5 Semantic Language Design

Semantics First!

Example: Calendar DSL

Syntactic Design
Syntax: Sharing Thoughts

Syntax: Agreed-upon representation for semantic concepts
User-Centered Design?

Problems of user-centered designs

- Biased (small sample sizes)
- Limited (traditions are innovation averse)
- Mostly syntax centered (questionnaires)

Focus on domains

Activity-Centered Design (Don Norman)
Focus on Semantics

- MDD / MDA
- Domain-Driven Design

“Ubiquitous Language”

English +
Names defined in the model
Powerful Opposition ...

First sentence: “The specification of a programming language starts with its syntax.”

Felleisen et al. 2009
Origin of Language

"I just invented it... I call it 'O'."

© Original Artist

search ID: c2a0694

Concept

Name
Advantages of the Semantics-First Approach

• Compositional design leads to better, more modular structure
• Language development is less ad hoc
• Modularity supports maintainability
## Syntax & Semantics

<table>
<thead>
<tr>
<th>Syntax / Symbol</th>
<th>Object in the real world</th>
</tr>
</thead>
<tbody>
<tr>
<td>Remote Control</td>
<td></td>
</tr>
<tr>
<td>Fernbedienung</td>
<td></td>
</tr>
<tr>
<td>Laser Pointer</td>
<td></td>
</tr>
</tbody>
</table>
## Syntax & Semantics

<table>
<thead>
<tr>
<th>Syntax / Symbol</th>
<th>Object in the ideal world</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>The notion of fiveness</td>
</tr>
<tr>
<td>Five</td>
<td></td>
</tr>
<tr>
<td>V</td>
<td></td>
</tr>
</tbody>
</table>

The notion of fiveness

Diagram:
- Symbol '5'
- Symbol 'V'
- Hand gesture
- Image of ancient philosopher
Metalanguage for Denoting Semantic Objects

<table>
<thead>
<tr>
<th>Syntax / Symbol</th>
<th>Object in the ideal world</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>5 :: Int</td>
</tr>
<tr>
<td>five = 5</td>
<td></td>
</tr>
<tr>
<td>v = 5</td>
<td></td>
</tr>
<tr>
<td>iiiii = 5</td>
<td></td>
</tr>
</tbody>
</table>
5 Semantic Language Design

Semantics First!

Example: Calendar DSL

Syntactic Design
Example: Calendars
A Calendar DSL

Are these possible in your favorite calendar tool?

Repeated events:
The first weekday before the 15th of each month (except June and November)

Event timing constraints:
Rehearsal at least 2 days before event

Event templates:
t: Meeting with candidate, t+2: Colloquium, t+5: Dinner
Domain Decomposition

- **Domain Decomposition**
- **Domain Modeling**
- **Semantics-Driven DSL Design**
- **Syntactic Design**

Diagram:
- Domain Realm
- DSL
- Domain
- Relationship
- Data Type
- Type Constructor

- D1, D2
- R
- S
- L1, L2
- Haskell
Domain Decomposition

Date & Time

Map

Appointment

Date

Time

Appointment

Domain Decomposition
Domain Decomposition

Date & Time

Map

Appointment

Date

Map

Appointment

Time

Map

Appointment
Domain Modeling

Date & Time

Map

Appointment

data Month = Jan | ... | Dec
type Day = Int
data Date = D Month Day

type Hour = Int
type Minute = Int
data Time = T Hour Minute

type CalDT a = Map (Date,Time) a
type CalD a = Map Date a
type CalT a = Map Time a
type Cal a = CalD (CalT a)

data Map a b = a :→ b
| Map a b :&: Map a b
Language Composition

```haskell
data Map a b = a -> b
  | Map a b & Map a b

type Hour = Int
type Minute = Int

data Time = T Hour Minute

data Month = Jan | ... | Dec
type Day = Int
data Date = D Month Day

type CalDT a = Map (Date,Time) a
1. type CalD a = Map Date a
2. type CalT a = Map Time a
3. type Cal a = CalD (CalT a)
```
5 Semantic Language Design

Semantics First!

Example: Calendar DSL

Syntactic Design
The Need For Syntax

data Map a b = a → b
| Map a b #: Map a b

type Hour = Int
type Minute = Int
data Time = T Hour Minute

data Month = Jan | ... | Dec
type Day = Int
data Date = D Month Day

type CalD a = Map (Date, Time) a
type CalD a = Map Date a
type CalT a = Map Time a
type Cal a = CalD (CalT a)

week52 :: CalD String
week52 = D Dec 30 → "Work" #: D Dec 31 → "Party"

week52 :: Cal String
week52 = D Dec 30 → (T 8 0 → "Work") #: D Dec 31 → (T 22 0 → "Party")
Syntactic Design

**data** Month = Jan | ... | Dec
**type** Day = Int

**data** Date = D Month Day

**data** Date = D Jan
...
**halloween** = oct 31

**type** Hour = Int
**type** Minute = Int

**data** Time = T Hour Minute

**hours** h = T h 0
**mins** m = T 0 m
am 12 = T 0 0
am h = hours h
pm 12 = T 12 0
pm h = hours (h+12)
**before** = flip (-)
**after** = (+)

**week52** :: Cal String
**week52** = dec 30 → (am 8 → "Work") :&: dec 31 → (pm 10 → "Party")

> **week52**
Dec-30 → 08:00 → "Work" & Dec-31 → 22:00 → "Party"
A Time Micro DSL

Smart constructors

- hours $h = T h 0$
- mins $m = T 0 m$
- am $12 = T 0 0$
- am $h = hours h$
- pm $12 = T 12 0$
- pm $h = hours (h+12)$
- before = flip (-)
- after = (+)

extend

- midnight = am 12
- noon = pm 12

Special Values

- every :: Time $\rightarrow$ Time $\rightarrow$ [Time]
- every $t$ $s = s$:every $t$ ($s+t$)

Generator

- startingAt = ($$
- upTo :: Time $\rightarrow$ [Time] $\rightarrow$ [Time]
- upTo _ [] = []
- upTo e (t:ts) | e==t = [t]
- otherwise = t:upTo e ts

- till = flip upTo

Glue Key Word

- filter

Filter

> mins 5 `before` midnight
23:55

> every (mins 25) `startingAt` pm 11
[23:00,23:25,23:50,00:15,00:40,01:05,01:30,01:55,02:20,02:45,03:10, ...]

> every (mins 25) `startingAt` pm 11 `till` (mins 5 `after` am 1)
[23:00,23:25,23:50,00:15,00:40,01:05]
Domain Integration Syntax

Function ≈ Calendar Template

Combines Times & Appointments into Calendars

partyAt :: Hour → CalT String
partyAt h = hours 2 `before` h → "Work Out"
   &: h → "Party"

work :: CalT String
work = am 8 :→ "Work" :&: pm 6 :→ "Dinner"

party :: CalT String
party = work :&: partyAt 9

week52 :: Cal String
week52 = dec 30 :→ work :&: dec 31 :→ party

> week52
Dec-30 → 08:00 → "Work" & 18:00 → "Dinner" &
Dec-31 → 08:00 → "Work" & 18:00 → "Dinner" & 19:00 → "Work Out" & 21:00 → "Party"
Revisit Calendar Domain

Repeated events:
The first weekday before the 15th of each month (except June and November)

Event timing constraints:
Rehearsal 2 days before event

Event templates:
$t$: Meeting with candidate, $t+2$: Colloquium, $t+5$: Dinner

Query on time domain

Queries on calendars

Function $(\text{partyAt})$
Still More Composition ...

**Schema Composition ≈ Nested Language Composition**

```haskell
type Cal a = CalD (CalT a)

week52 :: Cal String
week52 = dec 30 :-> work :&: dec 31 :-> party
```

**Nested Calendars**

```haskell
type Duration = Time

type Cal a = CalD (CalT (a, Duration))
```

**Product Type ≈ Language Product**
More Design Variation ...

```haskell
type CalDT a = Map (Date, Time) a
type CalD a = Map Date a
type CalT a = Map Time a
type Cal a = CalD (CalT a)

type Priority = Int

type Tentative a = [(Priority, a)]
type Cal a = CalD (CalT (Tentative a))

conflicts :: Cal a → [Time]
...