# An Approach for Facilitating Development of Web-based Information Systems

Ángel Israel Ortiz-Cornejo, Heriberto Cuayáhuitl y Carlos Pérez-Corona

Universidad Autónoma de Tlaxcala,
Department of Engineering and Technology
Intelligent Systems Research Group
Apartado Postal #140, Apizaco, Tlaxcala, 90300, Mexico
{aortiz, hcuayahu, cperez}@ingenieria.uatx.mx

**Abstract.** In this paper we present an approach for facilitating development of web-based information systems based on the MVC (Model-View-Controller) design pattern. Our approach consists in the automation of two components: View and Controller. For automating the *view* we propose the use of style sheets, which apply a consistent visual design to the pages that integrate a web-site. For automating the *controller* we propose a markup language (WSML) for specifying the structure and navigation features of web sites, as well as a code generator for automatic coding based on an algorithm using code templates that generates fully functional HTML code by parsing WSML documents. We illustrate our approach with an information system for evaluating basic education. This approach is very useful for building web-based information systems in new domains saving time and effort in the development phase.

## 1 Introduction

Our information-based society has created a large demand for the development of web-based information systems. Currently, expert programmers perform development once the design is written. However, the development phase may require re-recoding in the following scenarios: because of unsatisfied customers who request changes in the User Interface (UI), because usability testing (usually performed before finishing coding) might require changes in the UI, and because adding functionality after deployed applications might affect part of the coded UI. These scenarios illustrate the fact that a mechanism for coding automatically from a UI document could dramatically reduce time and effort spent in the web-based application development process. In the past, few research efforts have been undertaken in this area. For instance, *WebML* provides a markup language for specifying complex web sites at the conceptual level [1, 2], but it is a general modeling language missing specific primitives of data-entry applications. Current frameworks such as *Wizard* allow developers to specify web-based data entry applications based on XML documents in order to produce fully functional skeletons for the web pages [3], and *VoiceBuilder* allow developers to specify speech applications from either a GUI or web-based application stored in a high-level language in order to automate coding of speech applications [4]. Other tools such as GARP allow developers to generate web reports automatically from a database schema [5], and the *Ninf Portal* toolkit facilitate the development of Grid portals by automatically generating the portal front-end from an XML document [6]. Despite of the recent work on the rapid prototyping of web-based applications, we must continue developing either automatic or semi-automatic methods for facilitating development of web-based information systems.

In this research we present preliminary investigations towards rapid prototyping web-based systems, based on two markup languages and a code generator. In the remainder of this paper, we first describe the architecture of our approach, WISBuilder (Web-based Information Systems Builder) in section 2. We then describe our markup languages for specifying web content (WSML and WAML) in section 3. In section 4 we describe the code generator. In section 5 we describe a case study using our proposed approach. Finally, in section 6 we provide conclusions and future directions.

## 2 Architecture of WISBuilder

There are three types of tasks involved in the development of Web-based information systems using the MVC (Model-View-Controller) design pattern: 1) the structural design of a web site, 2) the web-site visual design, and 3) the development of web applications corresponding to the business-logic. A major problem in the design of web sites is that the MVC tasks are highly dependent among them. In figure 1 we illustrate the architecture of WISBuilder, in which we proposes a separation of tasks. In this architecture a web designer specifies web content through a GUI (Graphical User Interface), stored in XML documents (WSML and WAML). Such documents are given as input to the code generator, which uses code templates and style sheets in order to generate fully functional HTML code. Due to the fact that the style sheets as well as the code templates might be generated through any development environment, the tasks of the MVC can be developed in parallel.

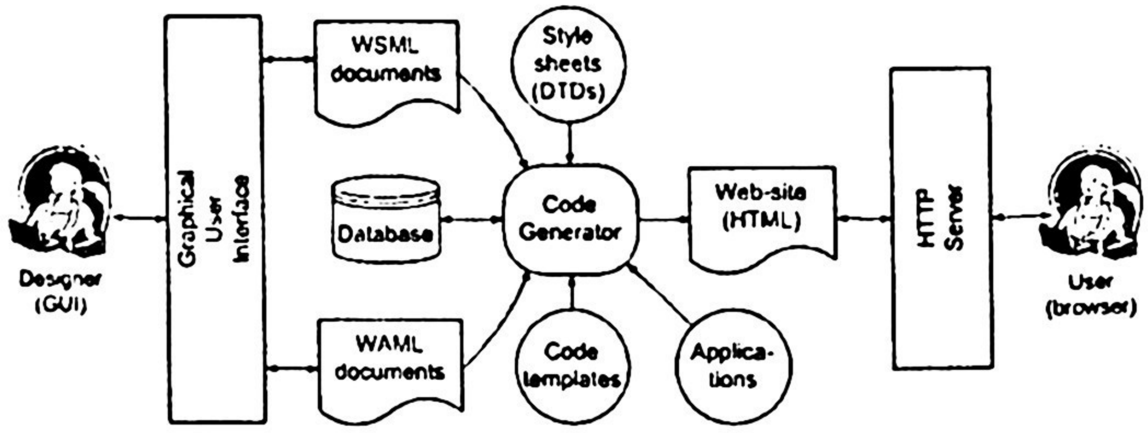


Fig. 1. The architecture of WISBuilder.

The goals of this architecture are threefold: 1) establish a clear separation of tasks for making them independent without loosing join collaboration, 2) build web-based information systems with a semi-automatic approach, and 3) make the development and maintenance tasks easier. We attempt to apply the MVC design pattern to this architecture through the use of two markup languages: WSML which specifies the structure of a web site, and WAML which specifies web applications. The *controller* is automated through the use of the WSML markup language and code templates. The WAML language is used for embedding applications into the web site and also uses code templates. The *view* is automated through the use of style sheets (stored in a repository) in order to provide a consistent view for the entire web-site. In this way, our approach attempt to facilitate development of web-based information systems.

## 3 The Markup Languages

We propose two markup languages in order to specify web content in a more declarative language and independent of visual presentation, the resulting documents are given as input to the code generator for generating fully functional code.

### 3.1 Web Site Modeling Language (WSML)

WSML (Web Site Modeling Language) is an XML language for modeling and designing web sites, which describes the elements contained into a web site. Usually, a web-site is integrated by a set of web pages, which are compounded of elements and navigation features. In the same way, a web-site in WSML is compounded of multiple web pages arranged into sections according to its contents. Each section contains at least one web page, and every page contains different kind of elements like forms, text, links to other pages, etc. The navigation among pages is the feature that allows a web page to access other pages, possibly involving a pre-processing such as a database query. Thus, WSML allows the specification of elements in a structured way, as well as the specification of navigation elements.

It is important to remark that this language allows the design of web sites without taking care about the appearance of web pages and elements inside pages, in such a way that the modeling of the site is independent of its visual style, which suggests a separate design. Another important feature of this language consists in separating the development of web applications from the modeling of the site; in other words, the user can develop applications independently of the web site design. Then, the code generator can embed automatically those applications into their corresponding places. However, the developed applications must be stored in a specific directory, so that the code generator can find them. The web applications embedded into the web pages can be created automatically using WAML (Web Application Modeling Language), a proposed markup language for describing web applications using code templates that can be written in various kinds of code (applets or scripts).

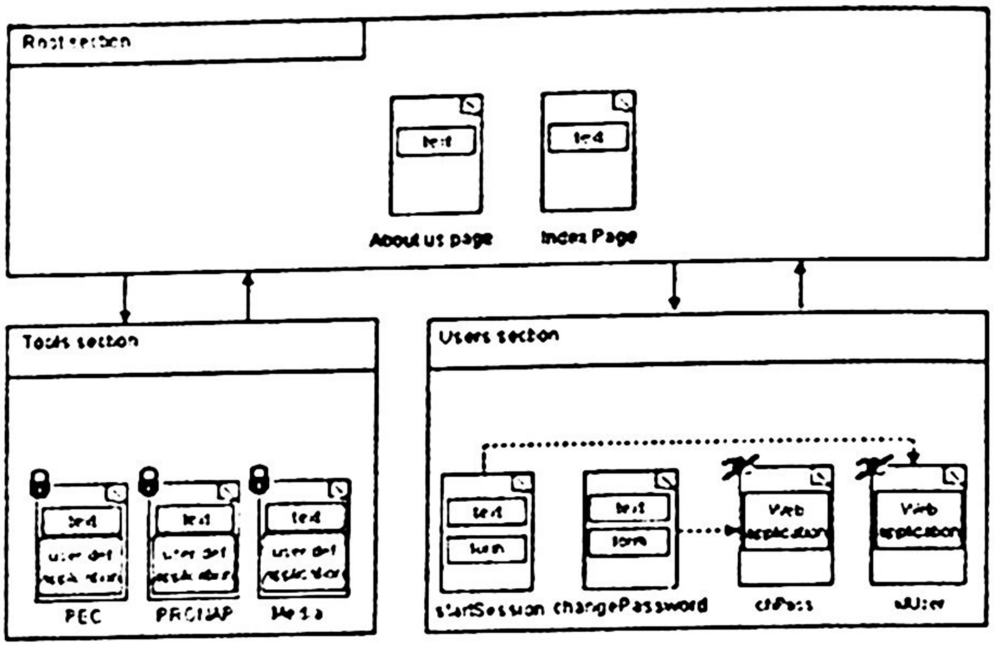


Fig. 1. An example of a web site in the design phase.

In figure 1 we show a web site with three sections: 1) the *Root* section which is the main section of a web-site, 2) the *Tools* section which represent the business logic, and 3) the *Users* section used for user authentication. Every section contains different pages with different content, which can be visible or invisible to the users. A visible page is available to the users; an invisible page can only be accessed when a process (like a form) is triggered. The pages marked with a *lock* are hidden to unlogged users; those pages are only available when a user has started a session. A page containing a form must invocate its corresponding page that contains the application; those pages which receive the requests, must import applications from a directory of applications. Such applications can be written in several kinds of code such as scripts, Java scripts, applets and WAML applications. The arrows linking sections refer to navigability, e.g. the section that points another can access the visible pages in that section.

In WSML, a web site is specified with an element called *website*, the root element (see the following XML document). The root element is divided in *section* elements, which have a unique name and might specify several web pages. The pages of a web site are represented as *webpage* elements, which require the definition of the following attributes: *name* referring to the name of the web page, and *private* referring that this page is only available for logged users. There are two elements used for controlling the navigation: *navList* which define a set of links to external or internal pages, and *navControl* which is used to link the internal sections into the site. Finally, a web page might contain many different elements like text paragraphs, imported files and photos, forms, web applications, and links to other pages, among others. These elements are considered in WSML and further details can be found in the appendixes A and B. Finally, consider the following XML document as a representation of figure 1.

```
<website name="SEPE">
  <section name=''ROOT''>
    <navControl section="ROOT"/>
    <navControl section="Tools"/>
    <navControl section="Users"/>
    <webpage name="index" title="Principal page.">
      <text>Welcome to the SEPE Web site...</text>
    </webpage>
    <webpage name="aboutUs" title=''More about us.''>
      <text>We are an institute ...</text>
    </webpage>
  <section/>

  <section name=''Tools''>
    <navControl section="ROOT"/>
    <navControl section="Tools"/>
    <webpage name="PEC'' private="yes">
      <text>The PEC is a national program for...</text>
      <appImport name=''PecApp'' type=''JavaScript''/>
    </webpage>
    <webpage name="PRONAP" private="yes">
      <appImport name=''PronapApp'' type=''Applet''/>
      <text>PRONAP (Programa Escuelas de Calidad)</text>
    </webpage>
    <webpage name="Media" private="yes">
      <text>This is the multimedia handler tool for...</text>
      <appImport name=''MediaApp'' type=''Applet''/>
    </webpage>
  <section/>

  <section name=''Users''>
    <navControl section="ROOT"/>
    <navControl section="Tools"/>
    <webpage name="startUserSession">
      <text style="bold">
        To start a user session fill the form and click "Send".
      </text>
      <form action="idUser.php">
        <input type="text" name="username" label="User name:"/>
        <input type="password" name="password" label="Password:"/>
        <submit label="Send"/>
      </form>
      <text>Have a nice day!</text>
    </webpage>
    <webpage name="changePassword">
      <text>
        In this section you can change your password.
      </text>
      <text style="bold">Please fill the form below.</text>
      <form action="chPass.php">
        <input type="name" name="name" label="Your user name:"/>
        <input type="password" name="pass" label="Your password:"/>
        <input type="password" name="newpass"
          label="Write a new password:"/>
        <submit label="submit"/>
      </form>
```

```
        </webpage>
        <webpage name="idUser" visible="no">
          <appImport name="identifyUserApp" type=''WAML''/>
        </webpage>
        <webpage name="chPass" visible="no">
          <appImport name="changePassApp" type=''WAML''/>
        </webpage>
      <section/>
    </website>
```

### 3.2 Web Application Markup Language (WAML)

WAML (Web Application Mark-up Language) is a markup language that specifies applications used for rapid prototyping web pages. Such applications are embedded into WSML through the element *appImport*. A WAML file defines web applications that can be translated to completely functional applications implemented in different kinds of code; for instance, JSP, PHP, or ASP. The current status of WAML considers three applications: database queries, database reports, and user sessions; and might be extended by frequently used applications. The following fragment of XML document is an example of an application to change password. This application receives the values "name", "pass" and "newpass" as variables; sent from the element *appImport* "changePassApp" (look at the previous XML document).

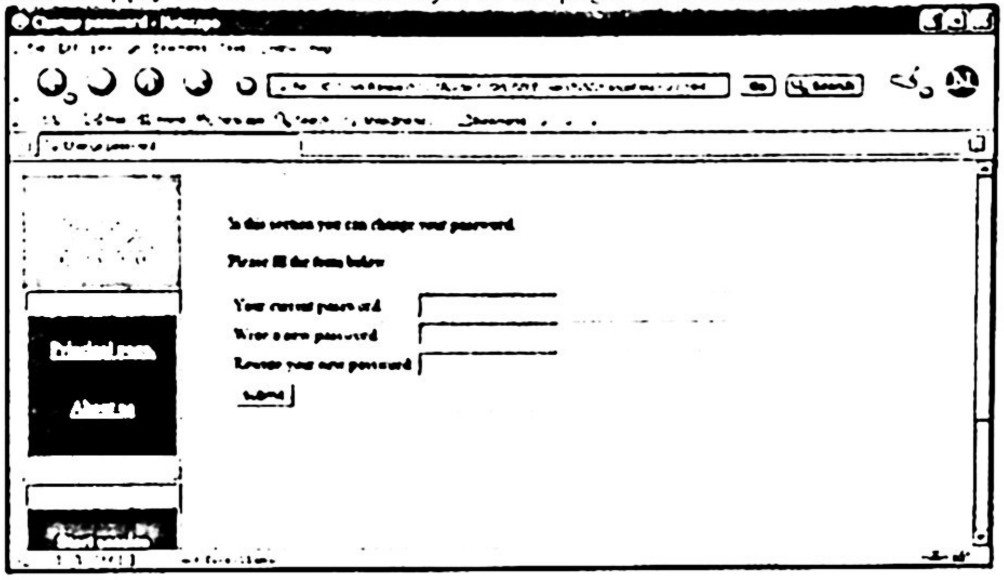
```
<application type="php" name="changePassApp">
  <sqlDo dbname="SEPE" usr="general" pass="general">
    <query>
      update into users set password=$newpass
      where name=$name and password=$pass
    </query>
    <onsuccess>
      <msg>The password was changed !</msg>
    </onsuccess>
    <onerror>
      <msg>The account doesn't exists or password is wrong.</msg>
    </onerror>
  </sqlDo>
</application>
```

## 4 The Code Generator

The code generation algorithm, called WAC (Web Application Coder), receives WSML documents and produces HTML web pages in a structure of directories, established by the web site design. This algorithm performs three main steps: the first step translates WSML elements for each web page defined in the site into HTML elements. For translating elements a set of code templates are used, which take the values of attributes and elements in order to generate fully functional HTML code. The second step consists in applying style to each web page; in other words, the instructions contained in each web page are graphically organized following the style sheet used.

Finally, in the third step the generated web pages are stored into a set of directories according the web site design.

Following the WAC algorithm (see figure 4.2), two inputs are needed: 1) the WSML document (well-formed and valid) and 2) the style sheet to apply. First, the root element of the WSML document is taken, as well as its attribute *name*. Then, a directory with that value is created. If a *webpage* element is found, then it is translated into HTML code using the code templates stored in the *templates* directory. If an *appImport* element is found, a file with the web application is loaded from the *scripts* repository in order to add it to the generated web page. Then, the style sheet is loaded and used to apply the desired visual style to the pages of the web site.


In this section you can change your password

Please fill the form below

Your current password

Write a new password

Retype your new password

Submit

About us

Fig. 4.1. Web page automatically generated from the element *startUserSession*.

The style sheet basically parses the elements contained in each page and generates HTML code. Finally, each page in the site is stored into its corresponding directory inside of the root directory; where each directory corresponds to a section in the site. Figure 4.1 shows the web page automatically generated using the following algorithm.

```
algorithm WAC(WSML document, String style)
input: a WSML document and an existing style sheet.
output: a set of web pages in HTML code and possibly embedding other kinds of code.

  root <- root element of the document
  sitename <- value of the attribute name in the root element
  create a directory called sitename
  switch to the recently created directory
  for each section element in root do
    sectionName <- value of the attribute name in the current element
    create a directory with the name specified by sectionName
    set the recently created directory as the current directory
     for each navList or navControl element in root do
       createNavList(current navList or navControl element)
     end
    for each  webpage element in root do
      createPage(current webpage element)
    end
```

```
end
function createPage(WSML webpage)
  name <- value of the attribute name of the webpage element
  extension <- the string "html"
  for each element in webpage do
     transformElement(element)
   end
   for each applmport element in webpage do
    importApplication(applmport)
   end
  load the stylesheet from the templates repository specified by the style attribute
  copy the instructions of the webpage element in the stylesheet
  page <- the document obtained from the previous step
  save the content of page, in the current directory, with its name and extension
end

function createNavList(WSML navElement)
   elemName <- the name of the element navElement
   if elemName = "navList" then
     transformElement(navElement)
   else
     navCS <- value of the attribute section of navElement
     pageArray <- all the visible web pages defined in the section navCS
     for each foundpage in pageArray do
          fplink <- create a relative link to foundpage
       linkArray <- append fplink
     end
     replace the navElement with the elements in linkArray
   end
end

function transformElement(WSML element)
  find a code template for element in the templates repository
  if found then
    apply the code template to element
   else
     keep element unmodified
   end
end

function importApplication(WSML instruction)
   imAppType <- value of the attribute type of the instruction element
   imAppName <- value of the attribute name of the instruction element
   if imAppType = "Script" then
     open the file called imAppName from scripts repository
     replace instruction with the content of the opened file
     extension <- the scripts language type of the imported application
   end
   if imAppType = "WAML" then
     wamlFile <- the root element of the file called imAppName
     in scripts repository
     generateScriptsCode(wamlFile)
     replace the instruction element with the code obtained in the previous step
   end
   if imAppType = "Applet" then
     replace instruction with an Applet invocation of the application called
     imAppName contained in the scripts repository
   end
end

function generateScriptsCode(WAML wamlapp)
```

```
    applanguage <- value of the attribute type of the wamlapp element
    select just the code templates for the language applanguage in the scripts repository
      for each applnstruction element in wamlapp do
          find a code template for applnstruction in the scripts repository
          if found then
              apply the code template to transform this element
          else print an error message
          end
      end
  end
end
```

Fig. 4.2. The code generation algorithm WAC (Web Application Coder).

## 5 Case Study: A System for Evaluating Basic Education

Our approach has been tested with a web-based information system for evaluating basic education. In this system we implement two evaluation models: PEC and PRONAP [7, 8]. These models are part of the national reform of education in order to improve issues that currently limit the performance in basic education. This system includes three subsystems: optical reading, multimedia, and evaluation. The optical reading sub-system facilitates the data-entry from surveys applied to students, lecturers, and family parents; this is accomplished using the software Remark Office OMR and a scanner with an automatic document feeder. The multimedia sub-system facilitates the transcription of data collected from two devices: audio-recorder and video-recorder. The evaluation sub-system facilitate the analysis, generation, and management of structured and unstructured information of the evaluation models PEC and PRONAP; we use ontologies for the qualitative evaluation and causal relations for the quantitative evaluation. The three subsystems must provide or retrieve contents from an integrated web-site. Thus, using our approach, such subsystems must specify their applications/content using the markup languages WSML and WAML, these subsystems do not have to worry about visual presentation; besides, they might save programming effort through using WSML and automatically generate code, or they might embed applications/scripts through the use of WAML. Our preliminary investigations suggest the following advantages: consistent visual view, less time and effort spent in programming, less costs in the development face, and less expertise required for designing web sites. We implemented this system using XSLT, JDOM, and Java [9].

## 6 Conclusions and Future Work

In this paper we have presented an approach for facilitating development of web-based information systems through markup languages and automatic code generation. For such purpose, we described a tool called WISBuilder based on the MVC pattern design in an attempt to automate the development of two components: view and controller. The view is a reusable XSL file that can be implemented in any development environment, with such file we apply a consistent view for the entire web site. The controller is specified through the use of two markup languages, and is intended to use a GUI environment for easy and fast development. Through the separation of these two components (view and controller), a web-based information system can be developed in parallel in order to save time and separate technical expertise. The markup languages WSML and WAML were very useful for specifying web contents. Such intermediate languages offer several advantages, for instance, web contents in a more declarative language, several kinds of code can be generated, and dynamic contents for adaptive navigation. The WAC algorithm was useful for automating the process of generating fully functional HTML code given the markup languages and code templates. Therefore, our approach facilitates the development of the *view* through reusability and consistent application of style sheets, and the *controller* through automatic code generation. In this regard, WISBuilder can be very useful for building web sites in new domains with a significant reduction in programming effort. The goals of WISBuilder are twofold: 1) to serve as a testbed for performing research in the area of web-based information systems, and 2) to save time and effort spent in the web development process so that more services might be automated in the world.

As an immediate work we plan to create a Graphical User Interface (GUI) tool for designing web sites based on the markup languages WSML and WAML, in such a way that WISBuilder might become into a CASE tool for software engineers. Also, we plan to perform more experiments in other domains. After such work, our approach could be extended into a more robust framework in order to include: adaptive navigation [10], validation of data-entry, and automatic testing of the generated web sites. Finally, we plan to offer WISBuilder freely available for educational and research purposes under request.

## Acknowledgements

We would like to thank CONACYT and the government of Tlaxcala for the financial support of this research, FOMIX project TLAX-2003-C02-12485. Also, we would like to thank the collaboration of the Direction of Research and Evaluation of Education of the Mexican Ministry of Education - Tlaxcala State.

## References

1. Ceri, S., Fratemali, P., Bongio, A.: Web Modelling Language (WebML): a Modelling Language for Designing Web sites. WWW Conference, 2000.
2. Brambilla M., S. Ceri, S. Comai, P. Fratemali: Model-driven development of Web Services and hypertext applications. 2003.
3. Turau, V.: A Framework for Automatic Generation of Web-based Data Entry Applications Based on XML. In Proceedings of Symposium on Applied Computing (SAC 2002), Madrid, Spain, March 2002.
4. Guillen M., Del Rosario M., Sosa V., and Hernandez H.: GARP: A tool for Creating Dynamic Web Reports Using XSL and XML Technologies, In Proceedings of ENC, Tlaxcala, México, September 08 - 12, 2003, 54 - 59.
5. Suzuma, T., Nakada, H., Saito, M., Matsuoka, S., Tanaka, Y., Sekiguchi, S: The Ninf Portal. An automatic Generation Tool for Grid Portals. Seattle, Washington, USA. November 2002.

6. Cuayáhuitl H., Rodriguez M., and Montiel J., VoiceBuilder: A Framework for Automatic Speech Application Development, México, April 2004.
7. PEC: Programa Escuelas de Calidad, http://www.escuelasdecalidad.net
8. PRONAP: Programa Nacional para la Actualización Permanente, http://pronap.ilce.edu.mx
9. Armstrong, E., Ball, J., Bodoff, S., Bode D., Fisher, M., Fordin, S., Green, D., Haase, K., Jendrock, E: The Java Web Services Tutorial. http://java.sun.com/webservices, Feb 2003.
10. Po-Hao Chang, Wooyoung Kim, and Gul Agha: An adaptive Programming Framework for Web Applications. 2003. The IEEE/IPSJ Symposium on Applications and the Internet (SAINT'04). Tokyo, Japan January 26 - 30, 2004.

## Appendix A: DTD of the WSML Markup Language

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- ========================================= -->
<!-- Elements for specifying contents of a web site -->
<!-- ========================================= -->
<!ELEMENT website (section+)>
<!ATTLIST website name CDATA #REQUIRED>
<!ELEMENT section (webpage+)>
<!ATTLIST section name NMTOKEN "ROOT">
<!ELEMENT webpage ((navControl | navList)*, (text | form | link | include
| appImport | attachXml | publish)*)>
<!ATTLIST webpage
            name NMTOKEN "index"
            section NMTOKEN "ROOT"
            title CDATA #REQUIRED
            private (yes | no) "no"
            visible (yes | no) "yes"
>
<!ELEMENT text (#PCDATA | link)*>
<!ATTLIST text style (title | p | heading | bold | italics | big | small)
"p">
<!ELEMENT link EMPTY>
<!ATTLIST link label CDATA #REQUIRED to CDATA #REQUIRED>
<!ELEMENT navList (link)+>
<!ATTLIST navList title CDATA #REQUIRED id ID #REQUIRED>
<!ELEMENT navControl EMPTY>
<!ATTLIST navControl section NMTOKEN #REQUIRED>
<!ELEMENT form ((input)+, submit, reset?)>
<!ATTLIST form action NMTOKEN #REQUIRED>
<!ELEMENT input EMPTY>
<!ATTLIST input
            type (button | checkbox | file | image | hidden | password |
radio | text) #REQUIRED
            name NMTOKEN #REQUIRED
            label CDATA ""
            maxlength NMTOKEN "20"
>
<!ELEMENT submit EMPTY>
<!ATTLIST submit label CDATA "submit">
<!ELEMENT reset EMPTY>
<!ATTLIST reset label CDATA "reset">
<!ELEMENT attachXml EMPTY>
<!ATTLIST attachXml path CDATA #REQUIRED stylesheet CDATA #REQUIRED>
<!ELEMENT include EMPTY>
<!ATTLIST include
            path CDATA #REQUIRED
            type (text | image | file | xml) #REQUIRED
>
<!ELEMENT publish EMPTY>
<!ATTLIST publish
            FileTypes NMTOKENS #REQUIRED
            dir CDATA #REQUIRED
            orderby (name | ext | size) "name"
            style (table | list) "list"
```

```
>
<!ELEMENT appImport EMPTY>
<!ATTLIST appImport
            name CDATA #REQUIRED
            type (Script | JS | Applet | WAML) #REQUIRED
>
```

## Appendix B: DTD of the WAML Markup Language

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!-- ------------------------------------------------------ -->
<!-- Elements for specifying applications embedded into WSML -->
<!-- ------------------------------------------------------ -->
<!ELEMENT application (param*, (msg | sqlDo | sqlReport | startSession))>
<!ATTLIST application
        type (jsp | php) #REQUIRED
        name NMTOKEN #REQUIRED
>
<!ELEMENT sqlDo ((query, (onerror? | onsuccess?)*)+)>
<!ATTLIST sqlDo
        dbname NMTOKEN #REQUIRED
        server NMTOKEN "127.0.0.1"
        usr NMTOKEN #REQUIRED
        pass NMTOKEN #REQUIRED
        driver (odbc | mysql | oracle) "odbc"
>
<!ELEMENT sqlReport ((query, (onerror? | onsuccess?)*)+)>
<!ATTLIST sqlReport
        dbname NMTOKEN #REQUIRED
        server NMTOKEN "127.0.0.1"
        usr NMTOKEN #REQUIRED
        pass NMTOKEN #REQUIRED
        driver (odbc | mysql | oracle) "odbc"
>
<!ELEMENT startSession ((requiredField)+,errorMsg)>
<!ATTLIST startSession
        dbname NMTOKEN #REQUIRED
        tablename NMTOKEN #REQUIRED
>
<!ELEMENT requiredField EMPTY>
<!ATTLIST requiredField name NMTOKEN #REQUIRED variable CDATA #REQUIRED>
<!ELEMENT query (#PCDATA)>
<!ELEMENT onerror (msg | sqlDo | sqlReport | (query, (onerror? | onsuc-
cess?)*))*>
<!ELEMENT onsuccess (msg | sqlDo | sqlReport | (query, (onerror? | onsuc-
cess?)*))*>
<!ELEMENT msg (#PCDATA)>
<!ELEMENT errorMsg (#PCDATA)>
<!ELEMENT param EMPTY>
<!ATTLIST param names NMTOKENS #REQUIRED>
```