Dialogové systémy

Luděk Bártek

Laboratory of Searching and Dialogue, Faculty of Informatics, Masaryk University, Brno

spring 2023

VoiceXML

Basic Information

- Language for dialogue strategies description.
- Part of the W3C Voice Browser Activity standards.
- Objective:
  - To bring advantages of web development and content delivery into the interactive voice applications.
- History:
  - 1995 – started development of the AT&T Phone Mark-up Language.
  - 1998 – conference organized by W3C focused to the voice browsing the web:
    - presented languages PML, VoxML, SpeechML, TalkML, VoiceHTML, . . .

VoiceXML

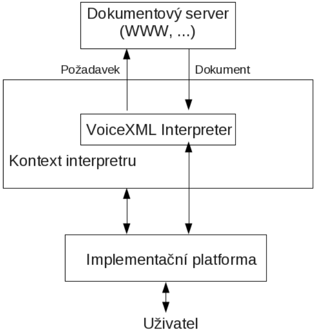
Basic Information

- History (cont):
  - 1999 – founded VoiceXML Forum – tries to merge effort to develop language for dialogues mark-up.
  - 2000:
    - released the VoiceXML 1.0 specification
    - the VoiceXML 1.0 specification accepted as the W3C Recommendation.
- Present state:
  - Recommendation – VoiceXML 2.1. (June 2007)
  - Draft – VoiceXML 3.0 (December 2010)

VoiceXML

Model of communication

Obrázek 1: Model of the Architecture of VoiceXML based applications (see VoiceXML 2.0 Recommendation)



VoiceXML

Applications Structure

- VoiceXML document(s):
  - Consists of forms.
  - User is in one of the conversational states at a given time.
  - The states transitions are defined by URI – they references the next dialogue step.
  - The dialogue ends when the transition is not defined.
- VoiceXML defines two types of dialogues:
  - Forms – defines process needed to obtain the values of a set of input fields.
  - Menu – offers a set of choices and references to next dialogue steps (forms).

VoiceXML

Applications structure – cont.

- Subdialogues:
  - analogy to the procedural program functions.
  - Used to perform particular part of the dialogue repeatedly, the input of e-mail address for example.
  - The subdialogues are realized using forms, they can get parameters and can return some value (see later)
- Session:
  - Starts when the user–VoiceXML interpret communication starts.
  - Finishes:
    - on user request (the connection termination request, the interpretation end request, . . .)
    - VoiceXML document – there is no defined transition to the next step, submitting data to the next processing, . . .
- Application – set of documents sharing the root document.

VoiceXML

- *vxml* – document root element.
- Must have following attributes:
  - *version* – used VoiceXML version
    - present version 2.1
    - the version must be supported by implementation platform – OptimTalk 1.9 – 2.1, JVoiceXML – partial support of version 2.1, VoiceGlue – version 2.0 support + some 2.1 options, . . .
  - *xmlns* – implicit name space declaration. It must contain the URI http://www.w3.org/2001/vxml.
  - *xml:lang* – the code of the interface natural language.
- The element contains:
  - One or more elements form,
  - element menu,
  - . . .

Form

- One of fundamental VoiceXML document elements.
- Bounded by tags $< form >$ a $< /form >$.
- Contains:
  - Set of input fields
  - a form's variables declaration – element *var*
  - defines grammars valid in an entire form
  - blocks of ECMAScript code.
  - ...
- Attributes:
  - *id* – mandatory attribute:
    - used as the form identifier
    - its value must be unique in the document
    - it can be used to transfer the control into the form.

Form

Form Interpretation Algorithm

- The Form Interpretation Algorithm (FIA) is the default algorithm used to interpret forms:
  1. Play all prompts those are child elements of the form.
  2. Repeat it until there is input field with undefined value.
     1. Select 1st suitable undefined field.
     2. Play all field prompts.
     3. Either acquire the field value value or process generated event (help, nomatch, . . .)
     4. Process the filled child of the input field.

Form
FIA – cont

- FIA may be terminated following ways:
  - The call should be transferred (using the element goto for example).
  - The data has to be send to the document server (the element submit).
  - The form should be explicitly terminated (the element exit).

Form
Example

```
<vxml version="2.0"
      xmlns="http://www.w3.org/2001/vxml"
      xml:lang="en-US">
 <form id="hello">
  <prompt>
     Hello world!
     This is our first VoiceXML form.
  </prompt>
 </form>
</vxml>
```

Form

Child Elements

- Input field - corresponds to different possibilities of how to enter form values:
  - field – user input, may be entered using a voice or dtmf.
  - record – used to record message from an user.
  - subdialogue – calls a dialogue processing some partial problem, entering address, date, . . . for example
- Control blocks:
  - block – a block of commands, can be used to output of data, input data processing, . . .
  - initial – the part of dialogue that is processed first. Used in mixed initiative dialogue strategy interfaces.
  - transfer – transfers user to a new location (application, human phone operator, . . .)
  - object – used to access platform depended functionality (dll, JSP+, servlet, . . .)

Form

Input Fields and Control Blocks – example

```
<vxml version="2.0"
   xmlns="http://www.w3.org/2001/vxml"
   xml:lang="en-US">
<form id="hello">
 <block name="hello">
  <prompt>Welcome to the VoiceXML!.</prompt>
 </block>
 <field name="greating">
  <prompt>Hello.</prompt>
  <grammar src="greatings.grxml"/>
  <noinput>
    <prompt>Tell mi something nice, like hello, hi,
     good day.</prompt>
  </noinput>
```

Form

Input Fields and Control Structures – Examples – cont.

```
    <nomatch>
      <prompt>
      I didn’t understand you, but thanks anyway.
      </prompt>
      <exit/>
    </nomatch>
    <noinput count="2">
     <prompt> When you don’t want to speek to me good
      bye.</prompt>
     <exit/>
    </noinput>
  </field>
  <filled>
    <prompt> you said <value expr="greating"/></prompt>
    <submit src="SomeURI" namelist="greating"/>
  </filled>
```

Field Element

- Represents a user input field. Either the voice or DTMF may be used to input data.
- Attributes:
  - name – The field name. Used to access the field value (using shadow variable with the same name).
  - expr – ECMAScript expression used to initialize the input field value.
  - cond – the condition that must be met to process the input field.
  - For more see specification.

Field Element

Cont.

- Content of the Element:
  - Prompt describing the value to enter (element prompt).
  - Grammar (element grammar) – a grammar describing the accepted answers.
    - Type of the grammar depends on used platform (on the used speech recognition module, for example Voxeo Prophecy, OptimTalk support SRGS, JVoiceXML supports JSGF,. . .).
  - Event handling:
    - noinput – no input from user detected
    - nomatch – the user input doesn't match the input field grammar
    - filled – allows to react on a correct input (on filling the input field)
    - . . .

Field Element

Usage Example

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0" xmlns="...">
 <form id="main">
  <field name="name">
   <prompt>Your first name</prompt>
   <grammar src="..." type="application/xml+srgs"/>
   <noinput>Enter your first name please
   </noinput>
   <nomatch>I'm sorry, but the value you enter does
       not match first names in a calendar.</nomatch>
  </field>
  <filled>
   <submit next="applicationURI" namelist="name"/>
  </filled>
 </form>
</vxml>
```

Element Record

- Used to record a message from user.
- It can be used to create voice recorder.
- Attributes:
  - name – input field name
  - expr – see the element field
  - cond – see the element field
  - beep – should be the start of recording signalled using a sound signal (beep)
  - maxtime – the maximum recording length
  - type – the recording mime-type, it must be supported by VoiceXML platform
  - . . .

Element record

- Element content:
    - Prompt(s) describing the requested input.
    - Possible event handling:
        - noinput – no user input detected.
        - connection.disconnect.hangup – user hang up prior the recording ended.

Record Element

Usage Example

```
<?xml version="1.0" encoding="utf-8"?>
<vxml version="2.0"
      xmlns="http://www.w3.org/2001/vxml">
 <form id="Recorder">
  <record  name="zaznam" beep="true" maxtime="30s"
   type="audio/x-wav">
   <prompt>I'm sorry but there is nobody you can
   talk to. You may left your message.</prompt>
   <noinput> I'm sorry but I don't hear anything.
   </noinput>
   <catch event="connection.disconnect.hangup">
    <submit next="http://some.uri.cz/recorder"/>
   </catch>
  </record>
 </form>
</vxml>
```

Element subdialogue

- Is used to call a subdialogue (a dialogue solving some partial problem, e.g.. input date, grade, . . .).
- The subdialogue can be called repeatingly with different parameters.
- Subdialogue calling:
  - element subdialogue – the subdialogue calling itself.
  - Contains:
    - param – parameter definition (its name and value).
    - filled – executable block handling what to do when the subdialogue ends.
  - Attributes:
    - name – subdialogue name. Is used as a shadow variable to access the returned value.
    - src – subdialogue form URI.
- Subdialogue code:
  - a form
  - terminated by element return.
    - the element return may contain parameter *namelist* containing the list of input fields of the subdialogues to be

Element subdialog

Example

```
<?xml version="1.0" encoding="utf-8"?>
<vxml version="2.0" xmlns="..." xml:lang="en-UK">
 <form id="demo">
  <block>
   <prompt>Example of using subdialogue
   </prompt>
  </block>
  <subdialog name="greating" src="#say_hello">
   <param name="param1" expr="'hi there'"/>
   <filled>
    <prompt>The subdialogue value is <value
expr="greating.great"/></prompt>
   </filled>
  </subdialog>
```

Element subdialog

Example – cont.

```
    <filled>
     <prompt>You said <value expr="greating.great"/>
     </prompt>
    </filled>
   </form>
  <form id="say_hello">
    <var name="param1"/>
     <field name="great">
      <prompt><value expr="param1"/></prompt>
      <grammar src="pozdrav.grxml"/>
      <noinput count="2">
        <prompt>You have not respond to the greating.
               Good bye.</prompt>
       <return/>
      </noinput>
```

Element subdialog

Example – cont.

```
    <nomatch>
     <prompt>I’m sorry I didn’t understand You,
            but I thank You anyway. Good bye.
     </prompt>
     <return/>
    </nomatch>
   </field>
   <filled>
    <return namelist="great"/>
   </filled>
 </form>
</vxml>
```

Element block

- Contains executable content.
  - attributes:
    - name – a block name.
    - expr – a shadow variable initial value.
    - cond – the condition it must be fulfilled to start the block execution.
  - structure – similar to the filled element:
    - control blocks – elements if, else, elseif
    - assignments – elements assign, clear, . . .
    - jump statements – elements goto, exit, return, . . .