Reflecting on Hamlets

by
René Pawlitzek
IBM Research – Zurich
rpa@zurich.ibm.com

October 20th, 2010

1. Introduction

The Hamlets web framework is a Java-based open source system for generating dynamic web pages. It was developed with simplicity and speed in mind, is easy-to-use and easy-to-understand, and the result of a radical software simplification effort. Familiarize yourself with the older Hamlets articles (starting with "Introducing Hamlets") before you continue reading. The links to the previous Hamlet articles can be found in Resources.

This article describes a small addition to the Hamlets framework: the new ExHamletHandler class to eliminate the potentially long if-then-else statements in HamletHandlers. Less then 100 lines of Java code are necessary for its implementation. The new ExHamletHandler uses Java Reflection to improve the structure of the code which provides the dynamic content for a template. In addition, performance is increased in combination with template compilation.

2. How it works

A Hamlet is a servlet extension that reads XHTML template files containing the presentation of a web page using SAX (the Simple API for XML) and uses a HamletHandler to dynamically add content to those places in the template which are marked with special tags and IDs (see Figure 1).

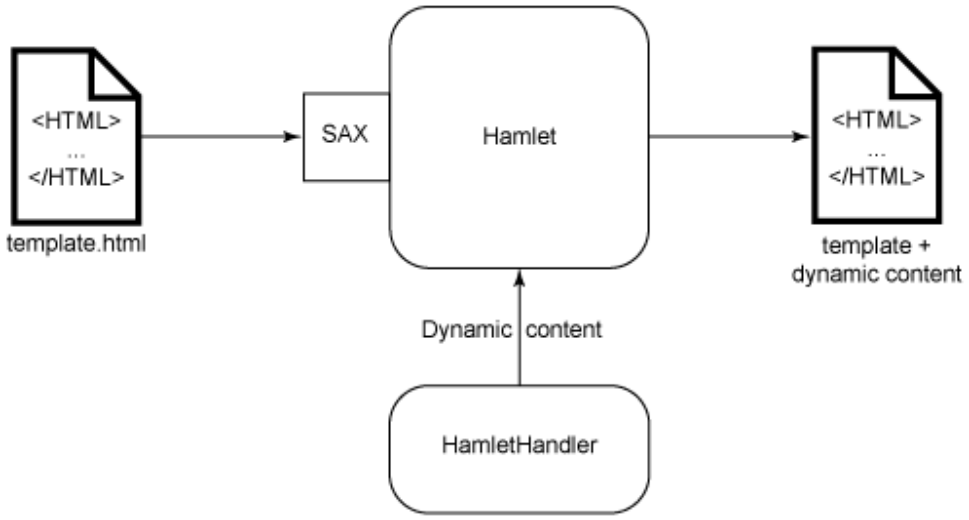

Figure 1: A Hamlet uses SAX to read content from a template file and calls a HamletHandler to add dynamic content

To provide the dynamic content for the following Guestbook template

```
<HTML>
  <HEAD>
    <TITLE>Guestbook</TITLE>
    <LINK REL="stylesheet" TYPE="text/css" HREF="GuestBook.css" MEDIA="all" />
  </HEAD>
  <BODY>
    <FORM ACTION="GuestBook.html" METHOD="POST">
      <DIV CLASS="Title">Guestbook</DIV>
      <DIV CLASS="Text">Add your comment here:</DIV>
      <DIV CLASS="Text"><TEXTAREA CLASS="entryfield" NAME="Comment"></TEXTAREA></DIV>
      <DIV CLASS="Text"><INPUT CLASS="button" TYPE="SUBMIT" VALUE="Add" /></DIV>
    </FORM>
    <DIV CLASS="Text">
      <REPEAT ID="Comments">
        <DIV CLASS="Text">Date: <REPLACE ID="Date">26.9.2007</REPLACE></DIV>
        <DIV CLASS="Text"><REPLACE ID="Comment">Hello world!</REPLACE></DIV>
        <BR/>
      </REPEAT>
    </DIV>
    <HR CLASS="Separator" />
    <INCLUDE SRC="Copyright.html"/>
  </BODY>
</HTML>
```

you extend the HamletHandler class and overwrite (in this example) two of the four callback methods getElementRepeatCount(), getElementReplacement(), getElementAttributes(), and getElementIncludeSource():

```
protected class GuestBookHandler extends HamletHandler {


  private int i = 0;
  private Comment curComment;
  private Vector<Comment> comments;
  private SimpleDateFormat format;


  public GuestBookHandler (Hamlet hamlet, Vector<Comment> comments) {
```

```
    super (hamlet);
    this.comments = comments;
    format = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
  } // GuestBookHandler


  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception {
    if (id.equals ("Comments"))
      return comments.size ();
    return 0;
  } // getElementRepeatCount


  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    if (id.equals ("Date")) {
      curComment = (Comment) comments.elementAt (i);
      return format.format (curComment.date);
    } else if (id.equals ("Comment")) {
      i++;
      return curComment.text;
    } // if
    return "?";
  } // getElementReplacement


} // GuestBookHandler
```

Both callback methods in the GuestBookHandler use an if-then-else statement to determine the dynamic processing which should happen for a particular ID. For the sequence

```
    <REPLACE ID="Comment">Hello world!</REPLACE>
```

in the template the following code is executed in the if-then-else statement of the getElementReplacement() method:

```
    if (id.equals ("Date")) {
      ...
    } else if (id.equals ("Comment")) {
      i++;
      return curComment.text;
    } // if
```

The code to produce the dynamic content is in general quite small, typically only a few lines long. Nevertheless, as shown in the example below, the if-then-else statements can become quite long and all processing is contained in a single method.

```
  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    if (id.equals ("Reviewed")) {
      return event.isReviewed () ? "Y" : "-";
    } else if (id.equals ("Reception")) {
      return dateFormat.format (event.getReceptionDate ());
    } else if (id.equals ("Generation")) {
      return dateFormat.format (event.getGenerationDate ());
    } else if (id.equals ("Signature")) {
      return event.getSignature ();
    } else if (id.equals ("Count")) {
      return "" + event.getEventCount ();
    } else if (id.equals ("SrcIP")) {
      return event.getSrcIP ();
    } else if (id.equals ("SrcPort")) {
```

```
      return "" + event.getSrcPort ();
    } else if (id.equals ("DstIP")) {
      return event.getDstIP ();
    } else if (id.equals ("DstPort")) {
      return "" + event.getDstPort ();
    } else if (id.equals ("Customer")) {
      return event.getCustomerName ();
    } else if (id.equals ("Sensor")) {
      return event.getSensorName ();
    } // if
    return "?";
  } // getElementReplacement
```

The new ExHamletHandler, which is introduced now, allows you to write the GuestBookHandler as follows:

```
protected class GuestBookHandler extends ExHamletHandler {


  private int i = 0;
  private Comment curComment;
  private Vector<Comment> comments;
  private SimpleDateFormat format;


  public GuestBookHandler (Hamlet hamlet, Vector<Comment> comments) {
    super (hamlet);
    this.comments = comments;
    format = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
  } // GuestBookHandler


  public int getCommentsRepeatCount (String id, String name, Attributes atts) {
    return comments.size ();
  } // getCommentsRepeatCount


  public String getDateReplacement (String id, String name, Attributes atts) {
    curComment = (Comment) comments.elementAt (i);
    return format.format (curComment.date);
  } // getDateReplacement


  public String getCommentReplacement (String id, String name, Attributes atts) {
    i++;
    return curComment.text;
  } // getCommentReplacement


} // GuestBookHandler
```

In the new GuestBookHandler, the two callback methods and their if-then-else statements have disappeared. Instead, they are replaced with the getCommentsRepeatCount(), getDateReplacement(), and getCommentReplacement() methods whose names match one of the following naming patterns:

```
get<ID>RepeatCount
get<ID>Replacement
get<ID>Attributes
get<ID>IncludeSource
```

As a result, the code to dynamically create the content for <REPLACE> tags is no longer exclusively contained in the getElementReplacement() callback, but distributed over a

number of get<ID>Replacement() callback methods. The same happens with getElementRepeatCount(), getElementAttributes(), and getElementIncludeSource(). Table 1 compares the old and the new GuestBookHandlers side by side.

3. How it is done

The old HamletHandler class implements the ContentHandler interface

```
public interface ContentHandler {

  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception;

  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception;

  public Attributes getElementAttributes (String id, String name, Attributes atts)
    throws Exception;

  public InputStream getElementIncludeSource (String id, String name, Attributes atts)
    throws Exception;

} // ContentHandler
```

and provides a default implementation for getElementRepeatCount(), getElementReplacement(), getElementAttributes(), and getElementIncludeSource():

```
public class HamletHandler implements ContentHandler {


  private  Hamlet  hamlet;


  public HamletHandler (Hamlet hamlet) {
    this.hamlet = hamlet;
  } // HamletHandler


  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception {
    return 0;
  } // getElementRepeatCount


  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    return "";
  } // getElementReplacement


  public Attributes getElementAttributes (String id, String name, Attributes atts)
    throws Exception {
    return atts;
  } // getElementAttributes


  public InputStream getElementIncludeSource (String id, String name, Attributes atts)
    throws Exception {
    URL url = new URL (hamlet.getIncludeURL (atts.getValue ("SRC")));
    return url.openStream ();
  } // getElementIncludeSource
```

```
} // HamletHandler
```

Dynamic content for the Guestbook template is provided by simply overwriting these default implementations in the GuestBookHandler class (see Figure 2):

```
protected class GuestBookHandler extends HamletHandler {


  private int i = 0;
  private Comment curComment;
  private Vector<Comment> comments;
  private SimpleDateFormat format;


  public GuestBookHandler (Hamlet hamlet, Vector<Comment> comments) {
    super (hamlet);
    this.comments = comments;
    format = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
  } // GuestBookHandler


  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception {
    if (id.equals ("Comments"))
      return comments.size ();
    return 0;
  } // getElementRepeatCount


  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    if (id.equals ("Date")) {
      curComment = (Comment) comments.elementAt (i);
      return format.format (curComment.date);
    } else if (id.equals ("Comment")) {
      i++;
      return curComment.text;
    } // if
    return "?";
  } // getElementReplacement


} // GuestBookHandler
```

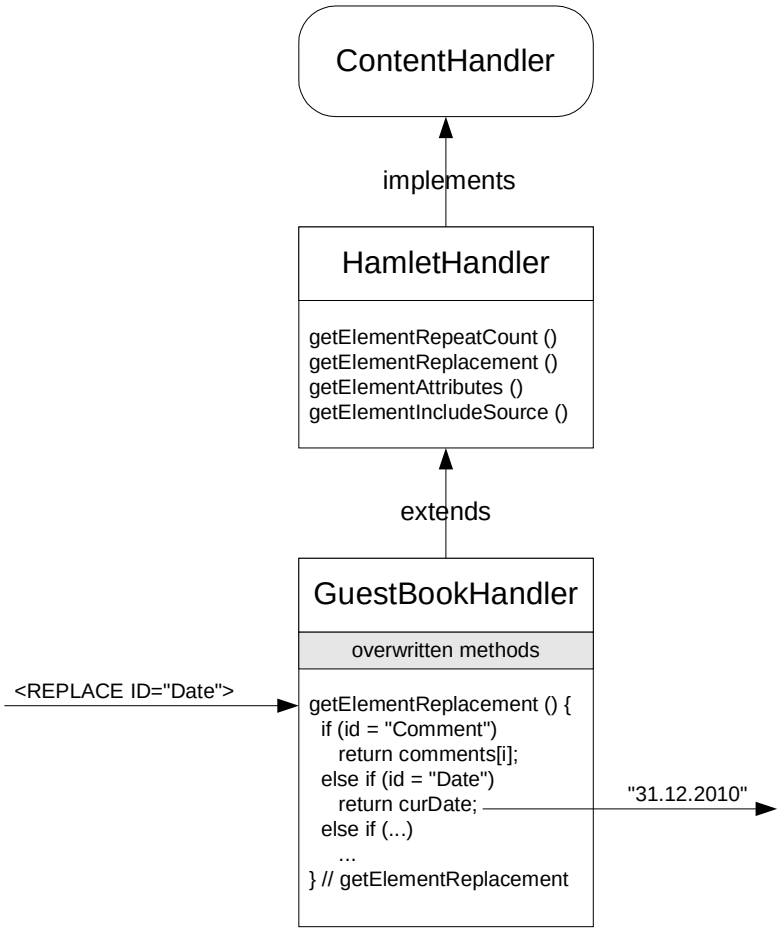

Figure 2: Providing dynamic content by overwriting callback functions

The new ExHamletHandler class also implements the ContentHandler interface and is implemented as follows:

```
public class ExHamletHandler implements ContentHandler {


  protected static  Class<?>[]  paramTypes =
    new Class<?>[] { String.class, String.class, Attributes.class };


  protected  Hamlet    hamlet;
  protected  Class<?>  c;


  public ExHamletHandler (Hamlet hamlet) {
    this.hamlet = hamlet;
    c = this.getClass ();
  } // ExHamletHandler


  public int getElementRepeatCount (String id, String name, Attributes atts)
    throws Exception {
    try {
```

```
      Method m = c.getMethod ("get" + id + "RepeatCount", paramTypes);
      Integer val = (Integer) m.invoke (this, new Object[] { id, name, atts });
      return val.intValue ();
    } catch (NoSuchMethodException e) {
      return 0;
    } // try
  } // getElementRepeatCount


  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    try {
      Method m = c.getMethod ("get" + id + "Replacement", paramTypes);
      return (String) m.invoke (this, new Object[] { id, name, atts });
    } catch (NoSuchMethodException e) {
      return "";
    } // try
  } // getElementReplacement


  public Attributes getElementAttributes (String id, String name, Attributes atts)
    throws Exception {
    try {
      Method m = c.getMethod ("get" + id + "Attributes", paramTypes);
      return (Attributes)  m.invoke (this, new Object[] { id, name, atts });
    } catch (NoSuchMethodException e) {
      return atts;
    } // try
  } // getElementAttributes


  public InputStream getElementIncludeSource (String id, String name, Attributes atts)
    throws Exception {
    try {
      if (id == null) throw new NoSuchMethodException ();
        Method m = c.getMethod ("get" + id + "IncludeSource", paramTypes)
      return (InputStream) m.invoke (this, new Object[] { id, name, atts });
    } catch (NoSuchMethodException e) {
      URL url = hamlet.getIncludeURL (atts.getValue ("SRC"));
      return url.openStream ();
    } // try
  } // getElementIncludeSource


} // ExHamletHandler
```

Contrary to the old HamletHandler, the ExHamletHandler's default implementations for getElementRepeatCount(), getElementReplacement(), getElementAttributes() and getElementIncludeSource() are not overwritten in the GuestBookHandler subclass in order to provide dynamic content for a particular template:

```
protected class GuestBookHandler extends ExHamletHandler {


  private int i = 0;
  private Comment curComment;
  private Vector<Comment> comments;
  private SimpleDateFormat format;


  public GuestBookHandler (Hamlet hamlet, Vector<Comment> comments) {
    super (hamlet);
    this.comments = comments;
    format = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
  } // GuestBookHandler


  public int getCommentsRepeatCount (String id, String name, Attributes atts) {
```

```
    return comments.size ();
  } // getCommentsRepeatCount


  public String getDateReplacement (String id, String name, Attributes atts) {
    curComment = (Comment) comments.elementAt (i);
    return format.format (curComment.date);
  } // getDateReplacement


  public String getCommentReplacement (String id, String name, Attributes atts) {
    i++;
    return curComment.text;
  } // getCommentReplacement


} // GuestBookHandler
```

Instead, the default implementations are inherited by the GuestBookHandler subclass and they use Java Reflection to call the subclass' own methods (namely getCommentsRepeatCount(), getDateReplacement(), and getCommentReplacement()) where dynamic content is provided.

At runtime, the following call sequence results. The Hamlets framework invokes the GuestBookHandler's getElementReplacement() method (inherited from its ExHamletHandler superclass) when the <REPLACE ID="Date"> tag is encountered during template parsing.

```
  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    try {
      Method m = c.getMethod ("get" + id + "Replacement", paramTypes);
      return (String) m.invoke (this, new Object[] { id, name, atts });
    } catch (NoSuchMethodException e) {
      return "";
    } // try
  } // getElementReplacement
```

In getElementReplacement(), c.getMethod() is used to locate and m.invoke() to call the getDateReplacement() method of the GuestBookHandler class. The method name "getDateReplacement" is dynamically composed with "get" + ID + "Replacement" where ID="Date". If getDateReplacement() cannot be called, an empty string is returned. Once the getDateReplacement() method is invoked, it provides the dynamic content for the template:

```
  public String getDateReplacement (String id, String name, Attributes atts) {
    curComment = (Comment) comments.elementAt (i);
    return format.format (curComment.date);
  } // getDateReplacement
```

With this new concept, each subclass of ExHamletHandler (e.g.: GuestBookHandler) inherits four callback methods from the ExHamletHandler class (namely getElementRepeatCount(), getElementReplacement(), getElementAttributes(), and getElementIncludeSource()). These methods are invoked by the Hamlets framework when special tags and IDs are encountered during template parsing. They use Java Reflection to call methods of their own class (e.g.: getDateReplacement) whose names are composed dynamically using the ID specified in the template (see Figure 3). As with

the old HamletHandler class, you have a complete separation of presentation (stored as XHTML in the template) and content (dynamically created by the Java code).

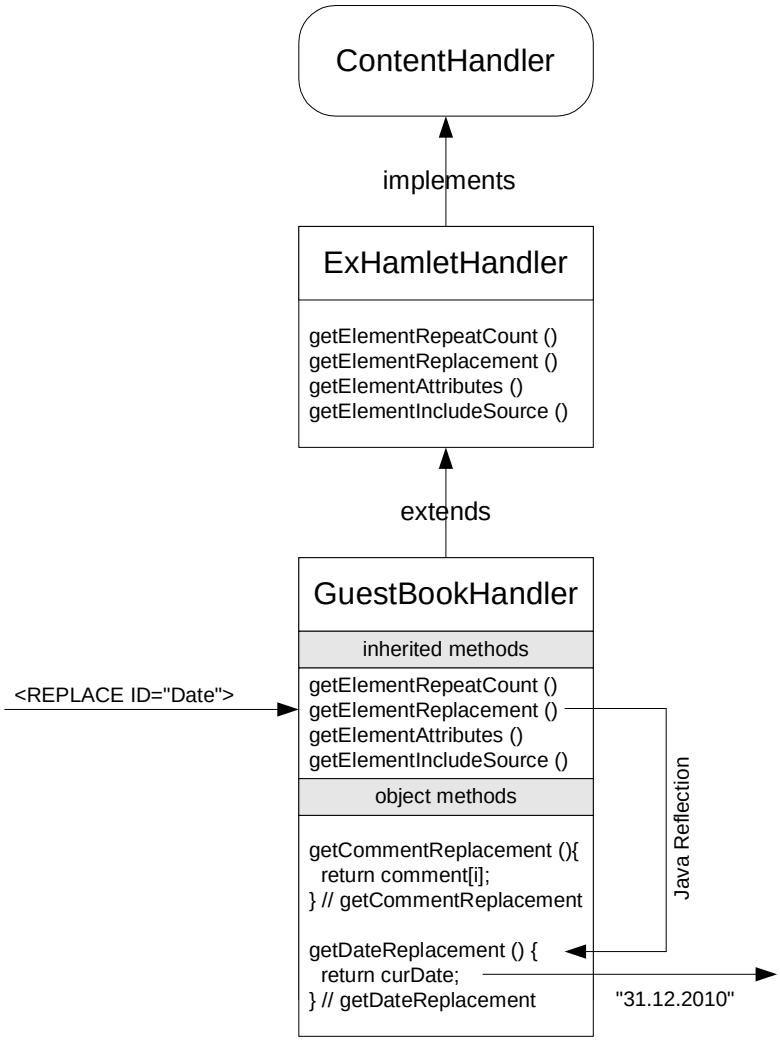

Figure 3: Providing dynamic content by adding object methods

4. How it performs

Java Reflection never had the reputation of being as fast as lightning. Thus, it cannot be expected that the new ExHamletHandler performs any better than the old HamletHandler which uses a simple if-then-else statement to determine the dynamic processing that needs to take place. A test program using the new ExHamletHandler

```
public class TestHandler1 extends ExHamletHandler {

  public TestHandler1 (Hamlet hamlet) {
    super (hamlet);
  } // TestHandler1
```

```
  public String getNameReplacement (String id, String name, Attributes atts)
    throws Exception {
    return MyName;
  } // getNameReplacement

} // TestHandler1
```

and a number of old HamletHandlers with if-then-else statements of various lengths

```
public class TestHandler2 extends HamletHandler {

  public TestHandler2 (Hamlet hamlet) {
    super (hamlet);
  } // TestHandler2

  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    if (id.equals ("Name"))
      return MyName;
    return "?";
  } // getElementReplacement

} // TestHandler2

public class TestHandler3 extends HamletHandler {

  public TestHandler3 (Hamlet hamlet) {
    super (hamlet);
  } // TestHandler3

  public String getElementReplacement (String id, String name, Attributes atts)
    throws Exception {
    if (id.equals ("aaaaaa"))
      return null;
    else if (id.equals ("bbbbbbb"))
      return null;
    else if (id.equals ("ccccccc"))
      return null;
    else if (id.equals ("ddddddd"))
      return null;
    else if (id.equals ("Name"))
      return MyName;
    return "?";
  } // getElementReplacement

} // TestHandler3
```

was written to measure the performance of each handler.

Handler	Extends	# of if	Time [ms] for 10’000’000 calls
TestHandler1	ExHamletHandler	-	13046
TestHandler2	HamletHandler	1	109
TestHandler3	HamletHandler	5	375
TestHandler4	HamletHandler	10	687
TestHandler5	HamletHandler	20	1297

As shown above, the new ExHamletHandler (extended by TestHandler1) is at least a magnitude slower than the old HamletHandler (extended by all other TestHandlers). Is it still useful then (without template compilation)? If you assume that on average 50 calls to

the HamletHandler are necessary to fill a template, the ExHamletHandler needs $6.52*10^{-5}$s and the old HamletHandler (with 20 if) only $6.49*10^{-6}$s. Even though you have a difference of a full magnitude, most often it will not matter, because the time to generate the dynamic content, to transfer the page, and to display the page will be much higher. So for most applications the new ExHamletHandler is fast enough (even without template compiliation).

5. How it compiles

As described in the “Compiling Hamlets” article, template compilation can be used to accelerate Hamlets. The new ExHamletHandler implements the ContentHandler interface like the old HamletHandler and therefore template compilation works as usual. However, the new ExHamletHandler makes it possible for the template compiler to generate direct calls to the methods which provide the dynamic content because the code to produce the content is no longer part of an if-then-else statement. In other words, the template compiler doesn’t generate calls for getElementRepeatCount(), getElementReplacement(), getElementAttributes() and getElementIncludeSource() any more (nor does it rely on Java Reflection to find the matching method to produce the dynamic content at runtime). Instead, it generates a getDateReplacement() call when a <REPLACE ID=”Date”> is encountered in the template (see Figure 4). So in the end, the new ExHamletHandler (in combination with template compilation) is the fastest way to fill a template!

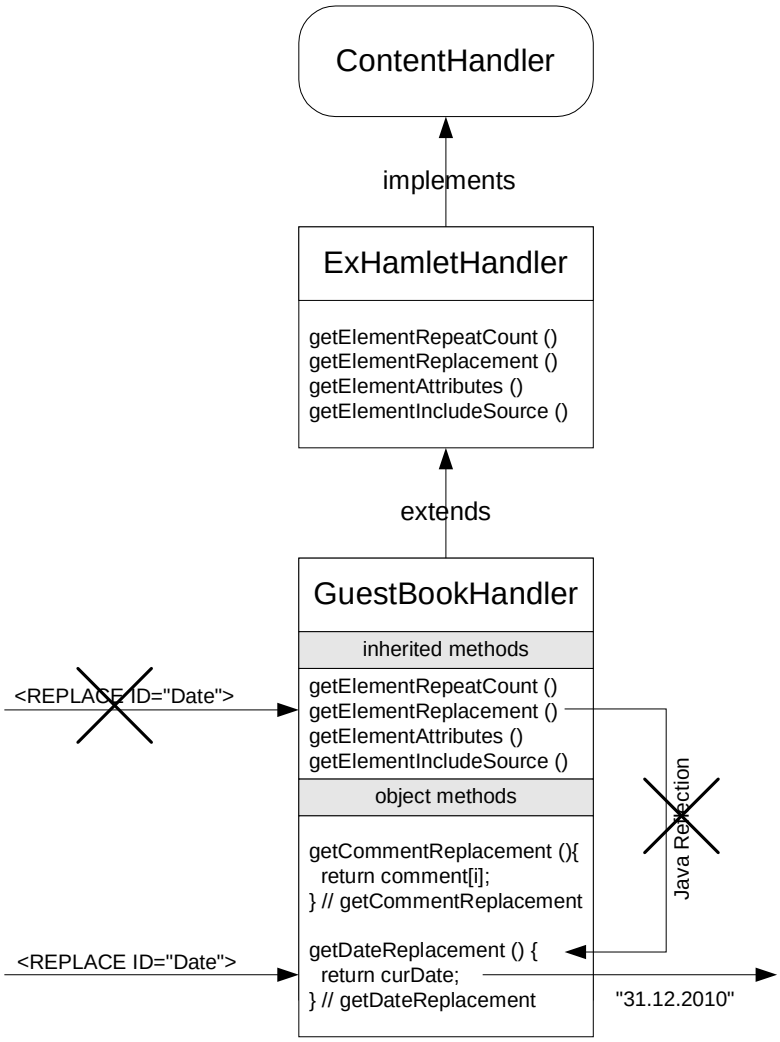

Figure 4: The ExHamletHandler enables the template compiler to generate direct calls.

The old GuestBookHandler extends HamletHandler and uses getElementRepeatCount() and getElementReplacement() with if-then-else statements to provide content.

```
protected class GuestBookHandler extends HamletHandler {


  private int i = 0;
  private Comment curComment;
  private Vector<Comment> comments;
  private SimpleDateFormat format;


  public GuestBookHandler (Hamlet hamlet,
    Vector<Comment> comments) {
    super (hamlet);
    this.comments = comments;
    format = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
  } // GuestBookHandler


  public int getElementRepeatCount (String id, String name,
    Attributes atts) {
    if (id.equals ("Comments"))
      return comments.size ();
    return 0;
  } // getElementRepeatCount


  public String getElementReplacement (String id, String name,
    Attributes atts)
    throws Exception {
    if (id.equals ("Date")) {
      curComment = (Comment) comments.elementAt (i);
      return format.format (curComment.date);
    } else if (id.equals ("Comment")) {
      i++;
      return curComment.text;
    } // if
    return "?";
  } // getElementReplacement


} // GuestBookHandler
```

The new GuestBookHandler extends ExHamletHandler and uses get<ID>RepeatCount() and get<ID>Replacement() to provide content.

```
protected class GuestBookHandler extends ExHamletHandler {


  private int i = 0;
  private Comment curComment;
  private Vector<Comment> comments;
  private SimpleDateFormat format;


  public GuestBookHandler (Hamlet hamlet,
    Vector<Comment> comments) {
    super (hamlet);
    this.comments = comments;
    format = new SimpleDateFormat ("yyyy-MM-dd HH:mm:ss");
  } // GuestBookHandler


  public int getCommentsRepeatCount (String id, String name,
    Attributes atts) {
    return comments.size ();
  } // getCommentsRepeatCount


  public String getDateReplacement (String id, String name,
    Attributes atts) {
    curComment = (Comment) comments.elementAt (i);
    return format.format (curComment.date);
  } // getDateReplacement


  public String getCommentReplacement (String id, String name,
    Attributes atts) {
    i++;
    return curComment.text;
  } // getCommentReplacement


} // GuestBookHandler
```

Table 1: Comparison of the old and the new GuestBookHandler

Implementation of the old HamletHandler class

```
public class HamletHandler implements ContentHandler {

  private  Hamlet  hamlet;

  public HamletHandler (Hamlet hamlet) {
    this.hamlet = hamlet;
  } // HamletHandler

  public int getElementRepeatCount (String id, String name,
    Attributes atts) throws Exception {
    return 0;
  } // getElementRepeatCount

  public String getElementReplacement (String id, String name,
    Attributes atts) throws Exception {
    return "";
  } // getElementReplacement

  public Attributes getElementAttributes (String id, String name,
    Attributes atts) throws Exception {
    return atts;
  } // getElementAttributes

  public InputStream getElementIncludeSource (String id,
    String name, Attributes atts) throws Exception {
    URL url = new URL (
      hamlet.getIncludeURL (atts.getValue ("SRC")));
    return url.openStream ();
  } // getElementIncludeSource

} // HamletHandler
```

Implementation of the new ExHamletHandler class

```
public class ExHamletHandler implements ContentHandler {

  protected static  Class<?>[]  paramTypes =  new Class<?>[]
    { String.class, String.class, Attributes.class };

  protected  Hamlet    hamlet;
  protected  Class<?>  c;

  public ExHamletHandler (Hamlet hamlet) {
    this.hamlet = hamlet;
    c = this.getClass ();
  } // ExHamletHandler

  public int getElementRepeatCount (String id, String name,
    Attributes atts) throws Exception {
    try {
      Method m =
        c.getMethod ("get" + id + "RepeatCount", paramTypes);
      Integer val = (Integer) m.invoke (this, new Object[]
        { id, name, atts });
      return val.intValue ();
    } catch (NoSuchMethodException e) {
      return 0;
    } // try
  } // getElementRepeatCount

  public String getElementReplacement (String id, String name,
    Attributes atts) throws Exception {
    try {
      Method m =
        c.getMethod ("get" + id + "Replacement", paramTypes);
      return (String) m.invoke (this, new Object[]
        { id, name, atts });
    } catch (NoSuchMethodException e) {
      return "";
    } // try
  } // getElementReplacement

  public Attributes getElementAttributes (String id, String name,
    Attributes atts) throws Exception {
    try {
      Method m =
        c.getMethod ("get" + id + "Attributes", paramTypes);
```

	`` return (Attributes) m.invoke (this, new Object[] { id, name, atts }); } catch (NoSuchMethodException e) { return atts; } // try } // getElementAttributes public InputStream getElementIncludeSource (String id, String name, Attributes atts) throws Exception { try { if (id == null) throw new NoSuchMethodException (); Method m = c.getMethod ("get" + id + "IncludeSource", paramTypes) return (InputStream) m.invoke (this, new Object[] { id, name, atts }); } catch (NoSuchMethodException e) { URL url = hamlet.getIncludeURL (atts.getValue ("SRC")); return url.openStream (); } // try } // getElementIncludeSource } // ExHamletHandler ```

Table 2: Implementation of the old and the new HamletHandler

6. Acknowledgements

I thank Christoph Miksovic for his valuable comments and suggestions.

Resources

- "Introducing Hamlets", René Pawlitzek (developerWorks, March 2005): This article introduces the basic ideas behind Hamlets and describes the very first implementation.

- "Programming Hamlets", René Pawlitzek (developerWorks, May 2005): This tutorial illustrates various aspects of Hamlet programming as it provides a number of practical Hamlet examples.

- "Implementing Hamlets", René Pawlitzek (developerWorks, February 2006): This article explains the current implementation of the Hamlets framework.

- "Compiling Hamlets", René Pawlitzek (developerWorks, June 2006): This article introduces a method of compiling Hamlet templates to improve application performance.

- "Embedding Hamlets", René Pawlitzek (developerWorks, June 2007): This article shows how to write a web-based user interface for embedded devices running OSGi.

- "Starting Hamlets", René Pawlitzek (IBM Research Report RZ 3747, August 2009): This article explains how to write your first web-based application in Java using Eclipse.

- Hamlets home page: Now that this project is open source, check it out on SourceForge.

- Hamlets Wikipedia entry in English, Català, Česky, Dansk, Deutsch, Español, Français, Italiano, Nederlands, Română, Русский, Српски / Srpski, Suomi, and 中文.

- Learn more about Java Reflection by following The Reflection API trail.

- Find out more about the dynamic aspects of Java programming by reading this series of IBM developerWorks articles.

About the author

René Pawlitzek is a citizen of Liechtenstein and holds an engineering degree in computer science from the Swiss Federal Institute of Technology (ETH Zürich). René works as a research and development engineer and as a project manager for the Services Management group at IBM Research-Zurich in Switzerland. Before coming to IBM, he worked in California for Hewlett-Packard, WindRiver Systems, and Borland International.