DML editor

I18n

precious stones
&
vegetables

Miha Filej

Internationalization

a word so long it doesn’t fit on a slide

i18n

Internationalization is usually abbreviated down to i18n.

$$\mathrm{i18n} = \mathrm{t9n} + \mathrm{l10n}$$

What does it mean?

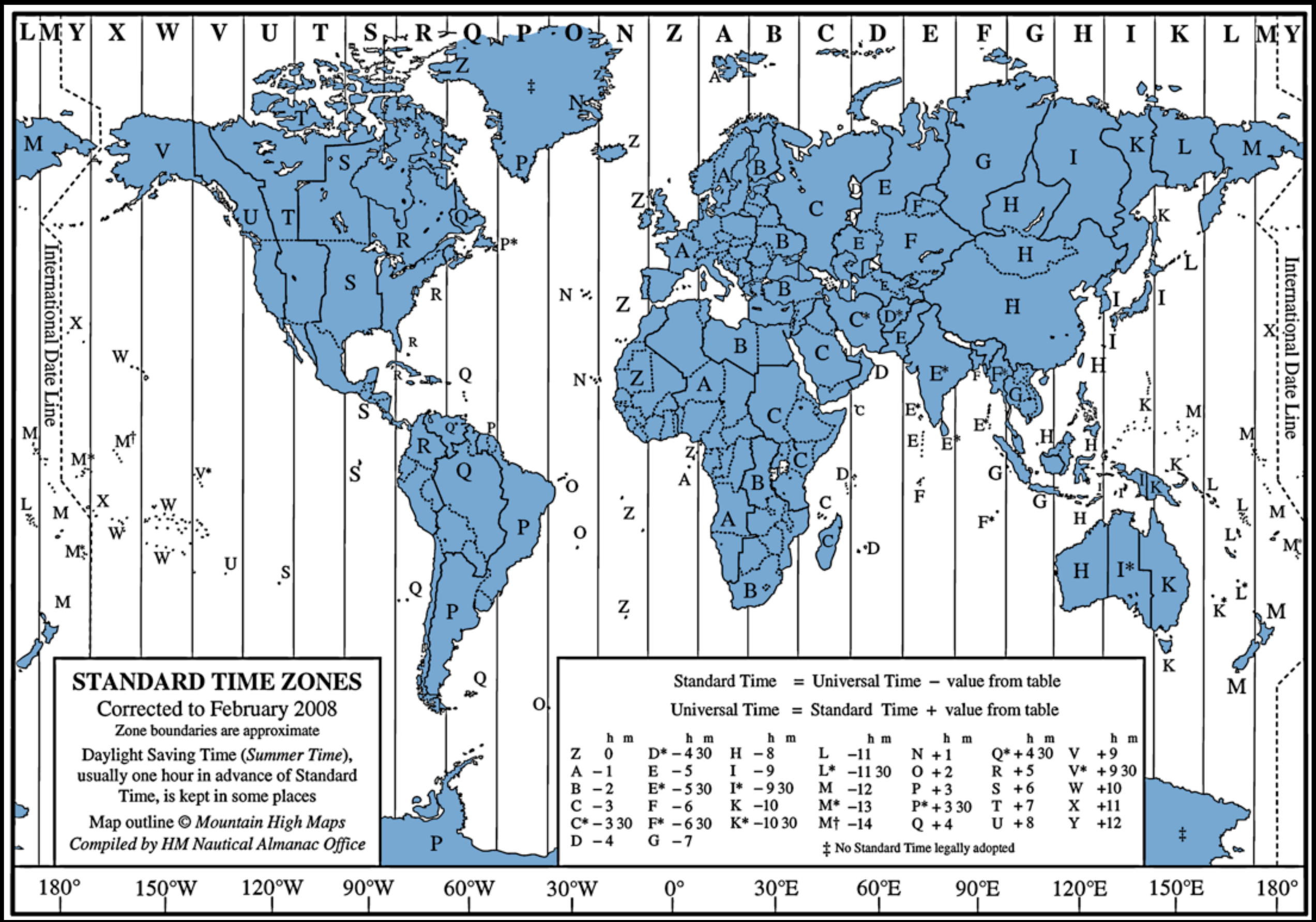
Translation

Lots and lots of strings in our application need to adapt to the user's language. Translation could easily be the most important part of the i18n process, but it is not enough by itself.

Localization

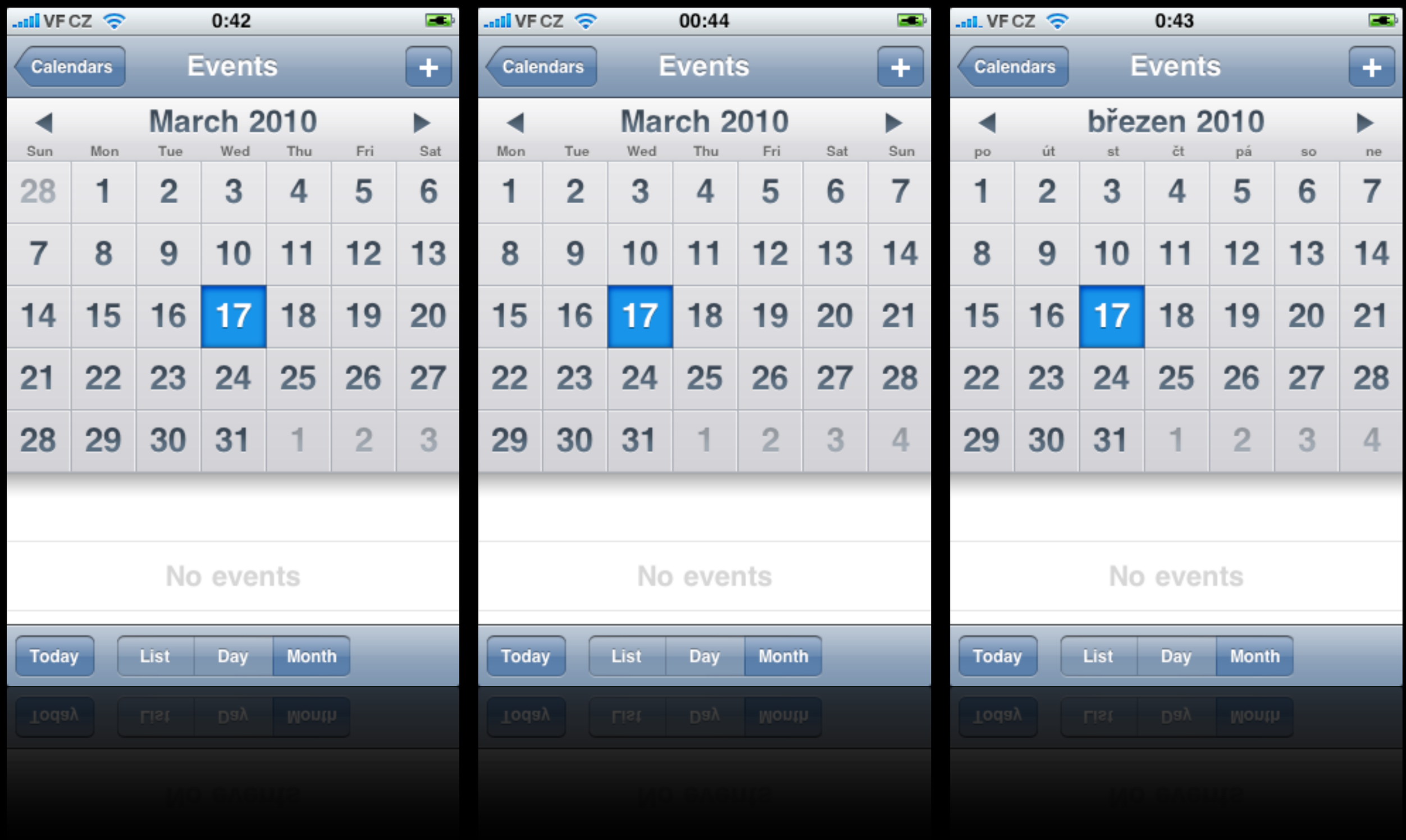
There is also localization.

It’s hard to achieve complete localization. It mostly depends on the needs of our application and on how much do we want to complicate our lives.



A very common case of l8n are timezones.

If our user base covers a big enough region, we have to make sure that every date that comes out of our application is handled within a proper timezone.



Usually we wan't our users to feel good, so we try to make our application behave nicely and provide aids. It is often useful to provide a calendar or so called date-picker along with the text box.

This are localized versions of the calendar on my phone. The one on the far right is the czech version. The other two are both English, but they're not the same. Can you guess which is the US and which is the UK version?

In the US week starts on Sunday.

```
> select created_at from publications;
+---------------------+
| created_at          |
+---------------------+
| 2010-03-17 00:49:31 |
| 2010-03-14 01:10:04 |
+---------------------+
2 rows in set (0.00 sec)
```

21 **minutes** ago
21 **minut** nazaj

3 **days** ago
pred 3 **dnevi**

Another way to enhance user experience is to hide ugly (database) date representations from the user.

A nicer way is to refer to a past time in the form of a sentence.

It’s not just about translating words, sentences change according to the context.

And there is also noun pluralization. We might have 1 “day” or 2 or 3 “days” ago.

```
count(n, "people")
```

1 person	1 človek
2 people	2 človeka
3 people	3 ljudje
…	…

1) So we probably want to have a function that takes a non-negative integer and a string
2) and returns a pluralized string
3) But apart from having to know the plural forms of the nouns, there are other exceptions, like dual in slovenian (we refer to a set with two items differently than to a set with more than two)

1) The localization issues we encountered so far are relatively easy to counter. Depending on which regions we plan to support we may encounter tougher problems.

2) One such example are regions that use right-to-left writing. This is the google localized for Egypt for example. The problem is tougher to tackle because it is not enough to modify the strings, but we have to change the way the UI is rendered.

Cultural differences

Cultural differences: offending words etc.

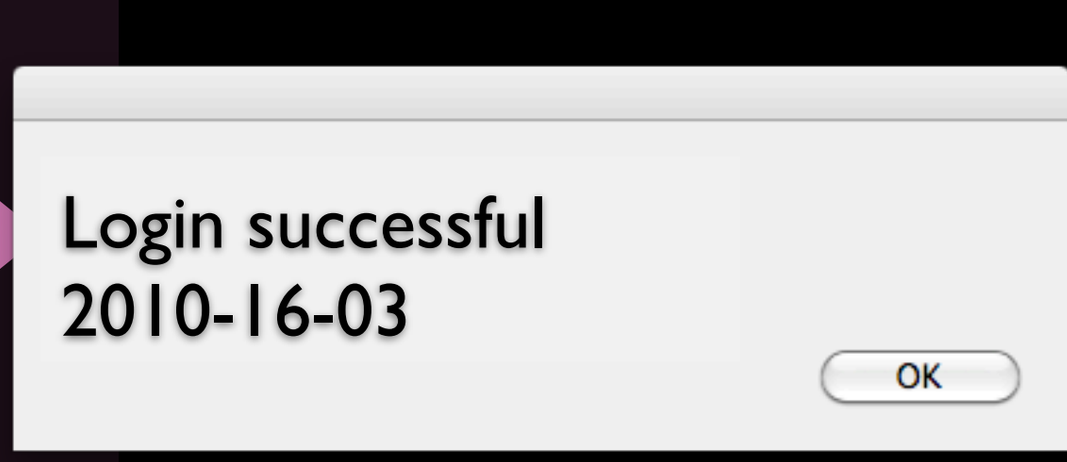
But this is further than we want to go. We probably just want to go step by step, translating strings first and localizing other data later on, depending on the importance.

Summary

- translation
- time and date formats
- timezones
- pluralization
- other language specific behavior

Problem

```
message = "Please log in"

user_input = prompt(msg)

if (verify(user_input)) {
  alert("Login successful"
        + Date.today())
} else {
  alert("Back off!")
}
```

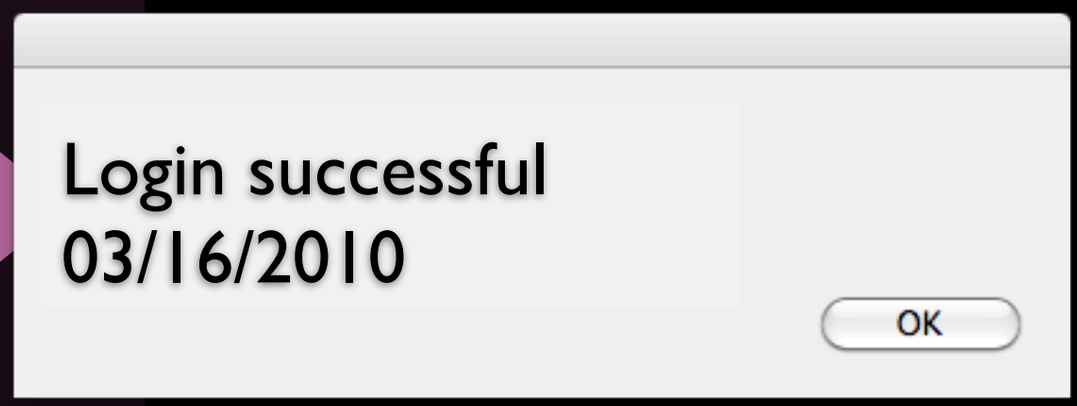


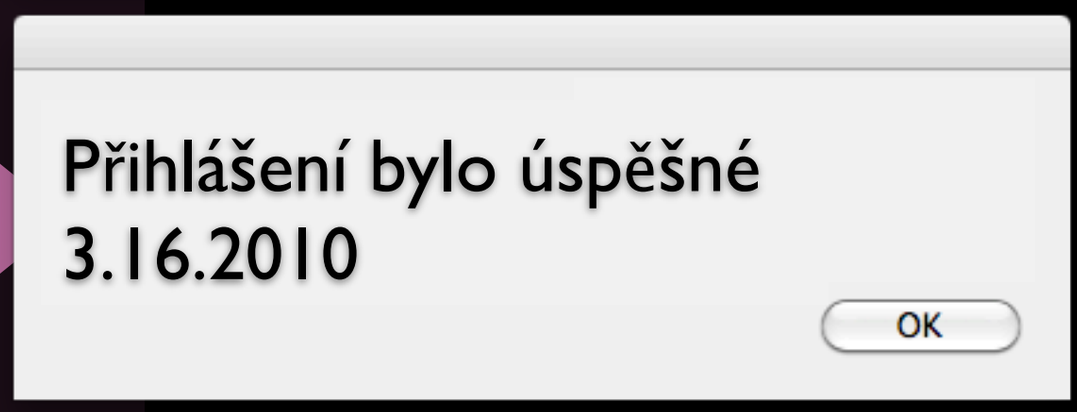
Basically we can reduce the problem to this:

We'd like to translate an application
We want to avoid any logic to handle the translations in the current code
We want to change the code as little as possible so we don't break things

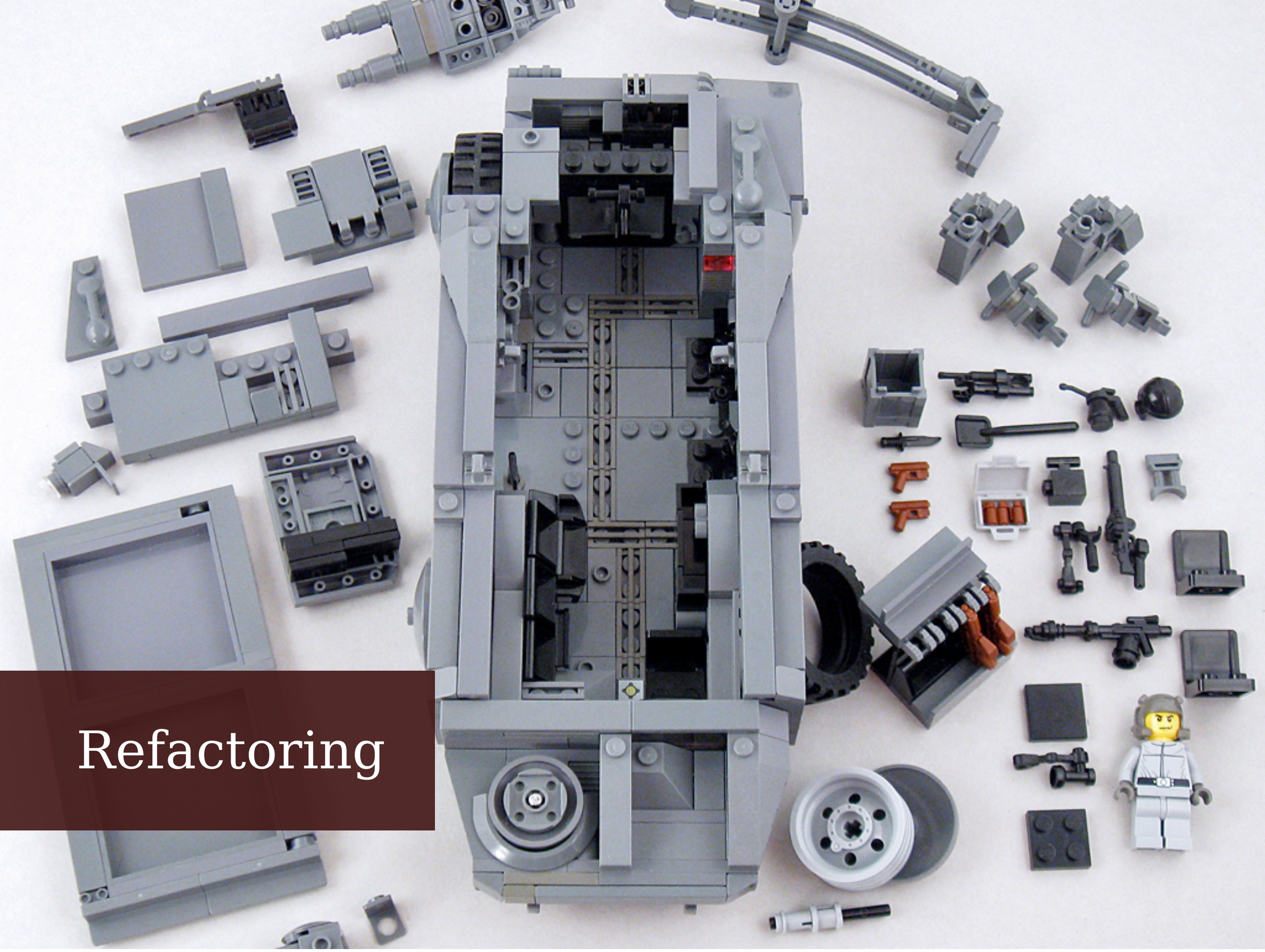
Goal

```
message = t("Please log in")

user_input = prompt(msg)

if (verify(user_input)) {
  alert(t("login.success"))
       + l(Date.today()))
} else {
  alert(t("login.error"))
}
```





Ideally we would set the locale at the beginning of the interaction with the user. Then the translation and localization functions would handle translation and localization depending on the set locale. We'd probably want to set their names to something short, like T and L in this example.



Refactoring

But all this changes in the code are very likely to introduce a lot of bugs. How do we cope with that?

integration

functional

unit

Possible solution: automated testing

Ruby

Ruby is a general purpose, highly dynamic, OO language.

It’s fairly new, conceived in japan ~ 1993, first public release in 1995. It only became popular outside japan in 1999, very popular in the last 5 years because of web frameworks.

> *I wanted a scripting language*
> *that was more powerful than Perl,*
> *and more object-oriented than Python.*
> *That's why I decided to design*
> *my own language.*

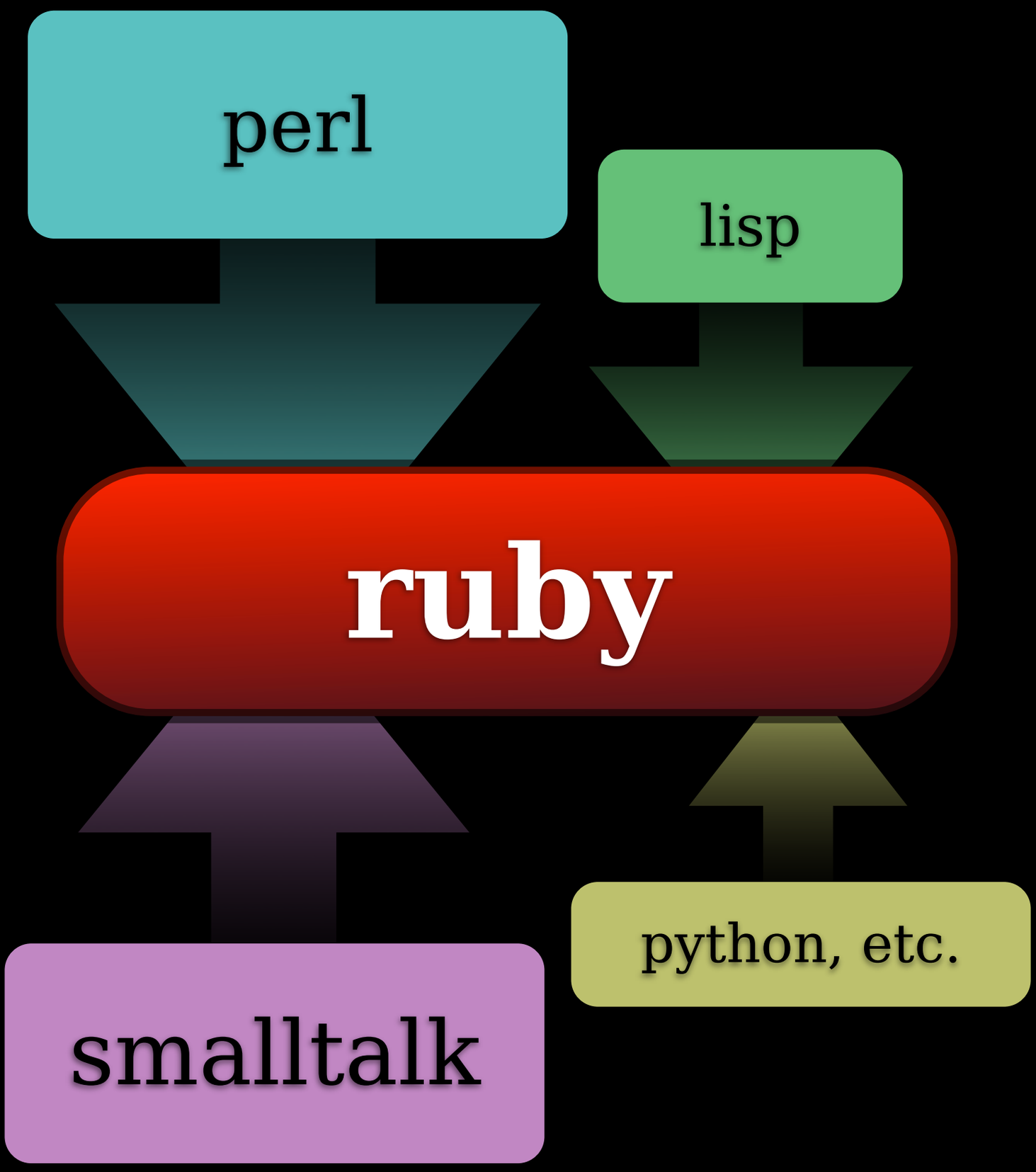
—Yukihiro Matsumoto

Ruby supports many programming paradigms (OO, functional, imperative, reflective - doesn't impose a programming style to the programmer)

It has dynamic and duck typing.
It is interpreted.

Implementations

MRI, JRuby (JVM), IronRuby (.NET),
MacRuby (ObjC), Rubinius…



- extremely dynamic
- emphasizes programmer friendliness
- designed for productivity and fun
- emphasizes human, rather than computer needs

But it can be slow at times, or have a bigger footprint than some other languages.

The main idea is that nowadays hardware is cheaper than programmers, and it seems to pay off.

Cucumber

Behaviour Driven Development with elegance and joy

1: Describe behaviour in plain text

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario: Add two numbers
    Given I have entered 50 into the calculator
    And I have entered 70 into the calculator
    When I press add
    Then the result should be 120 on the screen
```

2: Write a step definition in Ruby

```
Given /I have entered (.*) into the calculator/ do |n|
  calculator = Calculator.new
  calculator.push(n.to_i)
end
```

3: Run and watch it fail

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
      uninitialized constant Calculator (NameError)
      ./features/step_definitons/calculator_steps.rb:2:in `Given /
      features/addition.feature:7:in `Given I have entered 50 into
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

4. Write code to make the step pass

```
class Calculator
  def push(n)
    @args ||= []
    @args << n
  end
end
```

5. Run again and see the step pass

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/additi
    Then the result should be 120 on the screen      # features/additi
```

6. Repeat 2-5 until green like a cuke

```
$ cucumber features/addition.feature
Feature: Addition  # features/addition.feature
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers
  Scenario: Add two numbers                          # features/additi
    Given I have entered 50 into the calculator      # features/step_d
    And I have entered 70 into the calculator        # features/step_d
    When I press add                                 # features/step_d
    Then the result should be 120 on the screen      # features/step_d
```

7. Repeat 1-6 until the money runs out

Cucumber lets software development teams describe how software should behave in plain text. The text is written in a **business-readable domain-specific language** and serves as documentation, automated tests and development-aid - all rolled into one format.

We want swag!

The money raised for this campaign will be spent to produce Cucumber swag to promote Cucumber: T-shirts, cups and other things.

Download

You need Ruby installed. Then just run

```
gem install cucumber
```

from a command prompt. Now, run

```
cucumber --help
```

The **wiki** has more information.

Testimonials

Now let's connect ruby with the topic from earlier – automated testing.

The ruby community gained an interesting tool last year.

If you remember the stack from a few slides back, cucumber would sit right on top of integration testing. Id does integration testing, but with an attitude. It's designed for behavior driven development.

The idea is that we specify a feature we'd like our application to have, then
– write a cucumber test or (so called **feature**),
– which will fail at this stage, because we haven't actually written any code yet.

We then proceed with implementing the feature, making the steps of a feature pass one after another, until all of them are **green.**

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario Add two numbers
    Given I have entered <input_1> into the calculator
    And I have entered <input_2> into the calculator
    When I press <button>
    Then the result should be <output> on the screen

  Examples:
    | input_1 | input_2 | button | output |
    | 20      | 30      | add    | 50     |
    | 2       | 5       | add    | 7      |
    | 0       | 40      | add    | 40     |
```

This is an example of a cucumber feature. As you can see, it is written in plain text. The reason for this is that we want the feature specifications to be understood not only by programmers, but also by domain experts.

The idea is that before implementing something, before writing **any** code, you sit down with your customer/coworkers/boss/project manager and write down the specification for the feature, so everyone will understand what is being worked on and

And when something breaks, everyone can see what went wrong.

demo

[cucumber calculator demo]

```
Feature: Addition
  In order to avoid silly mistakes
  As a math idiot
  I want to be told the sum of two numbers

  Scenario Add two numbers
    Given I have entered <input_1> into the calculator
    And I have entered <input_2> into the calculator
    When I press <button>
    Then the result should be <output> on the screen

  Examples:
    | input_1 | input_2 | button | output |
    | 20      | 30      | add    | 50     |
    | 2       | 5       | add    | 7      |
    | 0       | 40      | add    | 40     |
```

Another interesting feature is that the syntax for specifying features is not fixed and can thus be written in any language.

```
Funzionalitá: somma
  Per evitare di fare errori stupidi
  Come utente
  Voglio sapere la somma di due numeri

  Scenario: la somma di due numeri
    Dato che ho inserito 5
    E che ho inserito 7
    Quando premo somma
    Allora il risultato deve essere 12
```

Calculator addition feature in italian.

フィーチャ: 加算
  バカな間違いを避けるために
  数学オンチとして
  2つの数の合計を知りたい

  シナリオテンプレート: 2つの数の加算について
    前提 <値1> を入力
    かつ <値2> を入力
    もし <ボタン> を押した
    ならば <結果> を表示

例:

値1	値2	ボタン	結果
20	30	add	50
2	5	add	7
0	40	add	40

And japanese.

back to i18n

GNU gettext

(a.k.a. the underscore method)

Probably the oldest well known i18n solution is GNU gettext. I’ve heard people refer to it as the grandfather or the dinosaur of Internationalization. A lot of modern solutions is still based on it, in one way or another.

ruby & i18n

What are the i18n tools available in the ruby ecosystem?

As I mentioned ruby became popular with web applications, and naturally i18n is very common in this field. So after 2005, when rails and other frameworks were being adopted more an more, different solutions surfaced.

There are a few ruby **gettext** implementations and others that use **database** or the **filesystems** to store translations, but each of them originated from different parts of the community and each project tried to solve **different pro**blems. There were major **incompatibilities** between them, they were often targeted at a specific framework and it was difficult to get something working for with new language and locale.

```
require "i18n"
```

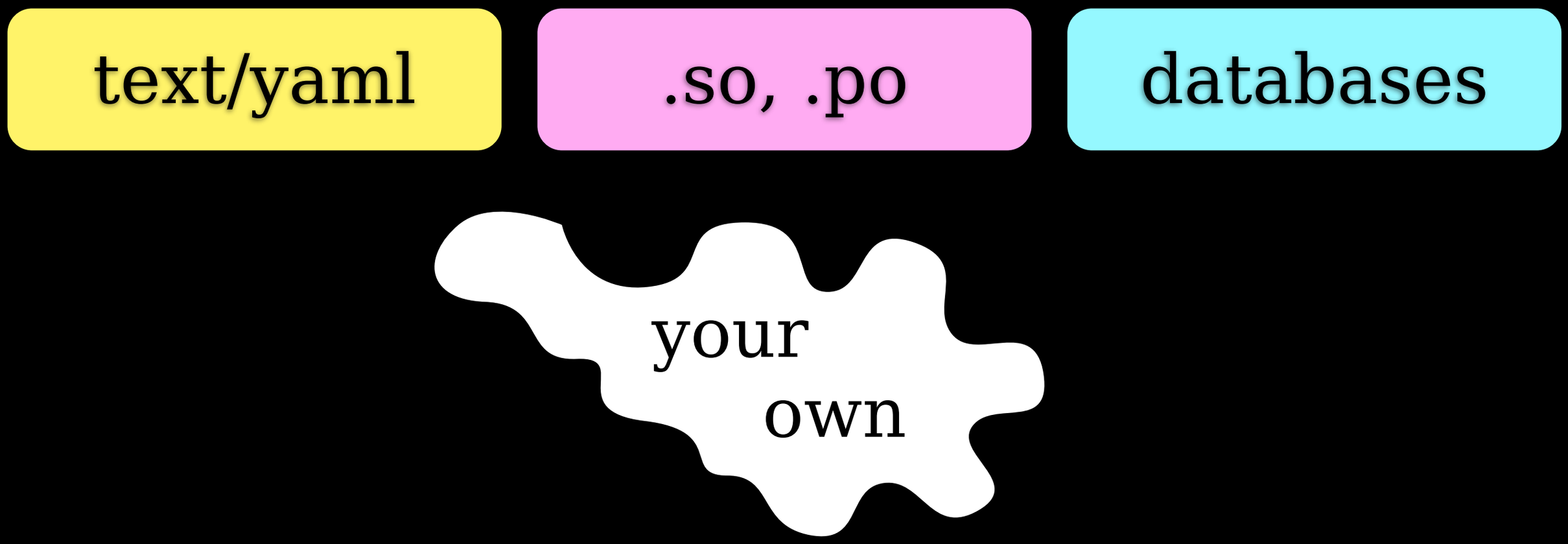
Then in 2007 an effort to make a generic i18n library emerged. People that were previously working on all those project started to work together, but their goals were too different and in they couldn't agree on much in a long time. They took a break and after half a year they agreed on a different approach:

Rather than solving all the translation and localization cases poorly, they decided to make a library that will provide an standard interface, so that it could be easily extended.

It is called i18n.

It provides the basic facilities for translating a language similar to english, and it provides some basic localization.

Backends



It comes with a few ways to store the translated data.

The main backend is called SimpleBackend and it stores translations into yaml files. There is support for storing translations into databases, and there is also basic support for gettext's .so and .po files.

There's a few more features, but the beauty is that there aren't many more. After the library was conceived, many different libraries that interface with it started emerging, which are now actually compatible between them, and a programmer can choose which one to use depending on the needs of the target language and locale.

demo

[activesupport pluralization demo]

Ticket #10919 (closed enhancement: wontfix)

incorrect pluralization

Opened 2 years ago
Last modified 2 years ago

Reported by:	srbaker	Assigned to:	core
Priority:	normal	Milestone:	2.x
Component:	ActiveSupport	Version:	edge
Severity:	normal	Keywords:	verified tiny
Cc:			

Description

Rails improperly pluralizes the word "penis". From the New Oxford American Dictionary:

penis |'pēnis| noun (pl. -nises or -nes |-nēz|) the male genital organ of higher vertebrates, carrying the duct for the transfer of sperm during copulation. In humans and most other mammals, it consists largely of erectile tissue and serves also for the elimination of urine.

As described here, there are two appropriate pluralizations of the word penis: penises or penes. The more common pluralization, penises, should be used.

While technically this is a defect, "enhancement" feels like a more appropriate word to describe this particular problem.

Attachments

- penis_enhancement.diff (1.0 kB) - added by *srbaker* on 01/24/08 22:22:46.
  Enhancement to add penis plural to the inflector.

Infamous rails ticket

ج

References

http://cukes.info/

http://dev.rubyonrails.org/ticket/10919

Credits

http://www.flickr.com/photos/28192677@N06/3611301682/
http://www.flickr.com/photos/cipherswarm/2414578959/
http://www.survivalworld.com/maps/index.html

Miha Filej
miha@filej.net
http://twitter.com/mfilej