WebDSL

A Domain-Specific Language for Dynamic Web Applications

Danny M. Groenewegen Zef Hemel Lennart C. L. Kats Eelco Visser

Delft University of Technology, The Netherlands

{d.m.groenewegen,z.hemel,l.c.l.kats,e.visser}@tudelft.nl

Abstract

WebDSL is a domain-specific language for the implementation of dynamic web applications with a rich data model. It consists of a core language with constructs to define entities, pages and business logic. Higher-level abstractions, modeling access control and workflow, are defined in a modular fashion as extensions of the core language.

Categories and Subject Descriptors D.1.2 [*Programming Techniques*]: Automatic Programming; D.2.3 [*Software Engineering*]: Coding Tools and Techniques

General Terms Languages, Design

1. Introduction

Developing web applications comprises many technical concerns, such as data representation, querying, and modification, input handling, user interface design, and navigation. Often separate languages are used to address these various concerns. For example, a typical web application may use the Java general-purpose programming language, the SQL query language, the JavaServer Faces (JSF) presentation language with the EL expression language for accessing data and XML for configuration of frameworks.

Having separate languages for different technical domains is conceptually appealing, but the way these languages evolved introduced a number of issues for web application programmers. One issue is the presence of redundancy and inconsistency among the languages, as shown by the EL expression language, which is essentially a non-strict subset of Java expressions. Another issue is in the lack of awareness of other languages in different tools that support them. For instance, the Java compiler performs static analysis, but is oblivious of other languages involved, such as SQL queries embedded in Java strings. Lack of static checking of such expressions increases the chance of injection attacks. Integrating aspects defined in different languages often also requires additional boilerplate code, which is repetitive and error-prone to implement.

The WebDSL project aims at providing an integrated web development platform, alleviating the issues associated with a heterogeneous environment. To deal with its inevitably increasing complexity that occurs as the project evolves, we formulated the approach of *code generation by model transformation* [2] to ensure a modular and extensible architecture, ensuring maintainability of its implementation.

2. WebDSL

WebDSL is a Domain-Specific Language (DSL) for the implementation of dynamic web applications with a rich data model [6, 5]. WebDSL consists of smaller sub-languages addressing different technical domains which are statically checked and transformed into one coherent web application.

WebDSL provides sub-languages for the specification of data models and for the definition of pages for viewing and editing objects in the data model. Consider Figure 1, which illustrates a simple web application. Its data model is described using *entity definitions* (e.g., `User`[1] and `ProgressMeeting`[2]), containing properties with a name and a type. *Page definitions*[3] can be parameterized with and instantiated for specific entities[4]. They specify a presentation[5] of a web page and its associated entities. Navigation between pages is expressed in the form of `navigate` elements that specify linked pages[10].

To provide the user with the ability to manipulate entities, the example page specifies `input`[6] elements that allow a user to input different properties of an entity. The modification of elements is finalized by means of an *action*[7], which specifies the operation to undertake when the page is to be saved. In the definition of an action definition[8] objects can be further modified and persisted to the database. Actions may affect navigation by specifying `return`[9] with a page reference.

Building upon the WebDSL core language elements, higher abstractions have been created in WebDSL for access control and workflow. The *access control* abstraction of

OOPSLA'08, October 19–23, 2008, Nashville, Tennessee, USA.
ACM 978-1-60558-220-7/08/10.

```
entity User {1
  username  :: String (id)
  password  :: Secret
  name      :: String
  manager   -> User
  employees -> Set<User>
  isAdmin   :: Bool
}
```

```
entity ProgressMeeting {2
  employee     -> User
  employeeView :: Text
  managerView  :: Text
  report       :: Text
  approved     :: Bool
  comment      :: Text
}
```

```
define page editUser3 (u : User4) {
  title { "Edit User: " output(u.name) }5
  section {
    header { "Edit User: " output(u.name) }
    form {
      par { "Name: "     input(u.name)6 }
      par { "Password: " input(u.password) }
      par { action("Save Changes", saveUser())7 }
    }
    action saveUser() {8
      u.persist(); return viewUser(u);9
    }
    navigate(home()) { "return to home page" }10
  }
}
```

```
rule page editUser(u : User) { principal = u }11
principal is User with credentials username, password12
```

```
procedure meeting(p : ProgressMeeting) {13
  process {
    (writeEmployeeView(p) |AND| writeManagerView(p));
    repeat {
      writeReport(p);
      (approveReport(p) |XOR| commentReport(p))
    } until finalizeReport(p)
  }
}
```

Figure 1. WebDSL example

WebDSL [1] offers an integrated and concise way of specifying access to the application components. Access is governed by rules that determine access to matching elements[11]. The application data can be accessed with the same expressions used in the action language, offering a linguistically integrated way to specify access checks. The resulting checks are statically verified and woven into the application. A user representation can be declared using the *principal*[12] definition, which offers a default login facility but is flexible to accomodate other types of authentication as well.

The workflow abstraction, *WebWorkFlow* [3], is an object-oriented workflow modeling language for high-level descriptions of workflows in web applications. Workflow descriptions define procedures[13] operating on domain objects, representing coordinated activities between different actors. Procedures, the basic elements of these activities, are composed using sequential and concurrent process combinators. These workflow definitions result in task pages, task lists, status pages and navigation between them.

3. Code generation by model transformation

The architecture of WebDSL follows the approach of *code generation by model transformation* [2]. It has been developed using Stratego/XT, a high-level term rewriting system that integrates model-to-model, model-to-code, and code-to-code transformations. The language provides *rewrite rules* for the definition of basic transformations, and *programmable strategies* for building complex transformations that control the application of rules. Using strategies, the WebDSL generator is divided into different transformation stages. Each consists of a set of rewrite rules that rewrite extensions of the WebDSL core language to more primitive language constructs. Using this technique of compilation by normalization [4], we gradually reduce the semantic gap between input and output model, thus avoiding the complexity associated by directly generating code from the input mode. Extensions of the WebDSL language, such as the access control and workflow abstractions are realized as plug-ins to the base language, extending the generator with new normalization rules.

4. Conclusion

WebDSL is a DSL that enables development of web applications at a high level abstraction with less boilerplate code. The approach of code generation by model transformation enables the generator to be easily extended with new, higher-level abstractions as illustrated by the access control and workflow extensions.

Acknowledgements This research was supported by NWO/JACQUARD projects 638.001.610, *MoDSE: Model-Driven Software Evolution*, and 612.063.512, *TFA: Transformations for Abstractions*.

References

[1] D. Groenewegen and E. Visser. Declarative access control for WebDSL: Combining language integration and separation of concerns. In *International Conference on Web Engineering (ICWE 2008)*. IEEE CS Press, July 2008.

[2] Z. Hemel, L. C. L. Kats, and E. Visser. Code generation by model transformation. A case study in transformation modularity. In J. Gray, A. Pierantonio, and A. Vallecillo, editors, *International Conference on Model Transformation (ICMT 2008)*, volume 5063 of *LNCS*. Springer, June 2008.

[3] Z. Hemel, R. Verhaaf, and E. Visser. Webworkflow: An object-oriented workflow modeling language for web applications. In K. Czarnecki, editor, *International Conference on Model Driven Engineering Languages and Systems (MODELS08)*, Lecture Notes in Computer Science. Springer, October 2008.

[4] L. C. L. Kats, M. Bravenboer, and E. Visser. Mixing source and bytecode. A case for compilation by normalization. In *Proceedings of the 23rd ACM SIGPLAN Conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA 2008)*, LNCS. ACM Press, October 2008.

[5] E. Visser. DSLs for the web (talk). In *Conference on Object-Oriented Programing, Systems, Languages, and Applications (OOPSLA 2008)*, Nashville, Tenessee, USA, October 2008.

[6] E. Visser. WebDSL: A case study in domain-specific language engineering. In *Generative and Transformational Techniques in Software Engineering (GTTSE 2007)*, volume 5235 of *LNCS*. Springer, 2008.