FormBuilder: A Novel Approach to Deal with View Development and Maintenance*

Tomas Cerny and Michael J. Donahoo

[1] Department of Computer Science and Engineering, Czech Technical University, Charles square 13, 121 35 Prague 2, CZ, tomas.cerny@fel.cvut.cz
[2] Department of Computer Science, Baylor University, P.O. Box 97356, 76798-7356 Waco, TX, US, jeff_donahoo@baylor.edu

Abstract. In most web applications, the attributes of entity classes directly determine the content of corresponding view forms. In addition, these entity classes often encapsulate constraints on associated properties. For example, a User entity class may have an email property with constraints on the form of the address; consequently, the view form for creating/updating users should include an email field and perform validation on the submitted data. Unfortunately, view form development is often done manually, an error-prone and tedious process. Form error detection is particularly difficult because the errors only manifest themselves at runtime because of weak type safety and limited mechanisms for constraint verification. In addition, entity modification may cause inconsistency with the corresponding form. In this paper, we propose a new tool, FormBuilder, which automates the development and maintenance of view forms. The application of this tool in production applications has repeatedly demonstrated the practical contribution of this approach.

Keywords: Code-generation, Form development, Client-side validation

1 Introduction

Enterprise application development often builds on a *3-tier architecture* [1], *object-oriented programming* (OOP) [1] [2] and at some point utilizes *model-view-controller* (MVC) pattern [2] to design application with user interface flexible towards changes in layout and application logic. When we look at MVC in more detail, the *model* represented by entity classes introduces dependencies for corresponding *view* forms. If we build an application using bottom-up approach then entity classes design precedes the view design. During the view development we need to propagate entity properties to the view forms and perhaps also to view tables. As one entity property may contain multiple additional constraints, the form development can be a challenging task, because most of the properties and constraints must be consistently used in forms. Entity property, its type, name or constraints then influence the input widget applied in the form. In addition, they

* This paper has been partially supported by the CTU SGS grant No. OHK3-118/10

also apply as widget parameters. Manual form development tends to be tedious and error-prone process that requires a lot of focus. To verify an inconsistent parameter setting is another challenge since many programming languages for view provide weak type safety and limited mechanisms for constraint verification. In addition to tedious parameter setup, a replicated decisions are made to choose an appropriate form input widget to apply for a particular entity property. Most likely we always use email input widget for property called email, date widget for date data types, etc.

In this paper we propose a novel approach to deal with problems stated above. We propose and implement view form generation from entity classes. Our approach allows us to make widget selection decision only once and apply the same decisions for all entities and possibly projects. Irregularities are also supported by configuration of exceptions. Such an approach fully eliminates manual form development. Furthermore, changes in entity classes during maintenance or evolution are immediately propagated to forms which cannot cause an inconsistency in the project. This approach provides the following benefits:

1. Generated forms always match the entity classes, this solves the problem of weak type safety for view languages (e.g. XHTML).
2. The choice of view widget is independent of specific entity class and used technology. This decouples the concept in more abstract level. Any later technology change is trivial.
3. Forms are generated with support of client-side validation and better human-computer interaction experience.
4. Application development time decreases as well as the potential for errors.
5. Our approach can be extended to provide:
   (a) Form field access control.
   (b) On-demand form generation.
   (c) Client capability determined form generation.

A tool called FormBuilder [3] capable of form generation is provided for Java Enterprise Edition (J2EE) [4] applications. The tool also contains widget libraries for JSF [5], RichFaces [6] and ICEFaces [7] that can be directly used in an enterprise application. FormBuilder is being used for development of ACM International Collegiate Programming Contest (ICPC) Contest Management system [8]. The transformation technique itself is being extended to a model level transformation that supports model-driven development (MDD) [9].

The rest of the paper is organized as follows. In Section 2, we provide a brief background of web development techniques with focus on Java technologies. Section 3 provides details about form generation. Section 4 illustrates the use of our approach on an example. Section 5 relates our work to others and we conclude in Section 6.

2 Background

Originally, web developers constructed application from static Hypertext Markup language HTML. Later dynamic pages with database-generated content became popular with connection of scripting languages such as PHP hypertext preprocessor, common gateway interface CGI scripts or Java Server Pages (JSP). These most of the time combined MVC [2], 3-tier and repository architectural patterns [1]. HTML user interface components provide the lowest common language supported by web browsers. Web applications built exclusively on HTML have limited capabilities and poor usability. Applets attempt to fill this gap between web and rich client interface; however, not all clients (browsers) have applets enabled and the initial load time is very slow. A more successful step toward rich web client utilizes server-side technology for developing web applications. Here the technology runs on the server side, and renders the user interface back to the client. An example of server-side technology is JavaServer Faces (JSF) [5], a component-based framework based on the MVC that simplifies the development of user interfaces for J2EE applications. J2EE is an industry standard for enterprise application development in Java. This standard introduces 3-tier architecture consisting of presentation tier, business logic tier and persistence tier as detailed below:

1. **The persistence tier** deals with serialization of application data. Java Persistence API (JPA) [10] is standard for persistence and object relational mapping (ORM). Furthermore, we may deal here with field validation (server-side) [11] or object queries.

2. **The business logic tier** consists of session beans or plain old java objects (POJO) that deal with business rules. Besides business logic we also find here web services or dependency injection, transaction management and for some frameworks also user sessions management. Java recommends the use of Enterprise Java Beans (EJB) [12] for this layer.

3. **The presentation tier** deals with view and with forms. This tier deals with presentation to the client. It might be responsible for both server-side and client-side validation, data conversion, navigation, etc. In J2EE we find here mostly JSP, JSF, XHTML, JavaScript or Cascade Stylesheets.

Fowler et al. [1] maps multiple existing enterprise frameworks to a general 3-tier architecture. Although some frameworks consist of 4 layers, the middle two can be merged in one. The middle layer varies the most among multiple frameworks.

In order to build an application we design and follow a plan that is built with consideration of various development methods such as agile development, unified process, test driven, etc. If we build an application with bottom-up approach, then we consider first the persistence layer and design database scheme and in our case Java entity domain model. Here we concern with mapping of relational tables from database on entities. These days although we rather use a tool, such

as Hibernate tools [13], to generate this mapping and Java entity model or inverse to generate database schema based on the Java model. An entity as understood in JPA holds data fields that represent mapped table attributes, associations and constraints. Every field has specified mapping to database table column and specified constraints as XML configuration or field annotation. Constraints are enforcing validation on submitted data before they can be persisted. This phase is pretty straight forward and can apply to a subset of overall application and iteratively evolves.

Business logic is build on the top of the persistence model and composes entities, their dependencies a provide complex operations and services. Business logic is very broad topic and more details can be found in literature [12].

The presentation layer provides the view and in some cases may provide controllers to handle user events[3]. View files form the application user interface, define layout of the application, menus for navigation, connect actions and present data. For data selections we often see tables, lists, trees, searches or filters. One specific instance of data, an entity, have its detail view. This detail view may provide read-only and editable mode, which is also influenced by user rights. What is presented by detail view is the selected entity and perhaps its associations. This view then consists of a form and multiple tables. The form which can be used for data modification should reflect with its input widgets the underlying entity fields. These input widgets are of various types which can be distinguished by the entity field data type. Unfortunately the data type can be ambiguous it could be used in multiple meanings. For example a *String* field can be understood as *short text*, *large text*, *email*, *password*, etc. as shown in Fig. 1. To decide which form input widget to use we also need to consider field constrains or name. In order to improve user satisfaction we may provide data validation on client's side. Unfortunately the client connects to the application with web browser that has generally limited capabilities and can only provide interpretation of HTML based languages, JavaScript, Cascade Stylesheets or similar. For this reason, if we want to provide the client with validation at his side, then we need to translate the validation to languages his web browser can understand.

[3] JSF suggests the controllers to be part of the presentation layer although Seam framework [14] introduces controllers in business layer as way to simplify design.

Fig. 1. Example form

Unfortunately all this work is being done manually which is a source for errors as the HTML based languages does not provide type safety and not many validation tools exist to check its compatibility with entities. Other problem is that this way a simple change in the domain model requires manual change in the view form which is often maintained by another developers.

3 Form Builder tool

In order to face issues introduced by manual form development we propose and implement a tool capable of form generation for underlying entity classes.

The tool works in multiple stages. First, it parses the input entity and field meta-data. Second, for each entity field in specified order it decides what form input widget to use. Third, it supplies entity field meta-data to the widget. Finally, it produces the entity form.

In the first stage the tool parses the entity to build a meta-data for each entity field. Many OOP languages provide an extension mechanisms, such as annotation, to keep additional meta-data. The entity may need to contain some of these extensions in order to fully determine the form field. Multiple of extensions exist for ORM and are provided by JPA standard [10]. Additionally, validation extensions exist, such as JavaBean validation (JSR 303) [11]. To fully determine form input widgets we provide our own set of extensions that are not available in any alternative library. Table 1 provides summary of extensions applicable to entity fields to fully determine form content [9].

In the second stage, with entity meta-data parsed, we need to map each field to a form input widget. The tool has, for a given entity, information about all fields and their meta-data available as tool field variables (Table 2). Based on these variables we can define conditions for a field-widget mapping (FWM) in tool's configuration file. Configuration also supports variable logical and arithmetical comparison in its decisions. An example configuration can look as:

- If $type == String$ use tag A
- If $type == Person$ use tag B
- If $entity\ field\ name$ = "x" use tag C
- If $class\ name\ and\ entity\ field\ name$ = "$x.x$" use tag D
- If $package,\ class\ name\ and\ entity\ field\ name$ = "$x.x.x$" use tag E
- If $maxLength > 100\ and\ maxLength < 150$ then use tag F
- If $dateTime == true$ then use tag G
- If $password == true$ then use tag H
- etc.

Based on the targeted view language (HTML, XHTML, XML, JSF, Facelets, etc.) a library of form input widget tags is built. Tag is simply a template for a widgets in the targeted language that uses tool variables instead of the actual values. FormBuilder then selects the template parses it and supplies the field variables to the output form. Tool variables applicable to widget tags are shown in Table 2. For example, in JSF exists an input widget for *text* as shown in Listing 1.1. A form tag for this *text* widget with use of FormBuilder variables could look like the one in Listing 1.2[4].

[4] text['key'] is related towards framework text internationalization (i18n)

Table 1. Entity class extensions

Extension	Description	Appl. type
Object-relational mapping related		
Column	Nullable, length	Any
Enumerated	Ordinal/string value	Enum
Lob	Binary data	Byte
Temporal	Database type Date, Time, TimeStamp	Date
Validation related		
Digits	Number with up to specified integer, fractional digits	Numeric, String
Max	Value most max	Num/String
Min	Value at least min	Num/String
Range	Value between min and max	Num/String
Size	Value size is between min and max	Collection, Map, Array
Pattern	Matches the reg-exp	String
Patterns	Matches the reg-exps	String
Length	Length in the range	String
Email	Match email	String
CreditCard-Number	Match credit card number	String
EAN	EAN (13) or UPC-A code	String
Future	Future date	Date
Past	Past date	Date
NotNull	Not null value	Any
NotEmpty	Not empty value	Any
Form widget related		
FormOrder	Field order in form	Any
InputLength	Input widget length	Any
JSPattern	JavaScript regular expression for validation	String
Password	Password expected	String
Link	Link expected	String
TextArea	Long text expected	String
Color	Color expected	Integer
Html	Html expected	String
Ignore	Do not generate field	Any
TableColumn	Will be used for table	Any
Type	To set specific widget	Class

Table 2. FormBuilder field variables

Variable	Description	Type
From extensions		
ignore	Is field ignored	Boolean
dateTime	Is field time	Boolean
dateStamp	Is field date and time	Boolean
dateDate	Is field date	Boolean
dateType	Textual date type	String
unique	Is unique	Boolean
maxLength	Max allowed length	Long
minLength	Min allowed length	Long
maxRange	Max allowed range	Long
minRange	Min allowed range	Long
past	Is date in past	Boolean
future	Is date in future	Boolean
email	Is it email	Boolean
notNull	Is it requred	Boolean
link	Is it URL link	Boolean
html	Is it HTML	Boolean
password	Is it password	Boolean
column	Is it table column	Boolean
jsPattern	JavaScript pattern	String
javaPattern	Java Pattern	String
label	Label for widget	String
collection	User defined collection	String
type	User defined type	String
useTag	User defined tag type	String
Syntetic		
size	Default size of widget	String
id	Unique widget id	String
value	Name of value	String
entityBean	Class name	String
entityBean-Lower	Lower letter class name	String
dataTyper	Field data type	String
dataType-Lower	Lower case data type	String

Listing 1.1. JSF Text input widget

```
<h:inputText
          id = ".."
    required = ".."
          ..
       value = ".."
       title = ".."
  styleClass = ".."
   maxlength = ".."
        size = ".."/>
```

Listing 1.2. JSF Text input widget tag

```
<h:outputText
      value = "#{text['$label']}:"
 styleClass = "label"/>

<h:inputText
         id = "#{prefix}$id"
   required = "$notNull"
      value = "#{$value}"
      title =
         "#{text['$entityBean.$id.t']}"
 styleClass = "value"
  maxlength = "$maxLength"
       size = "$size"/>
```

For all view language widgets we would build a tag library and define a mapping. Our current implementation provides tag library and mapping for JSF [5], RichFaces [6] and ICEFaces [7] widgets, which simples its use in existing projects. In addition these widgets also provide client-side validation. FormBulder then provides an appropriate widget for all non-ignored fields for which exists a defined mapping. We provide this tool as an open-source project [3] that experienced more than 1000 downloads till now.

4 Example Use

In previous chapter we introduced our tool and stages of its use. In this section we show an example use. We consider a J2EE enterprise application with large domain model. All entity classes in the model are annotated for ORM, validation and also with FormBuilder annotations (Table 1). An example entity for Person is shown in Listing 1.3.

FormBuilder reads each entity for which we aim to generate form. It uses field-widget mapping configuration and tag library supplied with our tool (such as Listing 1.2) to produce a particular form. A Person form that reflects Person entity is shown in Listing 1.4, this form is then rendered as Fig. 2. Forms are generated for all entities and it is possible to provide them for both read-only and editable modes. In addition, tables can be generated as well, as we show for Person entity in Fig. 3.

We use FormBuilder in large enterprise application [8] for more than 2 years where 36 form are auto-generated. There is no need for manual correction as it can fully generate all what is necessary. It is possible to specify special widget for a selected field in case we need to solve an unusual case.

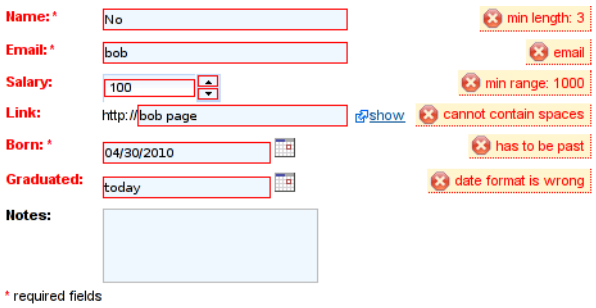

Fig. 2. Generated form

Name ▲	Email ⇕
Burnes Tom	tom@burnes.com
Doneck Bill	bill@doneck.com
Nowak Bob	bob@nowak.com
Smith John	john@smith.com

Fig. 3. Generated table

Listing 1.3. Java person entity

```
@Entity
@Table(name = "Person",
    catalog = "FormBuilder")
public class Person implements
                Serializable {
private String name;
private Date born;
..
@Column(name = "name",
    nullable = false,
      length = 100)
@NotEmpty
@Length(max = 100)
@FormOrder(1)
@FormTableColumn
public String getName() {
  return this.name;
}
public void setName(String name){
  this.name = name;
}
..
@Column(name = "born",
    nullable = false)
@Temporal(TemporalType.DATE)
@NotEmpty
@Past
@FormOrder(5)
public Date getBorn() {
  return this.born;
}
```

Listing 1.4. Generated form for JSF

```
<h:form id="formPerson">
 <h:outputText
      value = "#{text['person.name']}:"
styleClass = "label"/>

 <h:inputText
         id = "#{prefix}name"
  required = "true"
      value = "#{bean.name}"
      title =
        "#{text['person.name']}"
styleClass = "value"
 maxlength = "100"
       size = "30"/>

 <!-- other elements -->

 <h:outputText
      value = "#{text['person.born']}:"
styleClass = "label"/>
 <rich:calendar
           id = "#{prefix}born"
        value = "#{bean.born}"
     required = "true"
        title =
          "#{text[t.person.born]}"
 datePattern = "MM/dd/yyyy"
        popup = "true"/>
 <!-- other elements -->
</h:form>
```

5 Related Work

Our application development approach is not the only approach in this area that was proposed. Among features our tool provides we count view form generation from entities, table generation, the tool can propagate information used for client-side validation and improve human-computer interaction. Tool's objective is not to generate the entire application from domain objects, but simplify to apply for view development.

Related approaches vary in way the form is being generated. A tool seam-gen [14] for generation of CRUD applications motivated us to build FormBuilder. Seam-gen is provided by JBoss Seam [14] and is focused on entire application generation from a database schema. Seam-gen does not have many options to setup, which would provide more freedom and flexibility to the developer. Since there is no possibility to set user defined properties, such as input widget library in FormBuilder, seam-gen can hardly be used in production, unless the tool is recompiled for each change. FormBuilder is a good complement to seam-gen which allows developer to have more flexibility over the generated code.

A little bit different application is Naked-objects [15] proposed in 2002 by Fujitsu. They put all the effort on domain objects and from them auto-generates all user interface and application. All information necessary for user interface

generation are captured in domain objects. Such objects are then called rich domain objects. Irish government is using naked objects in their infrastructure.

Another different type of application generation is Direct to Web [16], a Java-based technology that expects the use of database from which it generates application CRUD functionality. Direct to Web provides user interface to specify generation options.

FileMaker [17] is a similar type of an application. It is non-java tool that is used to build standalone applications with data storage. In FileMaker you can very quickly generate the whole application from Entity-Relational (ER) scheme and user interface properties. FileMaker does not offer a lot of freedom options for developers and is bound to proprietary environment, but it successfully offers CRUD functionality with very friendly user interface.

Commom drawback for described applications is that, none of them offers user configurable components to use. Developers normally choose one from few options, but cannot set exactly what they want. Due to this significant drawback, applications are later re-developed manually, even though that many parts can be auto-generated by a tool.

A work on transformations in Model Driven Architecture (MDA) [18] [19] provide different view. Specifically, a survey on model transformations [18] provides a route map of MDA transformations to the solution including a template based approach. This approach is also used in seam-gen [14]. However, it is not easy for each application developer to access the templates because it requires recompiling the application generator, which is not desirable in most cases. We relate our approach to MDA in our previous work [9] where we define UML profiles for ORM, validation and form generation, that can be used by UML CASE tools in order to apply these extensions for MDA.

Many form generation tools exists as we mentioned in [9]. Some tools reflect its XML configuration for a form building. IBM XML Forms Generator [20] is a tool, in the form of eclipse plug-in, that can generate XForms [21] from given XML data instance. It can generate form elements that satisfy type and length constraints and control types according given XML scheme. Other form builders [22] [23] focus on form generation the way that user is provided with user interface where he clicks on what he wants to generate, this is not automated form generation as we propose. In our approach we define mapping and conditions under which we specify what input widget to use for an entity field that matches all user defined conditions. Then we let our tool to use the mapping and user defined widget tags to build forms for all entities. This approach allows us to use the mapping over and over again which reflects all future changes to entities. In addition to user freedom of tag definition we also show in our example tag library how to benefit from client-side validation [3].

Developers always attempt to avoid code duplication, in the idea of aspect oriented programming [24], a duplication that is automated is not an issue as it can provide benefits. Similar approach can be seen with FormBuilder where we may replicate significant amount of code and not receive any negative effect as the definition for the replicated code is being centralized.

6 Conclusion

Enterprise application view development is very complex, mostly because we need to handle multiple files and often we miss type-safety in view languages. If we can replace error-prone manual code development with auto-generated code that produces flawless code then we can only benefit. This papers shows one possible approach to reduce complexity and time for view development. As a results we provide a open-source tool FormBuilder capable of form and table generation from underlying entities. We provide a J2EE application example (available to download) [3] that uses our input widget library, which provides client-side validation as an extension for JSF-based applications. We believe our approach opens new perspective to application view development, where the entity is responsible not only for object data encapsulation, but also for view form generation. Similarly it is done for ORM, and data validation or aspect oriented security [25] [26]. In our tool we have reused existing JPA and Validation annotations for form generation to avoid wheel reinvention.

In order to improve our tool we could extended it as eclipse plug-in. Our approach could be used in addition to security where a security framework [25] [26] deciding user rights also influences form-generation per user. Our preliminary results with on-demand form-generation show that such an integration is possible. FormBuilder generates well defined form structure which could be beneficially used for integration test generation to evaluate application. The mapping configuration provide us with information what fields are expected and we could auto-generate data that will likely be accepted by the form (email address rather than just a text, etc.).

From the MDA point of view, the transformation in this paper is a transformation from entities to view code, i.e., a code-to-code transformation. Also, entities that are inputs to FormBuilder can be considered as code that is generated by incomplete transformations from a Platform Specific Model in MDA that applies ORM, Validation and Form generation UML profiles [9] [27].

References

1. Fowler, M. (2002) *Patterns of Enterprise Application Architecture*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
2. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995) *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.
3. Cerny, T., Donahoo, M. J., and Song, E. (2008). Formbuilder, collaborative learning institute - international collegiate programming contest. http://sourceforge.net/projects/form-builder.
4. Jendrock, E., Ball, J., Carson, D., Evans, I., Fordin, S., and Haase, K. (2010). The java ee 5 tutorial for sun java system application server 9.1.
5. (2010). Java server faces technology. http://java.sun.com/javaee/javaserverfaces.
6. (2010). Richfaces: Ajax jsf component library. http://jboss.org/richfaces.

7. (2010). Icefaces: Ajax jsf component library.
http://www.icefaces.org.
8. Cerny, T. (2009) The next generation web application framework for ICPC: Contest management system version 3 (CM3). Master's thesis. Baylor University, Computes Science dept. Waco, TX, USA.
http://cm.baylor.edu.
9. Cerny, T. and Song, E. (2010) A profile approach to using uml models for rich form generation, . apr., pp. 1 –8.
10. DeMichiel, L. and Keith, M. (2006). Jsr 220: Enterprise javabeans version 3.0. java persistence api.
http://jcp.org/aboutJava/communityprocess/final/jsr220/index.html.
11. (2010). Hibernate validator, open-source validation library for hibernate framework.
http://www.hibernate.org/subprojects/validator.html.
12. Panda, D., Rahman, R., and Lane, D. (2007) *Ejb 3 in Action*. Manning Publications Co., Greenwich, CT, USA.
13. (2010). Jboss hibernate: Java persistence framework.
http://www.hibernate.org .
14. (2010). Seam enterprise java development framework.
http://seamframework.org.
15. Pawson, R. and Matthews, R. (2001) Naked objects: a technique for designing more expressive systems. *SIGPLAN Not.*, **36**, 61–67.
16. (2010). Direct2web.
http://developer.apple.com/legacy/mac/library/documentation
/WebObjects/Developing$_{W}ith_{D}2W/Introduction/Introduction.html.$
17. (2010). Filemaker.
http://www.filemaker.com.
18. Czarnecki, K. and Helsen, S. (2006) Feature-based survey of model transformation approaches. *IBM Syst. J.*, **45**, 621–645.
19. France, R. B., Ghosh, S., Dinh-Trong, T., and Solberg, A. (2006) Model-driven development using uml 2.0: Promises and pitfalls. *Computer*, **39**, 59.
20. Kelly, K. E., Kratky, J. J., Speicher, S., Wells, K., and Chia, G. (2006). Ibm xml forms generator.
http://www.alphaworks.ibm.com/tech/xfg.
21. Birbeck, M. (2007). Xforms implementations, w3c xforms group wiki.
http://www.w3.org/MarkUp/Forms/wiki/XForms_Implementations.
22. (2010). Jotform - drag-n-drop form development.
http://www.jotform.com.
23. (2010). Orbeon form builder - drag-n-drop form development.
http://www.orbeon.com/forms/orbeon-form-builder.
24. Kiczales, G., Irwin, J., Lamping, J., Loingtier, J.-M., Lopes, C. V., Maeda, C., and Mendhekar, A. (1997) Aspect-oriented programming. *In ECOOP'97-Object-Oriented Programming, 11th European Conference*, June, pp. 220–242. Springer.
25. (2010). Spring security: Access-control framework.
http://static.springsource.org/spring-security.
26. (2010). Drools: Business logic integration platform.
http://jboss.org/drools.
27. Torres, A., Galante, R., and Pimenta, M. S. (2009) Towards a uml profile for model-driven object-relational mapping. *SBES '09: Proceedings of the 2009 XXIII Brazilian Symposium on Software Engineering*, Washington, DC, USA, pp. 94–103. IEEE Computer Society.