Application Development With Velocity

Bernd Wender
Christian Stoiber
Application Development Issues

• Overview: Velocity: what is it, what is it good for?
• Part 1: Code Generators
• Part 2: Web Applications With Velocity
What is Velocity?
What is Velocity?
What is it good for?

- Source-to-Source Translation
- XML Transformation
- XML Generation
- HTML Generation
- Web Applications
- and whatever you want to do with it
Where can I get it?

• Jakarta project home: jakarta.apache.org
• Open source
• Free of charge
• Simple to install -- just add velocity.jar to your classpath
• Current version: 1.3 Release Candidate 1
• Nice documentation and examples
Related Projects

- **Struts** -- MVC framework
- **Maverick** -- MVC framework
- **Turbine** -- web application framework
- **Webwork** -- web application framework
- **vDoclet** -- in-memory "DocInfo" model
- **JPublish** -- web publishing system
- **JeeWiz!** -- J2EE infrastructure code, deployment descriptors and build jobs
- **Luxor** -- XUL Toolkit
**Part 1: Code Generation**

- **Velocity Templates** (Java classes and/or interface stubs)
- **Code Generator using Velocity**
- **XML Data**
- **Java classes and/or interfaces**
Generator Basics

(XML) Source \( \rightarrow \) (XML) Parser \( \rightarrow \) Internal Representation (object hierarchy) \( \rightarrow \) Velocity \( \rightarrow \) Velocity Context \( \rightarrow \) Java Source

- (XML) Source is read by (XML) Parser
- (XML) Parser builds Internal Representation (object hierarchy)
- Internal Representation (object hierarchy) is linked to Velocity
- Velocity is used by Velocity Context
- Velocity Context is generated by Java Source
Example: Java Beans

- Nice to have, easy to use
- Quasi standard Java components in almost all application fields
- Beans are toolable
- Beans are introspectable
- Writing Java Beans is boring
- Generating Java Beans is fun!
A Simple Bean

• Has a default constructor
• Has a couple of private attributes
• Has a public getter for some of these attributes
• Has a public setter for some of these attributes
• Follows the Java bean naming convention
/** The Circle Bean */
public class Circle
{
    /** Bean constructor */
    public Circle()

    /** Center point */
    private Point centerPoint;

    /** Radius */
    private int radius;

    /** Yields the center point */
    public Point getCenterPoint() { return centerPoint; }

    /** Yields the radius */
    public int getRadius() { return radius; }

    /** Sets the center point */
    public void setCenterPoint(Point centerPoint) { this.centerPoint = centerPoint; }

    /** Sets the radius */
    public void setRadius(int radius) { this.radius = radius; }
}

Circle Bean
/* The Circle Bean */
public class Circle
{
    /* Bean constructor */
    public Circle()

    /* Center point */
    private Point centerPoint;

    /* Radius */
    private int radius;

    /* Yields the center point */
    public Point getCenterPoint() { return centerPoint; }

    /* Yields the radius */
    public int getRadius() { return radius; }

    /* Sets the center point */
    public void setCenterPoint(Point centerPoint) { this.centerPoint = centerPoint; }

    /* Sets the radius */
    public void setRadius(int radius) { this.radius = radius; }
}
The XML Circle Bean

```xml
<?xml version="1.0"?>

<bean name="Circle"
     package="com.circles"
     comment="The Circle Bean"
     author="Bernd Wender"
>

  <attribute name="centerPoint"
             type="Point"
             comment="Center Point"
             settable="true"
             gettable="true"
  />

  <attribute name="radius"
             type="int"
             comment="Radius"
             settable="true"
             gettable="true"
  />

</bean>
```
How to write a code generator?

- Define an internal representation
- Implement an XML parser (SAX is easy!)
- Build the generator framework for
  - file handling (parsing input, writing output)
  - velocity stuff (initialization, context)
  - error handling etc.
Step 1: Internal Representation

/** Represents a (simple) Java bean */
public class Bean
{
    /** Name of the bean */
    private String name = null;

    /** Package of the bean */
    private String beanPackage = null;

    /** Comment */
    private String comment = null;

    /** Author of the bean */
    private String author = null;

    /** List of attributes */
    private Map attributes = new HashMap();

    ...
The Attribute Bean

/** This class represents a Java bean attribute */
public class Attribute
{
    /** Name of the attribute */
    private String name = null;

    /** Type of the attribute */
    private String type = null;

    /** Comment */
    private String comment = null;

    /** Gettable? */
    private boolean gettable = true;

    /** Settable? */
    private boolean settable = true;

    ...

Getters and Setters
Step 3: Generator Framework

```
try {
    Velocity.init("velocity.properties");
    // Make a context object and populate with the data.
    VelocityContext context = new VelocityContext();
    createBean();
    context.put("bean", bean);
    Template template = null; // Get the Template object.
    try { template = Velocity.getTemplate(templateFile); }
    catch( ResourceNotFoundException rnfe )
    {
        System.out.println("Example : error : cannot find template " + templateFile);
    }
    catch( ParseErrorException pee )
    {
        System.out.println("Example : Syntax error in template " + templateFile + ":" + pee);
    }
    ...
```

Our bean representation (stored using the key "bean")
// Now have the template engine process your template
BufferedWriter writer = new BufferedWriter(
    new OutputStreamWriter(System.out));
if (template != null) template.merge(context, writer);
// Flush and cleanup
writer.flush();
writer.close();

catch( Exception e ) { System.out.println(e); }

...
package $bean.package;
import java.io.*;
import java.util.ArrayList;

/**$bean.comment
 * @author $bean.author
 * @version $Revision: $
 */
public class $bean.name {
#foreach( $attribute in $bean.attributes )
/** $attribute.comment */
    private $attribute.type $attribute.name = null;
#end
...

Our bean representation (stored using the key “bean”)
The Template

package $bean.package;
import java.io.*;
import java.util.ArrayList;

/**$bean.comment
 * @author $bean.author
 * @version $Revision: $
 */
public class $bean.name
{
#foreach( $attribute in $bean.attributes )
/** $attribute.comment */
    private $attribute.type $attribute.name = null;
#end
...

Shortcut for bean.getPackage()
The Template

Velocity is aware of Java collections!

public Map getAttributes()
The Template (cont.)

Getter section

Generate getter only if gettable attribute is set

#foreach( $attribute in $bean.attributes )
#if($attribute.isGettable())
/** Yields the $attribute.comment */
public $attribute.type get$attribute.capitalizedName
{
    return $attribute.name
}
#end
#end

...
Why Not Generating XML?

Will produce XML output equivalent to the original XML source
Other Velocity Goodies

- **#set** -- setting temporary objects within a template makes life easier
- **Macros** -- can be used wherever template code can be parametrized (e.g. for GUI components)
- **Utilities**: Velocity count, simple math etc.
- **#include, #parse**
Part 2: Web Apps With Velocity

Velocity Templates (HTML, JavaScript) -> Velocity servlet -> Database App.

HTML, JavaScript (forms based)
Web App Architecture

• Velocity can easily be used as key technology for MVC web app architectures (so called “model 2“ architectures)
• Replacement for JSP
• Advantage: process much simpler, no (on-the-fly) recompilations involved -- templates can be manipulated in place
• Nice internationalization using property files
Example: HTML macro

```
#macro( input $type $name $formName $size )
  #if( $tool.render("${formName}.isMandatory('$name')") == "true" )
    #set( $class = "mandatory" )
  #else
    #set( $class = "" )
  #end

  <input type="$type"
    name="$name"
    class="$!class"
    value="$tool.render("$!${formName}.$name")"
    size= "$size" />
#end
```
Example: Cascading Macros

A submit button is a special input element

Calls input macro

```
#macro( button $name $form $resourceKey )
#set( $value = $message.get($resourceKey) )
#input( "submit" $name "" $form $value -1)
#end
```
Web App Model 2 Architecture

Controller Servlet

Http-Request

Business-LogicBean

Model

Net Data

Template

Velocity Engine

View HTML
MVC Approach

• URL-Mapping (example)
  – http:localhost:8080/mywebapp/login.vm
  – web.xml: *.vm maps to Velocity
  – login.vm → Login.java
  – Login.java is the associated business logic bean of the login page (form)
  – Velocity engine generates correspondent HTML output
Form Based Approach

- Error handling
  - Error-Bean
  - Properties
  - Internationalization
  - Action-Form XY

- User session specific data
  - User-Bean

- Populates the form with data

- Velocity Engine

- HTML/JavaScript
Web App Workflow

Properties
BLB and Form Mappings

Controller Servlet

Velocity Engine

HTTP-Request

Action-Form

NetData

Business-LogicBean

Tool

Partially generated using Velocity

BLB-Pool

HTML
Putting It All Together

- Velocity is used for Web App Development
  - as a tool for **Java code** generation based on XML descriptions of “abstract web user interfaces“ (forms, business logic beans, ...)
  - we use the same XML description for the generation of the velocity **templates**
  - we use Velocity as servlet processing tool for the generation of the actual HTML / JavaScript representation of the Web App