	Lange Datumsangabe	MMMM d, yyyy	July 4, 2008
6	Mittellange Datumsangabe	MMM d, yyyy	Jul 4, 2008
7	Kurze Datumsangabe	M/d/yy	7/4/08
8	Muster für Datum und Uhrzeit. Zeit: {0}, Datum: {1}	{1} {0}	7/4/08 11:35 AM

Abbildung 8: en US Locale Datumsformate [ICU08]

Zahlenmuster (geerbt von Englisch) demo

0	Zahlenmuster	#,##0.###	1,234.567	-1,234.567
1	Währungsmuster	¤#,##0.00;(¤#,##0.00)	$1,234.57	($1,234.57)
2	Prozentangabe	#,##0%	123,457%	-123,457%
3	Wissenschaftliche Zahlen	#E0	1.23456789E9	-5E-20

Abbildung 9: en US Locale Zahlenformate [ICU08]

heiten, Besonderheiten beim Textvergleich für Sortierung und Suchen und Übersetzungen für Ländernamen, Währungsbezeichnungen, Schriftsysteme, Zeitzonen und Sprachen [CLDR08].

Externalisierung

Eine grundsätzliche Bestrebung bei der Entwicklung internationalisierter Applikationen ist es, ohne Neukompilierung zusätzliche Sprachen/Locales zu unterstützen. Um dieses Ziel zu erreichen, müssen diejenigen Teile der Produkts, welche zu lokalisieren sind, vom restlichen Source Code getrennt werden, und es muss ein Mechanismus existieren, um diese ausgelagerten Elemente der Software je nach gewähltem Locale während der Laufzeit einzubinden. Dieses Konzept wird als Externalisierung bezeichnet. Davon betroffen können verschiedenste Arten von Ressourcen sein. Angefangen bei einem String, der eine simple Grußformel enthält, über komplexere Nachrichten mit erst zur Laufzeit feststehenden Teilen bis hin zu Bildern, Sounds oder Videos.

Locale-sensitive Funktionen und Objekte

Um die in den Locales vorhandenen regionsspezifischen Informationen zu nutzen, findet ein weiteres Konzept Anwendung. Locale-sensitive Funktionen und Objekte wurden be-

reits in Abschnitt 2.3 als Methode zur Formatierung und Eingabevalidierung vorgestellt. Sie erhalten in der Regel ein Locale als Parameter, je nach Ausführung das Locale der Laufzeitumgebung oder ein zur Laufzeit bestimmbares, und passen ihre Funktionsweise an die Vorgaben des Locales an. Wesentliche Voraussetzung ist das Vorhandensein von Locale-Daten für die jeweils unterstützten Locales.

2.5.2 Stufen der I18n

Um die möglichen Ausbaustufen der Internationalisierung von Webapplikationen zu beurteilen, ist es nötig, eine Kategorisierung einzuführen und zu definieren, welche Eigenschaften eine Webapplikation der jeweiligen Kategorie aufweisen sollte. In Anlehnung an die in [SinPer05] verwendeten Kategorien der Lokalisierung von Websites und unter Einbeziehung der Ausführungen in [Stea04] werden folgende 4 Kategorien mit ihren Eigenschaften festgelegt, da bei der Recherche der relevanten Literatur keine existierende Kategorisierung von Webapplikationen hinsichtlich ihres Internationalisierungsgrades zu finden war.

- Standardisierte Webapplikation
- Lokalisierte Webapplikation
- Internationalisierte Webapplikation
- Kulturell angepasste Webapplikation

Standardisierte Webapplikation

Applikationen dieser Kategorie werden ohne jegliche Berücksichtigung von Themen der Internationalisierung entwickelt. Dementsprechend sind auch keine Besonderheiten bezüglich des Entwicklungsablaufes wie das Einplanen der Durchführung von Übersetzungen durch externe Unternehmen. Es sind mehrere Szenarien denkbar, weshalb eine Webapplikation ohne Berücksichtigung von Internationalisierung entwickelt wird. Es könnt etwa der Einsatz in einem Intranet oder Extranet mit bekannter Benutzergruppe und unternehmensweiten standardisierten Formaten vorgesehen sein, sodass für eine Internationalisierung keine Notwendigkeit besteht. Auch wirtschaftliche Überlegungen, dass der Aufwand zu hoch gegenüber den zu erwartenden Return-of-Invest oder technische Gründe, wie kurze Produktzyklen, die durch den Mehraufwand an Übersetzung und Testphasen nicht mehr eingehalten werden könnten, sind als Argumente denkbar.

Lokalisierte Webapplikation

In diesem Fall wird die Applikation an ein oder mehrere bestimmte Ziellocales angepasst, allerdings ist es kein Ziel der Entwicklung, die Anwendung so zu programmieren, dass eine Anpassung an weitere Locales möglichst einfach und ohne Neukompilierung erfolgen kann. Die Abstimmung auf das beabsichtigte Ziellocale ist im Allgemeinen gut, da meistens die EntwicklerInnen aus dem jeweiligen Gebiet stammen. Es kommen keine Konzepte der Internationalisierung zum Einsatz. Benötigte Muster für die Ein- und Ausgabe von Daten sind im Sourcecode direkt enthalten. Falls die Entwicklung mit einem Webapplication Framework erfolgt, wird möglicherweise dessen Internationalisierungsfähigkeit benutzt, um frameworkeigene Fehlermeldungen, die ausgegeben werden (etwa Fehlermeldungen bei der Validierung von Eingabedaten) an das Ziellocale anzugleichen. Für die Unterstützung mehrerer Locales wird jeweils eine eigens adaptierte Version der Applikation erstellt und gewartet.

Internationalisierte Webapplikation

Eine Applikation wird so entwickelt oder re-designed, dass sie möglichst einfach verschiedene Locales unterstützt. Es finden verstärkt Techniken und Konzepte der Internationalisierung Anwendung. Zu lokalisierende Elemente werden während des Designs und der Entwicklung identifiziert und die Konzepte der Internationalisierung so eingesetzt, dass eine nachträgliche Lokalisierung für andere Locales mit möglichst geringem Aufwand erfolgen kann. Das Ergebnis sollte ein einziges ausführbares Programm sein, welches keine locale-abhängigen Eigenheiten aufweist. Locale-spezifische Inhalte sind externalisiert und können über festgelegte Methode bzw. ein definiertes Format eingebunden und erweitert werden. Ausgaben und Benutzereingaben werden je nach Locale mittels angepasster Funktion behandelt.

Kulturell angepasste Webapplikation

Dieser Typ stellt eine Erweiterung der internationalisierten Webapplikation dar, die für jede unterstützte Kultur eine eigens angepasste Benutzerschnittstelle und/oder Funktionsumfang vorsieht. Das Thema kulturelle Anpassung ist schon bei der Entwicklung oder bei einem nachträglichen umfassenden Re-Design ein zentraler Punkt. Die Adaptierungen der Benutzerschnittstelle reichen von angepassten Farbprofilen und Grafiken sowie Hintergrundbildern, über ethnisch und kulturell passende Modelle bei der Abbildung von Menschen bis zur Bereitstellung zusätzlicher Funktionen, etwa nach den Ergebnissen von Nitish Singh & Arun Pereira, die im vorigen Abschnitt beschrieben wurden.

2.6 Voraussetzungen und Auswirkungen der Internationalisierung

Die Entwicklung internationalisierter Applikation im Allgemeinen und Webapplikationen im Speziellen erfordert es, dass einige Voraussetzungen quer durch die Applikation erfüllt sind. In diesem Abschnitt werden diese Voraussetzungen beschrieben aber auch die Auswirkungen auf den Entwicklungsprozess. Im Zuge der Ausführungen werden auch einige Anwendungsfälle beschrieben, die bei internationalisierten Webanwendungen typischerweise auftreten und es wird abschließend auch auf das Thema Übersetzung und Testen von internationalisierten Webapplikationen eingegangen.

2.6.1 Voraussetzungen

Wie im Abschnitt Grundlagen - Web Applikation beschrieben kommen unterschiedliche Serverprozesse zum Einsatz bzw. ergeben sich unterschiedliche Schichten in der Architektur einer Applikation. Auch in [Sach08] wird eine Erstreckung der Internationalisierung über alle Ebenen der Anwendung beschrieben und in Abbildung 10 schematisch dargestellt. In den einzelnen Schichten ergeben sich unterschiedliche Aspekte, die bei der Internationalisierung der Applikation zu berücksichtigen sind.

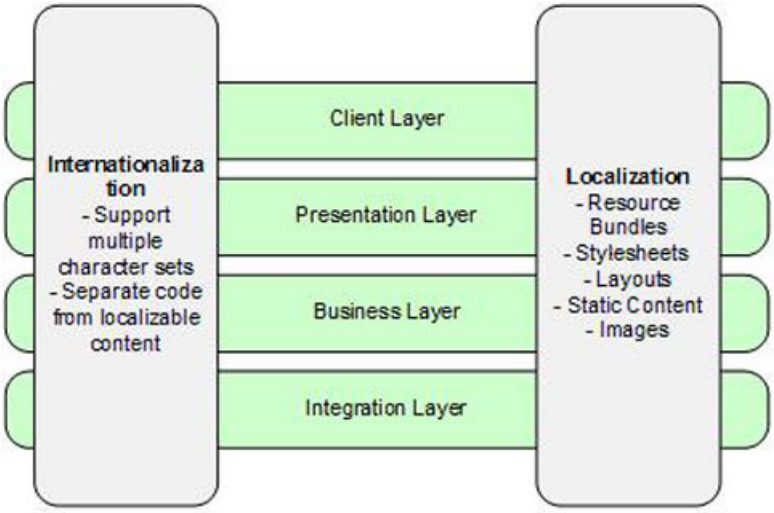


Abbildung 10: Erstreckung der i18n über Applikationsschichten [Sach08]

Database Layer

Je nachdem, in welcher Sprache die in der Datenbank abzulegenden, Daten vorliegen, ist ein geeigneter Zeichensatz zu verwenden. Sollen unterschiedliche Sprachen verarbeitet werden, wird der Zeichensatz der Wahl in der Regel Unicode sein. Dennoch kommt

es in der Praxis häufig vor, dass bereits bestehende Datenbanken angebunden werden müssen. In diesem Fall besteht die Möglichkeit, in anderen Zeichensätzen kodierte Daten zu migrieren, also z.B. in Unicode zu konvertieren. Unicode definiert, wie schon im Abschnitt 2.2.4 Unicode erwähnt sog. Compatibility Characters, um ein möglichst weitreichende Konvertierung zu ermöglichen. Weiters werden Tools zur Konvertierung meist von den Datenbankherstellern angeboten. Sollte eine komplette Migration aus irgend welchen Gründen (kein administrativer Zugriff auf die Datenquelle oder Legacy Systeme, die nur mit bestimmten Zeichensätzen umgehen können) nicht machbar sein, kann ein Konverter innerhalb der Applikation implementiert werden, der die abgefragten Daten zur Laufzeit in den gewünschten Zeichensatz umwandelt. Ein weiterer Punkt der zu berücksichtigen ist, sind etwaige in der Datenbank implementierte Prozeduren und Funktion (Stored Procedures), die möglicherweise angepasst werden müssen. Beachtet sollten auch die Sortiermechanismen der Datenbank werden. Bei vielen Datenbanken kann ein Sortierschema nur pro Datenbank festgelegt werden oder es wird im Hinblick auf Unicode der Unicode Collation Algorithm nur teilweise implementiert.

Bei der Modellierung des Datenbankkonzepts können sich abhängig vom Typ der Applikation Aspekte ergeben, die im Hinblick auf die Internationalisierung relevant sind. In vielen Fällen wird man sprach- oder länderspezifischen Content verwalten müssen, der abhängig vom gewählten Locale angezeigt werden soll. Somit muss ein Modell entwickelt werden, wie die gespeicherten Daten nach Sprache, Land oder Locale unterschieden werden können, um die BenutzerInnen nicht mit Daten in unbekannten Sprachen zu konfrontieren. In [Ditt02] wird diese Problematik mehrsprachiger Anwendungsdaten behandelt und zwischen eher statischen Daten, wie Farbkataloge oder Postleitzahlenverzeichnisse und dynamischeren Daten wie z.B. Artikel- oder Warengruppenbezeichnungen. Weiters wird ausgeführt, dass der Ablauf der Übersetzung dieser Daten festgelegt werden muss und dafür 2 Varianten denkbar wären. Der erste Ansatz legt eine default Sprache fest etwa Englisch. Die Daten werden dann zuerst in der default Sprache gepflegt und in definierten Abständen an ein Übersetzungsteam weitergegeben. Solange für gewisse Locales keine Übersetzung vorliegt, wird die default Sprache verwendet. Die zweite Variante legt die Voraussetzung fest, dass Daten erst dann im System verwendet werden können, wenn alle Übersetzungen vorhanden sind. Dazu könnte ein Workflow definiert werden. Das Durchlaufen des Workflows stellt natürlich gegenüber Variante 1 eine Verzögerung dar, andererseits müssen sich die AnwenderInnen nicht bis zu einer Übersetzung mit der default Sprache begnügen.

Business Layer

Ebenso wie auf der Datenbankebene müssen auch die Komponenten der Geschäftsebene mit dem Zeichensatz der zu bearbeitenden Daten umgehen können. Da, wie schon mehrfach erwähnt, Unicode die Basis für viele Internationalisierungs-bestrebungen ist, ist die Unterstützung für Unicode relativ hoch aber noch nicht flächendeckend vorhanden. Manche Programmiersprachen und Programmbibliotheken, wie z.B. Ruby sind nicht oder noch nicht vollständig Unicodefähig und müssen, wenn verfügbar, durch alternative Bibliotheken ergänzt werden [WirBau08]. Weiters sind meist auch auf dieser Ebene Locale-sensitive Funktion in Verwendung etwa zum Umrechnen von Währungen oder der Ermittlung von länderspezifischen Steuerraten. Für deren Anwendung muss das vom aktuellen User gewünschte Locale bekannt sein. Die Feststellung des Locales erfolgt auf der Präsentationsebene und muss von dort weitergereicht werden.

Presentation Layer

Auf der Präsentationsebene findet eine Konzentration der Internationalisierungsaspekte statt, da hier die Interaktion mit dem BenutzerInnen stattfindet und alle präsentierten Daten in einer den BenutzerInnen angepassten Form sein sollten. Hier kommen viele der in den vorherigen Abschnitten beschriebenen Aspekte zum Tragen. Es sollten also Datums- und Zeitangaben, Zahlen, Währungen, Sortierungen usw. in einer dem geforderten Locale angepassten Form aber auch Meldungen des Systems in der gewünschten Sprache erfolgen. Erreicht wird dies in der Regel durch locale-sensitive Objekte und Funktionen, die sowohl für das Formatieren der Ausgaben als auch das Einlesen der Eingaben ausgelegt sind. Für die Behandlung der Systemmeldungen kommen Externalisierungslösungen wie das in Abschnitt 2.3 Formate, Datumsangaben, Einheiten und Systemmeldungen beschrieben GNU gettext System. Bei Webanwendung treten zudem auch einige Unsicherheiten auf.

So z.B. ist die Frage zu klären, woher die Information für das von den BenutzerInnen gewünschte Locale kommt. Grundsätzlich ist es möglich, diese Information aus dem Accept-Language HTTP Header [W3C08], welcher eine Liste an gewünschten Locales bzw. auch nur Sprachcodes enthalten kann, auszulesen. Da diese Vorgabe im jeweiligen Browser einstellbar (siehe Abbildung 11) und normalerweise standardmäßig gleich der Einstellungen des Betriebsystems ist, stimmt sie in vielen Fällen mit den Wünschen der BenutzerInnen überein. Dies muss aber bei weitem nicht immer der Fall sein. Man denke an z.B. die deutschsprachigen BenutzerInnen in einem spanischen Internetcafe oder Ähnliches. Für diese Fälle sollte also in der Applikation eine Möglichkeit zur Sprachauswahl

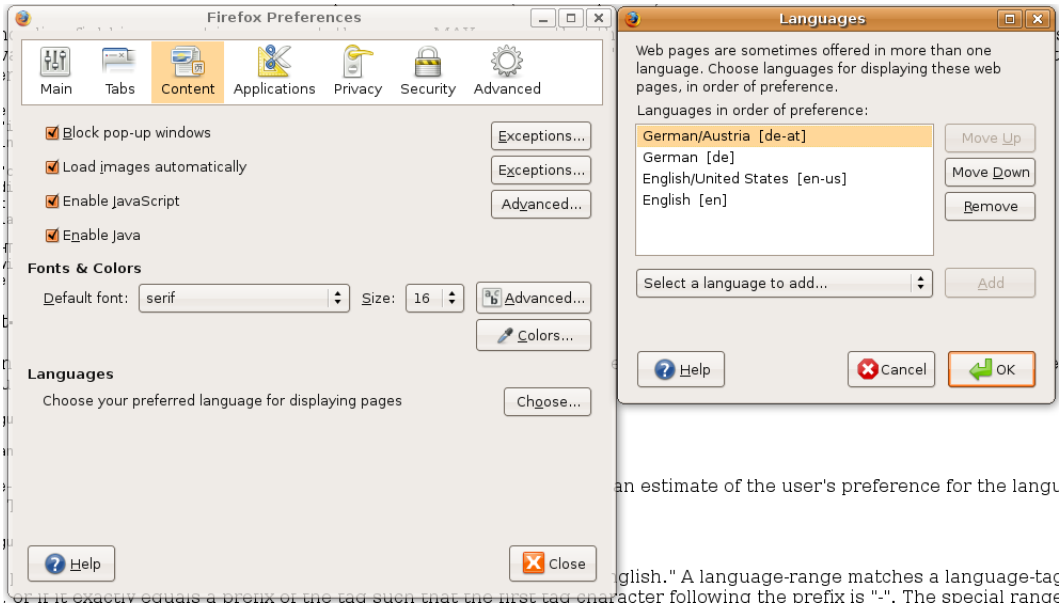


Abbildung 11: Auswahl bevorzugter Sprachen im Webbrowser Firefox [Sach08]

vorgesehen sein. Diese kann auf verschiedene Arten realisiert werden. Allerdings sollte man auch hierbei die Internationalisierung und kulturelle Unterschiede bedenken. Es ist zum Beispiel nicht unbedingt eine gute Idee diese Auswahl mit Landesflaggen zu realisieren, speziell dann wenn die Flagge nur als Symbol für eine Sprache dient [Yun02]. Welche Flagge würde unter diesem Gesichtspunkt verwendet, um z.B. Spanisch zu symbolisieren - die Spanische, die Mexikanische, die Argentinische usw. und wieviele BenutzerInnen würden durch dieses Symbol beleidigt oder werden sich eine andere Webseite suchen.

Ein weiterer Punkt ist die Darstellung und das Layout. Wie im Abschnitt 2.4 Farbe, Layout, Kultur beschrieben, sind grundsätzlich Aspekte des User Interface Design zu berücksichtigen. Bei der Verwendung unterschiedlicher Sprachen kann es z.B. durch die Übersetzungen zu Veränderungen der Textlänge kommen. Dies muss beim Design der Benutzerschnittstelle mit einkalkuliert werden und auch getestet werden. Weiters sollte, um eine korrekte Anzeige durch den Client zu gewährleisten, grundsätzlich jede Response mittels des HTTP Headers Parameter charset die Zeichenkodierung festlegen.

Client Layer

Die verwendeten Clients können bei Webanwendungen naturgemäß nicht vorgeschrieben werden, sondern nur empfohlen werden. Es gibt eine Menge an Webbrowsern, die teilweise doch recht unterschiedlich in der Behandlung gewisser Standards bzw. Teilen von Standards (z.B. CSS Formatierungen) sind, aber auch bei der Interpretation von

Javascript, welches durch die AJAX[8] neue Bedeutung gewonnen hat, gibt es zahlreiche Eigenheiten. Deshalb ist es um so wichtiger, die Bestrebungen der Internationalisierung und Lokalisierung auch an verschiedensten Browsern zu testen. Dieses Testen sollte zwar auch bei nicht-internationalisierten Anwendungen durchgeführt werden, nimmt aber, durch die Notwendigkeit mehrere sprachlich und kulturell angepasste Versionen zu testen, deutlich mehr Zeit in Anspruch.

2.6.2 Übersetzung und Testen

Wie bereits angedeutet, ergeben sich durch die Internationalisierung einer Anwendung zusätzlich Aufgaben, die den Ablauf der Entwicklung und auch des Testens beeinflussen. In [Stea04] wurde exemplarisch eine internationalisierte bzw. kulturell angepasste Webapplikation entwickelt und in den Ausführungen auch die Abläufe der Übersetzung und des Testens dargelegt. Aus diesen Ausführungen ist abzuleiten, dass um eine hohe Qualität der Übersetzung zu erreichen auf spezialisierte Unternehmen (sog. Localization Vendors) zurückgegriffen werden sollte. Es existieren zwar auch Ansätze für eine maschinelle Übersetzung, diese können aber in der Regel nur mit anschließender menschlicher Nachbearbeitung zufriedenstellende Ergebnisse liefern. Bei der Übersetzung in mehrere Sprachen kann es auch durchaus vorkommen, dass nicht nur ein sondern mehrere Localization Vendors koordiniert werden müssen, da ein Localization Vendor nicht alle geforderten Sprachen abdecken kann. Diesen Umständen ist bei der Projektplanung Rechnung zu tragen. Im Beispiel aus [Stea04] wird ein Workflow festgelegt, der einen Translation Coordinator vorsieht, der die vom Entwicklerteam extrahierten sprachabhängigen Dateien verifiziert und an ein, in diesem Fall konzerninternes, Translation Center weitergibt. Durch das Translation Center werden die Dateien an die jeweiligen Übersetzer und nach der Übersetzung wieder an den Translation Coordinator weitergeleitet. Dieser verifiziert die Übersetzungen und gibt diese an das Entwicklungsteam zur Verwendung.

Die Integration von verschiedenen Übersetzungen bzw. das Bereitstellen von verschiedensprachigen Benutzerschnittstellen erfordert auch Erweiterungen des Testablaufs. Wird eine Anwendung nach aktuellen Gesichtspunkten der Internationalisierung entwickelt, so ist ein zentraler Effekt, dass nur eine kompilierte Version der Anwendung für alle unterstützten Lokalisierungen verwendet wird. Dies ergibt den Vorteil das Funktionstests nicht für jede Sprachversion komplett durchgeführt werden müssen [Stea04].

[8] **A**synchronous **J**avascript **a**nd **X**ML

Darüberhinaus werden in [Stea04] z.B. Translation Tests, Globalization Feature Tests, Browser Tests und Usability Tests verwendet. Translation Tests sollen dabei, wie der Name vermuten lässt, darauf abzielen die Qualität der Übersetzungen zu verbessern aber auch zu prüfen, ob Übersetzungen für das Ziellocale passend sind. Dies ist für ein projekteigenes Testteam nicht ohne weiteres erreichbar, da dazu Personen notwendig sind, die die Sprache und Kultur kennen. Deshalb bietet sich auch beim Testen eine Zusammenarbeit mit dem Localization Vendor an. Durch Globalization Feature Tests soll überprüft werden, ob Locales richtig gesetzt werden und dementsprechend die vorgesehenen Formatierungen, Sortierungen usw. angezeigt werden. Davon betroffen sind auch Anpassungen an der Benutzerschnittstelle wie die Unterstützung von bidirektionalen Schriftsystemen und geänderte Anordnungen z.B. von Formularelementen.

Wie verhält sich das Design bei unterschiedlichen Browsern in den verschiedenen unterstützten Sprachvarianten, sollte ebenfalls abgedeckt werden, um einen gleichmäßigen Eindruck der Benutzerschnittstelle zu gewährleisten. Für Usability Tests skizziert [Stea04] zwei mögliche Vorgehensweisen. Das Testen durch ausgewählte Personen wie professionelle ÜbersetzerInnen, TechnikerInnen, einschlägige BeraterInnen oder auch Kunden kann Aufschlüsse zur Verbesserung der Verwendbarkeit der Webapplikation liefern. Als zweite Variante könnte eine Online-Umfrage unter den BenutzerInnen der Webapplikation zu Erkenntnissen über weitere Verbesserungsmaßnahmen führen.

Es ist also ersichtlich, dass neben den Anforderungen an die Implementation selbst auch weitere Tätigkeiten und Bereiche der Entwicklung einer Applikation durch die Internationalisierung betroffen sind, und dementsprechend angepasst werden müssen.

3 I18n Vergleichskriterien und Beispielkonzeption

Im theoretischen Teil dieser Arbeit wurde versucht, die möglichen Aspekte der Internationalisierung von Applikationen im Allgemeinen und von Web Applikationen im Speziellen möglichst umfassend darzulegen. Die dadurch gewonnenen Erkenntnisse werden nun dazu verwendet, ein System von Kriterien zu erstellen, mit dessen Hilfe die Internationalisierungfähigkeiten von Webentwicklungsframeworks und der zugrundeliegenden Programmiersprache beurteilt werden können.

3.1 Vergleichskriterien

Für die Kriterien wurde eine Gliederung in die Gebiete Allgemein, Text/Sprache, Formatierung, Systemnachrichten und Benutzerschnittstelle/Layout gewählt. Das Gebiet **Allgemein** beinhaltet dabei folgende Informationen:

- Programmiersprache
- zugrunde liegendes Architekturmuster
- benötigter Server
- Möglichkeiten zur Datenbankanbindung

Eine wichtige Information stellt dabei die verwendete Programmiersprache des Frameworks dar, weil in vielen Fällen diese schon I18n-Fähigkeiten enthält, welche ebenfalls verwendbar sind. Ein Großteil der existierenden Webentwicklungsframeworks verwenden das Model-View-Controller Architekturmuster, trotzdem wird es der Vollständigkeit halber aufgeführt. Die weiteren Punkte betreffen die Infrastruktur, die zum Betrieb einer Webapplikation mit dem jeweiligen Framework benötigt wird. Als erstes welcher Webserver bzw. Applicationserver wird vorausgesetzt und zweitens, da fast jede Webapplikation datenbankbasiert ist, welche Möglichkeiten zur Anbindung von Datenbanken sind vorgesehen: gibt es eine im Framework oder der Programmiersprache vorhandene Variante oder sind zusätzliche Frameworks nötig bzw. können diese integriert werden.

Die Kategorie **Text/Sprache** umfasst die Möglichkeiten zum Umgang und der Verarbeitung von Text in verschiedenen Sprachen und Schriftsystemen. In einem Blogeintrag vom 5. Mai 2008 schreibt Dr. Mark Davis, Senior International Architekt der Firma Google und Mitbegründer des Unicode Projektes, dass im Dezember 2007 nach Statistiken

von Google erstmals mehr Webseiten in Unicode (UTF-8) kodiert waren, als in sonstigen Zeichensätzen bzw. Kodierungen. Aus dieser Statistik ist auch ein klarer Trend Richtung Unicode UTF-8 im Web ablesbar. Deshalb ist die Unicode-Unterstützung durch Frameworks und zugrunde liegende Programmiersprachen ein wichtiges Kriterium für deren I18nFähigkeiten. Das Minimum hierbei ist die Möglichkeit UTF8 kodierte Seiten zu generieren, eine umfassende Unicode-Unterstützung geht allerdings weit darüber hinaus. Daraus ergeben sich folgende Punkte:

- Unicode Support (ja/nein, nur Kodierung, erweiterter Support, welche Unicodeversion)
- Text Boundary Detection
  - Buchstabengrenzen
  - Wortgrenzen
  - Satzgrenzen
  - locale-sensitive Anpassungen
- Sortierung
  - locale-sensitive Anpassungen
  - Handhabung von kombinierenden und äquivalenten Unicode-Zeichen
- Suchen
  - locale-sensitive Anpassungen
  - Handhabung von kombinierenden und äquivalenten Unicode-Zeichen
- Regular Expressions
  - gibt es eine RegEx Maschine
  - können RegEx beim Suchen oder der Validierung verwendet werden
  - ist die RegEx Maschine Unicodeaware

Die Punkte Text Boundary Detection, Sortierung und Suchen beziehen sich auf das Vorhandensein derartiger Funktionalitäten in der Programmiersprache oder dem jeweiligen Framework. Um den Grad der Unicode-Unterstützung erkennen zu können, zielen die jeweiligen Unterpunkte auf bestimmte Eigenschaften der Unicode-Spezifikation ab, vor allem die Anpassungen an die Eigenheiten unterschiedlicher Locales, aber auch den

richtigen Umgang mit kombinierenden Unicode-Zeichen also z.B. dem Erkennen von verschiedenen Repräsentationen von Umlauten bei der Suche.

Regular Expressions sind unter anderem eine flexible Möglichkeit zur Validierung von Eingabedaten. Existiert also für die Programmiersprache eine RegEx Maschine und ist es über das Framework möglich RegEx in das Validierungssystem zu integrieren, stellt dies eine nicht zu unterschätzende Erweiterung der Möglichkeiten dar. Eine zusätzliche Voraussetzung ist allerdings die Fähigkeit der RegEx Maschine mit Unicode und dessen Strukturen umzugehen.

Unter **Formatierung** wird der Umgang mit Datums- und Zeitangaben, Zahlen, Währungen und Maßeinheiten zusammengefasst und es werden folgende Kriterien verwendet:

- Datum und Zeitangaben:
  - locale-sensitive Formatierung
  - verschiedene vordefinierte Ausgabeformate
  - welche Kalendersysteme werden unterstützt
  - Zeitzonen
    - Datumsvergleich unter Berücksichtigung der Zeitzonen
  - Eingabe
    - locale-abhängige Eingabeformate und Validierung
    - GUI Komponente zur Datumsauswahl (Datechooser)
- Zahlen und Währungen
  - locale-sensitive Formatierung
  - verschiedene vordefinierte Ausgabeformate
- Maßeinheiten
  - locale-sensitive Formatierung
  - ist Umrechnung möglich

Eine wichtige Eigenschaft beim Umgang mit Datums- und Zeitangaben ist die korrekte Formatierung, abhängig von einen gewählten Locale. Dabei wäre es auch wünschenswert, zwischen verschiedenen Ausgabeformaten, z.B. rein numerischen oder Formaten mit Tagesnamen und Monatsnamen, wählen zu können, die den Vorgaben des Locales genügen. Die Unterstützung von zusätzlichen Kalendersystemen erweitert ebenso die

Möglichkeiten der Anpassung an Kulturen die nicht den gregorianischen Kalender verwenden. Da es bei international verwendeten Anwendungen auch dazu kommen kann, dass man mit Zeitangaben aus verschiedenen Zeitzonen operieren muss, sollte die Handhabung dieser von Seiten einer umfangreichen I18n-Bibliothek ebenfalls gewährleistet sein. Ein wichtiger Punkt bei Webanwendungen ist die Eingabe von Daten per Formular und deren Validierung. Im Fall von Datums- und Zeitangaben sollte das Formular bzw. der Validierungsmechanismus so flexibel sein, dass die Überprüfung auf Gültigkeit nach locale-sensitiven Formaten erfolgen kann. Weitere Möglichkeiten wären GUI Komponenten, die eine Kalenderansicht darstellen und somit die Datumseingabe zu einer Auswahl umgestalten. Bei Zahlen und Währungsangaben aber auch bei Maßeinheiten steht vor allem die locale-sensitive Formatierung im Vordergrund. Darüber hinaus wäre bei Maßeinheiten, durch das Vorhandensein von konstanten Umrechnungsfaktoren, die Umrechnung in andere Einheiten interessant.

Die nächste Gruppe **Systemnachrichten** beinhaltet Kriterien zum Umgang mit Meldungen des Systems, welche je nach Locale in der passenden Sprache ausgegeben werden müssen. Um dies zu ermöglichen, werden diese Meldungen ausgelagert also externalisiert.

- Externalisierung
  - Framework- bzw. Programmierspracheneigenes System
    - Umgang mit Pluralformen
    - Handhabung von dynamischen Elementen der Nachrichten
  - gibt es alternative ExternalisierungsSysteme
    - GNU gettext Unterstützung
    - Umgang mit Pluralformen
    - Handhabung von dynamischen Elementen der Nachrichten
- verwendbare Dateiformate
  - Editoren für Externalisierungsdateien verfügbar

Hierbei ist zu klären, ob das Framework bzw. die Programmiersprache ein System zur Auslagerung von Texten mitbringt, ob Implementierungen von bewährten Systemen wie z.B. GNU gettext existieren, oder ob weitere alternative Systeme verwendbar sind. Die Fähigkeiten im Umgang mit Pluralformen der gettext Bibliothek wurden bereits im Theorieteil beschrieben, für Framework- bzw. Programmierspracheneigene und alternative Systeme ist die Fähigkeit zu klären, da dies einen essentiellen Punkt bei

der korrekten Ausgabe von Nachrichten darstellt. Ebenso muss die Handhabung bzw. Einbindung von dynamischen Elementen (z.B. Anzahlen oder Bezeichner, die erst zur Laufzeit feststehen) in die auszugebenden Nachrichten von einem System zur Externalisierung bewältigt werden. Da die Auslagerung häufig in Dateien erfolgt, welche dann an Übersetzer weitergegeben werden, ist auch interessant welche Dateiformate (z.B. .po Dateien bei GNU gettext oder XML Formate) verwendbar sind und ob für das jeweilige Dateiformat Editoren zur einfacheren Bearbeitung vorhanden sind.

Die Kategorie **Benutzerschnittstelle und Layout** bildet den Abschluss mit folgenden Kriterien:

- Externalisierung von Bildern und Multimediainhalten (Sound / Video)
- AJAX Integration
  - Internationalisierung von JavaScript
    - locale-sensitive Formatierung (z.B. Datum, Währung)
    - Externalisierung von Strings möglich
- Template System
  - vom Framework vorgegeben / externes System nutzbar
  - verschiedensprachige Templates für eine Seite möglich
  - Anzeige nach Locale
  - Verhalten bei Anforderung eines nicht vorhandenen Locales
- Modularisierung
  - GUI Komponentenbasierend
    - konfigurierbare Komponenten
  - Erstellen eigener GUI Komponenten möglich

Da Bilder und Multimediainhalte, die regionsspezifische Elemente enthalten, je nach Locale gegen angepasste Versionen ausgetauscht werden müssen, sollte ein Mechanismus vorgesehen sein um dies ähnlich der Externalisierung von Texten abzuwickeln. Durch die zunehmende Verbreitung von AJAX und dessen Integration in viele Frameworks werden möglicherweise auch Ausgaben durch JavaScript generiert. Deshalb muss auch für Javascript eine Möglichkeit zur Einbindung verschiedensprachiger Strings nach Locale bzw. die Formatierung von Datums- oder Währungsangaben vorgesehen sein. Da die

Verwendung eines Template-Systems es ermöglicht, große Mengen an statischen Text direkt in verschiedensprachige Templates zu integrieren, ist das Vorhandensein eines solchen Systems ein wichtiges Kriterium für die I18n Fähigkeiten eines Frameworks. Es sollte in der Lage sein, verschiedene Templates für eine Seite je nach Locale zu wählen und falls ein nicht unterstütztes Locale angefordert wird, einen Mechanismus vorsehen, dass zuerst versucht wird, ein Template das zumindest im Sprachteil mit dem angeforderten übereinstimmt zu verwenden, ehe ein Default-Template zur Anwendung kommt. Um etwa für unterschiedliche Locales den Funktionsumfang anpassen zu können, wäre es wünschenswert, dass die verwendbaren Elemente als wiederverwendbare und konfigurierbare Komponenten vorliegen und auch das Erstellen eigener Komponenten möglich ist.

3.2 Beispielimplementierung

Um mit den oben definierten Kriterien Webapplicationframeworks hinsichtlich ihrer I18n-Fähigkeiten zu beurteilen, werden im Folgenden zu den Kategorien Text/Sprache und Formatierung beispielhafte Aufgaben festgelegt. Die Kategorien Systemnachrichten und Benutzerschnittstelle, Layout werden nicht mit gesonderten Aufgaben, sondern in anderen Beispielen integriert abgedeckt.

Zur Umsetzung des Beispiels kommen 3 aktuelle Frameworks zum Einsatz: Tapestry/JAVA, Zend Framework/PHP, ASP.Net/C#. In ersten Überlegungen war geplant, nur OpenSource Frameworks zu vergleichen und deshalb Ruby on Rails statt ASP.Net zu verwenden. Ruby on Rails bringt allerdings keine I18n Fähigkeiten mit, sondern es existieren einige PlugIns (u.a. gettext), mit unterschiedlichsten Funktionsumfang und Entwicklungsstand. Die gettext Implementierung, die in der Literatur zu Ruby on Rails oft verwendet wird und welche auch kontinuierlich weiter entwickelt wird, deckt allerdings nur den Bereich der Systemnachrichten ab. Somit wurde die Entscheidung getroffen, mit ASP.Net ein kommerzielles closed Source Framework in den Vergleich mit aufzunehmen, was diesen auch repräsentativer gestaltet.

Allgemeines zur Implementierung: Alle Dokumente werden in Unicode UTF-8 kodiert. Die grafische Oberfläche der Beispielanwendung soll in Deutsch und Englisch lokalisiert werden. Dazu sind die vom jeweiligen Framework vorgesehenen Techniken zu verwenden. Dies betrifft die folgenden Punkte:

- statische Texte wie die Beschreibung der einzelnen Beispiele
- Status und Fehlermeldungen (auch mit dynamischen Elementen etwa Anzahl gefundener Suchbegriffe)
- Grafiken (die Submit Buttons der Formulare werden als beschriftete Grafiken ausgeführt und müssen deshalb je nach gewähltem Locale der GUI angepasst werden)

Beispielaufgaben:

Text/Sprache: Da Unicode eine große Bedeutung bei der Internationalisierung von Webapplikationen hat, sollen die Fähigkeiten des jeweiligen Frameworks bzw. der zugrunde liegenden Programmiersprache bezüglich der Verarbeitung von Unicode getestet werden. Dies erfolgt mittels der folgenden 3 Aufgaben:

Text Boundary Detection

Eingabe:

- ein Beispieltext bestehend aus einigen Sätzen
- ein Locale, nach dessen Regeln die Detection durchgeführt wird.

Ausgabe:

- Eingabetext mit gekennzeichneten Wort und Satzgrenzen

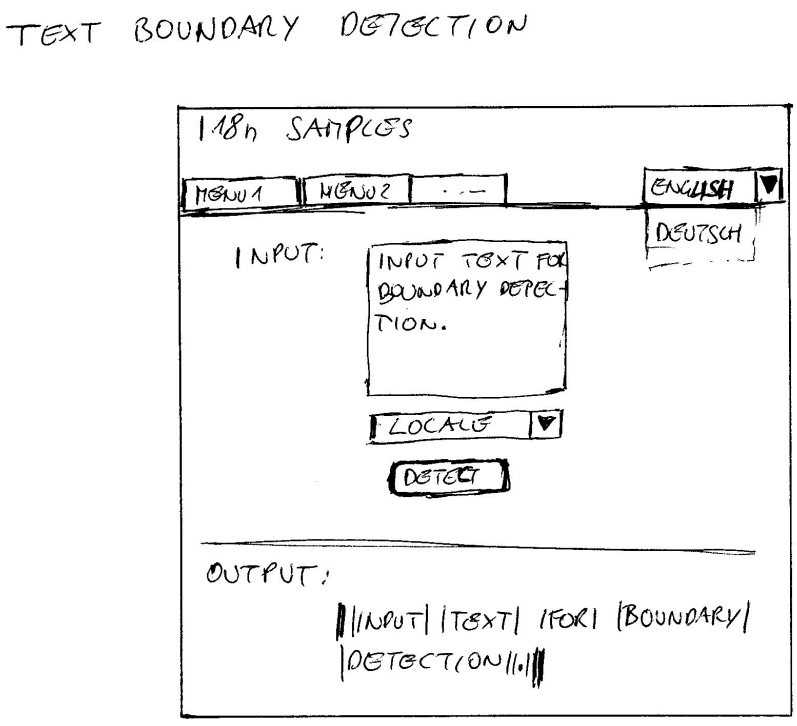


Abbildung 12: Skizze - Text Boundary Detection

Suchen

Eingabe:

- ein Beispieltext
- ein Suchbegriff

Ausgabe:

- Statusmeldung, wie oft der Begriff gefunden wurde
- Eingabetext mit hervorgehobenen Suchbegriffen

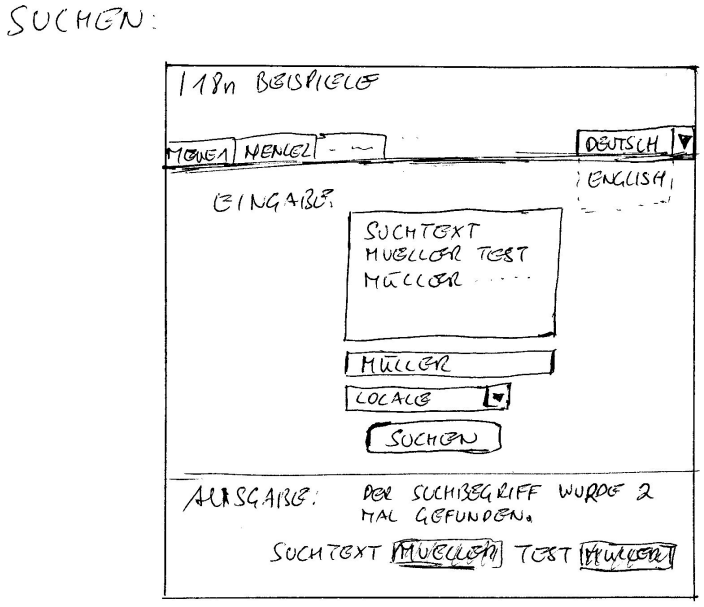


Abbildung 13: Skizze - Suchen

Da der Umgang mit Unicode-Eingabedaten getestet werden soll, ist z.B. von besonderem Interesse, ob die Suche so implementiert werden kann, dass z.B. für ein deutsches Locale die verschiedenen Repräsentationen von Umlauten und auch deren Expansion zu ae, ue, oe korrekt gefunden werden. Je nach dem, welche Möglichkeiten das Framwork bzw. die Programmiersprache bieten, können zusätzliche Optionen vorgesehen werden etwa eine case-sensitive Suche.

Sortierung

Eingabe:

- einige Wörter
- das Locale, nach dessen Regeln sortiert werden soll

Ausgabe:

- nach Regeln des Locales sortierte Liste der Eingabewörter

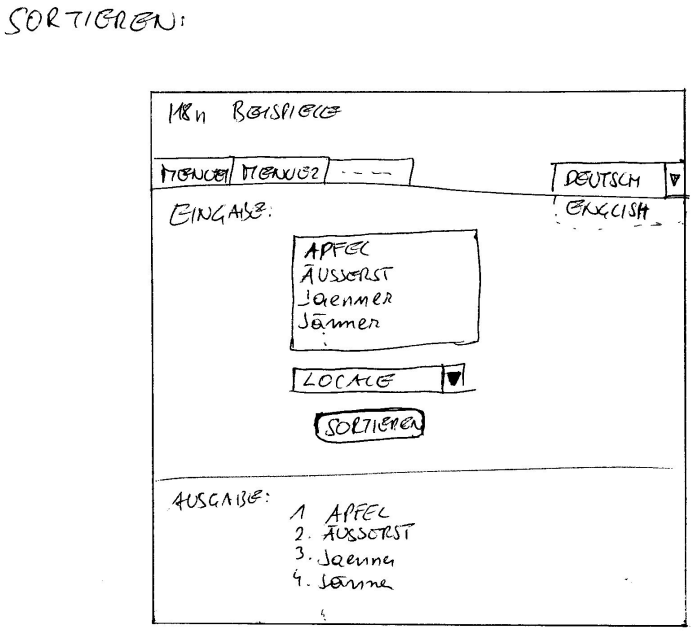


Abbildung 14: Skizze - Sortierung

Bei dieser Aufgabe soll die locale-abhängige Sortierung getestet werden, d.h. wird als Ausgabe-Locale z.B. schwedisch für Schweden gewählt, muss das [ä] nach dem [z] einsortiert werden, im Deutschen dagegen nach dem [a]. Je nach Möglichkeiten der Sortierung in der jeweiligen Programmiersprache können auch zusätzliche Eingabeoptionen vorgesehen werden, um z.B. case-sensitive zu sortieren. Es wird kein Beispiel zu Regular Expressions implementiert, da hier die Unterschiede zwischen den einzelnen Implementierungen sehr groß sind, und z.B. in [Frie08] Jeffrey E. F. Friedl – Reguläre Ausdrücke ausführlich behandelt werden.

Datum/Zeit: Bei Datums- und Zeitangaben ist einerseits die Eingabe bzw. die Validierung nach dem gewählten Locale der GUI und andererseits die korrekte Ausgabeformatierung von Interesse. Es werden zwei Datums- und Zeitwerte eingelesen und jeweils eine Zeitzone. Mit diesen Angaben wird ein Datumsvergleich durchgeführt und die beiden Werte nach den Formatierungsrichtlinien des ausgewählten Locales formatiert ausgegeben. Zusätzlich wird das Ergebnis des Vergleichs angeführt.

Eingabe:

- 2 Datums- und Zeitangaben mit Auswahl einer Zeitzone
- ein Ausgabe-Locale

Ausgabe:

- nach Locale formatierte Ausgabe der beiden Datums- und Zeitangaben
- Ergebnis eines Vergleichs der beiden Datums- und Zeitangaben (es soll geprüft werden, ob die Angaben gleich sind oder welche Angabe zetlich vor der anderen liegt)

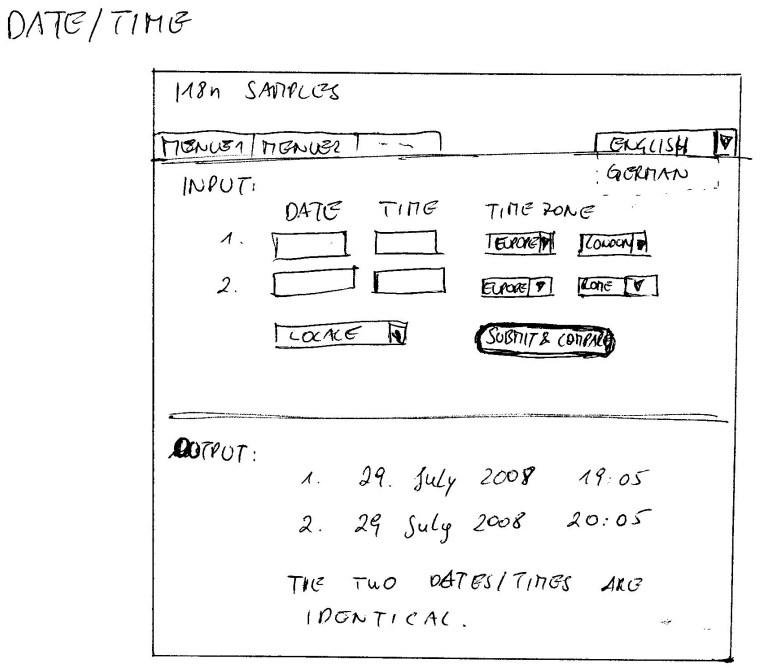


Abbildung 15: Skizze - Datum/Zeit

Zahlen/Währung/Maßeinheiten:

Eingabe:

- eine Zahl
- ein Ausgabe-Locale
- Auswahl ob als Zahl, Währung oder einer Maßeinheit formatiert werden soll
- bei Maßeinheit, eine der unterstützten Einheiten

Ausgabe:

- der nach der Auswahl formatierte Wert

Bei dieser Aufgabe wird die Fähigkeit, Zahlenwerte und Währungsangaben zu formatieren, getestet. Ein Zahlenwert soll wahlweise als Zahl, Währung bzw. Prozentangabe oder als Maßeinheit formatiert ausgegeben werden.

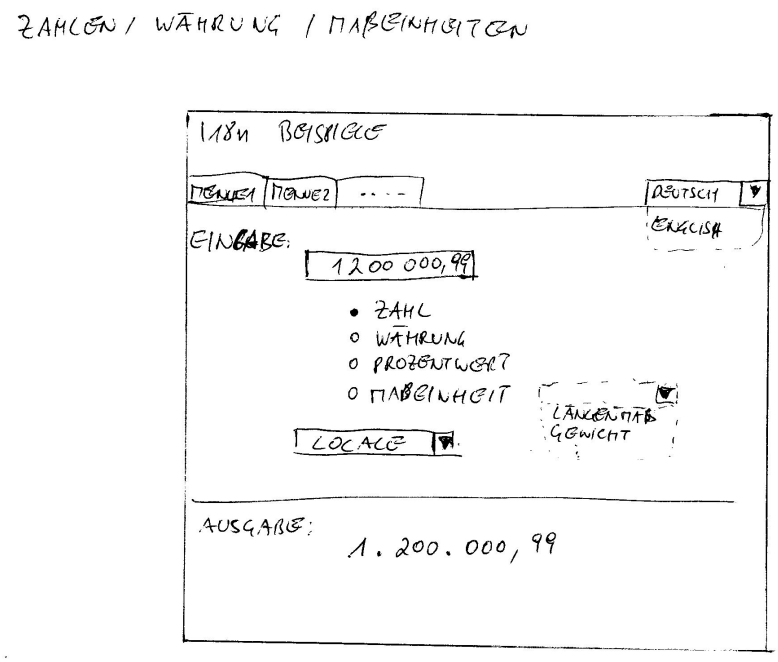


Abbildung 16: Skizze - Zahlen/Währung/Maßeinheiten

Vermutetes Ergebnis:

Es ist zu erwarten, dass sowohl die Kombination Tapestry/JAVA als auch ASP.Net/C# die umfassenderen I18n Fähigkeiten gegenüber Zend Framework/PHP aufweisen werden. Ein Grund für diese Hypothese ist die mangelnde Unicode-Unterstützung durch PHP, weshalb mit dieser Kombination die Aufgaben der Kategorie Text/Sprache wahrscheinlich nur mit unverhältnismäßig hohem Aufwand wenn überhaupt zu bewältigen sind. Ob nun die JAVA Variante oder die .NET Lösung besser abschneiden oder vielleicht sogar gleichwertig sein werden, wird der konkrete Vergleich zeigen.

4 Analyse und Vergleich

Die durch das Studium der Literatur zu den jeweiligen Frameworks bzw. Plattformen erworbenen Erkenntnisse, in Kombination mit den Eindrücken und Erfahrungen bei der Implementierung, der, im vorigen Kapitel beschriebenen, Beispiele, dienen nun als Basis zur Erstellung der Analyse und des Vergleichs der drei verwendeten Frameworks und Programmiersprachen hinsichtlich ihrer I18n Fähigkeiten. Dazu wird die im vorigen Kapitel eingeführte Gliederung der Kriterien verwendet und zu jeder Gruppe von Kriterien für die einzelnen Frameworks beschrieben, ob und wie sie die Anforderungen bewältigen.

4.1 Allgemein

Dieser Abschnitt liefert einen Überblick der verwendeten Versionen und einiger grundlegender Charakteristika der Frameworks und Programmiersprachen.

Tapestry Tapestry ist ein Open-Source Framework, welches die Programmiersprache Java verwendet. Das Framework basiert auf dem Model-View-Controller (MVC) Architekturmuster und bietet eine strenge Trennung zwischen HTML und Anwendungscode [Shi04]. Eine Anwendung besteht aus sog. Pages, welche sich wiederum jeweils aus einer HTML-Datei, einer XML Page Description und einer Java Klasse zusammensetzen. Zur Ausführung einer erstellten Webapplikation und des Frameworks wird ein Servlet Container wie etwa Apache Tomcat benötigt. Tapestry stellt Komponenten für Formulare und Formularelemente, Link- und Imagekomponenten, aber auch abstraktere, wie eine For-Komponente, welche in einer Schleife das komponenteneigene Template wiederholt in die generierte Seite einfügt, zur Verfügung. Zur Anbindung einer Datenbank, welche bei vielen Webapplikationen verwendet wird, bietet das Framework keine eigene Unterstützung. Man kann hierfür die Java eigenen Mittel (JDBC) oder auch spezielle Frameworks für diesen Zweck einsetzen: z.B. Hibernate. Für die Beispielimplementierung wurde Tapestry Version 4.1.5 und das Java SDK Version 1.6. Als Entwicklungsumgebung kam NetBeans Version 6.1 und Tomcat 6.0 zu Einsatz.

ASP.NET ASP.net ist eine Technologie von Microsoft zur Erstellung von Webanwendungen und Bestandteil des ebenfalls von Mircosoft stammenden .net Frameworks. Als Programmiersprache können grundsätzlich alle für .net geeigneten Sprachen verwendet werden, wie z.B. C# oder Visual Basic. Im Gegensatz zu vielen anderen Frameworks, kommt nicht das MVC Architekturmuster zum Einsatz, wenngleich an einer ASP.NET

MVC Extension gearbeitet wird. Das Rendern der angeforderten Page erfolgt im Zuge einer Request Processing Pipeline [EvHaRa08]. Als Server kommt der Internet Information Server (IIS) von Microsoft zum Einsatz, wobei seit IIS7 dessen Request Processing Pipeline und die von ASP.NET intergriert sind, und einige Doppelgleisigkeit, die sich in den älteren Versionen von IIS teilweise negativ auf die Performance auswirkten, beseitigt wurden. Für Entwicklungszwecke bringen sowohl Microsoft Visual Studio 2008 als auch das frei verfügbare Visual Web Developer Express 2008 einen Server mit. In ASP.Net enthalten sind neben Formular und HTML Komponenten mit ADO.NET und LINQ auch Möglichkeiten zur Anbindung von Datenquellen wie z.B. Datenbanken. Zur Beispielimplementierung wurde das .net Framework 3.5 und Visual Web Developer Express 2008 verwendet.

Zend Framework Das Zend Framework ist ein Open-Source Framework und verwendet als Programmiersprache PHP, wobei die PHP Version 5 Voraussetzung ist, da erst ab dieser Version die für das Zend Framework notwendige Objektorientierung vollständig unterstützt wird. Das Framework verwendet als Architekturmodell das Model-View-Controller Muster [Moeh08]. Als Server kann jeder Webserver mit PHP Unterstützung dienen also z.B. Apache mit mod_php. Das Zend Framework bringt unter anderem mit Zend_DB einen Datenbankabstraktionslayer mit, welcher den Zugriff auf verschiedene Datenbanken in einer einheitlichen API kapselt. Bei der Implementierung des Beispiels wurde das Zend Framework in der Version 1.5.3, PHP 5.2.5 und Zend Studio for Eclipse 6.0.1 verwendet.

4.2 Text/Sprache

Tapestry/Java Das Tapestry Framework stellt keine Klassen zum Umgang mit Unicode bereit, da diese bereits in der Java Laufzeit Umgebung vorhanden sind. Die Java Klasse `java.text.BreakIterator` wird dabei zur Analyse von Textbegrenzungen verwendet [DeiCza01]. Mittels `BreakIterator` können locale-sensitiv natürlich-sprachliche Buchstaben-, Satz- und Wortgrenzen aufgefunden werden. Die Anwendung ist relativ einfach. Die Klasse stellt für jede Art von Iterator (Character-, Word-, Sentence-) überladene Factory-Methoden zur Verfügung, die ein Instanz von `BreakIterator` zurückliefern. Werden diese Methoden ohne Parameter aufgerufen, wird das Default Locale der Java Runtime verwendet, es kann allerdings auch ein Locale als Parameter übergeben werden.

Abbildung 17 zeigt die Ausgabe der Aufgabe Auffinden von Text Begrenzungen, wie

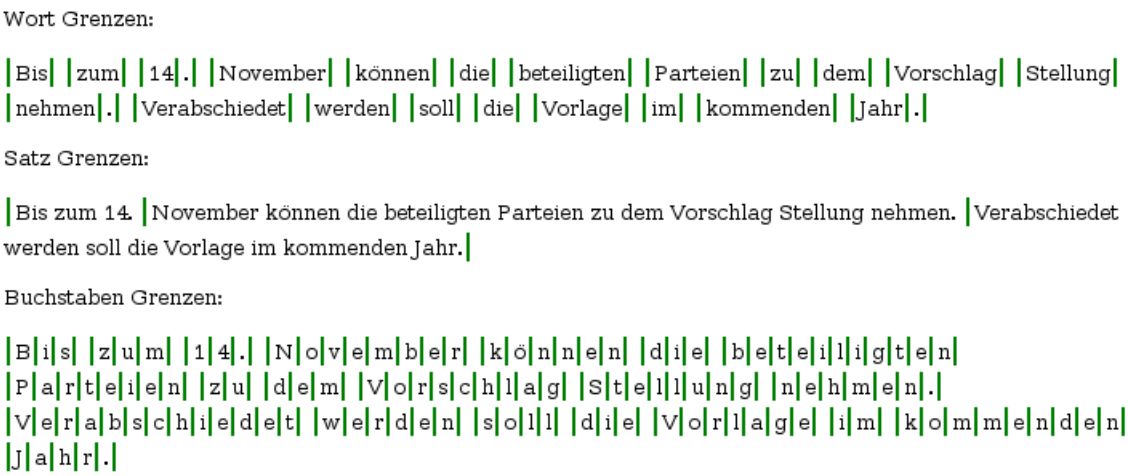


Abbildung 17: Tapestry/Java: Ausgabe beim Auffinden von Textbegrenzungen

sie mit Tapestry Java realisiert wurde. Man sieht auch die Grenzen des Verfahrens zum Auffinden von Satzgrenzen, da z.B. bei Datumsangaben wie 14. November eine Satzgrenze erkannt wird, obwohl diese für menschliche LeserInnen nicht vorhanden ist. Dasselbe passiert etwa bei Abkürzungen und ist nur auf syntaktischer Ebene auch nicht lösbar.

Um Textvergleiche auf Unicode-Basis durchzuführen, stellt Java folgende Klassen bereit:

- `java.text.Collator`
- `java.text.RuleBasedCollator`
- `java.text.CollationKey`
- `java.text.CollationElementIterator`

Der Ausgangspunkt in der Verwendung des Textvergleichssystems von Java ist dabei die Klasse `Collator` oder die spezifischer konfigurierbare Form der `RuleBasedCollator`, welche den eigentlichen Stringvergleich locale-sensitiv bzw. regel-basiert durchführen [DeiCza01]. Um Arrays oder Collections zu sortieren, existiert die statische Methode `sort` in den Klassen `java.util.Arrays` und `java.util.Collections`. Diese Methode erhält als Parameter die zu sortierende Collection bzw. das Array und ein Objekt, welches das `Comparator Interface` implementiert, dies ist für `Collator` und `RuleBaseCollator` der Fall, und führt die Sortierung nach den Vorgaben des Collator Objektes durch.

Die Abbildungen 18 und 19 zeigen die Sortierung von einzelnen Zeichen nach den Locales für Deutsch in Österreich und Schwedisch in Schweden. Auffällig ist die Sortierung

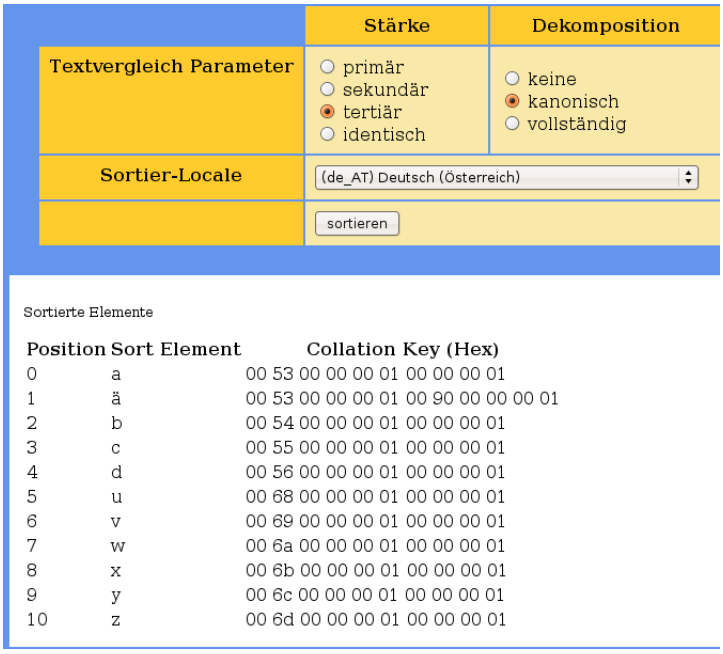


Abbildung 18: Tapestry/Java: Beispiel der deutschen Sortierung

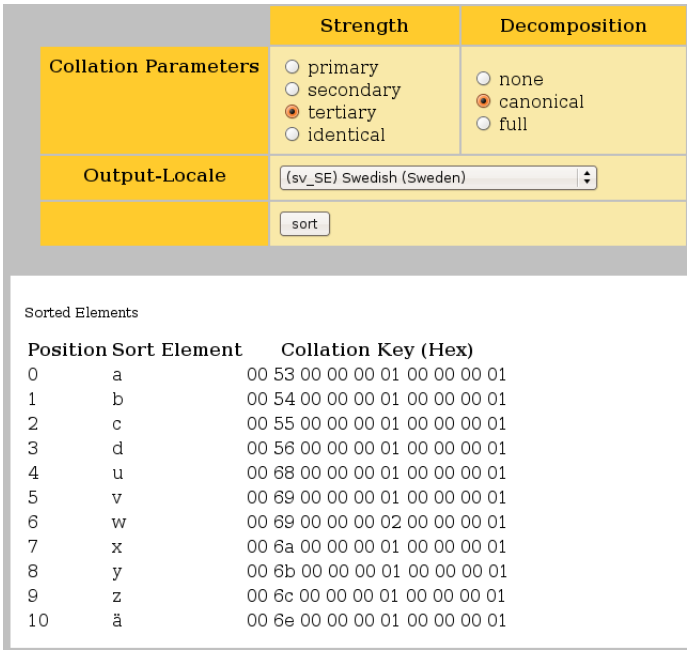


Abbildung 19: Tapestry/Java: Beispiel der schwedischen Sortierung

von [ä] nach dem [a] im Deutschen und nach dem [z] im Schwedischen. Weiters fällt bei der schwedischen Sortierung auf, dass die Sort Keys für [v] und [w], im Gegensatz zum Deutschen, auf der primären Ebene gleich sind (0x0069).

Für die Suche in Texten kommt `CollationElementIterator` zum Einsatz [DeiCza01]. Nur die `RuleBasedCollator` Klasse stellt eine Methode zur Erstellung eines CollationElementIterators zur Verfügung, deshalb müssen Collator Objekte zuerst mittels cast in RuleBasedCollator Objekte umgewandelt werden, ehe sowohl für den Suchstring als auch den zu durchsuchenden Text jeweils ein CollationElementIterator erstellt werden kann. Der CollationElementIterator stellt einerseits Methoden zum Durchlaufen der Elemente in beide Richtungen, andererseits auch Methoden um den Offset des aktuell behandelten Elements im zugrundeliegenden Text zu bestimmen, zur Verfügung. Unter Anwendung dieser Klassen bzw. Methoden können diverse Suchalgorithmen realisiert werden.

ASP.NET/C# ASP.Net bzw. das .Net Framework bietet Unicode-Support und verwendet intern die UTF-16 Codierung für Strings. Microsoft verwendet anstatt des Begriffs Locale den Begriff Culture. Die Bezeichnungen der Cultures folgen demselben Muster wie die der Locales, allerdings wird zur Trennung von Sprach- und Regionsidentifier der Bindestich und nicht der Unterstrich verwendet. Es gibt in der hierarchischen Anordnung der Cultures drei Typen [Smit06]:

- Invariant Culture: ist weder einer Sprache noch einer Region zugeordnet und bildet die Wurzel der Hierachie.
- Neutral Culture: ist einer Sprache aber keiner Region zugeordnet z.B. en oder de.
- Specific Culture: ist einer Sprache und einer Region zugeordnet z.B. en-US oder de-AT.

Weiters wird zwischen CurrentCulture und CurrentUICulture unterschieden und beides für den aktuellen Thread gespeichert [Smit06]. CurrentUICulture bestimmt dabei, welche Resourcen Datei also im Prinzip, welche Sprache für das User Interface verwendet wird und CurrentCulture, welche Regeln für das Formatieren von culture-sensitiven Inhalten oder z.B. das Sortieren angewendet werden. Demnach wäre es also denkbar, dass Menüpunkte und Text z.B. in Deutsch (falls UICulture de oder eine andere specific Culture mit Sprache de ist), die Tages- und Monatsnamen bei Datumsangaben aber z.B. französich (falls CurrentCulture fr oder eine andere specific Culture mit Sprache fr ist) ausgegeben werden.

Im .net Framework befinden sich culture-sensitive Klassen in der Regel im Namespace `System.Globalization`. Für den Umgang mit Texten sind dies:

- `System.Globalization.CultureInfo`
- `System.Globalization.CompareInfo`
- `System.Globalization.StringInfo`

Über die Klasse StringInfo kann ein TextElementEnumerator erstellt werden, welcher es erlaubt, natürlichsprachige Textelemente zu durchlaufen. Für das Auffinden von Wort- oder Satzbegrenzungen gibt es keine Unterstützung. Mit Hilfe des TextElementEnumerator können auch Suchfunktionen implementiert werden. Um mit precomposed und compatibility Charakters umgehen zu können, stellt die String Klasse die Methode normalize zur Verfügung, welche die vier im Unicode Standard beschriebenen Normalisierungsformen erzeugen kann. Seitens der StringInfo-Klasse gibt es noch Methoden, um einen Substring nach Textelementindizes zu erzeugen bzw. die Länge eines Strings in Textelementen zu berechnen.

Das Sortieren von Aufzählungen erfolgt mittels der culture-sensitiven Klasse CompareInfo. Eine Instanz von CompareInfo zur gewählten Culture kann z.B. an die statische Methode Array.Sort übergeben werden und wird dann dazu verwendet, ein eindimensionales Array culture-sensitive zu sortieren.

Zend Framework/PHP Das Zend Framework bzw. auch PHP stellen derzeit zumindestens in den als stable ausgewiesenen Releases keine weitreichendere Unicode-Unterstützung zur Verfügung. Es existieren die PHP Erweiterungen *mbstrings*, welche Implementierungen für einige Stringfunktionen wie z.B. `strlen` enthält, die mit MultiByte Zeichensätzen korrekt funktionieren, oder *iconv*, welches Funktionen zur Konvertierung von Zeichensätzen bereitstellt. Für die Funktionen, welche hier betrachtet wurden (locale-sensitives Auffinden von Textbegrenzungen, Sortieren und Suchen) wird erst PHP 6 Lösungen anbieten, da dieses durch die Einbindung der von IBM entwickelten ICU (International Components for Unicode) und die Umstellung auf Unicode als PHP internen Zeichensatz weitreichende Unicode-Unterstützung enthalten soll [Zmie08].

4.3 Formatierung

Tapestry/Java Auch bei der Formatierung von Datums- und Zeitangaben bzw. Zahlen, Währungen und Prozentangaben bietet Tapestry keine eigenen Klassen, sondern

es kommen die bereits in der Java Laufzeit Umgebung enthaltenen zum Einsatz. Zum Umgang mit Datums- und Zeitangaben existieren die Klassen:

- `java.util.Calendar`
- `java.util.Timezone`
- `java.util.Date`
- `java.text.DateFormat`

DateFormat wird dabei zur locale-sensitiven Ausgabe von Datums- und Zeitangaben und auch zum Parsen von lokalisierten Eingabe-Strings verwendet. Es werden sowohl für den Datums- als auch den Zeitteil vier vordefinierte Formatierungen (SHORT, MEDIUM, LONG, FULL) festgelegt. Weiters existiert die Möglichkeit, über die Subklasse SimpleDateFormat auch eigene Formatierungs-Muster zu definieren. Außerdem kann festgelegt werden, ob beim Parsen strikt nach dem gesetzten Muster oder mittels Heuristiken flexibler vorgegangen werden soll.

Die grundlegende Repräsentation einer Datums- und Zeitangabe erfolgt in Java durch `java.util.Date`, welches die Anzahl der Millisekunden seit 01. Jänner 1970 00:00:00 GMT speichert. Die meisten Funktionen der Date-Klasse, welche die Erstellung oder die Ausgabe eines Date-Objektes betreffen, sind als Deprecated gekennzeichnet und sollten demnach nicht mehr benutzt werden. Statt dessen bieten die Klassen DateFormat und Calendar Methoden zur Erstellung und der Ausgabe von Datums- und Zeitangaben und bieten mit der Einbindung der Klasse Timezone auch die Möglichkeit, eine Zeitzone für die Eingabe bzw. Erstellung und auch die Ausgabeformatierung festzulegen. Über konkrete Subklassen der Calendar-Klasse kann die Unterstützung für andere Kalendersysteme als den gregorianischen Kalender implementiert werden. Die Java Laufzeit-Umgebung enthält eine Implementierung des gregorianischen Kalender mittels der Klasse java.util.GregorianCalendar, nur falls das verwendete Locale *th_TH (thailändisch für Thailand)* ist wird `sun.util.BuddhistCalendar` bzw. für das Locale *ja_JP_JP (japanisch für Japan mit dem Variant JP)* wird `java.util.JapaneseImperialCalendar` instanziert.

Für die locale-sensitive Ein- und Ausgabe von Zahlen, Währungen und Prozentangaben wird die Klasse java.text.NumberFormat verwendet. Über diese kann wie bei den meisten locale-sensitiven Java Klassen durch verschiedene `getInstance` Methoden ein passendes Objekt für die Verarbeitung von Ganz-, Dezimalzahlen, Währungen und

Prozentangaben erzeugt werden. Diese stellen dann Methoden bereit, um lokalisierte Eingabestrings zu parsen, aber auch Zahlenwerte als lokalisierten String auszugeben. Für das Formatieren bzw. das Umrechnen von Maßeinheiten stellt weder das Tapestry Framework noch Java Lösungen bereit.

ASP.NET/C# Die Möglichkeiten zur culture-sensitiven Formatierung mit ASP.Net umfassen die Ein- und Ausgabe von Datums- und Zeitangaben und Zahlen. Sowohl die Klassen zur Repräsentation von Datumsangaben als auch diejenigen für Zahlenwerte enthalten Methoden zum Parsen von Eingabestrings, welche als Parameter einen culture-sensitiven IFormatProvider, der wiederum von der jeweiligen CulturInfo Instanz bereit gestellt wird, erhalten. Es sind verschiedene Standardformatierungen sowohl für Datumsangaben als auch Zahlen über `DateTimeFormatInfo` und `NumberFormatInfo` zur jeweils gewählten Culture definiert. Zu beachten ist, dass das Parsen und die Ausgabeformatierung nur für die Invariant Culture und Specific Cultures nicht aber für Neutral Cultures durchgeführt werden kann. Das Framework stellt deshalb über das CultureInfo Objekt Methoden bereit, um einerseits zu testen, ob eine Culture neutral ist (isNeutralCulture) und andererseits, um eine specific Culture (createSpecificCulture) zu generieren.

Bei der Verarbeitung von Zeitangaben für bestimmte Zeitzonen kann dies beim Parsen nur dann berücksichtigt werden, wenn die Abweichung zu UTC (Universal Time Coordinated) Bestandteil des Eingabe-Strings ist, ansonsten wird ein Datumsobjekt erstellt, welches die Zeitzone der .net Umgebung verwendet. Bei der Beispielimplementierung wurde folgende Variante gewählt: der Eingabe-String wurde ohne Angabe eines Offsets geparst und somit ein Objekt mit der Zeitzone der .net Umgebung erstellt. Danach wurden die Werte für Jahr, Monat, Tag, Stunden, Minuten aus dem durch das parsen erstellten Datumsobjekt und der ausgewählten Zeitzone ein neues Datumsobjekt erstellt, welches nun die gewünschte Zeitzone berücksichtigt. Ein korrekter Vergleich zweier Datumsobjekte mit verschiedenen Zeitzonen, kann mit der `CompareTo` Methode der Datumsobjekte durchgeführt werden.

Weiters bietet `.net` umfangreiche Unterstützung verschiedener Kalender. Jedem CultureInfo Objekt ist ein default Calendar für die jeweilige Culture zugewiesen, darüberhinaus sind auch alternative Kalender mit dem Objekt verknüpft, falls für die Culture solche zur Anwendung kommen (siehe Abbildung 20). Dieser kann über das DateTimeFormatInfo Objekt des verwendeten CultureInfo Objekts festgelegt und bei der Formatierung verwendet werden.

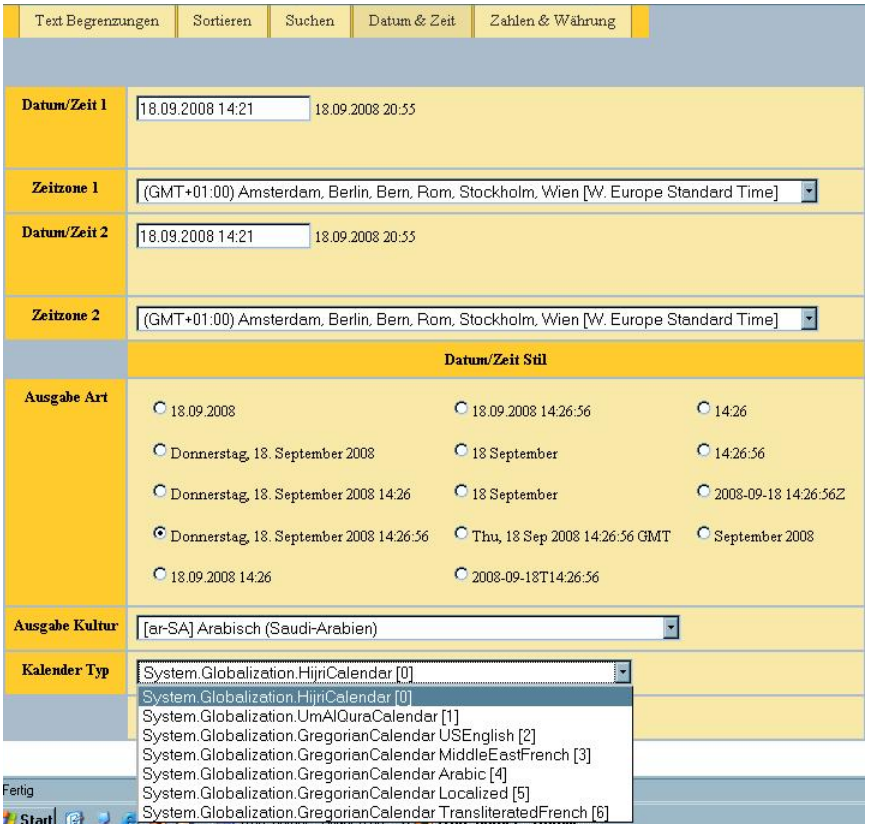


Abbildung 20: ASP.Net/C#: Beispiel Datumsformatierung

Zend Framework/PHP Das Zend Framework enthält die folgenden Klassen, um Formatierungen locale-sensitive zu bewerkstelligen:

- Zend_Locale
- Zend_Locale_Format
- Zend_Currency
- Zend_Date
- Zend_Measure

Zend_Locale_Format stellt dabei Methoden zum Einlesen von Zahlen und Datums- bzw. Zeitangaben bereit und wird von locale-sensitiven Klassen zur Ausgabe derselben verwendet. Das Verhalten dieser Methoden kann durch verschiedene Parameter beeinflußt werden, z.B. die Präzision einer Dezimalzahl oder ein selbst festgelegtes Zahlenformat.

```
//festlegen des Optionen-Arrays mit den aktuellen Locale
//und der gewünschten Stellen nach dem Komma
$optionen = array('locale'=>$locale, 'precision'=>2);

//setzen der Ausgabevariablen mit lokale-sensitv formatierter Zahl
$this->view->formattedNumber = Zend_Locale_Format::toFloat($number,
$optionen);
```

Die Formatierung einer Zahl als Währung übernimmt Zend_Currency. Auch deren Verhalten ist über Parameter beinflußbar, so kann z.B. bestimmt werden, ob die Kennzeichnung der Währungsangabe mittels Symbol, 3-stelligem Währungskürzel oder vollständigem Namen erfolgt. Darüber hinaus können über Zend_Currency auch Informationen abgerufen werden, etwa welche Währungen für ein bestimmtes Locale verwendbar sind oder in welchen Regionen die aktuelle Währung eingesetzt wird.

```
\\erstellen einer Zend_Currency Instanz für das aktuelle Locale
$currency = new Zend_Currency($locale);
\\die übergebene Zahl als Währung der Ausgabevariable zuweisen
$this->view->formattedNumber = $currency->toCurrency($number);
```

Mittels Zend_Date werden Datums- und Zeitangaben locale-sensitiv formatiert. Es verwendet zum Einlesen von lokalisierten Eingabestrings die Funktionen des bereits erwähnten Zend_Locale_Format. Weiters ist es bei der Ausgabe möglich, aus verschiedenen locale-sensitiven Ausgabeformaten für Datum und Zeit, die z.B. komplett numerisch sind oder lokalisierte Tages- und Monatsnamen oder Abkürzungen verwenden, zu wählen. Für die Date Instanzen können Zeitzonen gesetzt werden und die compare Methode liefert einen korrekten Vergleich zweier Datums- und Zeitangaben unter Berücksichtigung der Zeitzonen. Für die Validierung einer Datumseingabe per HTML Form steht ein locale-sensitiver Date Validator zur Verfügung. Eine Erweiterung von Zend_Date names Zend_Calendar, welche verschiedene Kalendersysteme unterstützen soll, ist, laut einem Eintrag des I18n Team Leaders des Zend Frameworks im Zend Framework Wiki, geplant.

```
\\erstellen einer Zend_Date Instanz für das gewählte Locale
$date1out = new Zend_Date(null, null, $locale);
\\setzten der gewählten Zeitzone
$date1out->setTimezone($timezone1);
\\setzen des validierten Eingabedatums
$date1out->set($datetime1);

//locale-sensitive Formatierung mit
//Ausgabeformat FULL (Tages- und Monatsnamen)
$this->view->formattedDateTime1 = $date1out->get(Zend_Date::DATE_FULL,
$locale)
." ".$date1out->get(Zend_Date::TIME_FULL, $locale);
```

Zend_Measure bietet Unterstützung Behandlung von Maßeinheiten. Zend_Measure stellt dabei keine Klasse dar, sondern besteht aus eine Ansammlung von Klassen, welche jeder einer Gruppe von Maßeinheiten zugeordnet ist. Unterstützt werden unter anderem Längenmaße, Gewicht, Temperatur, Beschleunigung aber auch Maßeinheiten für das Kochen und Ähnliches. Innerhalb der Gruppen kann zwischen verschiedenen Einheiten umgerechnet werden. Es ist möglich, Maßeinheiten aus lokalisierten Strings einzulesen, die lokalisierte Ausgabe ist allerdings erst für spätere Versionen des Frameworks geplant [Zend08].

4.4 Systemnachrichten

Tapestry/Java Für den Umgang mit Systemnachrichten bedient sich Tapestry grundsätzlich der Möglichkeiten von Java, erweitert diese aber für den Einsatz in Webapplikationen. Zur Auslagerung von Übersetzungen werden .properties Dateien verwendet. Jeder Tapestry Komponente, und das sind aufgrund der Tapestry Klassenhierarchie auch Pages, können properties Dateien zugeordnet werden. Die Namenskonvention ist dabei, dass die properties Datei den Namen der Komponente bzw. Page erweitert um [_] und einen Locale Identifier erhält [Shi04]. Dies kann folgendermaßen aussehen:

```
Home.page

Home.properties         #die default Übersetzung
Home_de.properties      #die deutsche Übersetzung
Home_fr_CA.properties   #die Übersetzung für französich in Canada
```

Darüber hinaus kann auch eine applikationsweite properties Datei erstellt werden, um Redundanzen mit mehrfach verwendeten Nachrichten zu vermeiden. Auf die ausgelagerten Strings kann sowohl im Programmcode der Komponononente als auch in deren Template zugegriffen werden. Der Zugriff aus dem Programmcode erfolgt über die Methode `getMessages().getMessage(String key)`, welche jede Komponente aufgrund des IComponent Interfaces enthält und die Nachricht als String zurückliefert. Innerhalb der Templates gibt es zwei Varianten:

```
01: <span jwcid="@Insert" value="message:tmptext">Text</span>
02: <span key="tmptext">Text</span>

#in der dazugehörigen properties Datei
tmptext=Ein beliebiger Text.
```

In Zeile 01 wird einer Insert Komponente mittels des Schlüsselwortes message die Nachricht mit dem Schlüssel tmptext zugewiesen, Zeile 02 zeigt einen Short-Cut der fast zum selben Ergebnis führt. Zu beachten ist allerdings, dass die Insert Komponente immer das umgebende span Tag bei der Ausgabe generiert, die zweite Variante nur dann, wenn darin andere Attribute (z.B. `class="..."`) enthalten sind. Weiters kann auch nur die Variante 01 so abgewandelt werden, dass Nachrichten mit dynamischen Elementen ausgegeben werden können:

```
<span jwcid="@Insert" value="ognl:messages.format('tmptextdyn', name)">
  Willkommen Name!
</span>

#in der dazugehörigen properties Datei
tmptextdyn=Willkommen {0}!
```

In diesem Fall würde der Platzhalter `{0}` durch den Inhalt der Page Property name ersetzt werden. Die Methode `format()` ist mehrfach überladen, um relativ einfach auch zwei oder drei Parameter übergeben zu können, darüber hinaus kann dann ein Array von Parametern verwendet werden.

Um Plural Formen zu behandeln, gibt es `java.text.ChoiceFormat`. Dieses bildet halb-offene Intervalle auf Strings ab. Diese Strings können als dynamische Parameter in Pattern eines `java.text.MessageFormat` Objektes eingebunden werden und auch selbst dynamische Parameter enthalten.

```
.properties Datei
# Ausgabe Anzahl der Suchergebnisse
searchMatchesPattern = There {0} of <b>{1}</b> found in the given Text.
noMatch = was <b>no</b> occurence
oneMatch = was <b>one</b> occurence
multipleMatches = were <b>{2}</b> occurences

# Java Source der Page

public String outputSearchMatches() {
this.getSearch().search();

//Message Format Objekt instanzieren und aktuelles Locale setzen
MessageFormat mf = new MessageFormat("");
mf.setLocale(this.getLocale());

//Limits der Intervalle des ChoiceFormat Objektes festlegen
double[] matchLimits = {0,1,2};

//Ausgabestrings zu den Intervallen festlegen und aus der Properties
```

```
//Datei nach aktuellem Locale holen
String [] matchStrings = {
    this.getMessages().getMessage("noMatch"),
    this.getMessages().getMessage("oneMatch"),
    this.getMessages().getMessage("multipleMatches")};

//ChoiceFormat mit Limits und AusgabeStrings instanzieren
ChoiceFormat cf = new ChoiceFormat(matchLimits, matchStrings);

//Muster des kompletten AusgabeStrings aus .properties Datei holen
// und an MessageFormat übergeben
String pattern = this.getMessages()
.getMessage("searchMatchesPattern");
mf.applyPattern(pattern);

//Format Objekte für die dynamischen Parameter setzen und
//an MessageFormat übergeben
Format[] formats = {cf, null,
NumberFormat.getInstance(this.getLocale())};
mf.setFormats(formats);

//werte für dynamische Parameter setzen und durch
//MessageFormat formatiert ausgeben
Object[] messageArguments = {null, this.getSearch().getSearchFor(),
     null};
messageArguments[0] = new Integer(this.getSearch().getFoundMatches());
messageArguments[2] = new Integer(this.getSearch().getFoundMatches());

return mf.format(messageArguments);
}
```

Auf diese Art können einfache Singular-Plural Regelungen, wie diese etwa im Englischen oder Deutschen auftreten, gehandhabt werden. Kompliziertere Regeln, die wie z.B. im Polnischen sich wiederholende Muster aufweisen können, werden nicht sinnvoll umgesetzt. Grundsätzlich gäbe es eine Gettext Implementierung für Java, welche auch für aufwendigere Pluralformen geeignet wäre, allerdings ist deren Einbindung in Tape-

stry nicht vorgesehen, d.h man würde die Funktionalität, die Tapestry zum Einbinden von Übersetzungen aus `.properties` Dateien in den Page Templates bereitstellt nicht verwenden könnnen.

ASP.NET/C# Das Externalisierungssystem von ASP.Net basiert auf XML Resource Dateien, welche Schlüssel/Wert Paare enthalten. Es wird dabei zwischen lokalen und globalen Resourcen und implizitem, explizitem und programmatischem Zugriff auf diese unterschieden [EvHaRa08] [Smit06]. Lokale Resourcen können für Pages, Controls und Master Pages erstellt werden und folgen der Namenskonvention:

```
Page: Default.aspx

Resource für die invariant Culture
Default.aspx.resx

Resource für neutral Culture de
Default.aspx.de.resx

Resource für specific Culture de-AT
Default.aspx.de.AT.resx
```

Auf lokale Resourcen kann durch alle oben genannten Varianten zugegriffen werden. Die implizite Variante erfolgt durch Verwendung von [Key.Property] als Schlüssel der Übersetzung. Dabei können nicht nur z.B. die Text Property eines Labels sondern alle Properties einer Control implizit mit einem locale-abhängigen Wert belegt werden. Um den impliziten Zugriff zu aktivieren, muss über ein meta Attribut ein Resourcekey definiert werden.

```
//[key.property]
searchButton.Text
searchButton.BackColor

<asp:Button ID="searchButton" runat="server" Text="DefaultText"
    meta:resourcekey="searchButton" />
```

Beim expliziten Zugriff kann einerseits ein beliebiger Schlüssel festgelegt werden und andererseits auch definiert werden, aus welcher Resource dieser ausgelesen werden kann.

```
<asp:Literal ID="Lit_Output" runat="server"
Text="<%$ Resources:i18n_dotnet, TOutput %>"></asp:Literal>
```

Auf globale Resourcen kann nur explizit oder programmatisch zugegriffen werden. Der programmatische Zugriff erfolgt über die Funktionen GetLocalResourceObject() und GetGlobalResourceObject(), welche zu einem Schlüssel die jeweilige Übersetzung liefert. Über diese Zugriffsvariante kann auch mit dynamischen Parametern in Übersetzungen umgegangen werden.

```
\\Übersetzung in .resx Datei
There were {0} occurences of {1} found in the given text.

\\auslesen der Übersetzung und
\\formatierung mit dynamischen
\\Parametern
object[] args = new object[]{ foundStartIndex.Count, searchFor };
String out = GetLocalResourceObject("TfoundMatches").ToString();
Label_output1.Text = String.Format(out, args);
```

Dynamische Parameter werden mittels nummer in der Übersetzung definiert und über die Funktion String.Format, welche den Ausgabestring und ein Array der Parameter erhält, ausgegeben. Für den Umgang mit Pluralformen ist durch das .net Framework bzw. ASP.Net keine Unterstützung vorgesehen.

Zend Framework/PHP Zend_Translate stellt die Lösung des Zend Frameworks in Hinblick auf das locale-sensitive Ausgeben von Strings dar und wird auch bei der Ausgabe von Systemnachrichten verwendet [Moeh08] [Zend08]. Es schafft eine einheitlich API, um auf verschiedene Quellformate mit Übersetzung zugreifen zu können und diese bei der Ausgabe zu verwenden. Unterstützt werden z.B. PHP Arrays, CSV-Dateien, Gettext .mo Dateien und einige XML Formate.

Grundsätzlich bietet das Framework große Flexibilität, wie die Übersetzungen organisiert werden. Es ist denkbar, die verschiedenen Formate zu mischen und für jeden Controller oder jeden View eine eigene Übersetzungsdatei festzulegen, oder sogar per Array die Übersetzungen in die Dateien aufzunehmen. Letzteres wird wenn überhaupt nur dann in Frage kommen, wenn die ProgrammierInnen selbst die Übersetzungen vornehmen. Bei großen Projekten wird man in Hinblick der Tatsache, dass die Übersetzungsdateien in der Regel an ein Subunternehmen weitergegeben werden, entweder eine

sinnvolle Ordnerstruktur zugrundelegen, oder aber eine Übersetzungsdatei pro Sprache für die ganze Applikation erstellen.

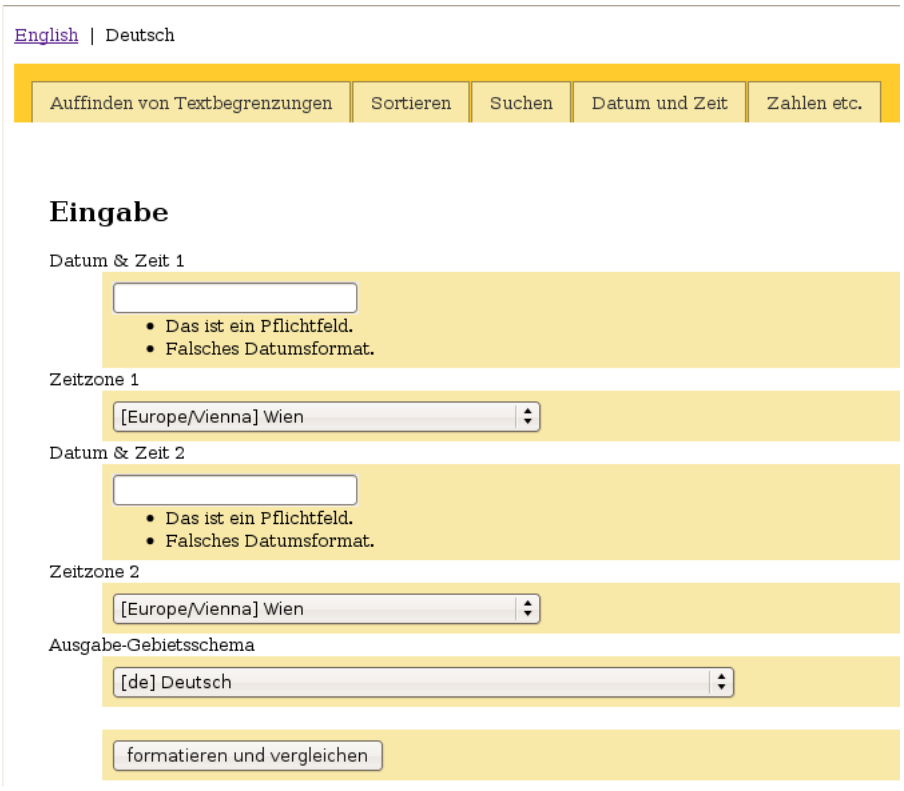


Abbildung 21: Zend/PHP: Beispiel Übersetzung mit Zend_Translate Deutsch

```
//eine Zend\_Translate Instanz erstellen, welche
//CSV Dateien verarbeitet - eine csv Datei mit
//zugehörigen Localestring zuweisen und setzten
//des Separator Zeichens
$translate = new Zend\_Translate(Zend\_Translate::AN\_CSV,
'../languages/en.csv',
'en',
array('separator' => ';'));

//eine weitere Übersetzung hinzufügen
$translate->addTranslation('../languages/de.csv', 'de');
//das aktuelle Locale fuer die Instanz festlegen
$translate->setLocale($locale);
```

```
//die Instanz in der Registry mit key Zend\_Translate
//setzen damit alle Objekte die Zend\_Translate
//verwenden auf diese Instanz zurückgreifen
Zend_Registry::set('Zend_Translate', $translate);
```

Es existiert die Möglichkeit, eine Instanz von Zend_Translate in der Zend_Registry, einer zentralen Datenstruktur auf die alle Elemente der Applikation Zugriff haben, unter dem vorgegebenen Schlüssel „Zend_Translate“ zu speichern, auf die dann alle Objekte, die Zend_Translate verwenden, zugreifen, falls keine andere Instanz manuell festgelegt wird. Durch diese Automatisierung können z.B. Fehlermeldungen von Form Validator Objekten zentral festgelegt werden.

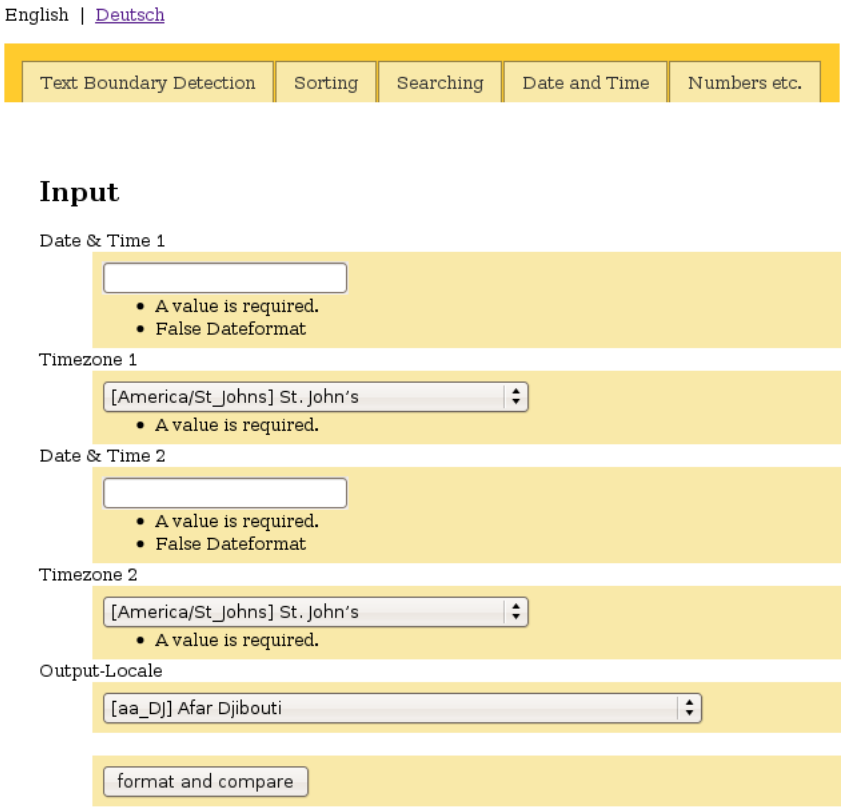


Abbildung 22: Zend/PHP: Beispiel Übersetzung mit Zend_Translate Englisch

Die Abbildungen 21 und 22 zeigen die Übersetzung einer Seite inklusive der Fehlermeldungen der Formelemente bei falschen Eingaben. In den Views kann über eine Funktion `translate(String key)` auch explizit auf Übersetzungen zugegriffen werden.

Die translate Methode kann auch mit dynamischen Parametern umgehen. Diese können entweder als Liste von Parametern oder als Array übergeben werden. Die Übersetzung muss dabei folgendem Syntax entsprechen:

```
$this->translate("%1\$s liegt vor  %2\$s.",
                 $datetime1,
                 $datetime2);
```

Der Umgang mit Pluralformen ist bei keinem der verwendbaren Adapter möglich, auch nicht mit dem gettext Adapter, dieser dient nur zur Verwendung des .mo Dateiformates, welches zwar nicht für Menschen lesbar ist, aber einige Performancevorteile bietet. Es existiert eine gettext Variante für PHP, allerdings wird in der Programmers Reference des Zend Frameworks empfohlen, diese nicht zu verwenden, da sie einige Nachteile wie die fehlende Threadsicherheit aufweist.

4.5 Benutzerschnittstelle / Layout

Tapestry/Java Im Bereich Benutzerschnittstelle/Layout liegen die Schwerpunkte des Tapestry Frameworks und dementsprechend stellt es auch einige nützliche Funktionen im Hinblick auf Internationalisierung bzw. Lokalisierung bereit. Diese beziehen sich hauptsächlich auf Automatisierungen, wenn bestimmte Konventionen eingehalten werden.

Es kann für jedes Template einer Komponente oder einer Page lokalisierte Templates geben (vgl. Abbildung 23). Diese müssen nach der Konvention Dateiname erweitert um Locale erstellt werden, um dann durch Tapestry nach dem gesetzten Locale verwendet zu werden. Alternative Template System sind nicht vorgesehen, auch deshalb weil das Template- und Komponentensystem einen Großteil des Tapestry Frameworks ausmacht.

Bilder oder z.B. auch CSS Dateien werden als sog. Assets in der zur Komponente bzw. Page gehörenden XML Datei definiert. Auch für Assets werden durch Tapestry, falls vorhanden, und der schon bei den Page Templates beschriebenen Namenkonvention folgend, lokalisierte Versionen automatisiert verwendet. Es kann ein Array von Assets, die CSS Dateien kapseln, an die stylesheets Property der Shell Komponente übergeben werden. Die einzelnen Assets werden dann durch die beschriebene Automatisierung durch Versionen für das jeweilige Locale ersetzt, falls diese vorhanden sind.

Die Ajax-Unterstützung wird durch Einbindung des Dojo Toolkit realisiert, welches selbst I18n Aspekte berücksichtigt. So implementiert das Dojo Toolkit ein eigenes Locale

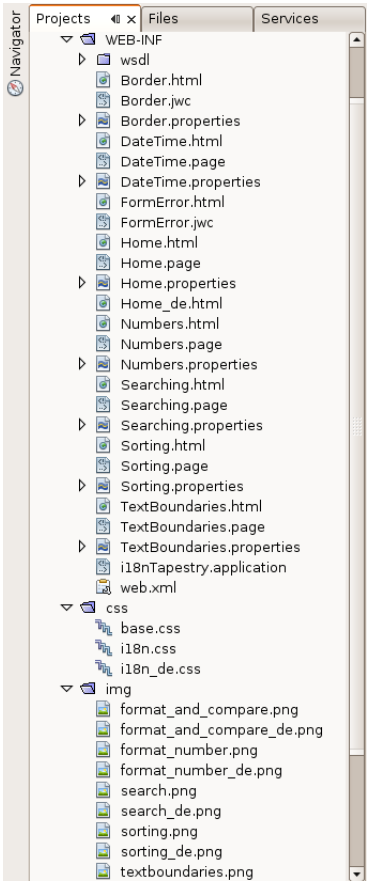


Abbildung 23: Tapestry/Java: Beispiel Verzeichnisstruktur Templates, CSS, Bilder

Model und bringt Locale Daten aus dem Unicode CLDR Projekt mit, um diese bei der client-seitigen Ein- und Ausgabe von Datums- und Zeitangaben, Zahlen und Währungen zu verwenden [Dojo08]. Weiters wird ein Resource Bundle Mechanismus auf Basis von JSON (JavaScript Object Notation) verwendet, um Meldungen lokalisieren zu können.

Da das Tapestry Framework sehr stark komponenten-orientiert ist, auch z.B. Pages werden nicht nur aus Komponenten aufgebaut, sondern sind nach der Objekthierarchie selbst Komponenten, ist es möglich, eigene wiederverwendbare Komponenten zu erstellen.

```
//Page oder Component Spezification
<asset name="baseStylesheet" path="/css/base.css"/>
<asset name="localeStylesheet" path="/css/i18n.css"/>

//Page oder Component Template
<html jwcid="@Shell" title="ognl:title"
 stylesheets="ognl:{assets.baseStylesheet, assets.localeStylesheet}">
```

ASP.NET/C# Um Bilder oder andere Dateien locale-sensitiv einzubinden, können Pfadangaben in den schon weiter oben beschriebenen Resource Dateien hinterlegt werden. Um z.B. culture-abhängig CSS Dateien einzubinden, kann folgender Code verwendet werden:

```
if(GetLocalResourceObject("css_i18n") != null) {

HtmlLink i18nCSS = new HtmlLink();
i18nCSS.Href = "/css/" + GetLocalResourceObject("css_i18n").ToString();
i18nCSS.Attributes.Add("rel", "Stylesheet");
i18nCSS.Attributes.Add("type", "text/css");

Page.Header.Controls.Add(i18nCSS);
}
```

Wird dieser Code in der `Page_Load` Funktion des Master Templates ausgeführt, wird in den Header jeder Seite, die das Master Template erweitert, ein link Element zu einer CSS Datei eingefügt, falls für die aktuelle Culture oder eine der Fallback Cultures ein Dateiname in der Resourcen Datei existiert. Ein Mechanismus, um verschiedensprachige

Templates für eine Seite zu erstellen, und diese dann je nach aktueller Culture zu verwenden, ist von Seiten des Frameworks nicht vorgesehen. Um allerdings statische Teile einer Seite zu lokalisieren, existiert die <asp:localize> Control, welche einen beliebigen Text je nach Culture aus einer Resourcen Datei holt und ausgibt.

Zur Integration von AJAX existiert ASP.Net AJAX einer Erweiterung von ASP.Net, die Controls wie den ScriptManager oder UpdatePanel bereit stellt, welche die Realisierung von Ajax Funktionen in Webseiten erleichtern und diverse Browsereigenheiten abstrahieren. In einem Artikel der MSDN wird beschrieben, wie Resource Dateien mit JavaScript genutzt werden können, die Realisierung dieser Vorgehensweise ist allerdings nur mit Visual Studio 2008 und nicht mit der frei erhältlichen Visual Web Developer Express Edition 2008 möglich.

ASP.Net enthält einige Komponenten wie Labels, Textboxes, DataGrids und weitere, welche als Controls bezeichnet werden und die Bausteine einer ASP.Net Applikation bilden. Es ist möglich, diese zu sog. CompositeControls zusammenzufassen oder auch eigene Controls zu erzeugen, welche dann in der Applikation wiederverwendet werden können.

Zend Framework/PHP Auch für die Lokalisierung der Benutzerschnittstelle kommt das bereits beim Abschnitt Systemnachrichten beschriebene Zend_Translate zum Einsatz, um diverse Ausgabestring wie etwa Menüpunkte und ähnliches je nach Locale auszugeben. Für die Lokalisierung kompletter Templates wird von Seiten des Frameworks keine Unterstützung geboten. Ajax Integration steht ab der Version 1.6 des Zend Frameworks, durch die Integration des weiter oben schon erwähnten Dojo Toolkits, zur Verfügung.

4.6 Resumeé

Alle Aufgaben konnten nur mit der Kombination **Tapestry** und Java erfüllt werden, was vor allem auf die weitreichende Unicode-Unterstützung von Java zurückzuführen ist. Die Kombination Tapestry/Java provitiert auch davon, dass Java seit der Version 1.1 kontinuierlich auch im Hinblick auf Internationalisierung entwickelt wurde und viele Funktionen bereits ausgereift bereitstellt. Tapestry erweitert die in Java vorhandenen Möglichkeiten geschickt um einige Funktionen im Hinblick auf Webapplikationen. So z.B. ist es oft gefordert oder im Hinblick auf Usability zu empfehlen, eine Auswahlmöglichkeit der Sprache der Benutzeroberfläche durch die BenutzerInnen auswählen zu

lassen. Das Tapestry Framework speichert das aktuell zu verwendende Locale in der sog. Engine und setzt diese automatisch nach der vom Browser (über den HTTP Header Accept_Language) angeforderten Locale. Wird allerdings das Locale der Engine in der Applikation gesetzt, erstellt das Framework ein Cookie und liest dieses bei zukünftigen Anfragen wieder aus. Wird die Sprache manuell geändert, kann die aktuelle Page allerdings nicht sofort in der neuen Sprache dargestellt werden, da diese durch den hohen Aufwand der Erstellung für ein Locale erstellt und in einem Page Pool gehalten werden. Deshalb muss nach dem Umschalten der Sprache entweder der Tapestry Request Cycle zurückgesetzt und die Page neu erstellt werden, oder über eine Zwischenseite, die etwa die Umstellung auf die neue Sprache bestätigt, ein neuer Request auf die Seite erfolgen. Diese Automatisierungen von oft auftretenden Problemen bei der Internationalisierung von Webapplikationen gepaart mit den ausgereiften Funktionen der Java Laufzeit Umgebung ergeben eine gute Grundlage zur Entwicklung internationalisierter Applikationen.

Bei **ASP.Net** fällt vor allem die Unterstützung vieler Kalendersysteme im Vergleich zu den beiden anderen Frameworks auf. Auch bei der Sortierung werden für manche Cultures alternative Sortierungen angeboten, etwa für Spanisch, welches eine moderne und eine traditionelle Sortierung definiert. Diese sind vom Framework fix vorgegeben und können nicht angepasst werden, wie das etwa durch das Tayloring des Unicode Collation Algorithm angedacht wäre, welcher allerdings von .Net nicht implementiert wird. Die Realisierung der oben beschriebenen Funktionalität, die Sprache der Benutzerumgebung manuell zu setzen, erfordert etwas mehr Handarbeit als dies bei Tapestry der Fall war, es wird aber mit der Page Funktion initialize_Culture ein Punkt in der Pipeline vorgegeben, an dem dies erfolgen kann. Die Verwendung von .resx Dateien, um Übersetzungen zu externalisieren, ist gut in das Framework integriert und bieten mit den möglichen Zugriffsvarianten (implizit, explizit und programmatisch) auch einiges an Flexibilität. Negativ fällt auf, dass bei der Referenzierung eines nicht existierenden Schlüssels, eine Exception geworfen wird, anstatt einfach den Schlüssel auszugeben.

Das **Zend Framework** kann im Unterschied zu Tapestry nicht auf eine Programmiersprache bzw. eine Laufzeit-Umgebung zurückgreifen, die bereits ein breite I18n Unterstützung bietet. Dies merkt man auch an der Tatsache, dass z.B. die Zend_Locale Klasse nicht mit den Locale Funktion, die PHP enthält, arbeitet, sondern eine eigene Implementierung darstellt. Das Framework bringt auch eigene Locale Daten mit, die auf den Daten des Unicode CLDR Projektes basieren. Man merkt dem Framework an, welches seit 02. September 2008 in der Version 1.6 vorliegt, dass es sich doch noch stark in der Weiterentwicklung befindet und auch einige geplante Projekte die Internationali-

sierung bzw. Lokalisierung betreffen zugunsten anderer Entwicklungsziele zurückgestellt werden, etwa die schon weiter oben erwähnte Zend_Calendar Klasse zur Unterstützung verschiedener Kalendersysteme. Nichts desto trotz bietet das Framework einige interessante und ausgereift wirkende Ansätze, wie die Bereitstellung einer einheitlichen API durch Zend_Translate, um auf verschiedene Quellformate von Übersetzungen zugreifen zu können.

4.7 Vergleichstabellen

	Tapestry	**ASP.NET**	**Zend Framework**
Programmiersprache	Java	C#	PHP
Architekturmuster	Model-View-Controller	WebForms (Code-behind-Model)	Model-View-Controller
Server	Servlet-Container (z.B. Tomcat)	Internet Information Server	Webserver mit PHP Unterstützung
Datenbankanbindung	JDBC oder zusätzliches Framework	ADO.Net, LINQ	Zend_Db

Tabelle 23: Vergleichskriterien Allgemein

	Tapestry	**ASP.NET**	**Zend Framework**
Unicode Support	erweiterter Support ab Java 1.5 Unicode 4.0	erweiterter Support	UTF-8 Codierung
Textbegrenzungen			
Buchstabengrenzen	ja	ja	nein
Wortgrenzen	ja	nein	nein
Satzgrenzen	ja	nein	nein
locale-sensitive Anpassungen	ja	nein	nein
Sortierung			
locale-sensitive Anpassungen	Ja / vorgegeben aber anpassbar	Ja / vorgegeben	nein
Kombinierende & äquivalente Unicode Zeichen	Normalisierung	Normalisierung	nein
Suchen			
locale-sensitive Anpassungen	Ja / vorgegeben aber anpassbar	Ja / vorgegeben	nein
Kombinierende & äquivalente Unicode Zeichen	Normalisierung	Normalisierung	nein
Regular Expressions			
RexEx Maschine vorhanden	ja	ja	ja
RegEx bei Suche oder Validierung verwendbar	ja	ja	ja
RegEx Maschine Unicode-aware	ja java.util.regex Basic Unicode Support	ja .net Framework Basic Unicode Support	ja PCRE (Perl Compatible Regular Expressions) Basic Unicode Support

Tabelle 24: Vergleichskriterien Text/Sprache

	Tapestry	**ASP.NET**	**Zend Framework**
Datums und Zeitangaben			
locale-sensitive Formatierung	ja	ja	ja
verschiedene vordefinierte Ausgabeformate	Datum, Zeit, Formate beliebig zu DatumZeit kombinierbar	Datum, Zeit, DatumZeit vorgegeben	Datum, Zeit, Formate beliebig zu DatumZeit kombinierbar
welche Kalendersysteme werden unterstützt	gregorianisch, buddistisch, japanisch imperial	ChineseLunisolar, EastAsianLunisolar, Gregorian, Hebrew, Hijri, Japanese, JapaneseLunisolar, Julian, Korean, KoreanLunisolar, Persian, Taiwan, TaiwanLunisolar, ThaiBuddhist, UmAlQura	gregorianisch
Zeitzonen	ja	ja	ja
Datumsvergleich unter Berücksichtigung der Zeitzonen	ja	ja	ja
locale-abhängige Eingabeformate	ja	ja	ja
locale-abhängige Validierung	ja	ja	ja
GUI Komponente zur Datumsauswahl (Datechooser)	ja	ja	nein
Zahlen und Währungen			
locale-sensitive Formatierung	ja	ja	ja
verschiedene vordefinierte Ausgabeformate	ja	ja	ja
Maßeinheiten			
locale-sensitive Formatierung	nein	nein	nein aber geplant
ist Umrechnung möglich	nein	nein	ja

Tabelle 25: Vergleichskriterien Formatierung

	Tapestry	**ASP.NET**	**Zend Framework**
Externalisierungssystem			
Framework bzw. programmierspracheneigenes System	ja	ja	ja
Umgang mit Pluralformen	ja einfache Pluralformen	nein	nein
Handhabung von dynamischen Elementen der Nachrichten	ja	ja	ja
gibt es alternative Externalisierungs-Systeme	nein	nein	nein
GNU gettext Unterstützung	Ja, gettext f. Java	nein (nur f. Mono/Linux)	als Adapter für Zend_Translate
Umgang mit Pluralformen	ja	-	nein
Handhabung von dynamischen Elementen der Nachrichten	ja	-	ja
Dateiformate			
verwendbare Dateiformate	.properties	.resx	.mo, .csv, verschiedene XML-Formate
Editoren für Externalisierungsdateien verfügbar	Texteditor	Visual Studio bzw. Visual Web Developer	z.B. poedit f. .mo, Texteditor f. csv

Tabelle 26: Vergleichskriterien Systemnachrichten

	Tapestry	**ASP.NET**	**Zend Framework**
Externalisierung von Bildern und Multimediainhalten (Sound/Video)	Ja per Assets	Externalisierung der Pfade	Externalisierung der Pfade
AJAX			
AJAX Integration	Dojo Toolkit	ASP.NET AJAX	Dojo Toolkit ab Version 1.6
Internationalisierung von JavaScript	ja	ja	ja
locale-sensitive Formatierung (z.B. Datum, Währung)	ja	ja	ja
Externalisierung von Strings möglich	ja Dojo per JSON	nur Visual Studio	ja Dojo per JSON
Templates			
Template System	ja	ja	ja
vom Framework vorgegeben	ja	ja	ja
externes System nutzbar	nein	nein	ja (z.B. Smarty)
verschiedensprachige Templates für eine Seite möglich	ja	nein	nein
Anzeige nach Locale	ja	-	-
Verhalten bei Anforderung eines nicht vorhandenen Locales	fallback Strategie bis zum default Locale	-	-
Modularisierung			
GUI Komponentenbasierend	ja	ja, Controls	nein
konfigurierbare Komponenten	ja	ja	-
Erstellen eigener GUI Komponenten möglich	ja	ja	-

Tabelle 27: Vergleichskriterien Benutzerschnittstelle / Layout

5 Zusammenfassung und Ausblick

Lokalisierung kann einen wichtigen Beitrag dazu leisten, die Akzeptanz eines Softwareproduktes durch die BenutzerInnen zu erhöhen und dadurch auch den wirtschaftlichen Erfolg desselben positiv zu beeinflussen, da es viele, der immer mehr werdenden, Computer- und InternetbenutzerInnen vorziehen Software in ihrer Muttersprache zu verwenden. Webinhalte sind dadurch besonders betroffen, da diese per se global verfügbar sind und nicht nur für eine bestimmte Region veröffentlicht werden, von Zensurmaßnahmen einmal abgesehen. Die Anpassung an die Sprache der BenutzerInnen ist dabei nur ein Aspekt. Um diesen einen vertrauten Eindruck zu vermitteln, müssen auch regionsspezifische Sortierungen, Formatierungen und andere Eigenheiten berücksichtigt werden. Die Internationalisierung beschäftigt sich mit der Anforderung, diese Anpassungen an verschiedene Regionen und Kulturen möglichst effizient und ohne eine Neukompilierung des Produktes zu bewerkstelligen. Weiters muss auch der Tatsache Rechnung getragen werden, dass die EntwicklerInnen in der Regel nicht mit allen zu unterstützenden Kulturen bzw. deren Unterschiede zur eigenen Kultur vertraut sind. Daher müssen diese Eigenheiten durch eine internationalisierte Software bzw. Programmierbibliothek mittels einheitlicher API abstrahiert werden.

Ein grundlegendes Problem für die Internationalisierung stellen die verschiedenen Schriftsysteme dar, für die sich unterschiedlichste Zeichensätze und -kodierungen ausgebildet haben. Mit dem Unicode-Projekt wurde daher versucht, einen einheitlichen Zeichensatz zu etablieren. Dies kann aufgrund der Tatsache, dass laut Statistiken von Google Ende 2007 erstmals mehr Webseiten in UTF-8 kodiert waren als in anderen Zeichensätzen, auch als gelungen bezeichnet werden. Neben der computer-internen Repräsentation der Zeichen ist auch die Darstellung bei der Ausgabe von Relevanz, da Schriftsysteme einige Unterschiede aufweisen, die für eine korrekte Darstellung eingehalten werden müssen. Weitere Aufgaben, die direkt mit dem Schriftsystem zusammenhängen sind die Sortierung, das Suchen und das Auffinden von Textbegrenzungen. Unicode ist deshalb nicht nur ein Zeichensatz und die dazugehörigen Kodierungen, sondern enthält auch Informationen zu den einzelnen Zeichen, Sortierschlüssel und darüber hinaus auch Beschreibungen von Algorithmen (z.B. den Unicode-Collation-Algorithm für den Textvergleich).

Formatierungen wie das Darstellen von Datums- und Zeitangaben oder von Zahlen sind kulturell ebenfalls sehr unterschiedlich. Die BenutzerInnen sollten die Möglichkeit

haben mit ihren gewohnten Formatierungen zu arbeiten, sowohl was die Eingabe solcher Werte betrifft, aber auch die Ausgabe sollte den jeweiligen Richtlinien entsprechen. Bei Datums- und Zeitangaben geht es auch darum, bei Ausgabeformaten mit Tages- oder Monatsname jeweils korrekte Übersetzungen auszugeben. Formatierungsrichtlinien und Übersetzungen für Elemente von Datums- und Zeitangaben und Zahlen bzw. Währungen werden als Locale-Daten zusammengefasst und in Laufzeit-Umgebungen und Frameworks integriert. Das Unicode Common Locale Data Repository stellt ein Projekt dar, welches Locale-Daten frei zu Verfügung stellt. Diese Locale-Daten werden in der Regel von locale-sensitiven Funktionen und Objekten verwendet, um Eingabe nach Locales zu parsen bzw. Ausgaben zu formatieren.

Große Unterschiede finden sich im Bereich der Farbgebung bzw. der Bedeutung von Farben in verschiedenen Kulturen. Mit ein und derselben Farbgebung kann man bei BenutzerInnen aus verschiedenen Kulturkreisen sehr unterschiedliche Assoziationen auslösen. Was in der einen Kultur Vertrauen und Sicherheit vermittelt, könnte in einer anderen Kultur das genau Gegenteil bewirken. Ähnlich verhält es sich mit der Verwendung von Zeichen und Symbolen. Auch das gewünschte Layout bzw. die Ausrichtung der Benutzerschnittstelle wird z.B. durch die Ausrichtung des Schriftsystems beeinflusst. Jemand aus einer Kultur, die ein Schriftsystem verwendet, in welchem Text von rechts nach links verläuft, erwartet in der Regel auch die Anordnung und das Aussehen der Benutzerschnittstelle in dieser Art. Im Bereich der Webapplikationen kommen hierfür bewährte Technologien zum Einsatz. Mit Cascading Stylesheets (CSS) können Farb- und Layoutgestaltungen vom Inhalt getrennt werden. Schafft man nun eine Möglichkeit, Stylesheets je nach gewähltem Locale bei der Generierung der angeforderten Seite einzubinden, kann somit das Layout je nach Locale geändert werden. Zur Festlegung der Ausrichtung enthält HTML das `direction` Attribut, welches bei Webapplikationen ebenfalls dynamisch, je nach Locale gesetzt werden kann, um die Ausrichtung der kompletten Seite bzw. einzelner Elemente zu beeinflussen. Kulturelle Unterschiede gehen allerdings über Farben und Layout hinaus. Basierend auf der Arbeit von Geert Hofstede, welcher 5 kulturelle Dimensionen zur Charakterisierung von Kulturen festgelegt hat, existieren weitere Arbeiten (z.B. [SinPer05]), welche sich damit beschäftigen, wie Webseiten aufgebaut bzw. welche Funktionen diese bieten sollten, um für Kulturen, die einer der 5 Dimensionen zugeordnet werden können, ansprechend zu sein.

Konzepte zur Lösung der beschriebenen Probleme sind Locales, das Externalisieren von zu lokalisierenden Inhalten und locale-sensitive Funktionen und Objekte. Das Locale, also Identifier für Sprachen bzw. Kombinationen aus Sprache und Region, stellt

den zentralen Dreh- und Angelpunkt dar. Es dient zur Bestimmung der gewünschten Sprache bzw. regionsspezifischer Einstellungen, d.h. es bestimmt, welche Übersetzungen aus den Externalisierungsdateien verwendet wird, oder welche Locale-Daten, also welche Datums- und Zahlenformate, möglich sind. Das Ziel bei der Externalisierung von zu lokalisierenden Inhalten, ist es eine ausführbare Datei zu erhalten, die keine kulturellen Voreinstellungen enthält, sondern diese zur Laufzeit aus den Externalisierungsdateien nach Bedarf einliest. Daraus ergibt sich einerseits der Vorteil, dass neue Übersetzungen oder andere externalisierte Daten hinzukommen können, ohne die gesammte Applikation neu kompilieren zu müssen, andererseits können so Übersetzungsdateien an die ÜbersetzerInnen weitergegeben werden, die keinen Source Code enthalten und somit vermieden werden, dass z.B. Schlüsselwörter mitübersetzt werden und dann zu Fehlern bei der Kompilierung oder zur Laufzeit führen. Aber auch die ÜbersetzerInnen müssen sich nicht durch Source Code arbeiten, mit dem sie in der Regel nicht vertraut sind. Danach wird eine Gliederung von Webapplikationen in standardisierte, lokalisierte, internationalisierte und kulturell angepasste beschrieben. Standardisiert heißt, keinerlei Berücksichtigung von Lokalisierung oder Internationalisierung. Lokalisiert bedeutet für ein oder mehrere Locales angepasst allerdings so, dass eine Anpassung an ein weiteres Locale Eingriffe in die Applikation erfordert. Internationalisierte Webapplikationen verwenden Konzepte und Prinzipien der I18n und sind daher so angelegt, dass ein weiteres Locale mit möglichst geringem Aufwand unterstützt werden kann. Kulturell angepasst bedeutet, dass nicht nur auf Übersetzung, Formatierung und leichte Anpassbarkeit an weitere Locales Rücksicht genommen wurde, sondern, dass kulturelle Charakteristika der Zielkulturen in das Layout und/oder den Funktionsumfang für die unterstützten Locales eingeflossen sind.

Den Abschluss des Theorieteiles bilden Ausführungen über die Vorraussetzungen und Auswirkungen der Internationalisierung von Webapplikationen. Dabei ergibt sich, dass die Internationalisierung grundsätzlich in jeder Schicht einer Applikationen eine mehr oder weniger große Rolle spielt. Angefangen bei der Unicodefähigkeit der einzelnen Schichten über die Sortierung beim Auslesen aus einer Datenbank bis hin zur Formatierung diverser Werte und der Ausgabe von Übersetzungen im Presentation Layer. Die Auswirkungen der Internationalisierung sind etwa die Verlängerung von Release-Zyklen durch die notwendige Übersetzung, die in der Regel durch ein anderes Unternehmen erfolgt, und die notwendigen zusätzlichen Tests.

Auf Basis der Erkenntnisse des Theorieteils werden einerseits Vergleichskriterien für die Fähigkeiten von Webentwicklungsframeworks erstellt und andererseits ein Beispiel

konzipiert, durch dessen Implementierung die Möglichkeiten der drei Frameworks Tapestry/Java, ASP.Net/C# und Zend Framework/PHP praktisch getestet werden. Die Kriterien gliedern sich in die folgenden Kategorien: Die Kategorie "Allgemein" bietet einen Überblick, welche Infrastruktur das jeweilige Framework benötigt. "Text/Sprache" zielt darauf ab, die Unicode Fähigkeiten zu vergleichen. Unter dem Überbegriff "Formatierungen" werden die Möglichkeiten zur Ein- und Ausgabe von Datums- und Zeitwerten und Zahlen zusammengefasst. "Systemnachrichten" widmet sich dem Umgang der Frameworks mit Status- und Fehlermeldungen bzw. generell der Externalisierung von Übersetzungen. Die Kategorie "Benutzerschnittstelle/Layout" beschäftigt sich mit den Möglichkeiten, das Layout locale-sensitiv zu verändern etwa durch verschiedensprachige Templates oder dem Externalisieren von Bildern oder anderen Multimediainhalten.

Der Vergleich zeigt, dass vor allem bei der erweiterten Unicode-Unterstützung, also nicht nur dem Umgang mit Unicode-Kodierungen, sondern auch der Nutzung von Zusatzinformationen (z.B. Character Properties, Default Collation Element Table etc.), doch sehr große Unterschiede auftreten. Auch Unterstützung für den Umgang mit Pluralformen, ist wenn überhaupt nur für simple Pluralformen wie im Deutschen oder Englischen, nicht aber für komplexere wie im Polnischen vorhanden. Das Formatieren von Datums- und Zeitangaben und von Zahlen und Währungen beherrschen alle verglichenen Frameworks, bei der Unterstützung unterschiedlicher Kalendersysteme sieht dies schon anders aus.

In Zukunft wird Unicode UTF-8 im Webbereich seine Stellung als Standardkodierung festigen und weiter ausbauen. Dies ist auch an der Tatsache abzuleiten, dass PHP, welches derzeit keine weitreichende Unicode-Unterstützung enthält, diese in der kommenden Version 6 enthalten und Unicode als interne Kodierung verwenden wird. Einen wichtigen Punkt bei der Entwicklung internationalisierter Applikationen ist es, sich bewusst zu machen, dass viele Dinge, die man aus seiner eigenen Kultur als selbstverständlich annimmt, in anderen Kulturen alles andere als selbstverständlich sind. Daher sollte man sich während der Planungs- bzw. Designphase über die zu unterstützenden Kulturen informieren, um deren Eigenheiten von Anfang an miteinplanen zu können. Für die Formatierung ist man von den Locale-Daten des Frameworks oder der Programmiersprache abhängig, da es ein nicht zu unterschätzender Aufwand wäre, für jedes zu unterstützende Locale und in weiterer Folge für eventuell später hinzukommende Locales diese Informationen zusammenzutragen. Man sieht allerdings am Beispiel des noch verhältnismäßig jungen Zend-Frameworks, dass Internationalisierung und Lokalisierung ein wichtiges Thema sind und diverse Erweiterungen der bestehenden Funktionalität

geplant sind. Vermutlich wird es in Zukunft vermehrt gefordert sein, Webapplikation internationalisiert zu entwickeln und für verschiedene Zielregionen zu lokalisieren.

Literatur

[BarBad98] Wendy Barber and Albert Badre, Culturability: The Merging of Culture and Usability, 4th Conference on Human Factors & the Web 1998/06/05, cited 20080530

http://zing.ncsl.nist.gov/hfweb/att4/proceedings/barber/index.html

[Ber98] Tim Berners-Lee, The World Wide Web: A very short personal history, written 1998/05/07, cited 20080505

http://www.w3.org/People/Berners-Lee/ShortHistory.html

[Bow03] Douglas Bowman, Sliding Doors of CSS, published 2003-10-20, cited 20080706

http://alistapart.com/articles/slidingdoors/

[Coop07] Alan Cooper, Robert Reimann and David Cronin, About Face 3 The Essentials of Interaction Design, Wiley 2007

[DeiCza01] Andrew Deitsch & David Czarnecki, JAVA Internationalization O'Reilly März 2001

[Ditt02] Kerstin Dittert, Architektur internationaler Anwendungen, OBJEKTspektrum, April 2002

http://www.oocon.de/pdf/architekturIntAnw.pdf

[Doel08] Mirko Dölle, Sprachgenie - Mehrsprachige Linux-Programme und -Skripte mit Gettext, c't Heft 08/2008 p. 184-187

[Dojo08] Dojo The Javascript Toolkit, The Book of Dojo, cited 20080715

http://dojotoolkit.org/book/dojo-book-1-0

[Drea07] Ulrich Drepper et. al., GNU gettext tools, version 0.17 Native Language Support Library and Tools Edition 0.17, 31 October 2007

[Dus03] Schahram Dustdar, Harald Gall, Manfred Hauswirth, Software Architekturen für Verteilte Systeme, Springer 2003

[Esse00] Bert Esselink, A Practical Guide to Localization, John Benjamins Publishing Company 2000

[EvHaRa08] Bill Evjen, Scott Hanselman, Devin Rader, Professional ASP.NET 3.5 in C# and VB, Wiley 2008

[Fau1880] Carl Faulmann, Schriftzeichen und Alphabete aller Zeiten und Völker, 1880 Nachdruck matrixverlag 2004

[Frie08] Jeffrey E. F. Friedl, Reguläre Ausdrücke, O'Reilly 3. Auflage 2008

[GaHeJoVl95] Erich Gamma et al., Design Patterns: Elements of reusable Object-oriented Software, Addison Wesley 1995

[Har07] Yannis Haralambous, Fonts & Encodings, O'Reilly 2007

[HeBus02] He Z., Bustard D.W. and Liu X., Software Internationalisation and Localisation: Practice and Evolution ACM 2002

[Hein04] Andreas M. Heinecke, Mensch-Computer-Interaktion, Fachbuchverlag Leipzig 2004

[Hofs05] Geert Hofstede and Gert Jan Hofstede, Cultures and Organizations Software of the Mind, McGraw-Hill 2005

[ICU08] The ICU Project cited 20080704

http://demo.icu-project.org/

[Kapp04] Gerti Kappel et. al., Web Engineering, dpunkt.verlag 2004

[Kub06] Kubota Tomohiro, Introduction to I18n, 17. June 2006, cited 20080212

http://www.debian.org/doc/manuals/intro-i18n/intro-i18n.pdf

[LISA08] LISA:*Localization Industry Standards Association*, cited 20080315

http://www.lisa.org

[Lun02] Ken Lunde, CJKV Information Processing, O'Reilly Oktober 2002

[Moeh08] Carsten Möhrke, Zend Framework Das Entwickler-Handbuch, Galileo Computing 2008

[RhDaLo05] George Rhoten, Mark Davis, Steven Loomis, CLDR 1.3: Overiew and What's new, 28th Internationalization and Unicode Conference 2005

http://unicode.org/cldr/data/docs/presentations/cldr_overview.ppt

[Sach08] Sachdev Puneet, Web Application Internationalization - The key is knowing your application and future audience, cited 20080620

http://www.iasahome.org/c/portal/layout?p_l_id=PUB.1.357

[Shi04] Howard M. Lewis Ship, Tapestry in Action, Manning 2004

[SinPer05] Nitish Singh & Arun Pereira, The culturally customized Web Site - Customizing Web Sites for the global Marketplace Elsevier 2005

[Smit06] Guy Smith-Ferrier, .NET Internationalization - The Developers Guide to Building Global Windows and Web Applications, Addison-Wesley 2006

[Stea04] Buck Stearns et. al., e-business Globalization Solution Design Guide: Getting Started, IBM Redbooks last updated 2004-02-13

[Uni06] The Unicode Consortium, The Unicode Standard 5.0, Addison-Wesley Oktober 2006

[UniTS08] Mark Davis, Ken Whistler, Unicode Technical Standard #10 Unicode Collation Algorithm, Version 5.1.0, last update: 2008.03.28, cited 20080630

http://www.unicode.org/reports/tr10/

[UniTS05] Mark Davis, Unicode Technical Standard #18 Unicode Regular Expressions, Version 11, last update: 2005.05.01, cited 20080714

http://www.unicode.org/reports/tr18/

[CLDR08] The Unicode Consortium, Common Locale Data Repository, cited 20080626

http://www.unicode.org/cldr/

[Yun02] John Yunker, beyond borders web globalization strategies, New Riders 2003

[W3C08] Fielding, et al., RFC 2616, cited 20080707

http://www.w3.org/Protocols/rfc2616/rfc2616-sec14.html

[W3Ccss08] Richard Ishida, Background images that support localization, www.w3c.org 2007-07-03, cited 20080706

http://www.w3.org/International/questions/qa-resizing-backgrounds

[W3Cws08] Addison Philips, et al., Web Services Internationalization (WS-I18N) W3C Working Draft 15 April 2008, cited 20080708

http://www.w3.org/TR/ws-i18n/

[WirBau08] Ralf Wirdemann, Thomas Baustert, Rapid Web Development mit Ruby on Rails 3. überarbeitete Auflage, Hanser 2008

[Zend08] Zend Framework, Programmers Reference Guide 1.5.3, cited 20080816

http://framework.zend.com/releases/ZendFramework-1.5.3/ZendFramework-1.5.3-manual-de.tar.gz

[Zmie08] Andrei Zmievski, PHP::$unicode->i18n(), cited 20080805

http://www.gravitonic.com/do_download.php?download_file=talks/phpmeetup-0708/php-unicode-i18n.pdf