domain specific languages

weniger Code und bessere Software mit DSLs

Alexander Lang

domain specific languages

weniger Code und bessere Software mit DSLs

Alexander Lang

About me

- Dipl.-Inf. (FH) Alexander Lang, Berlin
- Geschäftsführer von <*no name yet*>
- Entwicklung von Webapplikationen (*2.0*)
- “Enterprise Software”
- Extreme Programming / Agile

domain specific languages

- Begriffsklärung, Hintergrund
- Einsatz, Beispiele
- Implementierung

domain specific languages

- Begriffsklärung, Hintergrund
- Einsatz, Beispiele
- Implementierung

domain specific language (DSL)

problemspezifisch

ein Problem

domain specific language

Sprachen

menschenlesbar

maschinenlesbar

programmieren

“kleine Programmiersprachen”

Vorteil:

sehr einfach (klein)

Sprachelemente auf ein Problem zugeschnitten

“richtige” Programmiersprache:

allgemein verwendbar

eine Syntax/Vokabular

um viele Probleme zu lösen

unspezifisch

Zwei Arten von DSLs:

intern/extern

innerhalb einer

“großen” Sprache

gebunden an Syntax

Compiler/Interpreter der Muttersprache

intern/extern

komplett selbständig

eigene Syntax/ Vokabular

Parser erforderlich

domain specific languages

- Begriffsklärung, Hintergrund
- Einsatz, Beispiele
- Implementierung

komplexe Sprachen

keine
problemspezifischen
Sprachelemente

einfaches Problem -

komplexe Lösung


```
Calendar cal = Calendar.getInstance();
cal.set(Calendar.HOUR, 10);
cal.set(Calendar.MINUTE, 23);
cal.set(Calendar.DAY, 30);
cal.set(Calendar.MONTH, 9);
cal.set(Calendar.YEAR, 2006);
Date time = cal.getTime();
```

domain specific languages

- Begriffsklärung, Hintergrund
- Einsatz, Beispiele
- Implementierung

```
Calendar cal = Calendar.getInstance();
cal.set(Calendar.HOUR, 14);
cal.set(Calendar.MINUTE, 00);
cal.set(Calendar.DAY, 30);
cal.set(Calendar.MONTH, 9);
cal.set(Calendar.YEAR, 2006);
Date time = cal.getTime();
```

new SimpleDateFormat(...).parse

heute Mittagspause

heute 2 Uhr

today.at 2

DateTime extension von Ruby on Rails

DSL zur Beschreibung von Zeiten

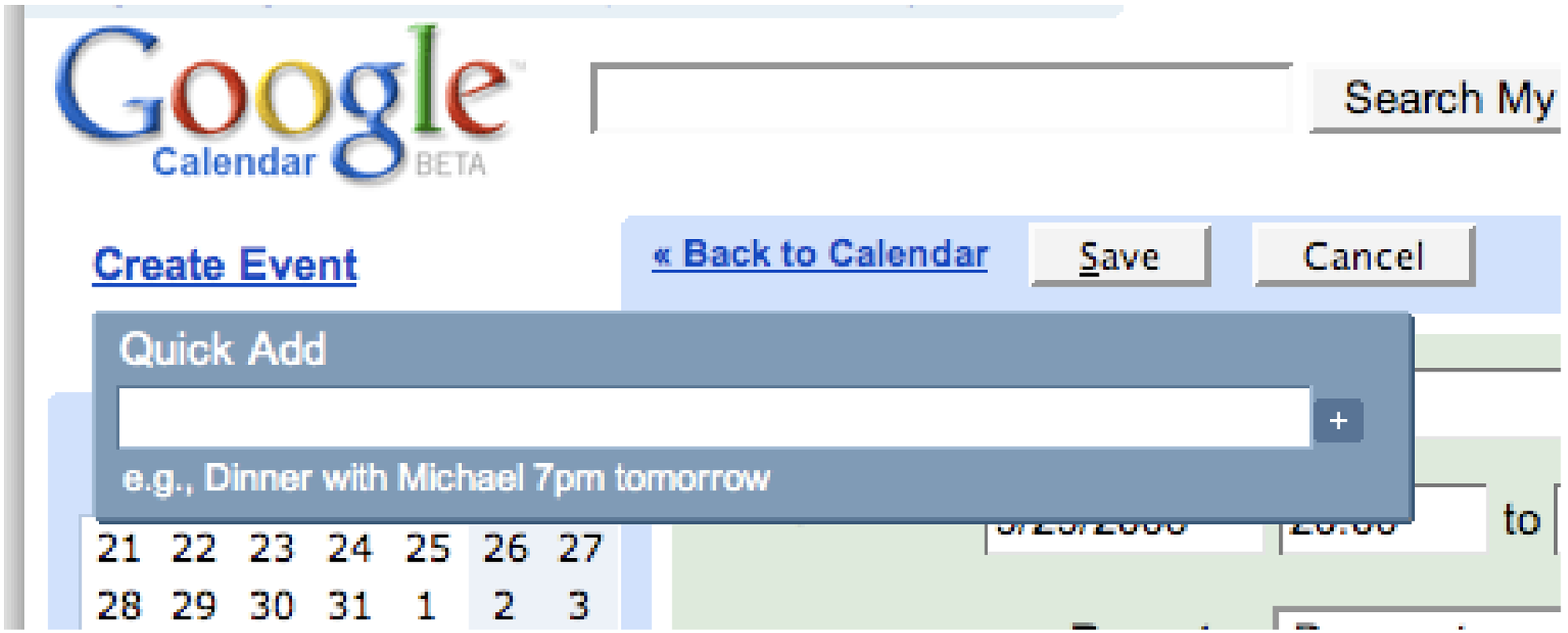
interne DSL, Ruby

~~today.at 2~~

2.days.ago

1.week.from_now.at_midnight

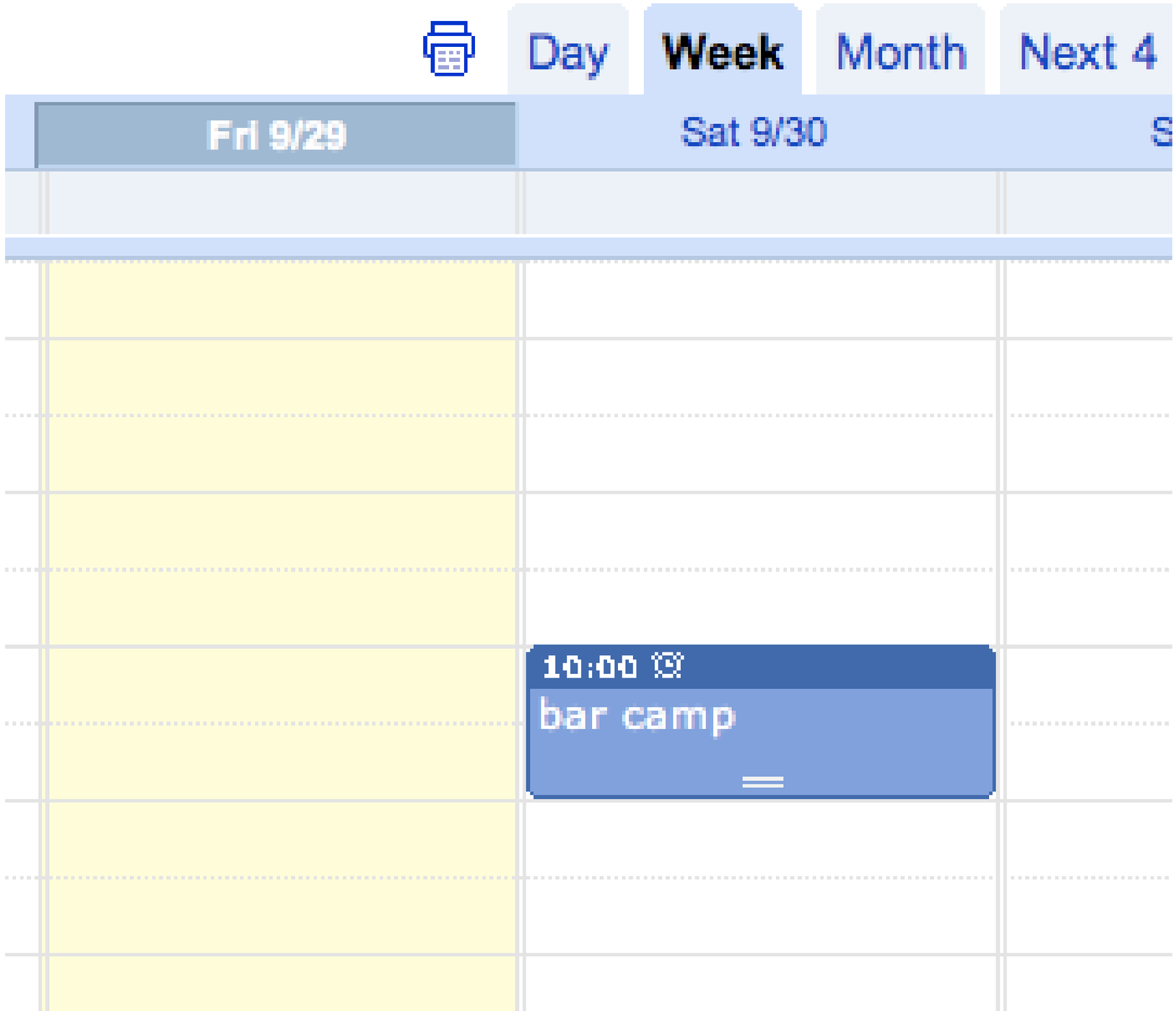
Google Calendar


Google
Calendar BETA
Search My
Create Event
« Back to Calendar
Save
Cancel
Quick Add
+
e.g., Dinner with Michael 7pm tomorrow
to
21 22 23 24 25 26 27
28 29 30 31 1 2 3

Quick Add

“bar camp tomorrow at 10am”

)6 at 10:00.



Datumsbeschreibung via DSL für Benutzer

Beispiel 2/3:

“SELECT * FROM users”

SQL

“SELECT * FROM users”

```
SELECT p.last_name, p.first_name, p.address, p.phone, p.email,
pos.full_name as position, e.salary, e.started_at, e.comments, c.id

FROM persons AS p

LEFT JOIN employees AS e ON p.id = e.id

LEFT JOIN citizens AS c ON c.id = p.id

LEFT JOIN position_lookup AS pos ON e.position_id = pos.id

WHERE p.id = (SELECT id FROM attendees WHERE event = 'bar
camp')
```

```
query = "SELECT p.last_name, p.first_name, p.address, p.phone, p.email, pos.full_n
query.execute
```

Problem: SQL generieren in <Lieblingsprogrammiersprache>

Problem: SQL generieren in Ruby

interne DSL

```
sql.select(

:persons =>
[:first_name, :last_name, :phone, :email],

:employees =>
[:position, :started_at, :comments],

:citizens => :id,

:position => :full_name

).from_persons

.where :id => :attendees, :event => 'bar
camp'
```

```
p.first_name, p.last_name, p.phone, p.email
FROM persons as p
```

```
:persons =>
[:first_name, :last_name, :phone, :email]

from_persons
```

~~LEFT JOIN~~

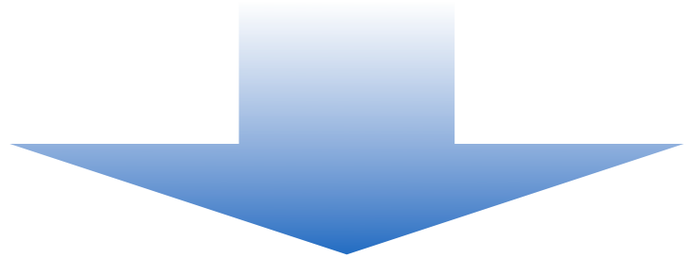
```
WHERE p.id = (SELECT id FROM attendees WHERE event =
'bar camp')
```

```
where :id => :attendees, :event => 'bar
camp'
```

```
WHERE p.id = (SELECT id FROM attendees WHERE event =
'bar camp')

from bar_camp_attendees
```

```
p.first_name, p.last_name, p.phone, p.email
FROM persons as p
```



```
:persons => :all_important_details
```

Syntaxübersetzung

Übersetzungslogik

Beispiel 3/3:

Prämien für Mitarbeiter

Business Natural Language

Jay Fields, http://jayfields.blogspot.com/2006/07/business-natural-language-material.html

Ermittlung der Prämienauszahlung

Unternehmensgewinn

John Jones

bonuses: $10,000 if the company posts a profit of 1 million

$20,000 if the company posts a profit of 2 million

$30,000 if the company posts a profit of 3 million or more

5% of toothbrush profits

bonus pay month: January

```
if Time.now.month == 1
  bonus += 1000000 if profit.total > 100000000
  bonus += 1000000 if profit.total > 200000000
  bonus += 1000000 if profit.total > 300000000
  bonus += profit.toothbrush * 0.05
end
```

domain specific language:

apply bonus of ten thousand dollars if the total profit is greater than one million dollars and the current month is equal to january

apply bonus of ten thousand dollars if the total profit is greater than two million dollars and the current month is equal to january

apply bonus of ten thousand dollars if the total profit is greater than three million dollars and the current month is equal to january

apply bonus of the toothbrush profit times five percent if the current month is equal to january

~~Programmierer~~

User

domain specific languages

- Begriffsklärung, Hintergrund
- Einsatz, Beispiele
- Implementierung

```
class BonusCalculationContext
  extend Verbosity
  bubbles :than, :is, :profit, :the, :to, :of, :bonus
  numerics :thousand=>1000, :million=>1000000,
           :one=>1, :two=>2, :three=>3, :five=>5, :ten=>10
  operations :greater=>">", :equal=>"==", :times=>"*", :less=>"<"
  constants :dollars=>100, :percent=>0.01, :january=>7, :february=>7, :march=>7

  def initialize
    @bonus_amount = 0
  end

  def self.evaluate(path_to_bnl)
    context = BonusCalculationContext.new
    context.extract_name(path_to_bnl)
    specifications = File.readlines(path_to_bnl)
    specifications.each { |spec| context.instance_eval(spec) }
    context.resulting_bonus
  end

  def last_profit
    @last_year_profit ||= Profit.find :first
  end

  def total(arg)
    result = eval "#{last_profit.total} #{arg}"
    return result.round if result.respond_to? :round
    result
  end

  def month(arg)
    ".month #{arg}"
  end

  def current(arg)
    eval "Time.now#{arg}"
  end

  def toothbrush(arg)
    eval("#{last_profit.toothbrush_in_cents} #{arg}").round
  end

  def drug(arg)
    eval("#{last_profit.drug_in_cents} #{arg}").round
  end

  def apply(amount)
    @bonus_amount += amount
  end

  def extract_name(path)
    name_parts = File.basename(path).chomp!('.txt').split('_')
    name_parts = name_parts.collect { |part| part.capitalize }
    @employee_name = name_parts.join(" ")
  end

  def resulting_bonus
    Bonus.new(@employee_name, @bonus_amount)
  end
end
```

apply bonus of ten thousand dollars if the total profit is greater than one million dollars and the current month is equal to january

```
spec = 'apply bonus of...'
object.instance_eval(spec)
```

Wort > Methodenaufruf

Bubble Words

the, of, bonus, and

apply bonus of ten thousand dollars if the total profit is greater than one million dollars and the current month is equal to january

Ersetzung

apply bonus of ten thousand dollars if the total profit is greater than one million dollars and the current month is equal to january

apply bonus of 10 thousand dollars if the total profit is greater than one million dollars and the current month is equal to january

apply bonus of 10 *1000
dollars if the total profit is greater
than one million dollars and the
current month is equal to january

apply bonus of 10 *1000
dollars if the total profit is > than
one million dollars and the
current month is equal to january

apply bonus of 10 *1000
dollars if the total profit is > than
one million dollars and the
Time.now.month is equal to january

apply bonus of 10 *1000
dollars if the total profit is > than
1 *1000000 dollars and the
Time.now.month is equal to january

apply bonus of 10 *1000
dollars if the total profit is > than
1 *1000000 dollars and the
Time.now.month is == to january

apply bonus of 10 *1000
dollars if the total profit is > than
1 *1000000 dollars and the
Time.now.month is == to 1

```
10 *1000
if total > 1 *1000000 and
Time.now.month ==1
```

```
eval “10 *1000
if total > 1 *1000000 and
Time.now.month ==1”
```

10.000

Zusammenfassung

DSL =

problemspezifische Sprache

enthält Sprachelemente
um bestimmtes
Problem zu lösen

nicht allgemein verwendbar

Einfachheit durch eingeschränkten Problembereich

Lösen kleine Probleme...

1.day.from_now

... und nicht ganz so kleinen Probleme ...

... der Entwickler

```
select
(:personal_details).from
(:users, :employees)
```

... aber auch der

Benutzer ...

Google Calendar BETA

Search My

Create Event

« Back to Calendar Save Cancel

Quick Add

+

e.g., Dinner with Michael 7pm tomorrow

21 22 23 24 25 26 27

28 29 30 31 1 2 3

to

apply bonus of ten thousand if total profit ...

intern - in

Muttersprache

eingebettet

einfach zu implementieren

z.B. in Ruby

extern - eigenständige Sprache

eigener Parser

Vorteile:

reduzieren Komplexität

Probleme besser beschreiben

weniger Code

weniger Aufwand

weniger Fehler

bessere Software

domain specific languages

weniger Code und bessere Software mit DSLs

Danke.

Alexander Lang
mailto: me@langalex.org