

# Patterns for Internationalization and Cross-Cultural Usability

PHILIPP BACHMANN, iRIX Software Engineering AG

---

Enabling a software application to be used in different cultures can vastly increase its market. The patterns in this catalog lay the foundation to later localize the application. This catalog consists of the following patterns: MULTI-CODED INSTRUCTION, MESSAGE CATALOG, INPUT EXAMPLE, CULTURALLY AND ENVIRONMENTALLY AGNOSTIC PERSISTENCE, CULTURE AND ENVIRONMENT AWARE PERSISTENCE, and STRUCTURE DRIVEN BY TARGET CULTURE. Most of these patterns abstract from the implementation in a concrete programming language. Rather, they present proven solutions to higher-level issues. So they primarily address people in an architecture or user experience role.

Another source for patterns on this topic is the “Internationalization” category of the Portland Pattern Repository’s Wiki.

The presentation of the patterns is similar to the style well known from [Alexander et al. 1977]. In part, these patterns are based upon other patterns. Typographic conventions for references to other patterns are also similar to [Alexander et al. 1977]. A Glossary provides thumbnails of many of these patterns and definitions of other terms used in this paper.

Categories and Subject Descriptors: D.2.2 [Software Engineering]: Design Tools and Techniques—*User interfaces*; D.2.7 [Software Engineering]: Distribution, Maintenance, and Enhancement—*Version control*; D.2.11 [Software Engineering]: Software Architectures—*Patterns*; H.1.2 [Models and Principles]: User/Machine Systems—*Software Psychology*; H.5.2 [Information Interfaces and Presentation (e.g. HCI)]: User Interfaces—*Ergonomics*

General Terms: Design

Additional Key Words and Phrases: Culture, Date, I18N, Internationalization, L10N, Language, Localization, Patterns, Time, User Experience

## ACM Reference Format:

Bachmann, Ph. 2013. Patterns for Internationalization and Cross-Cultural Usability, HILLSIDE Proc. of Conf. on Pattern Lang. of Prog. 20 (October 2013), 19 pages.

---

## 1. OVERVIEW

Enabling a software application to be used in different Cultures can vastly increase its market. Users will feel more at home then, become familiar with the application quicker and accept it more likely. The patterns in this catalog lay the foundation to later localize the application. This reasoning does not only hold for people from different cultures, but also for people with different learning styles[Fleming and Mills 1992]; the pattern MULTI-CODED INSTRUCTION addresses especially this fact.

The audience of this catalog consists of software architects and user interaction designers.

Table I lists the patterns proposed.

Many of these patterns share a common scheme: To build or adapt an application to more than one culture it needs to abstract from a single culture first. This implies generalizing any code that is somehow culture dependent. This step of building in the capability that an application can be adapted to other cultures at all is referred

---

This work is supported by iRIX Software Engineering AG.

Author’s address: Ph. Bachmann, c/o iRIX Software Engineering AG, Dornacher Str. 192, 4053 Basel, BS, Switzerland; email: bachlipp@web.de

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission. A preliminary version of this paper was presented in a writers’ workshop at the 20th Conference on Pattern Languages of Programs (PLoP). PLoP’13, October 23–26, Monticello, Illinois, USA. Copyright 2013 is held by the author(s). Parts of the Glossary first published in [Bachmann 2008]: Copyright 2008–2013 by the author and the Institute for Medical Informatics and Biostatistics, used here with permission from F. Hoffmann-La Roche Ltd. HILLSIDE 978-1-941652-00-8

Table I. : Thumbnails of Patterns for Internationalization and Cross-Cultural Usability

Section	Title	Intent
2.1	MULTI-CODED INSTRUCTION	Increase comprehensiveness of a user interface by means of redundant codings of the same information
2.2	MESSAGE CATALOG	Provide a foundation to adapt the user interface to different natural languages
2.3	INPUT EXAMPLE	Lower the burden imposed on the user when the application expects him or her to enter data in a certain format
3.1	CULTURALLY AND ENVIRONMENTALLY AGNOSTIC PERSISTENCE	Deal with culture dependent quantities like date and time that can be converted to another culture back and forth with virtually no losses in precision
3.2	CULTURE AND ENVIRONMENT AWARE PERSISTENCE	Deal with culture dependent quantities which unit is expected to be kept for future reference like a certain currency in the case of money
3.3	STRUCTURE DRIVEN BY TARGET CULTURE	Treat all cultures equally well to let any user feel at home

to as Internationalization, abbreviated as I18N. If an application has been built from several components, its Internationalization should be partitioned along the boundaries of these components both to allow for piecemeal Localization (see below) and to let each originator independently localize his or her component. Internationalization also implies the usage of a character set that contains the union of all characters valid in the cultures to support. Providing respective encodings is the goal of the Unicode standard, one example encoding from this standard is UTF-8.

The second step after internationalizing the application is the specialization to single cultures. This second step has to be repeated for every culture the application is intended to be rolled out to. This is referred to as Localization, abbreviated as L10N. Localization will best be carried out by people from the respective culture or at least by educated experts in the respective culture.

For the sake of understandability a simple categorization has been made: The patterns in Section 2. Application Data apply to interaction elements that are part of the software application. Examples are menu entries and labels. These elements are mainly only part of the presentation layer. The patterns in Section 3. User Data apply to business data that are e.g. being entered by the user or are otherwise not part of the product itself. These data are represented not only in the presentation layer, but in other LAYERS as well, last but not least in the persistence layer. The architectural style determines the exact boundary between the two categories. Compare the C statement `printf("Number: %d",42);` with `printf("%s %d", "Number:",42);`—both yield the same output “Number: 42”, but the latter treats the label the same way as the number. Compared to other architectural styles, in Adaptive Object Models[Yoder and Johnson 2002] more functionality is being similarly moved to those SHEARING LAYERS[Foote and Yoder 2000, pp 678–681] with a higher pace of change. In this case the solutions proposed in Section 2. Application Data would have to be moved to the SHEARING LAYERS that allow for higher adaptability at runtime with quite the same techniques as the functionality they are supposed to internationalize has already been shifted to those LAYERS with.

The presentation of the patterns is similar to the style well known from [Alexander et al. 1977]: Each pattern starts with a name. The first paragraph establishes the context, then after three diamonds follows the problem, which consists of a bold summary and a long body which explains the problem and the forces that have to be resolved. A bold solution summary is then introduced with the word “Therefore”. After the solution summary the solution and its consequences are elaborated on in more detail. Sometimes, another three diamonds set the stage for references to further patterns that complete this pattern.

In part, these patterns are based upon other patterns. Typographic conventions for references to other patterns are similar to [Alexander et al. 1977]. A Glossary provides thumbnails of many of these patterns and definitions of other terms used in this paper.

Another source for patterns on this topic is the “Internationalization” category of the Portland Pattern Repository’s Wiki: <http://www.c2.com/cgi/wiki?CategoryInternationalization>. Further sources of information are [International 2002] and [int 2003].

Accessibility considerations become increasingly important. Even though some of the patterns proposed may provide some guidance on how to improve accessibility, the improvement of an application for users with disabilities or special needs is beyond the scope of this pattern catalog.

## 2. APPLICATION DATA

Many interaction elements of a software application are part of the application itself and not meant to be changed by the user. Examples include menu entries and labels of buttons. These elements are mainly only part of the presentation layer, i.e. the View in a Model-View-Controller architecture [Buschmann et al. 2000b] or the Presentation in case the application realizes the Presentation-Abstraction-Control pattern [Buschmann et al. 2000d]. This section proposes patterns for this overall context.

### 2.1 Multi-Coded Instruction



2009, U.S. Department of Transportation [Federal Highway Administration 2009]

A graphical user interface offers a toolset to design interaction with the user. Some is driven by metaphors, some by convention. Each user is different.



**Different people need different media to comprehend best—some are more textually inclined, some more graphically. A user interface that presents only text to the user makes interaction hard for those who prefer graphics, an interface that solely consists of graphics is hard to understand for those who prefer textual labels. How to maximize comprehensibility of a graphical user interface?**

Information can be represented in different ways, e.g. graphically, textually etc. Each single representation alone might not provide the desired distinctness for a fraction of the audience: Up to 8% of the male (female to a much lesser extent) have red-green color deficiency in one way or the other<sup>1</sup>, so color alone will not work, at least 77% of the world population have no command of English [Mydans 2007], so text must be localized or supplemented with another representation of the same information. Even people who are able to interpret two specific representations have a penchant for one over the other. An application designer must have to take into account all of these users. It does not make sense to limit the reach of the application by stratifying the users

<sup>1</sup>Both the exact number and the sex ratio seem to depend on the epidemiological stratifier ethnicity.

into groups of different preferences and decide to support only one of them, because this stratification is likely to rather be artificial to the functionality of the application.

Pictograms annotated with tooltips still give a clear preference to those users who are good at memorizing the pictograms and acting accordingly. The other group of users has to move the mouse to get the same information from the tooltips the other group gets at a glance.

Especially mobile devices provide only limited screen sizes—this makes it even harder to design a comprehensible user interface.

Therefore:

**Use a minimum of two representations for every widget that will be perceived by the same kind of sensory organ of the user, e.g. a pictogram and a textual label. Make all visible at the same time. To further increase comprehensibility base these codings on a respective standard or *de facto* standard. You can go even further and ensure that the cultures the codings come from are not completely congruent—then the reach of the application will extend to people who understand only one of them.**

If the information will be represented in multiple different ways that are to be perceived by the same sense this is called a multi-coded presentation—if the representations are to be perceived by different senses, this is called a multimodal presentation. Both multi-coded and multimodal presentations intend to embrace different learning styles, in this pattern especially e.g. regarding the model published in [Fleming and Mills 1992] the Visual and Read / Write “perceptual modes”. To improve user perception, this pattern proposes multi-coded and not multimodal instruction, because the latter can easily cause overload and interferences of human perception finally resulting in distracting split attention, if the signals that stimulate different senses have not been carefully coordinated and synchronized.[Krummeck 2008, p 102] Furthermore, in general some modes may dominate others, e.g. in bi-sensory settings, visual stimuli dominate auditory[Tsay 2013] and haptic[Hecht and Reiner 2009] stimuli. Therefore multimodal are much more controversial than multi-coded presentations. Tri-sensory settings, however, do not exhibit this visual dominance.[Hecht and Reiner 2009]

Traffic signs are a good example for carefully designed multi-coded symbols: One coding is given by color (Portland Orange and white with U.S. pedestrian traffic lights or red for prohibition signs and blue—not green—for mandatory-signs in those countries that ratified the Vienna Convention on Road Signs and Signals), the other one by shape.

With Graphical User Interfaces, in the former OPEN LOOK interface by Sun Microsystems and AT&T Corporation iconified applications always have born both a pictogram and a textual label, see e.g. Figure 1a. Not all graphical user interfaces have followed this tradition—note that while the Dock in Apple OS X shows pictograms, but labels only in terms of tooltips, the smartphone operating system and interface Apple iOS shows both pictograms and labels permanently, thus follows the OPEN LOOK style again, see Figure 1b. Jensen Harris has shared the following insight regarding the Microsoft Ribbon:

“Something we’ve known from usability tests for several years now is that most people don’t click on an unlabeled 16x16 icon. Sure, Bold and Italic and Center and a few others get a lot of clicks, but the curve falls off after the first 8 or so. As a result, we try to label every command in the Ribbon.”[Harris 2005]

This pattern has similar implications as the Dual-Coding Theory by Allan Urho Paivio[Paivio 1971], but is more general: It suggests *at least* two codings and does not restrict one of them to text. Figure 1c shows an x mark and a check mark to indicate “no” and “yes”. Because these shapes have no unique meaning in different cultures, color is used as one orthogonal dimension—in this case, the colors of prohibitive and mandatory traffic signs as standardized by the Vienna Convention on Road Signs and Signals have been used not to place a burden on people with red-green color deficiency. Further, the filling is another orthogonal dimension, so in total three codings are used. The filling, however, does not carry any predefined meaning, so its primary purpose is to

enhance distinctness especially in black-and-white media. Pattern publications themselves are examples for this pattern: Most are illustrated by means of diagrams or photos to allow for easier access to them.

A limited number of pictograms seems to work even cross-culturally. Among those are the 50 pictograms issued by the U.S. Department of Transportation (DOT), which have been cross-culturally tested; the DOT pictograms are in the public domain.[Cook and Shanosky 1979] Figure 2 shows two examples of DOT pictograms that might be useful in the context of graphical user interfaces. Arrows seem to work cross-culturally, too, if taking the reading direction into account.[Fernandes 1995, pp 43–44]



If one of the codings is textual, you may want to build in a MESSAGE CATALOG to supply translations to several natural languages. See Section 2.2 for details.

## 2.2 Message Catalog

Language is the picture and counterpart of thought.

---

MARK HOPKINS

Still much of the user interface of an application makes use of natural language. Language is culture dependent.



**Applications place a burden on users if the native language of the user is not the language of the user interface. This increases the likelihood of making more mistakes in using the application and might ultimately result in abandoning this application. So what to do about the user interface when rolling out an application to different countries?**

It is hard enough to design a user interface to be usable within a single culture. There are nearly 7000 natural languages in the world—it is impossible to support all of them at once. So realistically only a subset will be supported. This subset might change—probably increase—over time. Language is not only vocabulary. Sentence structure also differs in different languages. The best translation is always to and not from his or her native language. It is not enough for an interface to follow a certain natural language; it should also adhere to the conventions set by the body of all applications in that language that predate the respective application. So a translation that is linguistically correct does not necessarily match the interaction expectations of the user he or she has acquired from having used other software—in English, for example, it would be quite confusing if a menu entry to terminate an application was labeled “File / Terminate” instead of “File / Exit”. The latter is more common not least because it is closer to the perspective of the user, who exists from the application.



(a) OPEN LOOK Sun Microsystems DeskSet Mailtool icon showing pictogram and label



(b) Apple iOS Email. The presentation follows the OPEN LOOK tradition.



(c) Rejection and confirmation by character, color and filling

Fig. 1: Three examples for multi-coding



(a) Mail



(b) Litter Disposal

Fig. 2: Two DOT pictograms as examples for one of at least two codings

1974, 1979, U.S. Department of Transportation  
[Cook and Shanosky 1979]

In general, natural language skills and programming skills are distributed among several persons. The application probably evolves over time, so its interface changes—then what about its several instances in foreign languages? Every non-trivial application is structured in terms of modules or libraries. Some of them probably come from external sources. Those of them that generate output or process user input show through the user interface of the final application. Responsiveness is important for user acceptance. Letting the natural language shown in a user interface dynamically depend on a language setting must not drain performance too much.

Therefore:

**Substitute a call to a lookup function into a culture-specific translation table for all occurrences of output strings in a natural language. These translation tables are commonly referred to as MESSAGE CATALOGS or Resource Bundles. Keep the substituted string as the key for the respective lookup. If lookup fails, the function should simply return the key passed. This is an instance of Internationalization as described in Section 1. Overview. Then hand over the task of creating concrete instances of these translation tables to specific cultures to native speakers in the respective natural language. This is an example for Localization as described in the section referred to. You will end up with one MESSAGE CATALOG per supported natural language and module of the application.**

Because text lengths in different translations vary quite a lot, first the application layout needs to be generalized such that it can adapt to different sizes of textual labels afterwards. So instead of specifying positions and sizes of textual interaction elements like buttons in terms of fixed numbers the layout should be delegated to a layout manager that arranges these elements relatively to each other taking their individual sizes into account. Most widget sets provide such functionality; a Java example for a layout manager is `java.awt.FlowLayout`. Especially on mobile devices with limited screen sizes the length of labels might have to be restricted to an absolute maximum across all cultures; in this case the choice among alternative translations of the same label will also be driven by the available space.

The solution proposes to use natural language keys for the lookup of translations. This has three consequences: Both the code and the translation tables remain readable, which would not have been the case if the solution would have proposed e.g. integral numbers or even UUIDs as surrogate keys instead. Translations can be added in a piecemeal manner, because the lookup function defaults to returning the key, if no appropriate translation can be found; especially there is no need for a translation table targeting the natural language of the keys. On the other hand, translations should be worked on not until these keys have somewhat stabilized, because the translators have to touch all existing translations when a key has changed interim.

One well established Internationalization library is GNU `gettext`, which does not interpret the translation tables, but uses tables compiled before to speed up lookup. Figure 3 shows two parallel examples of MESSAGE CATALOG from the `gettext` distribution itself in Finnish and Turkish before compilation using `msgfmt`. One possible usage would read `printf(gettext("write error"));`

`Gettext` can use different tables for different modules (“text domains”) of the application, when `dgettext()` is being used instead of `gettext()`. `Gettext` allows Localizations to evolve more or less independently from the internationalized application itself by means of so-called fuzzy entries, thus can be used to relax baselining

<pre># Copyright © 2007 Free Software Foundation, Inc. # Sami J. Laine &lt;sami.laine@iki.fi&gt;, 2002. # Lauri Nurmi &lt;lanurmi@iki.fi&gt;, 2007. ... #: gnuilib-lib/closeout.c:66 msgid "write error" msgstr "virhe kirjoitettaessa"  #: gnuilib-lib/error.c:125 msgid "Unknown system error" msgstr "Tuntematon järjestelmän virhe"  #: gnuilib-lib/getopt.c:530 gnuilib-lib/getopt.c:546 #, c-format msgid "%s: option '%s' is ambiguous\n" msgstr "%s: valitsin \"%s\" on moniselitteinen\n" ...</pre>	<pre># Copyright (C) 2006 Free Software Foundation, Inc. # Nilgün Belma Bugüner &lt;nilgun@buguner.name.tr&gt;, 2001, # ..., 2006. ... #: gnuilib-lib/closeout.c:66 msgid "write error" msgstr "yazma hatası"  #: gnuilib-lib/error.c:125 msgid "Unknown system error" msgstr "Bilinmeyen sistem hatası"  #: gnuilib-lib/getopt.c:530 gnuilib-lib/getopt.c:546 #, c-format msgid "%s: option '%s' is ambiguous\n" msgstr "%s: '%s' seçeneği belirsiz\n" ...</pre>
<p>(a) From the Finnish file <code>gettext-runtime/po/fi.po</code>, revision 1.11</p>	<p>(b) From the Turkish file <code>gettext-runtime/po/tr.po</code>, revision 1.8</p>

Fig. 3: Small fractions of `gettext MESSAGE CATALOGS` from `gettext`[Drepper et al. 2012] showing keys in English. Note the differences in punctuation as well.

as explained by pattern `COMPONENT BASELINE` by Ralph Thiim and Lise B. Hvatum[Thiim and Hvatum 2012, pp 8–11]. For C++, `Boost.Locale`[Beilis 2011] was an alternative to consider. One of its features is “Messages Formatting (Translation)”, and this interacts well with the existing `gettext` tools.

When an application is going to be localized to cultures which share the same natural languages or dialects thereof, it might be a valid simplification to start with not to consider the regional dialects. The most important point here is to decide on a single variety to support—i.e. not to mix up e.g. British and American English. To non-native speakers given the task to localize the application this might pose a challenge. This consistency requirement also includes e.g. typography. When it then turned out that there is a significant market in regions sharing the same natural language, but in different varieties, then it is time to refine the Localizations to consider these varieties to let more users feel really at home.

Be aware that the realm of the natural language might not be congruent with the realm of other aspects that have to be considered when adapting an application to other cultures. All of these aspects are referred to as `Cultural Facets`, besides natural language e.g. a unit system for quantities. Consider e.g. the different medical units used in different regions where German is spoken: In Northern and Central Switzerland the native language is a dialect of German, and as e.g. in former Eastern Germany the medical units are in accordance with the International System of Units (SI), e.g.  $\text{mmol l}^{-1}$  for blood sugar. In (former Western) Germany, however, many clinical laboratories and medical devices use units that predate the SI system, e.g.  $\text{mg dl}^{-1}$ .

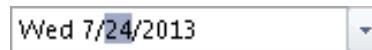
When implementing `MESSAGE CATALOG` it has to be somehow decided how the application should figure out the culture the user belongs to. This is not trivial, so it pays off to negotiate this topic with user experience experts. The most automated way is probably to use the cultural settings of the operating system, which in turn are bound to the currently logged in user.



Also processing of date and time, numbers etc. is affected as well. Some of these cases can be treated using the built-in facilities of the respective standard library, e.g. `stream facets` in C++ to change formatting of numbers, dates etc. [American National Standards Institute 2003, pp 415–463] and date / time classes or functions for dealing with different time zones. Quantities measured in a certain culture-specific unit, currencies etc. are

also affected. For quantities generalization implies transforming with a cultures-specific formula and exchanging the respective unit accordingly. This is further elaborated on in patterns CULTURALLY AND ENVIRONMENTALLY AGNOSTIC PERSISTENCE (see Section 3.1) and CULTURE AND ENVIRONMENT AWARE PERSISTENCE (see Section 3.2).

### 2.3 Input Example



Computer applications are not television sets—output is one thing, but at some point the user will input some data thus closing the interaction cycle.



**Sometimes a single text field allows for more efficient data entry than a set of fields or special widget even if the data to be entered is expected to have a certain kind of structure, because the user can stick with her or his keyboard and does not have to alternate between keyboard and mouse. If data can be entered by means of a widget specially designed for data structured this way, e.g. a calendar widget to enter dates, the user experience expert might also provide a text field to allow an alternative way for the user to enter the data directly—this would also enable him or her to copy and paste all data at once. How does the user know the expected format of the data to enter that second way?**

Many users have a history with computers. They have experienced the consequences of a multitude of design decisions: Sometimes they have to always adapt to the U.S. way to format numbers and dates, sometimes formatting and even programming languages have been localized, e.g. Microsoft Visual Basic for Applications for Excel 95. So they are uncertain of what format an application expects as input. The conventions differ even between countries with the same natural language.

Obvious examples are number and date formats. Another example are conversion factors, where the user might not instantly know whether to type in a certain number or its reciprocal or negative counterpart. These include currency exchange rates and factors or summands to convert between different systems of physical units. Errors in entry of such data can cause severe failures—and are hard to detect because such not round numbers are suggestive of precision, and precision has persuasive power, be the numbers correct or not.

Therefore:

**For each text field provide example input. This can be done by means of a default entry, if there exists some plausible value at all, or by text in a dedicated column in close proximity to the respective text field. If the anticipated screen size allows so, do not use tooltips but text permanently visible.**

If examples are presented separately from the field, i.e. not as default entries, then present them to the user in a way he or she can copy them from the example column and paste them into the respective text field. This is the same idea that also spawned the PROTOTYPE pattern.

Interpreting user input is always harder than outputting some value, because it is important to allow for some variability in formatting by the user as long as his or her input is not ambiguous.

Microsoft Office 2010 Outlook provides two ways to enter a calendar date: One via the text field printed on top of this pattern and the other one by means of the calendar representation shown in Figure 4, which follows the CALENDAR PICKER pattern described in [yah 2009]. The text field shows today's date as the default and can be edited by the user providing an alternative way to change the date to using the CALENDAR PICKER. If the user adapts the day via the text field, the application will automatically change the day of the week accordingly.

## 3. USER DATA

Data which change require a different treatment than data the application designer has considered constant. Such variable data include values entered or changed by the user, information fetched from a database and data that



Fig. 4: CALENDAR PICKER of Microsoft Office 2010 Outlook

are otherwise not part of the software application itself. Such data crosses more LAYERS than just the presentation layer, i.e. all three respective entities both in the Model-View-Controller architecture[Buschmann et al. 2000b] and in the Presentation-Abstraction-Control pattern[Buschmann et al. 2000d]. This section proposes patterns for this overall context.

The following patterns do not elaborate on the concrete choice of a numerical representation that appropriately models the respective data. This is another topic, and one respective rule “avoid float and double if exact answers are required” is discussed in [Bloch 2008, pp 218–220]. In pattern CULTURALLY AND ENVIRONMENTALLY AGNOSTIC PERSISTENCE the assumption is being made that the numerical representation is culture-independent, and even pattern CULTURE AND ENVIRONMENT AWARE PERSISTENCE still assumes that different numerical representations, if there are more than one at all, of the same entity for different cultures can at least be somehow converted into one another.

### 3.1 Culturally and Environmentally Agnostic Persistence

Data is being passed around the world. Client / server applications may have clients localized by means of MESSAGE CATALOG (see Section 2.2) and central servers with data repositories that span all cultures of the clients. The units of the quantities stored in these repositories differ from culture to culture. It is not required to keep track of the quantity in its original unit. The losses in precision multiple conversions back and forth may yield are tolerable.



**Consider a case where data from different cultures is not just stored in a central place, but also aggregated across cultures, e.g. compared with each other or summed up. Facing the cultural differences: How should the data from different cultures be persisted?**

Travellers know that date and time are different around the globe; these differences must be taken care of not only on a global scale, but already within large countries like the U.S. and even within some single states: The U.S. state of Indiana, for example, mainly observes Eastern Time (Coordinated Universal Time (UTC)–5 during the winter), but shares the Chicago, IL, metropolitan area in its northwest with Illinois and Wisconsin and the Evansville, IN, metropolitan area in its southwest with Illinois and Kentucky, both where Indiana observes Central Time (UTC–6 during the winter). Storing quantities in their local unit only works until the need arises to e.g. compare them across cultural boundaries. A date and time stored without reference to its time zone is called local time according to ISO 8601—and is probably the worst way to store it because no judgement can be made

about the permissibility of an operation involving a second local time, which may or may not come from the same time zone.

Different from numbers strings are not subject to arithmetical operations. But even those might pose challenges when storing them in a central place. Different character sets make it difficult to exchange them crossing cultural boundaries. Given the strings are intended to be interpreted as e.g. numbers or dates, this interpretation likely does not work from another culture than those that persisted the number or date.

Therefore:

**Process and store such data in a culturally and environmentally agnostic way and do the conversion to and from this common representation upon storing and reading it.**

This is a simple solution, even though it involves two conversions, one on writing, the other one on reading and presenting the data, even if the presentation takes place in the same culture the data was entered in. Note that conversions involving only integral number representations and addition are lossless back and forth aside from the risk of arithmetic overflow. And even potential losses resulting from conversions of floating point numbers or involving multiplications may happen at a scale that can well be tolerated.

Decide an internally used unit of a quantity first. Always keep in mind that this convention is an implementation detail. In case of date and time use e.g. UTC.

The same considerations apply to the way the libraries used adapt to cultural differences. Operating systems and libraries group the aspects that define a culture, the Cultural Facets—e.g. what character is used as a decimal point, which is the currency, etc.—, into a so-called Locale. Often, an application implicitly depends on a Locale fed into the application by means of an environment variable. While this is quite useful for end-user applications that manage persistence on their own, it is a bad idea for centralized server applications that have to deal with input from multiple cultures and store it. Most libraries apply Localization to serialization and deserialization regardless of whether the sink or source is the screen or a file, i.e. regardless of whether the presentation or persistence layer is affected. Microsoft Excel 2010, for example, not only visualizes numbers and dates Locale dependent, but also stores CSV files in a Locale dependent way; not even the delimiter is guaranteed to be a comma throughout the globe. Thus CSV files as understood and written by Microsoft Excel 2010 do not always conform to the format retrospectively defined in [Shafranovich 2005]. This pattern suggests to do it differently. So from within the code explicitly set the Locale right after application start to a fixed one or—even better—use those serialization and deserialization routines that explicitly take a Locale parameter instead of magically fetching this information from the environment. In case of UNIX Locales prefer the “C” Locale, because it is the only Locale available on every system.

Every modern operating system provides routines in its standard libraries to convert local date and time to and from UTC considering Daylight Saving Time, if any, or convert strings into instances of intrinsic data types and back using the formatting rules given by the Locale of the currently logged in user.

Regarding date and time there is another point to consider: Often, some data is associated with fixed time intervals, e.g. a day. In any other time zone this data will span two days, however. It might make sense to discretize the day to the smallest difference between all time zones in the world, which currently is 15 min. Then for every location in the world it requires the same calculations to display the data connected with an interval in the respective local time zone.

For physical quantities a culture-specific table can be maintained that maps the respective unit to the arithmetic operation required to express the same quantity in the standard unit the quantity is stored in internally. Note that depending on the unit a factor is not enough as sometimes the conversion is not multiplicative, but additive—or even logarithmic. Conversion between °C and °F is an example that involves both multiplication and addition, whereas conversion from a sound frequency to a musical key is logarithmic.

Converting a quantity from one unit to another might require an explicit designator of the unit that is not culture-specific. In case of time zones, the simplest representation is their respective offset from UTC, which changes with Daylight Saving Time, however. Many libraries also support the designators from the Time Zone Database

like `Australia/Lord_Howe`, that maps such strings to offsets from UTC taking Daylight Saving Time into account, e.g. Joda-Time and the IBM International Components for Unicode; if the latter designators change over time, the Time Zone Database links obsoleted designators to current ones, thus maintaining Portability in time.

When implementing CULTURALLY AND ENVIRONMENTALLY AGNOSTIC PERSISTENCE it has to be somehow decided how the application should figure out the culture the user belongs to. This is not trivial, so it pays off to negotiate this topic with user experience experts. The most automated way is probably to use the cultural settings of the operating system, which in turn are bound to the currently logged in user; in case of date and time in web applications note that HTTP does not transmit the time zone of the client to the server—therefore the only automatic solution is via client-side ECMA Script<sup>2</sup> given you do not intend to personalize the application and bind that information to the user, who would be quite confused then when he or she travels, however. Sometimes, the users might want to have more explicit choice regarding certain Cultural Facets, e.g. they know better than the application designer whether they prefer to see time in the their home time zone or always converted to the time zone they are currently in.

Certain domains bear specific conventions regarding fixed points of reference in presentation to the user, however. In these cases the pattern still applies, but the choice of the unit does not depend on the user. Because some users might not be aware of the respective convention, it is even more important to point him or her to it in the user interface. An example is travel. Date and time are almost always shown in the local time relative to the respective place, be it e.g. the starting point or the destination, and not in the local time of the user. To be able to calculate traveling times it is still advantageous to store any such time e.g. in UTC.

This is a similar solution as the one described in the pattern CANONICAL DATA MODEL [Hohpe and Woolf 2004a]; the context of the pattern referred to is messaging between several enterprise applications, however, whereas CULTURALLY AND ENVIRONMENTALLY AGNOSTIC PERSISTENCE affects persistence across multiple cultures within a single application.

### 3.2 Culture and Environment Aware Persistence

Data is being passed around the world. Client / server applications may have clients localized by means of MESSAGE CATALOG (see Section 2.2) and central servers with data repositories that span all cultures. The units of the quantities stored in these repositories differ from culture to culture. The application is expected to keep track of the unit a quantity has originally been entered in or has to reduce losses in precision due to conversions of a quantity from one unit to another one to the least possible.



**Consider a case where data from different cultures is not just *stored* in a central place, but also *aggregated* across cultures, e.g. compared with each other or summed up. Facing the cultural differences: How should the data from different cultures be persisted?**

Travellers know that currencies vary in different countries. The same applies to other units as well. Even if the unit does not seem to differ, there might be a “hidden” point the unit actually refers to that differs from country to country: Elevation above sea level in continental Europe is given in m, but the details differ: Somewhat simplified, the reference might be the Amsterdam Ordnance Datum as e.g. for the Netherlands and for Germany, the tide gauge at Marseille as e.g. for mainland France and Switzerland, the Sartorio mole in the Port of Trieste as for

---

<sup>2</sup>Note that the constructor of the standard `Date` class in ECMA Script taking a date string, which runs the same parser as `Date.parse()` does, exhibits inconsistent behavior in different implementations of ECMA Script, especially among different web browsers. Only a subset of ISO 8601 is supported by most. So it requires either some experimentation to find the intersection of formats accepted by the engines to support or engine-specific modification of the input string before creating an instance of `Date`. In other words, code passing dates in terms of strings to the ECMA Script class `Date` is not portable. On a German system with time zone `Europe/Berlin` `Date('2009/04/22 23:52 +01').toLocaleString()` correctly yields “Donnerstag, 23. April 2009 00:52:00” by more than one web browser, so this string representation of date and time was a good starting point.

Austria, but not for Italy, which uses the Genoa tide gauge. Even though there are standardization efforts, in Europe currently only 14 countries participate in those—less than 30%. Therefore for quite some time there will still be different reference points for different countries even within this single continent.

Some conversion algorithms prove quite volatile; exchange rates are in a state of flux, and services feeding the current rate accessed through a computer network are as a matter of principle not that available as local functions would be. With money, furthermore, at least retail currency exchange is not transitive—it is more expensive to indirectly convert a currency into another one than doing so directly, because buying and selling rates contain some fees for the dealer; therefore converting e.g. USD first into GBP and then GBP into EUR costs roughly twice as much than directly converting USD into EUR.

Both money and elevation above sea level are examples for quantities which original units likely are to be kept by the application for retrieval later on.

Therefore:

**Make persistence culture and environment aware: Store both the quantity in a format that is local to a culture and also store an unambiguously interpretable reference to that culture in close association with the quantity, i.e. store the amount of the quantity in a culture-specific unit and store this unit in a way that is not culture-specific. As long as data from the same culture is being processed, leave the data as-is. Otherwise dynamically convert it.**

Never simply store data in a central place in a format that is local to a culture without additionally storing an unambiguously interpretable reference to that cultural differentiator or Cultural Facet—otherwise you would not know later on how to interpret those amounts. The key of the Cultural Facet is the remaining small part of the information that needs to be persisted in a culturally and environmentally agnostic way; see Section 3.1. CULTURALLY AND ENVIRONMENTALLY AGNOSTIC PERSISTENCE for details.

In the case of e.g. money, upon first entry record both the amount and the currency, the latter probably in terms of the currency code according to ISO 4217, e.g. USD for United States dollar. If this quantity is later requested from within the same culture, simply present the persisted amount and the currency. Otherwise call a conversion routine to express the quantity in terms of the currency requested. When a user changes the amount in his or her local currency different from the original currency, convert the quantity back to the currency the quantity has been created in.

Within a culture numbers get processed at maximum accuracy. Processing data across cultural boundaries might cause losses in accuracy due to the conversion involved then.

The same considerations apply to the way the libraries used adapt to cultural differences. Operating systems and libraries group the aspects that define a culture, the Cultural Facets—e.g. what character is used as a decimal point, which is the currency, etc.—, into a so-called Locale. If the application needs to store strings entered by the user that contain e.g. numbers, then the key of the Locales valid for this user needs to be stored along with the string. Otherwise it is quite unlikely that the application can interpret the string as a number in the future. Again, the Locales key itself needs to be persisted in a culturally and environmentally agnostic way.

When implementing CULTURE AND ENVIRONMENT AWARE PERSISTENCE it has to be somehow decided how the application should figure out the culture the user belongs to. This is not trivial, so it pays off to negotiate this topic with user experience experts. The most automated way is probably to use the cultural settings of the operating system, which in turn are bound to the currently logged in user. Sometimes, however, the users might want to have more explicit control over certain Cultural Facets, e.g. they know better than the application designer whether they prefer monetary values to be shown in the currency the amounts once have been entered in, even if this is not the currency of the respective user, or always converted to the currency of the user.

### 3.3 Structure driven by Target Culture

Data that consist of distinct information like parts of names, phone numbers, geographical location etc. is entered and stored in a fielded form to let the computer more easily grasp that structure than when letting the user enter it as unstructured full text.



**Applications often force the user to enter data with an implicit structure in a structured way because they store them preserving structure. Sometimes the structure is specific to the culture of the content, however. Telephone numbering plans differ around the world, postal addressing schemes also do because postal systems differ. The quality of data will suffer for the majority of cultures that do not share the way to structure the data with the culture the application was originally developed for. How to design data entry and persistence to ensure a high quality of data entered for all cultures anticipated?**

Figure 5 shows the dialog Microsoft Office 2010 Outlook presents on entering a new address of a contact. Even though it is open to more than one culture by mentioning more than one term in three of its five labels, both the choice of fields and their order clearly stresses its origin. For addresses in Germany, for example, apart from domains like education and construction where state legislation is important it is quite unusual to maintain the state in address records, and the city comes after the ZIP code. If the person who enters an address into a form similar to this Figure takes the data from a system that distinguishes between street name and house number, he or she might have difficulties merging both into the single street address field in the order correct with regard to the culture the address refers to. For example, a valid street address in Norway reads “Storgata 57”, whereas a valid British address reads “57 King Street”.

Figure 6 shows the dialog Microsoft Office 2010 Outlook presents on entering a new phone number of a contact. For Germany extensions might be useful for people inside the same organization, but from the outside they do not need to be treated in a special way, because direct inward dialing is the norm. In 2002 Switzerland has implemented full-number dialing and number portability throughout the country, therefore for Swiss phone numbers there is no real need to distinguish between a city code and a local number any more.

Therefore:

**Present different forms to the user depending on the culture of the data to be entered and store the input accordingly. Because there are many cultures you cannot support in the first place, provide a default to handle even them. Later, specific implementations can be substituted for the default piece by piece.**

Apply this pattern recursively if you also want to precisely model subcultures, e.g. in the case of small villages where houses are uniquely identified just by a number you can drop the street name field given there has been a differentiation between street name and house number. Then you end up with not only a flat structure, but with a hierarchy that is driven by target culture. Resist the temptation to structure the data too much, however.

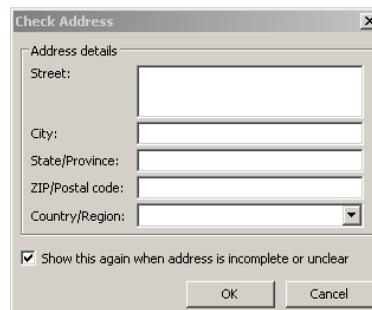


Fig. 5: Address dialog of Microsoft Office 2010 Outlook

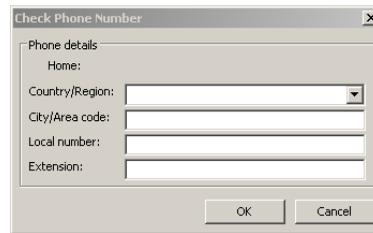


Fig. 6: Phone number dialog of Microsoft Office 2010 Outlook

A single street address field, into which the data can simply be copied verbatim from e.g. a letterhead might prove more useful than splitting the field up into a street name and a house number, because copying the whole street address preserves the order correct in the target culture, whereas manually splitting a concrete record and especially assembling it again is error-prone for people from a different culture.

For the labels consider to apply the pattern MESSAGE CATALOG (see Section 2.2)—this reduces the number of forms that would only differ in the natural language of the labels, but not in structure.

In the case of addresses the choice of a key could be e.g. the ISO 3166-1 alpha-2 country code or the name of the country in either French, the official language of the Universal Postal Union (UPU), or English, the other language accepted by the UPU. In case of phone numbers taking the country calling code as a key is a natural choice.

Implementation-wise, the most obvious way were `switch` statements. More advanced implementations probably will combine the design patterns FACTORY METHOD [Gamma et al. 1996d] and BRIDGE [Gamma et al. 1996b] instead. F# and Scala give the programmer discriminated unions to express this idea in an even more concise way, taking the target culture as the tag.

Using this pattern has the liability that search cannot be implemented in a unified way any more. One refinement therefore is to consider not just the input, but also formatted output and search as being culture-specific.

See [Fernandes 1995, pp 64–67] for further details on this topic.

## APPENDIX

### REFERENCES

- 1995, 2003. The Java Tutorials. Trail: Internationalization. Published online (1995, 2003). Retrieved July 23, 2013 from <http://docs.oracle.com/javase/tutorial/i18n/>
2009. Yahoo! Design Pattern Library: Calendar Picker. Published online (July 15, 2009). Retrieved July 23, 2013 from <http://developer.yahoo.com/ypatterns/selection/calendar.html>
- Ademar Aguiar and Joseph Yoder (Eds.). 2008. *PLoP '08: Proceedings of the 15th Conference on Pattern Languages of Programs*. ACM, New York, NY, USA.
- Christopher Alexander, Sara Ishikawa, and Murray Silverstein. 1977. *A Pattern Language. Towns, Buildings, Construction*. With MAX JACOBSON, INGRID FIKSDAHL-KING and SHLOMO ANGEL. Oxford University Press, New York.
- American National Standards Institute 2003. *ISO / IEC 14882:2003(E). Programming languages—C++. Langages de programmation – C++, International Standard* (second ed.). American National Standards Institute, New York, NY.
- Philipp Bachmann. 2008. Deferred cancellation. A behavioral pattern, see Aguiar and Yoder [2008], 1–17. DOI:<http://dx.doi.org/10.1145/1753196.1753218>
- Elisa Baniassad and Sebastian Fleissner. 2006. The Geography of Programming. In *Companion to the 21st ACM SIGPLAN Symposium on Object-oriented Programming Systems, Languages, and Applications (OOPSLA '06)*. ACM, New York, NY, USA, 510–520. DOI:<http://dx.doi.org/10.1145/1176617.1176625>
- Artyom Beilis. 2011. Boost.Locale. Published online (Nov. 2011). Retrieved January 12, 2014 from <http://www.boost.org/libs/locale/>
- Joshua Bloch. 2008. *Effective Java* (second ed.). Addison-Wesley, Upper Saddle River, NJ · Boston · Indianapolis · San Francisco · New York · Toronto · Montreal · London · Munich · Paris · Madrid · Capetown · Sydney · Tokyo · Singapore · Mexico City.

- Jan Bosch, Morven Gentleman, Christine Hofmeister, and Juha Kuusela (Eds.). 2002. *Software Architecture: System Design, Development and Maintenance, IFIP 17th World Computer Congress—TC2 Stream / 3rd IEEE/IFIP Conference on Software Architecture (WICSA3)*. Kluwer Academic Publishers, Dordrecht, Netherlands.
- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 2000a. *Layers* (1998, 1. korr. Nachdruck), Chapter 2: Architekturmuster, 32–53. In Buschmann et al. [2000c]. German translation of “Layers”.
- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 2000b. *Model-View-Controller* (1998, 1. korr. Nachdruck), Chapter 2: Architekturmuster, 124–144. In Buschmann et al. [2000c]. German translation of “Model-View-Controller”.
- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 2000c. *Pattern-orientierte Softwarearchitektur. Ein Pattern-System, deutsche Übersetzung von CHRISTIANE LÖCKENHOFF* (1998, 1. korr. Nachdruck). Addison-Wesley. An Imprint of Pearson Education, München · Boston · San Francisco · Harlow, England · Don Mills, Ontario · Sydney · Mexico City · Madrid · Amsterdam. German translation of “Pattern-Oriented Software Architecture. A System of Patterns”.
- Frank Buschmann, Regine Meunier, Hans Rohnert, Peter Sommerlad, and Michael Stal. 2000d. *Presentation-Abstraction-Control* (1998, 1. korr. Nachdruck), Chapter 2: Architekturmuster, 145–169. In Buschmann et al. [2000c]. German translation of “Presentation-Abstraction-Control”.
- Roger Cook and Don Shanosky. 1974, 1979. [DOT pictograms]. Published online (1974, 1979). Retrieved July 23, 2013 from <http://www.aiga.org/symbol-signs/>
- Ulrich Drepper, Peter Miller, Bruno Haible, Karl Eichwalder, and Michele Cicciotti. 2012. GNU gettext utilities. Version 0.18.2. Published online (Dec. 2012). Retrieved January 12, 2014 from <https://www.gnu.org/software/gettext/>
- Federal Highway Administration. 2009. *Manual on Uniform Traffic Control Devices. For Streets and Highways*. Claitor's Law Books and Publishing, Baton Rouge, LA, USA. Retrieved July 23, 2013 from [http://mutcd.fhwa.dot.gov/kno\\_2009r1r2.htm](http://mutcd.fhwa.dot.gov/kno_2009r1r2.htm)
- Tony Fernandes. 1995. *Global Interface Design [A Guide to Designing International User Interfaces]*. AP PROFESSIONAL, Boston · San Diego · New York · London · Sydney · Tokyo · Toronto.
- Neil D. Fleming and Colleen Mills. 1992. Not Another Inventory, Rather a Catalyst for Reflection. *To Improve the Academy* 11 (1992), 137–155. Retrieved September 27, 2013 from [http://www.vark-learn.com/documents/not\\_another\\_inventory.pdf](http://www.vark-learn.com/documents/not_another_inventory.pdf)
- Brian Foote and Joseph Yoder. 2000. Big Ball of Mud. See Harrison et al. [2000], Chapter 29, 653–692. Retrieved January 12, 2014 from <http://www.laputan.org/mud/mud.html>
- Richard P. Gabriel. 2002. *Writers' Workshops & the Work of Making Things. Patterns, Poetry...* Addison-Wesley, Boston · San Francisco · New York · Toronto · Montreal · London · Munich · Paris · Madrid · Capetown · Sydney · Tokyo · Singapore · Mexico City.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1996a. *Abstrakte Fabrik* (dritter, unveränderter Nachdruck), Chapter 3: Erzeugungsmuster, 107–118. In *Professionelle Softwareentwicklung* Gamma et al. [1996c]. German translation of “Abstract Factory”.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1996b. *Brücke* (dritter, unveränderter Nachdruck), Chapter 4: Strukturmuster, 186–198. In *Professionelle Softwareentwicklung* Gamma et al. [1996c]. German translation of “Bridge”.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1996c. *Entwurfsmuster. Elemente wiederverwendbarer objektorientierter Software, deutsche Übersetzung von DIRK RIEHLE* (dritter, unveränderter Nachdruck). Addison-Wesley-Longman, Bonn · Reading, Massachusetts · Menlo Park, California · New York · Harlow, England · Don Mills, Ontario · Sydney · Mexico City · Madrid · Amsterdam. German translation of “Design Patterns. Elements of Reusable Object-Oriented Software”.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1996d. *Fabrikmethode* (dritter, unveränderter Nachdruck), Chapter 3: Erzeugungsmuster, 131–143. In *Professionelle Softwareentwicklung* Gamma et al. [1996c]. German translation of “Factory Method”.
- Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. 1996e. *Prototyp* (dritter, unveränderter Nachdruck), Chapter 3: Erzeugungsmuster, 144–156. In *Professionelle Softwareentwicklung* Gamma et al. [1996c]. German translation of “Prototype”.
- Edward T[witchell] Hall. 1959. *The Silent Language*. Doubleday, Garden City, NY.
- Edward T[witchell] Hall. 1976. *Beyond Culture* (second ed.). Anchor Books, Garden City, NY.
- Christian Hammer. 2010. *Is Beyond Budgeting peculiarly Scandinavian, and if so what does it entail for its applicability across the world? A Cultural Analysis of Beyond Budgeting, SNF Project No. 7980, Beyond Budgeting – Research Program, the project is financed by Statoil ASA*. Technical Report SNF Report No. 13/10. Samfunns- og næringslivsforskning AS, Norges Handelshøyskole, Bergen. Retrieved February 18, 2014 from <http://www.nhh.no/Files/Filer/institutter/rrr/Beyond%20Budgeting/Hammer.pdf>
- Jensen Harris. 2005. An Office User Interface Blog. Enter the Ribbon. Published online (15 Sept. 2005). Retrieved May 3, 2014 from <http://blogs.msdn.com/b/jensenh/archive/2005/09/14/467126.aspx>
- Neil Harrison, Brian Foote, and Hans Rohnert (Eds.). 2000. *Pattern Languages of Program Design*. The Software Patterns Series; ed. by JOHN M. VLISSIDES, Vol. 4. Addison-Wesley. An imprint of Addison Wesley Longman, Inc., Reading, Massachusetts · Harlow, England · Menlo Park, California · Berkeley, California · Don Mills, Ontario · Sydney · Bonn · Amsterdam · Tokyo · Mexico City.
- David Hecht and Miriam Reiner. 2009. Sensory dominance in combinations of audio, visual and haptic stimuli. *Experimental Brain Research* 193, 2 (Feb. 2009), 307–314. Retrieved February 3, 2014 from <http://dx.doi.org/10.1007/s00221-008-1626-z>

- Geert Hofstede, Gert Jan Hofstede, and Michael Minkov. 2010. *Cultures and Organizations. Software of the Mind, Intercultural Cooperation and Its Importance for Survival* (third ed.). McGraw-Hill, New York.
- Gregor Hohpe and Bobby Woolf. 2004a. *Canonical Data Model* (14th printing May 2010), Chapter 8: Message Transformation, 355–360. In *The Addison Wesley Signature Series*; KENT BECK, MIKE COHN, and MARTIN FOWLER, *Consulting Editors* Hohpe and Woolf [2004b].
- Gregor Hohpe and Bobby Woolf. 2004b. *Enterprise Integration Patterns. Designing, Building, and Developing Messaging Solutions, with Contributions by* KYLE BROWN, CONRAD F. D'CRUZ, MARTIN FOWLER, SEAN NEVILLE, MICHAEL J. RETTIG, JONATHAN SIMON (14th printing May 2010). Addison-Wesley, Boston · San Francisco · New York · Toronto · Montreal · London · Munich · Paris · Madrid · Capetown · Sydney · Tokyo · Singapore · Mexico City.
- Robert J. House, Paul J. Hanges, Mansour Javidan, Peter W. Dorfman, and Vipin Gupta (Eds.). 2004. *Culture, Leadership, and Organizations. The GLOBE Study of 62 Societies*. SAGE Publications, Thousand Oaks, CA · London · New Delhi.
- Ronald [F.] Inglehart and Christian [P.] Welzel. 2010. Changing Mass Priorities: The Link between Modernization and Democracy. *Perspectives on Politics* 8 (June 2010), 551–567. Issue 02. DOI:<http://dx.doi.org/10.1017/S1537592710001258>
- Dr. International. 2002. *Developing International Software* (2nd ed.). Microsoft Press, Redmond, WA, USA.
- Rilla Khaled, Pippin Barr, Ronald Fischer, James Noble, and Robert Biddle. 2006. Factoring culture into the design of a persuasive game, See Robertson [2006], 213–220. DOI:<http://dx.doi.org/10.1145/1228175.1228213>
- Vanessa Krummeck. 2008. *Multimediale, multimediale, multimodale und interaktive Komponenten in mathematischen Lernumgebungen*. Dissertation. Lehrstuhl für Geometrie und Visualisierung, Fakultät für Mathematik, Technische Universität, München, Germany. Retrieved September 25, 2013 from <http://nbn-resolving.de/urn/resolver.pl?urn:nbn:de:bvb:91-diss-20071213-635943-1-7>
- Brenda Laurel. 1993. *Computers as Theatre*. Addison-Wesley, Boston · San Francisco · New York · Toronto · Montreal · London · Munich · Paris · Madrid · Capetown · Sydney · Tokyo · Singapore · Mexico City.
- Richard D[onald] Lewis. 2006. *When Cultures Collide. Leading Across Cultures, a Major New Edition of the Global Guide* (third ed.). Nicholas Brealey International, Boston, MA · London.
- Seth Mydans. 2007. Across cultures, English is the word. *The New York Times* (May 14, 2007). Retrieved May 19, 2013 from <http://www.nytimes.com/2007/05/14/world/asia/14iht-14englede.5705671.html?pagewanted=1&r=1>
- Richard E. Nisbett. 2003. *The Geography of Thought. How Asians and Westerners Think Differently... and Why*. The Free Press. A Division of Simon & Schuster Inc., New York · London · Toronto · Sydney · Singapore.
- Allan Urho Paivio. 1971. *Imagery and verbal processes*. Holt, Rinehart, and Winston, New York.
- Jesse Prinz. 2011. Culture and Cognitive Science. In *The Stanford Encyclopedia of Philosophy* (winter 2011), Edward N. Zalta (Ed.). Retrieved March 16, 2014 from <http://plato.stanford.edu/archives/win2011/entries/culture-cogsci/>
- Toni Robertson (Ed.). 2006. *Proceedings of the 2006 Australasian Computer-Human Interaction Conference, OZCHI 2006, November 20–24, 2006*. ACM International Conference Proceeding Series, Vol. 206. ACM, New York, NY, USA.
- Yakov Shafranovich. 2005. RFC 4180: Common Format and MIME Type for Comma-Separated Values (CSV) Files (Oct. 2005). Retrieved March 15, 2014 from <http://tools.ietf.org/html/rfc4180>
- Vas[y]l Taras, Julie Rowney, and Piers Steel. 2009. Half a century of measuring culture: Review of approaches, challenges, and limitations based on the analysis of 121 instruments for quantifying culture. *Journal of International Management* 15, 4 (2009), 357–373. DOI:<http://dx.doi.org/10.1016/j.intman.2008.08.005>
- Ralph Thiim and Lise [B.] Hvatum. 2012. Organizing and Building Software. Patterns for effective management of large and complex code bases (Oct. 2012). Retrieved May 18, 2013 from <http://www.hillside.net/plop/2012/papers/Group%203%20-%20Coyote/Organizing%20and%20Building%20Software.pdf> Workshopped at the 19th Pattern Languages of Programs (PLoP) conference, Tucson, AZ, USA.
- Fons Trompenaars and Charles Hampden-Turner. 1997. *Riding the Waves of Culture. Understanding Cultural Diversity in Business* (second ed.). Nicholas Brealey, London.
- Chia-Jung Tsay. 2013. Sight over sound in the judgment of music performance. *Proceedings of the National Academy of Sciences* (2013). DOI:<http://dx.doi.org/10.1073/pnas.1221454110>
- Joseph W. Yoder and Ralph Johnson. 2002. The Adaptive Object Model Architectural Style, see Bosch et al. [2002], 3–27. Retrieved January 12, 2014 from <http://www.adaptiveobjectmodel.com/WICSA3/ArchitectureOfAOMsWICSA3.pdf>

## Glossary

This section contains thumbnails and definitions of the most important patterns and terms used in this paper.

**Bridge.** The BRIDGE pattern separates an abstraction from its implementation. Behind the same abstraction different implementations of its functionality can coexist. Each of them can evolve independently.

[Gamma et al. 1996b]

*Component Baseline.* The COMPONENT BASELINE pattern proposes to decouple development teams by splitting a single, monolithic baseline into multiple baselines. The software artifacts of each single team are managed as a separate baseline and are allowed to evolve more independent from one another than when managing the whole application as a single baseline.[Thiim and Hvatum 2012]

*Cultural Facet.* A Cultural Facet denotes a certain representation or system of symbols that joins a group of people on the one hand and distinguishes this group from other groups on the other. Examples are theoretical linguistical aspects of natural language including reading direction, e.g. its syntax, system of units for quantities, timezone, currency, formatting of character stream representations of numbers and dates, calendar, numbering scheme for calendar weeks, first day of the week, sort order / collation, treatment of certain symbols as whitespace characters, paper sizes, keyboard layout, typography (e.g. punctuation) etc. The extent of a Cultural Facet is subject to definition in many cases. Be aware that the realm of a single Cultural Facet, e.g. a currency, might not be congruent with the realm of another one, e.g. a system of units for quantities.

Among the disciplines that investigate Cultural Facets are Semiotics, Theoretical Linguistics, Philosophy of Mind, Mathematics, Cognitive Science[Prinz 2011], Computer Science, and Sociology of Culture.

A similar, though somewhat finer-grained usage of the term “facet” was coined by the C++ Standard Library [American National Standards Institute 2003, pp 317–673].

*Cultural Value Factor.* A Cultural Value Factor denotes a certain value or belief that joins a group of people on the one hand and distinguishes this group from other groups on the other. There are several different sets of factors[Taras et al. 2009]: Edward Twitchell Hall, Jr., found the factors Monochronic vs. Polychronic[Hall 1959] and High vs. low context[Hall 1976]. Geert Hofstede is well-known for his empirical identification of both national and organizational factors, which he has called “cultural dimensions”; the national cultural dimensions are Power Distance, Individualism vs. Collectivism, Masculinity vs. Femininity, Uncertainty Avoidance, Long-Term Orientation (earlier) / Pragmatic vs. Normative (later), and Indulgence vs. Restraint, the last two stem from the World Values Survey, whereas the organizational cultural dimensions are Means-oriented vs. Goal-oriented, Internally driven vs. Externally driven, Easygoing work discipline vs. Strict work discipline, Local vs. Professional, Open system vs. Closed system, Employee-oriented vs. Work-oriented, Degree of acceptance of leadership style, and Degree of identification with your organisation.[Hofstede et al. 2010] The World Values Survey identified the following factors: Traditional values vs. Secular-rational values, and Survival values vs. Self-expression values. Alfonsus Trompenaars and Charles Hampden-Turner published the following factors: Universalism vs. Particularism, Individualism vs. Collectivism (communitarianism), Neutral vs. Emotional, Specific vs. Diffuse, Achievement vs. Ascription, Sequential vs. Synchronic and Internal vs. External control.[Trompenaars and Hampden-Turner 1997] The GLOBE study focused on these factors: Performance Orientation, Future Orientation, Gender Egalitarianism, Assertiveness, Individualism vs. Collectivism on both societal and organizational scales, Power Distance, Humane Orientation, and Uncertainty Avoidance.[House et al. 2004] Richard Donald Lewis arranged societal cultures on a triangle with the vertices Linear-Active, Multi-Active and Reactive.[Lewis 2006]

Cultural Value Factors correlate with ethnolinguistic aspects of natural language, style, point of view<sup>3</sup>, rites and rituals, learning, education and thought[Nisbett 2003], games[Khaled et al. 2006], political system, predominant religion, diversity, humor, collaboration, consumer motivation, computer programming paradigm [Baniassad and Fleissner 2006], adoption of Beyond Budgeting[Hammer 2010] etc. The extent of a Cultural Value Factor is subject to definition in many cases. By construction the realm of the value of a single Cultural Value Factor, e.g. equal degrees of Gender Egalitarianism, is not congruent with the realm of the value of another Cultural Value Factor, e.g. equal scores in Uncertainty Avoidance.

<sup>3</sup>Back in February 2013 Ron Goldman, Kevlin Henney, Ramkumar Iyer, Joshua Kerievsky, Christian Köppe, Mary Lynn Manns, Linda Rising, Douglas C. Schmidt, Joseph W. Yoder, and last but not least Joseph Bergin, the initiator, have discussed the cultural differences of form and style on the [patterns-discussion](#) mailing list.

Among the disciplines that deal with Cultural Value Factors are Cultural Anthropology, Social Psychology, Industrial and Organizational Psychology, Cultural Psychology, Social Philosophy, and Ethics.

The term “factor” originates from a statistical method called Factor Analysis at least some of the authors cited adhered to to mine or verify their insights.

*Culture.* A Culture consists of acquired features that join a group of people on the one hand and distinguish this group from other groups on the other. The detailed concepts of Culture differ. Throughout this paper the following definition applies: Culture consists of both values and beliefs—the Cultural Value Factors, that correlate to observed social behavior—and the ability of thinking in representations, e.g. symbols—the Cultural Facets; theater[Laurel 1993], art in general and natural language are both expressions of values and beliefs and examples for representations. Cultures are not carved in stone. They evolve over time, interact with each other and are being changed due to changes to their respective environments, e.g. the transition from an industrial to a knowledge society brings changes in values[Inglehart and Welzel 2010].

*Factory Method.* The FACTORY METHOD pattern delegates creation of objects to a method. This method encapsulates the knowledge of the exact class of the instance to create. One example is a method with a parameter that allows the caller to tell the method which class to instantiate.[Gamma et al. 1996d]

*Internationalization.* Internationalization refers to the cultural generalization of an application to prepare it for the forthcoming process of Localizations to the representational aspects of several national Cultures. Internationalization is an all-encompassing endeavor that has to consider all relevant Cultural Facets for the respective application. Internationalization is often abbreviated as I18N. A generalization of an application regarding Cultural Value Factors is much harder than regarding Cultural Facets or even impossible; therefore often this is not being subsumed under the term Internationalization.

*Layers.* In applications which can be structured into groups of subtasks where each group comprises its own level of abstraction, these levels can be reified into so-called LAYERS. In systems with such an architecture each layer only calls functionality provided by the next lower layer or by itself, but neither of any lower layer but the adjacent one nor of any upper layer. LAYERS contribute to loose coupling.[Buschmann et al. 2000a]

*Locale.* A Locale denotes a set of several Cultural Facets. A Locale is specific to a certain region, e.g. a country, and refers to all information to localize text, format numbers, dates, currencies etc. To identify a Locale, it often has a key unique among all Locales available, e.g. the tuple of a language code according to ISO 639-1 and an ISO 3166-1 alpha-2 country code, e.g. en\_GB for British English.

*Localization.* Localization refers to the specialization of an application to a specific national Culture considering all relevant Cultural Facets. The result is a new or modified Locale. To prepare an application for the task of Localization it must be internationalized first once. Localization is often abbreviated as L10N. Ideally, Localization also considers all applicable Cultural Value Factors.

*Portability.* Portability has two aspects: Portability with regard to environments or platforms means that an application requires no or only a few local changes to run on another platform than once planned for. The term platform can refer to operating system, hardware architecture or even a set of third party software the application interfaces with, e.g. a database management system. Portability in time means that an application can still be compiled after years have passed.

*Prototype.* The PROTOTYPE pattern discusses creation of instances by cloning existing instances instead of creating new instances and modifying them.[Gamma et al. 1996e] Interfaces and abstract classes that support cloning instances are also examples for ABSTRACT FACTORIES[Gamma et al. 1996a].

#### ACKNOWLEDGMENTS

Without the invaluable feedback of Jeffrey L. Overbey, who was the PLoP shepherd of this work, this paper would not have been the way it is now.

The author would like to thank all the participants of PLoP 2013 for their contributions. Special thanks go to the members of the Writers' Workshop [Gabriel 2002] "000101" the author participated in: Lori Flynn, Alfredo Goldman vel Lejbman, Lise B. Hvatum, David A. Mundie, Alexander Nowak, William F. Opdyke and last but not least Paul M. Chalekian and Andreas Fießler, its moderators. Their feedback had a significant impact on the current version of the paper.

Last but not least my thanks and love go to Cornelia Kneser, my wife, for her constant support throughout the writing of the paper.

Received May 2013; revised October 2013; accepted May 2014