von kerstin dittert

mehr zum thema:
java.sun.com/j2se/1.3/docs/guide/intl/
www.unicode.org

TURMBAU ZU BABEL: ARCHITEKTUR INTERNATIONALER ANWENDUNGEN

Im Zuge der Globalisierung streben expandierende Unternehmen die Vereinheitlichung ihrer lokalen Softwaresysteme an. Im Internet werden Dienstleistungen und Produkte über Ländergrenzen hinweg angeboten. Die Themen „Mehrsprachigkeit" und „Internationalisierung" stehen deshalb unter den Anforderungen an neue Anwendungen häufig ganz oben. Internationalisierung bedeutet jedoch mehr als die Übersetzung von Maskenbeschriftungen und Fehlermeldungen. Der Artikel beschreibt die architekturrelevanten Aspekte internationaler Anwendungen und liefert dem Softwarearchitekten Hinweise zur Umsetzung effektiver Softwarearchitekturen[1]).

Globale Märkte erfordern neue Prozesse

Lange bevor Softwareprodukte international eingesetzt werden sollten, entwickelte die Industrie schon Produkte für globale Märkte. Stellen Sie sich folgendes *Szenario* vor:

Ein Automobilhersteller des europäischen Fantasielandes Expando entwickelt einen neuen PKW, der im Heimatland zum Verkaufsschlager wird. Das Management beschließt, den Wagen auf den europäischen und amerikanischen Markt zu bringen. Ein Team erhält den Auftrag festzustellen, welche Ergänzungen notwendig sind, um das Fahrzeug international zu vertreiben.

Die Gruppe erstellt in aufwändiger Detailarbeit eine Liste aller Beschriftungen im Wageninneren. Sie fordert eine Übersetzung dieser Texte und zusätzlich der Betriebsanleitung in alle europäischen Sprachen an. Mit beträchtlichem Aufwand werden die Texte übersetzt. Der Wagen lässt sich nun für jedes Zielland anders konfigurieren:

- Alle Beschriftungen sind in der Landessprache verfasst.
- Das eingebaute Navigationssystem ist für jedes Land mit den entsprechenden Karten und der passenden Sprachausgabe ausgestattet.
- Für jedes Land gibt es eine andere Betriebsanleitung.

Das Marketing beginnt, der Wagen wird europaweit und in Amerika vertrieben. Zur Überraschung aller entwickelt sich der ausländische Umsatz jedoch mehr als schleppend. In England wird nicht ein einziges Auto verkauft. In Deutschland veröffentlicht der ADAC Testergebnisse, nach denen der Wagen bei 180 km/h in der Kurve ausbricht. In Amerika wird der Wagen gar nicht erst für den Verkauf zugelassen. Das Management beruft eine Krisensitzung ein. Es war doch so viel Aufwand in die Übersetzungsarbeit gesteckt worden! Wieso nehmen die Kunden den Wagen nicht an? Die Ursachen für die mangelnde Akzeptanz werden gemeinsam vom Marketing und Mitarbeitern der Konstruktionsabteilungen untersucht. Die Ergebnisse fallen ernüchternd aus:

- In England hatte man vergessen, das Lenkrad auf der rechten Seite einzubauen.
- In der amerikanischen Version hatte man es versäumt, den Tacho auf die Maßeinheit Meilen/Stunde umzurechnen.
- Auf Grund eines Tempolimits vom 130 km/h in Expando wurde die Konstruktion des Wagens auf die dortige Höchstgeschwindigkeit ausgelegt. Der Fehler wurde auch bei Prüfungen nicht entdeckt, da spezielle Tests für einzelne Länder-Konfigurationen nicht durchgeführt wurden.

Bei der Konzentration auf die sprachlichen Aspekte hatte man andere Aspekte, wie Maßeinheiten, technische Randbedingungen etc., völlig vergessen. Der gesamte Fertigungsprozess bis hin zur Qualitätssicherung war nicht auf die neuen Anforderungen abgestimmt worden.

So überspitzt die obige Geschichte auch sein mag: Die Erstellung international einsetzbarer Software stellt Sie als Softwarearchitekten vor ähnliche Probleme. Der Aspekt der Internationalisierung beeinflusst den gesamten Softwareentwicklungsprozess, von der Anforderungsanalyse bis hin zur Testplanung. Die erweiterten Anforderungen müssen Sie deshalb frühzeitig in Ihre Überlegungen mit einbeziehen. Neben verschiedenen Sprachen müssen Sie manchmal auch nationale Unterschiede – wie z. B. Währungen, gängige Formatierungen, Zeitzonen – berücksichtigen.

Je nach Zielgruppe und angestrebtem Markt kann es sogar erforderlich sein, kulturelle und politische Aspekte mit zu betrachten. Dies ist besonders wichtig für Web-Anwendungen, die sich an ein weltweites und unspezifisches Publikum wenden (z. B. Suchmaschinen, Free-Mail-Provider).

Der Internationalisierungsprozess umfasst die Lokalisierung sprachabhängiger Ressourcen, aber er geht über eine reine sprachabhängige Ressourcen-Anbindung hinaus (**vgl. Kasten 1**).

Dimensionen der Internationalisierung

Internationalisierung ist teuer und nicht jede Anwendung ist für den chinesischen

[1]) *Dieser Artikel wurde von der Autorin auch in [Sta02] veröffentlicht. Internationalisierung ist einer der im Buch beschriebenen Bausteine, die das Grundgerüst einer robusten Softwarearchitektur bilden.*

die autorin

Kerstin Dittert (E-Mail: kerstin.dittert@oocon.de) ist Geschäftsführerin der OOcon Informatik GmbH. Sie unterstützt Unternehmen in allen Gebieten des objektorientierten Softwareentwicklungsprozesses sowie in Technologiefragen. Hierbei ist sie unter anderem als Softwarearchitektin, Analytikerin, Designerin und Projektleiterin tätig.

- **Lokalisierung (l10n)[2]):** Anpassung einer bestehenden Software an eine andere Sprache oder ein anderes Land. Dieser Prozess umfasst die Erstellung und Anbindung angepasster Ressourcen, wie z. B. Texte, Bilder oder Sprachdateien. Die Art der Ressourcen-Anbindung ist dabei beliebig. Es kann sich sowohl um extern referenzierbare Dateien eines Softwarepakets handeln, als auch um unterschiedliche Anwendungsdistributionen für jede Sprache oder jedes Land.
- **Internationalisierung (i18n)[2]):** Ein Softwarepaket unterstützt mehrere Sprachen und unterschiedliche nationale Merkmale. Eine internationalisierte Anwendung beinhaltet also immer lokalisierte Ressourcen.

Kasten 1: Internationalisierung und Lokalisierung

Markt mit einer Fülle von über 40.000 Schriftzeichen geplant. Allgemeine generische Ansätze sind aufwändig in der Entwicklung, ohne dass damit zwangsläufig ein wirtschaftlicher Mehrwert verbunden ist. Zu Beginn des Architekturentwurfs sollten Sie deshalb die Anwender-Zielgruppe möglichst genau eingrenzen. Dabei sollten Sie folgende Fragen stellen:

- Welche Sprachen sollen unterstützt werden?
- Verwenden alle Sprachen ausschließlich lateinische Schriftzeichen?
- Wird Text immer horizontal von links nach rechts geschrieben?
- In welchen Ländern soll die Anwendung eingesetzt werden?

Sie und Ihr Team müssen herausfinden, wie hoch der Anteil sprachabhängiger oder landestypischer Anwendungslogik ist. Folgende Fragen können Ihnen dabei helfen:

- Welche Formatierungen werden benötigt?
- Müssen unterschiedliche Währungen verarbeitet werden?
- Enthalten die Anwendungsdaten Bezeichnungen, die übersetzt werden müssen?
- Enthält die Anwendung zeitabhängige Dienste?
- Welchen Umfang soll das Hilfesystem haben?
- Welche Anwendungsteile sollen über Tastaturkürzel (z. B. CTRL-C oder ALT-D) gesteuert werden?
- Gibt es kulturelle Unterschiede, die für die Anwendung relevant sind?

Haben Sie diese Fragen beantwortet, so können Sie entscheiden, welche Aspekte aus **Abbildung 1** für Ihre Architektur relevant sind. Vergessen Sie jedoch im Sog allgemeiner und generischer Ansätze eines niemals die Konzentration auf das Wesentliche. Treffen Sie eine Auswahl und grenzen Sie die Internationalisierungsaspekte ein.

Mehrsprachigkeit

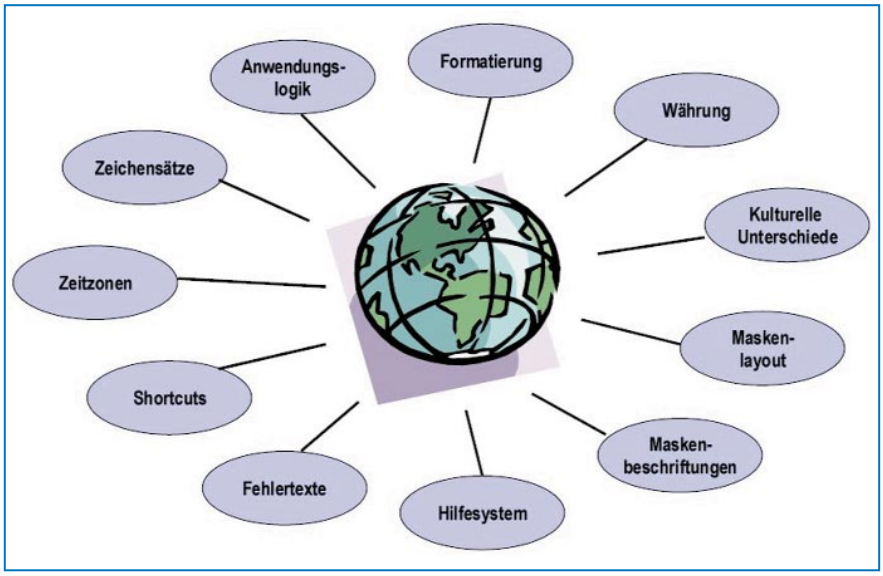
Fast alle internationalen Anwendungen unterstützen mehrere Sprachen, manche sogar auch Sprachvarianten, wie z. B. britisches und amerikanisches Englisch. Die Notwendigkeit solch feiner Unterscheidung müssen Sie sorgfältig abwägen. Die Ziele einer hohen Anwenderakzeptanz und des möglichst geringen Entwicklungs- und Wartungsaufwandes stehen sich komplementär gegenüber. Für die Rechtschreibprüfung einer Textverarbeitung ist die Berücksichtigung von Sprachvarianten unerlässlich. Die Web-Seiten einer Internet-Suchmaschine können hingegen ausschließlich in einer englischen Sprachvariante angeboten werden.

Sämtliche Oberflächenelemente müssen sprachabhängig beschriftet werden und werden deshalb als externe Ressourcen benötigt. Rückmeldungen an den Anwender – wie Fehlermeldungen, Statuszeilentexte etc. – müssen ebenfalls in allen Sprachen vorliegen und beim Satzbau den Regeln der jeweiligen Grammatik folgen.

Textkonstanten sind in verschiedenen Sprachen unterschiedlich lang und beanspruchen dementsprechend mehr oder weniger Platz. Grafische Benutzeroberflächen mehrsprachiger Systeme sollten deshalb niemals ein statisches Layout vorgeben. Verwenden Sie dennoch ein statisches Layout, so sind die Positionen und Größen der einzelnen Oberflächenelemente fest vorgegeben. Bei unbekannten Beschriftungslängen muss also vorsorglich einiges an zusätzlichem Platz reserviert werden, damit die Texte nicht abgeschnitten werden.

Statt dessen sollte sich die Größe beschrifteter Elemente (Buttons, Label, Menüeinträge usw.) dynamisch an den enthaltenen Text anpassen. In Java werden hierfür spezielle Layout-Management-Klassen eingesetzt, welche die Positionen und Größen einzelner *Controls* über Beziehungen vorgeben (**vgl. Tabelle 1**). Zur Laufzeit wird das Layout bezüglich der tatsächlichen Beschriftungslängen optimiert.

Die Oberflächenorientierung einer GUI folgt immer der Textausrichtung der



[2]) *Die häufig verwendeten Abkürzungen* **l10n** *und* **i18n** *basieren auf den englischen Bezeichnungen „Localization" und „Internationalization". Sie bestehen aus den Anfangs- und Endbuchstaben der Begriffe sowie der Anzahl dazwischenliegender Buchstaben.*

Sprache. Menüs werden z. B. in deutschsprachigen Ländern von links nach rechts angeordnet, in arabischsprachigen Ländern jedoch von rechts nach links. Ebenso werden die Elemente einer Maske entweder von links nach rechts oder umgekehrt angeordnet. Falls also Sprachen mit unterschiedlicher Textausrichtung unterstützt werden sollen (z. B. Englisch und Arabisch), so wird eine variable GUI-Komponentenorientierung benötigt. Hierfür muss das GUI-Layout-Management neben der variablen Positionierung und Größenbestimmung um dynamische Orientierung ergänzt werden.

Die folgende Heuristik kann Ihnen helfen, Ihre Architektur für mehrere Sprachen auszulegen:

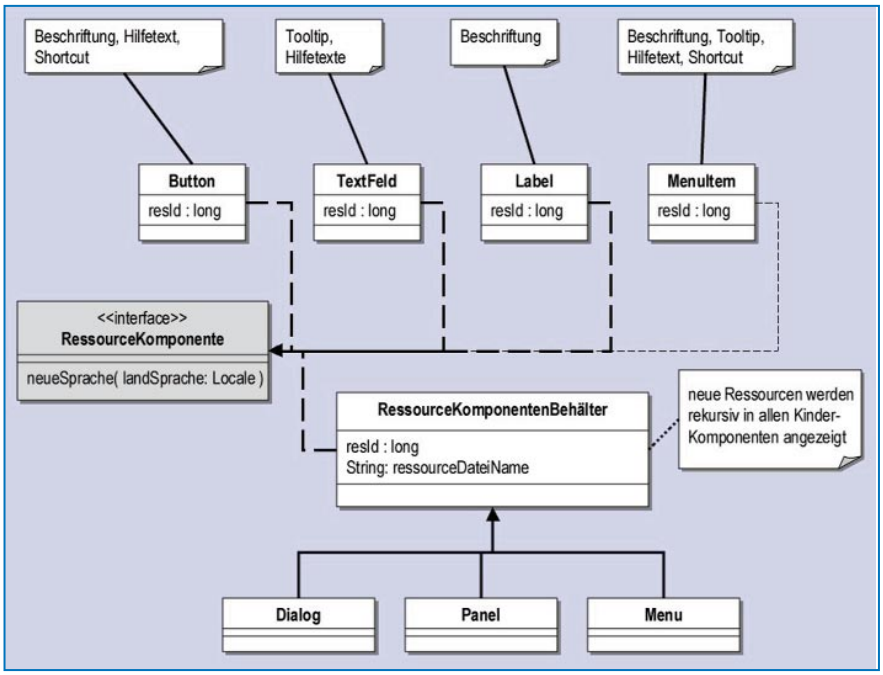
- Legen Sie zunächst die für das System erforderlichen Sprachen fest (z. B. Deutsch, Englisch, Arabisch).
- Überprüfen Sie sorgfältig, ob Sie unterschiedliche Sprachvarianten benötigen (z. B. Deutsch-Schweiz, Deutsch-Deutschland). Der Übersetzungsaufwand steht häufig in keinem vernünftigen Verhältnis zum Mehrwert innerhalb der Anwendung.
- Setzen Sie bei der Masken-Gestaltung Layout-Manager-Klassen ein. Diese passen die Größe und Position der Maskenelemente an die aktuellen Beschriftungslängen an.
- Sollte die von Ihnen eingesetzte Programmiersprache kein dynamisches Layout-Management unterstützen, so berücksichtigen Sie zusätzlichen Platzbedarf für andere Sprachen. Geben Sie in Entwurfsrichtlinien maximale Beschriftungslängen vor (z. B. 30 Zeichen).
- Ordnen Sie Menüs und andere Oberflächenelemente gemäß der Textausrichtung der verwendeten Sprache an.
- Legen Sie sämtliche sprachabhängigen Elemente in Ressourcen-Dateien ab (Oberflächenbeschriftungen, Fehlertexte, *Shortcuts*, *Tooltips*, Audio-Dateien usw.).

Externe Ressourcen

Sämtliche statischen Texte und alle übrigen sprachabhängigen Inhalte, wie Bilder oder Audio-Dateien, werden in sprachspezifischen Ressourcen-Dateien oder -Klassen von der Anwendungsimplementierung entkoppelt. **Abbildung 2** zeigt ein Design zur Anbindung von Ressourcen-Dateien innerhalb des GUI-Frameworks einer Anwendung.

Paket	Klassen	Verwendung
java.awt.package	ComponentOrientation	Variable Komponentenorientierung
	LayoutManager	dynamisches GUI-Layout
	Font	Laden spezieller Fonts
java.io	InputStreamReader, OutputStreamReader	Import/Export mit speziellen Kodierungen
java.lang	Character	Verarbeitung von Unicode-Zeichen
java.text	AttributedCharacterIterator, CharacterIterator, CollationElementIterator, StringCharacterIterator	Parsen von Strings
	BreakIterator	Bestimmung von Textgrenzen (Satz, Wort)
	Collator, CollationKey RuleBasesCollator	Sortierung
	DateFormat, DateFormatSymbols, SimpleDateFormat	Formatierung von Datum und Zeit
	DecimalFormat, DecimalFormatSymbols Format NumberFormat	Formatierung von Währung und numerischen Werten
	ChoiceFormat, MessageFormat	Formatierung von Fehlermeldungen und sonstigen Nachrichten
java.util	Calendar, GregorianCalendar, SimpleTimeZone, TimeZone	Zeitstempel, Zeitberechnungen, Kalender
	ListResourceBundle, PropertyResourceBundle, ResourceBundle	Anbindung lokalisierter Ressourcen
	Locale	Festlegung der Kombination Sprache/Land/Sprachvariante

Tabelle 1: Implementierung internationaler Anwendungen mit Hilfe der Java-Standard-Bibliotheken



Zur Laufzeit bestimmt das Programm die aktuelle Kombination von Land und Sprache (Locale-Objekt, **vgl. Tabelle 1**). Dies kann mit Betriebssystemmitteln, über ein Benutzerprofil oder auf Anforderung des Benutzers erfolgen. Diese Länder-Sprachkombination wird allen Anwendungsteilen zur Verfügung gestellt. Anschließend können die jeweils benötigten Ressourcen geladen werden. Im GUI-Framework laden alle Container ihre zugehörigen Ressourcen-Dateien und benachrichtigen rekursiv Sub-Container und Sub-*Controls* über die neuen Sprach-Ressourcen. Jeder GUI-*Control* kann dann seine zugehörigen Ressourcen-Elemente laden.

Auch Tastaturkürzel (wie ALT-S für Speichern) gehören zu den Daten, die erst nach Auswahl der Sprache zugeordnet werden können. Sie werden deshalb ebenfalls als Ressourcen verwaltet und den Oberflächenelementen dynamisch zugeordnet. Hierbei ist besonders darauf zu achten, dass die Kürzel als Buchstabe in der zugeordneten Beschriftung enthalten und pro Maske oder Menü eindeutig sind.

Das gesamte Hilfesystem (hierzu gehören auch *Tooltips* und Benutzerhandbücher) muss ebenfalls in allen Sprachen vorliegen.

Fehlermeldungen und sonstige Nachrichten an den Benutzer werden auch als Ressourcen gehalten und in einem zentralen Pool verwaltet. Das hat den angenehmen Nebeneffekt, dass eine Standardisierung der Nachrichtentexte einfach zu erreichen ist. Nachrichten können Platzhalter enthalten, die zur Laufzeit kontextabhängige Inhalte aufnehmen können. Ein Beispiel hierfür sind zwei Nachrichtenformate

- „Benutzer {0} ist im System nicht bekannt."
- „Unknown user {0}."

die zur Laufzeit mit dem Benutzernamen „Meier" gefüllt werden.

Achten Sie darauf, dass die Reihenfolge der Satzteile sowie die Interpunktion in jeder Sprache unterschiedlich sein können. Formatieren Sie Meldungen mit speziellen Klassen, welche die Platzhalter füllen und die Sprachgrammatik berücksichtigen. Einige Klassenbibliotheken bieten hierfür spezielle Formatierungsklassen an (**vgl. Tabelle 1**).

In der Standard-Vorgehensweise zur Ressourcen-Anbindung werden einzelne Ressourcen-Elemente mit Textkonstanten assoziiert. Ein einzelnes Element kann also nur über mehrere Zeichenketten-Vergleiche identifiziert werden. Bei größeren Ressourcen-Klassen mit Hunderten von Einträgen wird diese Vorgehensweise schnell inperformant. Abhilfe versprechen zwei Vorgehensweisen:

- Alle Masken-Ressourcen werden bei der Anwendungs-Initialisierung bereits im Hintergrund geladen. Die Bildschirmmasken werden instanziiert, bevor sie angezeigt werden. Verzögerungen beim Bildschirmaufbau können so vermieden werden.
- Es werden array-basierte Ressourcen-Klassen verwendet, die über einen numerischen Index direkt auf die benötigten Ressourcen-Elemente zugreifen.

Zeichensätze und Kodierung

Sobald die anwendungsrelevanten Sprachen und Sprachvarianten festgelegt worden sind, können geeignete Kodierungsstandards für alphanumerische Zeichen identifiziert werden. Der ASCII-Standard wird keinesfalls ausreichen, da dieser für den Einsatz in den USA optimiert wurde. Ein britischer Anwender würde bereits das £-Zeichen für die Währungsdarstellung vermissen. Im Idealfall kommt man im europäischen Raum mit einer der ISO-Kodierungen zur Unterstützung mehrer Sprachen aus. Die ISO-Kodierung 8859-1 ist beispielsweise für die Sprachen Albanisch, Baskisch, Katalanisch, Dänisch, Niederländisch, Englisch, Finnisch, Französisch, Deutsch, Isländisch, Norwegisch, Portugiesisch, Rätoromanisch, Schottisch, Spanisch und Schwedisch geeignet. Eine Übersicht weiterer gängiger Zeichensätze finden Sie in [Cza01].

Die Angabe geeigneter Kodierungen gewährleistet jedoch nur die richtige programminterne Darstellung der Zeichen. Sind auf dem Rechner des Anwenders keine dazu passenden Zeichensätze installiert, so kann es passieren, dass Buchstaben nicht darstellbar sind und als Kästchen oder sonstige Platzhalter am Bildschirm angezeigt werden. Ein solches Verhalten ist zum Beispiel zu erwarten, wenn auf einem Computer in Deutschland japanische oder arabische Schriftzeichen dargestellt werden sollten. In diesen Fällen muss gewährleistet werden, dass passende Zeichensätze mit zum Distributionsumfang der Software gehören. Hierfür gibt es spezielle Fonts, wie z. B. den „Bitstream Cyberbit", der mehr als 30.000 Symbole umfasst und unter anderem Arabisch, Hebräisch und Thailändisch darstellen kann (siehe [UniFnt]).

Stellen Sie also sicher, dass

- jede Sprache in einem passenden Zeichensatz kodiert wird und
- dass in der Laufzeitumgebung die passenden Fonts installiert sind.

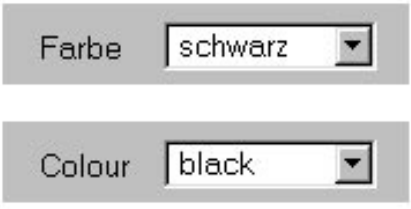
Mehrsprachige Anwendungsdaten

Die Notwendigkeit mehrsprachiger Anwendungsdaten wird häufig übersehen. Maskenübersetzungen und angepasste Fehlermeldungen nützen dem Anwender wenig, wenn er beispielsweise in Auswahllisten (für Farben, Warengruppen etc.) nur fremdsprachige Wörter sieht. **Abbildung 3** zeigt ein Beispiel für Farbauswahlboxen, deren Inhalte sich sprachabhängig ändern.

Die meisten sprechenden Bezeichnungen innerhalb der Anwendungsdaten müssen deshalb ebenfalls mehrsprachig vorliegen. Potenzielle Kandidaten dafür sind Attribute,

- die der Bezeichnung dienen und als Freitext eingegeben werden können (z.B. Artikelnamen),
- die einen endlichen Wertebereich haben (z. B. Artikeleigenschaften wie Farbe, Größe, Material). Derartige Attribute werden meist in Auswahllisten dargestellt (auch Comboboxen genannt).

Die Unterstützung mehrsprachiger Anwendungsdaten ist ein komplexes Thema. Neben dem Entwurf eines geeigneten Objekt- und Datenmodells müssen Mechanismen zur Übersetzung bereitgestellt werden. Die Erhaltung konsistenter Datenbestände erfordert häufig die Auszeichnung einer führenden Sprache. Besonders schwierig wird es, wenn Altdaten übernommen oder Fremdsysteme integriert werden müssen, da diese oft nicht mit den neuen Konzepten harmonieren.



Das Objekt- und Datenmodell muss um Sprachvariationen ergänzt werden. Im Objektmodell führt dies zu keiner weitreichenden Modellveränderung, da zur Laufzeit jeweils nur eine konkrete Sprachinstanz referenziert wird. Das Datenmodell muss jedoch für alle sprachabhängigen Anwendungsdaten um Sprach- und Länderschlüssel ergänzt werden (**vgl. Abb. 4.**).
Konzeptionell sind „relativ statische" Daten von „dynamischeren" Attributen zu unterscheiden. Eher statische Daten sind beispielsweise Farbkataloge und Postleitzahlenverzeichnisse. Diese werden periodisch und an zentraler Stelle geändert und können über geeignete Importschnittstellen aktualisiert werden. Sofern die Importquellen nur einsprachige Daten zur Verfügung stellen, ist anschließend eine manuelle Übersetzung erforderlich. Diese kann insbesondere auch bei Altdatenübernahmen erforderlich sein.

Im Gegensatz dazu sind Bezeichner – beispielsweise Artikelnamen und Warengruppennamen – dynamische Daten, die vom Anwender gepflegt werden. Da diese Datenpflege dezentral ist, muss festgelegt werden, wie die Übersetzungen der Daten erfolgen. Hierfür sind z. B. folgende Ansätze möglich:

- Es wird eine führende Systemsprache ausgezeichnet (z. B. Englisch). Bei der Bearbeitung sprachabhängiger Bezeichner muss die Übersetzung in der Systemsprache mit aktualisiert werden. Bei der Anzeige wird stets versucht, auf die Übersetzung der aktuell gewählten Sprache zuzugreifen. Liegt eine solche Übersetzung nicht vor, so wird die Bezeichnung statt dessen in der Systemsprache angezeigt. Übersetzungsdialoge stehen zusätzlich zur Verfügung. In regelmäßigen Abständen werden nicht übersetzte Bezeichnungen einem Übersetzungsteam automatisch vorgelegt. Der Vorteil dieser Vorgehensweise ist die einfache und schnelle Bearbeitung der Daten durch den Anwender. Qualität und Quantität der Übersetzungen können jedoch stark schwanken und sind durch das System nicht zu beeinflussen.
- Bezeichnungen müssen bei der Bearbeitung stets in alle Systemsprachen übersetzt werden. Hierfür kann ein Workflow definiert werden, in den ein Übersetzungsteam einbezogen wird. Erst nach der Übersetzung werden die Daten im System produktiv. Dem Vorteil einer hohen Anwenderunterstützung durch vollständig mehrsprachige Daten stehen hohe Kosten und Zeitverluste durch den Workflow-Prozess gegenüber.

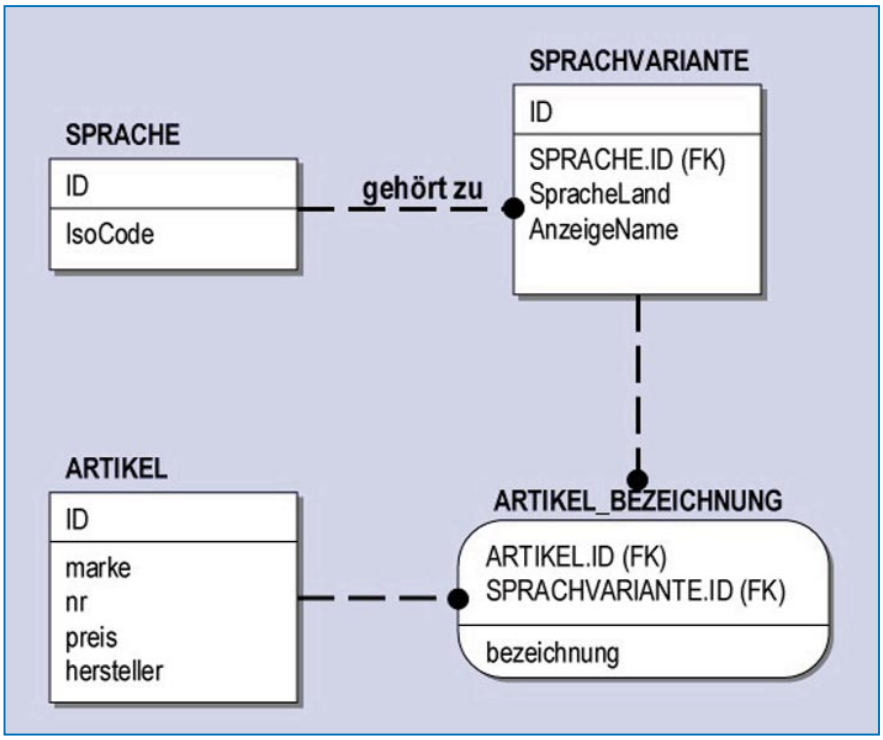


Abb. 4: ER-Diagramm für mehrsprachige Anwendungsdaten

Mehrsprachige Bezeichnungen erzeugen durch die permanente Übersetzungsproblematik einen hohen Arbeitsaufwand und damit verbundene Kosten. Das gilt sowohl für den Entwurf und die Implementierung des Systems, als auch für das Produktionssystem, das ständig manuelle Eingriffe erfordert.

Prüfen Sie deshalb sorgfältig, ob die Einführung mehrsprachiger Bezeichnungen einen großen Mehrwert des Systems darstellt. In den meisten Fällen ist es ausreichend, ausschließlich die eher statischen Anwendungsdaten, wie z. B. Kataloge, mehrsprachig vorliegen zu haben.

Bei der Umsetzung mehrsprachiger Anwendungsdaten

- sollten Sie eine Referenzsprache auszeichnen,
- den Umfang mehrsprachiger Daten auf das absolut erforderliche Minimum reduzieren,
- Daten- und Objektmodelle um Sprach- und Länderausprägungen erweitern,
- den Arbeitsablauf zur Bearbeitung der im produktiven Betrieb anfallenden Übersetzungsarbeiten definieren.

Formatierung

Datums- und Zeitangaben, Währungsfelder und numerische Angaben werden in unterschiedlichen Ländern auf verschiedene Art formatiert. Die Darstellungsreihenfolge, Trennzeichen und die Position einzelner Bezeichner variieren. Ignoriert man die länderspezifischen Konventionen, so kann das in einigen Fällen zur kompletten Fehlinterpretation der Anwendungsdaten führen. Die deutsche Datums- und Zeitangabe *„12.02.2001 14:22"* wird in anderen Ländern folgendermaßen dargestellt:

- Frankreich: *„12/02/2001 14:22"*
- USA: *„02/12/2001 2:22 PM"*
- Israel: *„14:22 12/02/2001"*

Die visuelle Formatierung eines Datumsfeldes unterscheidet sich also z. B. in Frankreich und den USA nicht, der Inhalt wird jedoch länderabhängig unterschiedlich interpretiert!

Die Formatierung von Datum, Zeitangaben, Geldbeträgen mit Währungssymbol sowie numerischen Feldern setzen Sie mit Hilfe spezieller Formatierungsklassen um. Hierfür werden Standardbibliotheken eingesetzt, welche die länderspezifischen Regeln enthalten und die bei Bedarf um zusätzliche Regeln ergänzt werden können (**vgl. Tabelle 1**).

Annahme	Gegenbeispiel
Das Alphabet besteht aus den Zeichen 'A' bis 'Z'.	In Dänemark folgen die Umlaute nach dem 'Z'.
Wörter werden durch Leerzeichen voneinander getrennt.	Im Thailändischen gibt es keine Worttrennung.
Die Interpunktion ist in allen Sprachen gleich.	Im Spanischen wird eine Frage von '¿' eingeleitet und mit '?' beendet.
Ein Buchstabe kann in einem Byte gespeichert werden.	8 Bit bieten Speicherplatz für maximal 256 Zeichen. Die chinesische Sprache kennt über 40.000 Zeichen!
Sprachen, die das gleiche Alphabet verwenden, haben auch die gleiche Sortierreihenfolge.	'A' und 'Ä' repräsentieren im Deutschen den gleichen Buchstaben mit unterschiedlicher Aussprache. Sie sind bezüglich der Sortierung äquivalent. Im Schwedischen stehen diese Textzeichen für unterschiedliche Buchstaben mit anderer Sortierung.
Es wird der gregorianische Kalender verwendet.	In einigen arabischsprachigen Ländern gilt zusätzlich der islamische Kalender.

Tabelle 2: Ungültige Annahmen bezüglich der Anwendungslogik

Anwendungslogik und Steuerung

Programmierer gehen oft implizit von bestimmten Annahmen aus, die in internationalen Anwendungen nicht immer gegeben sind. **Tabelle 2** nennt die häufigsten Unterstellungen und gibt Gegenbeispiele dazu an. Prüfen Sie, ob alle Annahmen für die Anwendung zutreffend sind. Ist dies nicht der Fall, so muss die Anwendungslogik darauf abgestimmt werden.

Sobald ein Programm länderübergreifend eingesetzt wird, sind häufig auch verschiedene Währungen mit im Spiel. Neben dem Formatierungsaspekt, der bereits angesprochen wurde, ist die Fähigkeit, mehrere Währungen zu verarbeiten, eine Anforderung, die auch die Anwendungslogik betrifft. Gegebenenfalls müssen Umrechnungen vorgenommen, Wechselkurse importiert und eine Systemwährung ausgezeichnet werden. Sie sollten frühzeitig feststellen, ob die Kurse zeitnah über entsprechende Service-Provider abgefragt werden müssen (z. B. bei Online-Banking-Anwendungen), oder ob die Umrechnungstabellen periodisch ins System importiert werden können.

Arbeitet eine Anwendung mit Zeitstempeln, so müssen Sie feststellen, ob Ihre Anwendung über mehrere Zeitzonen verteilt ist. Berücksichtigen Sie dabei, dass die Zeitzonen nicht immer mit Landesgrenzen zusammenfallen. Länder mit großer Flächenausdehnung, wie z. B. die USA, umfassen mehrere Zeitzonen. Bei zeitzonenübergreifenden Anwendungen müssen Sie einen zentralen Zeitstempelservice vorsehen.

- Stellen Sie sicher, dass die Anwendungslogik nicht durch Stringvergleiche gesteuert wird. Arbeiten Sie statt dessen mit numerischen Konstanten, Klassenbezeichnern usw. Überprüfen Sie die Einhaltung dieser Vorgaben in Code- und Design-Reviews.
- Setzen Sie spezielle Klassen für die Sortierung von Zeichenketten ein. Bei Bedarf können diese um sprachspezifische Sortierungsregeln ergänzt werden.
- Legen Sie im mehrwährungsfähigen System eine Referenzwährung fest und überprüfen Sie, wie zeitnah die Kursumrechnungen erfolgen müssen.
- Sehen Sie in zeitzonenübergreifenden Anwendungen einen Zeitstempelservice vor.

Kulturelle Besonderheiten

Soll eine Anwendung kulturübergreifend eingesetzt werden (z. B. im Internet), so ist besondere Vorsicht bei der Verwendung von Zeichen, Symbolen und Metaphern geboten. Im harmlosen Fall werden sie schlichtweg nicht verstanden, im ungünstigen Fall können sie aber auch eine völlig gegensätzliche Bedeutung haben und deshalb zu Akzeptanzverlusten führen. Beispielhaft hierfür ist die symbolische Verwendung von Zahlen: Vielen Westeuropäern gilt die 13 als Unglückszahl, in Hongkong ist es hingegen die 7. Mit der Zahl 666 verbinden die meisten Deutschen nichts, in den USA ist sie dagegen das Zeichen des Teufels.

Die Missverständnisse können noch weitaus größer werden, wenn Abbildungen von Gesten, Farben, Verkehrsschildern etc. zur Kommunikation mit dem Anwender eingesetzt werden. Die Bedeutung hängt oftmals stark vom Kulturkreis ab.

- Stimmen Sie die Bild- und Zeichensprache Ihrer Anwendung auf den Kulturkreis der Anwender ab.
- Ist die Zielgruppe stark heterogen, so verzichten Sie völlig auf den Einsatz von Symbolen und Metaphern.

Implementierung

Zur Implementierung der vorgestellten Konzepte stellen die meisten objektorientierten Sprachen geeignete Klassenbibliotheken zur Verfügung. **Tabelle 1** zeigt exemplarisch die wichtigsten Klassen der Java-Standard-Bibliotheken zur Umsetzung internationaler Anwendungen. ■

Literatur & Links

[Bab] BabelSite, Gemeinschaftsprojekt zur Internationalisierung des Internets. Interessantes über die Weltsprachen und viele Ressourcen zum Thema Internationalisierung (siehe: http://babel.alis.com)

[Cza01] D. Czarnecki, A. Deitsch, Java Internationalization, O'Reilly 2001

[Sta02] G. Starke, Effektive Softwarearchitekturen, Hanser 2002

[Unicode] Unicode Consortium, Unicode und Internationalisierung (siehe: http:// www.unicode.org)

[UniFnt] Unicode Fonts, Links zu verschiedenen Fonts, die ein großes Spektrum der Unicode-Zeichen darstellen können, u.a. der sehr umfassende Freeware-Font „Bitstream Cyberbit" (siehe: http://www.hclrss.demon.co.uk/unicode/fonts.html#general)