Domain-Specific Language Engineering

reflections on the design and implementation of domain-specific languages with a case study in the domain of web engineering

Eelco Visser

Software Engineering Research Group
Delft University of Technology
Netherlands

May 28, 2008

2. Groenewegen and Visser. Declarative access control for WebDSL: Combining language integration and separation of concerns. ICWE 2008


6. Vermolen, Visser. Heterogeneous Coupled Evolution of Software Languages
Using Domain-Specific Languages

- WebDSL: workflow, access control, user interface, data model

Language Design

- Inductive language design
- Model / code interaction
- Language evolution

Language Implementation

- Syntax definition
- Code generation by model transformation
- Combining analysis and transformation

Future

- Research and engineering challenges
Part I

Domain-Specific Languages
A domain-specific language (DSL) is a high-level software implementation language that supports concepts and abstractions that are related to a particular (application) domain.
Evaluation Criteria for DSLs

Expressivity
- reduction of programming effort (order of magnitude?)

Coverage
- adequacy of abstractions for application domain

Completeness
- generation of complete application

Portability
- independence of programming platform

Code Quality
- correctness, efficiency, robustness of generated code

Maintainability
- impact of changes to model
Evaluation Criteria for DSL Development Approaches

Productivity
- development time for a new language

Difficulty
- level of specialization required language development

Systematic and Predictable
- reproducibility of the language design and implementation process

Maintainability
- ease of changing and extending language
Incarnations of Domain-Specific Languages

Application frameworks (libraries)
  • language = application programmers interface

Internal languages / DS Embedded Languages
  • framework in language with semi-syntactic-extensibility

Embedded DSL / DS language extensions
  • extension of GPL with DS notation (e.g. Swul in MetaBorg)

Interpreted DSL
  • interpreted language, possibly through framework (e.g. XSLT)

Compiled DSL / External DSL
  • stand-alone language with compiler (generator)
Approaches vs Criteria

See paper

[Visser 2008]
WebWorkFlow: An Example DSL

Workflow

- coordination of *activities* performed by *participants*
  manipulating *artifacts* (documents)

WebWorkFlow: abstractions for workflow in web applications

- documents = objects in data model
- activities = procedures
- coordination = process descriptions
- participants = access control

Example: ProgressMeeting workflow

- coordinate progress meeting between manager and employee

[Hemel, Verhaaf & Visser 2008]
entity User {
    username :: String
    password :: Secret
    name :: String
    manager -> User
    employees -> Set<User>
}

definition ProgressMeeting {
    employee -> User
    employeeView :: Text
    managerView :: Text
    report :: Text
    approved :: Bool
    comment :: Text
}
procedure meeting(p : PM) {
    process {
        (writeEmployeeView(p)
        |AND| writeManagerView(p));
        repeat {
            writeReport(p);
            (approveReport(p)
             |XOR| commentReport(p))
        } until finalizeReport(p)
    }
}

procedure writeManagerView(p : PM) {
    who { principal = p.employee.manager }
    view {
        derive procedurePage from p
        for (view(employee), managerView)
    }
}
procedure meeting(p : PM) {
    process {
        (writeEmployeeView(p)
         |AND| writeManagerView(p));
        repeat {
            writeReport(p);
            (approveReport(p)
             |XOR| commentReport(p))
        } until finalizeReport(p)
    }
}

procedure writeEmployeeView(p : PM) {
    who { principal = p.employee }
    view {
        derive procedurePage from p
        for (view(employee), employeeView)
    }
}
procedure meeting(p : PM) {
    process {
        (writeEmployeeView(p)
        |AND| writeManagerView(p));
        repeat {
            writeReport(p);
            (approveReport(p)
            |XOR| commentReport(p))
        } until finalizeReport(p)
    }
}
procedure meeting(p : PM) {
    process {
        (writeEmployeeView(p)
        |AND| writeManagerView(p));
        repeat {
            writeReport(p);
            (approveReport(p)
                |XOR| commentReport(p))
        } until finalizeReport(p)
    }
}

procedure writeReport(p : PM) {
    who { principal = p.employee.manager }
    view {
        derive procedurePage from p
        for (view(employee),
            view(employeeView),
            view(managerView),
            report)
    }
}
procedure meeting(p : PM) {
    process {
        (writeEmployeeView(p) | AND | writeManagerView(p));
        repeat {
            writeReport(p);
            (approveReport(p) | XOR | commentReport(p))
        } until finalizeReport(p)
    }
}

procedure approveReport(p : PM) {
    who { principal = p.employee }
    do { p.approved := true; }
}

procedure commentReport(p : PM) {
    who { principal = p.employee }
    view {
        derive procedurePage from p for (view(employee), view(report), comments)
        do { email(commentNotification(p)); }
    }
}
procedure meeting(p : PM) {
    process {
        (writeEmployeeView(p)
         |AND| writeManagerView(p));
        repeat {
            writeReport(p);
            (approveReport(p)
             |XOR| commentReport(p))
        } until finalizeReport(p)
    }
}

procedure commentReport(p : PM) {
    who { principal = p.employee }
    view {
        derive procedurePage from p for (view(employee), view(report), comments)
    }
    do { email(commentNotification(p)); }
}
procedure meeting(p : PM) {
    process {
        (writeEmployeeView(p)
         |AND| writeManagerView(p));
        repeat {
            writeReport(p);
            (approveReport(p)
             |XOR| commentReport(p))
        } until finalizeReport(p)
    }
}

procedure finalizeReport(p : PM) {
    who { principal = p.employee.manager }
    when { p.report != "" && p.approved }
}
Progress Meeting: Breakdown

Source lines of code

$ wc -l *.app
  95  ac.app
 127  sourceMeeting.app
  53  templates.app
 275  total

Target lines of code (without unnecessary files)

$ wc -l view//*.xhtml
  2560  total
$ wc -l beans/*.java
  8289  total
$ wc -l domain/*.java
  733  total

----------- +
 11582  total  <---------- too high!
Using Domain-Specific Languages

- WebDSL: workflow, access control, user interface, data model

Language Design

- Inductive language design
- Model / code interaction
- Language evolution

Language Implementation

- Syntax definition
- Code generation by model transformation
- Combining analysis and transformation

Future

- Research and engineering challenges
Part II

Inductive vs Deductive Language Design
A systematic approach for finding domain-specific abstractions
Deductive (Top-down) Language Design

Determine scope
- requirements / domain analysis

Find abstractions
- imagination
- correspondence to other domains

Advantages / disadvantages
+ thorough analysis of domain
- late understanding / validation of requirements
- scope mismatch
- over design
- difficult to implement
Inductive (Bottom-up) Language Design

Determine scope

• identify existing (reference) applications
• identify technology

Find abstractions

• capture programming patterns
• imagination
• correspondence to other domains

Advantages / disadvantages

+ natural scope / focus
+ direct implementation
  - leaky abstractions: too narrow / close to technology
+ iterative: build abstractions on top of lower level ones
From Java/JPA/Seam/JSF to WebWorkFlow (and Back)

**Workflow**
- persistent control-flow → workflow

**Access control**
- conditional content and actions → access control

**Base WebDSL**
- modules, templates, derive

**Core WebDSL**
- JPA/Hibernate → data model
- JSF → page definitions
- Java/Seam → actions
Part III

Model / Code Interaction
How to adapt generated code?
(Non) Solutions to Model / Code Interaction

Generate Once
- No! modifying generated code destroys advantage of DSL

Protected Regions
- No! breaks generator encapsulation by exposing implementation to developer

Round-trip Engineering
- No! limits expressivity of modeling language

Built-in Types
- Yes: make external code available to model (but requires language/generator extension)

Action Language
- Yes: high-level does not imply declarative / non-procedural

Foreign Function Interface and Model Interface
- Yes! make external code available to model and make interface of model to external components
entity Issue {
  key :: String (id, unique)
  type -> IssueType
  priority -> IssuePriority
  status -> IssueStatus
  codename :: String
  title :: String
  description :: WikiText
  reporter -> User
  assignee -> User
  submitted :: Date
  updated :: Date
  due :: Date
  comments <> List<IssueComment>
}

define page issue(i : Issue) {
  ...
  output(i.description)
  ...
}
extend entity Page {
    function makeChange(.., newText: WikiText,...) : Page {
        PageDiff {
            ... 
            patch := newText.makePatch(this.content)
            ... 
        }
    }
}

extend entity PageDiff {
    function computeContent() : WikiText {
        if (next = null) {
            return patch.applyPatch(page.content);
        } else {
            return patch.applyPatch(next.content);
        }
    }
}
Language Integration and Separation of Concerns

User Interface

define page topic (topic : Topic) { ... }
define page editTopic(topic : Topic) { ... }

Data Model Extension

extend entity Topic { acl -> ACL }

Access Control Rules

predicate mayViewWeb(w : Web) {
    ((w.acl.view.length = 0) || memberOf(w.acl.view))
}
predicate mayEditWeb(w : Web) {
    memberOf(w.acl.edit)
}
rules page topic(topic : Topic) {
    mayViewTopic(topic)
}
rules page editTopic(topic : Topic) {
    mayEditTopic(topic)
}

[Groenewegen & Visser 2008]
HQL Embedding in WebDSL

```haskell
function sortedBlogEntries(b : Blog) : List<BlogEntry> {
    var entries : List<BlogEntry> :=
        select distinct e from BlogEntry as e, Blog as b
            where (b = ~b) and (e member of b._entries)
            order by e._created descending;
    return entries;
}
```

[Visser 2008]
Primitives in Stratego

// :: EpochTime -> ComponentTime
ePOCH2LOCAL-TIME =
  ?EpochTime(t)
  ; prim("SSL_epoch2localtime", t)
  ; prim-tuple-to-ComponentTime

Java extension in xTend (open ArchitectureWare)

VOID myJavaExtension(String param) :
  JAVA my.Type.staticMethod(java.lang.String);
Part IV

Language Evolution

[Vermolen & Visser 2008]
Data Model Evolution Requires Data Evolution

Data Model \rightarrow Data Model'

Data \rightarrow Transform \rightarrow Data'

Diagram showing the relationship between data models and data, highlighting the need for data model evolution.
Coupled Evolution: Data Models

Data Model — Evolve — Data Model'

Data — Transform — Data'
Coupled Evolution: Database Schemas
Coupled Evolution: XML Schemas

DTD → Evolve → DTD'

XML → Transform → XML'

Diagram showing the coupling between DTD and XML through evolution and transformation.
Coupled Evolution: Language Evolution

Grammar → Evolve → Grammar'

Program → Transform → Program'
Data Models, Schemas, Grammars have Models too

Data Model | Evolve | Data Model'
---|---|---
Data | Transform | Data'

Data Model Grammar
Coupled Evolution at All Levels in Model Hierarchy
Domain-Specific Transformation Language

Diagram:
- $M_{i+1}$ to DSTL
- $M_i$ to Evolve to $M_i'$
- $M_{i-1}$ to Transform to $M_{i-1}'$
Multi-Level Coupled Evolution

Data Model Grammar → Evolve → Data Model Grammar

Data Model → Evolve → Data Model'

Data → Transform → Data'

Transform
Using Domain-Specific Languages

- WebDSL: workflow, access control, user interface, data model

Language Design

- Inductive language design
- Model / code interaction
- Language evolution

Language Implementation

- Syntax definition
- Code generation by model transformation
- Combining analysis and transformation

Future

- Research and engineering challenges
Part V
Syntax Definition
Easy syntax definition requires generalized parsing

- declarative syntax definition
- modular syntax definition
- only full class of context-free grammars closed under composition
- embedded languages
- unrestricted context-free structure

Compare left recursion in SDF

```ml
Exp "." Id -> Exp {cons("FieldAccess")}
```

vs iteration in ANTLR

```java
FieldAccess :
  target=PrimaryExp ("." fields+=ID)* ;
```
Part VI

Code Generation by Model Transformation
Transforming Generated Code

**Code generation by term rewriting**
- generate structured representation instead of text

**Staged transformation**
- transform transformed code

**Horizontal and vertical separation of concerns**
- generator staging and language extension

**Global-to-local transformation**
- propagate context information

**Local-to-global transformation**
- generate non-local artifacts

**Transforming generated code**
- extend the target language, aspect weaving
property-to-java :
  |[ x :: srt ]| ->
  |[ private t x_field = e;
    public t x_get() {
      return x_field;
    }
    public void x_set(t value) {
      x_field = value;
    }
  ]|
  where t := <java-type> srt
  ; e := <initialization-expression> srt
  ; x_field := <concat-strings> ["_", x]
  ; X := <capitalize> x
  ; x_get := <concat-strings> ["get", X]
  ; x_set := <concat-strings> ["set", X]
Syntactic Normalization

NormalizeText :
  |[ text(e1,e2,e*) { elem* } ]| ->
  |[ text(e1) text(e2,e*) { elem* } ]|

NormalizeFor :
  |[ for(x : srt in e1 order by e2) {elem*} ]| ->
  |[ for(x : srt in e1 where true order by e2) {
       elem*
    }
  ]|
Staged Transformations

webdsl-to-seam =
  import-modules
  ; typecheck
  ; normalize-syntax
  ; expand-page-templates
  ; derive
  ; include-emitted-decs
  ; generate-code
  ; merge-partial-classes
  ; output-generated-files

normalize-syntax =
  toptdown(repeat(
    NormalizeText <+ NormalizeFor <+ ...
  ))

toptdown(s) =
  s; all(toptdown(s))
Horizontal and Vertical Separation of Concerns

WebDSL model

System

Generator aspect
elem-to-xhtml :
| [ outputWikiText(e) ] | -> %>
  <h:outputText styleClass="outputWikiText"
    value="#wdsl:wikiFormat(
      <%= estring %>, '<%= <AppRoot> %>
    )" escape="false" />
<%
  where estring := <arg-to-elem> e

  elem-to-xhtml :
  | [ inputWikiText(e) ] | -> ...

  builtin-java-type :
  SimpleSort("WikiText") -> | [ java.lang.String ] |
declare-page-argument :
    |[ x : srt ]| -> |[ x : srt ]|
    where if not(<TypeExists> srt) then
        typecheck-error(|["Illegal type ", srt,
            " for parameter ",x])
    end
    ; rules(TypeOf : x -> srt)

typecheck-variable :
    Var(x) -> Var(x){Type(srt)}
    where if not(srt := <TypeOf> x) then
        typecheck-error(|
            ["Undeclared variable ",x," referenced"])
    end
High-level Implementation of Edit Page

**Edit blog entry Test item**

<table>
<thead>
<tr>
<th>Title:</th>
<th>Test item</th>
</tr>
</thead>
<tbody>
<tr>
<td>Author:</td>
<td>Zef Hemel</td>
</tr>
<tr>
<td>Content:</td>
<td>Zef Hemel Lennart Kats Eelco Visser</td>
</tr>
</tbody>
</table>

```
define page editBlogEntry(e : BlogEntry) {
    derive editPage from e
}
```
define page editBlogEntry(e : BlogEntry) {
    section {
        header{"Edit blog entry "output(e.title)}
        form {
            table {
                row { "Title:" input(e.title) }
                row { "Author:" input(e.author) }
                row { "Content:" input(e.content) }
            }
            action("Save", save())
            action save() {
                e.save(); return blogEntry(e);}
        }
    }
}
derive-page :
  |[ derive editPage from e ]| ->
  |[ section{ header{"Edit " srt " " text(e.name)}
      form { table { row* }
        action("Save", save())
      }
    }
  ]|
  action save() {
    x.save(); return x_view(x);
  }
][
  where SimpleSort(x_view) := <type-of> e
  ; prop* := <type-of; entity-properties> e
  ; row* := <map(derive-edit-row(|x|))> prop*

derive-edit-row(|x|) :
  |[ y k srt (anno*) ]| ->
  |[ row{x_text "": " input(x.y)} ]|
[\textit{e.title} \text{ for } (e : \text{ BlogEntry} \text{ in } b.\text{entries})
\begin{align*}
\text{where } e.\text{created} &\text{ > date} \\
\text{order by } e.\text{created} &\text{ desc)}
\end{align*}]

\[\downarrow\]

\hspace{1cm}
\text{function lcf\textunderscore33(b: Blog, date: Date): List<String> \{ }
\hspace{1cm}
\text{var y : List<String> := \[]; }
\hspace{1cm}
\text{for(e : BlogEntry \text{ in } b.\text{entries}}
\hspace{2cm}
\text{where } e.\text{created} \text{ > date order by } e.\text{created desc})
\hspace{3cm}
\{ y.\text{add}(e.\text{title}); \}
\hspace{1cm}
\text{return y; }
\hspace{1cm}
\}
Local-to-Global Transformation

Lift :

\[
\begin{align*}
&[[ [ e \text{ for} (x: \text{srt} \text{ in } e2 \text{ where } e3 \text{ order by } e4) ] ] ] \\
&\rightarrow [[ x\_fun(arg*) ] ]
\end{align*}
\]

where \( x\_fun \) := <newname> "lcf"
; \( fvs \) := <collect-free-vars> (e, e2, e3, e4)
; \( param^* \) := <map(build-param)> \( fvs \)
; \( arg^* \) := <map(build-arg)> \( fvs \)
; <emit-webdsl-dec> [[
  function x_fun(param^*) : List<srt> {
    var y : List<srt> := [];
    for (x : srt in e2 where e3 order by e4) {
      y.add(e);
    }
    return y;
  }
]
]

emit-webdsl-dec : def -> def
where rules( AddWebDs1Def :+ def* -> [def | def*] )
Extending the Target Language

parameter-to-java :
| [ x : srt ] | -> |
| [ @Partial class x_PageBean { |
  private t _#x;
  public void set#x(t x) {
    _#x = x
  }
  public t get#x() {
    return _#x
  }
  @Partial void initializeParams() { bstm* }
} |
where bstm* := <parameter-to-initialization>
; t := <defined-java-type> srt
; x_PageBean := <CurrentPageBean>
Part VII

Combining Analysis and Transformation
Example: interaction between typechecking and (type-based) desugaring (derivation) in WebDSL
High-level definition of user interface

define page editNewsItem(i : NewsItem) { ... 
    table{ derive editRows from i }...
}
Interaction between Typechecking and Derivation

High-level definition of user interface

define page editNewsItem(i : NewsItem) { ...
    table{ derive editRows from i{NewsItem} }...
}

define page editNewsItem(i : NewsItem) { ...
    table{
        row{ "Name: " input(i.name) }
        row{ "Text: " input(i.text) }
        action("Save", save())
    }
}

define page editNewsItem(i : NewsItem) { ...
    table{
        row{ "Name: " inputString(i.name) }
        row{ "Text: " inputText(i.text) }
        action("Save", save())
    }
}

entity NewsItem {
    name :: String
    text :: Text
}
High-level definition of user interface

define page editNewsItem(i : NewsItem) { ...
    table{ derive editRows from i{NewsItem} }
} ...

Derive editRows

define page editNewsItem(i : NewsItem) { ...
    table()
    row{ "Name: " input(i.name) }
    row{ "Text: " input(i.text) }
    action("Save", save()) action save() { ... }
}...
High-level definition of user interface

define page editNewsItem(i : NewsItem) { ...
    table{
        derive editRows from i{NewsItem} ...
    }
}

Derive editRows

define page editNewsItem(i : NewsItem) { ...
    table(){
        row{ "Name: " input(i.name{String}) } 
        row{ "Text: " input(i.text{Text}) } 
        action("Save", save()) action save() { ... }... 
    }
}

entity NewsItem {
    name :: String
    text :: Text
}
High-level definition of user interface

```plaintext
define page editNewsItem(i : NewsItem) { ...
  table{ derive editRows from i{NewsItem} }...
}
```

**Derive** editRows

```plaintext
define page editNewsItem(i : NewsItem) { ...
  table()
  row{ "Name: " input(i.name{String}) }
  row{ "Text: " input(i.text{Text}) }
  action("Save", save()) action save() { ... }
}
```

**Derive** input

```plaintext
define page editNewsItem(i : NewsItem) { ...
  table()
  row{ "Name: " inputString(i.name) }
  row{ "Text: " inputText(i.text) }
  action("Save", save()) action save() { ... }
```
Which analysis values need to be recomputed after applying a transformation step?
Combining Attribute Grammars and Rewriting

Motivation

- attribute grammars better suited for analysis
- rewriting better suited for transformation

Idea

- unified formalism with attribute equations and rewrite rules (and strategies?)

Problems

- does not resolve the incremental update problem, rather
- makes the incremental update problem manifest
- what is the semantics of applying a rewrite / multiple rewrites in the context of attribute values?
- who is in control?

[Your Name 2008?]
Part VIII

Development Environments for Domain-Specific Languages
IDEs for DSLs

IDEs enhance productivity

- syntax highlighting, outline, folding, cross references, ...
- error checking, debugging, ...

DSLs require custom IDEs

- domain-specific syntax, checking, linking, ...

IDE development is expensive

- implementing Eclipse in Java requires lot of effort
- oAW xText does a good job
- no support for advanced syntax, e.g. embedded languages
- sdf2imp: Eclipse IMP plugin from syntax definition
Generating Editors for Embedded Languages

rules

webdsl-action-to-java-bean:
 [ [ action x_action(farg*) { stat* } ] ] ->
 [ [ package pkgname; ] ]

import pkgname2.*;

@Stateful @Name("~x_actionBean")
public class x_ActionBean implements x_Action {

  @Logger private Log log = initLog();

  RuleManager rules;

  @PersistenceContext(type = EXTENDED)
  private EntityManager entityManager;

  public String x_action() {

    @Remove @Destroy
    public void destroy() {

    }

  }

}]

where pkgname := <BeanPackage>;
pkgname2 := <DomainPackage>;
bslm* := <statements-to-java> stm*;
Using Domain-Specific Languages

- WebDSL: workflow, access control, user interface, data model

Language Design

- Inductive language design
- Model / code interaction
- Language evolution

Language Implementation

- Syntax definition
- Code generation by model transformation
- Transformation modularity
- Combining analysis and transformation

Future

- Research and engineering challenges
Part IX

DSL Economics
What are Viable Business Models for DSLs?

Use existing DSLs

- productivity increase
- costs: license fee, vendor lock-in,

Develop new DSLs

- sell as product
- use to provide service (build software faster)
- high investment
- keeping proprietary poses risk of vendor lock-in

Scope of DSLs

- particular application (DSM)
- technical or business domain = many applications
A DSL-based Software Company?

are you up to the challenge?
company has jobs for one or two students code generation of insurance products in XSLT
Part X

Domain-Specific Language Engineering Research Challenges
Generalized Parsing for the Masses
- error messages, error correction, semantic actions

Binary Extensible Parsers
- parse table composition

Binary Extensible Generators
- compilation/linking
- feature interaction

Generating IDEs
- from language definition to Eclipse plugin
Research Context for Master’s Thesis Projects

**MoDSE: Model-Driven Software Evolution**
- Developing and applying DSLs in anger
- Companies: De Amersfoortse, ATOS, Getronics, Avanade, ...

**ASSESS: Security**
- Access control, validation, data integrity, ...

**TFA: Transformations for Abstractions**
- Extensibility of syntax and transformations
- Generating editors from language definitions

**PDS: Pull Deployment of Services**
- Distributed software deployment & MDE
- Philips Healthcare

**Buildfarm 2.0: Continuous Integration**
- Automatic software building, testing, and releasing