The Power of XDoclet

CIS 764 - Database System Design

Darren Landoll
Computing and Information Sciences
Kansas State University

2/28/2005

Table of Contents

1. Introduction

XDoclet is basically just a code generation tool. Many applications have redundant code and/or interfaces and this is where XDoclet comes into play. You can update one source file and use XDoclet to regenerate the affected files. The incredible improvements to content management are clearly obvious.

XDoclet parses source code like JavaDoc. By reading JavaDoc tags embedded in source code, XDoclet uses predefined templates to generate code based on those tags. A common use of XDoclet is to embed tags in EJB's and automatically generate all of the interfaces, beans, and XML descriptors.

This paper examines the importance of XDoclet and provides simple examples that present some of the major features.

1.1 Concepts

Primary concepts:

- Write one Java class for each major component
- Save time by using XDoclet to generate redundant code
- Extensibility – add custom modules for special applications

1.2 References

[1] XDoclet home page http://xdoclet.sourceforge.net/xdoclet/index.html
[2] Continuous Integration, Martin Fowler
http://www.martinfowler.com/articles/continuousIntegration.html
[3] Enhance J2EE component reuse with XDoclet, , IBM Developerworks
http://www.arc-mind.com/downloads_protected/tutorials/xdoclet/ws-j2x-ltr.pdf
[4] XDoclet Tutorial: A Short Tutorial on XDoclet Templates, Liu Zehua
http://www.cais.ntu.edu.sg/~liuzh/xdoclet_tutorial/
[5] Use XDoclet to generate web service support files, IBM Developerworks
http://www.arc-mind.com/downloads_protected/tutorials/xdoclet/xdoclet2.pdf

2. Benefits

The primary benefits of XDoclet include the reduction of redundant work, J2EE made easy, support for leading servers and tools, extensibility, and open, distributed development [1]. Java is already powerful due to its architecture independence, but XDoclet takes this a step further by allowing you to create applications that are virtually server independent (JBoss, Websphere, MySQL, SQL Server, etc…).

3. XDoclet and Ant

XDoclet has been developed to operate within the Ant build system. Special targets inserted into the Ant *build.xml* file invoke XDoclet to generate the desired files.

Listed below is an example of the use of XDoclet tags in Ant:

```
<target name="ejbdoclet" depends="prepare">
    <ejbdoclet
        destdir="${samples.gen-src.dir}"
        mergedir="parent-fake-to-debug"
        excludedtags="@version,@author,@todo"
        addedtags="@xdoclet-generated at ${TODAY},@copyright The XDoclet Team"
        ejbspec="2.0"
        force="${samples.xdoclet.force}"
        verbose="false"
    >
    …
    </ejbdoclet>
</target>
```

4. EJB Generation

The EJB features of XDoclet are probably the most powerful. Writing an EJB by hand requires multiple files that are all very similar, but necessary to work in the J2EE architecture. XDoclet allows you to write one file for each EJB and the rest are automatically generated for you. For example, consider that you are developing EJB's for a web application and the customer asks for another field to be added to an EJB. Instead of updating the bean, interfaces, and XML descriptors, you make a few small changes to the one file and execute Ant to regenerate the affected files.

4.1 *ejbdoclet* Task

The ejbdoclet task parses directories specified for EJB's and generates the following J2EE resources. You can pick and choose which resources you need by including those tags in the *build.xml* file.

- Remote and/or local interfaces
- Value objects
- BMP and/or CMP beans
- Struts forms
- XML descriptor files
- and many others…

5. Web Generation

In a J2EE web application, you need descriptor files that describe the static and/or dynamic HTML pages being used in the program. The *webdoclet* task performs this function and generates files like *web.xml*.

5.1 *webdoclet* Task

The *webdoclet* task generates the following types of information.

- Servlet info
- Authentication info
- Page mappings

As shown in the web example from [3], servlet mappings can be defined with JavaDoc tags inserted into the code of a servlet as follows:

```
/*
…
 * @web.servlet-mapping url-pattern="/Basic/*"
 * @web.servlet-mapping url-pattern="*.Basic"
 * @web.servlet-mapping url-pattern="/BasicServlet"
...
 */
public class BasicServlet extends HttpServlet {
```

These tags would generate the following entries in the *web.xml* file:

```
<servlet-mapping>
<servlet-name>BasicServlet</servlet-name>
<url-pattern>/Basic/*</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>BasicServlet</servlet-name>
<url-pattern>*.Basic</url-pattern>
</servlet-mapping>
<servlet-mapping>
<servlet-name>BasicServlet</servlet-name>
<url-pattern>/BasicServlet</url-pattern>
</servlet-mapping>
```

6. Extending XDoclet

You are not limited to using the default templates included in the standard release of XDoclet. Custom templates may be created to implement special features needed by an application.

6.1 Non-J2EE Example

An excellent example of using XDoclet for something other than a web application is a command line module that could be used in console applications. The example presented in [4] allows you to declare members of a class as command line arguments. It uses this information to generate code that handles the command line processing for you.

Here is an excerpt from a Java source file using the custom XDoclet template:

```
public class CLArguments
{
... ...
/**
 * @clarguments.argname longname="inputurl" shortname="u" shortdesc="input_url"
 */
protected String inputURL = null;

/**
 * @clarguments.argname
 * longname="outputtype"
 * shortname="o"
 * shortdesc="output_type"
 */
protected String outputType = null;
... ...
}
```

7. DBills Example

The DBills example (http://www.cis.ksu.edu/~dlandoll/classes/764/tutorials/dbills.zip) uses XDoclet to generate source code for several EJB's of a simple bill management system. A static web descriptor is used in this example, but XDoclet is used for generating all the EJB's.

The remainder of this paper provides an overview of how the example is configured and what XDoclet generates.

7.1 Ant *build.xml*

This Ant target tells XDoclet what to do. If you look at the detail within the *ejbdoclet* task, you can see how XDoclet knows where to find the source files and where to put the generated files. I have specified information for JBoss only, but other servers could be included as well.

```
<target name="xdoclet-generate" depends="init">
    <taskdef
      name="ejbdoclet"
      classname="xdoclet.modules.ejb.EjbDocletTask"
      classpathref="base.path"
    />
    <ejbdoclet
      destdir="${build.generate.dir}"
      excludedtags="@version,@author"
      ejbspec="${ejb.version}"
      mergedir="${src.resources.dir}/xdoclet"
      force="${xdoclet.force}"
    >
      <fileset dir="${src.ejb.dir}">
        <include name="**/*Bean.java"/>
      </fileset>
<packageSubstitution packages="session,entity" substituteWith="interfaces"/>
      <valueobject/>
      <remoteinterface/>
      <localinterface/>
      <homeinterface/>
      <localhomeinterface/>
      <entitybmp/>
      <entitycmp/>
      <session/>
      <deploymentdescriptor destdir="${build.dir}/META-INF"/>
      <jboss version="3.0"
        xmlencoding="UTF-8"
        typemapping="mySQL"
        datasource="java:/BigDDS"
        destdir="${build.dir}/META-INF"
        validateXml="false"
      />
    </ejbdoclet>
  </target>
```

7.2 Directory Structure

- build
  - classes (compiled Java classes)
  - generate (generated source files by XDoclet)
  - META-INF (generated descriptor files)
    - ejb-jar.xml
    - jbosscmp-jdbc.xml
    - jboss.xml

- src
  - etc
    - WEB-INF (static web descriptor files)
  - main
    - ejb (EJB source files)
    - servlet (Servlet source files)
    - web (JSP files)

7.3 EJB Source Code

Class header in *BillAcctBean.java*:

```
/**
 * BillAcctBean represents an account you have with a company
 *
 * @ejb.bean name="BillAcct"
 *     local-jndi-name="ejb/BillAcctBean"
 *     type="CMP"
 *     view-type="local"
 *     schema="billAcctSchema"
 *     cmp-version="2.x"
 *
 *   @ejb.pk class="java.lang.Object"
 *   @ejb.persistence
 *      table-name="bill_acct"
 *
 * @ejb.finder
 *    query="SELECT OBJECT(a) FROM billAcctSchema as a"
 *    signature="java.util.Collection findAll()"
 *
 * @ejb.value-object
 *    name="BillAcct"
 *    match="*"
 *    extends="dbills.interfaces.AbstractValue“
 **/
public abstract class BillAcctBean implements EntityBean
```

Method headers are included for each field in the database table:

```
 /**
   * Returns the billAcctDesc
   * @return the billAcctDesc
   *
   * @ejb.persistent-field
   * @ejb.persistence
   *    column-name="bill_acct_desc"
   *     sql-type="VARCHAR"
   *
   * @ejb.interface-method
   */
```

```
   public abstract java.lang.String getBillAcctDesc();

/**
   * Sets the billAcctDesc
   *
   * @param java.lang.String the new billAcctDesc value
   *
   * @ejb.interface-method
   */
   public abstract void setBillAcctDesc(java.lang.String billAcctDesc);
```

7.4 Generated Files

Listed here are the files generated for BillAcctBean.

- BillAcctBean
  - BillAcctCMP.java (complete CMP entity bean)
  - BillAcctLocalHome.java (local home interface)
  - BillAcctLocal.java (local interface)
  - BillAcctValue.java (value object)
- Descriptor Files (other EJB's are also included in these files)
  - ejb-jar.xml
  - jbosscmp-jdbc.xml
  - jboss.xml