User Interface Code Generation For EJB-Based Data Models Using Intermediate Form Representations

Branko Milosavljević, Milan Vidaković, Srdjan Komazec, Gordana Milosavljević
Faculty of Engineering, University of Novi Sad
Trg D. Obradovića 5, Novi Sad, Yugoslavia
mbranko,minja,ksrki,grist @uns.ns.ac.yu

ABSTRACT

The use of J2EE platform enables data model development based on EJB components. Data modeling concepts of EJB technology, although resembling those of relational databases, are different and need new methods for automated user interface code generation. This paper presents a method for user interface code generation based on intermediate form representations that can be used to build equivalent user interfaces in multiple environments like standalone GUI applications, web, or wireless devices. The structure of the generated system, intermediate form representations, and the functionality of user interface concepts are described in detail.

1. INTRODUCTION

The use of the J2EE platform [4] simplifies the implementation of data-intensive applications, as the platform itself provides standard middleware services like object persistence, distributed transactions, and security. EJB components [3], intended for implementing the business logic, are not a mandatory part of a J2EE system. In our previous work, we have dealt with situations when the use of EJB components is not preferrable [12]. We argue that the main reason behind the use of EJB components is to provide the same application functionality to multiple types of clients, like standalone GUI applications, web browsers, and wireless devices. EJB entity beans with container-managed persistence (CMP) [3] enable designing data models using the new modeling concepts, instead of traditional concepts from relational databases. Although CMP entity beans are usually mapped to underlying relational databases, developers can solely use EJB-based data models to develop applications.

This paper deals with automated generation of coarse-grained user interface components providing access to data stored in EJB-based data models. These coarse-grained components are forms implementing basic operations on sets of entity beans. The functionality and layout of forms is specified using the intermediate form representation (IFR). The use of IFR enables the use of multiple environments for user interface implementation. That way, the same IFR form definition can be used to instantiate forms in Swing-based standalone GUI clients, JSP-based web clients, or WAP devices with WML browsers.

PPPJ 2003, 16-18 June 2003, Kilkenny City, Ireland.

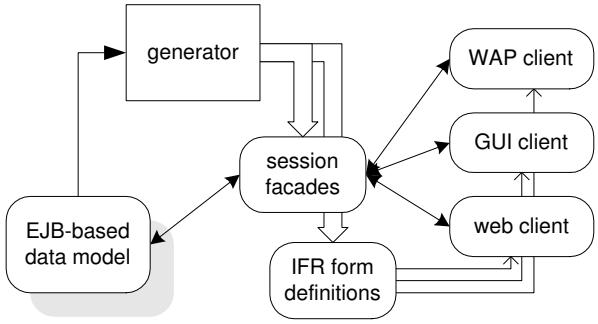

Figure 1: Code generation concept.

Figure 1 illustrates the user interface generation process and application architecture for the given EJB-based data model. The data model of interest is represented by entity beans. The generator uses bean introspection to create its own internal data model representation. Each entity bean in the data model is supplied with a generated session bean acting as a session façade [11]. Data manipulation forms access data through these façades. Another result of the generation process are IFR form definitions encoded as XML documents. IFR form definitions can be customized in terms of their functionality and layout by manually editing generated XML documents. A pre-written, generic template form uses the IFR document as a set of initialization parameters defining its functionality and layout. Client applications create concrete forms by instantiating the generic form class, supplying it with the corresponding IFR definition. There is a separate implementation of the generic form for each user interface technology. Repetitive generation of IFRs (in case of changes in the data model, for example) preserves customizations made in previous iterations.

The rest of the paper is structured as follows. Section 2 reviews the related work. Section 3 describes the generation of session bean façades. The structure of the intermediate form representation is given in Section 4. The functionality of the generic form is presented in Section 5. Finally, Section 6 concludes the paper and outlines some future research directions.

2. RELATED WORK

The area of model-based user interface development environments (MBUIDE) is currently an active research field. Some MBUIDE methods for object databases are presented in [9, 16, 15, 8]. Teallach [9, 8] is an MBUIDE developed for ODMG-compliant databases. However, so far there has been little work on designing an MBUIDE for EJB-based models. We develop our work based on user inter-

face concepts and generic form functionality concepts defined in [13].

The use of XML language for specifying documents acting as abstractions of parts of multi-tiered architectures is proposed in [10, 14]. The work in [10] uses XML to represent CORBA middle-tier components, application architecture and task graphs, thus enabling dynamic component configuration and facilitating the sharing of components between repositories. However, the user interface is tied to web environment. The use of XML as a means of user interface specification is proposed in [14]. This paper emphasizes the possibility of using the same user interface model for creating functionally equivalent user interfaces in multiple environments. In this paper, we incorporate the same idea into design of an MBUIDE system targeted for EJB-based models.

Architectural design of EJB-based systems is an issue described in [17, 11]. The emergence of design patterns in the field of EJB design has produced some widely accepted patterns. We use some of these, namely the session façade and the value object. Several commercially available or free tools deal with EJB component generation [2, 1, 5]. Sygel WMEE [6] is an advanced tool that uses UML models to build EJB data models, underlying database schemas, and JSP-based user interfaces. It supports repetitive code generation and application functionality customization, but does not produce user interface code for multiple environments.

3. GENERATING SESSION FAÇADES

Every entity bean from the domain model is accompanied by a generated session bean representing its session façade [11]. The generation process uses data collected during bean introspection. A session bean exposes basic operations on its corresponding entity bean set to clients. Clients are data manipulation forms in this case. Passing values between forms and session façades utilizes value object design pattern [11]. A value object is a compact representation of an entity bean's data. It is passed as a single unit in session bean method calls. This pattern is intended to avoid multiple network round-trips while setting values for entity bean's properties from remote clients.

Value objects are represented by instances of *ValueObject* class (see Figure 2). A value object consists of: (1) a primary key value for the corresponding entity bean, (2) a map of [name, value] pairs for CMR fields, and (3) a map of [name, value] pairs for CMP fields. The *primaryKey* attribute holds an instance of the entity bean's primary key class. A CMR field is represented by a *CMRField* class, having a collection of primary keys of beans taking part on the other side of the relationship. If the relationship multiplicity is "one", the *strRep* attribute contains the string representation of a related bean.

Instances of *CMPField* contain data from bean's CMP fields. CMP field types treated as primitive types include Java's primitive types, strings, and dates. Primitive type fields can be displayed and edited in forms. Other types of CMP fields are decomposed into tree structures having only primitive-type fields. Class *InnerField* represents a field of non-primitive type.

Classes *ValueObject*, *CMRField*, *CMPField*, and *InnerField* represent the value object pattern used for all entity beans. In contrast, the session façade is generated for each entity bean. The remote interface of the generated session bean is fixed, while its implementation is customized. Diagram in Figure 2 contains an example of a generated session façade's remote interface for entity bean named *Xxx*. Each generated session bean exposes three methods implementing basic data operations: *add*, *remove*, and *update*. Besides, there is a number of methods for finding entity beans that are invoked within the forms. Session façade generation process can be performed with tools like XDoclet [7].

4. INTERMEDIATE FORM REPRESENTATION

Each form, instantiated by an IFR document, implements data manipulation operations on a set of entity beans of a single type. An IFR document is an XML document; Listing 1 presents an example of an IFR definition for the form presented in Figures 4 and 5.

```
<?xml version="1.0" encoding="UTF-8"?>
<form name="Users" title="Users">
  <meta>
    <dateCreated>24.12.2002</dateCreated>
    <dateModified>24.12.2002</dateModified>
    ...
  </meta>
  <sorting>
    <sort name="username" order="ASC"/>
  </sorting>
  <allowedActions>
    <action name="add"/>
    <action name="update"/>
    <action name="remove"/>
    <action name="copy"/>
  </allowedActions>
  <bean name="ejb/User">
    <property name="username" type="java.lang.String">
      <format length="30" regexp=""/>
    </property>
    <property name="password" type="java.lang.String">
      <format length="30" regexp=""/>
    </property>
    <property name="firstName" type="java.lang.String">
      <format length="30" regexp=""/>
    </property>
    ...
    <property name="userGroup" type="cmr" bean="ejb/UserGroup"
      form="/userGroup.xml" multiplicity="One"/>
  </bean>
  <hooks>
    <hook operation="add" class="com.gint.blast.hooks.AddHook"/>
  </hooks>
  <customOperations>
    <operation class="com.gint.blast.custom.Reminder"
      label="Reminders..."/>
    <operation class="com.gint.blast.custom.DiskSpace"
      label="Disk space usage"/>
  </customOperations>
  <layouts>
    <swing>
      <size x="700" y="400" resizable="yes"/>
      <position x="100" y="100" center="yes"/>
      <mainProperties manager="XYLayout" width="680" height="100">
        <property name="username" beanview="yes" tableview="yes">
          <label short="username" long="Username:"
            position="20,20,-1,-1"/>
          <control type="javax.swing.JTextField"
            position="100,20,180,-1"/>
          <column width="120"/>
          <!--<read-only/>-->
          <taborder value="1">
        </property>
        ...
      </mainProperties>
      <propertyGroup name="Address" manager="XYLayout"
        width="680" height="100">
        ...
      </propertyGroup>
      <agg-field name="average" beanview="no" tableview="no">
        <label text="Avg daily proxy usage:"/>
        <control type="javax.swing.JTextField"
          width="20" align="RIGHT"/>
        <aggregator name="com.gint.blast.calc.AvgProxyUsage"/>
      </agg-field>
    </swing>
    <web>
      ...
    </web>
  </layouts>
</form>
```

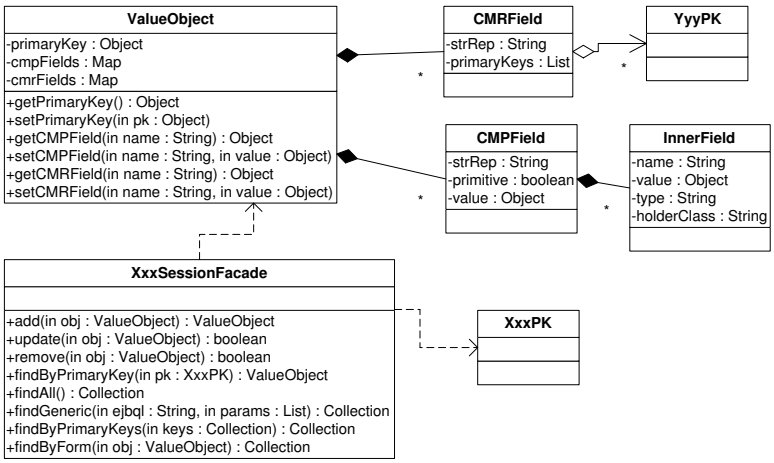

Figure 2: A value object and a generated session façade.

Listing 1: An example of an IFR document.

Description of a bean is given by the */form/bean* element. A bean is viewed as a list of properties. Each property description comprises property name, type, maximum length (if applicable), and a regular expression used for validation (if applicable). CMR properties are described by their name, the name of the form used for editing related beans, and relationship cardinality. A set of beans displayed in a form can be sorted by property values, as specified in the */form/sorting* element.

Each form provides the following standard operations on beans: addition, removal, update, copying, and query-by-form searching. Each of these operations can be disabled separately (see */form/allowedActions* element).

Form layout is defined separately for each implementation environment. Currently, we have implemented support for standalone GUI (Swing-based) and web (JSP-based) environments. The example in Listing 1 contains code only for Swing-based implementation (see */form/layout/swing* element). Bean properties, for purposes of form layout, are placed into groups. There is always one property group, defined by *mainProperties* element. Additional groups can be defined to form a hierarchy. Property groups rendering depends on the user interface environment used. Swing-based forms use tabbed panes and bordered panels to represent groups. Each property layout definition is supplied with label text and positioning constraints (depending on the chosen layout manager), as presented in */form/layouts/swing/ mainProperties/property* element.

Modifying the initially generated form layout is one aspect of form customization. Other aspects involve writing custom code that must implement some of the interfaces presented in Figure 3. Customization modules are invoked in a certain context modeled as *FormContext* class. Standard bean operations provide hooks for additional tasks to be performed at the same time. Hooks are declared in */form/hooks* element, and implement the *Hook* interface. Additional operations can be made available to the user (see */form/customOperations*). Each additional operation has to implement the *CustomOp* interface.

Two types of special fields, not listed as bean properties, can appear in forms. *Calculated fields* are fields associated with a single bean that derive its value from the given bean's properties. On the other hand, *aggregated fields* are associated with a set of beans and derive its value from their properties (for example, sum and average values). The calculation of their values is performed by instances of classes implementing *Calculator* and *Aggregator* interfaces, respectively. See */form/layouts/swing/calc-field* element in Listing 1 for an example.

The IFR document generator preserves all customizations made to the previous version of the IFR document. This feature enables incremental, prototype-based application development.

5. FORM LAYOUT AND FUNCTIONALITY

A form instantiated by an IFR document has two display modes: a table-view and a bean-view mode. The table-view mode presents a selected subset of properties of the given set of beans and provides for navigation through the set, standard operations, and custom operations. The bean-view mode is used for viewing/editing all properties. Figure represents an example of a Swing-based form instantiated by the IFR document presented in Listing 1.

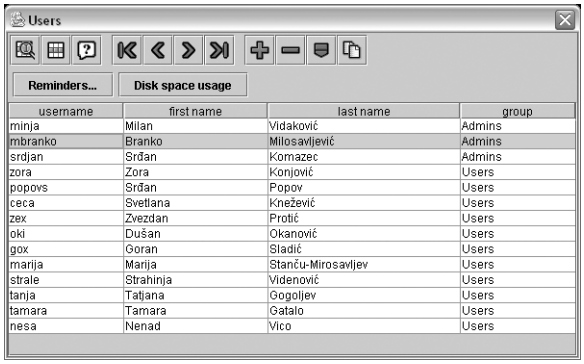

Figure 4: An example of a Swing-based form: table-view mode.

A CMP field is displayed in the bean-view mode by a label and an editing component whose type is specified in the IFR document. A CMR field is displayed with a label and a button. If the multiplicity of the other side of the relationship is "1", the string representation of the related bean is displayed next to the button (for example, the CMR field *userGroup* in Figure 5).

The button associated with the CMR field invokes a form used for editing related beans. In case of the relationship cardinality being "1:1" or "n:1" the invoked form is used for selecting the single related bean. For "1:n" cardinality the invoked form provides for editing the set of related beans. Relationships with "n:n" cardi-

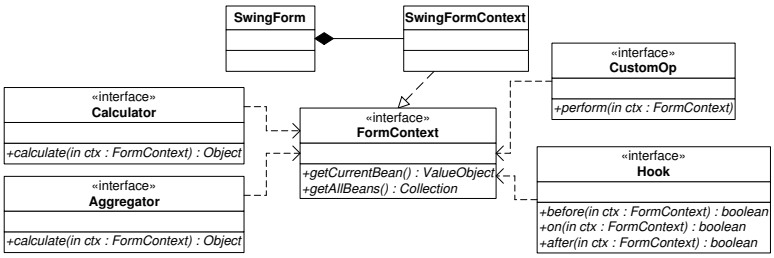

Figure 3: Form customization interfaces.

nality are handled by a special form used for connecting a set of existing beans with the currently edited bean.

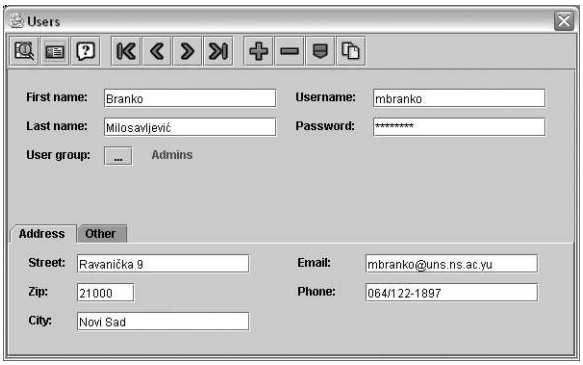

Figure 5: An example of a Swing-based form: bean-view mode.

6. CONCLUSIONS AND FUTURE WORK

The presented concept of generating user interface components enables rapid development of applications that can operate on arbitrary EJB-based data models. The central feature of the concept is the use of an intermediate form representation (IFR) as a description of form functionality and layout, which provides several benefits. (1) IFR extensibility provides support for new user interface environments. (2) The same IFR document can be used to instantiate forms in multiple user environments (e.g., standalone GUI applications, web, or wireless mobile devices). (3) A set of IFR documents, supplied with customization elements (hooks, custom operations, calculated and aggregated fields, etc.), can be used as an application functionality specification. (4) Repetitive code generation (in case of changes on the underlying data model, for example) preserves form customizations performed in the previous iteration. (5) Generator implementation is less complex, as opposed to the case of direct generation of Java client user interface code.

The main goal of the presented code generation system is to provide for rapid prototyping of applications dealing with existing EJB-based data models. It can be supplemented with other tools to aid initial development of systems following the J2EE architecture. Application prototyping for legacy relational databases can be done via creating a CMP entity bean layer on top of the given relational database. Future research directions regarding this system are directed towards designing a visual, multi-environment IFR document editor. The use of such tool can further speed up application development in the J2EE realm.

7. REFERENCES

[1] Ejbgen. http://www.beust.com/cedric/ejbgen/.

[2] Ejen (code generation system). http://ejen.sourceforge.net.

[3] Enterprise JavaBeans specification. Sun Microsystems. http://java.sun.com/products/ejb/docs.html.

[4] Java 2 platform, enterprise edition specification. Sun Microsystems. http://java.sun.com/j2ee/docs.html.

[5] Jenerator. http://www.visioncodified.com.

[6] Sygel wonder machine enterprise edition. http://www.sygel.com.

[7] XDoclet: Attribute-oriented programming. http://xdoclet.sourceforge.net.

[8] P. P. da Silva, T. Griffiths, and N. W. Paton. Generating user interface code in a model based user interface development environment. In *ACM Working Conf. on Advanced Visual Interfaces*, pages 155–160, Palermo, Italy, 2000.

[9] T. Griffiths, P. J. Barclay, J. McKirdy, N. W. Paton, P. D. Gray, J. Kennedy, R. Cooper, C. A. Goble, A. West, and M. Smyth. Teallach: A model-based user interface development environment for object databases. In *IEEE Conf. on User Interfaces to Data Intensive Systems (UIDIS'99)*, 1999.

[10] M. Li, O. F. Rana, and D. W. Walker. An XML-based component model for wrapping legacy code as Java/CORBA components. In *4th Intl. IEEE Conf. on High Performance Computing*, 2000.

[11] F. Marinescu. *EJB Design Patterns: Advanced Patterns, Processes, and Idioms*. Wiley, New York, 2002.

[12] B. Milosavljević, M. Vidaković, and Z. Konjović. Automatic code generation for database-oriented web applications. In *Principles and Practice of Programming in Java*, pages 59–64, Dublin, Ireland, 2002.

[13] G. Milosavljević and B. Perišić. An approach to automating large-scale business software systems construction phase. In *6th Conf. on Operational Research*, Thessaloniki, Greece, 2002.

[14] A. Müller, T. Mundt, and W. Lindner. Using XML to semi-automatically derive user interfaces. In *2nd IEEE Intl. Workshop on User Interfaces to Data Intensive Systems (UIDIS'01)*, 2001.

[15] W. J. Ray and A. Farrar. Object model driven code generation for the enterprise. In *12th IEEE Intl. Workshop on Rapid System Prototyping (RSP'01)*, 2001.

[16] J. Vanderdonckt and P. Berquin. Towards a very large model-based approach for user interface development. In *IEEE Conf. on User Interfaces to Data Intensive Systems (UIDIS'99)*, 1999.

[17] J. White. Enterprise JavaBean architecture and design issues. In *23rd IEEE Intl. Conf. on Software Engineering (ICSE'01)*, 2001.