DEFINITION AND DETECTION OF SEMANTIC WARNINGS FOR VOICEXML

Pattara Leelaprute and Tohru Kikuno
Graduate School of Information Science and Technology,
Osaka University
1-5 Yamadaoka Suita-shi, Osaka, JAPAN 565-0871
email: pattara@ist.osaka-u.ac.jp, kikuno@ist.osaka-u.ac.jp

Masahide Nakamura and Ken-ichi Matsumoto
Graduate School of Information Science,
Nara Institute of Science and Technology, Japan
8916-5 Takayama Ikoma, Nara, JAPAN 630-0192
email: masa-n@is.naist.jp, matumoto@is.naist.jp

ABSTRACT

VoiceXML is an XML-based language to describe interactive voice response services, which brings the advantages of Web-based development and delivery. Although the syntax of VoiceXML is prescribed by the VoiceXML DTD and the schema, the compliance with them is not a sufficient condition for the correctness of a VoiceXML script.

To detect the semantic flaws, we propose *semantic warnings* for VoiceXML. Focusing on the semantic aspects of voice interactive services, we define nine classes of warnings. Then, a warning detection framework with XPath and XSLT is presented. We also evaluate the advantages and limitations of the proposed method by qualitative discussions.

KEY WORDS

VoiceXML, semantic warnings, detection, XPath, XSLT

1 Introduction

In developing programmable services with the Internet, there is a major trend of using XML for representing platform-independent data, or even application programs themselves (called *XML scripts*). Many XML-based languages are currently standardized to specify various services, for instance; *WSDL* [20] for Web Services, *ebXML* [3] for electric commerce, *CPL* [9] for VoIP. The use of an XML-based language significantly improves the portability and interoperability of the programmable parts (including data and programs) of the service.

However, most XML-based languages are defined by DTDs (Document Type Definitions) or XML schemas, which cover *only syntax* of the target language. In general, the DTD and the schema are not expressive enough to specify *semantics* of the service. Therefore, there is a great possibility that non-expert developers make semantic flaws in the service logic, which cannot be validated by the DTD or the schema.

This paper especially focuses on *VoiceXML* (Voice Extensible Markup Language) [17] [18], which is designed to create services that can interact with user by voice or the DTMF (dialtone multi frequency, i.e., touch-tone) keypad. Many VoiceXML platforms are released for business use purpose (e.g., [5][15]). Services can build upon existing Internet standards and best practices to deliver a complete voice solution to the customers by integrating VoiceXML with their existing Web infrastructure. The syntax of VoiceXML is defined by the VoiceXML DTD and the schema. However, as far as we know, there is no definition or guideline by which developers can validate the semantic correctness of VoiceXML scripts.

The goal of this paper is to propose *semantic warnings* of VoiceXML, which are a simple guideline to detect the source of semantic flaws for any VoiceXML scripts. Focusing on the nature of voice interactive services, we define nine warning classes. Exploiting a practical VoiceXML platform, *OptimTalk* [11], we show that each warning can lead the platform to a problematic behavior.

We also present a framework to *detect* the semantic warnings. For this, we characterize each semantic warning as a problematic XML structure, and distill it using XPath [21]. Combining the XPath representation with XSLT [22], we implement a light-weight and generic warning detection framework well-feasible to unknown warnings that will be discovered in the future.

The proposed method is evaluated from the viewpoints of impact of warnings, property coverage, advantage and limitation of XPath, and completeness of the warnings. Integrated into development environments of VoiceXML applications, the proposed semantic warnings help developers to construct more semantically reliable voice applications.

2 Overview of VoiceXML

A *VoiceXML application* consists of one or more VoiceXML scripts that can call each other. A *VoiceXML script* is an XML document whose syntax definition is prescribed by the VoiceXML DTD and the schema. In the following, we pick up some important constructors to build a VoiceXML script. The full specification is found in [18].

Forms: A `<form>` is a basic dialog element to present information and gather user inputs, which is generally composed of several *form items*. The form items are subdivided into *input items* and *control items*. An input item specifies a *variable* (explained later) to gather from the user. Input items may have `<prompt>` elements to tell the user what to say or key in, `<filled>` elements to define the action

when the input is filled in, and *event handlers* (explained later) that process any resulting events. A typical input item is <field>, which assigns an input from the user to the associated variable via speech or DTMF recognition. A <field> may have <option> elements to offer a list of selective inputs. An <option> may specify attributes dtmf for an associated DTMF value, and value for an explicit value of the item. A <block> element is a typical control item that specifies an executable block procedure without user input.

Menus: A <menu> is a special form to prompt the user to make a choice and transitions to different places based on that choice. A <menu> has <choice> elements for making a list of selective items. A <choice> may specify a *text content* of an item, a next attribute for the URI of next dialog to be processed, and a dtmf attribute which explicitly associates a DTMF value to the item. For convenience, a <menu> may have dtmf attribute that implicitly assigns sequential DTMF digits to each of the first nine <choice>s. Inside a <menu>, an <enumerate> element that is used within a <prompt> element, automatically generates a prompt message telling all <choice>s available to the user. It is also used within a <field> that contains <option>s.

Variables: Variables in VoiceXML are declared by <var> elements, or by form items such like <field> with name attributes. A variable in <var> can be assigned by <assign>, while a variable in <field> is filled in when the user inputs the field. Variables may be referred in condition attributes (cond) or expression attributes (expr). ECMAScript[4] is used as the expression language in them. A <value> element returns the value of an expression. A <clear> element clears the current value. Each variable has a *scope*. If a variable is defined in a dialog element (i.e., <form>), its scope is within the dialog. If defined outside any dialog elements, the variable is treated as a global variable.

Control Flow: VoiceXML has several elements to operate the control flow of the script. For example, the elements <if>, <elseif> and <else> specify conditional branches. A <goto> element is used for the unconditional jump, while an <exit> terminates the script. A <submit> element is used to submit the gathered data to the server via HTTP GET or POST, by specifying variables in the namelist attribute.

Event Handling: A VoiceXML script has event handling that defines what to do when an event occurs. For example, a <noinput> element is triggered when user does not respond, and a <nomatch> element is triggered when some unrecognized input is given. The event-handling can have a count attribute which allows system to provide different actions when the same event occurs, depending on the number of occurrence.

Figure 1(a) shows an example VoiceXML script for a virtual restaurant where the user can order a dish from beef fish or chicken, and the cooking method.

This script has a <form> named "maincourse", and two <field>s named "maindish" and "cook", which are variables for the field.

By the field "maindish", the system first prompts a welcome message and enumerates (reads one by one) all options of Beef, Fish or Chicken. Then, the user selects a dish by saying one of the options, or by pressing a DTMF key. The DTMF values are defined in the <option>s. If the user does not respond until timeout, the <noinput> event occurs and prompts the user to answer again. After the user gives the matching input, the corresponding value attribute is set to the variable "maindish".

Subsequently, the field "cook" asks for the cooking method of the maindish. The user can answer 'grill', 'roast' or 'stew' (as specified in the external grammar. See [18] for the details). If the answer does not match, the <nomatch> event occurs and prompts the user for retry. After the user fills the matching input, the input is set to the variable "cook". Simultaneously, the <filled> element is executed and the <if> conditional branch is evaluated. Since the restaurant owner does not have *fish stew* in today's menu, the script does not accept the combination of 'fish' and 'stew' for the input. In this case, the value of 'cook' is cleared and the script prompts the user to select other cooking method for 'fish' again. Otherwise, the script submits the values of "maindish" and "cook" to the server as specified in the <submit>. Figure 1(b) shows an example of the dialog processed by the virtual restaurant specified in Figure 1(a).

3 VoiceXML Semantic Warning

Focusing on the nature of the voice interactive applications, we identify nine classes of warnings for VoiceXML scripts. For each warning, we give the definition, the effect and an illustrative example. Furthermore, we observe what actually happens, by executing the script with a practical VoiceXML platform — *OptimTalk* [11]. To save space, only the essential part of the script may be shown in each example.

3.1 Infinite Loop (LOOP)

Definition: Multiple <goto> elements form a closed loop from which the execution cannot escape.

Effect: An infinite loop imposes nonproductive endless actions. Moreover, the infinite loop could damage the system such like *stack overflow*, leading to the unexpected termination of the program.

Example: Figure 2 shows an example of LOOP. In this script, the second form "select_item" is called from the first form "welcome". Also, first form "welcome" is called from the second form "select_item". Since there is no path to escape this cycle, an infinite loop occurs. When we execute this script on OptimTalk,

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.w3.org/2001/vxml
   http://www.w3.org/TR/voicexml20/vxml.xsd">

<form id="maincourse">
  <field name="maindish">
    <prompt> Welcome to our restaurant.
      Please select your maindish. <enumerate/>
    </prompt>
    <option dtmf="1" value="beef"> Beef </option>
    <option dtmf="2" value="fish"> Fish </option>
    <option dtmf="3" value="chicken"> Chicken </option>
    <noinput> Please say one of <enumerate/></noinput>
  </field>

  <field name="cook">
   <grammar src="cooking.grxml"/>
    <prompt> You have selected <value expr="maindish"/>
      for the main dish. Please select 'grill',
      'roast' or 'stew' for the cooking method.
    </prompt>
    <nomatch> Sorry, please select only from 'grill',
      'roast' or 'stew'.
    </nomatch>
    <filled>
      <if cond="(cook=='stew') &amp;&amp;
        (maindish=='fish')">
        <prompt> Sorry, we have only
          'roast' and 'grill' for fish today.
        </prompt>
        <clear namelist="cook"/>
      <else/>
        <prompt> You have selected <value expr="cook"/>
          <value expr="maindish"/> for the order.
          Thank you for calling, goodbye.
        </prompt>
        <submit next="/cgi-bin/maincourse.cgi"
         method="post" namelist="maindish cook"/>
      </if>
    </filled>
  </field>
</form>
</vxml>
```

(a) VoiceXML script

System: Welcome to our restaurant.
Please select your maindish.
Beef
Fish
Chicken
User: (Silence)
System: Please say one of
Beef
Fish
Chicken
User: *fish*
System: You have selected fish for your main dish. Please select 'grill', 'roast' or 'stew' for the cooking method.
User: *steam*
System: Sorry, please select only from 'grill', 'roast' or 'stew'.
User: *stew*
System Sorry, we have only 'roast' and 'grill' for fish today.
User *grill*
System You have selected grill fish for the order.
Thank you for calling, goodbye.

(b) Dialog processed

Figure 1. Example of VoiceXML script for the virtual restaurant

```
<form id="welcome">
  <block>
    <prompt> Welcome to our virtual store. </prompt>
    <goto next="#select_item" />
  </block>
</form>

<form id="select_item">
  <block>
    <prompt> Sorry, our store is temporary closed
      for maintenance.
    </prompt>
    <goto next="#welcome" />
  </block>
</form>
```

Figure 2. Example of LOOP

the platform was terminated unexpectedly due to stack overflow.

3.2 Choice or Option Without Content (COWC)

Definition: A <choice> element (in a <menu>) or an <option> element (in a <field>) lacks the text content.

Effect: According to the schema, each <choice> (or <option>) has a text content of PCDATA that will become the selective subject for that choice. However, in case that the content is absent, there is no way for users to select that choice.

Example: We explain COWC with a correct VoiceXML script shown in Figure 3. This script is supposed to be a news system to provide choice of Todaynews, Weather or Sports. Now suppose that "Todaynews" in the first <choice> is absent. When executing the script, the user can only hear *Weather* and *Sports*. As a result, there is no way for the user to choose "Todaynews". With OptimTalk the script was unexpectedly terminated with an error message ("error.badfetch") when the user input any word other than "Weather" and "Sports".

3.3 Choice or Option Without Enumerate (COWE)

Definition: An <enumerate> element is not specified in any <prompt> elements in a <menu> (or a <field> with <option> elements).

Effect: An <enumerate> element is used for the system to read the items one by one, so that the user can listen

```
<menu>
  <prompt> Welcome to news service.
    Please say one of, <enumerate/>
  </prompt>
  <choice next="http://.../news/todaynews.vxml">
    Todaynews
  </choice>
  <choice next="http://.../news/weather.vxml">
    Weather
  </choice>
  <choice next="http://.../news/sports.vxml">
    Sports
  </choice>
</menu>
```

Figure 3. Example script used for COWC, COWE, IACB

```
<menu>
  <prompt> Welcome to news service.
    Please say one of, <enumerate/>
    or use DTMF to select the choice.
  </prompt>
  <choice dtmf="1" next="http://.../news/todaynews.vxml">
    Todaynews
  </choice>
  <choice dtmf="2" next="http://.../news/weather.vxml">
    Weather
  </choice>
  <choice dtmf="2" next="http://.../news/sports.vxml">
    Sports
  </choice>
</menu>
```

Figure 4. Example of OCCO

to the list and make a choice. If it is absent, the user cannot hear anything with regard to the choices, and does not know which choice should be chosen.

Example: Let us use the script in Figure 3 to explain. Suppose that the `<enumerate>` is absent from that script. When the script is executed, the user will hear nothing about the choices, thus is unable to make choices. Strangely however, if the user did input an appropriate choice, the selection was performed correctly with OptimTalk.

3.4 Identical Actions in the Conditional Branch (IACB)

Definition: The same actions are specified for all conditions of the conditional branch.

Effect: No matter which condition holds, the same action is executed. In such case, this conditional branch is meaningless and should be eliminated to reduce the structural complexity.

Example: In Figure 3, we replace all of the three URIs specified in `next` attributes with the same one. No matter which choice was taken, the same action was performed in the URI. This confused the user since the user doubted that his input had been wrong. In this case, the menu was completely redundant, so should be removed to improve the usability.

3.5 Overlapped Conditions in the Choices or Options (OCCO)

Definition: Multiple `<choice>` elements in a `<menu>` (or multiple `<option>` elements in a `<field>`) specify the same `dtmf` attribute or the same text content.

Effect: If multiple `<choice>` elements have the same `dtmf` attribute, a touch-tone input corresponds to multiple choices. Similarly, for the same text content (PCDATA), multiple choices could be taken by a single word of users. Thus, there is non-determinism which choice should be taken.

Example: Figure 4 shows an example. The second and third `<choice>`s have the same `dtmf` attributes (`dtmf="2"`). Therefore, if the user presses "2", a non-deterministic behavior occurs: which of Weather or Sports should be selected. With OptimTalk, the third `<choice>` jumping to "sports.vxml" was always selected when the user pressed "2". This also means that the user has no way to select the second choice for Weather with DTMF.

3.6 Redeclared Form items in Same Scope (RFSS)

Definition: A (single) name is used for declarations of different form items in the same scope, which result in a name re-declaration.

Effect: The VoiceXML specification does not specify the default behavior for the re-declaration for the redeclared form items. So, for a reference to the name, there is a non-determinism which item should be taken. This causes a platform-dependent issue.

Example: Figure 5 shows an example of RFSS. This script asks user for the clothes to order, and waits for the input. If the user answers 'tshirt', the `<goto>` attempts to jump to the field "select_size", to provide the selection of the size. However, since there exist two `<field>`s declared as "select_size", it is unclear which field should be taken. With OptimTalk, the field "select_size" that appeared first was selected. This is due to the implementation issue of OptimTalk. From the viewpoint of the user, it appears that no problem has occurred in the script.

3.7 Overlapped Counter in Event Handlings (OCEH)

Definition: Multiple event handlings in the same type specify the same `count` attribute and they become active simultaneously.

```
<form id="order">
  <field name="select_item">
    <grammar src="clothes.grxml"/>
    <prompt>What kind of clothes do you want to order?
    </prompt>
    <filled>
      <if cond="item=='tshirt'">
        <goto nextitem="select_size"/>
      <elseif cond="item=='pants'"/>
       . . .
      </if>
    </filled>
  </field>

  <field name="select_size">
    <prompt> Please select the size for your t-shirt.
    </prompt>
     . . .
  </field>

  <field name="select_size">
    <prompt> We have only size 'S' 'M' or 'L'.
    </prompt>
    . . .
  </field>
</form>
```

Figure 5. Example of RFSS

Effect: The `count` attribute in the same event handling allows the script to handle the same event differently, based on the number of the number of occurrence (counts) of the event. The specification of VoiceXML prescribes a *selection algorithm* to find the *best qualified* event handling element, when an event is thrown. Hence, even if there exist multiple elements with the same `count` attribute, only one of them is chosen to be executed by the algorithm. However, such overlapped counter just increases the complexity and unreachable event handlings, and thus should be avoided.

Example: Figure 6 shows an example. This script authenticates the user by the password within three times of challenges. There are four `<nomatch>` event handlings, each of which is activated when the user input unmatched password. The first three `<nomatch>`s are inherited to the child element `<field>`. Now, two `<nomatch>`s have the same counter (`count="3"`). So, handlings overlap when the user fails three times. Indeed, according to the selection algorithm, the latter `<nomatch>` ("You missed too many times. Bye") is always taken in this example. In this case, the former one is redundant, and should be removed to avoid the confusion.

3.8 Conflict of Implicit DTMF in Menu (CIDM)

Definition: A `<menu>` element with attribute `dtmf="true"` has a `<choice>` element with explicit `dtmf` attribute.

Effect: When the attribute `dtmf` is set to `true` in a `<menu>`, any `<choice>` that does not have explicit `dtmf` attributes are given the implicit sequential DTMFs "1", "2",...,"9". So, the `<choice>` that has an explicit `dtmf` attribute may conflict with the implicit one.

```
<form id="authenticate">
  <nomatch count="1"> Please try again.</nomatch>
  <nomatch count="2"> This is your last chance.</nomatch>
  <nomatch count="3"> We will connect you to the operator.
    <goto next="http://.../operator.vxml"/>
  </nomatch>
  <field name="password">
    <grammar src="authenticate.grxml"/>
    <prompt> Please say your password. </prompt>
    <nomatch count="3"> You missed too many times. Bye.
      <exit/>
    </nomatch>
  </field>
</form>
```

Figure 6. Example of OCEH

```
<menu dtmf="true">
  <prompt> Please select one of the following services.
    Please use DTMF 1-9 to select the choice. <enumerate/>
  </prompt>
  <choice next="#check_balance">
    Check balance
  </choice>
  <choice dtmf="1" next="#tranfer">
    Transfer money
  </choice>
  <choice next="#foreign_currency">
    Buy foreign currency
  </choice>
</menu>
```

Figure 7. Example of CIDM

Example: Figure 7 shows an example. In this script, DTMF values of 1, 2, and 3 are automatically assigned to each of the first three choices, because of the `<menu dtmf="true">`. However, since the second `<choice>` explicitly set `dtmf="1"`, the implicit assignment of the DTMF (=2) conflicts with it. OptimTalk caught an exception of "error.badfetch" when executing the script. It seems that OptimTalk implements the routine for detecting the conflict. Since OptimTalk could not execute this script because of the exception, the user could not use any service.

3.9 Usage of Undefined Variable (UUVB)

Definition: A variable is used before the value is set.

Effect: When a declared variable is used before the value is set, the value is assumed to be "undefined". This causes incorrect evaluation of conditions or expressions. Also, submitting the "undefined" variable to another URI may also cause unexpected input for the URI.

Example: We use Figure 1(a) to show an example. Assume that the field "maindish" is misarranged to be the last field in the form. Then, the conditional statement in the `<if>` will not be evaluated correctly because

the value of "maindish" is still undefined. Moreover, the undefined value of "maindish" will be submitted to "/cgi-bin/maincourse.cgi". In the script assumed, the system asks a cooking method first. However, since the value of "maindish" is still undefined, the message "You have selected *undefined* for the main dish" is prompted. Next, no matter which cooking method is given, the following `<if>` statement is not evaluated correctly. As a result, the script always takes the `<else>` and submits the undefined "maindish" to the server.

4 Detecting VoiceXML Semantic Warnings

This section proposes a framework to *detect* the semantic warnings in given XML scripts. Our key idea is to characterize each warning as a *problematic structure* of the XML document, and to distill the structure by exploiting *XPath* (XML Path Language) [21].

4.1 Principle of XPath Notation

XPath is a language to address arbitrary parts of an XML document [21]. Any node (or set of nodes) in a given XML document can be *located*. Also, XPath can apply various *functions* to the the nodes located. Specifically, these are performed by the following two steps:

Location: XPath addresses target nodes by a *location path* consisting of multiple *location steps* delimited by '/'. A location step is in the form of `axis::node test[predicates]`.

For example, `attribute::dtmf`, which means "select `dtmf` attribute of the current node", can be abbreviated to `@dtmf`. Also, `//choice` refers to all `<choice>`s that are descendants of the document root, which is equivalent to `/descendant-or-self::node()/child::choice`.

Functions: The functions apply various operations to the nodes located. We here introduce only several functions used in this paper. (1) $node$=$node$ or $node$!=$node$ computes the equivalence of $node$, (2) `string-length(`$string$`)` measures the length of $string$, (3) `contains(`$string_1$`,`$string_2$`)` checks the inclusion of $string_2$ in $string_1$, (4) `concat(`$string$`,`$string$`,` $string^*$`)` concatenates the $strings$, and (5) `count(`$node$`)` counts the number of $nodes$.

4.2 Describing Semantic Warnings with XPath

Using XPath, we try to describe each semantic warning. As an example, let us take COWC in Section 3.2. To detect COWC, we need to find any `<choice>` or `<option>` that do not contain any text contents.

First, we concentrate on a `<choice>`. Since a `<choice>` appears as a child of a `<menu>`, a location path `//menu/choice` selects all of `<choice>` elements that have some `<menu>` as direct parents. Next, an element `e` whose text contents is empty can be specified by `e[string-length(text())=0]`. Combining them together, we obtain `//menu/choice[string-length(text())=0]`, which can be read as "all `<choice>`s that do not contain any text contents". Finally, to check if a given VoiceXML script contains any such `<choice>`, we just have to know true or false of `count(//menu/choice[string-length(text())=0])>0`.

Similarly, we obtain the expression for `<option>`. Consequently, COWC can be described by the following XPath: "`count(//menu/choice[string-length(text())=0])>0`" or "`count(//menu/field[string-length(text())=0])>0`".

Table 1 presents XPath representation for all the proposed warnings.

4.3 Detecting Semantic Warnings

Once the semantic warnings are described in XPath, we just need to implement a warning detection system to generate the *detection report*. Taking full advantage of XPath, we exploit *XSLT* (Extensible Stylesheet Language Transformations) [22] for the report generation.

For each semantic warning, we describe a format of the report in XSLT, using the following XSLT syntax `<xsl:if test="XPath expression">` *warning message* `</xsl:if>`. The XSLT displays a certain message when the specified XPath expression becomes `true`. For example, a format rule for COWC can be described in XSLT as:

```
<xsl:if test=
   "count(//menu/choice[string-length
   (text())=0])>0"> COWC was detected.
   (content in <choice> is absent)
</xsl:if>
<xsl:if test=
   "count(//field/option[string-length
   (text())=0])>0"> COWC was detected.
   (content in <option> is absent)
</xsl:if>
```

We describe formats of all warnings in an XSLT file (let it be `warnings.xsl`). Then, for each given VoiceXML script, we embed a reference to `warnings.xsl` as a style sheet. Finally, we browse the VoiceXML script with a Web browser that implements the XSLT processor. The Web browser translates the given VoiceXML script into a warning report according to `warnings.xsl`.

Table 1. XPath representation of the proposed semantic warnings

Warning Name	Description in XPath
LOOP	count(//form[concat("#",@id)=following-sibling::form//goto/@next and //goto/@next=concat("#",following-sibling::form/@id)])>0
COWC	1. Content in <choice> is absent: count(//menu/choice[string-length(text())=0])>0 2. Content in <option> is absent: count(//field/option[string-length(text())=0])>0
COWE	1. <enumerate> in <menu> is absent: count(//menu[choice and count(prompt/enumerate)=0])>0 2. <enumerate> in <field> is absent: count(//field[option and count(prompt/enumerate)=0])>0
IACB	1. Identical action in <choice>s (within <menu>) exist: count(//menu[choice and count(choice[@next!=following-sibling::choice/@next])=0])>0 2. Identical action in <option>s (within <field>) exist: count(//field[option and count(option[@value!=following-sibling::option/@value])=0])>0
OCCO	1. Same dtmf attribute exists in <choice>s or <option>s: count(//menu/choice[@dtmf=following-sibling::choice/@dtmf])>0 or count(//field/option[@dtmf=following-sibling::option/@dtmf])>0 2. Content in <choice>s or <option>s are the same: count(//menu/choice[text()=following-sibling::choice/text()])>0 or count(//field/option[text()=following-sibling::option/text()])>0
RFSS	Case of redeclared <field>s: count(//field[@name=following-sibling::field/@name])>0
OCEH	1. Case of <nomatch>: count(//nomatch[@count=following::nomatch/@count])>0 2. Case of <noinput>: count(//noinput[@count=following::noinput/@count])>0
CIDM	count(//menu[@dtmf="true" and choice/@dtmf and contains("1 2 3 4 5 6 7 8 9", choice/@dtmf)])>0
UUVB	Case of <field>: (count(//field[@name and contains(preceding::value/@expr, @name)]) + count(//field[@name and contains(preceding::submit/@namelist, @name)]) + count(//field[@name and contains(preceding::if/@cond, @name)]) + count(//field[@name and contains(preceding::elseif/@cond, @name)])>0) and (count(//field[@name and contains(preceding::field/@name, @name)])+ count(//field[@name and contains(following::field/@name, @name)])=0)

5 Evaluation

5.1 Impact of Warnings

To investigate how serious each semantic warning is, we here try to classify the warnings into three *warning levels*; "critical", "moderate" and "tolerable". The classification is conducted with respect to viewpoints of both the developer and the users. In the following, we define the warning levels from the developer's view and user's view.

Critical Level
- *Developer's View*: Warnings causing a system down or unexpected script termination.
- *User's View*: Warnings making the service completely unavailable while the user is on the service.

Moderate Level
-*Developer's View*: Warnings causing unexpected behaviors, such as non-determinism and unspecified values. The behaviors vary heavily depending on the underlying system implementation.
-*User's View*: Warnings that can make users confused against his/her intention, generally causing usability problems of the service.

Tolerable Level:
-*Developer's View*: Warnings causing unreachable code or redundant parts in the script. Although the script is normally processed, these warnings may expose trivial mistakes, or restriction of some features that the developer intended to provide.
-*User's View*: Warnings that are not recognized by the user, but can restrict the original service behaviors that are supposed to be provided for the user.

Based on the execution results with OptimTalk (see Section 3), we classify the proposed warnings as shown in Table 2. The warnings in the critical level must be resolved

Table 2. The classification of the impact of semantic warnings

Level	Developer's viewpoint	User's viewpoint
Critical	COWC CIDM	COWC CIDM
Moderate	OCCO RFSS UUVB	IACB OCCO UUVB
Tolerable	IACB OCEH	RFSS OCEH

at the highest priority, since they may make the service totally unavailable. Resolving the moderate level should contribute to assuring the minimum quality of a script to deploy it as a *proper* service. This improves the portability of the script as well as the usability of the service. Finally, warnings in the tolerable level are helpful to eliminate redundant and ill-structured service logic, which improves readability and maintainability of the script. Thus, the proposed warnings can be used as a simple guideline to improve the quality of the script (service) from the semantic aspect.

5.2 Property Coverage

The key idea of the proposed warning detection framework is to characterize each warning by a *static* structure of the given VoiceXML script. This allows us to describe the nine warning classes smartly in XPath representation.

However, not all semantic properties can be described exactly in a static way. For example, LOOP (see Section 3.1) is a *dynamic property*, strictly speaking. That is, without executing the script, we would not be able to predict exactly how many `<goto>`s form an infinite loop. In fact, the XPath representation of LOOP in Table 1 covers only infinite loops with the length of 2. To detect loops with longer length, we can prepare an XPath expression for each length, and join them by disjunction. However, it is unrealistic to prepare expressions for all possible loops with any arbitrary length. Thus, the semantic properties covered by the proposed method are limited to the ones that can be reasonably described in static expressions.

5.3 Advantage and Limitation of XPath

In our warning detection framework, we extensively used XPath to describe and detect the proposed semantic warnings. The great advantage of introducing XPath is in its *generality*. Since XPath is a widely-known standard, we can reuse the existing modules and applications. Therefore, the implementation of the warning detection system becomes quite simple and easily-extendable, Also, since XPath is not tightly coupled with specific languages, the application of our framework is not limited within VoiceXML only, but is well feasible to detecting problematic warnings of other XML-based languages.

The limitation is in the *expressive power* of XPath. As the drawback of its generality, XPath is not always able to handle language-specific issues. For instance, VoiceXML adopts ECMAScript to specify conditions (as `cond` attribute) and expressions (as `expr` attribute) in the script. However, XPath does not have any means to parse strings written in ECMAScript.

Let us take UUVB (see Section 3.9) as an example. To locate a variable with the undefined value, we need to check every `cond` (and `expr`) attribute whether or not the variable exists in the attribute. However, since XPath cannot process the syntax of ECMAScript, we used the XPath `contains` function as a substitute for the exact matching of the variable, regarding the ECMAScript as an ordinary string. This imposes a restriction on UUVB that; a given VoiceXML scripts is not allowed to have any pair of names that the one overlaps another (e.g., `main` and `maindish`). To completely cover such language-specific issues, we may need to introduce more powerful languages, such as Perl and C, as a supplementary support.

5.4 Completeness

We have so far found nine classes of semantic warnings for VoiceXML. Then, we presented a warning detection framework with XPath, although LOOP and UUVB were partially covered as mentioned above. We believe that the proposed warnings provide simple but useful means for developers to build reliable VoiceXML scripts.

Of course, we cannot guarantee that the proposed nine classes are complete to cover all the semantic problems of VoiceXML. Further classes may be found in the future. Even in such a case, the proposed method is well feasible, as long as the new warnings can be described statically in XPath. In our future work, we plan to clarify how much of total semantic problems in VoiceXML can be covered by semantic warnings described in XPath.

We have examined only the semantic aspect of VoiceXML in this paper. However, investigating VoiceXML from the *security* viewpoint is another important problem to prevent the script from committing security incidents, such as *cross-site scripting* [1][10]. Currently, we consider that LOOP and UUVB could be abused for malicious security attacks, since LOOP can cause a DOS attack to a server, while UUVB can submit undefined values that cannot be processed correctly by the server. Formulating such security-related warnings for VoiceXML is a quite challenging issue and is also our long-term goal.

5.5 Related Work

Most published research in VoiceXML are about the application and integration, such as integration with the enterprise Web applications [2], and implementation of VoiceXML browser with SIP [12] of VoIP [13]. Few of research focus on the quality or correctness of VoiceXML script.

In [16], a high-level notation called *CRESS* is proposed to define and analyze voice interactive services. A service defined by CRESS can be compiled into formal descriptions such as LOTOS [7] and SDL [8], where the developer can evaluate the integrity of the service. CRESS is also able to compile the defined service into a VoiceXML script. The main difference of this approach and ours are; (a) CRESS cannot validate VoiceXML scripts directly, so the service needs to be given firstly in the CRESS representation. (b) CRESS does not specify any concrete semantic warnings, so the properties to be proven must be rigorously given by the developer.

The method presented in [14] tries to capture the *Web contents accessibility guideline* [19] with an XPath-based language called *SGSL*. This approach is similar to ours, in the sense that it distills a certain property as certain XML structures. However, it differs from ours significantly, since their target is *data* (i.e., HTML) while our target is *programs* (i.e., VoiceXML scripts). In our research, we chose XPath to take advantage of its generality. However, using SGSL might be a good option to capture more complex warnings.

6 Conclusion

In this paper, we have proposed nine classes of semantic warnings for VoiceXML and a warning detection method with XPath and XSLT. Each warning was applied to a practical VoiceXML platform – OptimTalk. It was shown that VoiceXML scripts with the semantic warnings can lead the platform to certain problematic situations. We have also evaluated the proposed method from several viewpoints. Practically, integrated into development environments of VoiceXML applications, the proposed semantic warnings help developers to construct more semantically reliable applications.

Finally, we summarize our future work. We are currently developing practical VoiceXML applications for a home network system [6]. Moreover, we investigate other types of warnings. Generalization of the semantic warnings for other XML-based language is also an interesting problem.

References

[1] CERT/CC, "Malicious HTML Tags Embedded in Client Web Requests", Feb. 2000, http://www.cert.org/advisories/CA-2000-02.html

[2] J. Chugh and V. Jagannathan, "Voice-Enabling Enterprise Applications", Proc. of Eleventh IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE'02), pp. 188-189, June. 2002

[3] ebXML Initiative, "ebXML - Enabling A Global Electronic Market", http://www.ebxml.org/

[4] Ecma International, "ECMAScript Language Specification (Standard ECMA-262)", http://www.ecma-international.org/publications/standards/Ecma-262.htm

[5] IBM, "Websphere Voice Server(TM), WebSphere Voice Toolkit", http://www-4.ibm.com/software/speech/enterprise/ep_1.html

[6] H. Igaki, M. Nakamura and K. Matsumoto, "Design and evaluation of the home network system using the service oriented architecture", Proc. of International Conference on E-business and Telecommunication Networks (ICETE2004), vol.1, pp.62-69, Aug. 2004.

[7] ISO/IEC, "Information Processing Systems – Open Systems Interconnection – LOTOS – A Formal Description Technique based on the Temporal Ordering of Observational Behavior", ISO/IEC 8807, International Organization for Standardization, Geneva, Switzerland, 1989.

[8] ITU, "Specification and Description Language", ITU-T Z.100, International Telecommunications Union, Geneva, Switzerland, 2000.

[9] J. Lennox and H. Schulzrinne, "Call processing language framework and requirements", Request for Comments 2824, Internet Engineering Task Force, May. 2000, http://www.ietf.org/rfc/rfc2824.txt?number=2824

[10] Microsoft Corporation, "Information on Cross-Site Scripting Security Vulnerability", Feb. 2000, http://www.microsoft.com/technet/Security/news/crssite.mspx

[11] OptimTalk, http://www.optimtalk.cz/

[12] J. Rosenberg, H. Schulzrinne, G. Camarillo, E. Schooler, A. Johnston, J. Peterson, R. Sparks, M. Handley and E. Schooler "SIP:session initiation protocol", Request for Comments 3261, Internet Engineering Task Force, Jun. 2002, http://www.ietf.org/rfc/rfc3261.txt?number=3261

[13] K. Singh, A. Nambi, H. Schulzrinne,"Integrating VoiceXML with SIP services", IEEE International Conference on Communications 2003 (ICC03), pp.784-788, May 2003.

[14] Y. Takata, T. Nakamura and H. Seki, "Automatic Accessibility Guideline Validation of XML Documents Based on a Specification Language," Proc. of 10th International Conference on Human-Computer Interaction (HCII 2003), pp.1040-1044, June 2003.

[15] Tellme Networks, "Tellme Studio", http://www.tellme.com/

[16] K. J. Turner, "Analysing interactive voice services", Journal of Computer Networks, Elsevier, Vol. 45, Issue 5, pp. 665-685, 2004.

[17] VoiceXML Forum, http://www.voicexml.org/

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

[19] W3C, "Web Content Accessibility Guideline (WCAG) Version 1.0 (W3C Recommendation)", May 1999, http://www.w3.org/TR/WAI-WEBCONTENT/

[20] W3C, "Web Services Description Language (WSDL) Version 1.1 (W3C Note)", Mar. 2001, http://www.w3.org/TR/wsdl

[21] W3C, "XML Path Language (XPath) Version 1.0 (W3C Recommendation)", Nov. 1999, http://www.w3.org/TR/xpath

[22] W3C, "XSL Transformations (XSLT) Version 1.0 (W3C Recommendation)", Nov. 1999, http://www.w3.org/TR/xslt