A Sweet WebDSL

Desugarings in a DSL for Web Applications

Eelco Visser

Software Engineering Research Group
Delft University of Technology
Netherlands

June 20, 2007
First MoDSE Workshop
The WebDSL Experiment

General problem

- Technology for DSL implementation available (Stratego/XT)
- To make DSLs a standard abstraction facility, we need
- A systematic approach to design of domain-specific languages

An experiment

- Take an application domain: web applications
- Design and implement a DSL for this domain
- Draw lessons from the experiment
- In the future: repeat this experiment for other domains
- Result: a method for DSL design with improved tool support

Status of the experiment

- Tutorial presented in MoDSE colloquium
- GTTSE summerschool paper and presentation upcoming
Desugarings in Implementation of WebDSL

**DSL design challenge**

- Find sweet spot in expressivity spectrum
- Flexible: adequately cover application domain
- High-level: order of magnitude reduction in effort (LOC)

**Sugar bullet: have your cake and eat it too**

- Core language with good coverage of domain
- Sugar: high-level constructs that provide good abstractions
- Desugarings: model-to-model transformations that transform sugar to core

**Proven concept**

- Haskell, SDF, Stratego, ...
- Compiler intermediate languages (not accessible to programmer)
This Talk

- WebDSL demonstration
- The WebDSL language by example
- Architecture of the generator
- Model-to-model desugarings
- Future work
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 Deft 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.

Recent Publications

- 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
- Transformations for Abstractions
- Concrete Syntax for Objects

People

- Martin Bravenboer
- [List of other people]
The Deft Software Engineering Research Group is concerned with methods and techniques for building high quality software. This not only includes software construction, but also requirements analysis, integration, testing, deployment, and making changes to software systems after release. The Group has a deep understanding of how people build and evolve software systems.

- to develop novel methods, techniques and tools that advance the way in which software is built and adjusted and
- to offer students an education that prepares them to take a leading role in complex software development projects.

Recent Publications

- 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
- Transformations for Abstractions
- Concrete Syntax for Objects

People

- Martin Bravenboer
Martin Bravenboer

Coordinates

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

Publications

- Preventing Injection Attacks with Syntax Embeddings (2007)
- Concrete Syntax for Objects (2004)

Projects

- Transformations for Abstractions (TFA)
- Capturing Timeline Variability with Transparent Configuration Environments (TraCE)

Generated with StatagoXT
Martin Bravenboer

Coordinates

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

Publications

- Grammar Engineering Support for Precedence Rule Recovery and Compatibility
  Checking (2007)
- Preventing Injection Attacks with Syntax Embeddings (2007)
- Concrete Syntax for Objects (2004)

Projects

- Transformations for Abstractions (TFA)
- Capturing Timeline Variability with Transparent Configuration Environments (TraCE)
# Edit Person Martin Bravenboer

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fullname</td>
<td>Martin Bravenboer</td>
</tr>
<tr>
<td>Email</td>
<td><a href="mailto:martin.bravenboer@gmail.com">martin.bravenboer@gmail.com</a></td>
</tr>
<tr>
<td>Homepage</td>
<td><a href="http://martin.bravenboer.name">http://martin.bravenboer.name</a></td>
</tr>
<tr>
<td>Photo</td>
<td>img/bravenboer.jpg</td>
</tr>
<tr>
<td>Address</td>
<td></td>
</tr>
<tr>
<td>Street</td>
<td>Mekelweg 4</td>
</tr>
<tr>
<td>City</td>
<td>Delft</td>
</tr>
<tr>
<td>Phone</td>
<td>+31 (0)51 27 83721</td>
</tr>
<tr>
<td>User</td>
<td></td>
</tr>
<tr>
<td>Blog</td>
<td></td>
</tr>
</tbody>
</table>

**Generated with StrategoXT**
Martin Bravenboer

Coordinates

homepage http://martin.bravenboer.name
email martin.bravenboer@gmail.com
address Mekelweg 4
Delft
phone +31 (0)15 27 63721

Publications

- Preventing Injection Attacks with Syntax Embeddings (2007)
- Concrete Syntax for Objects (2004)

Projects

- Transformations for Abstractions (TFA)
- Capturing Timeline Variability with Transparent Configuration Environments (TraCE)
Martin Bravenboer

Coordinates

homepage: http://martin.bravenboer.name
email: martin.bravenboer@gmail.com
address: Mekelweg 4
          Delft
phone: +31 (015) 27 63721

Publications

- Preventing Injection Attacks with Syntax Embeddings (2007)
- Concrete Syntax for Objects (2004)

Projects

- Transformations for Abstractions (TFA)
- Capturing Timeline Variability with Transparent Configuration Environments (TraCE)
Concrete Syntax for Objects

Subtitle: Domain-Specific Language Embedding and Assimilation without Restrictions

Year: 2004


Authors

- Martin Bravenboer
- Eelco Visser

New Person

Abstract

Application programmer's interfaces give access to domain knowledge encapsulated in class libraries without providing the appropriate notation for expressing domain composition. Since object oriented languages are designed for extensibility and reuse, the language constructs are often sufficient for expressing domain abstractions at the semantic level. However, they do not provide the right abstractions at the syntactic level. In this paper we describe MetaBorg, a method for providing concrete syntax for domain abstractions to application programmers. The method consists of embedding domain specific languages in a general purpose host language and assimilating the embedded domain code into the surrounding host code. Instead of extending the implementation of the host language, the assimilation phase implements domain abstractions in terms of existing APIs leaving the host language undisturbed. Indeed, MetaBorg can be considered a method for promoting APIs to the language level. The method is supported by proven and available technology, i.e., the syntax definition formalism SDF and the program transformation language and toolset Stratego XT. We illustrate the method with applications in three domains: code generation, XML generation, and user interface construction.

Projects
Edit Publication Concrete Syntax for Objects

Title
Concrete Syntax for Objects
Subtitle
Domain-Specific Language Embedding and Assimilation without Restrictions
Authors
- Martin Bravenboer [X]
- Eelco Visser [X]

Fullname

Email

Add new author

Year
2004
Abstract
Application programmer's interfaces give access to domain knowledge encapsulated in class libraries without providing the appropriate notation for expressing domain composition. Since object oriented languages are designed for extensibility and reuse, the language constructs are often sufficient for expressing domain abstractions at the semantic level. However, they do not provide the right abstractions at the syntactic level. In this paper we describe MetaBorg, a method for providing concrete syntax for domain abstractions to application programmers. The method consists of embedding domain specific languages in a general purpose host language and assimilating the embedded domain code into the surrounding host code. Instead of extending the implementation of the host language, the assimilation phase implements domain abstractions in terms of existing APIs leaving the host language undisturbed. Indeed, MetaBorg can be considered a method for promoting APIs to the language level. The method is supported by proven and available technology, i.e., the syntax

Projects
- TraCE [X]
- TFA [X]
Edit Publication Concrete Syntax for Objects

Title  Concrete Syntax for Objects
Subtitle Domain-Specific Language Embedding and Assimilation without Restrictions

Authors
- Martin Bravenboer [X]
- Ealco Visser [X]

Fullname
Email

Year  2004
Abstract
Application programmer's interfaces give access to domain knowledge encapsulated in class libraries without providing the appropriate notation for expressing domain composition. Since object-oriented languages are designed for extensibility and reuse, the language constructs are often sufficient for expressing domain abstractions at the semantic level. However, they do not provide the right abstractions at the syntactic level. In this paper we describe MetaBorg, a method for providing concrete syntax for domain abstractions to application programmers. The method consists of embedding domain specific languages in a general purpose host language and assimilating the embedded domain code into the surrounding host code. Instead of extending the implementation of the host language, the assimilation phase implements domain abstractions in terms of existing APIs leaving the host language undisturbed. Indeed, MetaBorg can be considered a method for promoting APIs to the language level. The method is supported by proven and available technology, i.e., the syntax

Projects
- TraCE [X]
- TFA [X]
Edit Publication Concrete Syntax for Objects

Title
Concrete Syntax for Objects

Subtitle
Domain-Specific Language Embedding and Assimilation without Restrictions

Authors
- Martin Bravenboer [X]

Fullname

Email

Add new author

Year
2004

Abstract
Application programmer's interfaces give access to domain knowledge encapsulated in class libraries without providing the appropriate notation for expressing domain composition. Since object-oriented languages are designed for extensibility and reuse, the language constructs are often sufficient for expressing domain abstractions at the semantic level. However, they do not provide the right abstractions at the syntactic level. In this paper we describe MetaBorg, a method for providing concrete syntax for domain abstractions to application programmers. The method consists of embedding domain specific languages in a general purpose host language and assimilating the embedded domain code into the surrounding host code. Instead of extending the implementation of the host language, the assimilation phase implements domain abstractions in terms of existing APIs leaving the host language undisturbed. Indeed, MetaBorg can be considered a method for promoting APIs to the language level. The method is supported by proven and available technology, i.e., the syntax

Projects
- TraCE [X]
- TFA [X]

Pdf

Save | Cancel
Edit Publication Concrete Syntax for Objects

Title: Concrete Syntax for Objects
Subtitle: Domain-Specific Language Embedding and Assimilation without Restrictions
Authors: ● Martin Bravenboer
Year: Eelco Visser
Abstract: Jos Warner
Lennart Kats
Joost Visser
Zef Hemel
Eric Bouwoper
Arie van Deursen

The interfaces give access to domain defined in class libraries without providing the
for expressing domain composition. Since
impressive composition and expressing the embedded domain code
at the semantic level. However, they do not
provide the right abstractions at the syntactic level. In this
paper we describe MetaBorg, a method for providing concrete syntax
for domain abstractions to application programmers. The method
consists of embedding domain-specific languages into a general
purpose host language and assimilating the embedded domain code
into the surrounding host code. Instead of extending the
implementation of the host language, the assimilation phase
implies domain abstractions in terms of existing APIs leaving
the host language undisturbed. Indeed, MetaBorg can be considered
a method for promoting APIs to the language level. The method is
supported by proven and available technology, i.e., the syntax

Projects: ● TraCE
● TFA


Save  Cancel
Edit Publication Concrete Syntax for Objects

Title: Concrete Syntax for Objects
Subtitle: Domain-Specific Language Embedding and Assimilation without Restrictions

Authors:
- Martin Bravenboer [X]
- Eelco Visser [X]

Year: 2004

Abstract:
Application programmer's interfaces give access to domain knowledge encapsulated in class libraries without providing the appropriate notation for expressing domain composition. Since object-oriented languages are designed for extensibility and reuse, the language constructs are often sufficient for expressing domain abstractions at the semantic level. However, they do not provide the right abstractions at the syntactic level. In this paper we describe MetaBorg, a method for providing concrete syntax for domain abstractions to application programmers. The method consists of embedding domain-specific languages in a general purpose host language and assimilating the embedded domain code into the surrounding host code. Instead of extending the implementation of the host language, the assimilation phase implements domain abstractions in terms of existing APIs leaving the host language undisturbed. Indeed, MetaBorg can be considered a method for promoting APIs to the language level. The method is supported by proven and available technology, i.e., the syntax

Projects:
- TraCE [X]
- TFA [X]
Concrete Syntax for Objects

Subtitle: Domain-Specific Language Embedding and Assimilation without Restrictions

Year: 2004


Authors

- Elco Visser
- Martin Bravenboer

New Person

Abstract

Application programmer's interfaces give access to domain knowledge encapsulated in class libraries without providing the appropriate notation for expressing domain composition. Since object-oriented languages are designed for extensibility and reuse, the language constructs are often sufficient for expressing domain abstractions at the semantic level. However, they do not provide the right abstractions at the syntactic level. In this paper we describe MetaBorg, a method for providing concrete syntax for domain abstractions to application programmers. The method consists of embedding domain-specific languages in a general purpose host language and assimilating the embedded domain code into the surrounding host code. Instead of extending the implementation of the host language, the assimilation phase implements domain abstractions in terms of existing APIs, leaving the host language undisturbed. Indeed, MetaBorg can be considered a method for promoting APIs to the language level. The method is supported by proven and available technology, i.e., the syntax definition formalism SDF and the program transformation language and toolset Stratego XT. We illustrate the method with applications in three domains: code generation, XML generation, and user interface construction.

Projects

http://127.0.0.1:8080/sorg/viewPerson.seam?person=3
Eelco Visser

Coordinates

homepage http://www.eelcovisser.net
email visser@acm.org
address Mekelweg 4
Delft
phone +31 (0)15 27 67088

Publications

- Domain-Specific Language Engineering (2007)
- Preventing Injection Attacks with Syntax Embeddings (2007)
- Transformations for Abstractions (2005)
- Concrete Syntax for Objects (2004)

Projects

- Model-Driven Software Evolution (MoDSE)
- Transformations for Abstractions (TFA)
- Capturing Timeline Variability with Transparent Configuration Environments (TraCE)
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 one 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 multiple dimensions of evolution. In regular evolution, the modeling language is used to make the changes. In meta model evolution, changes are required to the modeling notation. In platform evolution, the code generators and application framework change to reflect new requirements on the target platform. Finally, in abstraction evolution, new modeling languages are added to the set of (modeling) languages to reflect increased understanding of a technical or business domain. While MDE has been optimized for regular evolution, presently little or no support exists for
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 multiple dimensions of evolution. In regular evolution, the modeling language is used to make the changes. In meta model evolution, changes are required to the modeling notation. In platform evolution, the code generators and application framework change to reflect new requirements on the target platform. Finally, in abstraction evolution, new modeling languages are added to the set of (modeling) languages to reflect increased understanding of a technical or business domain.

While MDE has been optimized for regular evolution, presently little or no support exists for
Edit BlogEntry title here

Blog: Transformations and Abstractions
Title: title here
Created: 19/06/2007
Category: 
Intro:

Body:

Comments:
## Edit BlogEntry title here

<table>
<thead>
<tr>
<th>Field</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Blog</td>
<td>Transformations and Abstractions</td>
</tr>
<tr>
<td>Title</td>
<td>I'm really too lazy to write a blog</td>
</tr>
<tr>
<td>Created</td>
<td>19/06/2007</td>
</tr>
<tr>
<td>Category</td>
<td></td>
</tr>
<tr>
<td>Intro</td>
<td>I really have good intentions, but there is never time.</td>
</tr>
</tbody>
</table>

**Body**

**Comments**

[Save | Cancel]
I'm really to lazy to write a blog

19/00/2007

I really have good intentions, but there is never time.
Transformations and Abstractions

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 multiple dimensions of evolution. In regular evolution, the modeling language is used to make the changes. In meta model evolution, changes are required to the modeling notation. In platform evolution, the code generators and application framework change to reflect new requirements on the target platform. Finally, in abstraction evolution, new modeling languages are added to the set of (modeling) languages to reflect increased understanding of a technical or business domain. While MDE has been optimized for regular evolution, presently little or no support exists for metamodel, platform and abstraction evolution. In this paper, we analyze the problems raised by the evolution of model based software systems and identify challenges to be addressed by research in this area.

read more ...

Global Variables

During on of our chats on current affairs, Martin mentioned that Lennart Kais 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 ...

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
section pages.

define blogSidebar(blog : Blog) {
  personSidebar(blog.author)
}

define blogEntries() {}

define page viewBlog(blog : Blog) {
  main()

  var entries : List<BlogEntry> :=
    select distinct e from BlogEntry as e, Blog as b
    where e member of b.entries
    order by e._created descending;

  define blogEntries() {
    list{
      for(entry : BlogEntry in entries) {
        listitem { navigate(entry.name, viewBlogEntry(entry)) }
      }
    }
  }

  define sidebar() { blogSidebar(blog) }

  define manageMenu() {
    navigate("Edit", editBlog(blog))
    form{actionLink("New Blog", createNewBlogEntry())}
    action createNewBlogEntry() {
      var entry : BlogEntry :=
        BlogEntry{
          blog := blog
          title := "title here"
        };
    }
  }
}
Transformations and Abstractions

I'm really to lazy to write a blog

19/00/2007

I really have good intentions, but there is never time.

Read more | Edit | Delete

WebDSL rocks!

01/00/2007

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 | Edit | Delete

Global Variables

26/04/2007

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 | Edit | Delete
Address {
  street :: String
  city   :: String
  phone  :: String
}

Person {
  fullname :: String (name)
  email    :: Email
  homepage :: URL
  photo    :: Image
  address  <> Address
  user     -> User
  blog     -> Blog
}

ResearchGroup {
  acronym :: String (name)
  fullname :: String
  mission  :: Text
  logo     :: Image
  members  -> Set<Person>
  projects -> Set<ResearchProject>
  colloquia -> Set<Colloquium>
  news     -> List<News>
}
Blog {
    title :: String (name)
    author -> Person
    entries <> List<BlogEntry>
}

BlogEntry {
    blog -> Blog
    title :: String (name)
    created :: Date
    intro :: Text
    body :: Text
    comments <> List<BlogComment>
}

BlogComment {
    author -> Person
    text :: Text
}
define page viewResearchGroup(group : ResearchGroup) {

    for(p : Person in group.members) {
        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
- 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)) }
        } }
    }
}
WebDSL: Complex Page Layout

MPL - Flock

SERG

People Projects Manage Login

MPL
People
Publications
Projects
- MoDSE
- TFA
Colloquia

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

Done
WebDSL: Complex Page Layout
WebDSL: Complex Page Layout

```javascript
define page viewResearchGroup(group : ResearchGroup) {
    div("outersidebar"){
        div("logo"){ ... }
        div("sidebar"){ ... }
    }
    div("outerbody"){
        div("menubar"){
            div("menu"){ ... }
        }
        div("body"){
            section {
                header{text(group.fullname)}
                ... 
            }
        }
    }
}
```

boxes, sidebars, menus can all be styled using CSS
Publication{
  authors -> List<Person>
  ...
}

page viewPerson(person : Person) {
  var pubs : List<Publication> :=
    select pub from Publication as pub, Person as p
    where (p.id = ~person.id) and (p member of pub.authors)
    order by pub.year descending;
  for(p : Publication in pubs) { ... } 
}
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)
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) { ... }
        }
    }
}
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) "."
}
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
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
- Size of the implementation: 3500 LOC
- Time: first commit March 8, 2007 (3 months / 1 week ago)
- At most 50% spent on DSL
Rewriting with Concrete Syntax

entity-to-class :
    Entity(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*
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**

```
navigate(viewPublication(pub))\{text(pub.name)\}
```

**Abstraction**

```
output(pub)
```

**Desugaring rule**

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

**Enabled by type annotations on expressions**
Similar desugaring rules

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

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

Consequence

- output(e) sufficient for producing presentation
Input: Editing Entity Collection Associations

**Edit Publication Transformations for Abstractions**

- **Title**: Transformations for Abstractions
- **Subtitle**: 
- **Authors**: Eelco Visser [X]
  - Eelco Dolstra
  - Sander Mak
  - Ali Mesbah
  - Gerardo Geest
  - Martin Bravenboer
  - Eelco Visser
  - Jos Warmer
  - Lennart Kats
  - Joost Visser
  - Eric Bouwer
  - Arie van Deursen

**Abstract**: Language Stratego provides highlevel implementation of a wide range of

The aim is to integrate transformation in the development process and make it available to

requires the transformations provided by the programming environment to be extensible. 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|[ 
        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:
\[
[\text{input}(e)\{\} ] \rightarrow [\text{inputText}(e)\{\} ]
\]
where SimpleSort("Text") := <type-of> e

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

Consequence

- input(x.y.z) suffices for producing input of property
### Edit BlogEntry Global Variables

<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>
</tbody>
</table>

Done
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
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
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
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
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
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)