Rethink Smalltalk

Index

1. Introduction
2. Problem
3. Goal
4. Solution
5. Demo of version 1 & 3
6. References

Introduction

- Founded Cosmocows with Wouter Gazendam
- Wouter is main implementer of our development environment (on top of Visualworks Smalltalk)
- I, Mathieu van Echtelt, use this environment to create ‘models’
- These models are succesfully in use by fire departments and health care organizations.
- This presentation is about our development environment and how to create models with it.

Problem

Problem: Developing database-backed web applications for business administration is hard.

So what do Business Administration systems need?

1. Persistency for model and data.
2. Audit support of model and data changes
3. Authorization support for model and data changes
4. Transaction support for model and data changes
5. Multiple concurrent 'user' support for changing model and data.
6. Evolution support for asynchronous model and data changes
7. Multi paradigm support for model and data formats

When these issues are not generally solved from the beginning,
the complexity of your system grows exponential.

When these are solved, we can make a huge step in development productivity

“Code should not be cluttered with irrelevant statements that may even be inaccurate”
[Dan Ingalls]

Goal

Goal: Resurrect design principles behind Smalltalk … in order to enable a single individual or team to continuously create, change and host database-backed web applications in a fast, easy, safe and structured way.

What are Smalltalk's design principles?

Daniel H. H. Ingalls, 'Design Principles Behind Smalltalk',
BYTE Magazine, August 1981.

Only 3 out of 17 are highlighted:

1. Personal Mastery
2. Automatic Storage Management
3. Reactive Principle

Design Principle
“Personal Mastery”

“If a system is to serve the creative spirit, it must be entirely comprehensible to a single individual”.

>> Very difficult in multi paradigm (business) world:

1. Data source tier; e.g. SQL relational databases for data consistency & persistency,
2. Business domain tier; e.g. OO for data manipulation & behavior.
3. Client application tier; e.g. HTTP/IP, Javascript.
4. Presentation tier; e.g. XML/HTML, CSS, ODF

(In Squeak we use Slang for Virtual Machine construction or Seaside’s Canvas for XHTML construction. This way you can still work in one paradigm (in this case Smalltalk). We need more of these! But these techniques are only a part of the solution.)

Design Principle
“Storage Management”

“To be truly "object-oriented", a computer system must provide automatic storage management”.

>> Smalltalk wasn’t created for data persistency, integrity and migrations in the first place, like relational databases are.

“Code should not be cluttered with irrelevant statements that may even be inaccurate”
[Dan Ingalls about automatic storage management]

Design Principle
“Reactive Principle”

“Every component accessible to the user should be able to present itself in a meaningful way for observation and manipulation”.

>> In today’s (business) world this means observation and manipulation over the internet with a standard secure web browser by more than one user simultaneously, in an authorized way.

Solution

“Stick to relative simple problems” [Dan Ingalls]

What we have in production/in development to solve relative complex problems:

1. Model description language
2. Model transformations
3. Model dispatcher (installer/handler/code generator)
4. Object persistency, integrity, and migration
5. A web based IDE to develop Models.
6. A web based administration system to manage customers, contracts, logins

Solution
“1. Model Description language”

- All about business domain issues,
- Nothing about technical facilitation like data persistency & formats, rendering techniques, transaction & session and memory management).

Solution
“2. Model Transformations”

- Model Description is automatically transformed into expressions for all involved system layers (SQL, OO, HTML, ODF, JS*).
- But wait; doesn’t this break the rule of “Model View Control” separation? In 95% of the situations it doesn’t matter. Automate 95%, override 5%.
- Round trip

Solution
“3. Model dispatcher”

- Model dispatcher (/code generator) mutates all layers into required state
- Administrates and handles every development process step as an explicit “change instruction” (AddSchemaChange, AddSlotChange, ChangePropertyChange, etc)
- One conceptual change is often expressed in multiple mutations within and over several software layers and languages.
- It is extremely inefficient and error prone for a developer to create all mutations by hand.
- Guarantee consistency over complete system.

Solution

“4. Object persistency, integrity, and migration”

- Relational database backend (Postgresql)
- Every object can be stored (again 95% rule, but can be overridden)
- Automatic (virtual) garbage collection (3 strategies)
- Every object change is stored with it’s Class format
- “Current” model and data in cache.
- Lazy data migration

(A interesting topic of its own; for next years ESUG presentation….)

Solution

5. A web based IDE to express, test, analyze and configure Models.

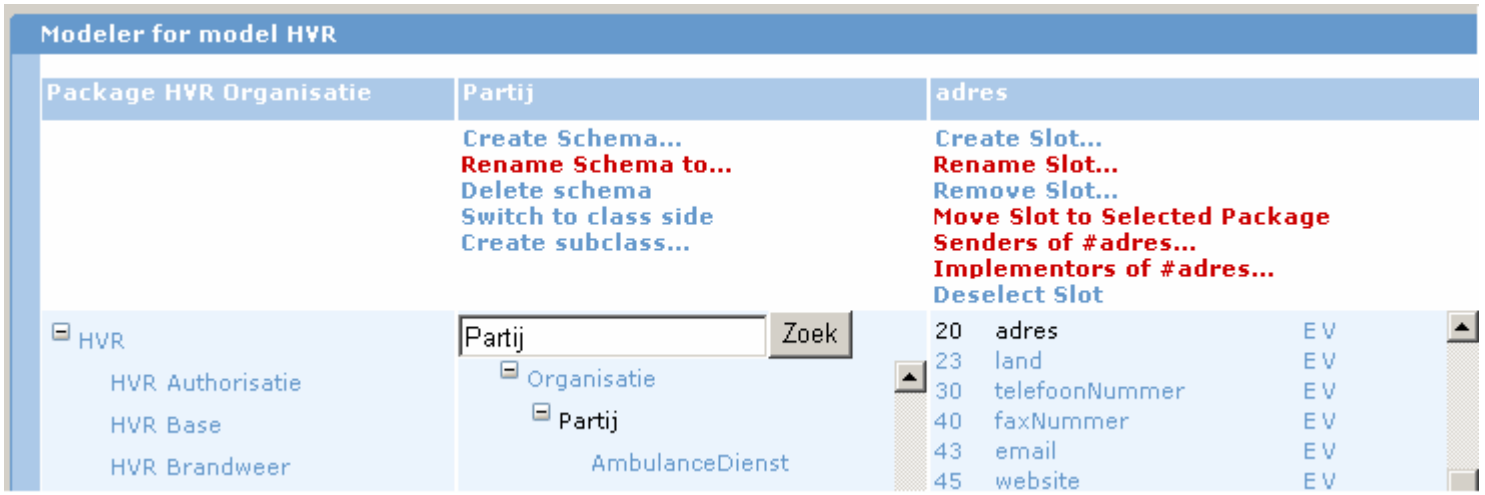
6. A web based administration system to manage customers, users, contracts, logins, and issues.

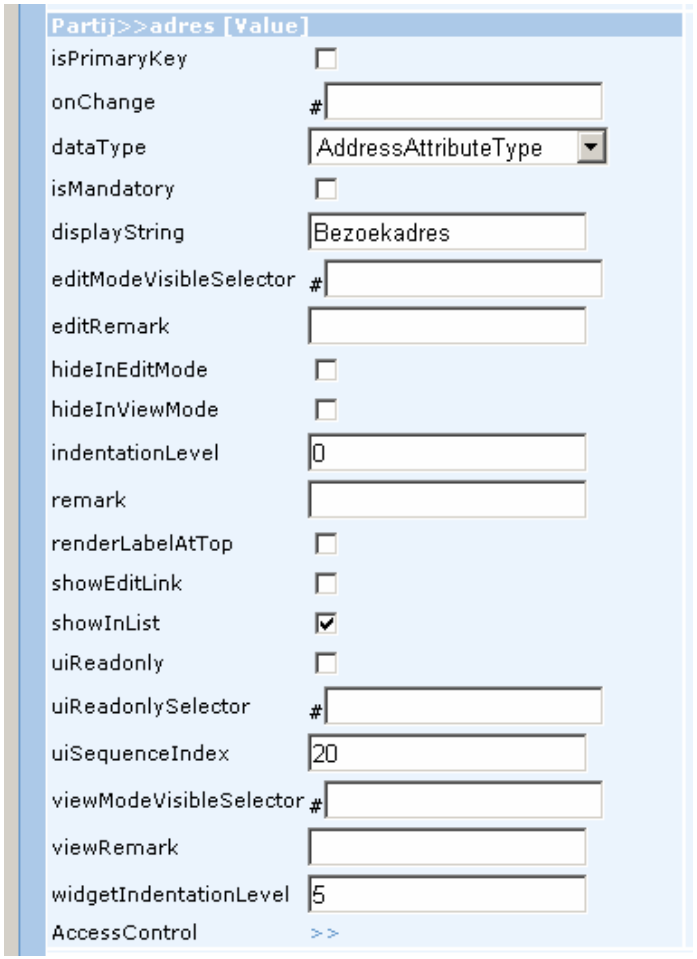
Demonstration

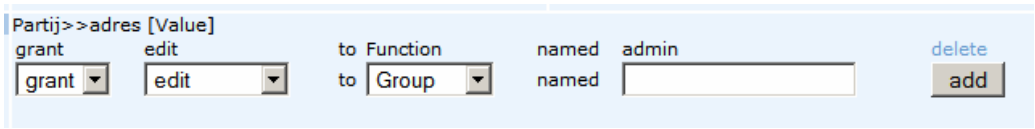
Three versions:

- Version 1: “15 meta tables” kernel + “web based list IDE”. demo, three perspectives (user, administrator, modeler)
- Version 2: “Concrete” Kernel + web based “classic smalltalk” IDE, No demo only some screen shots (see below)
- Version 3: “Syntheses” kernel + web based “sheet” IDE, demo

Version 2
“Classic Smalltalk” IDE




Partij>>adres [Value]
isPrimaryKey
onChange
dataType
AddressAttributeType
isMandatory
displayString
Bezoekadres
editModeVisibleSelector
editRemark
hideInEditMode
hideInViewMode
indentationLevel
0
remark
renderLabelAtTop
showEditLink
showInList
uiReadonly
uiReadonlySelector
uiSequenceIndex
20
viewModeVisibleSelector
viewRemark
widgetIndentationLevel
5
AccessControl
>>


Partij>>adres [Value]
grant
edit
to Function
named
admin
delete
grant
edit
to Group
named
add

References

- “Metadata and active object models” by Brian Foote and Joseph Yoder.
- “Versioning systems for evolution research” Romain Robbes and Michele Lanza.

Questions?

Mathieu at cosmocows.com