RI 08017, 4 November 2008 Computer Science

IBM Research Report

The Telecom Web Application Framework

Arun Kumar
IBM Research - India
ISID Campus, 4, Block - C,
Institutional Area, Vasant Kunj,
New Delhi - 110 070, India

Sheetal K. Agrawal
IBM Research - India
Embassy Golf Links Business Park,
Block D, Intermediate Ring Road,
Bengaluru 560071 INDIA.

Priyanka Manwani
IBM Software Group
ISID Campus, 4, Block - C,
Institutional Area, Vasant Kunj,
New Delhi - 110 070, India

The Telecom Web Application Framework

Arun Kumar
IBM Research - India
ISID Campus, 4, Block - C,
Institutional Area, Vasant Kunj,
New Delhi - 110070, INDIA.
kkarun@in.ibm.com

Sheetal K. Agarwal
IBM Research - India
ISID Campus, 4, Block - C,
Institutional Area, Vasant Kunj,
New Delhi - 110070, INDIA.
sheetaga@in.ibm.com

Priyanka Manwani
IBM Software Group
ISID Campus, 4, Block - C,
Institutional Area, Vasant Kunj,
New Delhi - 110070, INDIA.
pmanwani@in.ibm.com

ABSTRACT

The World Wide Telecom Web (WWTW) (also known as Spoken Web) is emerging as an alternate web for the underprivileged, navigable entirely through a voice based interface using an ordinary telephone. A node in the WWTW graph is a voice application called a *VoiceSite* and is accessible over a simple phone call. VoiceSites are hyperlinked through *Hyperspeech Transfer Protocol* (HSTP).
However, several challenges need to be addressed for large scale development, deployment, interworking and usability of such VoiceSites. First and foremost, the set of users that need to be supported vary from software developers to illiterate rural farmers. Second, the fact that the primary user interface is voice based and over telephony presents its own challenges and opportunities. Third, most voice application frameworks currently available make it hard to separate navigational flow from process flow which makes systematic application development even harder.
This paper presents Telecom Web Application Framework (TWAF) – an application development and deployment framework that is designed to support rapid development of Telecom Web applications (i.e. VoiceSites) while addressing the issues and requirements mentioned. We present the architecture, end-to-end design and a proof-of-concept implementation of the TWAF framework.

Keywords

Developing regions, Web Application Frameworks, Voice Applications, VoiceSites, World Wide Telecom Web

1. INTRODUCTION

Barely 22% of the world's population has access to Internet [21]. It implies that the impact of the World Wide Web has not been able to reach a large percentage of human population which is characterized by unaffordability, illiteracy and lack of locally relevant content. This essentially means that most of the innovative applications of the Web such as email, instant messaging, websites, blogs, mashups etc. are unavailable to them.
One of the major causes of this situation is expressed succintly in [16] that *"most designers of the world are focussed on designing for the wealthiest 10 percent"*. Several examples in [20] illustrate how designing for the other 90 percent can achieve innovative results that would benefit a much larger segment of human population and make their lives better.
Addressing this problem in the information technology domain and more specifically, in the context of online information and services, we have proposed World Wide Telecom Web (WWTW) [12] (also known as Spoken Web or Telecom Web) as an alternate web for the underprivileged. WWTW is envisioned as being complementary to the existing Web and is navigable entirely through a voice based interface using an ordinary telephone. A node in the WWTW graph is a voice application called a *VoiceSite* [11] and is accessible over a simple phone call. VoiceSites are created by end users through a voice based interface [11], hyperlinked through *Hyperspeech Transfer Protocol* (HSTP) [1] and can be browsed through a Telecom Web Browser [2]. Such a web of applications, driven by spoken commands, opens up tremendous opportunities for billions of people in developing countries to whom the existing Web is inaccessible due to illiteracy, language barriers, infrastructural problems [5], low disposable income and lack of content or services that are locally relevant.
Proliferation of websites, web applications and web based services took place due to availability of easy-to-use tools, development environments and frameworks. Specifically, a web tier application framework is meant to manage the interaction between web clients and the application's business logic [19]. It typically generates HTML, XML or XMl-based content to be rendered to the user. Web application frameworks strive to separate business logic from UI navigational logic.
Given the envisioned scale of deployment of VoiceSites in the World Wide Telecom Web and the special characterists of the target user segment listed above, a framework is needed to enable rapid creation and deployment of VoiceSites through a simplified mechanism. In this paper, we present *Telecom Web Application Framework (TWAF)* that is designed to fulfil these needs.
The Telecom Web Application Framework (TWAF) is an application framework for creating, deploying and managing interactive Telecom Web applications (i.e. VoiceSites) as well as *meta-Telecom Web applications*. A meta-Telecom Web application is a Telecom Web application that enables the end-user to create another Telecom Web application through a voice based interface, in addition to GUI/API interface as supported by traditional web application frameworks. Intuitively, meta-VoiceSites are VoiceSites that enable the end users to create their desired VoiceSites.
Traditional web application frameworks such as Apache Struts [8],

JavaServer Faces[1], J2EE BluePrints Web Application Framework (WAF) [19], Microsoft ASP.NET [7] etc., are meant to be used by developers to code up the web applications as seen and experienced by the end users. Compared to that, TWAF serves two distinct goals. First, developers can use it to create meta-VoiceSites through an Application Programming Interface (API) and/or a Graphical User Interface (GUI). Second, end users can utilize TWAF to create their VoiceSites through a Voice User Interface (VUI) [6] and/or a GUI, conforming to the structure laid out by one of the meta-VoiceSites.
The contributions of this paper are as follows:

- The paper presents TWAF, an application framework for rapid development and deployment of VoiceSites for World Wide Telecom Web (WWTW) – an alternative web for the underprivileged that is complementary to the existing one.

- The TWAF framework supports voice over telephony as the primary interface for creation of VoiceSites to support non IT-literate users, in addition to supporting APIs and GUIs for the developers.

- The TWAF framework's application development model enables creation of meta-applications (i.e. creator of other applications) as well.

- We follow a principled four-stage approach to perform the entire development process in a systematic and structured manner.

- We describe an end-to-end working prototype that demonstrates (1) specification of a schema for describing VoiceSite components (2) definition of a meta-application (i.e. VoiceSite Template) (3) creation and deployment of a VoiceSite on a runtime engine.

2. MOTIVATION

2.1 Background

In this sub-section, we briefly describe World Wide Telecom Web [12] – our vision of a web of VoiceSites which we believe has the potential to become the mainstream information web for the underprivileged in developing regions.
VoiceSites are voice driven applications that are *created by the subscribers* and *hosted in the network* [11]. A VoiceSite is represented by an associated phone number and can be accessed from any phone instrument, mobile or landline through an ordinary phone call to that number. A VoiceSite could be an individual's VoiceSite in which case it gets deployed against his phone number. In situations, where a VoiceSite represents a group, it gets deployed against a phone number accessible to the entire group. What makes them compelling for the underprivileged is the fact that VoiceSites themselves can be created through a voice interface over an ordinary phone call [11]. Usability wise this means that even illiterate or low IT-literate users can also easily create and manage their VoiceSites. Technology wise this means that VoiceSites can be created through the use of other special VoiceSites. These creator VoiceSites are called *meta-VoiceSites*.

[1]http://java.sun.com/javaee/javaserverfaces/overview.html

VoiceSites thus are analogous to websites in the WWW in terms of functionality and differ primarily in their user interface i.e. voice based interaction over a phone call. Similar to Hypertext Transfer Protocol (HTTP) links in the Web, VoiceSites can also be linked to other VoiceSites through Hyperspeech Transfer Protocol (HSTP) [1]. Such an interconnection of VoiceSites opens several possibilities for telephony voice applications and can potentially create a web parallel and complimentary to the existing World Wide Web, called World Wide Telecom Web (WWTW) [12] as shown in Figure 1.

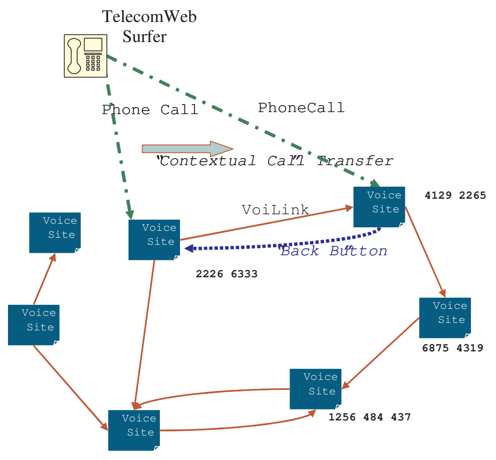


Figure 1: The World Wide Telecom Web

WWTW has tremendous implications for the underprivileged people in developing countries. It enables the non-IT literate people to access and offer information and services that were hitherto inaccessible to them, through affordable means. The ease of creation of VoiceSites enables the subscribers to become information providers as opposed to being simply information consumers. Several applications of Telecom Web and similar voice-based systems are emerging. Knowledge sharing tools taken for granted in the Web world could be made available to the underprivileged through a voice based interface. For instance, a voice based version of Wikipedia[2] – the online, community created encyclopedia is presented in [18]. A village community portal VoiceSite [3] creates an ecosystem of a closely knit rural community and provides them a channel for sharing locally relevant information as well as to network socially. VoiceSites could be used to remove inefficiencies in the current operational models of various unorganized sectors prevalent in the developing regions [10]. A voice based Wiki software is presented in [9].

2.2 Framework Requirements

It is evident that for rapid creation and deployment of VoiceSites in the Telecom Web, an application framework similar to the ones available for World Wide Web such as Struts, JSF, etc. would be needed. Next, we describe the technical and social requirements expected to be fulfilled by such an application framework. These include the following:

- First and foremost, an application framework for WWTW needs to cater to different kinds of users. It needs to provide an Application Programming Interface (API)

[2]http://www.wikipedia.org

and/or a Graphical User Interface (GUI) to be able to create VoiceSites in a systematic and structured manner. Second, it needs to provide a Voice User Interface (VUI) to naive, non-IT savvy users to be able to create their VoiceSites. Such VUIs for creating VoiceSites are also VoiceSites called *meta-VoiceSites.*

- The application framework should also provide an API and/or a GUI to developers for creating these meta-VoiceSites.
- The APIs/GUIs of the framework should be capable of generating website equivalents of the VoiceSites for better integration of Telecom Web with the existing World Wide Web.
- In addition to the development life-cycle, the application framework needs to support automatic deployment of generated VoiceSites (and equivalent websites) on a runtime engine.
- Finally, the compile time VoiceSite creation aspects of the framework as well as the runtime engine should both be able to integrate with existing IT systems such as databases, web services, etc. Also, the framework should be flexible to allow third party providers to add new modules to the framework for extension, customization and for differentiation among competing providers.

The requirements listed above are non-trivial to achieve and are somewhat different from the requirements fulfilled by existing web application frameworks.

2.3 A Running Example

A VoiceSite can belong to and represent an individual or it could represent a group or an entire community. Here, we take the example of a community VoiceSite that serves the purpose of a Village Portal [3].
The Village Portal VoiceSite is meant to be a local information sharing hub for a village and is managed by a local resident designated as the Village Portal administrator. A Village Portal could offer information and services such as an updated local bus/train schedule, doctor's visit timings in the local health center, movies being played in the nearby theatre, new government schemes launched, upcoming events for the village and classifieds for jobs, matrimonials, equipment rentals etc., among others. For illustrative purposes we take a simplified view of a Village Portal as depicted in Figure 2. It shows the overall structure of the application which consists of the following options:

- *Agricultural Consultancy Service :* allows villagers to browse an agriculture FAQ applicable for the village and post their questions to be answered by qualified experts. Systems such as COMMON-Sense Net [14] utilize environment monitoring data collected through the use of wireless sensor networks, to provide agricultural advice to farmers.
- *Health Center Service :* provides announcements related to upcoming health camps, disease outbreak notifications and other health related messages. Also, includes a locally relevant medical FAQ.

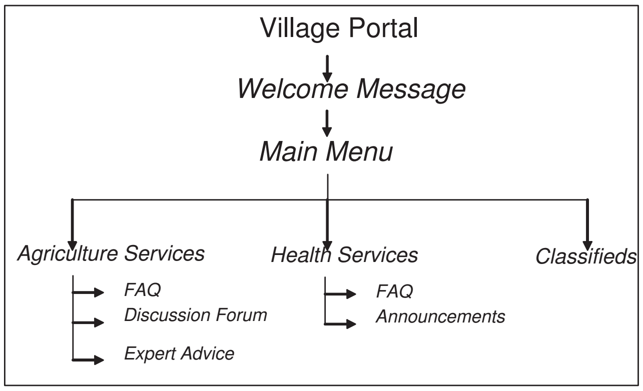


Figure 2: A Simplified Village Portal VoiceSite

- *Classifieds :* a section where people can record their professional and personal advertisements and browse through advertisements posted by other villagers.

We shall use this example throughout the paper to illustrate the working of the TWAF framework and its components.

3. SYSTEM ARCHITECTURE

The TWAF framework has an ambitious target to achieve in terms of integration. It needs to enable application integration to compose VoiceSite functionality from existing business logic components. Second, it needs to perform integration of UI components. Both of these need to be done differently as illustrated in [22, 13]. This is non-trivial especially given the fact that TWAF needs to cater to naive users in addition to developers and voice based interfaces form a primary component of the user interface.

3.1 System Overview

As shown in Figure3, TWAF has a 4-stage architecture to enforce a systematic approach for the entire VoiceSite development process.
Each stage covers an important part of the VoiceSite development lifecycle. The first stage is involved with providing a language for VoiceSite designers to specify new VoiceSite designs. In that context, TWAF acts as a template framework. It includes a *VoiceSite Template Definition language (VTDL)* for creating *VoiceSite Template Definitions*. These template definitions capture the UI and functional aspects of the desired class of VoiceSites to be generated. The later stages enable generation of end-user VoiceSites based upon these templates.
Once a VoiceSite template definition has been created, TWAF uses it to generate a meta-VoiceSite (or a meta-website) that enables an end-user to create his desired VoiceSite. This step essentially configures the Template Definition with end-user specific information and results into generation of a *Template Instance.* The template instance is specification of a deployable VoiceSite. The last stage of the TWAF architecture uses this template instance specification to generate the appropriate content for the runtime engine. TWAF's template engine is capable of generating VXML (for generating VoiceSites) as well HTML (for generating equivalent web sites).

3.2 VoiceSite Template Definition Language

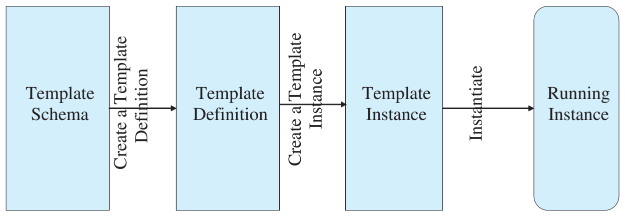


Figure 3: TWAF **- High Level Design**

VoiceSite Template Definition Language (VTDL) is an XML based language using which the VoiceSite Template designers can define the template structure for a desired class of VoiceSites to be generated. Figure 4, depicts the important elements of the schema for the VTDL. As shown, VTDL consists of various elements pertaining to different aspects of the structure of a VoiceSite Template. It consists of a *TemplateDefinition* and a library of *Components*. *TemplateDefinition* consists of a *CompositeComponent* (refer Figure 6) which is a choice between a sequence of one or more *Composite-Component*s, a choice of one or more *Composite-Component*s and a *LeafComponent*. This recursive nature of the *CompositeComponent* provides the flexibility needed for creating different kinds of template designs.

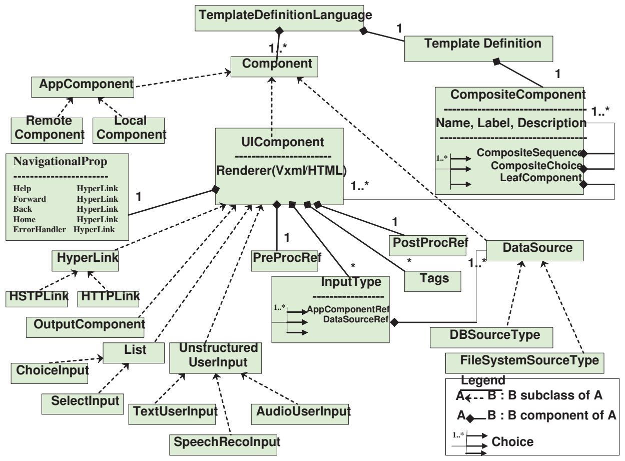


Figure 4: Template Definition Schema

```
<element name="TemplateDefinition" type="tns:TemplateType"></element>
<complexType name="TemplateType">
  <sequence>
     <element name="CompositeComponent" type="tns:CompositeComponentType"
     minOccurs="1" maxOccurs="1"> </element>
  </sequence>
  <attribute name="Name" type="string"></attribute>
  <attribute name="Version" type="string"></attribute>
</complexType>
```

Figure 5: Template Type

As shown in the figure, the component library consists of three types of components *AppComponent*, *UIComponent* (refer Figure 7) and *DataSourceComponent*. A *LeafComponent*, however, is composed of one or more *UIComponentType*s alone. A VoiceSite template is therefore essentially a collection of various kinds of *UIComponents* knit together

```
<complexType name="CompositeComponentType">
  <choice maxOccurs="unbounded" minOccurs="1">
      <sequence>
           <element name="CompositeSequence“ type="tns:CompositeComponentType"
              minOccurs="1“ maxOccurs="unbounded">  </element>
      </sequence>
      <choice>
           <element name="CompositeChoice“ type="tns:CompositeComponentType"
            minOccurs="1“ maxOccurs="unbounded"> </element>
       </choice>
      <element name="LeafComponent" type="tns:ComponentType“
         maxOccurs="unbounded" minOccurs="1"> </element>
   </choice>
   <attribute name="Name" type="string"></attribute>
   <attribute name="Description" type="string"></attribute>
   <attribute name="Label" type="string"></attribute>
</complexType>
```

Figure 6: Composite Component Type

in an interaction hierarchy as specified by the designer. A *UIComponentType* is designed to capture the nuances of user interface controls specific to the modality specified, i.e. speech or text. In addition, a *UIComponentType* typically contains a reference to one or more *AppComponentType*s and/or one or more *DataSourceComponentType*s.

```
<complexType name="UIComponentType">
  <complexContent>
     <extension base="tns:ComponentType">
       <sequence>
            <element name="PreProcRef" type="tns:methodCallType“
              maxOccurs="1" minOccurs="0">  </element>
            <element name="PostProcRef“ type="tns:methodCallType"></element>
            <element name="InputToApp" type="tns:inputType"
              maxOccurs="unbounded" minOccurs="0"> </element>
            <element name="OutputToApp" type="tns:outputType"
             maxOccurs="unbounded" minOccurs="0"></element>
            <element name="Description" type="string"></element>
            <element name="NavigationalProperties“
              type="tns:NavigationalPropertiesType"> </element>
            <element name="tag" type="string“ maxOccurs="unbounded"
              minOccurs="0"> </element>
         </sequence>
         <attribute name="Renderer" type="tns:RendererType"></attribute>
      </extension>
   </complexContent>
</complexType>
```

Figure 7: UIComponent Type

The *AppComponentType* captures the specification of components that implement business logic and may be encapsulated in a locally available component or a remote entity such as a web service. The *DataSourceComponentType* provides a mechanism to specify various data sources that can be utilized by the VoiceSite and may include databases, filesystem, etc.
Separating the specification of *UIComponent* from *AppComponent* and *DataSourceComponent* enables business logic to remain separate from presentation logic. TWAF, this way enables Model-View-Controller architecture based VoiceSite development.
Each *UIComponentType* consists of an attribute that specifies the renderer to be used. VXML and HTML are currently supported. A *UIComponentType* is composed of six components namely *InputType*, *OutputType* (not shown in figure), *PreProfRef*, *PostProfRef*, *Tags* and and optional *NavigationalProperties* element. The *InputType* and *OutputType* elements are meant to be references to those *AppComponent*s and *DataSourceComponent*s associated with the template that this component needs to use to get some input data or to send some output data respectively. *PreProcRef* and

PostProcRef specify actions to be performed before and after the use of this component respectively. Not shown in the figure but they consist of method calls to be invoked on an *AppComponent* or fetch/store data from a *DataSourceComponent* available to the *UIComponent* from the *InputType* and *OutputType* elements. *Tags* provide a mechanism to specify meta information about the component which could be used later to index the VoiceSite for searching purposes. *NavigationalProperties* capture the standard navigational, help or exception handling hyperlinks defined for this component.

The *UIComponent* element further gets derived into various types. The *OutputUIComponentType* allows specifying a UI control that renders an output to the user in the form of text, Audio, Video, SMS, MMS or an Image. It has an attribute *OutputResRef* that points to the location of the resource that is to be output to the user. A *ListType* component represents single or multidimensional lists. A list can be editable, sequential or a random (multi-)select list. The attribute *NumberofDimensions* determines the dimensionality of the list. An *UnstructuredInputType* is a UI component type for capturing input in the form of audio recording, free text or speech input to be recognized. Finally, an important subclass of *UIComponent* is a *HyperLink* component which could either represent a HTTP link or an HSTP [1] link. It has a *LinkText* attribute that represents the text that is presented in a GUI application or the grammar that is used in a voice application and is spoken to activate the link. It also contains a *LinkURI* element that points to the resource the hyperlink links to.

Third party vendors can extend this schema to define their own components called *Extended Templates*, described later in this paper.

3.3 Template Definition

Using VTDL, the developers create Template Definitions. A Template Definition captures the structure of a class of VoiceSites to be created and is essentially an XML document that specifies the UI components of the application and how they are interlinked to form an application flow. The UI components typically refer to other app components or data source components that are defined in the VTDL and have a realization available in the components repository of the framework. The instances of these app components and data source components, when included later, define the runtime behaviour of the VoiceSite generated from the template. The UI components in the template definition are abstract in nature and are rendered based on the deployment platform. For example, for a voice application the components are rendered in VXML whereas for a web based application they are rendered as HTML.

A template enables mass creation of a class of applications with similar features and the template definition captures all possible features of that application class. For creation of a template instance, the developer or the end user simply specifies which features to retain and configures those with data specific to him (through the interface provided). For each of the components included, the corresponding component realization, that renders or invokes this component should be available in the components repository of the framework. The template definition for the Village Portal scenario introduced earlier, is shown in Figure 8 and 9.

TWAF Components for Template Extensions

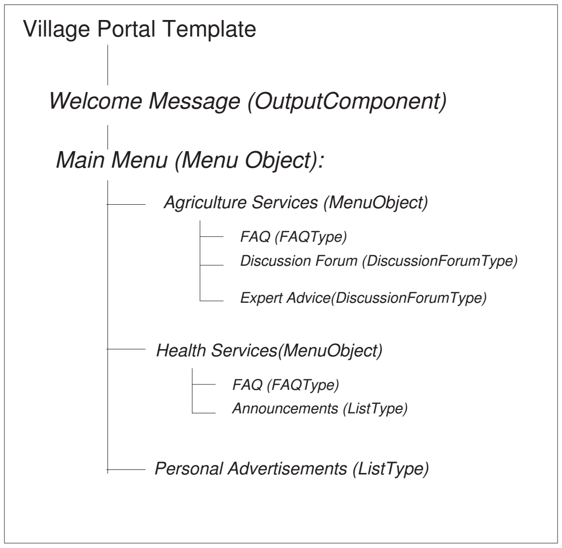

Village Portal Template
Welcome Message (OutputComponent)
Main Menu (Menu Object):
Agriculture Services (MenuObject)
FAQ (FAQType)
Discussion Forum (DiscussionForumType)
Expert Advice(DiscussionForumType)
Health Services(MenuObject)
FAQ (FAQType)
Announcements (ListType)
Personal Advertisements (ListType)

Figure 8: Template Definition for Village Portal

```
<tns:CompositeChoice Name="KioskMainMenu“ Label="">
   <tns:CompositeSequence Name="AgriService"
      Description="Agriculture discussion forums, announcements,FAQs and expert advice operation“  Label="">
   <tns:CompositeChoice Name="Agri Menu">
      <tns:LeafComponent Name="FAQList“ xsi:type="tns:ListType" Label="“ NumberOfDimensions="2">
      <tns:PreProcRef  xsi:type="tns:ListMethodCallType">
            <tns:methodName>getElements</tns:methodName>
            <tns:methodParams>
               <tns:category>OPEN</tns:category>
               <tns:numberOfElementsIn1Dim>5</tns:numberOfElementsIn1Dim>
               <tns:numberOfElementsIn2Dim>1</tns:numberOfElementsIn2Dim>
            </tns:methodParams>
       </tns:PreProcRef>
      <tns:PostProcRef> <tns:methodName>tns:methodName</tns:methodName>  </tns:PostProcRef>
      <tns:InputToApp Label=""></tns:InputToApp>
      <tns:OutputToApp>
            <tns:dataSink>tns:dataSink</tns:dataSink>
            <tns:parameterList>tns:parameterList</tns:parameterList>
      </tns:OutputToApp>
      <tns:Description>This is an FAQ on agriculture that is maintained by the voicesite administrator
       </tns:Description>
      <tns:NavigationalProperties>
            <tns:Help>http://tempuri.org</tns:Help>
            <tns:Forward>http://tempuri.org</tns:Forward>
            <tns:Back>http://tempuri.org</tns:Back>
            <tns:Home>http://tempuri.org</tns:Home>
            <tns:ErrorHandler>http://tempuri.org</tns:ErrorHandler>
       </tns:NavigationalProperties>
       <tns:tag>VillagePortalFAQ</tns:tag>
     </tns:LeafComponent>

     ............
```

Figure 9: Fragment of Template Definition from PoC

TWAF provides support for extension of its Template Schema to enable other developers and vendors to create Template Definitions specific to their needs and requirements. For this purpose, we define the notion of *TWAF Component* as a collection of artifacts that put together enable new VoiceSite functionality to be introduced into the TWAF framework. These artifacts consist of schema extensions, as well as executable code and related configuration files needed for the integration to happen.

3.4 Template Instance

A couple of actions need to be performed for creation of a VoiceSite Template instance from the corresponding VoiceSite Template definition. First, the user needs to select the components/ features available in the template definition that s/he needs in her VoiceSite. Second, each selected component has to be configured if required by the component. For instance, for a list of messages the component

may need to be configured to specify maximum number of messages to hold. In addition, selected components need to be customized by the user to reflect his/her organization, business or own personality. For instance, the prompts (if output is VUI) could be recorded in local language and may speak out the name and address of the business that the user owns. The interface provided to the user to perform these actions could be a VUI or a GUI. In either case, the TWAF framework needs to provide a meta-application (i.e. a meta-VoiceSite or a meta-website) for exposing the user interface to create a template instance. This meta-application could be generated from the template definition or could be developed manually when a new template definition is created.
As a result of this selection and configuration process, a new XML document is generated. In addition, the TWAF framework also generates a few configuration files that specify the mapping from component names used in the template to actual code and data files that realize those components. The XML document and the configuration file put together form the Template Instance. A VoiceSite Template Instance needs to be instantiated and executed in a runtime engine for others to be able to access it.

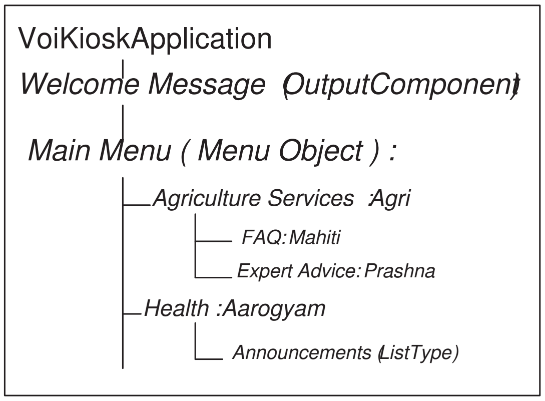
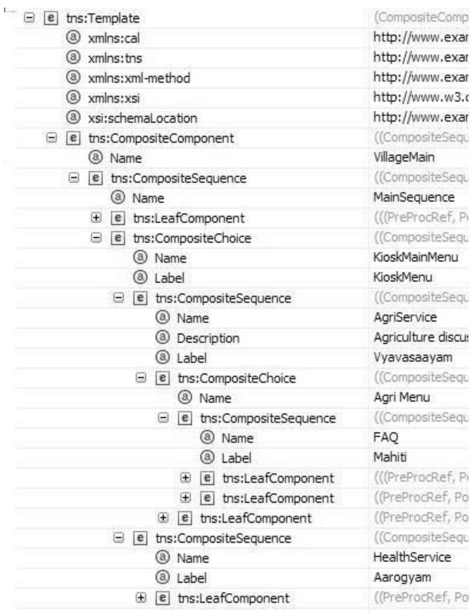


Figure 10: Village Portal Instance

An instance of the village portal template created by selecting the Agricultural services and the Health service only, is depicted in Figure 10. The services are labelled by the user as Agri and Aarogyam respectively. Figure 11 shows the screenshot of the corresponding XML file.

4. SYSTEM DESIGN AND IMPLEMENTATION

In this section, we present further details of the design of TWAF framework and our prototype implementation. As shown in Figure 12, the framework needs a Template Schema to start with. The framework also needs a library or repository of components and services that contains component code and stub code respectively. The Template Definition relies on this library to create the template. A *binding file* contains the mapping from the component name to the code module that realizes the component functionality. In addition, the framework has a component registry (not shown in figure) which contains an entry for all components (default components of TWAF as well as added by others). It can be looked up to determine whether a particular component exists in the repository.



```
⊟ e tns:Template                                   (CompositeComp
      @ xmlns:cal                                  http://www.exar
      @ xmlns:tns                                  http://www.exar
      @ xmlns:xml-method                           http://www.exar
      @ xmlns:xsi                                  http://www.w3.c
      @ xsi:schemaLocation                         http://www.exar
   ⊟ e tns:CompositeComponent                      ((CompositeSequ
         @ Name                                    VillageMain
      ⊟ e tns:CompositeSequence                    ((CompositeSequ
            @ Name                                 MainSequence
         ⊞ e tns:LeafComponent                     (((PreProcRef, P
         ⊟ e tns:CompositeChoice                   ((CompositeSequ
               @ Name                              KioskMainMenu
               @ Label                             KioskMenu
            ⊟ e tns:CompositeSequence              ((CompositeSequ
                  @ Name                           AgriService
                  @ Description                    Agriculture discu
                  @ Label                          Vyavasaayam
               ⊟ e tns:CompositeChoice             ((CompositeSequ
                     @ Name                        Agri Menu
                  ⊟ e tns:CompositeSequence        ((CompositeSequ
                        @ Name                     FAQ
                        @ Label                    Mahiti
                     ⊞ e tns:LeafComponent         (((PreProcRef, P
                     ⊞ e tns:LeafComponent         ((PreProcRef, Po
                  ⊞ e tns:LeafComponent            ((PreProcRef, Po
            ⊟ e tns:CompositeSequence              ((CompositeSequ
                  @ Name                           HealthService
                  @ Label                          Aarogyam
               ⊞ e tns:LeafComponent               ((PreProcRef, Po
```

Figure 11: Instance Screenshot

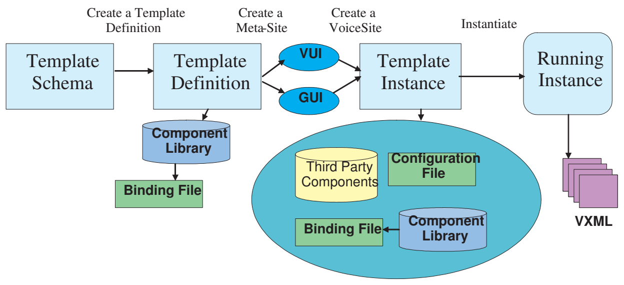


Figure 12: TWAF Detailed Design

TWAF's runtime engine is based upon Speakright[3] – an open source Java based framework for rapid development of voice applications using VoiceXML (VXML) and web applications using HTML. We first briefly describe Speakright to set the context for the remaining sections.

4.1 Speakright

Compared to other frameworks such as RDCs [4] and JSPs/VXML, Speakright uses a code based approach for developing voice/web applications. Applications are developed using Java which produces the VXML/JSP/HTML code at runtime. This approach enables developers to focus on application logic as opposed to the underlying presentation language. Speakright approach is more suitable for developing voice applications where no visual components are involved and hence are more amenable to the code generation approach. Web applications that are GUI intensive may not be suitable for this method though simple web applications can be generated. For our proof-of-concept implementation, we focussed on the use of Speakright for developing voice applications only.
Figure 13 shows the stack for a typical voice application involving Speakright. SpeakRight resides in the application code layer and has a Model-View-Control architecture sim-

[3]http://speakrightframework.blogspot.com/

ilar to GUI frameworks.

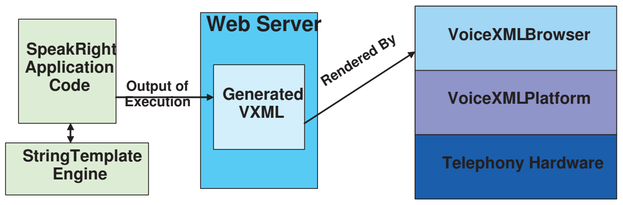


Figure 13: Speakright architecture

4.1.1 Flow Objects

Speakright uses the concept of *Flow objects* that form the basis of an application built using this framework. Flow objects form the *view* and *control* of the MVC architecture. A flow object encapsulates an interaction with the system where the user may be asked for an input or simply rendered an output. Each flow object is rendered as one or more VoiceXML pages during runtime. Flow objects can be extended using inheritance or composition. They can be nested to create larger flow objects. The application itself is a flow object containing one or more flow objects.
Flow objects implement the IFlow interface as given below:

- **IFlow getFirst():**
this retuns the first flow object to be run. A flow object with sub-flows would return its first sub-flow object. A leaf object (one with no sub-flows) returns itself.

- **IFlow getNext(IFlow current, SRResults results):**
getNext returns the next flow object to be run. It is passed the results of the previous flow object to help it decide. The results contain user input and other events sent by the VoiceXML platform.

- **void execute(ExecutionContext context):**
In the execute method, the flow object renders itself into a VoiceXML page.

4.1.2 Execution

Execution uses a flow stack. An application starts by pushing the application flow object (the outer-most flow object) onto the stack. Pushing a flow object is known as activation. If the application object's getFirst returns a sub-flow then the sub-flow is pushed onto the stack. This process continues until a leaf object is encountered. At this point all the flow objects on the stack are considered "active". Now the runtime executes the top-most stack object, calling its execute method. The rendered content (a VoiceXML page) is sent to the VoiceXML platform.
When the results of the VoiceXML page are returned, the runtime gives them to the top-most flow object in the stack, by calling its getNext method. This method can do one of three things:

- return null to indicate it has finished. A finished flow object is popped off the stack and the next flow-object is executed.

- return itself to indicate it wants to execute again.

- return a sub-flow, which is activated (pushed onto the stack).

The result is a new VoiceXML page that is generated. Execution continues like this until the flow stack is empty. Each flow object can invoke business logic operations before it is executed and after it completes execution.

4.2 VoiceSite Template Creation

VoiceSite Templates are creating using the VoiceSite Template Definition Language. For each of the UI components specified, there exists a Speakright class that renders that component. Thus every UI component in the template definition is represented as a flow object.
As discussed in the previous section, extension of existing VTDL constructs is feasible through the use of extended templates in the form of *TWAF Component*s. A typical TWAF component package includes the following:

- A *schema file* specifying extensions to basic constructs defined in VTDL and needed to interpret this component. Only extensions of existing components are allowed rather than arbitrary additions.

- A Java archive file (jar) containing either the entire component code or the stub code for the component. The stub code consists of the Speakright code corresponding to the flow object being represented by the component and the application logic such as interaction with a data source. To provide protected variation, we have defined a interface for the interaction of the component with the data source keeping in mind the Input/Output requirements of the component. The jar for each component includes the class implementing this interface.

- A set of prompt files, grammar files and audio files which are used by the stub code.

- a *binding file* providing a mapping of component names used in the template definition to the Java classes from the jar implementing the component as well as the prompt files, grammar files and audio files used by the component.

- a *deployment descriptor* specifying the locations for all of the above files.

Once an externally created *TWAF Component* is imported into TWAF, the framework updates its component registry. When a Template definition is being created, the TWAF framework exposes all the components - the default ones as well as extended ones obtained by searching for the extended templates in the component registry. The registry specifies the location of the extended template in the TWAF Component Repository. The TWAF Registry and TWAF Repository are part of the framework architecture.
The *TWAF Repository* contains all the extended templates. Once the extended template definition gets included in the Template Definition the rest of the steps carried out to instantiate a VoiceSite are the same as the ones followed during VoiceSite creation.
Figure 8 in Section 3 shows the VoiceSite template that we created to enable end users to create different portals for different villages.
The main component used for this template is the *ListType*. *FAQ* and *DiscussionForum* are derived from the *ListType*. The structure of the list component is shown in Figure 14.

```
<complexType name="ListType">
  <complexContent>
    <extension base="tns:UIComponentType">
      <attribute name="IsEditable" type="boolean"></attribute>
      <attribute name="IsMultiSelect" type="boolean"></attribute>
     <attribute name="NavigationLevel" type="tns:NavigationLevelType">
     </attribute>
     <attribute name="NumberOfDimensions" use="required">
       <simpleType>
          <restriction base="int">
             <minInclusive value="1"></minInclusive>
            <maxInclusive value="2"></maxInclusive>
          </restriction>
       </simpleType>
     </attribute>
    </extension>
  </complexContent>
</complexType>
<simpleType name="NavigationLevelType">
  <restriction base="string">
     <enumeration value="Sequential"></enumeration>
     <enumeration value="RandomSelect"></enumeration>
     <enumeration value=""></enumeration>
   </restriction>
</simpleType>
```

Figure 14: List Type

A FAQ type of list has a question and only one answer associated with it, while in a discussion forum we have a single level thread of communication. Each entry in the forum can have one or more replies or comments to it. We do not support multiple levels of discussion in this version. The Main Menu, Agriculture Services and Health Services are *CompositeChoice* components. The mapping of the remaining features are shown in Figure 8. The *ListType* has a *ListInterface* which specifies methods that can be used to initialize different types of lists and operations that can be performed on the list. The initialization methods are specified in the *PreProcRef* element defined in the *UIComponent*. Each of the leaf components in the template has a corresponding Speakright class which renders the UI part of the component and also implements the necessary interfaces of the component. This class is configured during instantiation and bound to the leaf component at runtime.

4.3 VoiceSite Creation

Instantiation of a voicesite from the corresponding template definition consists of the following activities:

- *Component Selection:* During instantiation, the users customize the template definition and select a subset of its features actually needed. The component selection module can be GUI based or VUI based. The user selects the desired features and fills in the *Label* attribute of the component. This value is used in the next step when grammars need to be generated. Once the components are selected, a new XML document is generated with the selected components and fed into the next step.

- *Grammar Generation:* A composite component that has a CompositeChoice or a Choice component, requires a grammar which specifies the word to be spoken by the user to select a component belonging to the choice component. The template definition specifies only the names of the components. The component labels that were populated by the user in the first step are now used to generate the appropriate grammars by the *Grammar Generation Module*. Grammars are generated only for *CompositeChoice* components. For the leaf components, if a grammar is required it is provided by the user and specified in the application configuration file.

- *Application Generation:* This is the final stage to create a running application. In this stage, in addition to the template instance XML document and the grammar files, the *binding file* and the *configuration file* are also required. As described earlier, while the binding file provides mapping from component names to corresponding java classes, the configuration file specifies parameters specific to components as well as home location for the generated grammars and location of audio files required for the application.

  During this stage, each component in the instance XML is parsed by a *Component Parser* and a Speakright class representing the component flow is generated. The composite components result in a composite flow that contain subflows corresponding to each component. The root flow represents the entire application and is now ready to execute. Figure 10 and 11 show a template instance for a Village Portal application.

In the proof-of-concept implementation, we developed a GUI based module to enable the users to create a template instance from the template definition. As shown earlier, we created an instance of the Village Portal Template. The user selects the features desired from the template and provides the labels for the features selected. Once the user selects the desired features and labels them, the grammar for *CompositeChoice* components are generated, by *Grammar Generator*, using labels. The Grammar Generator makes use of *Castor*[4] for this purpose. Finally, the *Application Generator* parses each component and sets the prompt file, grammar file and the Speakright class implementing each leaf component to produce an executable Village Portal VoiceSite. For generation Speakright classes too we made use of *Castor*. *CompositeChoice* components are instances of *MenuFlow* class in Speakright while *CompositeSequence* components are instances of a *BasicFlow* Speakright class which executes its subflows sequentially.

4.4 VoiceSite Deployment

A VoiceSite is deployed on an application server and rendered using a Voice Browser. In our case we used Tomcat Application Server and the Genesys Browser to deploy the generated voice applications. Each voice application has a phone number associated with it which acts as its URI. End users dial this phone number to access the voice application. The mapping between the phone number and the application is provided in the Genesys Voice Platform (GVP)[5]. For each application that is created, a *Deployment Descriptor* points to the location where the VXML code generated by the voice application is deployed on Tomcat. It also assigns a number for this application and configures the GVP accordingly. These VXML files (sample shown in Figure 15)

[4]http://www.castor.org
[5]http://www.genesyslab.com/products/genesys_voice_platform.asp

```
<?xml version="1.0" encoding="UTF-8"?><vxml xmlns="http//www.w3c.org/2001/vxml" version="2.0">
<form>
<catch event="connection.disconnect">
 <assign name="sr__res" expr="'1'"/>
 <submit next="router.jsp" namelist="sr__res" method="get"/> <exit/>
</catch>
<var name="sr__res" expr="'0'"/>
<field name="field1" >
<grammar type="application/srgs+xml" src="C:\sro1\KioskMainMenu.grxml"/>
<noinput>I'm sorry I didn't hear anything. What item would you like? </noinput>
<noinput count="2" >I still didn't hear anything.  Please say an item. </noinput>
<nomatch>I didn't get that. What item would you like? </nomatch>
<nomatch count="2" >I still didn't understand.  Please say an item you would like? </nomatch>
<prompt>Please select from the following VAgri health</prompt>
<nomatch count="3">
<assign name="sr__res" expr="'3'"/>
<submit next="router.jsp" namelist="field1 sr__res" method="get"/>
</nomatch>
<noinput count="3">
<assign name="sr__res" expr="'3'"/>
<submit next="router.jsp" namelist="field1 sr__res" method="get"/>
</noinput>
<filled>
<var name="sr__conf" expr="lastresult$.confidence"/>
<submit next="router.jsp" namelist="field1 sr__res sr__conf" method="get"/>
</filled>
</field>
</form>
</vxml>
```

Figure 15: Vxml Generated by Speakright

are then used by the Genesys browser and rendered when a user dials the number for the desired voice application.

5. RELATED WORK

There are several web application frameworks whose goals are similar to those of TWAF yet none of them serve the multiple goals that TWAF is designed to achieve.

Reusable Dialog Components (RDC) [4] is a framework for developing voice applications that requires integration of UI components and business logic at design time. It is a Struts based framework which has pre-defined Voice UI components that play a dominant part in the framework. Business logic related components possibly involving databases and web services also need to be wrapped as RDC components.

Hamlets [15] is an open source system for generating dynamic web-pages from XHTML templates. A Hamlet is basically an extension to a servlet. It reads XHTML template files and dynamically adds content where indicated by special tags, using callback functions. Hamlets provide an easy servlet-based content creation framework for web based applications and enforces the complete separation of content and presentation.

Java Server Faces (JSF)[6] and ASP.Net adopt a component based approach to web application development In ASP.Net, business code is connected to the UI components through events subscriptions. Events are generated by the components and stored in a separate file called *code behind* which resides along with the visual page layout.

DotNetNuke[7] is another web application framework considered ideal for creating, deploying and managing interactive web, intranet and extranet applications.

Apache Velocity[8] is a Template Engine for Java. It provides a simple yet powerful template language to reference objects defined in Java code. It enables Model-View-Controller (MVC) model based web application development by allowing web page designers to focus solely on creating a site with good user interface design while programmers can *parallely* focus solely on writing core business logic. In doing so, Velocity separates Java code from the web pages, making the web site more maintainable over its lifespan. It provides an alternative to Java Server Pages (JSPs) or PHP and can generate SQL, PostScript or XML from templates.

TWAF follows a component based approach similar to Microsoft ASP.NET and JavaServer Faces for the VoiceSite development process as this requires composition along the lines of enterprise application integration [22, 13]. On the other hand, for the runtime environment in which VoiceSites are instantiated and deployed it follows MVC Model 2 architecture since that portion of the framework is concerned with composition of presentation layer [22, 13]. TWAF is not a meta-framework and hence different from meta-frameworks such as Keel[9] and ROMA[10] that aim to provide an extensible platform for integrating various Java tools and frameworks together.

Reuse of code in addition to reuse of design is stressed in [17] and TWAF achieves code reuse through the use of component libraries that encapsulate application logic as well as reusable voice components that encapsulate presentation logic.

6. CONCLUSION

We presented the architecture, design and proof-of-concept implementation of TWAF – an end-to-end application framework for the World Wide Telecom Web - an emerging web for the underprivileged. The framework addresses a few non-trivial requirements compared to existing web application frameworks. It provides an API/GUI interface to software developers for creating VoiceSite Templates. In addition, it provides meta-VoiceSites that have a voice based interface for naive non-IT literate users to create their own VoiceSites. We demonstrated some of the important features of the framework through an end-to-end proof-of-concept implementation of a community VoiceSite.

The TWAF framework adopts best practices and follows a systematic, structured approach to enable rapid creation of VoiceSites by naive users. It also provides support for extension by third party component providers. In future, we intend to refine the APIs and the functionality of TWAF as well as develop the necessary tool to create a distributable software development kit for the World Wide Telecom Web.

7. REFERENCES

[1] S. Agarwal, D. Chakraborty, A. Kumar, A. A. Nanavati, and N. Rajput. HSTP: Hyperspeech Transfer Protocol. In *ACM Hypertext 2007*, UK, September 2007.

[2] S. K. Agarwal, A. Kumar, A. A. Nanavati, and N. Rajput. The World Wide Telecom Web Browser. In *WWW '08: Poster Proceedings of the 17th International World Wide Web Conference*, Beijing, China, 2008.

[3] S. K. Agarwal, A. Kumar, A. A. Nanavati, and N. Rajput. VoiKiosk: Increasing Reachability of Kiosks in Developing Regions. In *WWW '08: Poster Proceedings of the 17th International World Wide Web Conference*, Beijing, China, 2008.

[6]http://java.sun.com/javaee/javaserverfaces/overview.html
[7]http://www.dotnetnuke.com/
[8]http://velocity.apache.org/
[9]http://sourceforge.net/projects/keel/
[10]http://www.romaframework.org/

[4] R. P. Akolkar, T. Faruquie, J. Huerta, P. Kankar, N. Rajput, T. Raman, R. U. Udupa, and A. Verma. Reusable Dialog Component Framework for Rapid Voice Application Development. In *SIGSOFT Component Based Software Engineering*, Missouri, USA, May 2005.

[5] E. Brewer, M. Demmer, M. Ho, R. Honicky, J. Pal, M. Plauch, and S. Surana. The Challenges of Technology Research for Developing Regions. *IEEE Pervasive Computing*, 5(2):15–23, 2006.

[6] J. Chamberlain, G. Elliott, M. Klehr, and J. Baude. Speech user interface guide. 2006.

[7] B. Evjen, S. Hanselman, and D. Rader. Professional asp.net 3.5: In c# and vb. 2008.

[8] J. Holmes. Struts: The complete reference, 2nd edition. 2006.

[9] P. Kotkar, W. Thies, and S. Amarasinghe. An Audio Wiki for Publishing User-Generated Content in the Developing World. In *HCI for Community and International Development (Workshop at CHI 2008)*, Florence, Italy, April 2008.

[10] A. Kumar, N. Rajput, S. K. Agarwal, D. Chakraborty, and A. A. Nanavati. Organizing the Unorganized - Employing IT to Empower the Under-privileged. In *WWW '08: Proceedings of the 17th International World Wide Web Conference*, Beijing, China, 2008.

[11] A. Kumar, N. Rajput, D. Chakraborty, S. Agarwal, and A. A. Nanavati. Voiserv: Creation and delivery of converged services through voice for emerging economies. In *WoWMoM'07 Proceedings of the 2007 International Symposium on a World of Wireless, Mobile and Multimedia Networks*, Finland, June 2007.

[12] A. Kumar, N. Rajput, D. Chakraborty, S. Agarwal, and A. A. Nanavati. WWTW: A World Wide Telecom Web for Developing Regions. In *ACM SIGCOMM Workshop on Networked Systems For Developing Regions*, Aug 2007.

[13] H. Mili, M. Fayad, D. Brugali, D. Hamu, and D. Dori. Enterprise frameworks: issues and research directions. *Softw. Pract. Exper.*, 32(8):801–831, 2002.

[14] J. Panchard, S. Rao, P. T.V., H. Jamadagni, and J.-P. Hubaux. COMMON-Sense Net: Improved Water Management for Resource-Poor Farmers via Sensor Networks. In *International Conference on Information and Communication Technologies for Development*, Berkeley, USA, May 2006.

[15] R. Pawlitzek. Introducing Hamlets. *http://www-128.ibm.com/developerworks/web/library/wa-hamlets/*, Mar 2005.

[16] A. Rawsthorn. Design for the Unwealthiest 90 Percent. *http://www.iht.com/articles/2007/04/27/arts/design30.php. Last accessed Nov. 2008.*

[17] D. Schwabe, L. Esmeraldo, G. Rossi, and F. Lyardet. Engineering web applications for reuse. *IEEE MultiMedia*, 8(1):20–31, 2001.

[18] J. Sherwani, D. Yu, T. Paek, M. Czerwinski, Y. C. Ju, and A. Acero. Voicepedia: Towards speech-based access to unstructured information, interspeech 2007. In *Proc. Interspeech*, 2007.

[19] I. Singh, B. Stearns, M. Johnson, G. Murray, J. Inscore, L. Demichiel, N. Kassem, R. Sharma, R. Ortigas, R. Monzillo, S. Brydon, T. Ng, and V. Ramachandran. Designing Enterprise Applications with the J2EETM Platform, Second Edition. *http://java.sun.com/blueprints/guidelines/designing_enterprise_applications_2e/titlepage.html. Last accessed Nov. 2008.*

[20] C. E. Smith. Design for the Other 90%. In *Editions Assouline*, Sept 2007.

[21] I. W. Stats. World Internet Users and Population Stats. *http://www.internetworldstats.com/stats.htm*, June 2008.

[22] J. Yu, B. Benatallah, R. Saint-Paul, F. Casati, F. Daniel, and M. Matera. A Framework for Rapid Integration of Presentation Components. In *WWW '07: Proceedings of the 16th International World Wide Web Conference*, Banff, Canada, 2007.