Lingua Franca

Brian Foote and Don Roberts

Department of Computer Science
University of Illinois at Urbana-Champaign
1304 W. Springfield
Urbana, IL 61801 USA

foote@cs.uiuc.edu (217) 328-3523
droberts@cs.uiuc.edu (217) 244-0431

Wednesday, 29 July 1998

Fifth Conference on Patterns Languages of Programs (PLoP '98)
Monticello, Illinois, 11-14 August 1998

Contents

Abstract

This paper examines a set of three patterns that examine the issues that arise when data are to be shared among several applications in one or more different formats. When an ENGLISH ONLY/STANDARD REPRESENTATION can be used, conversion and translation are moot issues. However, this simplicity is not without its cost. When freely CONVERTIBLE CURRENCIES are available, data are not isolated and trapped by incompatible representations. However, as the number of formats increases, conversion becomes a complicated problem. A LINGUA FRANCA addresses the problem posed when representations proliferate. If a standard secondary tongue is adopted, a much smaller number of translators or converters is required to ensure universal translation.

Introduction

It has long been recognized that a variety of problems can be greatly simplified if everyone would just speak the same language. It has been recognized for nearly as long that it is impractical to expect this to always be so.

This paper examines a set of three patterns that addresses the problems one encounters when multiple representations and languages emerge in some domain. These patterns are presented using analogies drawn from the realms of natural language and currency conversion.

Where an ENGLISH ONLY/STANDARD REPRESENTATION can be established, communication is simplified. However, this generality is not without its cost. When freely CONVERTIBLE CURRENCIES are available, data are not isolated and trapped by incompatible representations. A LINGUA FRANCA addresses the problem posed when representations proliferate. If a standard secondary tongue is adopted, a smaller number of translators or converters is required to ensure universal translation.

ENGLISH ONLY

alias
STANDARD REPRESENTATION
COIN OF THE REALM
EURO
MONOPOLY
VHS

*In 1999, eleven European countries will begin replacing their currencies with a single, common European currency, the **euro**.*

❖ ❖ ❖

The benefits of standardization are impossible to deny.

There was a time when differing railroad gauges forced passengers and cargo crossing Europe to change trains at the borders. This state of affairs was an expense and inconvenience to everyone (perhaps except the customs agents).

No one would think of building an automobile with the gas pedal on the left and the brake on the right. Automakers all over the world universally place these pedals in this standard configuration.

You want to represent information in such a way that a number of different programs can use it.

Sometimes, you can greatly simplify things by mandating a single format. Sometimes, a defacto standard format will emerge anyway.

There is no question that if you can avoid using multiple representations for the same subject matter, you can avoid a lot of problems.

One force that comes into play here is that when multiple representations exist, and free conversion between them is desired, designers can be reduced to supporting the mere intersection of the capabilities of all the formats. Any enhancement that is added by a single representation is lost in the translations. However, when the need for such compatibility is removed, this restraint on the power of competing representations dissappears too. This is a mixed blessing. compatibility considerations, be they with competing products, or obsolete versions, can act as brakes on excess, as well as innovation.

Another force that encourages the emergence of standard representations is that they avoid duplication and redundancy, and encourage reuse. Once a standard representation is established, APIs, protocols, frameworks, and tools can emerge to help support it. The efficiencies brought about by such specialization can allow scarce resources to be focused on more worthy tasks than the generation of me-too representations.

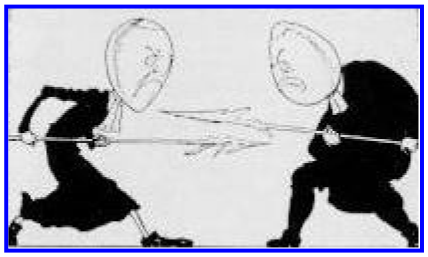
***Therefore*, Select one format to serve as your standard language, or vernacular. If a suitable candidate exists, embrace it. If none exists, seek to establish one yourself.**

The world of CPU architecture is a good place to find examples of beneficial convergence towards a handful of standard approaches. For instance, ten or twenty years ago there were a variety of incompatible, ad hoc schemes for representing floating point values. With the advent of the IEEE floating point standard [Severance 1998], these have all but vanished, and byte order is the only issue left to contend with when portable binary floating point values are to be dealt with. This makes the design of portable binary images and virtual machines easier.

Another place to look is in the realm of networking. Over the last several years, the TCP/IP Internet protocol suite has emerged overwhelmingly as the defacto standard. During the early days of the great internet boom, there was speculation as to what might replace TCP/IP. The verdict is in, and the winner is: TCP/IP. The importance of such incumbency is discussed below.

Another example is the convergence of word sizes towards standard powers of two. During the ’70s and ’80s, machines with 24, 36, and 60 bit words were still common. Over the last several years, these word sizes have been supplanted, and fully byte-addressable memories have become the norm. The effective elimination of odd word size considerations has allowed for a considerable simplication of subsequent portable binary formats. When you can assume such exceptional cases are gone, your job becomes easier.

Byte order itself has evolved to use either pure big-endian or pure little-endian [Cohen 1981] representations. The troublesome hybrid orders seen in older DEC and Z8000 processors, for example, have effectively vanished. This, in turn, has made dealing with networked binary easier. (Indeed, network byte order itself is an example of the LINGUA FRANCA pattern at work.)



Of course, standards have long been recognized as indispensable to programming language designers and implementers.

However, designing a representation of your own is not without its difficulties. One is that you must accommodate change. As with object-oriented frameworks, your external data or object formats must be able to stand the test of time. When a defacto standard exists, it is often because it has already met this test. A good designer knows not to discount the value of the experience that underlies such staying power.

Especially when you have total control over your representations, you must still confront change. Issues such as forward/backward and upward/downward compatibility arise. For this reason, you should design formats with room for future growth, by designing structural representations that can accommodate future extensions, and writing recognizers that can gracefully ignore material for which they were not designed. Indeed, this strategy has proven effect in the design of HTML.

The emergence of standard tongues puts native speakers at an advantage, and non-native speakers at a disadvantage. In the realm of technology, similar technical and economic advantages accrue to those best versed in a dominant representation. This places an organization that can mint the such standards with impunity in the catbird seat, and leaves others in a position where they must depend on that organization.

Certainly one way to avoid the complexity associated with supporting and converting among multiple formats is to vanquish any competing formats, and claim the entire field for yourself. The market gains the simplicity and stability that comes from everyone speaking the same language. You, of course, gain the entire market. Indeed, there are those who claim that software is a natural monopoly.

Therein lies a tale. If the reader will indulge us for a few paragraphs, we will look more closely into one such school of thought before returning to more technical turf, and examining the question of whether the marketplace of technical decisions is governed by similar mechanisms.

W. Brian Arthur is a mathematician, engineer, economist, and complexity theorist who might be called the "man who brought chaos to the marketplace". Arthur is best know for his work on role of positive feedback, or *increasing returns* [Arthur 1992] on economic competition.

Traditional markets, it is said, operate according to the law of *diminishing returns*. For instance, a mine operator cannot increase production indefinitely without exhausting the richest veins of ore and turning to increasingly marginal ones, which may prove more dangerous, and (more to the point, if your are an economist) *expensive* to mine than the original lode. Competitor who have not reached such a point of diminishing returns will be able to sell their ore more cheaply. According to classical economic theory, the average price of ore in such a market will reach a point of equilibrium. Cheaper producers will enjoy higher profits, and marginal producers will eventually limit production, or abandon it altogether.

Arthur argues that classical diminishing returns mechanisms hold sway in traditional, resource limited segments of the economy, such as agriculture, mining, and manufacturing. Arthur contends that some markets, particularly those in high-technology industries such as computers, software, pharmaceuticals, and aircraft a governed instead by a principle of increasing returns.

In a world of increasing returns, initial advantages, however slight or fortuitous, are amplified, and lead to further competitive advantages, and ever increasing returns. Put simply, the rich get richer; them that has get; the winner takes all. Such a market will anoint a "king of the hill", who, like regents everywhere, is next to impossible to depose. Arthur calls this phenomenon lock-in.

Why should high-technology marketplaces behave so differently? Arthur identifies an number of peculiarities that distinguish them from more mundane markets. Perhaps the most striking is *path dependence* or *history* effects. Here, the idea is that seemingly trivial events come in to play to confer an initial advantage on one competitor or other. Such an advantage, however it is come by, then snowballs, until our charmed competitor gains an insurmountable advantage. Arthur advanced the heretical (to classical economists) notion that were history to be replayed, chance events below the noise level might anoint a different winner, and that, in an increasing returns world, a victory may not always go to the best, but the first, or the luckiest.

Here it would seem, is another perspective on the mystery of why, all too often, Worse is Better [Gabriel 1991].

Arthur sometimes refers to certain path dependencies as *founder effects* [Arthur 1996], or events that occur early in the development of a technology that, had they been otherwise, might have led to some other technology's ascendancy instead. For instance, during the early '80s, the Intel 8088, a lower priced version of the 8086 with an 8-bit I/O bus, appeared only weeks before the comparable Motorola 68000 part. IBM considered both for its new PC product, and, ever conservative, opted for the Intel chip. A different choice might have changed history. The 68000's architecture was much more forgiving about the sorts of 64k boundary limitations that hobbled DOS and Windows for nearly a decade. It was a near thing.

In all of this, there is a strong flavor of the sort of "extreme sensitivity to initial conditions" that one sees in chaotic systems. Path dependency and founder effects are variations on the famous idea that a butterfly flapping its wings in China might change the weather in Monticello, Illinois a month later. But for a few trivial contingencies early on, the outcome could easily have been different, and almost certainly would be were you to "replay the tape". While it might be possible to broadly characterize the range of possible outcomes in such systems, specific predictions about particular outcomes cannot be made. Rather than a finding equilibrium at some single optimal, global maximum, such systems may settle into one of several, possibly suboptimal, local maxima. Each might be arrived at via one or more fickle, fortuitous, unlikely paths.

Another factor is what Arthur calls *groove-in*, and others call *learning effects*. High technology products require a significant investment in training. An MRI machine might require weeks of product specific operator instruction. Clerical staff might take months or years to master a word processing program. Once an investment in such training is made, users are reluctant to endure such ordeals again, and managers are equally disinclined to pay for it. Hence, the original vendor's lock is maintained. Palm Pilot users might invest a significant amount of time mastering Grafitti, and may be reluctant to learn a different set of gestures, even were a better palmtop to come along.

Network effects are a related notion. The greater a product's penetration in its market is, the more valuable it can become to its users. For instance, a telephone is of little use if you are the only person who has one. However, once everyone has one, it is a useful thing indeed. In a market where incompatible products are competing, the playing field can tilt ever more quickly in the direction of the product that is winning. Readers old enough to recall the VCR wars of the Reagan era will recall that once VHS picked up steam (perhaps because user's opted for longer playing times over Beta's picture quality), its hold on the market became ever stronger.

Since gaining an early advantage is so important in an increasing returns world, heavy early discounting to gain market share is the norm. The most extreme form of such discounting is to give a product away for nothing. Netscape exploited this gambit to gain an early, dominant position in the internet browser marketplace, and Microsoft matched it to counter them.

Eric Raymond, in The Cathedral and the Bazaar [Raymond 1997] vividly recounts how his free software application, *fetchmail*, gained mindshare and critical mass, and came to dominate its niche. This paper contains a wealth of interesting observations on collective development and debugging. To tie his tale to this one, the dissemination of free code to implement a file format is an effective means of promoting it, and of increasing its chances of gaining defacto standard-hood. His observations about the importance of early releases, building a user base, and attracting coders mesh well with increasing returns.

High technology products typically have enormous *up-front development costs*. The first dose of the latest hair-loss paliative may, in effect, cost the pharmaceutical maker that develops it hundreds of millions of dollars, while subsequent doses may be manufactured at the cost of a few dollars worth of chemicals. These costs have the effect of discouraging new players once a winner seems to have been determined, and increase their incentives to abandon the field to the dominant player.

Psychological factors [Arthur 1995] play a much more important role in an increasing returns marketplace than they do in a traditional one. For instance, a dominant player might attempt to preempt competitors by announcing a product months or years before it is available. So effective are discounting and vaporware ploys that IBM was explicitly prohibited from ever discussing unrealeased products, or giving products away, in the consent decrees it entered into to avoid anti-trust penalties.

Arthur argues that the increasing returns world is a sort of casino [Arthur 1996], in which each niche presents a unitary winner-take-all opportunity. One player wins a dominant position in, or outright monopoly over its market, and the others go home empty handed.

Arthur does not argue that these increasing returns monopolies are a bad thing. Indeed, they can encourage the emergence of standards, eliminate wasteful duplication, ensure that a network of trained users exists, and exploit economies of scale. Other potential players are freed to seek locks on niches of their own. Problems arise only when such players attempt to leverage their locked-in positions to gain an unfair head-start over competitors in a new game [Gates 1998].

In unexplored terrain, the pioneer has no choice but to blaze trails. However, once trails have been established, there is much to be gained by not second guessing their placement, despite the possibility that they follow suboptimal routes, or won’t get you exactly where you are trying to go.

Any buy vs. build decision must be made carefully. When a defacto standard representation exists for a particular domain, network effects, groove-in, cost, time, and convenience will all suggest embracing it.

Care must be taken to not permit such a standard to become a Procrustean bed. Where your needs diverge from those addressed by a standard, you may be better served by seizing control of your destiny and designing your own representation. When you control it in-house, you make the rules, and can rule by decree.

Of course, by going your own way, you risk exacerbating the curse of Babel, by adding yet another tongue to the mix, to say nothing of defeat at the hand of the dominant player. Does it all really come down to eat or be eaten? Fortunately, there is another route to consider. It is discussed in the LINGUA FRANCA pattern.

❖ ❖ ❖

The WINNING TEAM pattern addresses some of these issues from the perspective of evolving code itself. The FIRST ONE’S FREE pattern examines the role of discounting and vanityware as promotional strategies. Indeed, the notion of *increasing returns* meshes with the observations made there quite nicely.

With any monopoly, there are benefits and dangers. For instance, because competition is stifled, the need to innovate is diminished, and incremental adaptation can be deferred. The SOFTWARE TECTONICS pattern discusses the possible consequences of letting such strain accumulate.

CONVERTIBLE CURRENCIES

alias

TRANSLATION

BILINGUAL TRANSLATOR

DYNAMIC CHANGE OF REPRESENTATION

❖ ❖ ❖

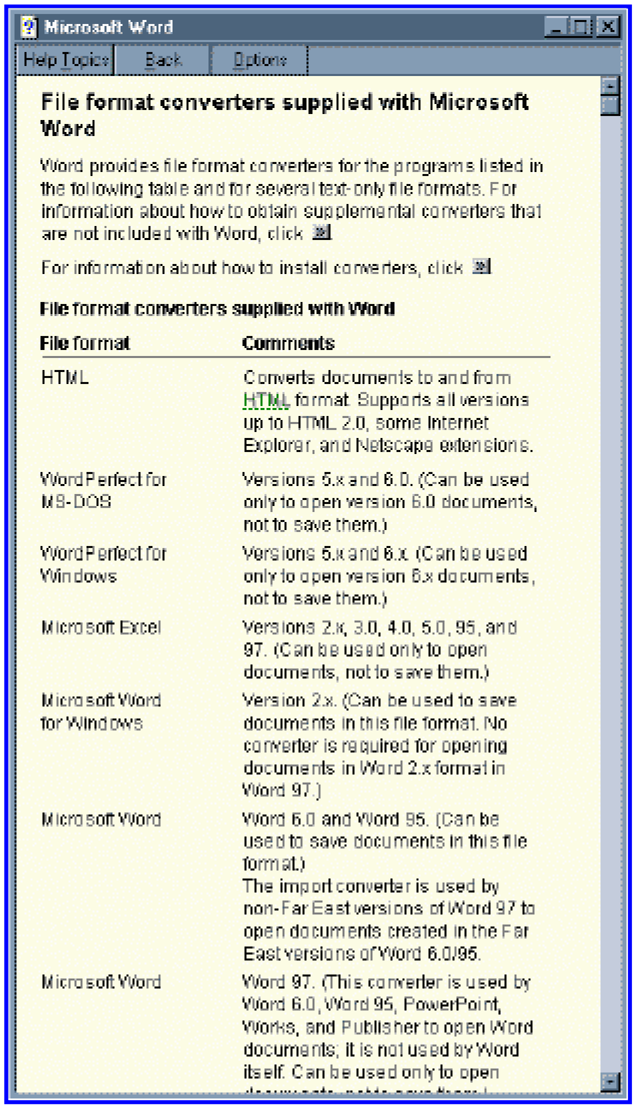
Sometimes, no single representation for data does everything we need. Sometimes we must match the representation to the task at hand. This matching process might be relatively coarse-grained or static, or fine-grained, and highly dynamic, with transformations taking place at runtime.

You want to represent information in such a way that a number of different programs can use it.

Representations are freely convertible when one can move back and forth between them with relative impunity. They need not be totally invertible functions. It is only necessary that the losses incurred be of minor practical significance.

Just as with currency conversions and translations among natural languages, there are costs associated with changing representations. There are a variety of forces and issues with which designers may need to concern themselves when designing a system that employs multiple, freely convertible representations.

Time: Conversions take time. Indeed, they can be extremely time consuming in some cases. In some cases, this overhead can be avoided by caching all or part of the the converted objects so that they need not be converted again until one of the copies changes. This strategy requires that a mechanism to make sure these multiple versions are consistent be put in place, and that these copies be invalidated or updated when changes occur.



File format converters supplied with Microsoft Word

Word provides file format converters for the programs listed in the following table and for several text-only file formats. For information about how to obtain supplemental converters that are not included with Word, click

For information about how to install converters, click

File format converters supplied with Word

File format	Comments
HTML	Converts documents to and from HTML format. Supports all versions up to HTML 2.0, some Internet Explorer, and Netscape extensions.
WordPerfect for MS-DOS	Versions 5.x and 6.0. (Can be used only to open version 6.0 documents, not to save them.)
WordPerfect for Windows	Versions 5.x and 6.x. (Can be used only to open version 6.x documents, not to save them.)
Microsoft Excel	Versions 2.x, 3.0, 4.0, 5.0, 95, and 97. (Can be used only to open documents, not to save them.)
Microsoft Word for Windows	Version 2.x. (Can be used to save documents in this file format. No converter is required for opening documents in Word 2.x format in Word 97.)
Microsoft Word	Word 6.0 and Word 95. (Can be used to save documents in this file format.) The import converter is used by non-Far East versions of Word 97 to open documents created in the Far East versions of Word 6.0/95.
Microsoft Word	Word 97. (This converter is used by Word 6.0, Word 95, PowerPoint, Works, and Publisher to open Word documents; it is not used by Word itself. Can be used only to open

Consistency: When multiple versions of the same data exist, questions of consistency naturally arise. The designer might require that a change made to any copy be propagated to all versions, or might designate a particular version as the authoritative copy. Indeed, many of the issues that arise in distributed systems and databases can arise when this pattern is employed.

Space: One representational scheme may take up more space than another. Indeed, trading off space and time is a recurring theme when such representations and conversions are devised. For example, some formats for representing images take more space than others. Issues such as pixel depth and compression strategies can dramatically affect the space requirements associated with images.

Overhead: Some representations are more costly to use than others. For instance rendering compressed images can incur more runtime computational overhead than uncompressed ones might.

Complexity: Some representations are more complicated than others. While complexity often comes hand-in-hand with power, it can mean that programmers might find these representations harder to deal with. When this complexity is exposed to users, it can cause confusion as well.

Representational scope can pose a classical "diminishing returns" dilemma for the designer. As you attempt to straddle more and more of the design space, you can move further afield from your areas of expertise, and embark on costly forays into unexplored territory. The obvious, easy gains may be exhausted, and additional generality might have to be bought at a cost of programmer time, reliability, training complexity, and efficiency.

Predictability: Not only do representations differ in the time that it takes to convert among them, and the overhead associated in using them, but they can differ in the degree to which this overhead is predictable. Here again, image compression schemes illustrate this issue. An all white bitmap might be represented quite concisely, while random noise might be essentially uncompressible. A consequence of this is that the overhead associated with rendering an image might vary tremendously in ways that are highly dependent on the data.

Utility: When a particular representation supports tools or operations that are unavailable in other formats, it may be easier to convert to this representation, use the tool, and convert back, than it is to attempt the manipulation in some other representation. For instance, some word processors might provide easy access to spelling checkers or grammar tools, while others might generate output suitable to certain printers or distribution formats.

This is the difference between finding the *right tool for the right job* and employing a *Swiss Army Knife*.

Fidelity: Fidelity is always an issue when data are converted. A major consideration in the design and use of conversions is data loss. When the target format is less rich than the source, detail must be sacrificed. When it is richer, decisions must be made as to reasonable default values and assumptions during conversion. When you convert from ASCII to Word 97 format, decisions about default fonts and sizes must be made. Conversely, when you convert to pure text formats, decisions about line breaks must be made, and font choices are lost. Of course nearly all formatting information to be lost.

When you convert from JPG to GIF89, you need to decide how to handle transparent background colors. Converting from GIF To JPG can result in a loss of image resolution, because of JPG's compression. So can a change from a 24-bit to an 8-bit image depth. Sometimes, the price of such conversions is obvious and immediate. Other times it is more subtle, as in multiple natural language translations, image processing, and printing.

Versions are just alternative representations. You can convert among these too. This can be thought of as a sort of floating exchange rate, or inflation adjustment.

Physical Unit conversions are conversions too. Ward Cunningham refers to this as the WHOLE VALUE pattern. Conversions work when specific objects are fungible commodities. When object identity is significant, conversions are harder to use.

***Therefore*, Provide for free conversion among various representations. Hence, one can use the representation best suited to the task at hand.**

Word processors usually provide mechanisms for importing and exporting a variety of formats besides their native formats. These provide compatibility with previous versions of the same product, products from other vendors, and standard formats such as ASCII text or Postscript. Some conversions may be read and not written, and vice versa. Postscript might be generated as output, but not read as input. Text with a variety of line separators might be read, but a only a single text format might be written. For instance, Microsoft Word can read data from Excel spreadsheets, but cannot save a document as a spreadsheet.

Another risk when multiple representations are involved is the emergence of TRADING BLOCS. These are collections of objects, programs, and products which all support their own, largely incompatible representations. Once, a Macintosh data file on a Windows platform found itself in the same predicament as an American in Paris.

This oft recounted tale from the journals of the Lewis and Clark expedition illustrates the joys of multiple translations:

Ordway filled in more details of the visit: "our officers took down Some of their language found it verry troublesome Speaking to them as all they Say to them has to go through Six languages, and hard to make them understand." The captains' questions were translated into French by Drouillard or Labiche, directed to Charbonneau, who conveyed them to Sacagawea in Hidatsa, who rephrased them in her native language to a young Shoshone who was with the tribe and could speak Salishan - "a gugling kind of languaje," noticed Clark, "Spoken much thro the Throught." Answers followed the same route in reverse.

To further illustrate this phenomenon, the following prose shows how Alta Vista's Babelfish translates an old version of this paper's abstract into French, and then retranslates the result into English.

This paper examines a set of three patterns that addresses the problems one encounters when multiple representations and languages emerge in some domain. When freely CONVERTIBLE CURRENCIES are available, data are not isolated and trapped by incompatible representations. Where a ENGLISH ONLY/STANDARD REPRESENTATION can be established, communication is simplified. However, this generality is not without its cost. A LINGUA FRANCA addresses the problem posed when representations proliferate. If a standard secondary tongue is adopted, a much smaller number of translators or converters is required to ensure universal translation.

Cet article examine un ensemble de trois configurations qui adresse les problèmes un rencontre quand les representations multiples et les langages émergent dans un certain domaine. Quand librement les DEVISES de CONVERTIBLE sont disponibles, des données ne sont pas isolées et ne sont pas emprisonnées par les représentations incompatible. Là où une REPRÉSENTATION ANGLAISE d'cOnly/standard peut être établie, la transmission est simplifiée. Cependant, cette généralité n'est pas sans son coût. Une LINGUA-FRANCA adresse le problème a posé quand les représentations prolifèrent. Si une langue secondaire standard est adoptée, un plus petit nombre de traducteurs ou de convertisseurs est exigé pour assurer la traduction universelle.

This article examines a whole of three configurations which addresses the problems a meeting when the multiple representations and the languages emergent in a certain field. When freely the CURRENCIES of CONVERTIBLE are available, from the data are not isolated and are not imprisoned by the representations incompatible. Where an ENGLISH REPRESENTATION of ONLY/STANDARD can be established, the transmission is simplified. However, this general information is not without its cost. A LINGUA-FRANCA addresses the problem posed when the representations proliferate. If a standard secondary language is adopted, a smaller number of translators or converters is required to ensure the universal translation.

Currency Converter pages abound on the web. For example:

Convert amount 1 with rate of 07/25/98 mm/dd/yy

US Dollar . USD
German Mark . DEM
Thai Baht . THB
Afghanistan Afghani . AFA

TO

US Dollar . USD
German Mark . DEM
Malaysian Ringgit . MYR
Afghanistan Afghani . AFA

Convert Now!

164 Currency Converter ©1997-1998 by OANDA

❖ ❖ ❖

Contrast ENGLISH ONLY/STANDARD REPRESENTATION and LINGUA FRANCA.

Contrast ADAPTER too.

The OBSERVER pattern can be used to implement mechanisms to enforce mutual consistency among multiple views of a set of objects.

Free convertability among different *subjects*, *guises*, *facets*, *roles*, *aspects*, *extensions*, "*unknowns*", *perspectives*, or *views* can counteract the tendency to build monolithic, one-size-fits-all objects, and encourage the emergence of better factored architectures that exploit finer-grained, convertible parts using representations that fit the task at-hand.

LINGUA FRANCA

alias
ENGLISH
U.S. DOLLAR

❖ ❖ ❖

You want to represent information in such a way that a number of different programs can use it.

Otherwise compatible data are stored in a variety of different formats. Converting among them can require, in the worst case, a conversion program for each pair of formats. One problem that is often seen when representations proliferate is that the number of converters needed to convert among them can increase with the square of the number of representations.

***Therefore*, Allow one format to serve as a universal second language, a lingua franca. Then conversion between arbitrary formats will require only two conversions, one from the first format to the lingua franca, the second from the lingua franca to the second format.**

Just as with currency conversions, there are costs associated with changing representations. Conversions take time. In this approach, two conversion are usually needed, one to convert to the standard representation and one to convert from the standard representation. If these are extremely time consuming, this can be a serious drawback. Another force that impacts this solution is lossiness. If the translations to and from the standard representation lose information (e.g., GIF->JPEG conversion), then this solution may not be acceptable. For instance, consider the case where the standard representation is JPEG (a poor choice). If we want to convert from GIF to TIFF, we must convert from GIF to JPEG and then JPEG to TIFF. However, each of these conversions degrades the image slightly. If this series of conversions occurs many times, the image will get increasingly corrupted.

Examples abound.

Word processors usual provide mechanisms for importing and exporting a variety of formats besides their native formats.

Jeff Poskanzer's Portable Bitmap Tools which come with the X Window System, are one example. PBM format is widely used as a *lingua franca*.

Byte Code or other machine independent code representations are examples too, in the sense that multiple languages can be translated to these representations, and a single native code generator that translates from this representation to native code can be written for each native instruction set or operating system.

Byte code representations serve as a lingua franca that stands between compilers, and machines. Multiple languages can compiled to a single intermediate representation. Each platform can either interpret this representation or translate it to native code. They have a long history. Kay recalls that he first saw them in Euler. Smalltalk has used bytecode to represent executable methods since the mid '70s. UCSD Pascal also used machine independent byte code. The phenomenal popularity of the Java programming language has led to a resurgence of interest in byte code. Indeed, it is because byte code serves as a lingua franca that stands between the compiler, and the sundry platforms on which Java is to run, that its write-once/run anywhere promise can be even contemplated. When virtual machine instruction sets are well-designed, they can support multiple languages as well as multiple platforms.

Object brokers, such as brokers complying with the CORBA standards, play a similar role. COM and DCOM play a similar role. Multiple languages use IDL or programmatic interface to access language independent brokers. These, in turn supply objects from a variety of sources. The object model itself serves as sort of lingua franca.

POSIX serves as a lingua franca for application programmers. It allows them write portable programs for a variety of platforms using Unix-like APIs.

SQL has become the lingua franca of database programming.

The table below illustrates the problem. To convert among all the currencies in question, a conversion between each is needed. For the 10 currencies shown, the number of conversions needed is N^2-N (discounting the identity conversions).

	USD	DEM	CNY	BEF	GBP	IEP	MYR	OMR	RUB	CHF
US Dollar . USD	1	1.77	8.28	36.60	0.61	0.7	3.86	0.385	6.14	1.48
German Mark . DEM	0.56	1	4.67	20.64	0.35	0.4	2.17	0.217	3.46	0.84
Chinese Yuan Renminbi . CNY	0.12	0.21	1	4.422	0.07	0.1	0.47	0.047	0.74	0.18
Belgian Franc . BEF	0.03	0.05	0.23	1	0.02	0	0.11	0.011	0.17	0.04
British Pound . GBP	1.63	2.9	13.5	59.77	1	1.2	6.3	0.629	10	2.42
Irish Punt . IEP	1.42	2.52	11.7	51.92	0.87	1	5.47	0.546	8.71	2.1
Malaysian Ringgit . MYR	0.26	0.46	2.15	9.495	0.16	0.2	1	0.1	1.59	0.38
Omani Rial . OMR	2.6	4.61	21.5	95.07	1.59	1.8	10	1	15.9	3.85
Russian Rouble . RUB	0.16	0.29	1.35	5.963	0.1	0.1	0.63	0.063	1	0.24
Swiss Franc . CHF	0.68	1.2	5.59	24.71	0.41	0.5	2.6	0.26	4.14	1

	USD	DEM	CNY	BEF	GBP	IEP	MYR	OMR	RUB	CHF
US Dollar . USD	1	1.77	8.28	36.60	0.61	0.7	3.86	0.385	6.14	1.48
German Mark . DEM	0.56	---	---	---	---	---	---	---	---	---
Chinese Yuan Renminbi . CNY	0.12	---	---	---	---	---	---	---	---	---
Belgian Franc . BEF	0.03	---	---	---	---	---	---	---	---	---
British Pound . GBP	1.63	---	---	---	---	---	---	---	---	---
Irish Punt . IEP	1.42	---	---	---	---	---	---	---	---	---
Malaysian Ringgit . MYR	0.26	---	---	---	---	---	---	---	---	---
Omani Rial . OMR	2.6	---	---	---	---	---	---	---	---	---
Russian Rouble . RUB	0.16	---	---	---	---	---	---	---	---	---
Swiss Franc . CHF	0.68	---	---	---	---	---	---	---	---	---

The table above illustrates how this problem is simplified when one representation (in this case, the dollar (USD)), becomes the lingua franca. Here, the number of conversion needed becomes 2*N-1.

Of course, each time a full conversion is needed, two conversions needed to be done. One converts from the first currency into dollars, and the second converts from dollars to the second currency. The time and complexity involved in performing two conversions are traded-off against the simplicity of providing a smaller number of converters.

On one level, the example above is artificially simple. Maintaining a fully stocked table of numeric conversions among all the currencies listed is an easy programming task. However, in general, format conversions will be more complex. They may require intricate calculations rather than simple numeric conversions. For that matter, anyone who thinks real world currency conversion is a simple problem hasn’t traveled very much.

In reality, the tables are not usually as tidy as the one shown above. Often, an handful of representations, rather than one single representation, may serve as lingua francas. Indeed, in the world of currencies, several currencies may play the role of RESERVE CURRENCIES. For instance, the Japanese Yen, the German Mark, the British Pound, the French and Swiss Francs, the Rouble, and the CFA Franc in Africa all serve this role in different parts of the world. Graphics files may be exchanged in JPG, BMP, GIF, or PPM. Documents may travel as Word Documents, Postscript, PDF, Rich Text Format, or ASCII. Programmers may elect to hedge their bets, and support several such "reserve" representations. Even so, the overall conversion burden remains more linear rather than quadratic, and the fundamental value of using these freely convertible fallback representations remains.

❖ ❖ ❖

The term lingua franca comes from an Italian phrase for "Frankish language". The term harkens back to the traditional role of French as the "language of diplomacy". The underlying idea was that no matter what languages two diplomats might speak at home, they could always communicate if both had a command of French. Indeed, at one time it was not unusual for aristocrats and royalty in the courts of eastern Europe to speak French in lieu of the native tongues of their subjects. The term is something of an anachronism. At one time Latin and Greek played this role among scholars. These days, English has assumed the role of the *lingua franca* in many parts of the world, and is the language of choice for discourse among scientists and aviators. French is still widely spoken in parts of Africa, Canada, the South Pacific and the Caribbean, and, of course, in France, where it is used exclusively. Its role as a *lingua franca*, or second language of choice, is seen most often in francophone Africa, where it is the official national language of several former French colonies, while being the first language of only a handful of inhabitants.

Contrast CONVERTIBLE CURRENCIES and ENGLISH ONLY/STANDARD REPRESENTATION.

The notion of falling back on a lingua franca is a common idea, and anyone who works with software will come across it sooner or later. Indeed, it is often taken for granted.

Discussion

Interestingly, many of the forces that drive the emergence of standard and non-standard representations resemble those that drive biological speciation. Analogs to geographic separation, convergent evolution, niches, divergent needs, and specialization can all be seen. For instance, for many years, data formats in the Macintosh world evolved independently, like marsupials and monotremes, oblivious to the placental explosion in the PC world. Indeed, once the PC world began to encroach, the effect on such native fauna was not unlike that of feral dogs and cats on Australia's wildlife in the 19th century. On the other hand, we dare say that few will mourn the loss of one's compliment hardware arithmetic. In general, it is not always obvious how we should distinguish priceless, irreplaceable info-diversity from pointless blind alleys for which the best requiem is a chorus of "good riddance".

The forces that drive representations towards oligarchy and monopoly, and the forces that drive them to Balkanization and Babel are in constant tension. A common tongue promotes simplicity, harmony and community. Yet, an obsolete or dead language is of no use to anyone (save an occasional scholar). A standard that is inadequate to address the needs of its users, and promotes incompatible extensions (e.g. Pascal's lack of string support), can lead to a din of diverging dialects, which must ultimately be reconciled. When can one size fit all? It is essential that lines of communication be kept open, translations maintained, and that the need for a lingua franca be kept in mind as simple domains grow more rich and complex.

While in an ideal world, a minimal number of representations for data might be needed, in ours, this is seldom the case. When a wide range of representations are present, one (or more) will frequently emerge as a lingua franca. Such a common format permits the benefits of focused, domain-specific representations to be retained, while providing a path from such representations to a wealth of others.

In those domains where a single, standard representation has not, cannot, or should not prevail, the utility of having one or more representations serve as lingua francas is sufficiently compeling so as to virtually ensure their emergence.

Standards are a good thing. When a single tongue can be spoken by all, enormous simplifications are possible. Yet, one should tailor ones objects and representations to one's problem. When it is easy to convert among these, one can let one's data metamorphosize into whatever guise a problem demands, safe in the certainty that you can convert back once your are finished. To conquer the curse of Babel, we can emulate the diplomats of old, and search for and use a lingua franca.

Acknowledgments

We are indebted, as usual, to the members of the UIUC Software Architecture Group's Patterns Workshop, Ian Chai, Ralph Johnson, Dragos Manolescu, Weerasak Witthawaskul, and Brian Marick, who worked over an early draft with their customary thoroughness, and suggested a variety of valuable improvements.

Molly Lojo proofread an early version of this paper and spotted some embarrassing gaffes.

We are grateful to OANDA for making it easy to embed their currency converters in third-party web pages.

Charles Herring directed us to W. Brian Arthur's work on increasing returns, and suggested its applicability to the "marketplace" of software architecture.

We'd also like to acknowledge the sacrifices made by international currency speculators Ian Chai and Weerasak Witthawaskul who heroically maintained long positions in the Malaysian Ringgit and Thai Bhat through the 1997-1998 currency crisis, and were thus able to supply the banknotes depicted in this paper.

Hans Rohnert suggested some useful elaboration on our linguistic theme.

Joshua Kerievsky, our PLoP '98shepherd, helped to guide these patterns towards the form in which you now see them, while keeping the wolves at bay.

References

[Alexander 1979]
Christopher Alexander
The Timeless Way of Building
Oxford University Press, Oxford, UK, 1979
http://www.oup-usa.org/

[Alexander et. al 1977]
C. Alexander, S. Ishikawa, and M. Silverstein
A Pattern Language
Oxford University Press, Oxford, UK, 1977
http://www.oup-usa.org/

[Arthur 1992]
W. Brian Arthur
Increasing Returns and the New World of Business
Scientific American, February 1990

[Arthur 1995]
W. Brian Arthur
Complexity in Economic and Financial Markets
Complexity, vol. 1, no. 1, April 1995.

[Arthur 1996]
W. Brian Arthur
Positive Feedbacks in the Economy
Harvard Business Review, July-August 1996

[Cohen 1981]
Danny Cohen
On Holy Wars and a Plea for Peace
University of Southern California/Information Sciences Institute
USC/ISI IEN 137, 1 April 1980
IEEE Computer, Vol. 14, No. 10, October 1981

[Coplien 1995]
James O. Coplien
A Generative Development-Process Pattern Language
First Conference on Pattern Languages of Programs (PLoP '94)
Monticello, Illinois, August 1994
Pattern Languages of Program Design
edited by James O. Coplien and Douglas C. Schmidt
Addison-Wesley, 1995

[Cunningham 1995]
Ward Cunningham
The CHECKS Pattern Language of Information Integrity
First Conference on Pattern Languages of Programs (PLoP '94)
Monticello, Illinois, August 1994
Pattern Languages of Program Design
edited by James O. Coplien and Douglas C. Schmidt
Addison-Wesley, 1995

[Foote & Opdyke 1995]
Brian Foote and William F. Opdyke
Lifecycle and Refactoring Patterns that Support Evolution and Reuse
First Conference on Patterns Languages of Programs (PLoP '94)

Monticello, Illinois, August 1994
Pattern Languages of Program Design
edited by James O. Coplien and Douglas C. Schmidt
Addison-Wesley, 1995

This volume is part of the Addison-Wesley Software Patterns Series.

[Foote & Yoder 1996]
Brian Foote and Joseph W. Yoder
Evolution, Architecture, and Metamorphosis
Second Conference on Patterns Languages of Programs (PLoP '95)
Monticello, Illinois, September 1995
Pattern Languages of Program Design 2
edited by John M. Vlissides, James O. Coplien, and Norman L. Kerth
Addison-Wesley, 1996

This volume is part of the Addison-Wesley Software Patterns Series.

[Foote & Yoder 1998]
Brian Foote and Joseph W. Yoder
The Selfish Class
Third Conference on Patterns Languages of Programs (PLoP '96)
Monticello, Illinois, September 1996
Technical Report #WUCS-97-07, September 1996
Department of Computer Science, Washington University
Pattern Languages of Program Design 3
edited by Robert Martin, Dirk Riehle, and Frank Buschmann
Addison-Wesley, 1998
This volume is part of the Addison-Wesley Software Patterns Series.
Brian also wrote an introduction for this volume.

[Gabriel 1991]
Richard P. Gabriel
Lisp: Good News Bad News and How to Win Big
http://laputa.isdn.uiuc.edu/worse-is-better.html

[Gamma et alii 1995]
Eric Gamma, Richard Helm, Ralph Johnson, and John Vlissides
Design Patterns: Elements of Reusable Object-Oriented Software
Addison-Wesley Longman, Reading, MA, 1995

[Gates 1998]
Dominic Gates
The Pretext Interview: Brian Arthur talks to Dominic Gates
Pretext Magazine, May 1998

[Raymond 1997]
Eric S. Raymond
The Cathedral and the Bazaar
Linux-Kongreß 97
Würzburg, Germany, 21-23 May, 1997

[Roberts & Johnson 1996]
Don Roberts and Ralph E. Johnson
Evolve Frameworks into Domain-Specific Languages
Third Conference on Pattern Languages of Programs (PLoP '96)
Monticello, Illinois, September 1996
Pattern Languages of Program Design 3
edited by Robert Martin, Dirk Riehle, and Frank Buschmann
Addison-Wesley, 1997

[Severance 1998]
Charles Severance
William Kahan: An Interview with the Old Man of Floating-Point
An abbreviated version appeared in IEEE Computer, March 1998

This page has been referenced 00706 times since 5/11/98.

Brian Foote **foote@cs.uiuc.edu**
Last Modified: ***29 July 1998***