Guided Development with Multiple Domain-Specific Languages

Anders Hessellund[1], Krzysztof Czarnecki[2], and Andrzej Wąsowski[1]

[1] IT University of Copenhagen, Denmark
{hessellund,wasowski}@itu.dk
[2] University of Waterloo, Canada
kczarnec@swen.uwaterloo.ca

Abstract. We study the Apache Open for Business (OFBiz), an industrial-strength platform for enterprise applications. OFBiz is an example of a substantial project using model-driven development with multiple domain-specific languages (DSLs). We identify consistency management as one of its key challenges. To address this challenge, we present SmartEMF, which is an extension of the Eclipse Modeling Framework that provides support for representing, checking, and maintaining constraints in the context of multiple loosely-coupled DSLs. SmartEMF provides a simple form of *user guidance* by computing the valid set of editing operations that are available in a given context. We evaluate the prototype by applying it to the OFBiz project.

1 Introduction

Successful development and customization of ever more complex enterprise systems depends on effective collaboration between several stakeholders as well as on a flexible and coherent conceptualization of the problem domain. Among the different approaches towards tackling this challenge, *domain-specific modeling* seems especially promising. Domain-specific modeling can be defined as the systematic application of *domain-specific languages* (DSLs) in the design and programming phases of a development project. In complex projects, multiple DSLs are usually necessary in order to cope with different concerns. This requirement raises the need to manage the consistency among several models in multiple DSLs, which is the focus of this paper.

We give an example of an industrial-strength enterprise application framework that uses multiple DSLs, namely Apache Open for Business (OFBiz) [1]. We analyze the use of multiple DSLs in OFBiz applications by studying the OFBiz documentation, issue tracking system, developer forums, and the OFBiz implementation artifacts. We identify consistency management, and in particular ensuring referential integrity across models, as one of the key challenges of multi-DSL development. We want to address these challenges in a non-invasive way that can be incorporated in an existing development process and system architecture.

To address the problems identified in the OFBiz study, we introduce SmartEMF, which is an extension of the Eclipse Modeling Framework (EMF) [2]. SmartEMF provides support for representing, checking, and maintaining constraints using Prolog. SmartEMF can represent and check four kinds of constraints that we identified in OFBiz DSLs. Furthermore, it provides a simple form of *user guidance* by computing the valid set of editing operations that can be applied in a given context based on the current state of all models.

We believe that our study of OFBiz offers a valuable example of how multiple DSLs are used in industry today and the challenges that arise from such use. We are not aware of other quantitative studies of using multiple DSLs to describe a single system. Furthermore, although Prolog has been previously used to represent models and provide constraint checking and editing guidance, three aspects of SmartEMF are novel: (i) the use of Prolog to compute *multiple* valid operations; (ii) the exposition of how Prolog's *higher-order queries* can elegantly support constraint checking and the computation of valid operations; (iii) the support for loosely coupled DSLs by defining the valid target domain for *name-based references* through an annotation mechanism. The last capability makes it possible for SmartEMF to automatically support existing DSLs represented as XML Schemas that use name-based references to cross-link elements of individual models without the need to create a single, integrated metamodel.

The paper is structured as follows. Section 2 introduces OFBiz—our case study. Section 3 elaborates on issues in the typical process of developing applications in OFBiz. Section 4 describe the different kinds of consistency constraints that we have identified in OFBiz. Section 5 presents our tool SmartEMF, which addresses the issues by guided development with multiple DSLs. Section 6 discusses the solution and examines possible alternative approaches. Section 7 describes the related work and finally, section 8 concludes the paper and suggests possible future work.

2 Motivating Example: Apache Open for Business

The Apache Open for Business (OFBiz) framework [1] is an open source platform for building enterprise automation software, such as Enterprise Resource Planning (ERP), Content Management System (CMS), Customer Relationship Management (CRM), and Electronic Commerce systems. OFBiz is a top-level project at the Apache Foundation. Its users include both large companies, such as British Telecom and United Airlines [3], and a range of small and medium-sized ones. The framework is an excellent example of state-of-the-art, industrial-strength application development with multiple DSLs.

From a technical viewpoint, OFBiz is a J2EE framework that delivers a service-oriented architecture with persistent business objects, its own web application framework, and support for business rules, workflow, role-based security, and localization. OFBiz based applications are expressed using multiple DSLs. The core of OFBiz is an engine that can load and interpret more than fifteen DSLs (Figure 1). Each DSL covers a different aspect of application

Table 1. Overview of the OFBiz DSLs

Tier	DSL	Description	No. of Elements
Data	Entity Model	Define business objects, attributes, and relations	23
	Fieldtype Model	Define attribute types	3
	Entity Config	Configure data sources, files, and transactions	19
	Entity Group	Configure active models and entities	2
	Entity ECA	Define events, conditions, and actions for entities	5
Service	Service Def.	Define service interfaces and permissions	18
	Service Group	Configure active models and services	3
	Service Config	Configure security, threading, and service engine	13
	Service ECA	Define events, conditions, and actions for services	6
	Minilang	Implement services	154
	XPDL	Define workflows	89
UI	Screen	Implement screens and layout	65
	Form	Implement user forms and data binding	57
	Menu	Implement menus	27
	Tree	Implement visual tree structures and data binding	38
WWW	Site Config	Define web controller behaviour	15
	Regions Def.	Define screen regions	3

development such as defining business objects, services, graphical user interfaces, and workflows. Each DSL is defined using an XML Schema, and individual models are represented simply as XML documents. Table 1 provides a list of the DSLs. The table also specifies the number of elements in the schema of each DSL as an estimate of its size.

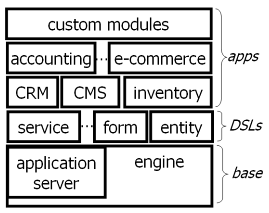


Fig. 1. The architecture

OFBiz applications are implemented as *modules* on top of the engine and the DSL layer (Figure 1). Each module typically consists of 20 to 60 models expressed in different DSLs and sometimes also custom Java code. The framework includes predefined modules such as Inventory, Customer Service, Product Catalogs, Order Entry, Accounting, and other ERP functions. Table 2 lists the artifacts constituting two of the predefined modules, including artifact sizes and numbers of cross-references among the artifacts. The framework is highly extensible allowing custom modules to be build on existing ones.

3 Application Development with Multiple DSLs in OFBiz

To understand how multiple DSLs are used in OFBiz, we analyzed the freely available documentation, including tutorials, Wiki sites, user forums, the project's issue tracking system [4], and the actual source code (stable build, September 2, 2006).

Table 2. DSL usage statistics for selected OFBiz modules

DSLs in 'Accounting' module	No. of Models	No. of Elements	No. of Cross-refs
Entity Model	2	2105	723
Entity Group	1	140	138
Service Def.	18	1726	433
Service Group	1	15	10
Service ECA	2	59	57
Minilang	16	2127	277
Form	11	2400	1141
Site Config	1	1087	228
Screen	11	1889	648
Tree	1	25	4
Menu	2	268	45

DSLs in 'Content' module	No. of Models	No. of Elements	No. of Cross-refs
Entity Model	1	1005	271
Entity Group	1	71	70
Service Def.	6	1334	389
Service ECA	1	10	9
Minilang	9	2718	506
Form	13	3487	1699
Site Config	1	1443	284
Screen	12	2303	796
Tree	2	122	11
Menu	9	366	107

Table 3. Summary of a sample OFBiz customization

Reference	OFBIZ-93: Support BillingAcct & PaymentMethod for Payment
Link	`http://issues.apache.org/jira/browse/OFBIZ-93`
Module	Accounting
Problem	The requirement is that a customer be able to use a billing account plus another form of payment, such as a credit card, for a payment on an order. The billing account is to be used first.
Solution	• New service declaration `captureBillingAccountPayment` • Java implementation of this service • An extra parameter in the `calcBillingAccountBalance` service definition • Minor changes to 3 existing service implementations and a few utility methods • Minor changes to a single screen definition

The recommended OFBiz application development process involves a bottom-up development of new models or customization of existing ones according to the tiered architecture of OFBiz [5]. The first step is to define business objects and data models using the data-tier DSLs (Table 1). Then services are defined using the service-tier DSLs and, in complex cases, also Java and scripting languages. The third step is to implement the user interface using the user interface DSLs and possibly HTML/CSS code. A particular customization employs one or more of these steps depending on its purpose and requirements.

Multiple DSLs are involved not only in the development of complete OFBiz applications, but even in small customizations of the existing ones. Descriptions of customizations are available in the OFBiz issue tracking system. In our study we have selected a sample set of eleven completed customizations of predefined applications, all categorized as *new features* or *improvement requests*. Table 3 shows an example of such a customization. The number of affected artifacts for each of the eleven customizations are listed in Table 4. The average number of affected artifacts in the selected set was five, which approximates well the number of DSLs used in an average customization.

Table 4. Number of affected artifacts per customization

Issue	Affected modules	No. of artifacts
OFBIZ-16	ECommerce	3
OFBIZ-93	Accounting	13
OFBIZ-113	Order	1
OFBIZ-188	ECommerce	2
OFBIZ-338	Manufacturing	7
OFBIZ-339	WorkEffort	6
OFBIZ-361	Webtools	5
OFBIZ-435	Marketing	4
OFBIZ-540	WorkEffort, Catalog, Product	14
OFBIZ-557	Product	4
OFBIZ-580	WorkEffort	6

We have examined the discussions in the OFBiz issue forum [4] related to the issues in Table 4 and have found that the customizations typically required several iterations of changes to the involved artifacts before they were correctly implemented. A very common problem is inconsistency among the new or modified artifacts and the existing artifacts, mainly caused by dangling references. Currently developers use ordinary XML and Java editors to implement their customizations. These tools offer little help to keep the artifacts consistent. To check for inconsistencies, the developers start up the application and run test scenarios, which is time-consuming and error-prone. According to the OFBiz forum [6], one of the main future tool requirements is better consistency checks and editing guidance that could visualize how different artifacts are related.

4 Consistency Constraints in OFBiz

Our survey has revealed that inconsistency was one of the main development problems. We will now illustrate this problem with some concrete examples taken from the OFBiz framework. We cover both the problem of consistency within a single artifact and consistency among multiple artifacts. On the surface, these cases do not seem to differ: in either the goals are to avoid dangling references, to enforce typing, and to satisfy other constraints. In practice, the mechanisms for expressing references and enforcing constraints within and across artifacts are likely to differ. Different artifacts need to support independent editing and storage and also may belong to different technical spaces, e.g., XML and Java. In the following we identify four kinds of constraints that need to be maintained in application development. Unfortunately, the current OFBiz tools cannot represent, check, and maintain these constraints.

(1) Well-formedness of individual artifacts. Currently, all OFBiz DSLs are XML-based, which means that well-formedness can be established by checking whether a model conforms to its schema. Unfortunately, XML Schemas have serious limitations. In particular, element and attribute declarations are context insensitive and therefore cannot express whether their presence depends on the presence of other elements or attributes in their context [7, Sec 4.3-4.4]. However, OFBiz requires expressing such constraints. For example, according to the OFBiz

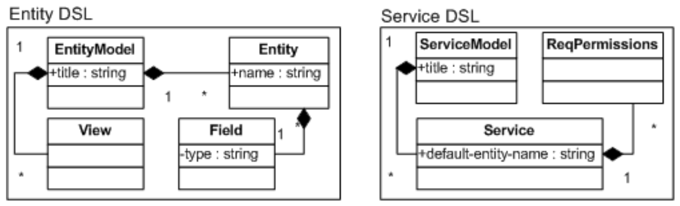


Fig. 2. Simplified excerpt from the metamodels of two DSLs

documentation, if the `alias` element in the `Entity` DSL contains the `group-by` attribute, it should also contain the `function` attribute.

(2) Simple referential integrity across artifacts. A serious and frequent problem is that of referential integrity across models. Multiple DSLs often refer to each other because they represent different views of the same system. We have identified more than 50 such references across the OFBiz DSLs. For example, each `Service` in the `Service Definition` language needs to refer to an `Entity` in the `Entity Model` language since services operate on entities. In OFBiz, as illustrated in Figure 2, all such references across DSLs are *name-based*: the value of the `default-entity-name` attribute in `Service` should match the `name` attribute of the corresponding `Entity`. Sadly, there is no mechanism in XML Schema to enforce this. Observe that *typed references*, absent from XML Schema, would offer only a partial solution to the problem since name-based references between XML and Java still need to be enforced. Cross-model name references are used also in other approaches, e.g., in Microsoft DSL Software Factories [8].

(3) References with additional constraints. One can also find more complex constraints imposed on references among OFBiz models, for example, between models expressed in the `Form` and the `Entity` languages. A typical `Form` on a webpage possesses fields linked to attributes of some `Entity`. A `Registration` form may, for instance, have fields `Firstname`, `Lastname` and `Password` generated using a reference to a `Person`. However, it is sometimes necessary to override these generated fields. We may want to create a password text widget instead of the default textfield for the `Password` field. In this case, the reference from the `Registration` form to the `Person` entity has the additional constraint that the overridden field must correspond to an attribute on the entity. More generally, all overriden form fields should correspond to attributes on the entity that the form refers to. If this constraint is violated, the engine will not be able to render the overriden form fields correctly since it can not determine their origin in the entity layer.

(4) Style constraints. A fourth class of constraints suitable for OFBiz installations are style constraints. Enforcement of such constraints is not necessary to execute OFBiz applications but facilitates maintenance in the long run. For instance, the OFBiz designers have consciously adopted typical J2EE design patterns. An example of a style constraint is to require that entity models conform to the *ObjectRole* pattern as discussed in the OFBiz forums [9]: all

entities with a name that ends with *Role* should connect entities that do not end with *Role*. This constraint ensures that relationships in the entity layer are only specified between entities and not between relationships.

5 SmartEMF

SmartEMF is an extension of the Eclipse Modeling Framework (EMF) [2] that aims at addressing the consistency management challenges identified in the previous sections. SmartEMF provides support for (i) representing, (ii) checking, and (iii) maintaining constraints of the four categories identified in Section 4.

SmartEMF builds on EMF—an implementation of an essential subset of the Meta Object Facility [10]. EMF is a platform for defining DSLs that has several advantages over XML. In contrast to XML, EMF supports typed references, proper many-to-many relationships, and a standard cross-model reference mechanism. EMF has an editing and rendering API with a command framework and an adapter layer for integration with model editors. A generator of tree-based editors is included, while graphical editors are supported via the Graphical Modeling Framework. In contrast to the string-based Document Object Model (DOM) of XML, the EMF editing API is strongly typed.

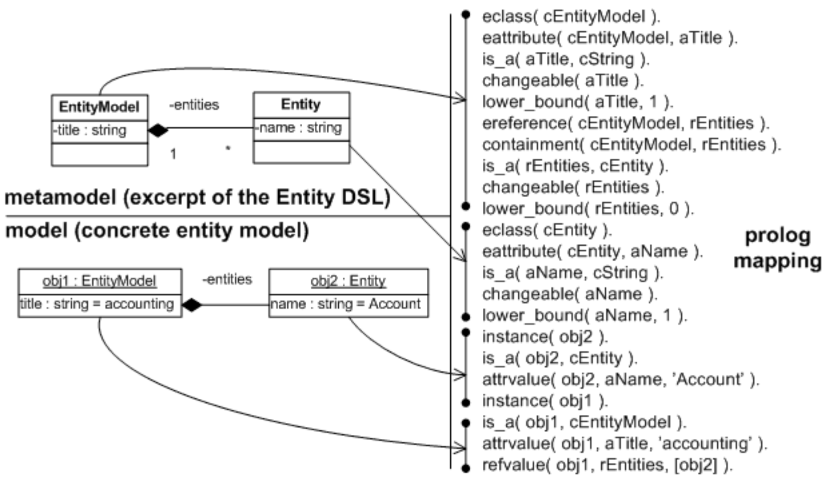
SmartEMF achieves constraint checking and editing guidance using a logical representation of EMF models. The logical representation is maintained in parallel to the model. Constraints are expressed as Prolog rules and a Prolog inference engine is used to evaluate them. For a given model a set of valid operations is inferred and presented, guiding the user to select valid targets for references. The following sections explain each of these aspects.

5.1 Ecore-to-Prolog Mapping

SmartEMF assumes that a metamodel of each DSL is given in EMF's *Ecore* notation, which closely resembles MOF [10]. EMF offers bi-directional bridges between Ecore and other technical spaces, such as XML and Java. In particular, XML Schema Definition (XSD) files of OFBiz DSLs and the corresponding XML documents can be automatically imported into EMF, which makes them accessible as Ecore models and instances.

Figure 3 shows an excerpt of the mapping from Ecore to Prolog for a fragment of the Entity model from Figure 2. The mapping is directly inspired by the GEMS project [11]. Similar to GEMS, all elements of an Ecore model representing the DSL metamodel and all elements of an Ecore model instance representing a concrete model in the DSL are declared as facts in the fact base. For example, an Ecore class is represented as a fact using the `eclass` predicate with a unique identifier of the class as an argument. N-ary predicates are used to assert relations such as between an attribute and its containing class or between an integer attribute and its upper bound.

Upon startup, our prototype initializes the fact base by traversing and asserting model elements from Ecore, EMF's embedded XML metamodel, and



```
eclass( cEntityModel ).
eattribute( cEntityModel, aTitle ).
is_a( aTitle, cString ).
changeable( aTitle ).
lower_bound( aTitle, 1 ).
ereference( cEntityModel, rEntities ).
containment( cEntityModel, rEntities ).
is_a( rEntities, cEntity ).
changeable( rEntities ).
lower_bound( rEntities, 0 ).
eclass( cEntity ).
eattribute( cEntity, aName ).
is_a( aName, cString ).
changeable( aName ).
lower_bound( aName, 1 ).
instance( obj2 ).
is_a( obj2, cEntity ).
attrvalue( obj2, aName, 'Account' ).
instance( obj1 ).
is_a( obj1, cEntityModel ).
attrvalue( obj1, aTitle, 'accounting' ).
refvalue( obj1, rEntities, [obj2] ).
```

Fig. 3. Mapping from Ecore to Prolog

all relevant DSLs and their instances. The resulting fact base then serves as the underlying representation of a reflective Ecore editor, which manipulates and queries both the EMF object model and the fact base. SmartEMF extends the standard EMF editing commands, such as *add*, *set*, and *delete*, to propagate changes of the model to the Prolog fact base.

5.2 Representing Constraints

Consistency constraints from all of the categories discussed in Section 4 can be represented as Prolog rules. Since all the DSLs and models are represented in the Prolog fact base, constraints spanning one or more DSLs are expressed naturally.

A simple example of a well-formedness constraint (1) is *required_value_present*: every mandatory feature should have a value. Every constraint consist of two parts: a name (`required_value_present`) and a rule. The rule expresses a negation of the constraint, so that the rule is satisfied whenever the constraint is violated. In our example this happens if a mandatory feature, i.e., a feature with a lower bound of 1, has value `id_UNSET`. As shown below such constraints are relatively simple to read and write.

```
% name
constraint( required_value_present ) .
% rule representing the negation of the constraint
required_value_present( Object, Feature ) :-
lower_bound( Feature, 1 ) ,
( attrvalue( Object, Feature, id_UNSET ) ;
  refvalue( Object, Feature, id_UNSET ) ) .
```

Another class of constraints (2) considers consistency relations across distinct DSLs. A DSL can refer to another one in two ways. Either by using types from the other language or by name-based references.

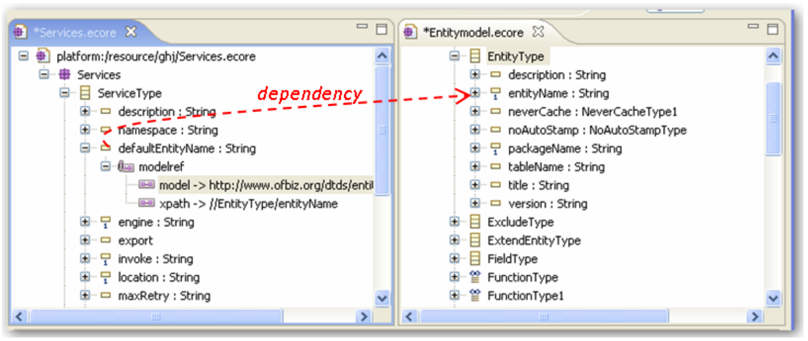


Fig. 4. Name-based reference from the *Services* DSL to the *Entitymodel* DSL

Typed references are natively supported by Ecore and our mapping to Prolog. However, name-based references require additional information in the Ecore model, which SmartEMF supports with the *modelref* annotation. Figure 4 shows how the sample reference from *Service* to *Entity* from Figure 2 is represented using the annotation. The annotation consists of two key/value pairs: a *model* key which denotes the target DSL by its namespace and an *xpath* key which is an XPath query that identifies the target element in that DSL. Provided with the corresponding values of the two keys in the *modelref* annotation, SmartEMF can determine the set of legal values (a *valid domain*) of an annotated model element. In the example, the annotation on the *defaultEntityName* attribute of the service shows that the valid values for this attribute are names of entities in the *Entitymodel* DSL. The rule expressing the negation of this constraint follows:

```
% name
constraint( no_dangling_modelrefs ).
% rule
no_dangling_modelrefs( Object, AnnotatedFeature ) :-
  modelref( AnnotatedFeature, DomainFeature ) ,
  attrvalue( Object, AnnotatedFeature, Value ) ,
  not( attrvalue( _ , DomainFeature, Value ) ) .
```

5.3 Constraint Checking Using Higher-Order Queries

Constraint checking utilizes Prolog's support for higher-order queries. The meta-logical `call` predicate facilitates such queries. The `call` predicate invokes a goal with an optional set of extra arguments. Since all our constraints are declared using the custom `constraint` predicate, we can easily compute the set of all constraints in the fact base. By using the `call` predicate, we can then determine which constraints are violated, i.e., evaluate to true for a given binding of their variables. The `check` rule states this query:

```
check( Object, Violations ) :-
  findall( [ Goal, Object, Feature ] ,
    ( constraint( Goal ) ,
    call( Goal, Object, Feature ) ) ,
    ViolationsUnsorted ) ,
  sort( ViolationsUnsorted, Violations ) .
```

If the `check` predicate is evaluated with the `Object` variable bound to an object then the result is a binding of the `Violations` variable to an empty list in case of no constraint violations or a list of tuples. In the latter case each tuple consists of the violated constraint (a goal), the concrete object, and the feature of that object. If the `check` predicate is evaluated with two variables, the query produces *all* constraint violations in the *entire* fact base in one shot.

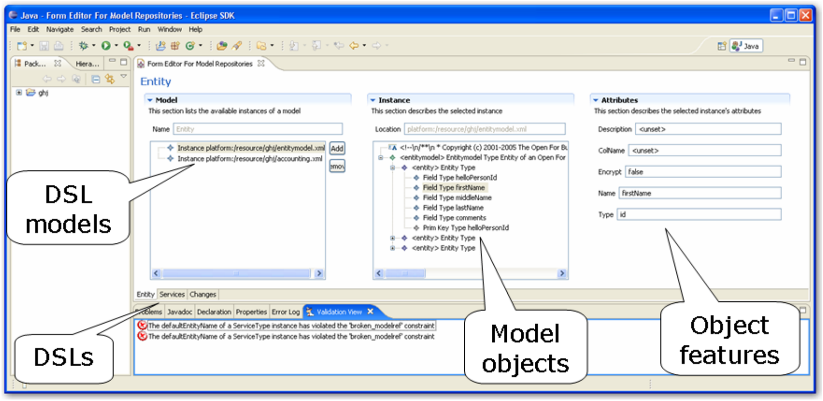
5.4 Preconditions of Editing Operations

Simple editing guidance can be offered by computing the set of editing operations (and possibly some or all of their arguments) that are available in a given context based on the current state of the fact base. This facility is achieved by representing the preconditions of editing operations such as *add*, *set*, and *delete* as Prolog rules and querying them using a higher-order query.

```
% name
operation( add ) .
% rule
add( Object, Feature, AddableTypes ) :-
   instance( Object ) ,
   is_a( Object, ObjectType ) ,
   containment(  ObjectType, Feature ) ,
   upper_bound( Feature, Upper ),
   refvalue( Object, Feature, CurrentValues ) ,
   not( length( CurrentValues, Upper ) ) ,
   is_a( Feature, AddableTypes ) .
```

The above listing shows a declaration of the *add* operation, which adds a child element to a containment reference. Similarly to constraints declarations, the precondition consists of a fact declaring the rule name and the rule. The rule states that only the instances of the feature's type can be added and only as long as the number of instances in the containment list does not exceed the upper bound. Similar preconditions are declared for other operations.

We determine valid editing operations in a context by using the higher-order query `operations` shown below. Depending on whether the `Object` and/or the `Feature` variables are bound, we can either determine all valid operations, all valid operations for a given object, or all valid values for a given feature.


DSL models
DSLs
Model objects
Object features
Java - Form Editor For Model Repositories - Eclipse SDK
File Edit Navigate Search Project Run Window Help
Pack... Hiera...
Form Editor For Model Repositories
Add
Entity Services Changes
Problems Javadoc Declaration Properties Error Log Validation View

Entity

Model
This section lists the available instances of a model
Name: Entity
Instance platform:/resource/ghj/entitymodel.xml
Instance platform:/resource/ghj/accounting.xml

Instance
This section describes the selected instance
Location: platform:/resource/ghj/entitymodel.xml
<!--\n/**\n * Copyright (c) 2001-2005 The Open For Bu
<entitymodel> Entitymodel Type Entity of an Open For
<entity> Entity Type
Field Type helloPersonId
Field Type firstName
Field Type middleName
Field Type lastName
Field Type comments
Prim Key Type helloPersonId
<entity> Entity Type
<entity> Entity Type

Attributes
This section describes the selected instance's attributes
Description: <unset>
ColName: <unset>
Encrypt: false
Name: firstName
Type: id

The defaultEntityName of a ServiceType instance has violated the 'broken_modelref' constraint
The defaultEntityName of a ServiceType instance has violated the 'broken_modelref' constraint

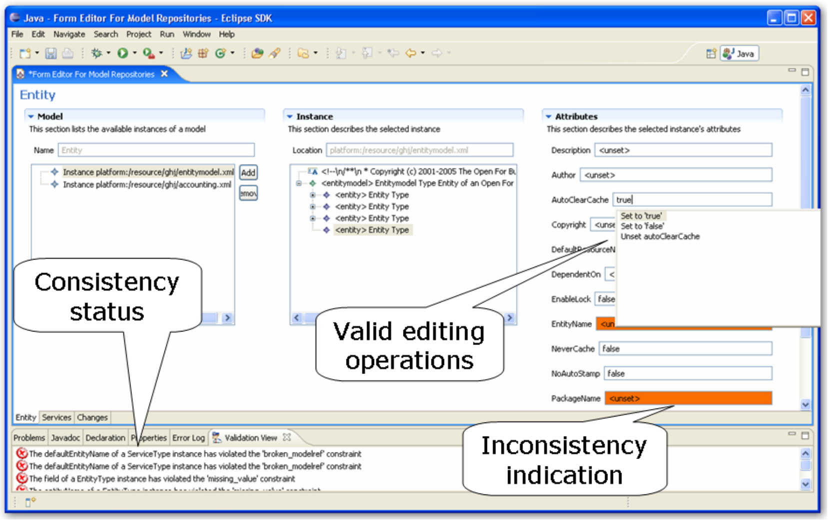
Fig. 5. SmartEMF's reflective editor

```
operations( Object, Feature, Operations ) :-
  findall( [ Goal, Object, Feature, Value ] ,
    ( operation( Goal ) ,
      call( Goal, Object, Feature, Value ) ) ,
    OperationsUnsorted ) ,
  sort( OperationsUnsorted, Operations ) .
```

5.5 Reflective Editor

SmartEMF provides a reflective editor that exploits the underlying representation and previously described queries. It is a form-based editor implemented as an Eclipse EMF plugin (Figure 5). It enables users to access and modify instances of different DSLs in a uniform way. Each DSL is represented in a tab containing three columns. The first column lists different models in the selected DSL. The second column contains a hierarchical view of the model elements in the currently selected model. The third column displays the features of a selected model element. All objects and feature values can be loaded, edited, and serialized respecting the individual file formats of their DSLs. Specifically, in the OFBiz case, every modified model is saved in XML conforming to the original DSL-defining XSDs.

The editor uses reflective capabilities of the regular EMF object model in order to structure the user interface. When the user selects an object or a feature, the framework queries the underlying representation for valid editing operations using the `operations` predicate from Section 5.4. The resulting tuples are presented in the form of various visual or textual hints as shown in Figure 6. Since the framework simultaneously queries the representation using the *check* predicate, a list of inconsistencies is available, which is both shown in the bottom as well as using other visual hints.


Java - Form Editor For Model Repositories - Eclipse SDK
File Edit Navigate Search Project Run Window Help
*Form Editor For Model Repositories
Entity
Model
Instance
Attributes
Add
Consistency status
Valid editing operations
Inconsistency indication
Entity Services Changes
Problems Javadoc Declaration Properties Error Log Validation View

This section lists the available instances of a model

Name: Entity

Instance platform:/resource/ghj/entitymodel.xml
Instance platform:/resource/ghj/accounting.xml

This section describes the selected instance

Location: platform:/resource/ghj/entitymodel.xml

```
<!--\n/**\n * Copyright (c) 2001-2005 The Open For Bu
<entitymodel> Entitymodel Type Entity of an Open For
  <entity> Entity Type
  <entity> Entity Type
  <entity> Entity Type
  <entity> Entity Type
```

This section describes the selected instance's attributes

Description	<unset>
Author	<unset>
AutoClearCache	true
Copyright	<unse
DefaultResourceN	
DependentOn	<
EnableLock	false
EntityName	<un
NeverCache	false
NoAutoStamp	false
PackageName	<unset>

Set to 'true'
Set to 'false'
Unset autoClearCache

```
The defaultEntityName of a ServiceType instance has violated the 'broken_modelref' constraint
The defaultEntityName of a ServiceType instance has violated the 'broken_modelref' constraint
The field of a EntityType instance has violated the 'missing_value' constraint
```

Fig. 6. Guidance and consistency management in the SmartEMF editor

The reflective nature of the SmartEMF editor is one of its main advantages. Most EMF editors are generated and hence customized for a particular set of models. In contrast, the SmartEMF editor allows the user to quickly include new DSLs, new instances, and new constraints just by changing the loading configuration of the editor. Upon loading, the editor automatically adapts to the current selection of DSLs while still providing guidance and consistency management. Of course, the query facilities of SmartEMF could also be used from specialized and generated editors.

6 Discussion and Suggestions for Future Work

Experience in applying SmartEMF to OFBiz. OFBiz applications are traditionally developed using built-in XML and Java editors of IDEs like Eclipse. As described in the previous section, these tools do not offer any cross-language support or editing guidance. In our experiments, we annotated the `Entity`, `Service definition`, and `Form` languages and loaded all models in these languages from the *accounting* module. The setup comprised 31 models with a total of 6231 model elements and 2297 potentially broken cross-references. We implemented a set of simple customizations, such as extending an entity, revising a service, and displaying the results in the user interface, experiencing no performance problems with checking and guidance.

We have not yet performed any user studies apart from the customizations that we ourselves have done. We do, however, expect a significant drop in undetected inconsistencies when SmartEMF is systematically applied to OFBiz projects. The guidance facilities should also speed up development since they provide very concrete hints on how to complete the models in a given installation. Empirical studies need to be performed in order to validate these claims.

Applicability to other DSL-based infrastructures. A growing number of XML-based infrastructures (e.g., Struts and Spring) use similar mechanisms as OFBiz and we expect that these infrastructures could benefit equally well from SmartEMF's guidance and consistency management facilities. Also, we wish to examine how Java-customizations can be related to DSL artifacts. One possible approach would be to extract Ecore-based models from Java code as it has been done in framework-specific modeling languages [12].

Prolog as a constraint language. An object-oriented language such as OCL, which supports a concise expression of constraints and navigation, would seem to be a better choice than Prolog. Alternatively an approach based on a relational database and SQL queries could have been adopted. While both solutions are more elegant than a pure Java approach, they do not offer the advantages of Prolog. The main advantages of Prolog are: (i) the ability to infer possible solutions by using free variables in a query, (ii) higher-order predicates such as *call*, which support highly concise and expressive global queries and (iii) an implicit representation of the solution space (unlike in databases).

More advanced guidance. Extending SmartEMF to provide guidance for more complex (composite) operations, e.g., refactorings, and for sequences of operations are interesting future work items. For operation sequencing, both pre- and post-conditions of operations need to be considered. Finally, an extra layer of guidance could be to prescribe the order in which model elements are actually created. This layer could be achieved by introducing modeling workflows or instantiation plans in the manner suggested by Lahtinen et al. [13].

7 Related Work

Consistency management. Tracking inconsistencies between models is not a new research field. The *ViewPoint* method [14] is one of the best treatments of the subject. Recently several authors have addressed the problem, also considering repairs. Mens et al. [15] propose a graph-based approach where different models are represented as a single graph. The graph can be searched for erroneous patterns. Each pattern has a corresponding repair that can be applied to the graph. Similarly, other approaches support repairs by annotating OCL-constraints with repair actions [16] or even generating repair actions automatically [17]. More advanced repairs and diagnostics may be offered by implementing model checks as transformations as suggested by Bézivin and Jouault [18]. The checking aspect of all these approaches is impressive, but the guidance is limited to the predefined repairs. SmartEMF's ability to search for possible editing operations is not present in any of these.

Mapping models to a logical fact base. We have adopted the idea of mapping an Ecore-based model to Prolog from the GEMS project [11]. The primary emphasis in the GEMS project is on creating a graphical, guided editor for one or more DSLs based on a single, thightly-integrated meta-metamodel and allowing automatic configuration of models. SmartEMF draws on this approach, but uses pure Ecore as its meta-metamodel rather than a custom Ecore-based meta-metamodel. This choice allows SmartEMF to leverage existing languages by loading, editing, and saving different models in a schema-conformant way. Furthermore, in the case of multiple DSLs, GEMS would require a composite metamodel with multiple *aspects*, or views. SmartEMF handles this case by using a generic editor which can display both regular cross-model references as well as name-based references. Another Prolog-based approach is the Design Critiquing facility of the ArgoUML tool described by Robbins et al. [19]. This approach is focused on critiquing rather than suggesting possible edits as SmartEMF.

Multiple DSL development. Development scenarios with multiple DSL is sparsely described in the existing literature. One of the best empirical studies of DSL usage to date is Tolvanen and Kelly's work [20], which is primarily concerned with single DSLs. There are, as far as we know, no comprehensive empirical studies of industrial instances of multiple DSL applications like OFBiz. An early theoretical reference is the DRACO system [21], which contains a systematic approach for dealing with multiple DSLs. Recently, Warmer and Kleppe [8] have described an approach based on multiple DSLs, but that work does not provide empirical data such as the size of the involved DSLs and the number of cross-references among them.

Partial models and name-based references. Warmer and Kleppe [8] advocate the use of loosely coupled, *partial models*. Name-based references are used to integrate multiple DSLs. Their approach uses a consistency checking mechanism tailored to a specific set of languages.

8 Conclusion

We have examined the problem of working with multiple DSLs in domain-specific modeling. Our main contributions are: (i) a qualitative study of the architecture and development problems of an industrial-strength, multiple DSL application, OFBiz, and (ii) SmartEMF—an EMF-based framework offering consistency checking and editing guidance. We have tested our prototype within OFBiz development scenarios, applying it to the problems that were identified in the study as common issues in OFBiz development. Preliminary experiments suggest that a guidance tool can significantly help in maintaining consistency during development with multiple DSLs.

References

1. The Apache Software Foundation: The Apache Open for Business Project. http://ofbiz.apache.org/ (2007) Seen March 8, 2007.
2. Budinsky, F., Steinberg, D., Merks, E., Ellersick, R., Grose, T.J.: Eclipse Modeling Framework: a Developer's Guide. Addison-Wesley (2004)

3. Chen, S.: Opening Up Enterprise Software: Why Enterprises are Adopting Open Source Applications (2006) http://www.opensourcestrategies.com/slides/.
4. The Apache Software Foundation: The Open for Business Project. Issue Tracking System. https://issues.apache.org/jira/browse/OFBIZ (seen 2007/03/22)
5. Undersun Consulting LLC: OFBiz Framework Quick Reference Book, ver. 1.5.1. http://bigfiles.ofbiz.org/FrameworkIntro/01MainDiagram.pdf (2004) Seen 2007/03/26.
6. Jones, D.E.: Requirements for an OFBiz IDE. http://www.nabble.com/Re%3A-requirements-for-an-OFBiz-IDE-p8066093.html. (2006) Seen 2007/03/27.
7. Møller, A., Schwartzbach, M.I.: An Introduction to XML and Web Technologies. Addison-Wesley (2006)
8. Warmer, J., Kleppe, A.: Building a Flexible Software Factory Using Partial Domain Specific Models. Proc. of The 6th OOPSLA Workshop on Domain-Specific Modeling (2006) http://www.dsmforum.org/events/DSM06/.
9. Howe, C.: Party Relationship Best Practices. http://www.nabble.com/Party-Relationship-Best-Practices-p5453154.html (2006) Seen 2007/03/27.
10. Object Management Group: Meta-Object Facility. http://www.omg.org/mof/ (2007) Seen March 12, 2007.
11. White, J., Schmidt, D., Nechypurenko, A., Wuchner, E.: Domain-Specific Intelligence Frameworks for Assisting Modelers in Combinatorically Challenging Domains. In: GPCE4QoS. (2006)
12. Antkiewicz, M., Czarnecki, K.: Framework-Specific Modeling Languages with Round-Trip Engineering. In: Proc. Int'l Conf. MoDELS 2006. Volume 4199 of LNCS., Springer-Verlag (2006) 200–214
13. Lahtinen, S., Peltonen, J., Hammouda, I., Koskimies, K.: Guided Model Creation: A Task-Driven Approach. In: VLHCC '06: Proc. of the Visual Languages and Human-Centric Computing. (2006) 89–94
14. Nuseibeh, B., Kramer, J., Finkelstein, A.: Expressing the relationships between multiple views in requirements specification. In: ICSE '93: Proc. of the 15th Int'l Conf. on Software Engineering. (1993) 187–196
15. Mens, T., Van Der Straeten, R., D'Hondt, M.: Detecting and Resolving Model Inconsistencies Using Transformation Dependency Analysis. In: Proc. Int'l Conf. MoDELS 2006. Volume 4199 of LNCS., Springer-Verlag (2006) 200–214
16. Kolovos, D.S., Paige, R.F., Polack, F.A.: On the Evolution of OCL for Capturing Structural Constraints in Modelling Languages. In: Proc. Dagstuhl Workshop on Rigorous Methods for Software Construction and Analysis. (2007)
17. Nentwich, C., Emmerich, W., Finkelstein, A.: Consistency Management with Repair Actions. In: Proc. of the 25th Int'l Conf. on Software Engineering, May 3-10, 2003, Portland, Oregon, USA. (2003) 455–464
18. Bézivin, J., Jouault, F.: Using ATL for Checking Models. In: GraMoT workshop, 4th Int'l Conf. on Generative Programming and Component Engineering. (2005)
19. Robbins, J.E., Hilbert, D.M., Redmiles, D.F.: Software Architecture Critics in Argo. In: IUI '98: Proc. of the 3rd Int'l Conf. on Intelligent User Interfaces, New York, NY, USA, ACM Press (1998) 141–144
20. Tolvanen, J.P., Kelly, S.: Defining Domain-Specific Modeling Languages to Automate Product Derivation: Collected Experiences. In: Proc. of Int'l Conf. SPLC'05, Rennes, France. (2005) 198–209
21. Neighbors, J.M.: Software Construction using Components. PhD thesis, UC Irvine (1980) Tech. Report UCI-ICS-TR-160.