УДК 681.3.06

Ю.В. Коваленко

ОСОБЕННОСТИ ПРОЕКТИРОВАНИЯ ВЕБ-ПРИЛОЖЕНИЙ ДЛЯ ОБЕСПЕЧЕНИЯ ИНТЕРОПЕРАБЕЛЬНОСТИ ИНФОРМАЦИОННЫХ СИСТЕМ

Аннотация. *Рассмотрены особенности проектирования веб-приложений для обеспечения интероперабельности информационных систем. Предложена архитектура веб-приложения, которая позволяет обеспечить интероперабельность информационных систем, использующих различные технологии разработки веб-приложений.*

Ключевые слова: *веб-приложение, интероперабельность, информационная система, архитектура, технология.*

Современные информационные системы (ИС) характеризуются большим разнообразием технологий разработки веб-приложений, которые используются для создания пользовательского интерфейса. Наиболее распространенными технологиями являются ASP.Net и PHP, которые используют HTML и каскадные таблицы стилей (CSS) для создания пользовательского интерфейса. Для взаимодействия с базами данных используется язык структурированных запросов SQL, который включает в себя подмножество операторов языка определения данных (DDL) и языка манипулирования данными (DML). Для обеспечения интероперабельности ИС, использующих различные технологии разработки веб-приложений, необходимо разработать архитектуру веб-приложения, которая позволит обеспечить взаимодействие между ИС, использующими различные технологии разработки веб-приложений.

Для достижения интероперабельности ИС, использующих различные технологии разработки веб-приложений, предлагается использовать следующую архитектуру веб-приложения (рис.1):

1) бизнес-логика веб-приложения (независима от технологии разработки веб-приложения);
2) доступ к данным (зависит от технологии разработки веб-приложения, используемой системы управления базами данных);
3) технология разработки веб-приложения (технология разработки веб-приложения, которая используется для создания пользовательского интерфейса).

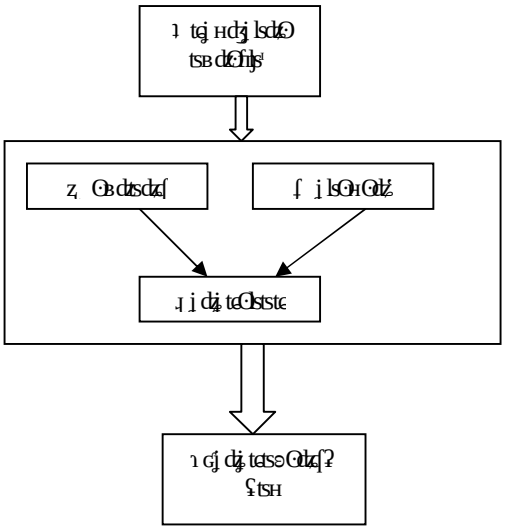

Рис.1. Архитектура веб-приложения для обеспечения интероперабельности ИС, использующих различные технологии разработки веб-приложений

Технология разработки веб-приложения:

- Бизнес-логика и доступ к данным должны быть реализованы с использованием технологии разработки веб-приложения, которая используется для создания пользовательского интерфейса (например, ASP.Net);
- Доступ к данным должен быть реализован с использованием технологии разработки веб-приложения, которая используется для создания пользовательского интерфейса, и должен использовать API системы управления базами данных, которая используется в ИС, для взаимодействия с базами данных. Для обеспечения интероперабельности ИС, использующих различные системы управления базами данных, необходимо использовать стандартный API, который поддерживается всеми системами управления базами данных, например, ODBC или JDBC;
- Бизнес-логика должна быть реализована с использованием технологии разработки веб-приложения, которая используется для создания пользовательского интерфейса, и должна использовать API доступа к данным, который используется в ИС, для взаимодействия с базами данных.

© Ю.В. Коваленко

- [illegible]
- [illegible]
- [illegible]
- [illegible]

[illegible]

- [illegible]
- [illegible]
- [illegible]

[illegible]

[illegible]

```
a = 1;
b = 2;
//x=a+b;
//y=b-a;
```

[illegible] (// [illegible]) [illegible]

```
List<string> program = new List<string>();
string line;
using (StreamReader sr = File.OpenText(filepath))
{
   while (!sr.EndOfStream)
   {
      line= sr.ReadLine();
      if (line.Contains("//"))
      {
         program.Add("int " + line [2] + ";");
         program.Add(line.Substring(2));
      }
      else program.Add(line);
   }
}
Output.PutResult(program, filepath);
```

[illegible] StreamReader. [illegible] filepath [illegible]

[illegible] PutResult [illegible] Output.

[illegible]:

```
a = 1;
b = 2;
int x;
x=a+b;
int y;
y=b-a;
```

[illegible].

[illegible] HTML [illegible]:

```
List<string> program = new List<string>();
string line;
program.Add("<html>");
program.Add("<head><title>MyClass.cs</title></head>");
program.Add("<body><font face=\"Courier\">");
using (StreamReader sr = File.OpenText(filepath))
{
   while (!sr.EndOfStream)
   {
      line = sr.ReadLine();
      line = Regex.Replace(line, "&", "&amp;");
      line = Regex.Replace(line, "<", "&lt;");
      line = Regex.Replace(line, ">", "&gt;");
      line = Regex.Replace(line, " ", " ");
      line = Regex.Replace(line, "\t", "    ");
      line = Regex.Replace(line, "\n", "<br>\n");
      line = Regex.Replace(line, "\"", "&quot;");
      program.Add(line + "<br/>");
   }
}
program.Add("</font></body></html>");
Output.PutResult(program, filepath + ".html");
```

[illegible] html, head, body, [illegible] MyClass.cs [illegible]:

```
using System;
class MyClass
{
   static void Main()
   {
      Console.WriteLine("Hello!");
      Console.ReadLine();
   }
}
```

[illegible] HTML [illegible]

```
<html>
<head><title>MyClass.cs</title></head>
<body><font face="Courier">
using System;<br/>
<br/>
```

```
class MyClass<br/>
{<br/>
    static void Main()<br/>
    {<br/>
        Console.WriteLine(&quot;Hello!&quo
t;);<br/>
        Console.ReadLine();<br/>
    }<br/>
}<br/>
</font></body></html>
```

[illegible] XML. [illegible] XSLT [illegible] XML-[illegible]:

```
<?xml-stylesheet type="text/xsl" href="variables.xsl"?>
<VARIABLES>
  <VARIABLE Active="true">
    <VAR>A</VAR>
    <VAL>1</VAL>
  </VARIABLE>
  <VARIABLE Active="true">
    <VAR>B</VAR>
    <VAL>2</VAL>
  </VARIABLE>
  <VARIABLE Active="true">
    <VAR>C</VAR>
    <VAL>3</VAL>
  </VARIABLE>
  <VARIABLE Active="false">
    <VAR>D</VAR>
    <VAL>4</VAL>
  </VARIABLE>
</VARIABLES>
```

[illegible] D. [illegible] Active [illegible] false. [illegible] true. [illegible] D [illegible]. [illegible] variables.xsl. [illegible] XML-[illegible]

```
<xsl:stylesheet xmlns:xsl="htt
2p://www.w3.org/TR/WD-xsl">
  <xsl:template match="/">
    <xsl:for-each select="VARIABLES/VARIABLE[@Active='true']">
      const int <xsl:value-of select="VAR"/> =
      <xsl:value-of select="VAL"/>;<BR/>
    </xsl:for-each>
  </xsl:template>
</xsl:stylesheet>
```

[illegible] VARIABLE, [illegible] Active, [illegible] true, [illegible] "const int", [illegible] (VAR), [illegible] (VAL). [illegible].

```
const int A = 1;
const int B = 2;
const int C = 3;
```

[illegible] XSLT [illegible] T4. [illegible] Visual Studio [illegible] T4. [illegible] ".tt", [illegible] Visual Basic [illegible] C# [illegible]:

```
<#@ template debug="false" language="C#" #>
<#@ output extension=".cs" #>
<# string[] vars = new string [] {"A", "B", "C"}; #>
class MyClass {
<#
  foreach (string variable in vars)
  { #>
  private int <#= variable #> = 0;
<# } #>
}
```

[illegible] PHP [illegible] ASP.Net. [illegible] <# [illegible] #> [illegible]:

```
class MyClass {
  private int A = 0;
  private int B = 0;
  private int C = 0;
}
```

[illegible]

```
List<string> program = new List<string>();
string line;
Regex pattern = new Regex(@"^//\s*(?<val>.*?)\s*=", RegexOptions.Singleline);
using (StreamReader sr = File.OpenText(filepath))
{
    while (!sr.EndOfStream)
    {
        line = sr.ReadLine();
        foreach (Match m in pattern.Matches(line))
        {
            program.Add("private int " + m.Groups["val"].Value + ";");
        }
        program.Add(Regex.Replace(line,@"^//\s*",""));
    }
}
Output.PutResult(program, filepath);
```

[illegible] "^//\s* (?<val>.*?) \s*=". [illegible] <val>), [illegible].
[illegible]:

```
// a = 1;
// b = 2;
// c = a + b;
```

[illegible]

```
private int a;
```

```
a = 1;
private int b;
b = 2;
private int c;
c = a + b;
```

[illegible]

1. http://www.intuit.ru/department/se/incodegen/2/
2. Vogel P Practical Code Generation in .NET : Covering Visual Studio 2005, 2008 and 2010 Boston: Pearson Education, Inc., 2010.
3. Kelly S., Tolvanen J Domain-specific modeling : Enabling Full Code Generation Hoboken, New Jersey: John Wiley & Sons, Inc., 2008.
4. Fertalj K., Brcic M A Source Code Generator Based on UML Specification International journal of computers and communications, Issue 1, Volume 2, 2008.

© [illegible]