Combining VoiceXML with CCXML

A Comparative Study

Daniel Amyot and Renato Simoes
School of Information Technology and Engineering
University of Ottawa
Ottawa, Canada
damyot@site.uottawa.ca, renatops@yahoo.com

***Abstract*— Many Interactive Voice Response (IVR) systems use the popular VoiceXML standard for managing vocal dialogs. For call control aspects, such systems often use the Session Initiation Protocol (SIP) or a similar protocol. W3C is currently developing a new *Call Control eXtensible Markup Language* (CCXML) standard, at a higher abstraction level than SIP and which could hide the latter in order to accelerate the development of complex VoIP solutions that have an IVR component. But will this really be the case? This paper presents a comparative study base on a simple Personal Assistant system. Although there are undeniable benefits to a CCXML-VoiceXML approach, many observations and lessons lead us to believe that developers will face several limitations and potential pitfalls.**

Keywords— Call Control, CCXML, IVR, Telephone services, SIP, VoiceXML.

I. INTRODUCTION

Nowadays, both the public telephone network and enterprise telephony systems have become complex distributed and heterogeneous networks that integrate multiple technologies, recent and old ones alike. In the last decade, applications such as interactive voice response (IVR) have become widely available thanks in part to Voice-over-IP (VoIP) infrastructures, the Internet, and XML-based technologies. IVR applications are automated systems where the user interacts with a computer controller voice signal (recorded speech or computer generated) via telephone keys or speech recognition. Typical IVR services include information retrieval (banks, flights, taxes), virtual secretaries, bookings and payments, just to name a few.

The *Voice Extensible Markup Language* (VoiceXML, or simply VXML) is one technology that nowadays plays an essential role in the IVR world. With the recent growth of Internet users accessing information on the Web, information and service providers recognized the need to offer access to both telephone and Web users. VoiceXML allows one to access Web services using a voice interface, typically the telephone. This markup language enables the fast development and delivery of services based on Web technologies [1]. The W3C first standardized VoiceXML in 2000, with a second version available since 2004 [7]. Even if a standard exists, commercial VoiceXML servers do not always conform to it, which may result in code portability and reusability issues.

Typically, VoiceXML relies on the *Session Initiation Protocol* (SIP) for call control and session handling. However, VoiceXML is defined independently of SIP and can hence be used on top of other call control protocols. In the past few years, the W3C recognized the need for a call control language that would be based on Web technologies, and hence it initiated the standardization of a *Call Control eXtensible Markup Language* (CCXML). CCXML is designed to complement dialog systems such as VoiceXML by providing advanced telephony functions that better support VoIP-based features such as conference control, parallel/sequential Find Me Follow Me, click-to-dial, and many others. It also can be used as a third-party call control manager in any telephony system [6].

In this paper, we compare the conventional approach to design IVR applications that combines SIP as a control language and VoiceXML as the interaction language, with a newer approach that would use CCXML instead of SIP. Our goal is to understand what the benefits and drawbacks of using CCXML really are. To study and illustrate some of the concepts, we have developed a simple IVR application (a *Personal Assistant*, or simply *PA*) using VoiceXML on top of SIP, although the same could not be prototyped with CCXML and VoiceXML due to the absence of CCXML execution environments at the time this paper was written (this has recently changed, see [2][5]). However, we still illustrate how such combination would look like and use it as a basis for our comparison. While exploring various VoiceXML servers to support this experiment, we have also learned some interesting lessons on interoperability issues between these tools.

An overview of the main technologies involved (VoiceXML and CCXML) is presented in Section II. We assume the reader has some basic knowledge of SIP [3]. The Personal Assistant is first designed using VoiceXML and CCXML in Section III, whereas Section IV describes how such an application was implemented using VoiceXML and SIP directly. Both approaches are compared in Section V, and several lessons learned are included. We then conclude with a few remarks and future work items in Section VI.

II. Overview of VoiceXML and CCXML

A. VoiceXML Highlights

VoiceXML is a markup language that intends to bring the advantages of Web-based development to IVR applications. It is designed for creating audio dialogs that feature synthesized speech, digitized audio, recognition of spoken and DTMF key input, recording of spoken input, telephony, and mixed initiative conversations. Targeting a familiar client/server paradigm, VoiceXML brings the full power of Web development and content delivery to voice response applications and frees the developers from low-level programming and resource management issues.

In a typical VoiceXML-based architecture, Web servers maintain the overall service logic, perform database and legacy system operations, and produce dialogs. A VoiceXML document/script specifies each interaction dialog to be conducted by a VoiceXML interpreter. User input affects dialog interpretation and is collected into requests submitted back to the Web server. The latter replies with another VoiceXML document to continue the user's session with other dialogs. From a design perspective, VoiceXML minimizes client/server interactions by specifying multiple interactions per document, it separates user interaction code (in VoiceXML) from service logic (e.g., in CGI scripts), and it promotes service portability across implementation platforms.

There are however several issues with this language when used for IVR applications. For instance, when working as a call controller, VoiceXML offers limited advanced telephony functions, and the called party cannot be placed in an IVR. In the next sections we will introduce SIP and CCXML as alternative call controllers. Both can interact with VoiceXML and provide a rich telephony control.

B. CCXML Highlights

CCXML provides declarative markups used to describe telephony call control. As its name suggests, this scripting language is based on XML. It can be used with a dialog system such as VoiceXML, but the call control model in CCXML has been designed to be sufficiently abstract so that it can accommodate all major definitions including Parlay, SLEE, and JAIN SIP [4].

CCXML complements dialog systems by providing advanced telephony functions, by handling various asynchronous events, and by offering the ability to give each active call leg its own dedicated interpreter, hence overcoming the limitation of VoiceXML mentioned in the previous section. This language can also be used as a third-party call control manager in any telephony system; it supports sophisticated multiple-call handling and control, including the ability to place *outgoing* calls. CCXML is capable of receiving events and messages from external computational entities, interacting with an outside call queue, or placing calls on behalf of a document server. Call legs are considered as audio sinks and sources which can be combined to form arbitrary networks, in support of sophisticated conference call features.

Although the CCXML specification suggests a mechanism for passing information back and forth between CCXML and voice dialogs such as VoiceXML, these two languages are not mutually dependent. A CCXML implementation may or may not support voice dialogs, or may support dialog languages other than VoiceXML, but in either case the dialog language has to be changed to comply with this passing of information suggested in [6].

III. Personal Assistant with VoiceXML and CCXML

The call flow of our simple Personal Assistant application, based on the example found in [6], is illustrated in Fig. 1. It presents, from a user's perspective, both success and voice mail scenarios.

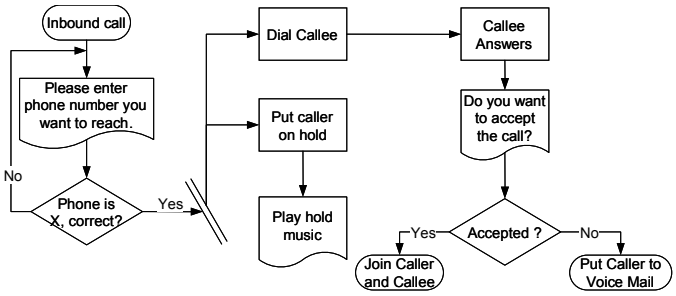

Figure 1. Personal Assistant call flow.

Fig. 2 shows the various components and interconnections in a typical CCXML-oriented deployment. Note that this is a logical view, as it is possible that in some implementation the Call Control and the VoiceXML Browser reside in a single physical box. CCXML is responsible for the signalling. The "? Stack" is the bridge between the CCXML and the VoiceXML parser, to be connected by a protocol unspecified in the CCXML standard. The Web Server is responsible for business logic and for on-the-fly generation of VoiceXML scripts.

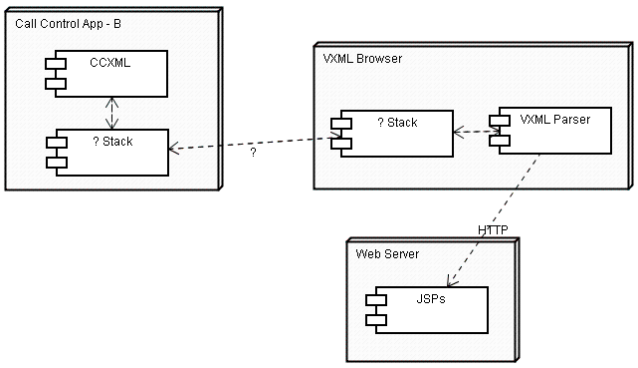

Figure 2. Deployment Diagram Using CCXML and VoiceXML.

Fig. 3 shows the CCXML script describing the behavior of our Personal Assistant. The code is broken down into transitions that are triggered by events. Each transition is tagged with a number on the left side; this number is not part of the language but it is used here for reference in the upcoming sections. XML comments (italicized) at the end of each block provide explanations for the code snippets.

```
<?xml version="1.0" encoding="UTF-8"?>
<ccxml xmlns="http://www.w3.org/2002/09/ccxml" version="1.0">
 <var expr="'initial'" name="currentstate" />
 <var name="in_connectionid" />
 <var name="dlg_onhold" />
 <var name="out_connectionid" />
 <var name="accepted" />
 <eventprocessor statevariable="currentstate">
  <!--this block defines and initializes variables used in the
   other blocks -->
```

1

```
  <transition event="connection.alerting" name="evt"
              state="initial">
    <assign expr="evt.connectionid" name="in_connectionid" />
    <accept />
  </transition>
  <!-- this block is triggered once the caller tries to connect
   into the system. The call is automatically accepted -->
```

2

```
  <transition event="connection.connected" name="evt"
              state="initial">
    <assign expr="'welcoming_caller'" name="currentstate" />
    <dialogstart src="'welcome_message.jsp'" />
  </transition>
  <!-- once the call is accepted and the caller is actually
  connected to the system, this block is triggered and the
  system places the caller into an IVR passing the system the
  initial URL src to be played. -->
```

3

```
  <transition event="dialog.exit" name="evt"
              state="welcoming_caller">
    <assign expr="evt.calleephone" name="tophone" />
    <!-- place the caller on hold -->
    <dialogstart dialogid="dlg_onhold"
      connectionid="in_connectionid" src="'holdmusic.jsp'" />
    <!-- Contact the callee. -->
    <assign expr="'contacting_target'" name="currentstate" />
    <createcall dest="'tel: ' + evt.calleephone"
                connectionid="out_connectionid" />
  </transition>
  <!--The welcome_message script is responsible for getting
  from the caller the callee's phone number. This script ends
  once it gets this information. This block is triggered once
  the dialog of this script exits. This block is then
  responsible for retrieving the number entered by the caller
  tophone), for putting the caller on hold, and for creating a
  new call leg to contact the callee (tophone) -->
```

4

```
  <transition event="connection.connected" name="evt"
              state="contacting_target">
  <!-- Ask the target if she would like to accept the call -->
    <assign expr="'waiting_for_target_answer'"
            name="currentstate" />
    <dialogstart src="'outbound_greetings.jsp'" />
  </transition>
  <!-- once the callee is connected, this block is triggered in
  order to place the callee into an IVR. This IVR session will
  ask the callee if he/she wants or not to accept the call from
  the caller -->
```

5

```
  <transition event="dialog.exit" name="evt"
              state="waiting_for_target_answer">
    <assign expr="evt.accepted" name="accepted" />
    <if cond="accepted == 'false'">
      <disconnect connectionid="out_connectionid" />
    </if>
    <assign expr="'stop_hold'" name="currentstate" />
    <dialogterminate dialogid="dlg_onhold" />
  </transition>
  <!-- the script that gets the response from the callee exits,
   and this block is triggered. The callee's input (accepted)
   is retrieved and checked for "false" or "true" content. If
   the content is "false" the callee leg is disconnected;
   otherwise the hold script that was being played to the
   caller is requested to be terminated -->
```

6

```
  <transition event="dialog.exit" name="evt" state="stop_hold">
    <if cond="accepted == 'false'">
      <assign expr="'voice_mail'" name="currentstate" />
      <dialogstart connectionid="in_connectionid"
                   src="'vm.jsp'" />
    <else />
      <join id1="in_connectionid" id2="out_connectionid" />
      <assign expr="'talking'" name="currentstate" />
    </if>
  </transition>
  <!-- once the hold script being played to the caller ends,
   this block is triggered and checks the accepted variable
   content again and decides if the caller should be redirected
   to the voice mail script, or the caller should be joined
   (bridged) to the callee. -->
```

7

```
  <transition event="connection.disconnected" name="evt">
    <if cond="evt.connectionid == in_connectionid">
      <exit />
    </if>
  </transition>
```

```
   <!-- once either the caller or the callee hangs up, the other
   party is disconnected and the CCXML draws to an end. -->

 </eventprocessor>
</ccxml>
```

Figure 3. CCXML call control script for the Personal Assistant.

One of the scenarios supported by the CCXML script in Fig. 3 is represented as a sequence diagram in Fig. 4 (labels have been added as comments to support traceability to the original script). The VoiceXML code is generated on the fly using several *Java Server Page* (JSP) scripts. For instance, `welcome_message.jsp` asks the user for entering a callee's phone number, `holdmusic.jsp` plays a sound file to the caller while the callee is being contacted, and finally the script `outbound_greetings.jsp` (shown below) asks the callee if he/she wants to accept the call, and passes the result back to the call control floor in the "accepted" variable:

```
<vxml version="1.0">
  <var name="callid" expr="'<%= callid %>'"/>
  <form id="callee_confirm">
    <field name="confirm" type="boolean">
      <prompt>
        <audio src="acceptcall.wav"/>
      </prompt>
      <nomatch count="1"> <reprompt/> </nomatch>
      <nomatch> <exit/> </nomatch>
      <noinput> <reprompt/> </noinput>
      <noinput count="3"> <exit/> </noinput>
      <filled>
        <if cond="confirm">
          <assign name="accepted" value="true" />
          <exit namelist="accepted" />
        <else/>
          <assign name="accepted" value="false" />
          <exit namelist="accepted" />
        </if>
      </filled>
    </field>
  </form>
</vxml>
```

IV. PERSONAL ASSISTANT WITH VOICEXML AND SIP

This section details the implementation of the PA application described in Section III, only this time using SIP directly as the call control language instead of CCXML. SIP was chosen as it is a widely used technology, and it has the capability of doing all the call control functionality that CCXML proposes to do, but with more efforts from the developer.

Unlike CCXML, SIP does not have a standard callback mechanism used between the VXML Browser and the SIP Server. In CCXML, this is hidden from the developer, but using SIP such callback must be defined and coded. Fig. 5 shows the callback mechanism we implemented with RMI from the Web server to the SIP server. Note that the response collected from the user via the IVR is actually first sent to the Web server using HTTP, rather than to the SIP server using RMI. This is not the only way to achieve such communication but it is sufficient to code an alternative development to CCXML that will help compare the approaches.

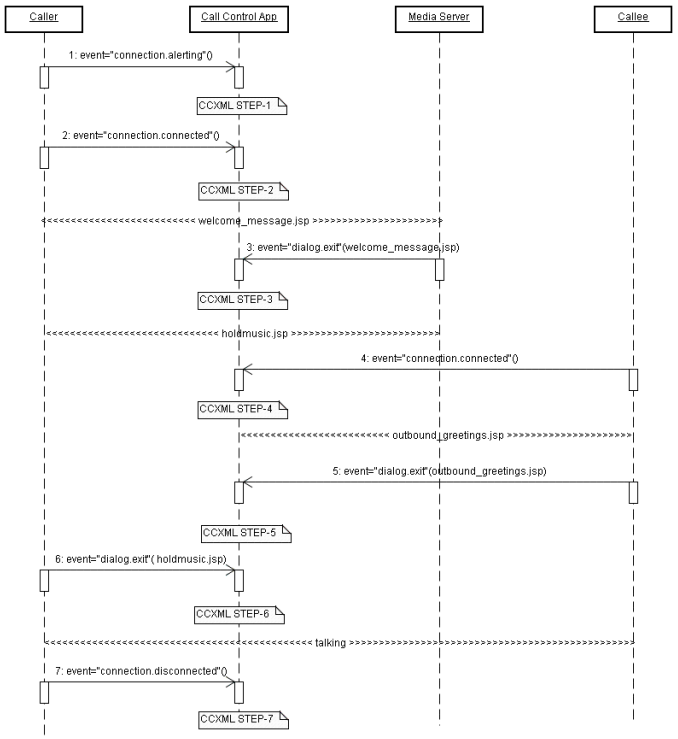

Figure 4. CCXML scenario - the callee has accepted the call.

In Fig. 5, the SIP server is responsible for the SIP signalling corresponding to the call control. A SIP stack exists in the VoiceXML Browser to interact with the SIP server. The Web server is responsible for business logic and for on-the-fly generation of VoiceXML documents, and for the RMI callbacks.

During this experiment, an actual implementation of the Personal Assistant was developed based on this architecture. The sequence diagram in Fig. 6 represents the same scenario as in Fig. 4 only this time it was generated from the SIP implementation call flow. Note that "CCXML STEP-N" comments are included to facilitate the correlation with the sequence diagram in Fig. 4.

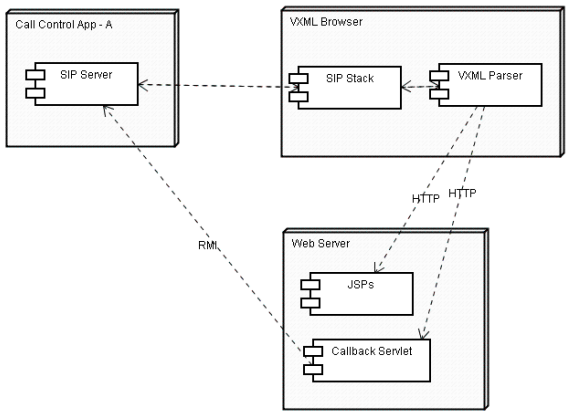

Figure 5. Deployment architecture based on SIP and VoiceXML.

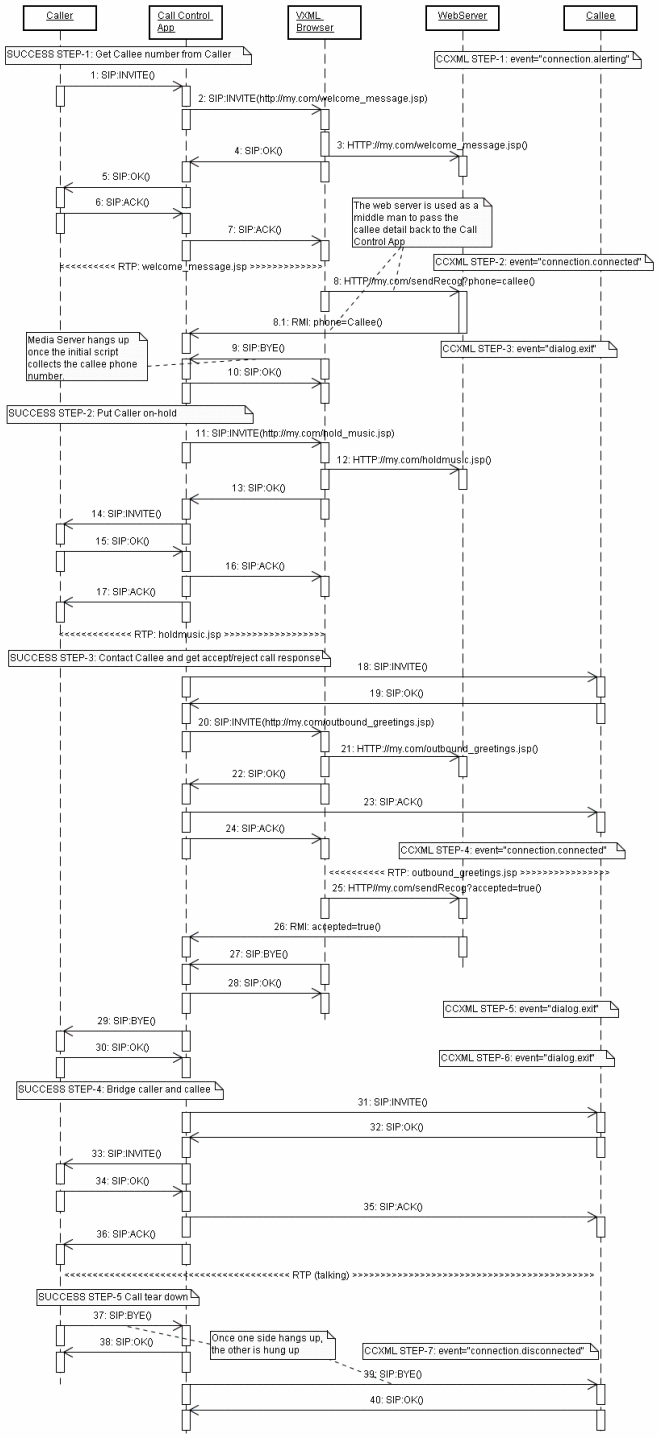

Figure 6. SIP scenario - the callee has accepted the call.

V. COMPARISON AND LESSONS LEARNED

Should developers move to CCXML or stick to plain SIP (or similar protocols) when writing IVR applications in VoiceXML? Our experiment and further study of the proposed CCXML standard led us to the following observations:

- *Code complexity*: The CCXML code is more concise than the equivalent SIP/Java code required by an order

of magnitude. SIP developers have to worry about many low level details that are abstracted by CCXML.

- *Flexibility*: SIP is more flexible and can in fact do all that CCXML can, whereas the opposite is not true.
- *Maintainability*: CCXML is generally simpler and easier to understand, however XML allows developers to make spelling errors that will not be caught until runtime, unlike the use of Java APIs for SIP.
- *Extensibility*: XML-based languages are by definition extensible, whereas SIP is practically static. However, the extensibility offered by XML at times leads to the proliferation of additions that break code portability and interoperability (recall what happened to HTML).
- *Coupling*: In a pure SIP approach, the SIP server and the media server are very decoupled and specialized: the first is responsible for call control, and the second for media and VoiceXML. In a CCXML approach, the CCXML server is expected to be strongly coupled to the VXML Browser and, as the protocol for communication between these two elements are not part of the specifications, they will probably have to either come from the same vendor, or they will both coexist as part of the same media server platform. That would make the media server handle some of the business logic, which goes against the specialization trend.

The above observations are summarized in Table 1, where the scores in bold represent the best characteristics:

TABLE I. COMPARISON SUMMARY.

Criteria	SIP + VXML	CCXML + VXML
Code Complexity	Higher	**Lower**
Flexibility	**Higher**	Lower
Maintainability	**Higher**	Lower
Extensibility	**N/A**	**Higher**
Coupling	**Lower**	Higher

Additionally, we have observed several issues related to the portability of the VoiceXML code itself. Incompatibilities were detected while using different VoiceXML media servers during our experiment, namely *Nuance's NVP 3.0* (http://nuance.com) and *Brooktrout's Snowshore A1* (http://www.brooktrout.com), and we have studied others available from *Convedia* (http://convedia.com) and *BeVocal* (http://www.bevocal.com). The following list of incompatibilities is not exhaustive, but it illustrates the current state of the VoiceXML world.

- *Shadow variables*: These are platform-specific variables used to modify or access additional information from a VXML tag. For instance, the Nuance `record` element has `name$.termchar` (that holds the key pressed by the user) and many others. Such variables differ from platform to platform.
- *Proprietary grammar*: Nuance is a speech recognition platform and a media server. In order to accommodate this capability, Nuance includes its own Grammar Specification Language, used inside the VXML code.
- *Proprietary tags*: Bevocal has its own extensions for the VoiceXML 2.0 specification, including tags like `<bevocal:listen>` and `<bevocal:whisper>`. Although a new XML namespace is defined, the logic may then get lost and hence portability is hindered. Nuance has also added special tags to support its voice recognition capabilities.
- *Partial VoiceXML support*: Whereas Nuance claims to support 100% of VoiceXML 2.0, Snowshore does not fully support VoiceXML 2.0 yet but claims not to have any proprietary tag.

VI. CONCLUSIONS

CCXML provides appropriate abstractions and offers capabilities that are easier and faster to use a plain SIP-based approach to the development of Call Control floor for IVR applications. The SIP-level flexibility may still be required in some situations, and one item for future work would be to determine if a hybrid solution is possible. VoiceXML still has a bright future ahead, and CCXML's will depend on the availability of good development and execution environments, which should start to appear this year.

At the time this paper was written, CCXML parsers were not available. However, since then, some initiatives from industry have led to CCXML implementations that interact or integrate with VoiceXML servers [2][5]. The simple Personal Assistant application described here could be implemented in CCXML, which would enable performance comparisons between the two approaches discussed in this paper.

As stated in Section V, although XML is good at representing data, using it to describe behaviour could lead to many issues when the language is abused by tool vendors and users through extensions and partial conformance to the standard, as we have observed for the simpler VoiceXML language. A standard API for generating multiplatform VoiceXML and CCXML could alleviate some of these issues.

REFERENCES

[1] E. Anderson, S. Breitenbach, T. Burd, N. Chidambaram, P. Houle, D. Newsome, X. Tang, and X. Zhu, VoiceXML Early Adopter, Peer Information Inc., 1st edition, August 2001.

[2] Hewlett-Packard, HP Open Call Media Platform, 2006. http:// h20208.www2.hp.com/opencall/products/media/ocmp/index.jsp

[3] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, SIP: Session Initiation Protocol, Request for Comments 3261, Internet Engineering Task Force, June 2002. http://www.ietf.org/rfc/rfc3880.txt

[4] Sun Microsystems, JSR-000032 JAIN SIP Specification (Final Release), Java Community Process JSR032, 2002. http://www.jcp.org/aboutJava/communityprocess/final/jsr032

[5] Voxeo Corporation, Prophecy IVR Voice Platform, 2006. http://www.voxeo.com/products/voicexml-ivr-platform.jsp

[6] W3C, Voice Browser Call Control: CCXML Version 1.0, W3C Working Draft 29 June 2005. http://www.w3.org/TR/ccxml/

[7] W3C, Voice Extensible Markup Language (VoiceXML) Version 2.0, Recommendation 16 March 2004. http://www.w3.org/TR/voicexml20/