Domain-Specific Language Engineering: Part II
Scrap your Boilertemplate

Eelco Visser
Software Engineering Research Group
Delft University of Technology
Netherlands

June 14, 2007
MoDSE Colloquium
How far did we get?

- language for domain models
- with some refinements to support sophisticated crud operations
- generate entity classes, session beans, and JSF pages
- coarse grained language: templates are fairly big (lots of reuse), but also inflexible

Techniques

- declarative syntax definition
- rewrite rules with concrete syntax
- strategies (a bit)
Generating CRUD pages from domain models

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

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

User {  
  username :: String  
  password :: Secret  
  person -> Person  
}
Domain-driven Application Generation

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
What’s next?

- Scrap your boilerplate
  - a core language for web applications with
  - presentation and page flow
  - typechecking
  - data input and actions
  - query language

- Not all abstractions can be generative
  - user-defined templates
  - modules

- More sugar, please!
  - unlimited expressive power with model-to-model transformations

- Unfinished business
  - outlook on more stuff to do
Part I

Scrap your Boilertemplate™
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**

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

→ **URL**

```
/viewResearchGroup.seam?group=1
```

**Page navigation**

```java
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) {
    div("outersidebar"){
        div("logo"){ ... }
        div("sidebar"){ ... }
    }
    div("outerbody"){
        div("menubar"){
            div("menu"){ ... }
        }
        div("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;
    width: 10em;
}

.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
```html
<div class="sidebar">
  <ul>
    <li>
      <a href="#">Research Group</a>
    </li>
    <li>
      <a href="#">People</a>
    </li>
    <li>
      <a href="#">Publications</a>
    </li>
    <li>
      <a href="#">Projects</a>
      <ul>
        <li>
          <a href="#">Project 1</a>
        </li>
        <li>
          <a href="#">Project 2</a>
        </li>
      </ul>
    </li>
  </ul>
</div>
```
Composing Presentations: Styling Sidebar

```css
.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;
}
```
div("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 li:hover ul,
div.menu ul ul ul li:hover ul,
div.menu ul ul li:hover table
{
    display: block;
    width: 9em;
    border: ...;
    background-color: white;
}
Styling Presentations

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
Template Call

- concrete syntax
  \[ f(e_1, \ldots, e_m) \{ elem_1 \ldots elem_n \} \]
- abstract syntax
  \[ \text{TemplateCall}(f, [e_1, \ldots, e_m], [elem_1, \ldots, elem_n]) \]
- expression and element argument lists are optional

Examples

- div("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**
  
  \[
  \text{for}( x : \text{sort in e} ) \{ \text{elem*} \}
  \]

- **abstract syntax**
  
  \[
  \text{For}(x, \text{sort, e, elem*})
  \]

**Example**

- **list**
  
  \[
  \text{list} \{ \\
  \quad \text{for}(p : \text{ResearchProject in pers.groups}) \{ \\
  \quad \text{listitem} \{ \\
  \quad \quad \text{navigate}(p\text{.acronym, viewResearchProject(p))} \\
  \quad \}
  \}
  \}
  \]
Part II

Translating to JSF+Seam
Mapping Pages to JSF + Seam

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
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 :
  TemplateCall("div", [String(x)], elems) -> %>
    <div class="<%= x %>">
      <%= <elems-to-xhtml> elems ::*%>
    </div>
  </%

Mapping Presentation Elements to JSF: Nested Sections

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

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

```plaintext
elem-to-xhtml :
  TemplateCall("section", [], elems1) -> elems2
  where {
    | SectionDepth
    : rules( SectionDepth := (SectionDepth <+ !0); inc > )
    ; elems2 := <elems-to-xhtml> elems1
    |}
elem-to-xhtml :
  TemplateCall("header", [], elems) -> %>
  <~n:tag><%= <elems-to-xhtml> elems ::*%></~n:tag>
  <%
  where n := <SectionDepth <+ !1>
    ; tag := <concat-strings>[/"h", <int-to-string> n]
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 :
  TemplateCall("navigate", [ThisCall(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 :
    (Arg(x, s@SimpleSort(x_Class)), e) ->
    %><f:param name="<%= x %>"> value="<%= el %>"> /</%>
  where <defined-java-type> s
    ; el := <arg-to-value-string> FieldAccess(e, "id")
list{ for ( project : ResearchProject ) {
    listitem { navigate(project.acronym, viewResearchProject(project)) }
}}

<ul>
    <ui:repeat var="project" 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,e,elems1) -> %>
    <ui:repeat var="<%= x %>">
        <%= elems2 ::*%>
    </ui:repeat>
<%
    where el := <arg-to-value-string> e
    ; elems2 := <elems-to-xhtml> elems1
%>
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 :
  Arg(x, SimpleSort(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 :
  Arg(x, SimpleSort(x_Class)) -> bstm*|
    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
Part III

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)
Typechecking: Rules

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, "/"] )
Data Input and Actions

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
Data Input and Actions: Translation

```java
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();
    }
}
```

```html
<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

Done
@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)
Queries: Embedding HQL

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();
    }
}
Queries: Syntax and Type Checking

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 IV

Not all abstraction can be generative
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
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

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>403</td>
</tr>
<tr>
<td></td>
<td>171</td>
</tr>
<tr>
<td></td>
<td>91</td>
</tr>
<tr>
<td></td>
<td>without new &amp; all menus</td>
</tr>
<tr>
<td></td>
<td>not counting } on single line</td>
</tr>
<tr>
<td>viewBlog.xhtml</td>
<td>353</td>
</tr>
<tr>
<td></td>
<td>164</td>
</tr>
<tr>
<td></td>
<td>without new &amp; all menus</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>
New & All Menus

WebDSL rocks!

but textareas should be a tad larger ... and even possible to include wiki style markup in text. For instance, if I put **bold text** it should end up as bold text. But why do we even bother?

Ok, I don't get them anymore. It is also possible to include text:

1. first item
2. second item

read more ...

Global Variables

During on of our chats on current affairs, that Lennart Kats had proposed to introduce global variables in the ActionScript language. My second reaction was to immediately not to add some sort of C style global variables syntax for a programming pattern that we considered somewhat improper, at least.

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...
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></td>
<td>715</td>
</tr>
<tr>
<td>*.xhtml</td>
<td>17329</td>
</tr>
<tr>
<td>*.java</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>
<th><strong>without } on a line</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>serg.app</td>
<td>983</td>
<td>715</td>
</tr>
<tr>
<td>*.xhtml</td>
<td>17329</td>
<td></td>
</tr>
<tr>
<td>*.java</td>
<td>1848</td>
<td></td>
</tr>
<tr>
<td>*BeanInterface.java</td>
<td>4069</td>
<td>entity classes</td>
</tr>
<tr>
<td>*Bean.java</td>
<td>15588</td>
<td></td>
</tr>
<tr>
<td>generated java</td>
<td>21505</td>
<td></td>
</tr>
<tr>
<td>generated total</td>
<td>38834</td>
<td></td>
</tr>
<tr>
<td>generated : source</td>
<td>39.5</td>
<td></td>
</tr>
<tr>
<td>serg-full.app</td>
<td>15970</td>
<td></td>
</tr>
<tr>
<td>generated : source</td>
<td>2.4</td>
<td></td>
</tr>
<tr>
<td>serg-full.app</td>
<td>9165</td>
<td><strong>without } on a line</strong></td>
</tr>
<tr>
<td>generated : source</td>
<td>4.2</td>
<td></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
Templates: Reusing Page Fragments

Define a fragment once

    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

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

Template definition calls other templates

define main() {
    div("outersidebar") { logo() sidebar() }
    div("outerbody") {
        div("menubar") { menu() }
        body()
        footer()
    }
}

(Re)define hook templates locally

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

```plaintext
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

```plaintext
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*} ] | -> | [ div(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) {
    div("main"){
        div("outersidebar") {
            div("logo"){ ... } div("sidebar"){ ... }
        }
        div("outerbody") {
            div("menubar") { div("menu") { ... } }
            div("body") { ... } div("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 system costs as little as 11 LOC

```plaintext
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 V

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

Subtitle:

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.
Output: Entity Links

Pattern

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

Abstraction

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

Desugaring rule

DeriveOutputSimpleRefAssociation :
\[
\begin{align*}
&\left[\text{output(e)\{}\right] \rightarrow \left[\text{navigate($viewY(e)$)\{text(e.name)\}}\right] \\
\text{where SimpleSort($Y$) := } &<\text{type-of}> \text{ e} \\
&; <\text{defined-java-type}> \text{ SimpleSort($Y$)} \\
&; $viewY := <\text{concat-strings}>["view", \ Y]
\end{align*}
\]

Enabled by type annotations on expressions
Similar desugaring rules

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

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

Consequence

- output(e) sufficient for producing presentation
Edit Publication Transformations for Abstractions

Title: Transformations for Abstractions
Subtitle:
Authors:
- Eelco Visser [X]

Year: 2023
Abstract:
Language Stratego provides highlevel implementation of a wide range of transformation tools. Our aim is to integrate transformation into the development process and make it available to everyone. The transformations provided by the programming environment to be extensible. This paper...
**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**

```plaintext
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|[ div("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 :
  \[[\text{input}(e)\{}\]\] -> \[[\text{inputText}(e)\}\]\]
  where SimpleSort("Text") := <type-of> e

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

Consequence

- \text{input}(x.y.z)\) suffices for producing input of property
<table>
<thead>
<tr>
<th>Blog</th>
<th>Transformations and Abstractions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title</td>
<td>Global Variables</td>
</tr>
<tr>
<td>Created</td>
<td>26/04/2007</td>
</tr>
<tr>
<td>Category</td>
<td></td>
</tr>
<tr>
<td>Intro</td>
<td>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).</td>
</tr>
<tr>
<td>Body</td>
<td></td>
</tr>
</tbody>
</table>
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 VI

Demonstration
Part VII

Unfinished Business
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
- ...
Modeling Web Applications: DSL expressivity

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
IDEs for DSLs

- New DSL not supported by IDE (Eclipse)
- Generate Eclipse plugin from language definition
  - syntax highlighting
  - syntax checking
  - type checking
  - 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