alodar systems™

Advanced Development Tip

WHITE PAPER

Automating J2EE Code Generation: Intercalate for J2EE

Intercalate is a template-based modeling tool that turns a single high level description of your data into many concrete implementations you need for real software. It creates Java source, SQL, tag files, html pages, and virtually everything else you need to produce a J2EE application. With the addition of some open source tools, it can also address Web Services requirements.
The Sun J2EE specification has incredible flexibility. The cost of that flexibility is a high entry barrier -- even the most trivial J2EE applications require a lot of custom developed software to handle even the simplest tasks. Recognizing this, a number of vendors automate the generation of commonly recurring elements with tools based on pictures or rules. For applications that use these only these common elements, the increase in efficiency is well worth the investment in the tool.
If you need to do something uncommon, however, these picture or rule based tools sometimes do not handle the situation gracefully. Rather than forcing the solution to fit the tool, it is sometimes more efficient to develop the code for a special case totally outside of the tool. One becomes a "power" developer, writing a custom solution for the special case from scratch. But one of the appeals for Java of course is the enhanced possibility of reuse and increased efficiency over time. If the custom solution is developed outside of a tool based development framework, it is difficult to ensure that the solution gets used for other, similar special cases. In each instance you find yourself starting virtually from scratch again, aided only by a simple text editor to cut and paste blocks of code.
For those projects that require writing code, Intercalate addresses this dilemma by generating the code from a template with "fill in the blanks" logic. You use your favorite tools to work with the results, and the best tool for the job becomes your environment. Intercalate-generated code can be read into JBuilder, Eclipse, Visual Age, Emacs, Notepad, or any other tool that meets your needs. This scales up very well as uncommon case arise - the template gets a bit more complicated, but the technique works, and the results are robust. Further, if the same case arises elsewhere, Intercalate can automatically use the same solution. This protects you from inefficiencies in dealing with change

But how do I edit the code?

From the Intercalate perspective, the centerpiece of your enterprise application is the parameterized list of data describing your schema. Once created, it may be processed, filtered, manipulated, ruled, or whatever, but it is still data being moved from one place to another. Further, creating this schema in one form or another is critical to all well designed enterprise applications - Intercalate just uses that information directly.
It does not take that long to write the SQL to create a single table, or the JavaBean to access it. It does take a lot of time, though, to do it over and over, especially if the schema is changing. In addition, coordinating the backend tables, middle layer business logic, and front end forms and reports is a time consuming process. Intercalate addresses this by giving all of these layers access to the one and only schema representation - changes are immediately reflected at all levels of the application.
In conjunction with readily available open source frameworks, a complete J2EE application is easily created and maintained based on a simple text file holding the schema. Change that one representation, and the changes ripple through the application wherever needed.

Never reinvent the wheel

It is truly amazing how the Internet and open source software has changed the process of writing applications. Whereas before, one usually started with design, developers now start by looking to see if someone else has already solved the problem with a free solution. Because Intercalate does not hide the code from the developer, and because writing templates is straightforward, it makes it very easy to integrate open source tools. For example, if one uses Intercalate to generate XDoclet tags inside the Java source code, the open source project XDoclet will generate interface classes and XML descriptor files. The combination of these two tools forms a powerful implementation approach to creating and maintaining enterprise applications.

The examples to follow will leverage a number of open source projects in order to implement a complete J2EE application. We have selected Tomcat as the web server, JBoss as the applications server, MySQL as the database, and Struts will be used as the MVC web front end. We will also show how Axis can extend the application to web services.

Focus on the interfaces using Intercalate

The key to most enterprise applications is the user or program interface: what, where, and how the data is going to be entered, transferred, processed, and stored. The interface is where the data is touched either by a person or by a process. The Intercalate property file or the metadata of the application readily captures this information. The concept of a property file is crucial to understanding how Intercalate works. If you are not familiar with property files, see the Intercalate Tutorial

Once the property file is defined, the complete application is created through Intercalate. Any change in the interface only requires a single modification in the property file. For example, when an additional field is added to the customer information, Intercalate propagates this change through the entire application. Without Intercalate, every part of the application which touches the customer data will have to be manually modified. By automating the process, we reduce the risk of missing a table, or incorrectly implementing a form. We demonstrate this ability to make quick changes at the end of the example.

On to the example

Enterprise applications consist of many layers, but they can be broken down into a few fundamental parts. The user provides data through input forms, and generates reports for output. The main data processing occurs in the business logic middle layer and the data is stored in the persistent object backend using a relational database. So we have:

- Forms and reports (Struts/Tomcat)
- Business logic (Session Beans/JBoss)
- Persistent objects (Entity Beans/JBoss)
- Database (MySQL)

Start with the database

Many other tasks are difficult to specify until the schema is pinned down. We thus begin with how the data will be represented. For this example, we will start with the database tables and their relationships. To get a feel for the Intercalate process, we will start with a single Customer table. (For more information about Intercalate, see the Intercalate Tutorial.)

Customer Table

Key Type	Column name	Data Type	Allows null
Primary	customerId	Int	No
	firstName	Varchar(30)	Yes
	lastName	Varchar(30)	No
	email	Varchar(30)	No

Intercalate Property List

Before proceeding, we should encode the information for the Customer table into the high level parameterized form we have been talking about. If you are not concerned about the exact structure of a property list, skip down to "Intercalate Templates". Given that most developers will be using the Property List Editor to create these, the exact format is less important than the high level structure.

Each attribute of a single table is a Hashtable (list by name) of key-value pairs.

For example:

```
{
 primaryKey = "";
 attributeName = "customerId";
 columnType = "INT";
}
```

tells us that this attribute is a primary key, that the attributeName is "customerId", and that the columnType is "INT". Developers have a great deal of freedom in what they name their keys - as long as keys match between the template and plist, they can be

named anything.

These attributes are collected in a Vector called 'attributes', stored in the hashtable for the entire database table. (The table gets its own high level table because we want to store its name as well.)

```
{
 entityName = "Customer";
 attributes = (
 {
      primaryKey = "";
      attributeName = "customerId";
      columnType = "INT";
 },
 {
      attributeName = "firstName";
      columnType = "VARCHAR";
      columnSize = "30";
 },
 {
      isRequired = "";
      attributeName = "lastName";
      columnType = "VARCHAR";
      columnSize = "30";
 },
 {
      isRequired = "";
      attributeName = "email";
      columnType = "VARCHAR";
      columnSize = "30";
 });
}
```

Finally, while we could create a plist out of just this one table definition pretty much as is, we will eventually want a second table. If we create an "entities" vector holding them now, inside a hashtable, then we can add more database tables easily by just putting more table definitions in the same structure.

Thus, our final plist appears as:

```
tableName = "$entityName$";
columnName = "$attributeName(capitalize)$";
entities = (
{
 entityName = "Customer";
 attributes = (
 {
      primaryKey = "";
      attributeName = "customerId";
      columnType = "INT";
 },
 {
      attributeName = "firstName";
      columnType = "VARCHAR";
```

```
        columnSize = "30";
  },
  {
        isRequired = "";
        attributeName = "lastName";
        columnType = "VARCHAR";
        columnSize = "30";
  },
  {
        isRequired = "";
        attributeName = "email";
        columnType = "VARCHAR";
        columnSize = "30";
  });
});
```

Intercalate Template

The Intecalate template for the SQL script that creates the Customer table looks something like this. (We did add some additional carriage returns and indentation for clarity. See the example code for the actual template.)

```
$write create.sql$
$for entities$
CREATE TABLE $tableName$
(
     $for attributes$
  $columnName$ $columnType$
              $if columnSize$($columnSize$)$end$
              $if primaryKey$ PRIMARY KEY$end$
              $if isRequired$ NOT NULL$end$
              $unless _last$,$end$
     $end$
);
$end$
```

Before we analyze the template, we will need to have the property list that it operates on - while the data in a given property list may vary, the structure is intimately tied to the template it works with.

The Intercalate keywords and keys are enclosed in dollar signs $. (We chose this as it is an infrequently used character.) To make a dollar sign appear in the output text, we escape it with a second dollar sign: $$. In addition to our keys, a small but general set of keywords controls the form and function of the output.

The first Intercalate keyword in this template is 'write', which must be followed by a file name. In this template, we have written $write create.sql$, which directs the output of the template to a file called create.sql.

The next keyword is 'for', which must be followed by an array to loop over. The loop is terminated by an explicit 'end' keyword. In this template, we wish to loop over a list of attributes, so we write $for attributes$ to look at each attribute in turn. While looping over an attribute, Intercalate will pull values out of the hashtable and replace the variables listed in the template. For example, while looping over $for attributes$, $attributeName$ will be replaced in turn by "customerId, "firstName", "lastName", and "email".

We defined $columnName$ and $columnType$ as special keys at the top of the plist so that we could do special formatting. If you look at the definition of columnName, you can see that it is merely the attributeName capitalized. There is nothing special about the names we have chose - they need only be self consistent in a single plist and its corresponding templates.

The if keyword only outputstext if the key exists. For example, $if primaryKey$ PRIMARY KEYend will only ouput PRIMARY KEY if the primaryKey key value exists. The opposite is true for the unless keyword which will output the contained text when the key value does not exist in the property list.

Results of Intercalate

Issuing the command: intercalate <template file> <property list file>

```
CREATE TABLE Customer
(
     CustomerId INT PRIMARY KEY,
     FirstName VARCHAR(30),
     LastName VARCHAR(30) NOT NULL,
     Email VARCHAR(30) NOT NULL

);
```

Adding relationships

Now let's add two more tables to the database: Address and Item.

Address has a one-to-one relationship to the Customer table, explicitly making the assumption that there can be only one address for each customer.

Item, however, has a one-to-many relationship with the Customer table where there can be more than one item for each customer.

To create a many-to-many relationship, create two one-to-many relationships with an intermediate table.

Item Table

Key Type	Column name	Data type	Allows null
Primary	itemId	Int	No
	description	Varchar(30)	Yes
Foreign	customerId	Int	No

Address Table

Key type	Column name	Data type	Allows null
Primary	addressId	Int	No
	street1	Varchar(30)	No
	street2	Varchar(30)	Yes
	city	Varchar(30)	No
	state	Varchar(30)	No
	zip	Varchar(30	No
Foreign	customerId	Int	No

Relationship

Customer 1 — 1 Address

Customer 1 — * Item

Intercalate Template

Since we are using MySQL, the table type will be set to InnoDB to enforce referential integrity. Again the format of the template has been change for readability.

```
$write create.sql$
$for entities$
CREATE TABLE $tableName$
```

```
(
    $for attributes$
        $columnName$ $columnType$
        $if columnSize$($columnSize$)$end$
        $if primaryKey$ PRIMARY KEY$end$
        $if isRequired$ NOT NULL$end$
        $unless _last$,$end$
    $end$
) TYPE = InnoDB;
    $for relationships$
        $if foreignKey$
            ALTER TABLE $tableName$ ADD ($columnName$ $columnType$);
            ALTER TABLE $tableName$ ADD INDEX ($columnName$);
            ALTER TABLE $tableName$ ADD FOREIGN KEY ($columnName$)
                    REFERENCES $referenceTable$ ($referenceKey$);
        $end$
    $end$
$end$
```

We have added a new "TYPE" statement to our create SQL script template so that we can use the InnoDB table type. This gives us transaction support. By adding this to the template, all of our tables will now use this special table type. As you have seen before, we could add a key to our plist and an if to our template to control this on a table by table basis.

We have also added support for relationships. The foreign key is added, made an index and linked to the parent table using the script contained in the $if foreign key$ ... end keywords.

Next, let's take a look at how the relationships are described in the property list.

PROPERTY LIST

```
{
    tableName = "$entityName$";
    columnName = "$attributeName(capitalize)$";
    referenceTable = "$targetEntityName$";
    referenceKey = "$attributeName(capitalize)$";
    entities = (
    {
        entityName = "Customer";
        attributes = (
        {
            primaryKey = "";
            attributeName = "customerId";
            columnType = "INT";
        },
        {
            attributeName = "firstName";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "lastName";
            columnType = "VARCHAR";
```

```
            columnSize = "30";
        },
        {
            attributeName = "email";
            columnType = "VARCHAR";
            columnSize = "30";
        });
    },
    {
        entityName = "Item";
        attributes = (
        {
            primaryKey = "";
            attributeName = "itemId";
            columnType = "INT";
        },
        {
            attributeName = "description";
            columnType = "VARCHAR";
            columnSize = "30";
        });
        relationships = (
        {
            foreignKey = "";
            targetEntityName = "Customer";
            attributeName = "customerId";
            columnType = "INT";
        });
    },
    {
        entityName = "Address";
        attributes = (
        {
            primaryKey = "";
            attributeName = "addressId";
            columnType = "INT";
        },
        {
            attributeName = "street1";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "street2";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
```

```
            attributeName = "city";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "state";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "zip";
            columnType = "VARCHAR";
            columnSize = "30";
        });
        relationships = (
        {
            foreignKey = "";
            targetEntityName = "Customer";
            attributeName = "customerId";
            columnType = "INT";
        });
    });
}
```

The relationship information is added just like we had the attribute information - a named vector in an entity. The relationship is defined for the table that contains the foreign key. The target table, foreign key column and type is given.

Results of Intercalate

Based on this template and property list, here are the results:

```
CREATE TABLE Customer
(
    CustomerId INT PRIMARY KEY,
    FirstName VARCHAR(30),
    LastName VARCHAR(30) NOT NULL,
    Email VARCHAR(30) NOT NULL

) TYPE = InnoDB;


CREATE TABLE Item
(
    ItemId INT PRIMARY KEY,
    Description VARCHAR(30) NOT NULL

) TYPE = InnoDB;
ALTER TABLE Item ADD (CustomerId INT);
ALTER TABLE Item ADD INDEX (CustomerId);
ALTER TABLE Item ADD FOREIGN KEY (CustomerId) REFERENCES Customer (CustomerId);
```

```
CREATE TABLE Address
(
     AddressId INT PRIMARY KEY,
     Street1 VARCHAR(30) NOT NULL,
     Street2 VARCHAR(30),
     City VARCHAR(30) NOT NULL,
     State VARCHAR(30) NOT NULL,
     Zip VARCHAR(30) NOT NULL

) TYPE = InnoDB;
ALTER TABLE Address ADD (CustomerId INT);
ALTER TABLE Address ADD INDEX (CustomerId);
ALTER TABLE Address ADD FOREIGN KEY (CustomerId) REFERENCES Customer (CustomerId);
```

Configuring JBoss for MySQL

The MySQL JDBC driver may be downloaded from the MySQL website. The jar file mysql-connector-java-2.0.x-bin.jar should be copied to the <jboss-3.0.x install>/server/default/lib directory. A sample configuration file for MySQL, mysql-service.xml, is located in the <jboss-3.0.x install>/docs/example/jca directory. Modify the configuration parameters ConnectionURL, UserName, and Password appropriate for you MySQL installation and copy the file mysql-services.xml to the <jboss-3.0.x install>/server/default/deploy directory. This should deploy the MySQL datasource connection pool.

Entity Bean with CMP 2.0

There are a number of different techniques to create persistent objects, Java Data Objects (JDO) such as Castor, Sun's alternative JDO specification, Data Access Object (DAO) such as jRelationalFramework, Object to Relational Mapping such as CocoBase or TopLink and/or Entity Beans. We will be discussing the Entity Bean solution using CMP 2.0 since it appears to becoming the industry standard and CMP 2.0 has address many of the issues with the original CMP 1.0. The DAO solution using jRelationalFramework will be available as part of the Intercalate download.

We will use Intercalate to create the Entity Bean class. We will add the required information to the class definitions for XDoclet, so that it can generate the supporting interface classes and XML descriptor files.

INTERCALATE TEMPLATE

```
$for entities$
$write ejbBeanClass.java$
package $ejbPackage$;

/**
 * @author Mark Lui
 *
 * @ejb:bean
 *     type="CMP"
 *     cmp-version="2.x"
 *     name="$ejbName$"
 *     jndi-name="$jndiName$"
 *     local-jndi-name="$localJndiName$"
 *     view-type="both"
 *
 * @ejb:util
 *       generate="physical"
 *
```

```
 * @jboss:table-name $tableName$
 * @jboss:create-table "false"
 */
public abstract class $ejbBeanClass$ implements javax.ejb.EntityBean
{

    /**
    * Context set by container
    */
    private javax.ejb.EntityContext _entityContext;

    $for attributes$
    /**
    * Returns the $attributeName$
    *
    * @return the $attributeName$
    *
    * @ejb:persistent-field
    * $if primaryKey$@ejb:pk-field$end$
    *
    * @jboss:column-name $columnName$
    */
    public abstract $ivarType$ get$ivarAccessor$();

    /**
    * Sets the $attributeName$
    *
    * @param $attributeName$ the new $attributeName$
    */
    public abstract void set$ivarAccessor$($ivarType$ $attributeName$);
    $end$
    $for relationships$$if toMany$
    /**
    * Returns a collection of local $targetEjbName$
    *
    * @return a collection of local $targetEjbName$
    *
    * @ejb:interface-method view-type="local"
    * @ejb:relation
    *       name = "$ejbName$-$targetEjbName$"
    *       role-name = "$ejbName$-has-$targetEjbName$"
    *
    */
    public abstract java.util.Collection get$targetIvarAccessor$();

    /**
    * Sets a collection of local $targetEjbName$
    *
```

```
     * @param $targetEntityVariable$ a collection of local $targetEjbName$
     *
     * @ejb:interface-method view-type="local"
     */
    public abstract void set$targetIvarAccessor$(java.util.Collections
$targetEntityVariable$);

    /**
     * Adds to collection of local $targetEjbName$
     *
     * @param $targetEjbDataObject$ data value object of $targetEjbName$
     *
     * @ejb:interface-method view-type="both"
     */
    public void add$targetIvarAccessor$($targetEjbValueData$ $targetEjbDataObject$)
            throws javax.naming.NamingException, javax.ejb.CreateException
    {
        $targetEjbLocalHome$ $targetEjbHomeObject$ =
                ($targetEjbLocalHome$)$targetEjbUtil$.getLocalHome();
        $targetEjbLocalBeanClass$ $targetEjbBeanObject$ =
                $targetEjbHomeObject$.create($targetEjbDataObject$);
        this.get$targetIvarAccessor$().add($targetEjbBeanObject$);
    }
    $end$$if foreignKey$
    /**
     * Returns a local $targetEjbName$
     *
     * @return a local $targetEjbName$
     *
     * @ejb:interface-method view-type="local"
     *
     * @ejb:relation
     *     name="$targetEjbName$-$ejbName$"
     *     role-name="$ejbName$-has-$targetEjbName$"
     *
     * @jboss:relation
     *       fk-constraint = "true"
     *       fk-column = "$targetColumnName$"
     *       related-pk-field = "$targetAttributeName$"
     */
    public abstract $targetLocalInterface$ get$targetIvarAccessor$();

    /**
     * Sets a local $targetEjbName$
     *
     * @param languageCode a local $targetEjbName$
     *
     * @ejb:interface-method view-type="local"
```

```
     */
    public abstract void set$targetIvarAccessor$($targetLocalInterface$
$targetEntityVariable$);
    $end$$unless foreignKey$$unless toMany$
    /**
    * Returns a local $targetEjbName$
    *
    * @return a local $targetEjbName$
    *
    * @ejb:interface-method view-type="local"
    * @ejb:relation
    *       name = "$ejbName$-$targetEjbName$"
    *       role-name = "$ejbName$-has-$targetEjbName$"
    *
    */
    public abstract $targetLocalInterface$ get$targetIvarAccessor$();

    /**
    * Sets a local $targetEjbName$
    *
    * @param myOrders a local $targetEjbName$
    *
    * @ejb:interface-method view-type="local"
    */
    public abstract void set$targetIvarAccessor$($targetLocalInterface$
$targetEntityVariable$);

    /**
    * Sets local $targetEjbName$
    *
    * @param $targetEjbDataObject$ data value object of $targetEjbName$
    *
    * @ejb:interface-method view-type="both"
    */
    public void set$targetIvarAccessor$($targetEjbValueData$ $targetEjbDataObject$)
            throws javax.naming.NamingException, javax.ejb.CreateException
    {
        $targetEjbLocalHome$ $targetEjbHomeObject$ =
                ($targetEjbLocalHome$)$targetEjbUtil$.getLocalHome();
        $targetEjbLocalBeanClass$ $targetEjbBeanObject$ =
                $targetEjbHomeObject$.create($targetEjbDataObject$);
        this.set$targetIvarAccessor$($targetEjbBeanObject$);
    }
    $end$$end$$end$
    /**
    * Gets a data object representing this instance
    *
    * @return a data object representing this instance
    *
```

```
     * @ejb:interface-method view-type="both"
     */
     public $ejbValueData$ getData()
     {
       // We don't implement it here. XDoclet will create a subclass that implements it
properly.
       return null;
     }

     /**
     * Sets a data object representing this instance
     *
     * @param data a data object holding new values
     *
     * @ejb:interface-method view-type="both"
     */
     public void setData($ejbValueData$ data)
     {
       // We don't implement it here. XDoclet will create a subclass that implements it
properly.
     }

     /**
     * When the client invokes a create method, the EJB container invokes the ejbCreate
method.
     * Typically, an ejbCreate method in an entity bean performs the following tasks:
     *
     * Inserts the entity state into the database.
     * Initializes the instance variables.
     * Returns the primary key.
     *
     *
     * @param data the data object used to initialise the new instance
     * @return the primary key of the new instance
     *
     * @ejb:create-method
     */
     public $ejbPK$ ejbCreate($ejbValueData$ data) throws javax.ejb.CreateException
     {
       $for attributes$$if primaryKey$set$ivarAccessor$(data.get$ivarAccessor$());$end$$e
nd$
       setData(data);
       // EJB 2.0 spec says return null for CMP ejbCreate methods.
       return null;
     }

     /**
     * The container invokes thos method immediately after it calls ejbCreate.
     *
```

```
     * @param data the data object used to initialise the new instance
     */
     public void ejbPostCreate($ejbValueData$ data) throws javax.ejb.CreateException
     {
     }

     // Implementation of the javax.ejb.EntityBean interface methods

     /**
     * The container invokes setEntityContext just once - when it creates the bean
instance.
     */
     public void setEntityContext(javax.ejb.EntityContext entityContext)
     {
       _entityContext = entityContext;
     }

     /**
     * At the end of the life cycle, the container removes the instance from
     * the pool and invokes this method.
     */
     public void unsetEntityContext()
     {
       _entityContext = null;
     }

     /**
     * The container invokes this method to instruct the instance to synchronize its
     * state by loading it state from the underlying database.
     */
     public void ejbLoad()
     {
     }

     /**
     * The container invokes this method when the instance is taken out of the pool of
     * available instances to become associated with a specific EJB object.
     */
     public void ejbActivate()
     {
     }

     /**
     * The container invokes this method on an instance before the instance becomes
     * disassociated with a specific EJB object.
     */
     public void ejbPassivate()
     {
```

```
        }

        /**
        * The container invokes this method before it removes the EJB object that is
        * currently associated with the instance.
        */
        public void ejbRemove() throws javax.ejb.RemoveException
        {
        }

        /**
        * The container invokes this method to instruct the instance to synchronize its
        * state by storing it to the underlying database.
        */
        public void ejbStore()
        {
        }
}
$end$
```

PROPERTY LIST

```
{
        null = "";
        tableName = "$entityName$";
        columnName = "$attributeName(capitalize)$";
        targetColumnName = "$targetAttributeName(capitalize)$";
        referenceTable = "$targetEntityName$";
        referenceKey = "$attributeName(capitalize)$";
        ivarType = "$mysql2java.columnType$";
        ivarAccessor = "$attributeName(capitalize)$";
        targetIvarAccessor = "$targetEntityName$";
        ejbPackage = "$packageName$.ejb$null$";
        interfacePackage = "$packageName$.interfaces$null$";
        ejbName = "$entityName$";
        targetEjbName = "$targetEntityName$";
        targetLocalInterface = "$interfacePackage$.$targetEjbName$Local$null$";
        ejbBeanClass = "$ejbName$Bean$null$";
        targetEjbLocalBeanClass = "$interfacePackage$.$targetEjbName$Local$null$";
        targetEjbLocalHome = "$interfacePackage$.$targetEjbName$LocalHome$null$";
        targetEjbUtil = "$interfacePackage$.$targetEjbName$Util$null$";
        targetEjbHomeObject = "$targetEntityName(lowercase)$Home$null$";
        targetEjbBeanObject = "$targetEntityName(lowercase)$";
        jndiName = "$interfacePackage$.$ejbName$";
        localJndiName = "$interfacePackage$.$ejbName$Local$null$";
        ejbValueData = "$interfacePackage$.$ejbName$Data$null$";
        targetEjbValueData = "$interfacePackage$.$targetEjbName$Data$null$";
        targetEjbDataObject = "$targetEntityName(lowercase)$Data$null$";
        ejbPK = "$interfacePackage$.$ejbName$PK$null$";
        mysql2java = {
```

```
        VARCHAR = "String";
        INT = "Integer";
    };
    packageName = "com.alodar.example";
    entities = (
    {
        entityName = "Customer";
        attributes = (
        {
            primaryKey = "";
            attributeName = "customerId";
            columnType = "INT";
        },
        {
            attributeName = "firstName";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "lastName";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "email";
            columnType = "VARCHAR";
            columnSize = "30";
        });
        relationships = (
        {
            toMany = "";
            targetEntityName = "Item";
            targetEntityVariable = "item";
            targetAttributeName = "customerId";
            attributeName = "customerId";
        },
        {
            targetEntityName = "Address";
            targetEntityVariable = "address";
            targetAttributeName = "customerId";
            attributeName = "customerId";
        });
    },
    {
        entityName = "Item";
        attributes = (
        {
            primaryKey = "";
```

```
            attributeName = "itemId";
            columnType = "INT";
        },
        {
            attributeName = "description";
            columnType = "VARCHAR";
            columnSize = "30";
        });
        relationships = (
        {
            foreignKey = "";
            targetEntityName = "Customer";
            targetEntityVariable = "customer";
            targetAttributeName = "customerId";
            attributeName = "customerId";
            columnType = "INT";
        });
    },
    {
        entityName = "Address";
        attributes = (
        {
            primaryKey = "";
            attributeName = "addressId";
            columnType = "INT";
        },
        {
            attributeName = "street1";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "street2";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "city";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "state";
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "zip";
```

```
            columnType = "VARCHAR";
            columnSize = "30";
        });
        relationships = (
        {
            foreignKey = "";
            targetEntityName = "Customer";
            targetEntityVariable = "customer";
            targetAttributeName = "customerId";
            attributeName = "customerId";
            columnType = "INT";
        });
    });
}
```

RESULTS OF INTERCALATE

The resultant source code after running Intercalate and XDoclet may be found a the link below.
Source code for Bean and Interfaces classes

Session Beans

Let's move on to the session bean classes that will support the business logic of our example J2EE application. As an example, this bean class will only support creating, editing, and deleting the persistent entity beans described above. Of course, this example may be extented to more complex business operations.

INTERCALATE TEMPLATE

```
$for entities$
$write ejbBusinessClass.java$
package $ejbPackage$;

/**
 * @author Mark Lui
 *
 * @ejb:bean
 *    type="Stateless"
 *    name="$ejbBusinessClassName$"
 *    jndi-name="$jndiNameBO$"
 *    local-jndi-name="$localJndiNameBO$"
 *    view-type="both"
 *
 * @ejb:util
 *       generate="physical"
 *
 */
public class $ejbBusinessClass$ implements javax.ejb.SessionBean
{

    /**
    * get $ejbName$ data object
    *
    * @param key - primary key
```

```
*
* @ejb:interface-method view-type="both"
*/
public $ejbValueData$ get(int key) throws $ejbBusinessException$
{

    try
    {

        $ejbLocalHome$ $ejbHomeObject$ =
                ($ejbLocalHome$)$ejbUtil$.getLocalHome();
        $ejbLocalBeanClass$ $ejbBeanObject$ =
                $ejbHomeObject$.findByPrimaryKey(new $ejbPK$(new Integer(key)));

        return $ejbBeanObject$.getData();

    }
    catch(Exception e)
    {
        throw new $ejbBusinessException$(e.getMessage());
    }
}

/**
* creates new $ejbName$
*
* @param $ejbDataObject$ - data object
*
* @ejb:interface-method view-type="both"
*/
public void create($ejbValueData$ $ejbDataObject$)
        throws $ejbBusinessException$
{

    try
    {

        $ejbLocalHome$ $ejbHomeObject$ =
                ($ejbLocalHome$)$ejbUtil$.getLocalHome();
        $ejbHomeObject$.create($ejbDataObject$);

    }
    catch(Exception e)
    {
        throw new $ejbBusinessException$(e.getMessage());
    }

}
```

```
$for relationships$$if toMany$
/**
* Adds $targetEjbName$
*
* @param $targetEjbDataObject$ - data object
* @param key - $ejbName$ primary key
*
* @ejb:interface-method view-type="both"
*/
public void add$targetIvarAccessor$($targetEjbValueData$ $targetEjbDataObject$,
        int key) throws $ejbBusinessException$
{
    try
    {
        $ejbLocalHome$ $ejbHomeObject$ =
                ($ejbLocalHome$)$ejbUtil$.getLocalHome();
        $ejbLocalBeanClass$ $ejbBeanObject$ =
                $ejbHomeObject$.findByPrimaryKey(new $ejbPK$(new Integer(key)));
        $ejbBeanObject$.add$targetIvarAccessor$($targetEjbDataObject$);
    }
    catch(Exception e)
    {
        throw new $ejbBusinessException$(e.getMessage());
    }

}
$end$$unless foreignKey$$unless toMany$
/**
* Sets $targetEjbName$
*
* @param $targetEjbDataObject$ - data object
* @param key - $ejbName$ primary key
*
* @ejb:interface-method view-type="both"
*/
public void set$targetIvarAccessor$($targetEjbValueData$ $targetEjbDataObject$,
        int key) throws $ejbBusinessException$
{
    try
    {
        $ejbLocalHome$ $ejbHomeObject$ =
                ($ejbLocalHome$)$ejbUtil$.getLocalHome();
        $ejbLocalBeanClass$ $ejbBeanObject$ =
                $ejbHomeObject$.findByPrimaryKey(new $ejbPK$(new Integer(key)));
        $ejbBeanObject$.set$targetIvarAccessor$($targetEjbDataObject$);
    }
    catch(Exception e)
    {
```

```
            throw new $ejbBusinessException$(e.getMessage());
        }

    }
    $end$$end$$end$
    /**
    * updates $ejbName$
    *
    * @param $ejbDataObject$ - data object
    *
    * @ejb:interface-method view-type="both"
    */
    public void update($ejbValueData$ $ejbDataObject$)
            throws $ejbBusinessException$
    {

        try
        {

            $ejbLocalHome$ $ejbHomeObject$ =
                    ($ejbLocalHome$)$ejbUtil$.getLocalHome();
            $ejbLocalBeanClass$ $ejbBeanObject$ =
                    $ejbHomeObject$.findByPrimaryKey($for attributes$$if primaryKey$new
$ejbPK$($ejbDataObject$.get$ivarAccessor$()));$end$$end$
            $ejbBeanObject$.setData($ejbDataObject$);
        }
        catch(Exception e)
        {

            throw new $ejbBusinessException$(e.getMessage());
        }

    }

    /**
    * The container invokes this method when the instance is taken out of the pool of
    * available instances to become associated with a specific EJB object.
    */
    public void ejbActivate()
    {
    }

    /**
    * The container invokes this method on an instance before the instance becomes
    * disassociated with a specific EJB object.
    */
    public void ejbPassivate()
    {
```

```
    }

    /**
    * The container invokes this method before it removes the EJB object that is
    * currently associated with the instance.
    */
    public void ejbRemove()
    {
    }

    /**
    * The container invokes setSessionContext just once - when it creates the bean
instance.
    */
    public void setSessionContext(javax.ejb.SessionContext sessionContext) {}
}
$end$
```

PROPERTY LIST

```
{
    null = "";
    tableName = "$entityName$";
    columnName = "$attributeName(capitalize)$";
    targetColumnName = "$targetAttributeName(capitalize)$";
    referenceTable = "$targetEntityName$";
    referenceKey = "$attributeName(capitalize)$";
    ivarType = "$mysql2java.columnType$";
    ivarAccessor = "$attributeName(capitalize)$";
    targetIvarAccessor = "$targetEntityName$";
    ejbPackage = "$packageName$.ejb$null$";
    interfacePackage = "$packageName$.interfaces$null$";
    ejbName = "$entityName$";
    targetEjbName = "$targetEntityName$";
    targetLocalInterface = "$interfacePackage$.$targetEjbName$Local$null$";
    ejbBeanClass = "$ejbName$Bean$null$";
    ejbLocalBeanClass = "$interfacePackage$.$ejbName$Local$null$";
    ejbBusinessClass = "$ejbName$BOBean$null$";
    ejbBusinessClassName = "$ejbName$BO$null$";
    ejbBusinessException = "$ejbName$BOException$null$";
    ejbLocalHome = "$interfacePackage$.$ejbName$LocalHome$null$";
    ejbUtil = "$interfacePackage$.$ejbName$Util$null$";
    ejbHomeObject = "$entityName(lowercase)$Home$null$";
    ejbBeanObject = "$entityName(lowercase)$";
    jndiName = "$interfacePackage$.$ejbName$";
    localJndiName = "$interfacePackage$.$ejbName$Local$null$";
    jndiNameBO = "$interfacePackage$.$ejbName$BO$null$";
    localJndiNameBO = "$interfacePackage$.$ejbName$BOLocal$null$";
    ejbValueData = "$interfacePackage$.$ejbName$Data$null$";
```

```
targetEjbValueData = "$interfacePackage$.$targetEjbName$Data$null$";
ejbDataObject = "$entityName(lowercase)$Data$null$";
targetEjbDataObject = "$targetEntityName(lowercase)$Data$null$";
ejbPK = "$interfacePackage$.$ejbName$PK$null$";
mysql2java = {
    VARCHAR = "String";
    INT = "Integer";
};
packageName = "com.alodar.example";
entities = (
{
    entityName = "Customer";
    attributes = (
    {
        primaryKey = "";
        attributeName = "customerId";
        columnType = "INT";
    },
    {
        attributeName = "firstName";
        columnType = "VARCHAR";
        columnSize = "30";
    },
    {
        attributeName = "lastName";
        columnType = "VARCHAR";
        columnSize = "30";
    },
    {
        attributeName = "email";
        columnType = "VARCHAR";
        columnSize = "30";
    });
    relationships = (
    {
        toMany = "";
        targetEntityName = "Item";
        targetEntityVariable = "item";
        targetAttributeName = "customerId";
        attributeName = "customerId";
    },
    {
        targetEntityName = "Address";
        targetEntityVariable = "address";
        targetAttributeName = "customerId";
        attributeName = "customerId";
    });
},
```

```
        {
            entityName = "Item";
            attributes = (
            {
                primaryKey = "";
                attributeName = "itemId";
                columnType = "INT";
            },
            {
                attributeName = "description";
                columnType = "VARCHAR";
                columnSize = "30";
            });
            relationships = (
            {
                foreignKey = "";
                targetEntityName = "Customer";
                targetEntityVariable = "customer";
                targetAttributeName = "customerId";
                attributeName = "customerId";
                columnType = "INT";
            });
        },
        {
            entityName = "Address";
            attributes = (
            {
                primaryKey = "";
                attributeName = "addressId";
                columnType = "INT";
            },
            {
                attributeName = "street1";
                columnType = "VARCHAR";
                columnSize = "30";
            },
            {
                attributeName = "street2";
                columnType = "VARCHAR";
                columnSize = "30";
            },
            {
                attributeName = "city";
                columnType = "VARCHAR";
                columnSize = "30";
            },
            {
                attributeName = "state";
```

```
            columnType = "VARCHAR";
            columnSize = "30";
        },
        {
            attributeName = "zip";
            columnType = "VARCHAR";
            columnSize = "30";
        });
        relationships = (
        {
            foreignKey = "";
            targetEntityName = "Customer";
            targetEntityVariable = "customer";
            targetAttributeName = "customerId";
            attributeName = "customerId";
            columnType = "INT";
        });
    });
}
```

RESULTS OF INTERCALATE

The resultant source code after running Intercalate and XDoclet may be found a the link below.

Source code for Bean and Interfaces classes

Web Front End using Struts

We will be using the Jakarta open source MVC framework Struts for the web front end of this J2EE example. The current example only supports creating a customer, address, or item. More operations will be added in the future.

INTERCALATE TEMPLATE FOR ACTION

```
$for entities$
$write action.java$
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionServlet;
import org.apache.struts.util.PropertyUtils;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;

import java.io.IOException;


public class $action$ extends Action
{
```

```
    public ActionForward perform(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException
    {

        if(isCancelled(request))
        {
            return mapping.findForward("cancel");
        }

        try
        {

            String action = request.getParameter("action");

            String userString = request.getRemoteUser();

            $ejbBusinessHome$ $ejbBusinessHomeObject$ =
                    ($ejbBusinessHome$)$ejbBusinessUtil$.getHome();

            $ejbBusiness$ $ejbBusinessObject$ = $ejbBusinessHomeObject$.create();

            if(action.equals("create"))
            {
                $ejbValueData$ $ejbDataObject$ =
                        new $ejbValueData$();
                PropertyUtils.copyProperties($ejbDataObject$, form);
                $ejbBusinessObject$.create(($ejbDataObject$));
            }
            else if(action.equals("edit"))
            {
                $ejbValueData$ $ejbDataObject$ =
                        $ejbBusinessObject$.get(Integer.parseInt(request.
getParameter("key")));
                $actionForm$ $actionFormObject$ = new $actionForm$();
                PropertyUtils.copyProperties(form, $ejbDataObject$);
                request.setAttribute("$actionForm$", $actionFormObject$);
            }
            else if(action.equals("update"))
            {
                $ejbValueData$ $ejbDataObject$ =
                        new $ejbValueData$();
                PropertyUtils.copyProperties($ejbDataObject$, form);
                $ejbBusinessObject$.update($ejbDataObject$);
            }

            return mapping.findForward("success");
```

```
            }
            catch(Throwable e)
            {
                System.out.println(e.getMessage());
                request.setAttribute("error", e.getMessage());
                return mapping.findForward("failure");
            }

        }

}
$end$
```

INTERCALATE TEMPLATE FOR FORM

```
$for entities$
$write actionForm.java$
import javax.servlet.http.HttpServletRequest;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionMapping;

import java.sql.Timestamp;


public final class $actionForm$ extends ActionForm
{

    /**
     * The maintenance action we are performing (Create or Edit).
     */
    private String action;
    $for attributes$
    private $ivarType$ $attributeName$;$end$

    public $actionForm$()
    {
        action = "Create";
        $for attributes$
        $attributeName$ = null;$end$
    }

    /**
     * Return the maintenance action.
     */
    public String getAction()
    {
```

```
    return (this.action);
    }

    /**
     * Set the maintenance action.
     *
     * @param action The new maintenance action.
     */
    public void setAction(String action)
    {
        this.action = action;
    }

    $for attributes$
    /**
     * Return the $attributeName$.
     */
    public $ivarType$ get$ivarAccessor$()
    {
    return $attributeName$;
    }

    /**
     * Set the $attributeName$.
     *
     * @param $attributeName$ The new $attributeName$
     */
    public void set$ivarAccessor$($ivarType$ $attributeName$)
    {
        this.$attributeName$ = $attributeName$;
    }
    $end$

    /**
     * Reset all properties to their default values.
     *
     * @param mapping The mapping used to select this instance
     * @param request The servlet request we are processing
     */
    public void reset(ActionMapping mapping, HttpServletRequest request)
    {
        action = "Create";
        $for attributes$
        $attributeName$ = null;$end$
    }

    /**
     * Validate the properties that have been set from this HTTP request,
```

```
         * and return an ActionErrors object that encapsulates any
         * validation errors that have been found.  If no errors are found, return
         * null or an ActionErrors object with no
         * recorded error messages.
         *
         * @param mapping The mapping used to select this instance
         * @param request The servlet request we are processing
         */
        public ActionErrors validate(ActionMapping mapping,
                                     HttpServletRequest request)
        {
            ActionErrors errors = new ActionErrors();
            return errors;
        }

}
$end$
```

PROPERTY LIST

```
{
    null = "";
    tableName = "$entityName$";
    columnName = "$attributeName(capitalize)$";
    targetColumnName = "$targetAttributeName(capitalize)$";
    referenceTable = "$targetEntityName$";
    referenceKey = "$attributeName(capitalize)$";
    ivarType = "$mysql2java.columnType$";
    ivarAccessor = "$attributeName(capitalize)$";
    targetIvarAccessor = "$targetEntityName$";
    ejbPackage = "$packageName$.ejb$null$";
    interfacePackage = "$packageName$.interfaces$null$";
    ejbName = "$entityName$";
    targetEjbName = "$targetEntityName$";
    targetLocalInterface = "$interfacePackage$.$targetEjbName$Local$null$";
    ejbBeanClass = "$ejbName$Bean$null$";
    ejbLocalBeanClass = "$interfacePackage$.$ejbName$Local$null$";
    targetEjbLocalBeanClass = "$interfacePackage$.$targetEjbName$Local$null$";
    ejbBusinessClass = "$ejbName$BOBean$null$";
    ejbBusinessClassName = "$ejbName$BO$null$";
    ejbBusinessException = "$ejbName$BOException$null$";
    ejbLocalHome = "$interfacePackage$.$ejbName$LocalHome$null$";
    targetEjbLocalHome = "$interfacePackage$.$targetEjbName$LocalHome$null$";
    ejbUtil = "$interfacePackage$.$ejbName$Util$null$";
    targetEjbUtil = "$interfacePackage$.$targetEjbName$Util$null$";
    ejbHomeObject = "$entityName(lowercase)$Home$null$";
    targetEjbHomeObject = "$targetEntityName(lowercase)$Home$null$";
    ejbBeanObject = "$entityName(lowercase)$";
    targetEjbBeanObject = "$targetEntityName(lowercase)$";
    jndiName = "$interfacePackage$.$ejbName$";
```

```
localJndiName = "$interfacePackage$.$ejbName$Local$null$";
jndiNameBO = "$interfacePackage$.$ejbName$BO$null$";
localJndiNameBO = "$interfacePackage$.$ejbName$BOLocal$null$";
ejbValueData = "$interfacePackage$.$ejbName$Data$null$";
targetEjbValueData = "$interfacePackage$.$targetEjbName$Data$null$";
ejbDataObject = "$entityName(lowercase)$Data$null$";
targetEjbDataObject = "$targetEntityName(lowercase)$Data$null$";
ejbPK = "$interfacePackage$.$ejbName$PK$null$";
action = "$entityName$Action$null$";
ejbBusinessHome = "$interfacePackage$.$ejbName$BOHome$null$";
ejbBusinessHomeObject = "$entityName(lowercase)$BOHome$null$";
ejbBusinessUtil = "$interfacePackage$.$ejbName$BOUtil$null$";
ejbBusiness = "$interfacePackage$.$ejbName$BO$null$";
ejbBusinessObject = "$entityName(lowercase)$BO$null$";
actionForm = "$entityName$Form$null$";
actionFormObject = "$entityName(lowercase)$Form$null$";
formName = "$entityName(lowercase)$Form$null$";
formObject = "$entityName$";
objectName = "$entityName(lowercase)$";
configFile = "struts-config";
mysql2java = {
    VARCHAR = "String";
    INT = "Integer";
};
packageName = "com.alodar.example";
entities = (
{
    entityName = "Customer";
    pageTitle = "Customer Entry";
    attributes = (
    {
        primaryKey = "";
        attributeName = "customerId";
        columnType = "INT";
        attributeDetailDisplayName = "Customer ID";
    },
    {
        attributeName = "firstName";
        columnType = "VARCHAR";
        columnSize = "30";
        attributeDetailDisplayName = "First Name";
    },
    {
        attributeName = "lastName";
        columnType = "VARCHAR";
        columnSize = "30";
        attributeDetailDisplayName = "Last Name";
    },
```

```
            {
                attributeName = "email";
                columnType = "VARCHAR";
                columnSize = "30";
                attributeDetailDisplayName = "Email Address";
            });
            relationships = (
            {
                toMany = "";
                targetEntityName = "Item";
                targetEntityVariable = "item";
                targetAttributeName = "customerId";
                attributeName = "customerId";
            },
            {
                targetEntityName = "Address";
                targetEntityVariable = "address";
                targetAttributeName = "customerId";
                attributeName = "customerId";
            });
        },
        {
            entityName = "Item";
            pageTitle = "Item Input Page";
            attributes = (
            {
                primaryKey = "";
                attributeName = "itemId";
                columnType = "INT";
                attributeDetailDisplayName = "Item ID";
            },
            {
                attributeName = "description";
                columnType = "VARCHAR";
                columnSize = "30";
                attributeDetailDisplayName = "Description";
            });
            relationships = (
            {
                foreignKey = "";
                targetEntityName = "Customer";
                targetEntityVariable = "customer";
                targetAttributeName = "customerId";
                attributeName = "customerId";
                columnType = "INT";
            });
        },
        {
```

```
entityName = "Address";
pageTitle = "Address Input Page";
attributes = (
{
    primaryKey = "";
    attributeName = "addressId";
    columnType = "INT";
    attributeDetailDisplayName = "Address ID";
},
{
    attributeName = "street1";
    columnType = "VARCHAR";
    columnSize = "30";
    attributeDetailDisplayName = "Street (first line)";
},
{
    attributeName = "street2";
    columnType = "VARCHAR";
    columnSize = "30";
    attributeDetailDisplayName = "Street (second line)";
},
{
    attributeName = "city";
    columnType = "VARCHAR";
    columnSize = "30";
    attributeDetailDisplayName = "City";
},
{
    attributeName = "state";
    columnType = "VARCHAR";
    columnSize = "30";
    attributeDetailDisplayName = "State";
},
{
    attributeName = "zip";
    columnType = "VARCHAR";
    columnSize = "30";
    attributeDetailDisplayName = "Zip Code";
});
relationships = (
{
    foreignKey = "";
    targetEntityName = "Customer";
    targetEntityVariable = "customer";
    targetAttributeName = "customerId";
    attributeName = "customerId";
    columnType = "INT";
});
```

```
    });
}
```

RESULTS OF INTERCALATE

The resultant source code after running Intercalate may be found at the link below.

Source code for Action and Form classes

Adding Web Services

In order to demonstrate extending this example to web services we will separate the web front end and the business logic communicating via SOAP. The session beans will be exposed as web services using the JBoss.net plugin to the JBoss application server. JBoss.net integrates JBoss with Apache Axis.

The current release version of JBoss 3.0 includes the JBoss.net plugin. JBoss.net works out of the box using the bundled Jetty web server. The Tomcat web server requires some configuration. We will be using Jetty for this example. More information about configuring JBoss.net using Tomcat can be found at this site.

The fastest method of getting JBoss.net up and running is to start JBoss with -c all option:

run.bat (.sh) -c all.

To start JBoss.net using the default deployment directory copy the following jar files from the directory <jboss-3.0.x install>/server/all/lib to <jboss-3.0.x install>/server/default/lib:

axis.jar

tt-bytecode.jar

commons-logging.jar

wdsl4j.jar

Then copy axis-config.xml from the directory <jboss-3.0.x install>/server/all/conf to <jboss-3.0.x install>/server/default/conf

Finally copy jboss-net.sar from <jboss-3.0.x install>/server/all/deploy to <jboss-3.0.x install>/server/default/deploy. If JBoss is running, JBoss.net should deploy and be ready to use. This is one of the beauties of JBoss, hot deployment.

JBoss.net can either expose custom classes or session beans as web services. Using custom classes is similar to using Axis deployed using Tomcat as a container. The wsdd file is renamed web-services.xml and bundled with the class files in a wsr file. An example may be found at this site. However, unlike using just the Tomcat container, the web service may be hot deployed by simply copying the wsr file to the deploy directory. For this example, the session bean will be exposed as web services.

A modified version of XDoclet may be found in the JBoss CVS tree under the directory thirdparty/xdoclet. The version of XDoclet included with this example also contains further modification to the jboss-net.j template which allow the data object classes to be serialized and passed as arguments or return type in the web service calls. Using this version of XDoclet, exposing a session bean as a web service is as simple as adding the following class tag:

```
 * @jboss-net:web-service urn="$ejbBusinessClassName$"
 *                       expose-all="true"
```

where the urn is the target endpoint address and expose-all="true" exposes all methods in the web service call.

The following tag is added to the entity bean class to register the XDoclet generated data object so that the data object may be passed as an argument or as a return type in the web services call:

```
 * @jboss-net:data-object
 *       urn="$ejbDataClass$"
```

where the urn is the registered name of the bean class.

Here is the resultant web-service.xml configuration file. Note that the <service> tags for the session beans and the <typemapping> tags for the data object classes.

Intercalate Template for Action

Here is the new template for the Struts Action class incorporating the new web services support. The web server and application server may now be placed on separate systems and communicate across the Internet.

```
$for entities$
$write action.java$
import org.apache.struts.action.ActionForm;
import org.apache.struts.action.ActionForward;
import org.apache.struts.action.Action;
import org.apache.struts.action.ActionMapping;
import org.apache.struts.action.ActionErrors;
import org.apache.struts.action.ActionError;
import org.apache.struts.action.ActionServlet;
import org.apache.struts.util.PropertyUtils;

import org.apache.axis.client.Call;
import org.apache.axis.client.Service;
import org.apache.axis.encoding.XMLType;
import javax.xml.rpc.ParameterMode;
import javax.xml.rpc.namespace.QName;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.servlet.ServletException;

import java.io.IOException;


public class $action$ extends Action
{

    public ActionForward perform(ActionMapping mapping, ActionForm form,
            HttpServletRequest request, HttpServletResponse response)
            throws IOException, ServletException
    {

        if(isCancelled(request))
        {
            return mapping.findForward("cancel");
        }

        try
        {

            String action = request.getParameter("action");
```

```
            String userString = request.getRemoteUser();

            String endpoint = "http://localhost:8080/axis/services/
$ejbBusinessClassName$";

            Service  service = new Service();
            Call     call    = (Call) service.createCall();
            QName    qn      = new QName( "http://net.alodar.com/intercalate-demo",
"$ejbDataClass$" );

            call.registerTypeMapping($ejbValueData$.class, qn,
                        new org.apache.axis.encoding.ser.BeanSerializerFactory($ejbValueD
ata$.class, qn),
                        new org.apache.axis.encoding.ser.BeanDeserializerFactory($ejbValu
eData$.class, qn));

            call.setTargetEndpointAddress( new java.net.URL(endpoint) );

            if(action.equals("create"))
            {
                call.setOperationName("create");
                call.addParameter("arg0", qn, ParameterMode.PARAM_MODE_IN);
                call.setReturnType(XMLType.XSD_STRING);

                $ejbValueData$ $ejbDataObject$ =
                        new $ejbValueData$();
                PropertyUtils.copyProperties($ejbDataObject$, form);
                String ret = (String)call.invoke( new Object[] { $ejbDataObject$ } );
                if(ret != null)
                {
                    throw new Exception(ret);
                }
            }
            else if(action.equals("edit"))
            {
                call.setOperationName("get");
                call.addParameter("arg0", XMLType.XSD_INT , ParameterMode.PARAM_MODE_
IN);
                call.setReturnType(qn);

                $ejbValueData$ $ejbDataObject$ = ($ejbValueData$)call.invoke(
                        new Object[] { new Integer(request.getParameter("key")) } );
                $actionForm$ $actionFormObject$ = new $actionForm$();
                PropertyUtils.copyProperties(form, $ejbDataObject$);
                request.setAttribute("$actionForm$", $actionFormObject$);
            }
            else if(action.equals("update"))
            {
                call.setOperationName("update");
                call.addParameter("arg0", qn, ParameterMode.PARAM_MODE_IN);
```

```
                    call.setReturnType(XMLType.XSD_STRING);

                    $ejbValueData$ $ejbDataObject$ =
                            new $ejbValueData$();
                    PropertyUtils.copyProperties($ejbDataObject$, form);
                    String ret = (String)call.invoke( new Object[] { $ejbDataObject$ } );
                    if(ret != null)
                    {
                        throw new Exception(ret);
                    }
                }

                return mapping.findForward("success");


            }
            catch(Throwable e)
            {
                System.out.println(e.getMessage());
                request.setAttribute("error", e.getMessage());
                return mapping.findForward("failure");
            }

        }

}
$end$
```

PROPERTY LIST

```
{
    null = "";
    tableName = "$entityName$";
    columnName = "$attributeName(capitalize)$";
    targetColumnName = "$targetAttributeName(capitalize)$";
    referenceTable = "$targetEntityName$";
    referenceKey = "$attributeName(capitalize)$";
    ivarType = "$mysql2java.columnType$";
    ivarAccessor = "$attributeName(capitalize)$";
    targetIvarAccessor = "$targetEntityName$";
    ejbPackage = "$packageName$.ejb$null$";
    webPackage = "$packageName$";
    interfacePackage = "$packageName$.interfaces$null$";
    ejbName = "$entityName$";
    targetEjbName = "$targetEntityName$";
    targetLocalInterface = "$interfacePackage$.$targetEjbName$Local$null$";
    ejbBeanClass = "$ejbName$Bean$null$";
    ejbLocalBeanClass = "$interfacePackage$.$ejbName$Local$null$";
    targetEjbLocalBeanClass = "$interfacePackage$.$targetEjbName$Local$null$";
    ejbBusinessClass = "$ejbName$BOBean$null$";
```

```
ejbBusinessClassName = "$ejbName$BO$null$";
ejbBusinessException = "$ejbName$BOException$null$";
ejbLocalHome = "$interfacePackage$.$ejbName$LocalHome$null$";
targetEjbLocalHome = "$interfacePackage$.$targetEjbName$LocalHome$null$";
ejbUtil = "$interfacePackage$.$ejbName$Util$null$";
targetEjbUtil = "$interfacePackage$.$targetEjbName$Util$null$";
ejbHomeObject = "$entityName(lowercase)$Home$null$";
targetEjbHomeObject = "$targetEntityName(lowercase)$Home$null$";
ejbBeanObject = "$entityName(lowercase)$";
targetEjbBeanObject = "$targetEntityName(lowercase)$";
jndiName = "$interfacePackage$.$ejbName$";
localJndiName = "$interfacePackage$.$ejbName$Local$null$";
jndiNameBO = "$interfacePackage$.$ejbName$BO$null$";
localJndiNameBO = "$interfacePackage$.$ejbName$BOLocal$null$";
ejbValueData = "$interfacePackage$.$ejbName$Data$null$";
ejbDataClass = "$ejbName$Data$null$";
targetEjbValueData = "$interfacePackage$.$targetEjbName$Data$null$";
ejbDataObject = "$entityName(lowercase)$Data$null$";
targetEjbDataObject = "$targetEntityName(lowercase)$Data$null$";
ejbPK = "$interfacePackage$.$ejbName$PK$null$";
action = "$entityName$Action$null$";
ejbBusinessHome = "$interfacePackage$.$ejbName$BOHome$null$";
ejbBusinessHomeObject = "$entityName(lowercase)$BOHome$null$";
ejbBusinessUtil = "$interfacePackage$.$ejbName$BOUtil$null$";
ejbBusiness = "$interfacePackage$.$ejbName$BO$null$";
ejbBusinessObject = "$entityName(lowercase)$BO$null$";
actionForm = "$entityName$Form$null$";
actionFormObject = "$entityName(lowercase)$Form$null$";
formName = "$entityName(lowercase)$Form$null$";
formObject = "$entityName$";
objectName = "$entityName(lowercase)$";
configFile = "struts-config";
wsConfig = "web-service";
wsClass = "$entityName$WS$null$";
wsPackage = "$packageName$.ws$null$";
wsException = "$ejbName$WsException$null$";
ejbBusinessLocalHome = "$interfacePackage$.$ejbName$BOLocalHome$null$";
ejbBusinessLocalHomeObject = "$entityName(lowercase)$BOLocalHome$null$";
ejbBusinessLocal = "$interfacePackage$.$ejbName$BOLocal$null$";
ejbBusinessLocalObject = "$entityName(lowercase)$BOLocal$null$";
mysql2java = {
    VARCHAR = "String";
    INT = "Integer";
};
packageName = "com.alodar.example";
entities = (
{
    entityName = "Customer";
```

```
        pageTitle = "Customer Entry";
        attributes = (
        {
            primaryKey = "";
            attributeName = "customerId";
            columnType = "INT";
            attributeDetailDisplayName = "Customer ID";
        },
        {
            attributeName = "firstName";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "First Name";
        },
        {
            attributeName = "lastName";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "Last Name";
        },
        {
            attributeName = "email";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "Email Address";
        });
        relationships = (
        {
            toMany = "";
            targetEntityName = "Item";
            targetEntityVariable = "item";
            targetAttributeName = "customerId";
            attributeName = "customerId";
        },
        {
            targetEntityName = "Address";
            targetEntityVariable = "address";
            targetAttributeName = "customerId";
            attributeName = "customerId";
        });
    },
    {
        entityName = "Item";
        pageTitle = "Item Input Page";
        attributes = (
        {
            primaryKey = "";
            attributeName = "itemId";
```

```
            columnType = "INT";
            attributeDetailDisplayName = "Item ID";
        },
        {
            attributeName = "description";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "Description";
        });
        relationships = (
        {
            foreignKey = "";
            targetEntityName = "Customer";
            targetEntityVariable = "customer";
            targetAttributeName = "customerId";
            attributeName = "customerId";
            columnType = "INT";
        });
    },
    {
        entityName = "Address";
        pageTitle = "Address Input Page";
        attributes = (
        {
            primaryKey = "";
            attributeName = "addressId";
            columnType = "INT";
            attributeDetailDisplayName = "Address ID";
        },
        {
            attributeName = "street1";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "Street (first line)";
        },
        {
            attributeName = "street2";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "Street (second line)";
        },
        {
            attributeName = "city";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "City";
        },
        {
```

```
            attributeName = "state";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "State";
        },
        {
            attributeName = "zip";
            columnType = "VARCHAR";
            columnSize = "30";
            attributeDetailDisplayName = "Zip Code";
        });
        relationships = (
        {
            foreignKey = "";
            targetEntityName = "Customer";
            targetEntityVariable = "customer";
            targetAttributeName = "customerId";
            attributeName = "customerId";
            columnType = "INT";
        });
    });
}
```

Results of Intercalate

The resultant source code after running Intercalate on the property list and templates inlcuding web services support may be found at the link below.

Source code for Web Services version

Making modifications

This part of the example demostrates the real beauty of using Intercalate. Suppose your client decides to add a title field to the customer table. This change is as simple as adding the following to the property list.

```
entityName = "Customer";
pageTitle = "Customer Entry";
attributes = (
{
    primaryKey = "";
    attributeName = "customerId";
    columnType = "INT";
    attributeDetailDisplayName = "Customer ID";
},
        {
    attributeName = "title";
    columnType = "VARCHAR";
    columnSize = "30";
    attributeDetailDisplayName = "Title";
        },
{
    attributeName = "firstName";
    columnType = "VARCHAR";
```

```
        columnSize = "30";
        attributeDetailDisplayName = "First Name";
    },
    {
        attributeName = "lastName";
        columnType = "VARCHAR";
        columnSize = "30";
        attributeDetailDisplayName = "Last Name";
    },
    {
        attributeName = "email";
        columnType = "VARCHAR";
        columnSize = "30";
        attributeDetailDisplayName = "Email Address";
    });
```

Run the Ant process using Intercalate and the entire application is recreated including the additional field.

Results of Intercalate

The resultant source code after running Intercalate on the modified property list may be found at the link below.

Source code after modifying the property file

Download

This example requires JDK 1.3 or greater and Ant 1.5. JAVA_HOME and ANT_HOME must defined as the home directory for the JDK and Ant respectively.

Download Intercalate Application including the J2EE example here.

Running the example

Issue the ant command in the intecalate root directory. Copy the intercalate-demo.ear file to the <jboss-3.0.x install>/server/default/deploy directory and the example should deploy. The webpage for adding a customer, address, and item may be accessed using the following links respectively:

http://localhost:8080/myApp/customer.jsp

http://localhost:8080/myApp/address.jsp

http://localhost:8080/myApp/item.jsp