Separation of Concerns in VoiceXML Applications

Sukhada P. Bhingarkar

Abstract— Many commercial applications provide customer services over the web like flight tracking, emergency notification, order inquiry etc. VoiceXML is an enabling technology for creating streamlined speech-based interface for such web-based information services. Whereas in computing, aspect-oriented programming (AOP) is a programming paradigm, which aims to increase modularity. AOP includes programming methods and tools that support the modularization of concerns at the level of the source code. The aim of this paper is to integrate AOP with VoiceXML. Aspect-Oriented Programming (AOP) encapsulates common low-level scattered code within reusable components called aspects. There are certain tags in VoiceXML like '<nomatch>', '<noinput>', '<error>' which appear commonly in every VoiceXML document. These tags can be considered as the concerns and can be put inside an aspect. This eliminates the need to programmatically write these tags in every VoiceXML document and modularizes the crosscutting-concerns.

Index Terms— AOP, AspectJ, ASR, IVR, TTS, VoiceXML, VXML

1 Introduction

The WWW has become primary source of information all over the world and accounts for the major proportion of entire Internet traffic. The next leading edge for the research on the web is to make it accessible via voice and audio. Considerable work has been done in this direction, which includes the design of VoiceXML and voice browsers. It allows voice applications to be developed and deployed in an analogous way to HTML for visual applications.

Aspect orientation is not a completely new approach to writing software. Aspect orientation is becoming a commonly adopted and de facto approach to practicing older ideas that can be traced to almost the beginning of software development. Development environments and tools that weave code, pragma instructions, and even debuggers all contain some of the behavior that underlies the aspect-oriented approach. It is a more modular implementation of the advantages that these technologies have brought to their own domains in the past. Aspect-oriented programming entails breaking down program logic into distinct parts called as *concerns.* All programming paradigms support some level of grouping and encapsulation of concerns into separate, independent entities by providing abstractions (e.g., procedures, modules, classes, methods) that can be used for implementing, abstracting and composing these concerns. But some concerns defy these forms of implementation and are called *crosscutting concerns* because they "cut across" multiple abstractions in a program. This paper tries to integrate this feature of AOP with VoiceXML. VoiceXML has certain tags which appear across the scope of every VoiceXML page. These tags are crosscutting concerns that can be defined in a modular fashion with the help of aspect-oriented programming. This helps modularization and code reuse in VoiceXML based applications.

The rest of the paper is organized in following way: section II presents related work in this field. Section III and IV discusses VoiceXML and AOP respectgively. The paper, then, proposes an integration of AOP with VoiceXML in Section V. Finally, the paper finishes with conclusion in Section VI.

2 Related Work

During past few years, several applications are developed to assist visually impaired, technologically uneducated and underprivileged people to acoustically access the information that was originally intended to be accessed visually via a personal computer (PC). Voice response facilities are used for various kinds of information over the phone: time, weather, horoscopes, sports, cultural events and so on. Nuance is one of the providers for speech and imaging solutions for businesses and customers around the world that is based on IVR systems. The VoiceXML along with Nuance 8.5 speech recognition software can reduce cost and effort of deploying voice-driven services. In [1], a VoiceXML-driven audio wiki application is presented that is accessible via both the Public Switched Telephone Network and the Internet. Users may access wiki content via fixed or mobile phones or via a PC using web browser or a Voice over IP service. Silog is a biometric authentication system that extends the conventional PC logon process using voice verification [2]. SeeCCT is a prototype of a multimodal social networking system designed for sharing geographical bookmarks [3]. In [4], VoiceXML portal is developed that allows people with a mobile or usual phone to get informed about cultural activities by dialing a phone number and by interacting with a computer via voice. Domain-specific dialogs are created in native languages viz. slavic languages using VoiceXML [5]. HearSay is a non-visual Web browser developed for visually impaired users [6].

AspectJ is one of the oldest and well-known aspect languages and it helped to bring AOP to the mainstream. AspectJ is an extension of the Java language which defines a special syntax for declaring aspects. The first versions of AspectJ featured compile-time source code weaving and bytecode weaving. It later merged with AspectWerkz which brought load-time weaving as well as AspectWerkz's annotation style to the language. AOP is used in a variety of fields. Aspect-oriented software development had played an important role in the design and implementation of PUMA which is a framework for the development of applications that analyze and transform C or C++ source code [7]. A framework for middleware design is invented which is based on the Concurrent Event-based Aspect-Oriented paradigm [8]. A fully dynamic and reconfigurable monitoring system is designed based on the concept of Adaptable Aspect-Oriented Programming (AAOP) in which a set of AOP aspects is used to run an application in a manner specified by the adaptability strategy [9]. The model can be used to implement systems that are able to monitor an application and its execution environment and perform actions such as changing the current set of resource management constraints applied to an application if the application/environment conditions change. An aspect-oriented approach is advocated as an improvement to the object oriented approach in dealing with the issues of code tangling and scattering in case of multilevel security [10].

3 VOICEXML

VoiceXML is HTML of Voice Web. It is W3C's standard XML format for specifying interactive voice dialogues between a human and a computer. VoiceXML is a markup language for creating voice user interfaces. It uses Automatic Speech Recognition (ASR) and/or touchtone (DTMF keypad) for input, and prerecorded audio and text-to-speech (TTS) synthesis for output. Numerous commercial vendors such as IBM, TellMe and BeVocal provide voice browsers that can be used to "play" VoiceXML documents. Current voice interfaces to the web are of two types: voice interface to screen display and voice-only interface. The dynamic voice interface presented in this paper is a voice-only interface that uses a telephone as an input and output device. Figure 1 shows the core architecture of VoiceXML applications.

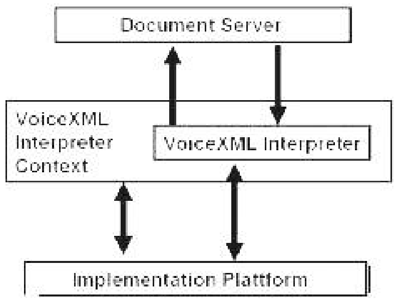


Fig. 1. Architectural Model of VoiceXML

A document server processes requests from a client application, the VoiceXML Interpreter, through the VoiceXML interpreter context. The server produces VoiceXML documents in reply, which are processed by the VoiceXML Interpreter. The VoiceXML interpreter context may monitor user inputs in parallel with the VoiceXML interpreter. The implementation platform is controlled by the VoiceXML interpreter context and by the VoiceXML interpreter.

4 ASPECT ORIENTED PROGRAMMING (AOP)

The main idea of AOP is to isolate the cross-cutting concerns from the application code thereby modularizing them as a different entity. A cross-cutting concern is behavior, and often data, that is used across the scope of a piece of software. It may be a constraint that is a characteristic of your software or simply behavior that every class must perform. The most common example of a cross-cutting concern is that of logging. Logging is a cross-cutting concern because it affects many areas across the software system and it intrudes on the business logic. Logging is potentially applied across many classes, and it is this form of horizontal application of the logging aspect that gives cross-cutting its name. Some central AOP concepts are as follows:

Aspects: A modularization of a concern that cuts across multiple objects. Transaction management is a good example of a crosscutting concern in J2EE applications.

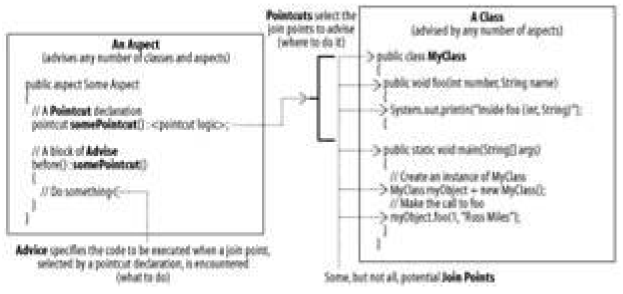
• *Sukhada Bhingarkar is with the Computer Engineering Dept., MIT collegee of Engineering, Kothrud, Pune, India – 411038*
E-mail:sukhada.bhingarkar@gmail.com

Join Point: A point during the execution of a program, such as the execution of a method or the handling of an exception

Advice: The code that is executed when an aspect is invoked is called *advice*. Advice contains its own set of rules as to when it is to be invoked in relation to the join point that has been triggered. Different types of advice include "around," "before" and "after" advice.

Pointcut: A predicate that matches join points. Advice is associated with a pointcut expression and runs at any join point matched by the pointcut (for example, the execution of a method with a certain name).

Figure 2 shows the relationships between join points, aspects, pointcuts, advice, and your application classes.


An Aspect
(advises any number of classes and aspects)
Pointcuts select the join points to advise (where to do it)
A Class
(advised by any number of aspects)
Advice specifies the code to be executed when a join point, selected by a pointcut declaration, is encountered (what to do)
Some, but not all, potential Join Points

```
public aspect Some Aspect
{
  // A Pointcut declaration
  pointcut somePointcut() : <pointcut logic>;

  // A block of Advise
  before() : somePointcut()
  {
    // Do something
  }
}
```

```
public class MyClass
{
  public void foo(int number, String name)
  {
    System.out.println("Inside foo (int, String)");
  }

  public static void main(String[] args)
  {
    // Create an instance of MyClass
    MyClass myObject = new MyClass();
    // Make the call to foo
    myObject.foo(1, "Russ Miles");
  }
}
```

Fig.2. Relationship of AOP concepts with application class [11]

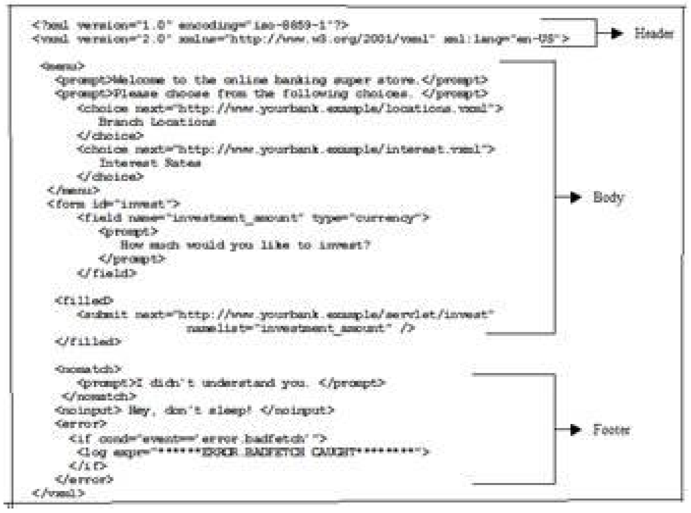
Figure 3 shows a simple business logic class along with an aspect applied to this class.

```
//Business logic class

package com.aop;
public class MyClass
{
     public void foo(int number, String name)
     {
          System.out.println("Inside foo (int, String)");
     }

     public static void main(String[] args)
     {
           MyClass myObject = new MyClass();
          myObject.foo(1, "Ram Sham");
     }
}
---------------------------------------------------------------------------------------

//An Aspect applied to the class

public aspect HelloWorld
{
  pointcut callPointcut() : call(void com.aop.MyClass.foo(int, String));

  before() : callPointcut()
  {
    System.out.println("Hello World");
    System.out.println( "In the advice attached to the call pointcut");
  }
}
```

Fig.3. Business logic class along with aspect

5 INTEGRATING AOP WITH VOICEXML

VoiceXML tags such as '<nomatch>', '<noinput>' '<error>' are usually common tags in most of the VoiceXML documents. Figure 4 shows VoiceXML example that illustrates basic capabilities of VoiceXML. This VoiceXML is logically divided into three parts: header, body and footer. The header part can contain tags showing xml and vxml version. The body part contains core business logic while footer part has error handing and <naomatch>, <noinput> tags.



```
<?xml version="1.0" encoding="iso-8859-1"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml" xml:lang="en-US">

 <menu>
   <prompt>Welcome to the online banking super store.</prompt>
   <prompt>Please choose from the following choices. </prompt>
      <choice next="http://www.yourbank.example/locations.vxml">
         Branch Locations
      </choice>
      <choice next="http://www.yourbank.example/interest.vxml">
         Interest Rates
      </choice>
  </menu>
  <form id="invest">
      <field name="investment_amount" type="currency">
         <prompt>
            How much would you like to invest?
         </prompt>
      </field>

   <filled>
      <submit next="http://www.yourbank.example/servlet/invest"
                     namelist="investment_amount" />
   </filled>

   <nomatch>
      <prompt>I didn't understand you. </prompt>
    </nomatch>
   <noinput> Hey, don't sleep! </noinput>
   <error>
     <if cond="event=='error.badfetch'">
      <log expr="******ERROR.BADFETCH CAUGHT********">
     </if>
   </error>
</vxml>
```

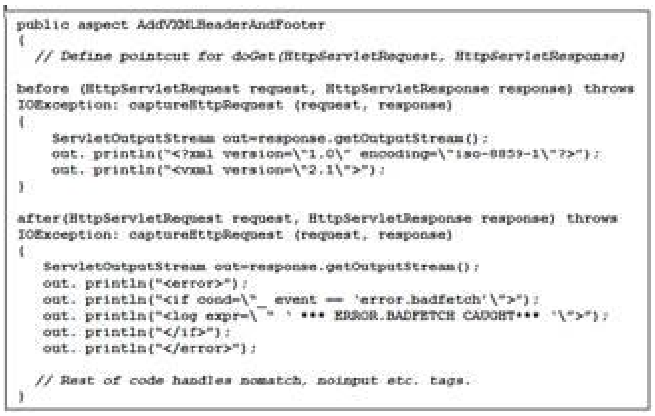
Fig. 4. VoiceXML example

The commonly occurring tags in VoiceXML can be considered as crosscutting concerns and are encapsulated in a special class i.e. an aspect. Figure 5 shows the Servlet that is used to construct VoiceXML.

```
public class AOServlet extends HttpServlet
{
   public void doGet(HttpServletRequest request, HttpServletResponse
   response) throws ServletException, IOException
         {
               ServletOutputStream out = response.getOutputStream();
               // core business logic code
         }
}
```

Fig. 5. Servlet generating VoiceXML code

doGet() or doPost() methods of Servlet concentrates only on core logic code i.e. body part of VoiceXML. The header and footer parts of VoiceXML will be taken care by an aspect class shown in figure 6.



```
public aspect AddVXMLHeaderAndFooter
{
  // Define pointcut for doGet(HttpServletRequest, HttpServletResponse)

before (HttpServletRequest request, HttpServletResponse response) throws
IOException: captureHttpRequest (request, response)
{
    ServletOutputStream out=response.getOutputStream();
    out. println("<?xml version=\"1.0\" encoding=\"iso-8859-1\"?>");
    out. println("<vxml version=\"2.1\">");
}

after(HttpServletRequest request, HttpServletResponse response) throws
IOException: captureHttpRequest (request, response)
{
   ServletOutputStream out=response.getOutputStream();
   out. println("<error>");
   out. println("<if cond=\"_ event == 'error.badfetch'\">");
   out. println("<log expr=\ " ' *** ERROR.BADFETCH CAUGHT*** '\">");
   out. println("</if>");
   out. println("</error>");

  // Rest of code handles nomatch, noinput etc. tags.
}
```

Fig. 6. Aspect applied to Servlet

6 CONCLUSION

VoiceXML is a language to create voice-user interfaces while AOP allows us to dynamically modify our static model to include the code required to fulfill the secondary requirements without having to modify the original static model. Integration of AOP with VoiceXML better separates the concerns of Voice based applications, thereby providing modularization. This helps developers to concentrate on core logic and promotes code reuse.

REFERENCES

[1] Constantinos Kolias et al, “Design and implementation of a VoiceXML-driven wiki application for assistive environments on the web”, Personal and Ubiquitous Computing, 2010, Volume 14, Number 6, 527-539, Springer-Verlag London Limited 2010

[2] Sergio Grau, Tony Allen, Nasser Sherkat, “Silog: Speech input logon”, Knowledge-Based Systems, Volume 22, Issue 7, October 2009, Pages 535-539

[3] Stan Kurkovsky et al., “Mobile Voice Access in Social Networking Systems”, in the Proceedings of 5th IEEE International Conference on Information Technology: New Generations, Las Vegas, USA, 7-9 Apr. 2008

[4] Evangelia Boufardea et al, “A Dynamic Voice Portal for Delivery of Cultural Content” in the Proceedings of 3rd International Conference on Internet and Web Applications and Services (ICIW’08), Athens, 8-13 Jun. 2008

[5] Brkic M., Matetic M., “VoiceXML for Slavic Languages Application Development”, in the Proceedings of IEEE international conference on Human System Interactions, Krakow, Poland, 25-27 May 2008

[6] Borodin Y, Mahmud J, Ramakrishman IV, Stent A (2007) The hearsay non-visual web browser. In: ACM international conference proceeding series, proceedings of the 2007 international cross-disciplinary conference on web accessibility (W4A), vol 225. Banff, Canada, pp 128–129

[7] Matthias Urban, Daniel Lohmann, Olaf Spinczyk, “The Aspect-Oriented Design of the PUMA C/C++ Parser Framework”. In: AOSD '10: Proceedings of the 9th International Conference on Aspect-Oriented Software Development, March 2010.

[8] Edgar Marques, Luís Veiga, Paulo Ferreira, “An extensible framework for middleware design based on concurrent event-based AOP”, ARM '10: Proceedings of the 9th International Workshop on Adaptive and Reflective Middleware, November 2010

[9] Arkadiusz Janik, Krzysztof Zielinski, “AAOP-based dynamically reconfigurable monitoring system”, Information and Software Technology, Volume 52 Issue 4, April 2010

[10] S. Kotrappa, Prakash J. Kulkarni, “Multilevel Security Using Aspect Oriented Programming AspectJ”, ARTCOM '10: Proceedings of the 2010 International Conference on Advances in Recent Technologies in Communication and Computing, October 2010

[11] Russell Miles, “AspectJ Cookbook”, O'Reilly, 2004