Internationalisierung

Von der Zeichenkodierung über Webservices zu ... : Der Gegenstand der Internationalisierung verändert sich mit jeder (neuen) Technologie

Felix Sasaki

Internationalisierung ist ein Bereich, mit dem viele in Berührung kommen: Webseitenautoren, Programmierer von Datenbanken, Entwickler von Webservices etc. Der Artikel vermittelt einen Überblick zur Thematik, von grundlegenden Aspekten wie Zeichenkodierung bis zu neueren Entwicklungen wie internationalisierten Web Services.

Abgrenzung: Internationalisierung (i18n) versus Lokalisierung (l10n)

„Internationalisierung“ (abgekürzt „i18n“) bedeutet ein Produkt oder eine Technologie derart zu gestalten, dass sie für verschiedene Sprach- und Kulturräume, anpassbar sind. Eng mit i18n verbunden ist die „Lokalisierung“ („l10n“), d. h. der eigentliche Anpassungsprozess. Die Beziehung von i18n und l10n zeigen Abbildung 1 und 2 am Beispiel von Java „ResourceBundle“.

```
import java.util.*;
public class MeineRessource extends ListResourceBundle {
      public Object[][] getContents() {
              return inhalt;
      }
      static final Object[][] inhalt = {
              {"hello", "Hello"},
              {"bye", "Good bye"},
      };
 }
public class MeineRessource_de extends ListResourceBundle {
      public Object[][] getContents() {
              return inhalt;
      }
      static final Object[][] inhalt = {
              {"hello", "Hallo"},
              {"bye", "Auf Wiedersehen"},
      };
 }
```

Abb. 1

„ResourceBundle“ sammeln „Locale“ spezifischen Ressourcen, z. B. die Klasse „MeineRessource“. „Locale“ bezeichnet ein Land oder eine Region, z.B. „GERMAN“. Die Methode „getBundle“, vgl. Abb. 2, wählt eine Locale spezifische Ressource aus, z. B. „MeineRessource_de“. „getString“ gibt aus dieser Ressource einen String zu einem gegebenen Schlüssel zurück.

Der Internationalisierungsaspekt ist in diesem Beispiel die Bereitstellung der Technologie „RessourceBundle“. Die Lokalisierung nutzt diese Technologie z. B. zur Erstellung von Locale spezifischen Texten.

```
import java.util.*;
class I18nGruss {
       public static void main(String[] args){
       Locale meineLocale = Locale.GERMAN;
       ResourceBundle meineRessource =
       ResourceBundle.getBundle("MeineRessource",meineLocale);
       String gruss = meineRessource.getString("hello");
       System.out.println(gruss);
       }
}
```

Abb. 2

Zeichenkodierung

Die Grundlage jeder Internationalisierung ist die Zeichenkodierung. „RessourceBundle“ wären nutzlos ohne einen umfassenden Zeichenvorrat für verschiedene Schriftsysteme. Die größte Verbreitung hat heutzutage der Zeichenvorrat im „Unicode“ Standard [1]. In Unicode werden Zeichen verschiedener Länder zu so genannten „Skripts“ zusammengefasst. Unicode unterscheidet bei der Kodierung von Zeichen (vgl. [2]):

- Zeichenvorrat, z. B. Zeichen des lateinischen Alphabets „Basic Latin“. Historische oder regionale, visuelle Varianten (z. B. die Variante eines „A“ in Sütterlinschrift) werden als unterschiedliche „Glyphen“ aufgefasst; die Darstellung geschieht mittels Fonts (z. B. „Arial“). Glyphen- oder Fontvarianten stellen bei Unicode keine eigenständigen Zeichen dar.
- Zuweisung numerischer Identifikatoren („code point“, „character number“, „code position“), z. B. hexadezimal 5A für das Zeichen „Z“.
- Wahl eines Basisdatentyps („code units“), z. B. Byte oder Doppelbyte.
- Beschreibung der Byte Abfolge („byte order“).

Die drei Unicode Kodierungen („encoding forms“) UTF-8, UTF-16 und UTF-32 umfassen die gleichen Code Points. Sie unterscheiden sich hinsichtlich der Wahl des Basisdatentyps (Byte, Doppelbyte oder zwei Doppelbytes). Diese und weitere Namen für Zeichenkodierungen sind als „IANA charset identifier“ standardisiert [3].

Unicode wird in vielen Technologien wie Java oder XML angewendet. In neuerer Zeit ist ein Standard für „internationalized ressource identifier“ (IRI) [4] entwickelt worden, der internationalisierte Domänennamen wie http://点心和烤鸭.w3.mag.keio.ac.jp ermöglicht. Allerdings verarbeitet noch nicht jeder Browser IRI. Zudem eröffnen IRI Sicherheitsprobleme, die unter dem Stichwort „Pishing“ traurige Berühmtheit erlangt haben. Technologien zur Lösung dieser Probleme sind in der Entwicklung, vgl. [5].

In XML oder z. B. HTML Version 4.0 ist Unicode das „document character set“. Jede andere Zeichenkodierung (z.B. ASCII, iso-8859-1 etc.) kann verwendet werden, so lange sie eine Teilmenge des Unicode Zeichenvorrats umfasst. Acht geben muss man auf die verschiedenen Positionen, in denen die Zeichenkodierung deklariert werden kann:

- als Teil des in einer Server-Konfiguration festgelegten HTTP Content-Type Header, z.B. „text/html; charset=utf-8“.
- in der XML Deklaration, z.B. „<?xml version=“1.0“ encoding=“utf-8“>“.
- im <meta/> Element, z.B. „<meta http-equiv="Content-Type" content="text/html; charset=utf-8" />“.

Das Zusammenspiel dieser Deklarationen wird in [6] näher beschrieben.

Schritftformatierung: Direktionalität

Auch bei der Anordnung von Zeichen gibt es Internationalisierungsaspekte zu beachten. Skripts wie Arabisch oder Hebräisch werden von Rechts nach Links dargestellt („visual order“). Die Speicherung der Zeichen erfolgt normalerweise in der Abfolge, wie sie geschrieben werden („logical order“). Im Unicode Standard haben alle Zeichen eine Eigenschaft „Direktionalität“. Der Unicode bidirektionale Algorithmus [7] nutzt diese Information, um aus Quellcode in „logical order“ die Visualisierung in „visual order“ zu erzeugen. Bei der Mischung von Skripts mit unterschiedlicher Direktionalität braucht der Algorithmus jedoch eventuell Zusatzinformationen, vgl. Abbildung 3.

```
Quellcode: english1 HEBREW2 english3 HEBREW4 english5
Mögliche Visualisierung 1: english1 2WERBEH english3 4WERBEH english5
Mögliche Visualisierung 2: english1 4WERBEH english3 2WERBEH english5
```

Abb. 3

Visualisierung 1 des Quellcodes passt zu einem englischen Text mit zwei hebräischen Zitaten. Die Zeichen in den Zitaten werden von rechts nach links dargestellt. Visualisierung 2 passt zu einem englischen Text mit einem hebräischen Zitat, welches selbst Englisch enthält („english3“). Hier reicht die Direktionalitätsinformation der Zeichen nicht aus.

Visualisierung 2 beinhaltet eine neue direktionale Einbettung („directional embedding level“) für das hebräische Zitat. Es gibt verschiedene Arten diese Einbettung zu markieren. Unicode bietet die Code Points U+202B (genannt „RIGHT-TO-LEFT EMBEDDING“), und U+202C („POP DIRECTIONAL FORMATTING“ zur Aufhebung der Einbettung). In HTML [8] kann und sollte stattdessen das „dir“ Attribut verwendet werden.

```
Erzeugung der Visualisierung 2 mittels Unicode Code Points:
english1 U+202B HEBREW2 english3 HEBREW4 U+202C english5
Erzeugung der Visualisierung 2 durch das "dir" Attribut in HTML:
english1 <SPAN dir="RTL">HEBREW2 english3 HEBREW4</SPAN> english5
```

Abb. 4

Die Direktionalitätseigenschaft der Zeichen wird hierbei nicht ausgeschaltet; der Unicode bidirektionale Algorithmus bekommt nur zusätzliche Informationen für die direktionale Einbettung. Um hingegen die Direktionalitätsinformation zu überschreiben, können die Unicode Codepoints U+202D (LEFT-TO-RIGHT OVERRIDE) bzw. U+202E (RIGHT-TO-LEFT OVERRIDE) genutzt werden. Für HTML wird stattdessen das Element „bdo“ mit den Werten „LTR“ oder „RTL“ empfohlen. Erzwungene Direktionalität kann sinnvoll sein, wenn z. B. ein Text transliteriertes, d. h. im lateinischer Schrift dargestelltes Hebräisch enthält (Abb. 5).

```
Quellcode mit Unicode Code Points:
english1 U+202E HEBREW-TRANSLITERIERT U+202C english3
Quellcode in HTML:
english1 <bdo dir="RTL">HEBREW-TRANSLITERIERT</bdo> english3
Visualisierung: english1 TREIRETILSNART-WERBEH english3
```

Abb. 5

CSS [9] bietet die Eigenschaften „direction“ und „unicode-bidi“ für die gleichen Effekte wie „dir“ und „bdo“ in HTML. Die HTML Attribute haben vor CSS Angaben Vorrang.

Operationen auf Zeichen

Die grundlegendste Operation auf Zeichen ist das Zählen. Damit die beschriebenen Unterschiede zwischen Basisdatentypen und Byteabfolge keinen Einfluss haben, sind Code Points die zu wählende Zähleinheit. Technologien wie Java oder XQuery zählen Zeichen im Sinne von Code Points. Sie unterscheiden sich allerdings in einem wesentlichen Punkt, vgl. Abbildung 6.

```
Eingabe: "a"
Regulärer Audruck: a?
Ergebnis Java: 2 matches
Ergebnis XQuery: 1 match
```

Abb. 6

Java zählt Zeichengrenzen mit, z. B. die Grenze zwischen dem nullten und dem ersten Zeichen „a“. Deshalb trifft der reguläre Ausdruck „a?“ zweimal. XQuery hingegen zählt nur die Zeichen an sich. Der gleiche reguläre Ausdruck trifft nur einmal.

Eine weitere Operation ist das Ordnen oder Indizieren von Zeichenketten. Kollationen beschreiben die Ordnungsrelation zwischen den Zeichen. Abbildung 7 zeigt hierzu zwei Anwendungen der Funktion „compare“ in XQuery.

```
fn:compare('Strasse', 'Straße')
fn:compare('Strasse', 'Straße', 'http://example.com/meineOrdnung/deutsch')
```

Abb. 7

Im ersten Beispiel von „compare“ lautet der Rückgabewert der Funktion “-1“. Das bedeutet, dass der erste Eingabewert geringer ist als der zweite. Hierbei kommt die Default Kollation von XQuery zur Anwendung: Die Ordnungsrelation wird durch die numerische Ordnung der Code Points bestimmt. Im zweiten Beispiel von „compare“ wird eine zusätzliche, hier fiktive URI eine Kollation für das Deutsche angegeben. Der Rückgabewert lautet „0“, wenn diese Kollation keine Unterschied macht zwischen „ß“ und „ss“.

Beim Zeichenvergleich muss man Repräsentationsvarianten beachten. Ein „ç“ kann als ein Codepoint U+00E7 oder als zwei Codepoints U+0063 U+0327 („c“ und „¸“) repräsentiert werden. Die Erzeugung identischer Repräsentationen bezeichnet man als „Normalisierung“. Die in Unicode definierten Normalisierungsformen sind in [10] beschrieben. Die Normalisierungsform „NFC“, bei der „ç“ als ein Codepoint repräsentiert wird, ist weit verbreitet.

Internationalisierung ausgewählter Datentypen: Zeitangaben

Die folgenden Beispiele zur Internationalisierung verlassen die Ebene der Zeichen und wenden sich Datentypen für Zeitangaben zu. Diese weisen eine verwirrende Vielfalt auf, vgl. [11]:

- inkrementelle Zeitangaben sind zeitzonenunabhängige, numerischer Werte. In Java gibt es z. B. die Klasse „java.util.Date“, deren Objekte Zeitangaben als Millisekunden wiedergeben, gezählt seit 0 Uhr am 1 Januar 1970.
- feldbasierte Zeitangaben trennen die Zeitinformation in Felder wie „Jahr“ oder „Tag“. Ein Beispiel ist Ausgabe der Methode „toString“ in „java.util.Date“, vgl. Abb. 8. Die Ausgabe, die aus dem inkrementellen Wert von „java.util.Date“ erzeugt wird, trennt die inkrementelle Zeitangabe in Jahre, Tage, Stunden, Minuten und Sekunden. In der hier lokalisierten Ausgabe ist zusätzlich eine Zeitzone „JST“ („Japanese Standard Time“) angegeben. Es gibt feldbasierte Zeitangaben mit bzw. ohne Zeitzoneninformationen.

```
Date d = new Date();
System.out.println(d.toString());
Ausgabe: Sun Nov 20 15:50:50 JST 2005
```

Abb. 8

Der Begriff „Zeitzone“ ist definiert durch zwei Konzepte: UTC („Universal Coordinated Time“) und die Abweichung („Offset“) von UTC. Das Offset kann dabei für die gleiche Region je nach Datum verschieden sein, bedingt z. B. durch die Sommerzeit. Daten über Regionen, Zeitzonen und variable Offsets sind in der „Olson timezone database“ [12] gespeichert. Jede Zeitzone ist durch eine ID gekennzeichnet, z. B. „Pacific Time“. Offsetveränderungen für die gleiche Region können durch unterschiedliche IDs ausgedrückt werden, z. B. „PDT“ („Pacific Daylight Time“, UTC-7) für die Sommerzeit versus „PST“ („Pacific Standard Time“, UTC-8).

Noch verwirrender wird die Thematik, wenn man Zeitangaben in verschiedenen Programmiersprachen betrachtet. Java und viele Programmiersprachen nutzen inkrementelle Zeitangaben; nur in Ausgaben wie mit „toString“ gibt es feldbasierte Zeitangaben. Die SQL Datentypen „date“, „time“ und „timestamp“ sind hingegen feldbasiert und haben kein Offset zu UTC. Ein „java.util.Date“ Objekt ist jedoch auch eine inkrementelle Zeitangabe, wenn es aus einer SQL Datenbank etwa mittels JDBC erzeugt wird. XML Schema [13] schließlich definiert feldbasierte Datentypen wie „time“ mit einem optionalen „time zone indicator“. Es handelt sich dabei jedoch um einen numerischen Wert, der das „Offset“ beschreibt. Die angesprochenen Informationen der Olson Datenbank über regional und jahreszeitlich bedingte, variable Offsets werden nicht berücksichtigt.

Informationen zu Locale in Webservices

Von Zeichenkodierung und Schriftformatierung über Datentyp spezifischen Aspekten gehend wird im Folgenden die Internationalisierung von Web Services kurz angesprochen. „WS-I18N („Web Services Internationalization“) [14] hat sich zum Ziel genommen, ein Rahmenwerk für sprach-, kultur- oder regionsbezogene Informationen zu schaffen. Dies umfasst die Locale, die Zeitzone oder weitere internationalisierungsbezogene Informationen. Abb. 9 zeigt die Angabe von benutzerspezifischen Informationen zum metrischen System in SOAP.

```
<i18n:preferences>
 <ldml:measurementSystem type="metric"/>
</i18n:preferences>
<i18n:preferences>
 <ldml:alias source="de_DE"/>
</i18n:preferences>
```

Abb. 9

Zur Bestimmung der Locale bei Webservices wird teilweise der HTTP „Accept-language header“ verwendet. Hier kommt der Standard RFC 3066 zur Anwendung, der ISO 639 (Sprachcodes) und ISO 3166 (Ländercodes) nutzt. Vor kurzem ist von der IETF ein Standard [15] verabschiedet worden, der eine wesentlich feinere Unterteilung erlaubt. Damit können Skriptunterschiede oder Unterschiede in der gleichen Region explizit gemacht werden. Letztere sind z. B. bedeutsam für Kollationen neuer versus alter deutscher Rechtschreibung.

Internationalisierung in Markupsprachen

Internationalisierung in Markupsprachen ist unter zwei Aspekten interessant. Zum einen haben ausgewählte Zeichen in Markupsprachen eine besondere Rolle oder sollten nicht verwendet werden. Dieser Aspekt betrifft z. B. die erwähnten Unicode Code Points zur Markierung von Direktionalität in HTML und wird in [16] erläutert. Zum anderen ist es sinnvoll, Markup für Direktionalität, zeitbezogene Informationen oder Locale Information in einem Tag Set zusammenzufassen. Dies ist das Ziel der W3C Arbeitsgruppe „Internationalization Tag Set“ (ITS). Gegenwärtig hat die Arbeitsgruppe Anforderungen für ITS [17] entwickelt und eine erste Implementierung [18] vorgestellt. Hier treffen sich auch wieder Internationalisierung und Lokalisierung. So ist ein für die Lokalisierung wichtiger Bestandteil von ITS die Information zur Übersetzbarkeit von Element- oder Attributinhalten, vgl. Abb. 10. Attribute wie „its:translate“ zeigen an, ob ein String übersetzt werden soll.

```
<book>
 <head>...</head>
 <body its:translate="yes"> ...
  <p>And he said: you need a new <quote its:translate="no">motherboard</quote>
  </p> ...
 </body>
</book>
```

Abb. 10

Ausblick

Zum Schluss soll noch ein Ausblick auf zukünftige Perspektiven der Internationalisierung geben werden. Es handelt sich um die Markup zur Unterstützung von automatischen Übersetzungen. Dieses wird im Rahmen der ITS Arbeitsgruppe entwickelt, vgl. [19]. Das Beispiel zeigt, wie vielfältig Internationalisierung ist: Ihr Gegenstand verändert sich mit jeder (neuen) Technologie.