Specifically

- Design and implementation of a domain-specific language for building web applications

Generally

- A systematic approach to designing domain-specific languages?

Outline

- 0: The domain-specific language engineering experiment
- 1: Capturing programming patterns
- 2: Scrap your boilerplate
- 3: More Sugar, please!
Part I

Domain-Specific Language Engineering
(Introduction)
Domain-Specific Languages: The Momentum

Many approaches

- Domain-specific languages
- Model-driven architecture
- Software factories
- Language workbenches
- Intentional programming

One goal

- Programming at higher-level of abstraction
- by capturing domain knowledge in language + generator
- Reduce effort of software development and maintenance
- by an order of magnitude
Terminology

Domain

- specialized area of software development
- technical domain (e.g. database management)
- application / business domain (e.g. insurance)

Domain-specific language (DSL)

- a language: a set of well-formed sentences
- concrete syntax may be textual or visual
- has an abstract syntax
- domain-specific: special features / assumptions for domain

Model

- DSL ‘program’

Generator

- translates models to implementations in a general-purpose language (GPL)
Domain-Specific Languages: The Challenge

**Design and implementation**

- Designing domain-specific languages *systematically*
- How do you come up with a new DSL?
- Is there a systematic approach?
- How to keep the implementation small and maintainable?

**Evolution**

- Keep DSL in synch with technology, domain, requirements
- How to make generators portable?
- How to migrate models when DSL is adapted?

**Project: Model-Driven Software Evolution**

- Domain: enterprise software
My DSL Design Experience

**SDF2: syntax definition**
- Incremental improvements to an existing language
- Well developed theory, (some) local expertise
- Parser implemented in plain C

**Stratego: program transformation**
- New language
- Based on six years experience with term rewriting in ASF+SDF
- Inspired by strategies in ELAN
- ATerms for term representation and garbage collection
- Theory on term rewriting not really helpful

**Nix: software deployment**
- New language, inspired by lazy functional programming and
- Research languages for software build management
- Using ATerm library for term representation
- Hardly any theory on software deployment
- Developed by Eelco Dolstra
Some Observations

Domain should be well understood (by someone)

- designing a DSL not a good method for exploring new domain
- purpose of DSL is to make development more productive

Basic technology should be available

- libraries, frameworks, development experience, code base

Abstraction gap

- considerable gain in abstraction should be possible

Common themes

- concise core language capturing essence of domain
- extended with syntactic sugar and desugaring transformations
An Experiment: WebDSLs

Experiment

- Take a new domain: web applications
- Develop a DSL (set of DSLs) for this domain
- Observe elements for a standard process
- (Repeat in the future for other domains)

Experience with domain

- Using web applications: extensive
- Implementation
  - HTML, CSS
  - Maintenance of several wiki-based sites (since 2000?)
  - Tweaking TWiki (Perl)
  - Few experiments with servlets

- In summary: minimal experience
Fixing Some Variables

Contributions of this tutorial

- Experience report
- Introduction to Stratego/XT from an application perspective
- Ideas for systematic DSL development

You should not expect comparisons of

- Techniques and tools for DSL definition
- Visual vs textual languages
- Web programming languages and technologies

Discussions about these topics welcome (off-line)
Part II

Domain Analysis
Scope: what types of web applications?

- Content-management system
- Wiki-like
  - editable via browser
- Rich domain model
  - instead of generic text
  - objects in domain classes
  - generic queries, aggregations, etc.
- Example: web site of a research group
The Software Engineering Research Group

Mission

Software engineering is concerned with methods and techniques for building high quality software systems. This not only includes software construction, but also requirements analysis, design, system integration, testing, deployment, and making changes to software systems after their first release.

The mission of the Delft Software Engineering Research Group is:

1. to develop a deep understanding of how people build and evolve software systems;
2. to develop novel methods, techniques and tools that advance the way in which software is built and adjusted; and
3. to offer students an education that prepares them to take a leading role in complex software development projects.

Research at the Delft Software Engineering Research Group is centered around two themes, software evolution and embedded software, which are studied separately as well as in combination in two laboratories:

- The Software Evolution Research Laboratory (SWERL), and
- The Embedded Software Laboratory (ESL)
2. to develop novel methods, techniques and tools that advance the way in which software is built and adjusted; and

3. to offer students an education that prepares them to take a leading role in complex software development projects.

Research at the Delft Software Engineering Research Group is centered around two themes, software evolution and embedded software, which are studied separately as well as in combination in two laboratories:

- The Software Evolution Research Laboratory (SWERL), and
- The Embedded Software Laboratory (ESL)

News

2007-04-25
Peter Kluin was elected Computer Science Teacher of the Year 2006/2007. The election was organized by the study society Christian Huygens http://ch.tudelft.nl/index.php.

2007-04-20
Arne van Deusen appointed jury member for the Dutch finals of the Imagine Cup in Amsterdam on June 6th, 2007.

2007-04-16
Paper: Understanding Execution Traces Using Massive Sequence and Circular Bundle Views by Bas Cornelissen et al. accepted by International Conference on Program Comprehension. The paper proposes to gain an understanding of software behavior by means of a scalable trace visualization technique. See the ExTraVis homepage for
software engineering groups in the project.

2006-10-15

Felco Visser, formerly at Utrecht University, has been appointed associate professor. He will be continuing his work on program transformation and generation and lead the ModSE and TFA projects. Martin Braeken joins him, for the time being as guest PhD student from Utrecht University.

2006-10-01

Andy Zaidman appointed postdoc in the Reconstructor Project.

Recent Publications

2007


- B. Graaf and A. van Deursen (2007). Visualization of Domain-Specific Modelling Languages
Eelco Visser

News
Looking for postdocs and PhD students in the following project:

- Model-Driven Software Evolution (Jacquard 2006) - 2 postdocs + 2 PhD students

Coordinates

- Associate professor
- Software Engineering Research Group
- Department of Software Technology
- Electrical Engineering, Mathematics and Computer Science (EWI)
- Delft University of Technology
- Delft, The Netherlands (CEST/CET)

- Email: visser@acm.org
- http://www.st.tudelft.nl/~eelco
- http://www.eelcovisser.net
- Blog

Recent Papers

- Model-driven software evolution: A research agenda (McDSE’07)
- Declarative, Formal, and Extensible Syntax Definition for AspectJ (COPSLA’06)
- Stratego/XT 0.16: Components for Transformation Systems (PEPM’06)
- Stratego/XT Manual (documentation)
- Transformations for Abstractions (Keynote SCAM’05)
- Generalized Type-Based Disambiguation of Meta Programs with Concrete Object Syntax (GPCE’05)
TUD-SERG Technical Report Series

Our technical report series, started in 2007, contains preprints of our Scientific Publications. They are listed in reverse chronological order.

2007

<table>
<thead>
<tr>
<th>Report ID</th>
<th>Author(s)</th>
<th>Title</th>
<th>Appeared as</th>
</tr>
</thead>
<tbody>
<tr>
<td>TUD-SERG-2007-011</td>
<td>Alex Feldman, Greg Provan, Arjan van Gemund</td>
<td>On the performance of SAFARI algorithms</td>
<td></td>
</tr>
<tr>
<td>TUD-SERG-2007-010</td>
<td>Marius Marin, Leon Moonen, Arie van Deursen</td>
<td>Documenting Typical Crosscutting Concerns</td>
<td></td>
</tr>
<tr>
<td>TUD-SERG-2007-005</td>
<td>Marius Marin, Leon Moonen, Arie</td>
<td>ScQuTo: Query-Based</td>
<td></td>
</tr>
</tbody>
</table>
Research Colloquium

The SERG group meets (at least) once in the two weeks to learn about and exchange ideas on recent research carried out by the group's researchers (Faculty members, Postdocs, PhD students). Occasionally researchers from other organizations are invited to present their latest work.

**Time and Place**

Thursday, 11:00 - 12:00
Room: 9.130 (EWI)

**Upcoming Presentations**

<table>
<thead>
<tr>
<th>Date</th>
<th>Speaker</th>
<th>Title</th>
<th>Extras</th>
</tr>
</thead>
<tbody>
<tr>
<td>24-05-2007</td>
<td>Celco Visser</td>
<td>Domain-Specific Language Engineering</td>
<td>MoDSE Colloquium, 10:30-12:30, <a href="#">abstract</a></td>
</tr>
<tr>
<td>07-06-2007</td>
<td>Andy Zacimen</td>
<td>On How Developers Test Open Source Software Systems</td>
<td>12:45, <a href="#">abstract</a></td>
</tr>
<tr>
<td>14-06-2007</td>
<td>iba</td>
<td>iba</td>
<td>MoDSE Colloquium in Bordewijkzaal (9.130) 10:30-12:30, <a href="#">abstract</a></td>
</tr>
<tr>
<td>20-06-2007</td>
<td>iba</td>
<td>MoDSE workshop</td>
<td>all day in Sneijderszaal</td>
</tr>
</tbody>
</table>

**See Also**
- Past Presentations
<table>
<thead>
<tr>
<th>Name</th>
<th>Start</th>
<th>End</th>
<th>Supervisor(s)</th>
<th>Site</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zeeser Lubsen</td>
<td>2007</td>
<td>2008</td>
<td>Andy Zaidman</td>
<td>Software Improvement Group, Reconstructor Project</td>
<td>Co-evolution of test and production code</td>
</tr>
<tr>
<td>Jippe Holwerda</td>
<td>2007</td>
<td>2008</td>
<td>Leon Moonen</td>
<td>Compuware</td>
<td>Semi-automatic MDD Remodularization in OptimalJ</td>
</tr>
<tr>
<td>Denny Groenewegen</td>
<td>2006</td>
<td>2007</td>
<td>Elco Visser</td>
<td>TUD</td>
<td>Web-application security</td>
</tr>
<tr>
<td>Jonathan Joubert</td>
<td>2006</td>
<td>2007</td>
<td>Elco Visser</td>
<td>Finalist</td>
<td>Model-driven online development, deployment and maintenance of web applications</td>
</tr>
<tr>
<td>Gerardo Geest</td>
<td>2006</td>
<td>2007</td>
<td>Elco Visser</td>
<td>Avanade</td>
<td>Evolution of DSLs with an application to webservises</td>
</tr>
<tr>
<td>Mulo Emmanuel</td>
<td>2007</td>
<td>2007</td>
<td>Andy Zaidman, Arie van Deursen</td>
<td>Philips Medical Systems</td>
<td>Architectural design for testability</td>
</tr>
<tr>
<td>Xia Chao</td>
<td>2007</td>
<td>2008</td>
<td>Gerd Gross</td>
<td>Imec, Leuven</td>
<td>Development of a new task scheduling component for multimedia platforms</td>
</tr>
<tr>
<td>Justin</td>
<td>2007</td>
<td>2009</td>
<td>Ali Moonen</td>
<td>TORDock</td>
<td>Testing advanced Web interfaces</td>
</tr>
</tbody>
</table>
IntraSE - An Intranet for the Software Engineering Research Group

This web site aims at sharing internal information of the TU Delft Software Engineering Research Group.

Contents of this intranet site:

- Serg Meetings
- Action Points, Including
  - SERG Web Site
  - Calendar
  - Research Colloquium
  - Technical Report Series
- Howto's
- Project Codes
- Research Output
- Teaching Howto's
- Coffee Machine Maintenance Roster
- News-Events-Visitors Policy

Some useful links for (new) TWiki users

- The TWiki homepage which describes the idea behind wiki and this particular instance: TWiki.
- Welcome Guest and Taste of TWiki are two tutorials.
- A sandbox where you can safely play around to test things without messing up anything (NB: everything on this twiki is stored under a revision management system so there no need for worries anyway).
WebHome (edit)

See below for help in editing this page.

- Associate professor
- [http://www.es.wi.tudelft.nl][Software Engineering Research Group]
- [http://www.es.wi.tudelft.nl][Department of Software Technology]
- [http://www.es.wi.tudelft.nl][Electrical Engineering, Mathematics and Computer Science (EEMCS)]
- Delft, The Netherlands
- Email: mailto:visser@emc.org
- http://www.es.wi.tudelft.nl/~eelco
- http://www.eelcovisser.net
- [http://blog.eelcovisser.net][Blog]

---

Your signature for easy copy and paste: -- main.eelcovisser -- 30 Jun 2007

Access keys: G = Cancel, K = Checkpoint, Q = Quiet Save, S = Save, P = Preview

- Release edit lock  help
- Minor changes, don't notify  help

Cancel  Checkpoint  QuietSave  Save  Preview

---

Formatting help:
- **bold** put word/phrase in asterisks: "your phrase"
- *bullet list* 3 spaces, asterisk, 1 space:    * your text
- **headings** 3 dashes, 1 to 6 pluses, 1 space: ------ Your Heading

Done
SERG: Domain Model

- Text
- News
- Conferences
- Publications
- Researchers
- Homepages
- Photos
- Research projects
- Software
- Technical reports
- Courses
- Students
- Thesis projects (workflow!)
- Travel (accounting)
- Meetings
- ...

...
Domain Analysis: Deductive vs Inductive

Deductive (top-down)
- Analyze the problem domain
- Define abstract requirements
- Useful/necessary for new domains
- Risk: may be difficult to implement

Inductive (bottom-up)
- Look at existing applications / frameworks in the domain
- Find common programming patterns
- Define abstractions that capture these patterns
- Risk: abstractions too close to existing practice
- Solution: iterative abstraction
Technology: Deductive vs Inductive

**Deductive (top-down)**

- Start with requirements obtained from domain analysis
- Match to existing technology and/or build your own components
- Advantage: perfect fit for requirements
- Risk: poor reuse, lot of effort, solution incompatible with mainstream technology

**Inductive (bottom-up)**

- What technology (libraries, frameworks) is available?
- How is this technology typically used?
  - e.g. ORM frameworks abstract from DBMS
- Incremental introduction of abstractions
  - Abstract from boilerplate in use of frameworks
  - Quick turn-around time for abstractions:
  - Implementation technique is clear
My Technology Stack

I chose the Java route (or it chose me)

- Java: programming
- Servlets: handling web requests
- JSP: simple presentation layer
- SQL: database management
- JDBC: database connection
- Hibernate: object-relational mapping
- JSF: better presentation layer
- EJB3:
- Seam: integration framework
Universe of Virtual machines

Many alternatives available

- PHP, Ruby/Rails, .Net, ...

Thousands of virtual machines

- each combination of languages, libraries, frameworks constitutes a virtual machine to target in software development
- each enterprise system/application we develop may require a different combination
- similar to situation in embedded systems, where the peculiarities of different hardware architectures have to be dealt with
- if we’re developing the essence of an enterprise system, can we abstract from the details of the different virtual machines?
Part III

Capturing Common Programming Patterns
Tiers

- Presentation layer
  - Java Server Faces (JSF)
- Session beans
  - Java objects that connect presentation and domain objects
- Domain objects
  - store persistent data
  - correspond to 'real-world' concepts

And

- A bit of configuration (XML)
Architecture of Seam/JSF Web Application

```
/editPerson.seam
Eelco Visser
Save

/editPerson.xhtml
EditPersonBean.java

EntityManage em;

public void save(){...}

Person p;

h:form
h:inputText
h:commandButton

#{p.fullname}
#{save()}
```

Table:

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fullname</td>
<td>Eelco Visser</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:visser@acm.org">visser@acm.org</a></td>
</tr>
<tr>
<td>Homepage</td>
<td><a href="http://www.eelcovisser.net">http://www.eelcovisser.net</a></td>
</tr>
<tr>
<td>Photo</td>
<td>/img/eelcovisser.jpg</td>
</tr>
<tr>
<td>Address</td>
<td>------------------</td>
</tr>
<tr>
<td>Street</td>
<td>Makelweg 4</td>
</tr>
<tr>
<td>City</td>
<td>Delft</td>
</tr>
<tr>
<td>Phone</td>
<td>+31 (015) 27 87088</td>
</tr>
<tr>
<td>user</td>
<td>EelcoVisser</td>
</tr>
</tbody>
</table>
Java persistence

- Java 5 annotations for declaration of object-relational mapping
- Vendor independent interface
- Hibernate provides implementation and special purpose annotations
- Entity class corresponds to table in database

Class annotated with `@Entity, empty constructor`

```java
@Entity
public class Publication {

    public Publication () {

        // properties

    }

}```
Entities have an identifier as primary key

```java
@Id @GeneratedValue
private Long id;

public Long getId() {
    return id;
}

private void setId(Long id) {
    this.id = id;
}
```
Properties represent data (columns in database)

private String title;

public String getTitle() {
    return title;
}

public void setTitle(String title) {
    this.title = title;
}
Properties referring to other entities require annotations

@ManyToOne
@JoinColumn(name = "PublicationAuthor")
@Cascade({
    CascadeType.PERSIST,
    CascadeType.SAVE_UPDATE,
    CascadeType.MERGE
})
private Person author = new Person();

public Person getAuthor() {
    return author;
}

public void setAuthor(Person author) {
    this.author = author;
}
The essence of an entity class is simple

- class name
- list of properties, i.e., (name, type) pairs

Example

```java
Publication {  
    title : String  
    author : Person  
    year : Int  
    abstract : String  
    pdf : String  
}

Person {  
    fullname : String  
    email : String  
    homepage : String  
}```
Implementing a DSL

- Definition of concrete syntax
- Parser
- Definition of abstract syntax
- Transformation of models to Java code
Implementing a DSL

- Definition of concrete syntax
  - using the syntax definition formalism SDF
- Parser

- Definition of abstract syntax

- Transformation of models to Java code
Implementing a DSL

- Definition of concrete syntax
  - using the syntax definition formalism SDF

- Parser
  - generate from syntax definition

- Definition of abstract syntax

- Transformation of models to Java code
Implementing a DSL

- Definition of concrete syntax
  - using the syntax definition formalism SDF
- Parser
  - generate from syntax definition
- Definition of abstract syntax
  - generate from syntax definition
- Transformation of models to Java code
Implementing a DSL

- Definition of concrete syntax
  - using the syntax definition formalism SDF
- Parser
  - generate from syntax definition
- Definition of abstract syntax
  - generate from syntax definition
- Transformation of models to Java code
  - implement using term rewrite rules
Implementing a DSL

- Definition of concrete syntax
  - using the syntax definition formalism SDF
- Parser
  - generate from syntax definition
- Definition of abstract syntax
  - generate from syntax definition
- Transformation of models to Java code
  - implement using term rewrite rules
  - use concrete syntax of target language to make rules readable
module DomainModel
exports

lexical syntax
  [a-zA-Z][a-zA-Z0-9_]*  ->  Id
  [0-9]+  ->  Int
  ""~\["\n]*""  ->  String
  [\t\n\r]  ->  LAYOUT
  "//"~\[\n\r]* [\n\r]  ->  LAYOUT

context-free syntax
  Entity  ->  Definition
  Id "{"Property*"}"  ->  Entity  {cons("Entity")}
  Id ":" Sort  ->  Property  {cons("Property")}
  Id  ->  Sort  {cons("SimpleSort")}
Generate abstract syntax definition from syntax definition

signature
  constructors
    SimpleSort : Id -> Sort
    Property   : Id * Sort -> Property
    Entity     : Id * List(Property) -> Entity
    : Entity -> Definition
    : String -> Id

sdf2rtg -i DomainModel.def -m DomainModel -o DomainModel.rtg
rtg2sig -i DomainModel.rtg -o DomainModel.str --module DomainModel
Domain Model: Parsing

Generate parser from syntax definition

`sdf2table -i DomainModel.def -o DomainModel.tbl -m DomainModel`

Parsing gives abstract syntax terms

input: text

```plaintext
Person {
  fullname : String
  email : String
  homepage : String
}
```

output: term

```plaintext
Entity("Person",
  [ Property("fullname", SimpleSort("String"))
  , Property("email", SimpleSort("String"))
  , Property("homepage", SimpleSort("String"))
  ]
)
```

`sglri -p DomainModel.tbl -i publication.dom | pp-aterm`
Concrete syntax

@Entity
public class Publication {
    public Publication () { }
}

Abstract syntax

ClassDec(
    ClassDecHead(
        [MarkerAnno(TypeName(Id("Entity"))), Public()],
        Id("Publication"),
        None(), None(), None(),
    ClassBody(
        [ConstrDec(
            ConstrDecHead([Public()],None(),Id("Publication"),[],None(),
            ConstrBody(None(), []))
        ])
    )
)
entity-to-class:
  Entity(x, prop*) ->
ClassDec(
  ClassDecHead(
    [MarkerAnno(TypeName(Id("Entity"))), Public()]
    , Id(x)
    , None(), None(), None()),
ClassBody(
  [ConstrDec(
    ConstrDecHead([Public()]), None(), Id(x), [], None()),
    ConstrBody(None(), []))
  ]
)
Use concrete syntax of Java in transformation rules

entity-to-class :
  || x_Class { prop* } || ->
  ||
    @Entity
    public class x_Class {
      public x_Class () { }
    }
  ||

Properties

- code fragment is parsed (syntax check)
- transformation produces term representation, not text
- generated code can easily be further transformed
```
entity-to-class:
| [ x_Class { prop* } ] | -> |
|
  @Entity public class x_Class {
  public x_Class () { }

  @Id @GeneratedValue private Long id;

  public Long getId() {
    return id;
  }
  private void setId(Long id) {
    this.id = id;
  }

  ~*cbd*
  }
}|
where cbd* := <mapconcat(property-to-gettersetter)> prop*
```
property-to-gettersetter :
 |[ x_prop : s ]| ->
 |[
  private t x_prop;

  public t x_get() {
    return title;
  }
  public void x_set(t x) {
    this.x = x;
  }
 ]|

where t := <builtin-java-type> s
; x_get := <property-getter> x_prop
; x_set := <property-setter> x_prop

builtin-java-type :
  SimpleSort("String") -> type|
[ String ]|
property-to-property-code(|x_Class) :
  |[ x_prop : s ]| ->
  |
    @ManyToMany
    @Cascade({CascadeType.PERSIST,
                CascadeType.SAVE_UPDATE,
                CascadeType.MERGE})

  private t x_prop;

  public t x_get() { return x_prop; }

  public void x_set(t x_prop) { this.x_prop = x_prop; }

  [x_Class, x_Prop]

where t := <defined-java-type> s
 ; x_Prop := <capitalize-string> x_prop
 ; x_get := <property-getter> x_prop
 ; x_set := <property-setter> x_prop
 ; columnname := <concat-strings>[x_Class, x_Prop]
Propagate declared entities

\[
\text{declare-entity} = \\
\text{?} | [\ x_{\text{Class}} \ \{ \text{prop}^* \} \ ] | \\
; \ \text{rules(} \\
\quad \text{defined-java-type :} \\
\quad \quad \text{SimpleSort}(x_{\text{Class}}) \rightarrow \text{type} | [x_{\text{Class}}] | \\
\text{)}
\]

Dynamic rewrite rules

- add new rewrite rules at run-time
- rules inherit variable bindings from their definition context
- propagate context-sensitive information
- e.g., the Java type for a declared entity
Composing a code generator

```javascript
webdsl-generator =
xtc-io-wrap(webdsl-options,
    parse-webdsl
    ; alltd(declare-entity)
    ; collect(entity-to-class)
    ; output-generated-files
)
```

What it does

- invoke parser to read input
- define dynamic rules for all declared entities
- generate java code for each entity declaration
- pretty-print each generated class to separate file
Recipe

- Find reoccurring programming patterns
- Factor out the repetitive code
- Turn parameters into DSL constructs
- Repetitive code fragments become rhs of rewrite rule
Part IV

Capturing More Programming Patterns
```java
Person {
    fullname : String
    email    : String
    homepage : String
    photo    : String
    address  : Address
    user     : User
}

Address {
    street : String
    city   : String
    phone  : String
}

User {
    username : String
    password : String
    person   : Person
}
```
Architecture of Seam/JSF Web Application
### Java Server Faces: viewPerson.xhtml

```html
<h1> <h:outputText value="#{viewPerson.person.fullname}"/> </h1>
<table>
<tr>
<td> <h:outputText value="Fullname"/> </td>
<td> <h:outputText value="#{viewPerson.person.fullname}"/> </td>
</tr>
<tr>
<td> <h:outputText value="Address"/> </td> <td> </td>
</tr>
<tr>
<td> <h:outputText value="Street"/> </td> <td> <h:outputText value="#{viewPerson.person.address.street}"/> </td>
</tr>
<tr>
<td> <h:outputText value="User"/> </td> <td>
<s:link view="/viewUser.xhtml"
    value="#{viewPerson.person.user.name}" propagation="none">
    <f:param name="user" value="#{viewPerson.person.user.id}"/>
</s:link>
</td>
</tr>
</table>
```
entity-to-xhtml-viewEntity : 
  |[ x_Class { props } ]| -> 
  %>
  <h1>
    <h:outputText value="<%=x_Class%> #{<%=x_class%>.name}"/>
  </h1>
  <table>
    <%= rows :::* %>
  </table>
  <%
  where x_class := <decapitalize-string> x_Class 
    ; rows := <map(row-in-view-form(|x_class|)> props } 
  %>
row-in-view-form(|x_class) : 
prop@[ [ x_prop : s ]] -> 
%
<tr>
  <td> <h:outputText value="<%=x_prop%>"/> </td>
  <td> <%= input %></td>
</tr>
<% where input := <property-to-view-component(|x_class)> prop
Generating JSF Pages: Properties (2)

Output of property

property-to-view-component(|x_class|) :
   |[ x_prop : String ]| ->
   %>
       <h:outputText value="#{<%=x_class%>.<%=x_prop%}>"/>
   %>

Input of property

property-to-edit-component(|x_component|) :
   |[ x_prop : String ]| ->
   %>
       <h:inputText value="#{<%=x_component%>.<%=x_prop%}>"/>
   %>
Seam component can be approached directly from JSF page

```java
@Stateful // can keep state between requests
@Name("viewPerson") // component name
public class ViewPersonBean
    implements ViewPersonBeanInterface
{
    ...

    @Destroy @Remove // required for stateful beans
    public void destroy() { }
}
Seam Session Beans: ViewPersonBean.java

Necessary services are obtained by \textit{injection}

@Logger
private Log log;
// generating log messages

@PersistenceContext(type = EXTENDED)
private EntityManager em;
// interface to the database

@In
private FacesMessages facesMessages;
// generating screen messages

No need to pass parameters or use factories
Domain object made available to JSF via property

```java
private Person person;
public void setPerson(Person person) { this.person = person; }
public Person getPerson() { return person; }
```

Identity of object is passed to page via request parameter

```java
@RequestParameter("person")
private Long personId;
```

Initialization of object based on parameter identifier

```java
@Create @Begin
public void initialize() {
    if (personId == null) {
        person = new Person();
    } else {
        person = em.find(Person.class, personId);
    }
}
```
Generating Session Beans

Replacing names in boilerplate code

entity-to-session-bean :

|[ x_Class { prop* } ]| -> |

@Stateful
@Name("~viewX")

public class x_ViewBean implements x_ViewBeanInterface {
...

@Destroy @Remove
public void destroy() { }
}

where x_ViewBean := ... ; x_ViewBeanInterface := ...
create-local-interface(|x_Interface) :
class ->
  |[
    @Local
    public interface x_Interface {
      ~*methodsdecs
    }
  ]|
  where methodsdecs := <extract-method-signatures> class

extract-method-signatures =
collect(method-dec-to-abstract-method-dec)

method-dec-to-abstract-method-dec :
  |[ mod* t x(arg*) { bstm* } ]| -> |[ mod* t x(arg*); ]|
  where <fetch(?Public())> mod*
Summary

Domain modeling language

- generate entity classes

Generate user interface

- very basic UI for viewing and editing objects

What have we learned?

- understanding of the basics of the technology
- setup of a complete generator

Next

- refining domain modeling language
- consider proper UI
Part V

Refining Programming Patterns
Special types allow to generate refined behaviour

Person {
    fullname : String
    email    : Email
    homepage : URL
    photo    : Image
    address  : Address
    user     : User
}

User {
    username : String
    password : Secret
    person   : Person
}

property-to-edit-component(|x_component) :
    |[ x_prop : Text ]| ->
    %><h:inputTextarea value="#{<%=x_component%>.<%=x_prop%>}"/><% 

property-to-edit-component(|x_component) :
    |[ x_prop : Secret ]| ->
    %><h:inputSecret value="#{<%=x_component%>.<%=x_prop%>}"/><%
Collections

Publication {
    title : String
    authors : List<Person>
    year : Int
    pubabstract : Text
    projects : Set<ResearchProject>
    pdf : URL
}

**Generate Many-to-Many Associations**

property-to-property-code(|x_Class|) :
[ x_prop : List<t> ]| - > |[
    @ManyToMany
    @Cascade(
        CascadeType.PERSIST,
        CascadeType.SAVE_UPDATE,
        CascadeType.MERGE
    )
    private List<t> x_prop = new LinkedList<t>();
] |
Refining Associations

Value Types

• title :: String

Composite Associations

• address <> Address

Reference Associations

• authors -> List<Person>

Publication { 
  title :: String
  authors -> List<Person>
  year :: Int
  pubabstract :: Text
  projects -> Set<ResearchProject>
  pdf :: URL
}

Person { 
  fullname :: String
  email :: Email
  homepage :: URL
  photo :: Image
  address <> Address
  user -> User
}
Generating JSF Pages: Unfolding Entities

Person {
  fullname :: String
  email :: Email
  homepage :: URL
  photo :: Image
  address <> Address
  user -> User
}

Address {
  street :: String
  city :: String
}

User {
  username :: String
  password :: Secret
  person -> Person
}
row-in-edit-form(|x_component) : 
|[ x_prop <> s ]| -> 
%
<h:outputText value="<%=x_prop%>"/>
<%= row* :::*%>
<%
where <defined-java-type> s
 ; prop* := <properties> s
 ; x_sub_comp := <concat-strings>[x_component,".",x_prop]
 ; row* := <edit-form-rows(|x_sub_compt)> prop*
Recipe

- Turn programming patterns into generator rules

DSL

- Language for domain models
- With refinements to support sophisticated crud operations
- Generate entity classes, session beans, and JSF pages

Techniques

- Declarative syntax definition
- Term rewriting with concrete syntax

Next

- Scrap your boilertemplate: refactoring the generator
Part VI

Scrap your Boilertemplate™
Generating CRUD pages from domain models

Person {
  fullname :: String
  email :: Email
  homepage :: URL
  photo :: Image
  address <> Address
  user -> User
}

Address {
  street :: String
  city :: String
}

User {
  username :: String
  password :: Secret
  person -> Person
}
For each entity generate several types of artifacts

- Entity class
- View page
- Edit page
- Page with all objects
- Search page
- ...

For each page generate

- JSF file
- Java session bean
- Local interface of session bean
But you don’t want that!

**Limited expressivity**

- Adding new type of page requires extending the generator

**Code duplication in templates**

- Templates are large
- Similar coding patterns are used in different templates
- Only a complete page type is considered as a reusable pattern

**Time for template refactoring**

- Intermediate language for defining presentations

‘This does not look like a compiler’

Eelco Visser to Martin Bravenboer
Granularity and Expressivity

Scale of granularity and expressivity

- Maximal expressivity/reuse, minimal flexibility
  - few constructs from which a complete application is 'generated'
  - coarse grained language provides good reuse
  - large chunk of code can be reused at once
  - provides (too) little flexibility

- Maximal flexibility, minimal expressivity/reuse
  - 1 construct corresponds to 1 GPL construct
  - the language now mimicks the GPL and no productivity gains are to be expected

Find a balance between these extremes
Language for defining page presentation and flow

• Derive from one definition
  • the JSF presentation
  • the Java implementation of the session beans

• Inspiration: \LaTeX
  • \TeX provides basic machinery for typesetting
  • \LaTeX provides abstractions for structuring documents
  • \LaTeX philosophy: separate layout from content
  • Advantage over XML/HTML: user-definable abstraction mechanism (macros)
ResearchGroup {
    acronym :: String (name)
    fullname :: String
    mission :: Text
    logo :: Image
    members -> Set<Person>
    projects -> Set<ResearchProject>
    colloquia -> Set<Colloquium>
    news -> List<News>
}

Page Flow

Page definition

define page viewResearchGroup(group : ResearchGroup) {
  <presentation>
}

→ URL

/viewResearchGroup.seam?group=1

Page navigation

navigate(pers.group.acronym,viewResearchGroup(pers.group))

→ Link

<a href="/viewResearchGroup.seam?group=1">SERG</a>
MetaProgramming Lab

Mission
To do cool meta programming stuff.

Recent Publications
- Transformations for Abstractions
- Model-Driven Software Evolution: A Research Agenda
- Domain-Specific Language Engineering
- Grammar Engineering Support for Precedence Rule Recovery and Compatibility Checking
- Preventing Injection Attacks with Syntax Embeddings

People
- Martin Bravenboer
- Eelco Visser
define page viewResearchGroup(group : ResearchGroup) {
    section {
        header{text(group.fullname)}
        section {
            header{"Mission"}
            outputText(group.mission)
        }
    }
    section {
        header{"Recent Publications"}
        list { ... }
    }
    section {
        header{"People"}
        list { for(p : Person in group.members) {
            listitem { navigate(p.name, viewPerson(p)) }
        } }
    }
}
**MetaProgramming Lab**

**Mission**
To do cool meta programming stuff.

**Recent Publications**
- Transformations for Abstractions
- Model-Driven Software Evolution: A Research Agenda
- Domain-Specific Language Engineering
define page viewResearchGroup(group : ResearchGroup) {
    block("outersidebar"){
        block("logo"){
            ... 
        }
        block("sidebar"){
            ... 
        }
    }
    block("outerbody"){
        block("menubar"){
            block("menu"){
                ... 
            }
        }
        block("body"){
            section {
                header{
                    text(group.fullname)
                }
                ... 
            }
        }
    }
}
Composing Presentations: Page Layout with CSS

```css
.outersidebar {
position : absolute;
overflow : hidden;
top : 0px;
left : 10px;
margin-top : 10px;
width : 10em;
}

.logo {
text-align : left;
}

.sidebar {
    top : 0px;
    margin-top : 20px;
    color : darkblue;
    border-right : 1px dotted;
}

.outerbody {
position : absolute;
top : 10px;
left : 12.5em;
right : 40px;
}

.menubar {
    height : 62px;
    border-bottom : 1px dotted;
    color : darkblue;
}

.body {
    position : relative;
top : 20px;
margin-bottom : 2.5em;
}
```
MetaProgramming Lab

Mission
To do cool meta programming stuff.

Recent Publications
- Transformations for Abstractions
- Model-Driven Software Evolution: A Research Agenda
- Domain-Specific Language Engineering
Composинг Presentations: Sidebar is just a list

def block("sidebar"){
    list {
        listitem {
            navigate(group.acronym, viewResearchGroup(group))
        }
        listitem{
            navigate("People", groupMembers(group))
        }
        listitem{
            navigate("Publications", groupPublications(group))
        }
        listitem{
            navigate("Projects", groupProjects(group))
            list{ for( p : ResearchProject in group.projectsList ) {
                listitem{ navigate(p.name, viewResearchProject(p)) }
            } }
        }
    }
    ...
}
}
.sidebar ul {
    list-style: none;
    margin: 0em;
    padding: 0px;
}

.sidebar ul li {
    margin: 0em;
    padding: 0px;
}

.sidebar ul ul {
    list-style-type: square;
    font-size: .8em;
    padding: .2em;
    margin-left: 1em;
}

Composing Presentations: Drop Down Menus

![Diagram of a drop-down menu with options for projects and management, along with a mission statement and recent publications.](image.png)
block("menu") { 
  list{
    listitem{
      "People"
      list{ for(person : Person) {
          listitem{ navigate(person.name, viewPerson(person)) } 
      } 
    }
  }
}
list {
listitem {
  "Projects"
  list { for(p : ResearchProject) {
      listitem { navigate(p.acronym, viewResearchProject(p)) } 
    } 
  }
}
...}
div.menu ul ul,
div.menu ul li:hover ul ul,
div.menu ul ul li:hover ul ul,
div.menu ul li table
{
    display: none;
}

div.menu ul li:hover ul,
div.menu ul ul li:hover ul,
div.menu ul ul ul li:hover ul,
div.menu ul li:hover table
{
    display : block;
    width    : 9em;
    border   : ...;
    background-color : white;
}

Current elements provide basics

- CSS goes a long way
- AJAX/JavaScript is complementary
  - map to appropriate JSF tag library (e.g. richfaces)
  - keep abstractions general
Presentation Language Constructs: Template Call

**Template Call**

- concrete syntax
  \[ f(e_1,\ldots,e_m) \{ e_{el1} \ldots e_{elm} \} \]
- abstract syntax
  \[ \text{TemplateCall}(f, [e_1,\ldots,e_m], [e_{el1}, \ldots, e_{elm}]) \]
- expression and element argument lists are optional

**Examples**

- `block("menu") \{ \ldots \}`
- `section \{ header{ \ldots } \ldots \}`
- `list \{ listitem \{ \ldots \} \ldots \}`
- `table \{ row{ \ldots } row{ \ldots } \}`
- `text(group.name)`
- `navigate(pub.name, viewPublication(pub))`
Presentation Language Constructs: Iteration

Iteration

- concrete syntax
  
  ```
  for( x : sort in e ) { elem* }
  ```

- abstract syntax
  
  ```
  For(x, sort, e, elem*)
  ```

Example

- list 
  
  ```
  list {
      for(p : ResearchProject in pers.groups) {
          listitem {
              navigate(p.acronym, viewResearchProject(p))
          }
      }
  }
  ```
Page to JSF

- presentation elements to JSF components
- object access expressions to JSF EL expressions

Page to Seam Session Bean

- connect JSF page to entity objects
- properties for page arguments
- datamodels for iteration
Mapping Pages to JSF+Seam

User { name :: String }
page viewUser(user : User) {
    text(user.name)
}

@Stateful @Name("viewUser")
class viewUserBean {
    @PersistenceContext
    EntityManager em;
    @RequestParameter("user")
    private Long userId;
    property User user;
    @Begin @Create
    public void initialize() {
        user =
            em.find(User.class,userId)
    }
}

<html ...> ...
<body>
    <h:outputText value="#{viewUser.user.name}"/>
</body>
</html>
Basic element

elem-to-xhtml :
  Text(x)  ->  %> <h:outputText value="<%=x%>"/> <%

Recursive call

elem-to-xhtml :
  |[ block(str){ elem* } ]|  -> %>
    <div class="<%= str %>">
      <%= <elems-to-xhtml> elems* :::*%>
    </div>
  <%
navigate(viewPerson(p)) { text(p.name) }

<s:link view="/viewPerson.xhtml">
    <f:param> name="person" value="#{p.id}" />
    <h:outputText value="#{p.name}" />
</s:link>

elem-to-xhtml :
    |[ navigate(p(args)) { elems1 } ]| ->
    %> <s:link view="/<%= p %>.xhtml"><%= conc((params, elems2) ::*)
    %></s:link> <%
    where <IsPage> p
        ; fargs := <TemplateArguments> p
        ; params := <zip(bind-param)> (fargs, args)
        ; elems2 := <elems-to-xhtml> elems1
    bind-param :
        (|[ x : s ]|, e) ->
        %><f:param name="<%= x %>"> value="<%= el %>"> /</%<%
        where <defined-java-type> s
            ; el := <arg-to-value-string> |[ e.id ]|
list{ for ( project : ResearchProject ) {
   listitem { navigate(project.acronym,viewResearchProject(project)) } }
}

<ul>
  <ui:repeat var="project" value="#{viewResearchGroup.group.projectsList}"
    value="#{viewResearchGroup.group.projectsList}">
    <li>
      <s:link view="/viewResearchProject.xhtml">
        <f:param name="researchProject" value="#{project.id}"/>
        <h:outputText value="#{project.name}"/>
      </s:link>
    </li>
  </ui:repeat>
</ul>

elem-to-xhtml :
    |\[ for(x : s in e){elem*} ]| ->
    %>
    <ui:repeat var="<%= x %>"> value="<%= el %>">
      <%= <elems-to-xhtml> elem* ::*%>
    </ui:repeat>
    <%
    where el := <arg-to-value-string> e
Mapping Presentation Elements to JSF: Nested Sections

section{
  header{"Foo"} ...
  section{ header{"Bar"} ... }
}

<h1>Foo</h1> ...
<h2>Bar</h2> ...

elem-to-xhtml :
  |[ section{elems1} ]| -> elems2
  where {
  | SectionDepth
    : rules( SectionDepth := <SectionDepth; inc> )
    ; elems2 := <elems-to-xhtml> elems1
  |}

elem-to-xhtml :
  |[ header{elems} ]| ->
  %>
  <~n:tag><%= <elems-to-xhtml> elems ::*%></~n:tag>
  <%
  where n := <SectionDepth>
    ; tag := <concat-strings>["h", <int-to-string> n]
User { name :: String }  
page viewUser(user : User) {
    text(user.name)
}

@Stateful @Name("viewUser")
class viewUserBean {
    @PersistenceContext
    EntityManager em;
    @RequestParameter("user")
    private Long userId;
    property User user;
    @Create @Begin
    public void initialize() {
        user =
            em.find(User.class,userId)
    }
}

<html ...> ...
<body>
    <h:outputText value="#{viewUser.user.name}"/>
</body>
</html>
page-to-java :
def@[ define page x_page(args){elems1} ]| ->
compilation-unit|[  
@Stateful @Name("~x_page")
  public class x_PageBean implements x_PageBeanInterface {

    @PersistenceContext private EntityManager em;

    @Create @Begin public void initialize() { bstm* }  

    @Destroy @Remove public void destroy() {}  

    ~*cbd*
  }
]

where x_Page := <capitalize-string> x_page
; x_PageBean := <concat-strings> [x_Page, "Bean"]
; cbd* := <collect(page-elem-to-method)> def
; bstm* := <collect(page-elem-to-init)> def
Mapping Pages to Seam: Page Arguments

argument-to-bean-property :
| [ x : x_Class ] | ->
| [ @RequestParameter("~x") private Long x_Id; 
  private x_Class x;
  public void x_set(x_Class x) { this.x = x; }
  public x_Class x_get() { return x; } |
where x_Id := <concat-strings>[x, "Id"]

argument-to-initialization :
| [ x : x_Class ] | ->
| [ if (x_Id == null) { x = new x_Class(); } 
  else { x = em.find(x_Class.class, x_Id); } |
where x_Id := <concat-strings>[x, "Id"]
Now that looks more like a compiler!

- language constructs that do one thing
- translation rules with (mostly) small right-hand sides

Next

- Extensions
  - completing the core language with
  - typechecking, actions, queries

- Not all abstraction can be generative
  - abstraction mechanisms for the application developer
  - templates, modules

- More sugar, please
  - enriching the DSL with higher level abstractions
  - implemented using desugarings (model-to-model transformations)
Part VIII

Demonstration
Part IX

Extensions
Typechecking

**JSF**
- JSF pages ’compiled’ at run-time
- Many causes of errors unchecked
  - Missing or non-supported tags
  - References to non-existing properties
  - References to non-existing components
- Cause run-time exceptions

**Seam**
- Seam component annotations scanned at deployment-time
- Method not declared in @Local interface not found (silent)

**WebDSL**
- WebDSL programs are statically typechecked
- Typechecker annotates expressions with their type, which is key to type-based desugarings
Typechecking: Example

User {
    name :: String
}
define page viewUser(user : User) {
    text(user.fullname)
    text(us.name)
}

$ dsl-to-seam -i test.app
[error] definition viewUser/text/: expression 'user.fullname' has type error
[error] definition viewUser/text/: variable 'us' has no declared type

(error messages are not quite as pretty yet)
typecheck-iterator:
  For(x, s, e1, elems1) -> For(x, s, e2, elems2)
where in-tc-context(id
    ; e2 := <typecheck-expression> e1
    ; <should-have-list-type> e2
    ; {\| TypeOf
        : if not(<java-type> s) then
            typecheck-error(\[
              "index ", x, " has invalid type ", s
            ])
        else
            rules( TypeOf : x -> s )
        end
    ; elems2 := <typecheck-page-elements> elems1
  |}
  | ["iterator ", x, "/"] )
Edit Person Eelco Visser

Fullname: Eelco Visser
Email: visser@acm.org
Homepage: http://www.eelcovisser.net
Photo: /img/eelcovisser.jpg
Address:
Street: Mekelweg 4
City: Delft
Phone: +31 (015) 27 87088
User: EelcoVisser
Blog: Transformations and Abstractions

Save  Cancel
User { name :: String }

page editUser(user : User) {
    form{
        inputString(user.name)
        action("Save", save())
        action save() {
            user.save();
            return viewUser(user);
        }
    }
}

@Stateful @Name("editUser")
class viewUserBean {
    property User user;
    @End public String save() {
        em.persist(this.getUser());
        return "/viewUser.seam" + "+?user=" + user.getId();
    }
}

<h:form>
    <h:inputText value="#{editUser.user.username}"/>
    <h:commandButton type="submit" value="Save"
        action="#{editUser.save()}"/>
</h:form>
Action Language Constructs

Expressions

- Object creation: Person{ name := e ... }
- Set creation: { e1, e2, ... }
- List creation: [ e1, e2, ... ]
- Variables, constants, field access

Statements

- Assignment: person.blog := Blog{ title := name };
- Method call: publication.authors.remove(author);
- Return: return viewUser(u); (page-flow)

Embed Java (subset)?

+ solid syntax and semantics
- no control over what is used
- no translation to other platforms
- typechecking and other analyses much harder (reuse dryad?)
Create new Person

Fullname
Email
Homepage
Photo
Address
Street
City
Phone
User
Blog

Save  Cancel

generated with Stratego/XT
Page Local Variables

User { name :: String }

page createUser() {
  var user : User := User{};
  form{
    inputString(user.name)
    action("Save", save())
    action save() {
      user.save();
      return viewUser(user);
    }
  }
}

@Stateful @Name("editUser")
class viewUserBean {
  property User user;
  @Create @Begin
  public void initialize() {
    user = new User();
  }
  @End
  public String save() {
    em.persist(this.getUser());
    return "/viewUser.seam" + "?user=" + user.getId();
  }
}

<h:form>
  <h:inputText value="#{createUser.user.username}"/>
  <h:commandButton type="submit" value="Save"
    action="#{createUser.save()}"/>
</h:form>
Martin Bravenboer

Coordinates

- homepage: http://martin.bravenboer.name
- email: martin.bravenboer@gmail.com
- address: Mekelweg 4
- Delft
- phone: 015

Publications

- Preventing Injection Attacks with Syntax Embeddings (2007)
User{ name :: String } Publication{ authors -> List<User> }

page viewUser(user : User) {
  var pubs : List<Publication> :=
  select pub from Publication as pub, User as u
  where (u.id = ~user.id) and (u member of pub.authors)
  order by pub.year descending;
  for(p : Publication in pubs) { ... } 
}

class viewUserBean {
  property List<Publication> pubs;
  @Factory("pubs") public void initPubs() {
    pubs = em.createQuery("select pub from Publication as pub, User as u
    where (u.id = :param1) and (u member of pub.authors)
    order by pub.year descending"
).setParameter("param1", this.getUser().getId()).getResultList();
  }
}
Syntax

• Hibernate queries are composed as strings and parsed at run-time
• In WebDSL query is parsed by the generator
  • Syntax of HQL is embedded in syntax of WebDSL
  • Generated HQL pretty-printer is used to 'generate' queries in Java code

Typechecking

• Hibernate queries are typechecked at run-time
• In WebDSL query is checked against entity declarations and local variables used as parameters (under construction)
Part X

Not all abstraction can be generative
Protoyping experiment

- Expanding the SERG webapplication on demand
- Think of a domain model to add to the website (what properties?)
- Think of the presentation and editing interface
Application programmer needs abstraction mechanisms

- Naming reusable fragments
- Avoiding code duplication
- Building a library

Templates

- Named pieces of code with parameters and hooks

Modules

- Organization of code base
- Library of reusable code
Consider viewBlog

Transformations and Abstractions

WebDSL rocks!

but textareas should be a tad larger ... and now they are! It is even possible to include wiki style markup in text. For instance, if I include a text between asterixes, as in foo, it should end up as bold text. But why do I get these strike through texts?

Ok, I don't get them anymore. It is also possible to define lists

1. first item
2. second item

read more ...

Global Variables

During on of our chats on current affairs, Martin mentioned that Lennart Kats had proposed to introduce global variables in Stratego. My first reaction was of course outrage. My second reaction was to immediately add it to the compiler. The proposal was not to add some sort of C style global variables, but rather to provide better syntax for a programming pattern that was already well established (although considered somewhat improper, at least by me).

read more ...

Model-Driven Software Evolution: A Research Agenda

Software systems need to evolve, and systems built using model driven approaches are no exception. What complicates model driven engineering is that it requires...
Blog Domain Model

```plaintext
Blog {
    title      :: String (name)
    author     -> Person
    entries    <> List<BlogEntry>
    categories -> List<Category>
}

BlogEntry {
    blog       -> Blog
    title      :: String (name)
    created    :: Date
    category   -> Category
    intro      :: Text
    body       :: Text
    comments   <> List<BlogComment>
}
```
### Some numbers about viewBlog

<table>
<thead>
<tr>
<th>file name</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blog + BlogEntry</td>
<td>16</td>
</tr>
<tr>
<td>BlogEntry.java</td>
<td>116</td>
</tr>
<tr>
<td>Blog.java</td>
<td>85</td>
</tr>
<tr>
<td>generated : source</td>
<td>201 : 16 = 12.5</td>
</tr>
<tr>
<td>viewBlog.app</td>
<td>91</td>
</tr>
<tr>
<td>viewBlog.xhtml</td>
<td>164</td>
</tr>
<tr>
<td>ViewBlogBeanInterface.java</td>
<td>32</td>
</tr>
<tr>
<td>ViewBlogBean.java</td>
<td>131</td>
</tr>
<tr>
<td>generated : source</td>
<td>327 : 91 = 3.6</td>
</tr>
</tbody>
</table>
## Some numbers about SERG application

<table>
<thead>
<tr>
<th>file name</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>serg.app</td>
<td>983</td>
</tr>
<tr>
<td>*.xhtml</td>
<td>17329</td>
</tr>
<tr>
<td>*.java (ent.)</td>
<td>1848</td>
</tr>
<tr>
<td>*BeanInterface.java</td>
<td>4069</td>
</tr>
<tr>
<td>*Bean.java</td>
<td>15588</td>
</tr>
<tr>
<td>generated java</td>
<td>21505</td>
</tr>
<tr>
<td>generated total</td>
<td>38834</td>
</tr>
<tr>
<td>generated : source</td>
<td>39.5</td>
</tr>
</tbody>
</table>
Some numbers about SERG application

<table>
<thead>
<tr>
<th>file name</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>serg.app</td>
<td>983</td>
</tr>
<tr>
<td>*.xhtml</td>
<td>17329</td>
</tr>
<tr>
<td>*.java (ent.)</td>
<td>1848</td>
</tr>
<tr>
<td>*BeanInterface.java</td>
<td>4069</td>
</tr>
<tr>
<td>*Bean.java</td>
<td>15588</td>
</tr>
<tr>
<td>generated java</td>
<td>21505</td>
</tr>
<tr>
<td>generated total</td>
<td>38834</td>
</tr>
<tr>
<td>generated : source</td>
<td>39.5</td>
</tr>
<tr>
<td>serg-full.app</td>
<td>9165</td>
</tr>
<tr>
<td>generated : source</td>
<td>4.2</td>
</tr>
</tbody>
</table>
### Some numbers about SERG application (revisited)

<table>
<thead>
<tr>
<th>file name</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>serg.app</td>
<td>983</td>
</tr>
<tr>
<td>serg-full.app</td>
<td>9165</td>
</tr>
<tr>
<td>generated : source</td>
<td>9.3</td>
</tr>
<tr>
<td>generated total</td>
<td>38834</td>
</tr>
<tr>
<td>generated : source</td>
<td>4.2</td>
</tr>
</tbody>
</table>
Some numbers about SERG application (revisited)

<table>
<thead>
<tr>
<th>file name</th>
<th>LOC</th>
</tr>
</thead>
<tbody>
<tr>
<td>serg.app</td>
<td>983</td>
</tr>
<tr>
<td>serg-full.app</td>
<td>9165</td>
</tr>
<tr>
<td>generated : source</td>
<td>9.3</td>
</tr>
<tr>
<td>generated total</td>
<td>38834</td>
</tr>
<tr>
<td>generated : source</td>
<td>4.2</td>
</tr>
</tbody>
</table>

**Basic WebDSL**

- Reduce code size to 25%

**WebDSL with model-to-model transformations**

- Reduce code size to 2.5%
- By means of template expansion and desugaring

**Note**

- These numbers are not definitive; full blown application will require more DSL code
Define a fragment once

```plaintext
define logo() {
    navigate(home()){image("/img/serg-logo-color-smaller.png")}
}
define footer() {
    "generated with 
    navigate("Stratego/XT", url("http://www.strategoxt.org"))
}
define menu() {
    list{ listitem { "People" ... } } ...
}
```

Reuse fragment in many pages

```plaintext
define page home() {
    block("menubar"){ logo() menu() }
    section{ ... }
    footer()
}
```
Templates with Hooks

Template definition calls other templates

```plaintext
define main() {
    block("outersidebar") { logo() sidebar() }
    block("outerbody") {
        block("menubar") { menu() }
        body()
        footer()
    }
}
```

(Re)define hook templates locally

```plaintext
define page viewBlog(blog : Blog) {
    main()
    define sidebar(){ blogSidebar(blog) }
    define body() {
        section{ header{ text(blog.title) }
            for(entry : BlogEntry in blog.entries) { ... }
        }
    }
}
```
Templates with Entity Parameters

Pass objects to template definitions

define personSidebar(p : Person) {
    list {
        listitem { navigate(p.name, viewPerson(p)) }
        listitem { navigate("Publications", personPublications(p)) }
        listitem { navigate("Blog", viewBlog(p.blog)) blogEntries() }
        listitem { "Projects" listProjectAcronyms(p) }
    }
}

Reuse same template in different contexts

define page viewPerson(person : Person) {
    main()
    define sidebar() { personSidebar(person) } ...
}
define page personPublications(person : Person) {
    main()
    define sidebar() { personSidebar(person) } ...
}
declare-template-definition =
  ?def@| [ define mod* x(farg*){elem*} ] |
  ; rules( TemplateDef : x -> def )

expand-template-call :
  |[ x(e*){elem1*} ]| -> |[ block(str){elem2*} ]|
  where <TemplateDef;rename>x => |[define mod* x(farg*){elem3*}]|
    ; { | Subst
      : <zip(bind-variable)> (farg*, <alltd(Subst)> e*)
      ; elem2* := <map(expand-element)> elem3*
      ; str := x
    |}

bind-variable =
  ?(Arg(x, s), e); rules( Subst : Var(x) -> e )
define page viewBlog(blog : Blog) {
    main()
    define sidebar(){ ... }
    define body() { ... }
}

Expands to

define page viewBlog(blog : Blog) {
    block("main"){
        block("outersidebar") {
            block("logo"){ ... } block("sidebar"){ ... }
        }
        block("outerbody") {
            block("menubar") { block("menu") { ... } }
            block("body") { ... } block("footer") { }
        }
    }
}

Trail of template expansion can be used in stylesheets
module publications
section domain definition.

Publication {
    title :: String (name)
    subtitle :: String
    year :: Int
    pdf :: URL
    authors -> List<Person>
    abstract :: Text
    projects -> Set<ResearchProject>
}

section presenting publications.

define showPublication(pub : Publication) {
    for(author : Person in pub.authors){
        navigate(author.name, viewPerson(author)) "", " 
    }
    navigate(pub.name, viewPublication(pub)) ", 
    text(pub.year) "."
}
Module Imports

application org.webdsl.serg

description
   This application organizes information relevant for a research group, including people, publications, students, projects, colloquia, etc.
end

imports app/templates
imports app/people
imports app/access
imports app/blog
imports app/colloquium
imports app/publications
imports app/projects
imports app/groups
imports app/news
imports app/issues
A simple module systems costs as little as 11 LOC

import-modules =
    topdown(try(already-imported <+ import-module))

already-imported :
    Imports(name) -> Section(name, [])
    where <Imported> name

import-module :
    Imports(name) -> mod
    where mod := <xtc-parse-webdsl-module>FILE(<concat-strings>[name, "
    ; rules( Imported : name )

But then you don’t get separate compilation
Part XI

More Sugar, Please!
Higher-Level Language Constructs aka Syntactic Sugar

- An assessment of WebDSL
  - flexibility
    - some patterns tedious to encode

- Solution
  - identify common patterns
  - define higher-level constructs (syntactic sugar)
  - implement using desugaring transformation rules
  - aka model-to-model transformations

- Examples
  - links to entities
  - editing associations
  - edit pages
Transformations for Abstractions

Title: Transformations for Abstractions
Submit:
Year: 2005

Authors
- Eelco Visser

Abstract
The transformation language Stratego provides high-level abstractions for implementation of a wide range of transformations. Our aim is to integrate transformation in the software development process and make it available to programmers. This requires the transformations provided by the programming environment to be extensible. This paper presents a case study in the implementation of extensible programming environments using Stratego, by developing a small collection of language extensions and several typical transformations for these languages.
Pattern

\[ \text{navigate(viewPublication(pub))}\{\text{text(pub.name)}\} \]

Abstraction

\[ \text{output(pub)} \]

Desugaring rule

\[
\text{DeriveOutputSimpleRefAssociation : } \\
\quad |[ \text{output(e)}\{\} ]| \rightarrow |[ \text{navigate($viewY(e))}\{\text{text(e.name)}\} ]| \\
\text{where SimpleSort($Y) := <type-of> e} \\
\quad ; <\text{defined-java-type}> \text{SimpleSort($Y)} \\
\quad ; $viewY := <\text{concat-strings}>["view", Y]
\]

Enabled by type annotations on expressions
Similar desugaring rules

DeriveOutputText:
\[ [ \text{output}(e)\{\} ] \mapsto [ \text{navigate}(\text{url}(e))\{\text{text}(e)\} ] \]
where SimpleSort("URL") := \text{type-of} e

DeriveOutputText:
\[ [ \text{output}(e)\{\} ] \mapsto [ \text{image}(e)\{\} ] \]
where SimpleSort("Image") := \text{type-of} e

Consequence

- \text{output}(e) \text{ sufficient for producing presentation}
Edit Publication Transformations for Abstractions

Title: Transformations for Abstractions
Subtitle: 
Authors: Eelco Visser

Year: 
Abstract: Language Stratego provides high-level implementation of a wide range of transformations. The aim is to integrate transformation in the development process and make it available to users. This paper...
Input: Editing Entity Collection Associations

Ingredients

- List of names of entities already in collection
- Link to remove entity from collection [X]
- Select menu to add a new (existing) entity to collection

Pattern

```java
list { for(person : Person in publication.authors) {
    listitem{ text(person.name) " "
        actionLink("[X]", removePerson(person)) }
} }
select(person : Person, addPerson(person))

action removePerson(person : Person) {
    publication.authors.remove(person);
}
action addPerson(person : Person) {
    publication.authors.add(person);
}
```
Desugaring rule

```
DeriveInputAssociationList :
  elem|[ input(e){} ]| ->
  elem|[ 
    block("inputAssociationList"){
      list { for(x : $X in e){ listitem {
        text(x.name) " "
        actionLink("[X]", $removeX(x))
        action $removeX(x : $X) { e.remove(x); } 
      } } }
      select(x1 : $X, $addX(x1))
      action $addX(x : $X) { e.add(x); }
    }]

  where |[ List<$X> ]| := <type-of> e
    ; x        := <decapitalize-string; newname> $X
    ; x1      := <decapitalize-string; newname> $X
    ; $viewX   := <concat-strings>"["view", $X]
    ; $removeX := <concat-strings; newname>"["remove", $X]
    ; $addX    := <concat-strings; newname>"["add", $X]
```
Similar desugaring rules

DeriveInputText :
  \[ \text{input}(e) \rightarrow \text{inputText}(e) \]
  where \text{SimpleSort}("Text") := \text{type-of} e

DeriveInputSecret :
  \[ \text{input}(e) \rightarrow \text{inputSecret}(e) \]
  where \text{SimpleSort}("Secret") := \text{type-of} e

Consequence

- \text{input}(x.y.z) \text{ suffices for producing input of property}
Edit BlogEntry Global Variables

Blog: Transformations and Abstractions
Title: Global Variables
Created: 26/04/2007
Category: 
Intro: During our recent discussions on current topics, Martin mentioned that Lennart Kats had proposed to introduce global variables in Stratego. My initial reaction was one of concern. My second reaction was to immediately add it to the compiler. The proposal was not to add some sort of C-style global variables, but rather to provide better syntax for a programming pattern that was already well established (although considered somewhat improper, at least by me).
Edit Page for Entity

Ingredients

- Input box for each property of an entity organized in a table
- Save and Cancel buttons

Pattern

```javascript
form {
    table {
        row{ "Blog" input(entry.blog) }
        row{ "Title" input(entry.title) }
        row{ "Created" input(entry.created) }
        row{ "Category" input(entry.category) }
        row{ "Intro" input(entry.intro) }
        row{ "Body" input(entry.body) }
    }
    action("Save", save()) action("Cancel", cancel())
    action cancel() { return viewBlogEntry(entry); }
    action save() { entry.save(); return viewBlogEntry(entry); }
}
```
Desugaring rules

entity-to-edit-form :  
| [ $X : $Y { prop* } ] | ->  
| [ 
form {
    table { elem* }
    action("Save", save())
    action("Cancel", cancel())
}
action cancel() { return $viewX(x); }
action save() { x.save(); return $viewX(x); }
]|  
where $viewX := <concat-strings>["view", $X]
; x := <decapitalize-string> $X
; str := $X
; elem* := <map(property-to-edit-row(|x))> prop*

property-to-edit-row(|x) :  
| [ y k s (anno*) ] | ->  | [ row { str input(x.y) } ] |
where str := <capitalize-string> y
DSL Design: Balance between Salt and Sugar

Salt (core language)

- low-level constructs guarantee sufficient expressivity
- completeness: can everything (in the domain) be expressed?

Sugar (syntactic abstractions)

- high-level constructs support high productivity
- completeness: conceptually easy things should be easily expressable
Part XII

Demonstration
Part XIII
Implementation
The WebDSL Generator

Transformation pipeline

- Parsing
- Importing modules
- Desugaring
- Declaring definitions
- Typechecking (also of embedded queries)
- Template expansion
- Derivation
- Code generation (JPA/Hibernate + Seam + JSF)
- Write code models to file

Implementation / metrics

- Implemented in Stratego/XT
- Rewrite rules with concrete syntax
- Time: first commit March 8, 2007 (3 months / 1 week ago)
- At most 50% spent on DSL
The WebDSL Generator

Syntax

- 1081 HQL.sdf // migrated from antlr grammar (included)
- 44 MixHQL.sdf // generated
- 9 StrategoWebDSL.sdf
- 86 WebDslMix.sdf // generated
- 215 WebDSL.sdf
- 1435 total

Generator (129 rules)

- 271 dsl-to-seam.str
- 109 generator.str
- 280 java-code.str
- 234 java-Entity.str
- 432 java-page.str
- 49 java-utils.str
- 507 xhtml-page.str
- 1882 total

Transformations (166 rules)

- 591 desugar.str
- 194 expand-page.str
- 116 java-types.str
- 112 register-declarations.str
- 524 types.str
- 1537 total

Utils 380 LOC - should be in library
Part XIV

Unfinished Business
Modeling Web Applications

Implementation is no longer an obstacle

• Easy to try alternative scenarios

Domain modeling

• Coupling
• Inverse associations or queries
• Roles
• Subtyping
• ...

Interaction modeling

• UI design
• Interaction patterns
• ...

Completeness of WebDSL

- Loose ends
  - Pagination of query results
  - Collections of value types
  - Punctuation in generated output (commas, delimiters, ...)
  - Better URLs

- More default interaction patterns
  - Identify styles of interaction and generate good defaults
  - In particular associations

- Rich(er) userinterface
  - Integration of iteration with UI components
  - Using AJAX JSF components
  - Single page user interface (e.g. using Echo2) (Jonathan Joubert)
Completeness of WebDSL

- Input validation and conversion
- Security
  - authentication and access control (Danny Groenewegen)
  - Preventing injection attacks (seems to be covered well by base frameworks?)
- Workflow: business process modeling
- and of course: business logic
  - what is needed? (what is business logic, by the way?)

Engineering

- Testing of WebDSL applications
Implementation of WebDSL

- Pretty-printed error messages (instead of dumping terms)
- Templates that abstract over template element (not only via hooks)
- Fully typechecking HQL expressions
- Easier name mangling with guaranteed consistency (?)
- Optimization of database queries

General Concerns

- DSL interaction and separate compilation (Sander Mak)
  - modular typechecking, template expansion, ...
  - generate modular code (depends on target platform)
- Reusable framework for DSL implementation
  - parameterized with syntax definition
  - organizes main generator pipeline
  - generation of multiple files
  - import chasing
Programming Environment

IDEs for DSLs

- New DSL not supported by IDE (Eclipse)
- Generate Eclipse plugin from language definition
  - syntax highlighting
  - syntax checking
  - typechecking
  - refactoring
  - ...
- Integrate Stratego/XT with Safari (IBM)

Visualization

- Visual views
  - class diagrams
  - page flow diagrams
- Editing via visual views?
Deployment

**Status**

- Generation of JSF and Java source files
- Skeleton of application source tree generated by seam-gen
- Manual build steps
  - .app to code (make)
  - code to .war/.ear (ant)
  - activation of database & webserver

**Future**

- Generate complete source tree
- Integrate building of the source tree (build .war file)
- Automatic deployment and activation of the webserver
- WebDSL virtual machine
  - drop foo.app and activate
  - server takes care of code generation, deployment, activation
  - using Nix deployment system
Evolution

Data conversion
- Adapting entity declarations leads to new database scheme
- Convert data in old database to new one
- Define relation mapping old entities to new ones
- Generate scripts for existing tools?

Model migration
- Changing DSL definition requires adapting existing models

Abstraction evolution
- Model sweetening: apply new sugar to old models

Harvesting from legacy code
- Transform legacy EJB applications to WebDSL?
- JSF to page definitions
- Entity classes to entity declarations
- Session beans to actions
Summary: Properties of a good DSL

- Core language that covers needed domain expressivity
- Syntactic extensions that allow concise expression
- Facilities to build a library
  - Modules for organization of code base
  - Parametric abstraction over DSL fragments
Summary: How to develop a DSL?

- Choose high-level technology
  - DSL should not readdress problems already solved by technology
- Start with large chunks of programs
  - Understand the technology
  - Recognize common patterns
- Setup a basic generator early on
  - Makes it easy to experiment with alternative implementation strategies
- Don’t try to find core language from the start
  - Result may be too close to target
  - E.g., modeling language that covers all EJB concepts
- Don’t over specialize syntax
  - Template call vs header, section, ... as constructs
- Don’t over generalize syntax (XML)
Future

- Extend WebDSL (see ideas before)
- Apply to industrial case studies
- Abstractions for application (business) domains?
  - finance, insurance, ...
- Repeat exercise for other domains
- Develop systematic method for building new modeling languages