

Ein kurzes Tutorial zu VoiceXML: Hauptseminar  
Dialogsysteme und VoiceXML in der Venice  
International University (VIU) Mai 2004

Christoph Ringlstetter  
Center for Information and Language (CIS)  
Ludwig-Maximilians-University of Munich  
kristof@cis.uni-muenchen.de

## Einleitung

VoiceXML ist ein XML-Dialekt, der für das Design von interaktiven Dialogsystemen benutzt wird. Wie alle XML-Anwendungen ist VoiceXML plattformunabhängig.<sup>1</sup> Als Input eines vollständigen VoiceXML Systems dienen entweder gesprochene Sprache oder DTMF Telefonsignale. Der Output ist gesprochene Sprache, die entweder mit Hilfe von Sprachsynthese oder durch Zusammenfügen vorher aufgenommenen Sprachfragmente erzeugt wird. Anwendungsgebiet für VoiceXML basierte Dialogsysteme sind interaktive Telefonanfragesysteme wie Flug-/Zuginformation (bereits realisiert), Weckservice (bereits realisiert) Voice Control Systeme für Fahrzeuge (teilweise realisiert) oder Banktransaktionssysteme (als Alternative zum Call Center bereits im Einsatz). Die Architektur eines VoiceXML Systems besteht aus Telefon, VoiceXML Plattform, Webserver und statischen VoiceXML Seiten. Die VoiceXML Plattform wiederum besteht aus folgenden teilweise optionalen Elementen: dem VoiceXML Interpreter, der automatischen Spracherkennung, einer Text to speech Komponente (TTS), der Audiowiedergabe, einer DTMF Schnittstelle sowie den Telefonierressourcen.

## 1 Zentrale Konzepte von VoiceXML 2.0

Ein VoiceXML Dokument kann als eine Art Finite State Dialog-Maschine gesehen werden. Der Benutzer ist zu jedem Zeitpunkt in einem wohldefinierten Konversationszustand. Übergänge von einem Zustand in einen anderen werden mit Unified Resource Locators (URL) definiert, die deterministisch zum nächsten Dialog führen. Die Ausführung einer VoiceXML Anwendung terminiert, wenn ein Dialog keinen Nachfolgedialog besitzt oder explizit ein `<exit>` bzw. `<disconnect>` Event ausgelöst wird. Nachfolgend werden Dialogfiguren in VoiceXML 2.0 aufgeführt:

- **Formulare `<form>`** : Definieren eine Benutzeraktion, die Werte für eine Menge von Feldvariablen `<field>` sammelt. Jedes Feld kann durch eine Grammatik mit Hilfe des `<grammar>` Elements spezifiziert werden, die den erlaubten Input festlegt.
- **Menüs `<menu>`**: Dem Benutzer wird eine Menge möglicher Optionen mit Hilfe von `<choice>` Elementen präsentiert. Dabei führt jede Option zu einem Nachfolgedialog oder einem Event.
- **Subdialoge `<subdialog>`** : Ähnlich zu Funktionsaufrufen in einer Programmiersprache. Sie starten eine Unterbrechung nach der das Programm zum übergeordneten (aufrufenden Dialog) zurückkehrt. Variablenbelegungen und geladene Grammatiken des aufrufenden Dialogs sind nach der Rückkehr und während des Subdialogs aktiv.

---

<sup>1</sup>Diese Unabhängigkeit trifft natürlich nur auf VoiceXML Plattformen zu, die auch den vollen VoiceXML 2.0 Standard einschließlich der W3C SRGF-Grammatiken implementieren.

## 2 Ein VoiceXML Document

Ein VoiceXML Dokument ist eine Kollektion von Dialogbeschreibungen, die in Formularen bzw. Menüs umgesetzt sind. Das nachfolgende “Hallo-Welt-Programm” soll wie gewöhnlich “Hallo Welt” ausgeben. Das Dokument beginnt mit einem Prolog, der die XML Deklaration enthält. Das Wurzelement eines VoiceXML Dokuments ist immer das `<vxml>` Element.

Beispielprogramm: “Hallo Welt”

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<vxml 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"
      version="2.0">

  <!--Hello World Program -->
  <form>
    <block>
      <prompt>Hallo Welt</prompt>
    </block>
  </form>
</vxml>
```

Da hier kein Nachfolgedialog in Form eines anderen `<form>` Elements oder eines `<menu>` Elements vorhanden ist, endet der Dialog nachdem das `<form>` Element abgearbeitet ist. Dieses sehr einfache Formular hat lediglich die Aufgabe “Hallo Welt” auszugeben. Dies wird mit dem `<prompt>` Element erreicht.

### Kurzüberblick: Das `<prompt>` Element

Das `<prompt>` Element kontrolliert die Ausgabe synthetisierter bzw. vorab aufgenommener Sprache. Es sind diverse Attribute zulässig um etwa das barge-in (die Möglichkeit des Benutzers “dazwischenzureden”) steuern bzw. um die Ausgabecharakteristika zu beeinflussen. Hierzu ist ein ganzes Arsenal von Subelementen der “speech synthesis mark up language” (SSML) verfügbar. Beispiel für die Formatierung der Ausgabe ist das `<emphasis>` Element.

```
<prompt>Bitte <emphasis>nennen</emphasis> sie den Abflugort</prompt>
```

Der Vollständigkeit halber sei angegeben, dass das `<prompt>` Element weggelassen werden kann, wenn sein Inhalt nur aus “parsed character data” (PCDATA) besteht, also kein Markup enthält. Oben könnte man also auch “`<block> Hallo Welt </block>`” schreiben. Man sollte allerdings aufgrund der besseren Lesbarkeit der Programme auf diese Verkürzung verzichten.

## Kurzüberblick: Das `<block>` Element

Das `<block>` Element enthält nicht-interaktive Anweisungen wie etwa eine prompt-Sequenz oder eine Reihe auszuführender Anweisungen. Für Block ist ein Namensattribut `name` definiert, mit dem die Ausführung des Blocks aus anderen Dokumentteilen gesteuert werden kann.

## Kurzüberblick: wichtige Konzepte von VoiceXML

- Sitzungen(sessions): Eine Sitzung wird eröffnet, wenn der Benutzer über die Input-Schnittstelle (Telefon, Stdin, Mikrofon) mit dem VoiceXML-Interpreter in Kontakt tritt. Sie fährt fort solange Dokumente geladen und bearbeitet werden. Die Sitzung endet, wenn der Benutzer, der Interpreter oder das Dokument das Ende herbeiführen.
- Applikationen: Eine Applikation ist eine Menge von VoiceXML-Dokumenten, die dasselbe Applikations-Wurzel-Dokument teilen. Solange der Benutzer mit einem Dokument der Applikation interagiert, bleibt das Wurzeldokument ständig geladen. Alle Dokumente der Applikation können auf die Variablen des Wurzeldokuments zugreifen. Die Grammatik des Wurzeldokuments bleibt für alle Blattdokumente aktiv.
- Grammatiken: Mit jedem Dialog sind eine oder mehrere Grammatiken verbunden. In geleiteten Applikationen sind nur die Grammatik des Wurzeldokuments und die des aktuellen Dialogs aktiv. In mixed-initiative Anwendungen können Dialoge flags enthalten, die ihre Grammatik aktiv macht. Falls der Benutzer etwas sagt (“hot word”) das durch eine dieser Grammatiken erkannt wird, hüpft der Interpreter zum zugehörigen Dialog.
- Events: Mit dem `<form>` bzw. `<menu>` Mechanismus werden “normale” Dialogverläufe modelliert, wo i. a. Formularfelder gefüllt werden. Von der VoiceXML-Plattform (Interpreter) ausgelöste “events” modellieren Abweichungen vom Standardverlauf.
  - keine Benutzerantwort: `<noinput>`
  - unverständliche Benutzerantwort: `<nomatch>`
  - Hilfeanfrage: `<help>`
  - Fehler in einem VoiceXML-Dokument `<error>`

Events werden in `<catch>` Elementen behandelt. Diese werden auf untergeordnete Dokumentenebenen vererbt (s.u.).

- Links: Mit dem `<link>` Element können mixed-initiative Dialoge modelliert werden. Matcht eine Benutzereingabe die jeweilige `<link>` Grammatik, so kann im Link zu einen neuen Ziel der Sitzung verzweigt werden.

## 3 Formulare als zentrales Konzept von VoiceXML

Ein Formular besteht aus folgenden Formular-Items:

- Zu interpretierende Items: Eine Folge von Elementen die bei der Interpretation des Formulars besucht werden. Diese Formular-Items werden in zwei Gruppen eingeteilt: die Input-Items die vom Benutzer gefüllt werden und die Kontroll-Items. Die Formular-Items haben eine Wächterbedingung. Per default testet der Interpretationsalgorithmus, ob die zugehörige Formular-Item-Variable einen Wert hat. Der Interpretationsalgorithmus besucht top-down das erste Item mit nicht erfüllter Wächterbedingung (default: dessen Variable keinen Wert hat).
- Event Handler
- Non-Form Item Variablen
- Filled Aktionen: prozedurale Blöcke, die bei bestimmten Kombinationen von Variablenbelegungen der Input-Items ausgeführt werden.

### 3.1 Attribute des Formularelements

- *id*: Name des Formulars
- *scope*: Gültigkeitsbereich einer im Formular definierten Grammatik. Die Belegung kann *dialog* oder *document* sein.

### 3.2 Input-Items

Input-Items spezifizieren Variable für die der Benutzer Werte eingeben soll.

- `<field>`:
- `<subdialog>`:

### 3.3 Kontroll-Items

- `<block>`: prozedurale Statement-Folge
- `<initial>`: initiale Interaktion in mixed initiative Formularen
- `<if>`, `<elseif>`, `<else/>`: Kontrollstruktur für Bedingungen

### 3.4 Attribute des Formular-Items block

*name*: definiert den Variablennamen des Blocks als Form-Item-Variable

*expr*: der initiale Wert (default) einer Variable ist undefined

*cond*: definiert einen Ausdruck der auf true auswerten muss damit auf das Item zugegriffen wird. Default-Wert ist true.

### 3.5 Das Formular-Item <field>

Zum “Verstehen” einer Benutzeräußerung wird in VXML das <field> Element eingesetzt. Es gehört zu den Input Items und dort zur Untergruppe der <form> Items. Im <field> Element wird zunächst die erforderliche Grammatik über das <grammar> Element aktiviert. Dann wird eine Systemäußerung über das <prompt> Element ausgegeben, die die Dialogsituation einführt also etwa den erwarteten Benutzer-Input mitteilt. Danach “hört das System zu” d. h. es wartet, bis entweder die Benutzeräußerung eine Regel der aktiven Grammatik erfüllt, ein <nomatch> Event bei unzulässigem Input ausgelöst wird oder durch ein *timeout* ein <noinput> Event ausgelöst wird. Schließlich wird die Reaktion auf das eingetretene Ereignis ( <filled>, <nomatch> oder <noinput> ) implementiert. Jedes <field> Element kann mit einem optionalen *name* Attribut eine sogenannte Form-Item-Variable im ECMA-Skript-Format definieren, die nach einer erfolgreichen Äußerung den Teil der Grammatik als String enthält, der gematcht hat. Im <filled> Element kann der Variablenwert der Field-Item-Variable(n) abgefragt und mithilfe von Kontrollstrukturen eine entsprechende Systemreaktion implementiert werden.

Beispiel einer ja/nein Auswertung

```
<if cond="alternativ-frage == 'ja'">
  <prompt> Sie haben ja gesagt </prompt>
<else/>
  <prompt> Sie haben nein gesagt </prompt>
</if>
```

#### Wichtige Attribute des <field>| Elements

- *name*: wird als Form-Item-Variable verwendet. Enthält *name* einen Wert, so wird das <field> Element vom Formular-Interpretations-Algorithmus (FIA) der VoiceXML Implementierung nicht berücksichtigt.
- *expr*: ECMA-Ausdruck <sup>2</sup> dessen Ergebnis *name* als Initialwert zugewiesen wird.
- *cond*: Ein ECMA Ausdruck der jedesmal ausgeführt wird, wenn der FIA das nächste auszuführende Element auswählt. Falls er zu “true” auswertet, wird das <field> Element berücksichtigt.
- *modal*: Kann “true” oder “false” sein. Bei “true” (default) sind lediglich lokale Grammatiken aktiv. “False” aktiviert alle anderen Grammatiken im Scope.
- *type*: Der angegebene Wert aktiviert eine Built-In-Grammatik, die hier zum Einsatz kommen soll.

---

<sup>2</sup>ECMA( European Computer Manufacturers Association )-Skript ist die programmtechnische Basis von VoiceXML. Es basiert hauptsächlich auf Javascript und JScript und wird von den großen Browseranbietern unterstützt. In der VoiceXML Spezifikation wird vom W3C seine Implementation auch für VoiceXML Browser verlangt. Code in ECMA-Skript wird durch das <script>| Element eingeleitet.

- *slot*: Alias zum *name* Attribut, für Grammatiken, die eine Äußerung in mehrere Teile zerlegen und an mehrere verschiedene `<field>` Elemente zurückgeben.

### Einfache Auswahllisten innerhalb des `<field>` Elements: Das `<option>` Element

Eine einfache Auswahlliste wird innerhalb des `<field>` Elements als Liste von `<option>` Elementen implementiert. Jede Option enthält einen PCDATA Grammatikausdruck, ähnlich den `<item>` Elementen im W3C-XML Grammatikformat. Im vorausgehenden `<prompt>` des Feldes wird das System durch das `<enumerate/>` Element veranlasst alle nachfolgenden Optionen auszugeben.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<vxml version="2.0">
<form id="auswahl">
  <field name="gebiet">
    <prompt> Wählen Sie eines der nachfolgenden Gebiete aus <enumerate/>
    </prompt>
    <option value="formal">
      Mathematik
    </option>
    <option value="praktisch">
      Programmierung
    </option>
    <option value="theoretisch">
      Linguistik
    </option>

    <filled>
      <prompt> Sie sind anscheinend <value expr="gebiet"/> interessiert
      </prompt>
    </filled>
  </field>
</form>
</vxml>
```

Übung: Welchen Wert hat "gebiet" jeweils, wenn das *value* Attribut weggelassen wird. Welche Eingaben sind zulässig, wenn das *accept* Attribut auf "approximate" gesetzt wird.<sup>3</sup> Lassen sie alternativ dtmf-Eingaben zu.<sup>4</sup>

Attribute des `<option>` Elements:

<sup>3</sup>Im von uns verwendeten VXML-Interpreter ist dieses Attribut nicht implementiert. Zum Bearbeiten der Übung ist deshalb die W3C Spezifikation zu verwenden.

<sup>4</sup>Hier kann es für die LINUX-Version der verwendeten Software Probleme mit dem DTMF-Simulator geben. In diesem Fall muss auf einer originären Shell ohne X oder KDE Unterstützung gearbeitet werden.

- `accept`: Mit dem Wert "exact" (default) wird exaktes matching verlangt, während "approximate" auch ähnliche Benutzereingaben zulässt.
- `dtmf`: statt Spracheingabe wird alternativ eine dtmf-Sequenz akzeptiert.
- `value`: Hier kann ein Zuweisungswert für die Feld-Item-Variable definiert werden. Andernfalls wird der dtmf-Wert bzw. die Option zugewiesen.

### Grammatiken innerhalb des `<field>` Elements

Mit dem `<grammar>` Element können innerhalb des `<field>` Elements Grammatiken definiert werden, die den zulässigen User-Input spezifizieren und den Rückgabewert an die ECMA-Field-Item Variable festlegen.

### Wichtige Attribute des `<grammar>` Elements

- `src`: Dieses Attribut enthält als Wert die URI einer externen Grammatik für das Feldelement. Inline Grammatiken und externe Grammatiken sind dabei alternativ, können also nicht gleichzeitig für ein und dasselbe Feld auftreten.
- `type`: Das `type` Attribut hat als Wert den MIME Typ der Feldgrammatik. Fehlt das Attribut so wird der Grammatiktyp mittels Code-Analyse festgelegt. Folgende Typen sind erlaubt:
  - `application/grammar+xml`: Das W3C XML Grammatikformat.
  - `application/grammar`: Das erweiterte W3C BNF Grammatikformat.
  - `application/x-gsl`: Die Nuace Grammatik Spezifikationsprache.
  - `application/x-jsgf`: Das Sun Java Grammatikformat.
- `root`: Für die Inline Verwendung einer Grammatik im W3C-XML Format muss ein `root` Element angegeben werden.
- `scope`: Für Grammatiken die innerhalb von Feldern spezifiziert werden ist immer das Feld selbst ihr Gültigkeitsbereich (`scope`). Das `scope` Attribut kann für diese Grammatiken nicht verwendet werden.

### Beispiele zum `<grammar>` Element innerhalb eines Feldes

Inline-Grammatiken: Der Inhalt des `<grammar>` Elements stellt die Regelmenge der Grammatik dar:<sup>5</sup>

```
<field name="kommando">
  <grammar root="command">
    <rule id="command">
      <ruleref uri="#action"/> <ruleref uri="#object"/>
    </rule>
    <rule id="action">
      <ruleref uri="#action"/> <ruleref uri="#object"/>
      <one-of>
```

---

<sup>5</sup>Beispiel aus [5].

```

        <item>open</item>
        <item>close</item>
    </one-of>
</rule>
<rule id="object">
    <one-of>
        <item>window</item>
        <item>file</item>
    </one-of>
</rule>
</grammar>
</field>

```

Externe Grammatiken: Im `<grammar>` Element wird über das `src` Attribut eine URI definiert, wo die Datei mit der Grammatik gefunden werden kann.

```

<field name="kommando">
<grammar src="./grammars/command.grxml"
</field>

```

### 3.6 Das `<filled>` Element auf Formularlevel

Die Verwendung des `<filled>` Elements ist in `<field>` und in `<form>` möglich. Auf Formularlevel kann sich das Element auf mehrere Input-Items beziehen. Es sind besondere Attribute möglich:

- *mode*: Als Werte sind "all" und "any" möglich. Über "any" wird festgelegt, dass das `<filled>` Element immer dann aufgerufen wird, wenn irgendeines der Input-Items des Formulars durch die letzte Benutzeräußerung gefüllt worden ist. Über "all" (default) wird das `<filled>` Element erst dann aktiviert, wenn alle Input-Items einen Wert haben.
- *namelist*: Durch Komma getrennt kann mit diesem Attribut eine Liste der für das `<filled>` Element relevanten Input-Items angegeben werden. Als default sind alle Input-Items des Formulars relevant.

### 3.7 Shadow Variablen im Scope von `<filled>`

Unabhängig davon, ob das `<filled>` Element innerhalb oder außerhalb eines Formulars verwendet wird, stehen für das Element Shadow Variablen zur Verfügung um die Benutzeräußerung genauer zu untersuchen. Der Name einer solchen Variablen besteht aus dem Wert des *name* Attributs des jeweiligen Input-Items, gefolgt von "\$." und dem Shadowattribut.

- `name$.utterance`: Enthält die Benutzeräußerung als String.
- `name$.confidence`: Enthält einen Konfidenzwert zwischen 0.0 und 1.0 für die Erkennungsqualität.
- `name$.inputmode`: Hier ist entweder "dtmf" oder "voice" abgespeichert.
- `name$.interpretation`: Enthält die Interpretation der Benutzeräußerung durch die Grammatik.

Beispiel zu den Shadow-Variablen des <field> Elementes

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<vxml version="2.0">
<form id="auswahl">
  <field name="gebiet" >
    <prompt> Für was interessieren Sie sich?
    </prompt>
    <grammar root="main">
      <rule id="main">
        <ruleref special="GARBAGE"/>
        <one-of>
          <item tag="MATHEMATIK"> Mathematik          </item>
          <item tag="PROGRAMMIERUNG"> Programmierung </item>
          <item tag="LINGUISTIK"> Linguistik          </item>
        </one-of>
        <ruleref special="GARBAGE"/>
      </rule>
    </grammar>

    <filled>
      <prompt> Sie sind anscheinend an <value expr="gebiet"/> interessiert
      </prompt>
      <prompt> Geäußert haben Sie <value expr="gebiet$.utterance"/>.
      </prompt>
      <prompt> Interpretiert haben wir <value expr="gebiet$.interpretation"/>.
      </prompt>
      <prompt> Es wurde mit Konfidenz <value expr="gebiet$.confidence"/> erkannt
      </prompt>
      <prompt> Der Inputmodus war <value expr="gebiet$.inputmode"/>
      </prompt>
    </filled>
  </field>
</form>
</vxml>
```

## 4 Der Formular Interpretations Algorithmus

Der Formularinterpretationsalgorithmus besteht aus einer Hauptschleife mit drei Phasen:

- **select:** Das nächste nicht gefüllte Formularitem wird ausgewählt. Falls man von der letzten Iteration eine "goto nextitem" Anweisung hat, so folgt man dieser. Sonst wird das erste Item gewählt, dessen "guard Bedingung" falsch ist.
- **collect:** Das selektierte Item wird besucht. Es erfolgen Prompt-Ausgabe und Grammatikaktivierung. Der Algorithmus wartet dann auf Eingabe

und wirft gegebenenfalls einen `<noinput>`-timeout.

- **process:** Eingabeauswertung, indem die Formular-Item-Variablen gefüllt werden. Filled-Elemente werden aufgerufen. Etwa als Eingabevalidierung. Events werden durch den Eventhandler behandelt. Das Ende der Interpretation des Formulars wird erreicht, wenn keine Items mehr ausgewählt werden können oder ein Kontroll-Transfer-Element wie `<goto>` erreicht wird.<sup>6</sup>

## 5 Elemente der Dialogsteuerung und Multidokumentapplikationen

### 5.1 Elemente der Dialogsteuerung

1. Wiederholung eines Dialogabschnitts: Hierzu wird das `<reprompt>` Element verwendet. Durch einen Reprompt wird der Formular Interpretationsalgorithmus veranlasst das aktuelle Feld noch einmal aufzurufen. Jeder Reprompt erhöht das *count* Attribut. Die Elemente `<nomatch>` und `<noinput>` erzeugen automatisch einen Reprompt.
2. Ausführen des nächsten Dialogabschnitts: Ist ein Dialogelement abgeschlossen, etwa durch `<filled>`, so wird automatisch top-down im VoiceXML Dokument das nächste Element angesprochen. Soll zu einem anderen Dialogelement oder zu einem anderen Dialog gesprungen werden, so benutzt man das `<goto>` Element. Die Sprunganweisung zu einem anderen Feld-Item kann beispielsweise so aussehen:

```
<goto nextitem="Lehrveranstaltungen" />
```

Um Elemente die einen automatischen Reprompt veranlassen verlassen zu können, muss ebenfalls ein `<goto>` verwendet werden.

3. Ausführen eines anderen Dialogs: Der Wechsel in einen anderen Dialog erfolgt auch per Sprunganweisung. Im `<goto>` Element wird das Attribut *next* verwendet, wo eine URI angegeben wird. Soll im neuen Dokument ein bestimmter Dialog angesprochen werden, so muss dessen Name mit einem vorangestellten “#” angegeben werden.
4. Beenden der Session: Die Session beendet sich automatisch, wenn nach dem letzten Dialog kein Folgedialog mehr angegeben wird. Aktiv beendet wird eine Session mit dem `<disconnect>` Element. Es beendet die Verbindung mit dem Anrufer, nicht aber die VXML Applikation. Es wird ein *connection.disconnect.hangup* Event ausgelöst. Mit dem `<exit>` Element dagegen wird die VXML Applikation, nicht aber das Telefonat beendet.

### 5.2 Multi-Dokument-Applikationen

Zur Erstellung einer VXML Anwendung die aus mehreren VXML Dokumenten besteht, muss eines der Dokumente als “Root-Dokument” (application root document) definiert werden. Die restlichen Dokumente bezeichnet man dann als

---

<sup>6</sup>Ausführlich zum Formularinterpretationsalgorithmus vgl. [2] S. 271ff.

“Blatt-Dokument” (application leaf documents). In jedem Blatt Dokument wird im `<vxml>` Tag ein Attribut *application* angegeben, das als Wert den Pfadnamen des Root-Dokuments erhält. Wird ein Blatt-Dokument aktiv, so wird zunächst dessen Root-Dokument geladen, so dass etwaige Grammatiken oder globale Variablen vor Ausführung des Blatt-Dokuments aktiviert werden. Gleichzeitig ist immer höchstens ein Blatt-Dokument aktiv. Es bestehen damit folgende Transitionsmöglichkeiten:

- Root nach Blatt: Der vorhandene Applikatinskontext des Root-Dokuments wie Variablen, Eventhandler und Grammatiken bleibt unverändert. Zusätzlich wird das Blatt-Dokument geladen.
- Blatt nach Blatt innerhalb derselben Applikation: Das heißt beide Blatt-Dokumente haben dasselbe Root-Dokument. Wiederum bleibt der Applikatinskontext unverändert. Das neue Blatt-Dokument wird geladen. Der Dokumentenkontext des vorhergehenden Blatt-Dokuments wird zerstört. D.h. alle Variablenwerte werden gelöscht, geladene Grammatiken sind nicht mehr aktiv.
- Blatt nach Root innerhalb derselben Applikation: Der Applikationskontext bleibt erhalten. Sollen Daten aus dem Dokumentenontext des Blattes weiter genutzt werden, so müssen diese in den Applikationskontext kopiert werden.
- Applikation nach Applikation: Wird von einem Blatt ein neues Blatt mit einem anderen Wert im *application* Attribut angesprochen oder von einem Blatt-/Root-Dokument eine andere Applikation aufgerufen, so wird der alte Applikationskontext durch einen neuen ersetzt. Alle vorher aktiven Variablen, Grammatiken und Event-Handler werden der Garbage Collection zugeführt.

## 6 Variablen in VoiceXML

Ein VoiceXML Interpreter hat nach der Spezifikation des W3C eine aktive ECMA-Skript-Engine. Deshalb wurden Variablen in VXML so konzipiert, dass einfach der Zugriff auf ECMA-Skript-Variablen ermöglicht wurde. Damit sind auch alle Auswertungsvorgänge nach ECMA verlagert. Folgende Elemente sind im Variablenkonzept realisiert:

- **Deklaration `<var>`:** Das `<var>` Element besteht nur aus einem Tag, der am Ende mit `/` abgeschlossen wird. Attribute sind *name* und *expr* (optional). Das Attribut *name* muss ein gültiger ECMA-Skript Bezeichner sein und *expr* muss ein gültiger ECMA-Skript Ausdruck sein. Die `<var>` Elemente werden zur Laufzeit als ECMA-Skript Variable im aktuellen Scope erzeugt. Ist das optionale *expr* Attribut angegeben, so wird der Ausdruck durch die ECMA-Skript Engine ausgeführt und das Ergebnis der Variable als Wert zugewiesen. Sonst wird die Variable mit “undefined” belegt.
- **Wertzuweisung `<assign>`:** Einer bereits deklarierten Variablen kann mit dem `<assign>` Element ein Wert zugewiesen werden. Es besitzt die glei-

chen Attribute wie das `<var>` Element, allerdings ist das *expr* Attribut jetzt nicht mehr optional. Die Syntax ist also:

```
<assign name="ECMA-Skript-Var" expr="ECMA-Script-Expr" />
```

Soll einer Variablen der Inhalt einer anderen Variablen zugewiesen werden, so wird für ECMA-Skript-Expr einfach der Name der Quellvariablen eingesetzt (=gültiger ECMA-Ausdruck).

- **Wertzugriff <value>**: Mit `<value expr="ECMA-Script-Expr" />` wird ein String zurückgegeben, der die Auswertung des ECMA-Skript Ausdrucks enthält. Wiederum ist als Wert von *expr* ein Variablenname zulässig.
- **Skopus**: Die Sichtbarkeit von Variablen folgt in VoiceXML der Dokumentenhierarchie. Eine Variable die in einem höheren Kontext deklariert wird, ist in einem hierarchisch tiefer liegenden Scope sichtbar. Im Interpreterkontext gibt es einige implementationsabhängigen Variablen auf die nur Lesezugriff besteht. Dem Konzept globaler Variabler vergleichbar sind solche Variablen die im "root-Dokument" der Anwendung deklariert werden. Ihr Kontext wird Application-Kontext genannt. Dann folgen in der Hierarchie die Variablen des Dokumenten-Kontextes. Sie sind Kindelemente des jeweiligen `<vxml>` Dokuments. Variablen des Dialog-Kontextes werden in den Dialogelementen `<form>` bzw. `<menu>` deklariert. Einem sogenannten "Anonymus-Kontext" gehören Variablen an die in den einzelnen Form-Items wie `<field>`, `<catch>` oder `<block>` definiert werden. In den höheren Kontexten werden die impliziten Variablen *session*, *application*, *document* und *dialog* angelegt, durch die in den unteren Kontexten bei Verschattung mittels Präfix-Punkt-Notation auf Variable der höheren Kontexte zugegriffen werden kann.

Eine Besonderheit ist, dass alles innerhalb der Anführungszeichen für VXML ein String Literal ist. Damit etwas auch als String-Typ ausgewertet wird müssen zusätzlich einfache Anführungszeichen verwendet werden.

```
<var name="begruessung" expr="'Buongiorno'" />.
```

Beispiel zu Variablen in VXML:

```
<?xml version="1.0" encoding="ISO-6651-1">
<vxml version="2.0">
  <var name="Morgenzeitung" expr="'Süddeutsche'"/>
  <prompt> Sie lesen manchmal <value expr="Morgenzeitung"/></prompt>
  <assign name="Morgenzeitung" expr="'Herald Tribune'"/>
  <prompt> Und manchmal lesen sie <value expr="Morgenzeitung"/></prompt>
</vxml>
```

Beispiel zu Verschattung in VXML:

```
<?xml version="1.0" encoding="ISO-6651-1">
```

```

<vxml version="2.0">
  <var name="karte" expr="'Ober'"/>
  <form id="kartenspiel">
    <var name="karte" expr="'Unter'"/>
    <prompt> Der <value expr="document.karte"/> sticht den
      <value expr="karte" /> .</prompt>
  </form>
<var name="karte" expr="'Unter'"
</vxml>

```

## 7 Ausgaben in VXML: Das <prompt> Element

Das <prompt> Element kontrolliert die Ausgabe synthetisierter bzw. vorab aufgenommener Sprache. Von den möglichen Attributen werden wir die nachfolgenden verwenden:

- **cond:** Der EMCA Ausdruck muss auf true ausgewertet werden, damit der FIA das <prompt> berücksichtigt.
- **count:** Es wird eine Schranke festgelegt, wie oft der FIA das Formular Item aufgerufen haben muss, damit dieses <prompt> berücksichtigt wird. Das *count* Attribut wird zum sogenannten “tapered prompting” verwendet. Ein Aufforderungstext wird nocheinmal in einer fokussierteren Form wiederholt.
- **timeout:** Regelung der Zeitdauer, wie lange gewartet wird, bis ein “no input Event” ausgelöst wird. Es können Sekunden “s” bzw Millisekunden “ms” angegeben werden, also etwa *timeout* = “5s” oder *timeout* = “300ms”.

### Beispiel zum *count* Attribut

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<vxml version="2.0">
<!--count Attribut -->
<form>
  <field>
    <grammar root="main">
      <rule id="main">
        <one-of>
          <item> Mathematik      </item>
          <item> Programmierung </item>
          <item> Linguistik     </item>
        </one-of>
      </rule>
    </grammar>

    <prompt count ="1"> Welches Teilgebiet der Computerlinguistik
      interessiert Sie?

```

```

</prompt>
<prompt count ="2"> Wählen Sie Mathematik, Programmierung oder Linguistik.
</prompt>
<prompt count ="3"> Sprechen Sie deutlicher.
                        Mathematik, Programmierung oder Linguistik?
</prompt>
<nomatch> Ich habe sie leider nicht verstanden.
          <reprompt/>
</nomatch>
</field>
</form>
</vxml>

```

### Beispiel zum *cond* Attribut

```

<?xml version="1.0" encoding="ISO-8859-1" ?>
<vxml version="2.0">
<!--cond Attribut -->
<form>
  <block>
    <var name="zufall" expr="Math.random()" />
    <prompt cond="zufall <= .33">
      Guten Morgen
    </prompt>
    <prompt cond="zufall > .33 && zufall <= .66">
      Mahlzeit
    </prompt>
    <prompt cond="zufall > .66">
      Guten Abend
    </prompt>
  </block>

  <field>
    <grammar root="main">
      <rule id="main">
        <one-of>
          <item> Mathematik      </item>
          <item> Programmierung </item>
          <item> Linguistik     </item>
        </one-of>
      </rule>
    </grammar>

    <prompt count ="1"> Welches Teilgebiet der Computerlinguistik
                        interessiert Sie?
    </prompt>
    <nomatch> Ich habe sie leider nicht verstanden.
              <reprompt/>
    </nomatch>

```

```
</field>
</form>
</vxml>
```

Übung: Optimitalk unterstützt das *cond* Attribut bei `<prompt>` nicht. Formulieren Sie die `<prompt>` Elemente des Begrüßungsblocks neu, indem Sie Sie jeweils in eine `<if>`-Bedingung einbetten. Verwenden Sie statt “`<$`” , “`‘$’`” und “`&`” die entsprechenden XML-Entities. Formulieren sie das Skript alternativ mit einer `<if>`, `<elseif>`, `<else>` Konstruktion.

### Auswahl der `<prompt>` Elemente im FIA-Algorithmus in der collect Phase

1. Als “prompt-Queue” wird eine Liste aller prompts in der Reihenfolge ihres Auftretens im VoiceXML Dokument angelegt.
2. Die Ausdrücke im *cond* Attribut werden ausgeführt und diejenigen prompts die eine “false-condition” haben, werden aus der Queue gestrichen.
3. Ermittlung des korrekten *count*: Der höchste *count* Wert der `<prompt>` Elemente in der Queue deren Wert kleiner gleich dem des aktuellen Wiederholungszählers ist.
4. Entfernung aller `<prompt>` Elemente aus der Liste, die nicht den korrekten *count* haben d.h. deren *count* nicht den Anforderungen des ermittelten korrekten count genügt.

Nach Erstellung dieser Queue werden ihre Elemente in der Reihenfolge ihres Auftretens im VXML Dokument ausgegeben.

Fragen zum *count* Beispiel: können bei `count=2` mehrere `<prompts>` abgespielt werden? Was passiert, wenn der dritte `count` auf 4 gesetzt wird. Was passiert wenn man das `<reprompt>` Element entfernt?

## 7.1 Barge-in

Mit dem *barge – in* Attribut des `<prompt>` Elements wird festgelegt, ob der Benutzer eine Systemäußerung mit eigenen Äußerungen unterbrechen darf. Wenn das Barge-in innerhalb einer Sequenz von mehreren Systemäußerungen erfolgt, so werden alle weiteren `prompt` Elemente nicht mehr abgespielt. Ist das *barge – in* Attribut auf “false” gesetzt, so wird der Benutzerinput während der Systemäußerung nicht gepuffert, sondern geht verloren. Bezüglich des *bargeintype* Attributes werden “speech” und “hotword” unterschieden. Dabei stoppt “speech” sofort den Systemoutput, während “hotword” erst nach einem match mit einer aktiven Grammatik ein barge-in auslöst.

## 8 Events in VoiceXML

Events werden von der Plattform geworfen wenn ein bestimmtes Benutzerverhalten vorliegt (Benutzer antwortet nicht, nicht in der vorgesehenen Weise, fordert Hilfe an etc.), wenn der VoiceXML Interpreter einen semantischen Fehler

in der Applikation findet oder wenn er auf ein `<throw>` - Element trifft. Für jedes Element in dem ein Event vorkommen kann existiert die folgende Liste von "catch" Elementen: `<catch>`, `<error>`, `<help>`, `<noinput>`, `<nomatch>`. Soweit diese catch Elemente in einem Element nicht vorhanden sind werden sie von den Eltern-Elementen geerbt. Events werden durch das `<throw>` Element geworfen, wobei der Name des Events im *event* Attribut angegeben wird. Soll das Programm auf einen Event reagieren, so muss dieser mit dem `<catch>` Element gefangen werden. Dieses Element ist ein sogenannter Event-Handler und enthält ausführbaren Code.

## Beispiel zum Event Handling

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<vxml version="2.0">
<!--throw und catch -->
<form id="zugangskontrolle">
  <field name="user_id" type="digits">
    <prompt>Geben Sie die Benutzerkennung ein </prompt>
  </field>
  <field name="passwort">
    <prompt>Geben Sie ihr Passwort ein </prompt>
    <grammar root="root">
      <rule id="root" scope="public">
        markusplatz
      </rule>
    </grammar>
    <help> Es ist der Name eines bekannten Platzes </help>
    <catch event="nomatch noinput" count="3">
      <prompt> Sicherheitsverletzung!</prompt>
      <submit next="http://www.example.com/apprehend_felon.vxml"
        namelist="user_id" />
    </catch>
  </field>
</form>
</vxml>
```

Das hier verwendete `<submit>` Element ruft mit HTTP-GET-POST ein anderes Dokument, meistens ein Programm, auf. Solche Programme sind als script auf einem Webserver organisiert. Innerhalb des `<submit>` Elements können über das Attribut *namelist* die Variablen des aktuellen Dokuments an das Programm übergeben werden.

### 8.1 Deklaration eigener Eventhandler

Ein mit `<catch>` deklarierter Eventhandler kann im Application-, Dokumenten-, Dialog- oder Formular-Item Skopus liegen. Ein Event-Handler arbeitet nach dem "Beobachterprinzip" innerhalb seines Skopus. Sobald innerhalb des Skopus passende Ereignisse geworfen werden, wird der Code innerhalb des zuständigen `<catch>` Elements ausgeführt. Die internen Variablen *\_event* und *\_message* enthalten dabei den Namen des Events der geworfen wurde und eine möglicherweise

mitgegebene Nachricht.

```
<catch event="ereignis.eins ereignis.zwei">
<if cond="_event == 'ereignis.eins'">
  <value expr="_message">
    <audio src="./AUDIO/eins.wav"/>
  </value>
</if>
<else/>
  <value expr="_message">
    <audio src="./AUDIO/zwei.wav"/>
  </value>
</if>
</catch>
```

Ein Event-Handler kann mehrere Events verarbeiten. Dazu wird die Liste der Events durch blank getrennt als Wert des *event* Attributes angegeben. Das Gegenüber zu diesem Attribut ist das *event* Attribut im `<throw>` Element.

```
<if cond="wahl == 1">
  <throw event="ereignis.eins"
    message="das erste Ereignis"/>
</if>
<else/>
  <throw event="ereignis.zwei"
    message="das zweite Ereignis"/>
</if>
```

Attribute des Event-Handlers `<catch>` sind:

- *event*: Der oder die Events die behandelt werden sollen. Die verschiedenen Events werden durch eine mit blanks getrennte Liste angegeben. Im Fall mehrerer Events wird für jeden Event ein eigenes *count* Attribut geführt.
- *count*: In diesem Attribut wird die Auftretenszahl des Events gespeichert. Jedes Formular, Menü und jedes Form-Item führt einen Zähler für jeden aufgetretenen Event während es besucht wird. Das `<count>` Element dient dazu mehrfache Auftreten desselben Events unterschiedlich zu behandeln.
- *cond*: Für dieses Attribut kann eine Bedingung angegeben werden, die zu true auswerten muss damit der Event behandelt wird.

Attribute des `<throw>` Elements sind:

- *event*: Der Name des ausgelösten Events
- *eventexpr*: Ein ECMA Ausdruck der zum Namen des ausgelösten Events ausgewertet wird
- *message*: Ein Message String der zusätzlichen Kontext zum geworfenen Event an den Event-Handler übergibt. Für vordefinierte Events die von der Plattform ausgelöst werden ist die Nachricht plattformabhängig
- *messageexpr*: Ein ECMA Ausdruck, der zum Message String ausgewertet wird

## 8.2 Built In Events mit Kurz-Notationen

Die catch-Elemente `<error>`, `<help>`, `<noinput>`, `<nomatch>` sind Kurznotationen für die ausführlichen Schreibweisen wie etwa: `<catch event="nomatch">`.

## 8.3 Auswahlalgorithmus für die Event-Behandlung

Wenn die Anwendung einen Event wirft, wird der Skopus in dem der Event aufgetreten ist untersucht um das passendste `<catch>`-Element zu finden, das den Event behandeln kann. Dabei wird gemäß der W3C VoiceXML Spezifikation wie folgt vorgegangen:

1. Es wird eine Liste aus allen catch-Deklarationen gebildet die im aktuellen Skopus und dem umgebenden Skopus liegen (Formular-Item, Formular, Dokument, Root-Dokument der Applikation und Interpreter Kontext). Die Liste wird nach Skopus beginnend mit dem aktuellen Skopus und innerhalb jedes Skopus nach Dokumentenordnung geordnet.
2. Aus der Liste werden alle catch-Deklarationen entfernt, deren *event* Attribut nicht dem geworfenen Event entspricht oder deren *cond* Attribut zu false auswertet.
3. Finde den korrekten *count* also den höchsten *count* unter den Elementen der Liste der kleiner oder gleich dem aktuellen *count* Wert ist.
4. Wähle das erste Element aus der Liste mit dem korrekten *count*

# 9 Grammatiken in VoiceXML

Die zulässigen Eingaben für Input Items werden mithilfe von Grammatiken spezifiziert. Voice XML selbst legt sich auch in seiner 2.0 Spezifikation nicht auf ein bestimmtes Grammatikformat fest, verlangt allerdings von Applikationen dass sie das Speech Recognition Grammar Format (SRGF) des W3C implementieren. Bekannte Formate neben SRGF sind das Java Speech Grammar Format (JSGF) von Sun Microsystems, die Grammar Specification Language (GSL) von Nuace.<sup>7</sup> Das SRGF ist in zwei Notationen definiert, der älteren ABNF-Notation und im XML-Format. Wir beschränken uns im Folgenden auf das W3C XML-Format wie es in [6] spezifiziert wird.

## 9.1 Allgemeines zu VoiceXML Grammatiken

Grammatiken werden mit Hilfe des `<grammar>` Elements angegeben. Bei der Erkennung von Benutzeräußerungen und deren Zuordnung zu einem Input-Item ist stets eine Grammatik aktiv. Diese kann den Skopus "Item", "Formular" oder "Dokument" besitzen. Inline Grammatiken werden direkt unter dem `<grammar>` Element angegeben. Externe Grammatiken werden über eine URI-Angabe im *scr* Attribut festgelegt. Für verschiedene Standardaufgaben gibt es sogenannte "Built-In-Grammatiken. Beispiele sind "boolean", "digits", "time" und "date". Diese können als Attributswert von *type* im jeweiligen Input-Item angegeben

<sup>7</sup>Die von uns verwendete Plattform OptimTalk implementiert SRGS, JSGF und über VoiceXML hinausgehend das Microsoftformat.

werden. Der Skopus der Grammatik bestimmt sich für Form-Level Grammatiken nach dem Wert des *scope* Attributs (s.u.). Für Input Item Grammatiken ist der Skopus nur das betreffende Item. Ein Skopus kann nicht angegeben werden. Link Grammatiken haben also Skopus das Element das das `<link>` Element enthält. Ist es im root-Dokument einer VXML-Anwendung definiert, so ist seine Grammatik auch in den Blatt-Dokumenten definiert. Menü-Grammatiken sind per default im Dialog aktiv, also für das jeweilige `<menu>` Element.

## 9.2 Die Attribute des Elements `<grammar>`

Die nachfolgenden Attribute können innerhalb des `grammar` Elements definiert werden.

- `src`: URI, Pfad zu einem externen Grammatikfile
- `scope`: Wird hier “document” angegeben, so ist die Grammatik in allen Dialogen des Dokuments aktiv. Wird der Attributwert auf “dialog” gesetzt, so ist die Grammatik nur innerhalb des gegebenen Formulars aktiv. Default Wert ist “dialog”.
- `type`: Hier kann ein Grammatiktyp angegeben werden. Ohne Angabe bestimmt der Interpreter den Typ aus der codierten Grammatik selbst.
- `caching`: Wird der Wert “safe” angegeben, so holt der Browser jedes angeforderte Objekt neu. Ist “fast” angegeben, so können caches verwendet werden.
- `fetchtimeout`: Bestimmt die Wartezeit nach der ein `error.badfetch` Event ausgelöst wird.
- `xml:lang`: Sprache der Grammatik, z. B. “DE” für Deutsch.
- `mode`: Zulässige Werte für den Eingabemodus sind “voice” oder “dtmf”.
- `root`: Legt die Startregel der Grammatik fest. Für Inline Grammatiken vom Typ des W3C XML Grammatikformat MIME-Type “application/grammar+xml” ist zwingend ein root-Element anzugeben.
- `version`: Default ist hier “1.0”
- `weight`: Spezifiziert das “Gewicht” der Grammatik. Also den Wert für die Wahrscheinlichkeit des Vorkommens eines grammatikalischen Strings. Vorgabewert “1.0”, erhöhte Wahrscheinlichkeit wäre etwa “4.0”.

## 9.3 Das Speech Recognition Grammar Format (SRGF)

Definiert wird das Grammatikformat des W3C in der SRGF-Spezifikation [6]. Die Ausdrucksstärke des dort definierten Grammatikformats entspricht derjenigen kontextfreier Grammatiken. Als Formate stehen Augmented Backus Naur Form (ABNF) und ein XML Grammatikformat zur Verfügung. Input der Grammatiken ist ein Audio-Strom, der gesprochene Sprache enthalten kann die dann darauf untersucht wird ob sie auf die entsprechende Grammatik matcht. Der Output der Grammatiken ist eine Beschreibung der Resultate mit Details

über die gesprochenen Inhalte. Beschränkungen des SRGRF sind u. a. fehlende Konstrukte zu folgenden Features: Ermittlung von sprecher-adaptiven Daten, Spracherkenner-Konfigurationsdaten zu timeouts, Erkennungsgrenzen(thresholds), Größe von Suchergebnissen oder N-best Resultatssammlungen, Laden von Lexika, sowie Sprechererkennung. Das Format muss gemäß der VoiceXML 2.0 Spezifikation [5] durch alle VoiceXML Plattformen implementiert werden.

### 9.3.1 Elemente von SRGF Grammatiken

- **Tokens:** Kleinste Einheit, die der Spracherkenner als Output übergibt. Üblicherweise sind das Wörter die dann den terminalen Symbolen einer kontextfreien Grammatik entsprechen. Jede Form von unmarkiertem Text in einer Regel-Definition wird als Tokenfolge, getrennt durch Whitespaces betrachtet. Mittels des Elementes `<token>` können Multi-Token definiert werden: `<token> FC St. Pauli </token>`. Das `<token>` Element hat ein optionales *lang* Attribut, das dazu benutzt werden kann, eine Sprache und damit die zu erwartende Aussprachevariante festzulegen.
- **Items:** Definieren die Alternativen des Grammatikelements `<one-of>` bzw. Abschnitte mit optionalem Inhalt oder Wiederholungsmöglichkeit:

```
<grammar>
  <one-of>
    <item><token>FC Bayern München</token></item>
    <item><token>Werder Bremen</token></item>
  </one-of>
</grammar>
```

Als Attribut können `<item>` Elemente *repeat* mitführen. Dort kann mit “0-1” ein Optionales Item oder etwa mit “1-3” eine bis zu dreimalige Wiederholung festgelegt werden. Mit “2-” wird spezifiziert, dass der Ausdruck mindestens zweimal wiederholt wird.

- **Regeln:** Mit dem `<rule>` Element wird eine Grammatikregel definiert. Zum Verweis auf die Regel innerhalb der Grammatik kann sie durch das *id* Attribut mit einem Namen ausgezeichnet werden. Über das Referenzierungselement `<ruleref>` können innerhalb einer Regel andere Regeln aufgerufen werden.
- **Sonderregeln:** Mit dem Attribut *special* des `<ruleref>` Elements können verschiedene Sonderregeln des SRGF XML-Formats referenziert werden.
  - `<ruleref special="#NULL"/>` Matcht automatisch, auch wenn der Benutzer nichts äußert.
  - `<ruleref special="#VOID"/>` Stimmt nie überein, egal was der Benutzer äußert. Diese Sonderregel wird zu Entwicklungszwecken eingesetzt, wenn man bestimmte Grammatikteile ausblenden will.
  - `<ruleref special="#GARBAGE"/>` Die “Garbage-Regel” matcht die Benutzereingabe bis dahin, wo die nächste Regel oder das nächste Token matcht. Diese Sonderregel wird zum Keyword-Spotting verwendet.

```

<grammar>
<example>I want to fly home</example>
<ruleref special="GARBAGE"/>
fly
<ruleref special="GARBAGE"/>
home
</grammar>

```

### 9.3.2 Die Semantik von SRGF Grammatiken

Die Rückgabe einer Grammatik an die Applikation soll oftmals nicht die bloße Benutzeräußerung sein, sondern ein semantisches Resultat. In SRGF Grammatiken im XML Format werden solche sogenannten "semantic interpretation tags" im `<tag>` Element oder im `tag` Attribut der Grammatikelemente `<one-of>`, `<token>`, `<ruleref>` oder `<item>` gespeichert.

```

<?xml version="1.0" encoding="ISO-8859-1">
<!DOCTYPE grammar PUBLIC "-//W3C//DTD GRAMMAR 1.0//DE"
"http://www.w3.org/TR/speech-grammar/grammar.dtd">

<grammar xmlns="http://www.w3.org/2001/06/grammar"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:SchemaLocation="http://www.w3.org/2001/06/grammar
http://www.w3.org/TR/speech-grammar/grammar.xsd"
xml:lang="DE" version="1.0" >
<rule id="ja">
<one-of tag="ja">
<item>ja</item>
<item lang="en-US">yes</item>
<item lang="fr-CA">oui</item>
</one-of>
</rule>
<rule id="nein">
<one-of>
<item>nein</item>
<item tag="nein" lang="en-US">no</item>
<item tag="nein" lang="fr-CA">no</item>
</one-of>
</rule>
<rule id="root" scope="public">
<one-of>
<item>ruleref uri="#nein"</item>
<item>ruleref uri="#ja"</item>
</one-of>
</rule>
</grammar>

```

Neben dem `tag` Attribut kommt das `<tag>` Element zum Einsatz um etwa im Fall von mixed-initiative mehrere Slots eines Dialogs gleichzeitig zu belegen oder um per `"."`-Notation in einer Feldvariable mehrere "Zellen" zu belegen.

## 10 Applikationen mit gemischter Initiative

Gemischte Initiative bedeutet eine höhere Ebene der Natürlichkeit des Dialogs zu modellieren. Anders als im “geführten Dialog”, wo das System ständig die Initiative behält kann hier der Benutzer durch eine Äußerung direkt in den Dialogablauf eingreifen. Unterschieden wird gemischte Initiative innerhalb eines Dialogs sowie die dialogübergreifende gemischte Initiative. Beispiel für eine Benutzeräußerung die gemischte Initiative verlangt ist etwa die Folgende: “Ich bin im 5. Semester Hauptfach Computerlinguistik und benötige Informationen zu den Hauptseminarsveranstaltungen bei Prof. Schulz.” Hier werden zugleich mehrere Informationen mitgeteilt und das System muß verschiedene “slots”, die es sonst per Einzelabfrage ermittelt füllt. Nachdem vom System so ein eindeutiger Dialogzustand festgestellt ist, wird eine sinnvolle Systemreaktion ermittelt.

### 10.1 Mixed Initiative Dialoge

Hierzu wird ein `<form>` Element mit einem `<initial>` Element eröffnet. Danach folgen mehrere `<field>` Elemente, die von einer gemeinsamen Grammatik gesteuert werden. Ziel der Grammatik, die dann als “Multi-Slot-Grammatik” bezeichnet wird, ist es mit einer Benutzeräußerung gleichzeitig mehrere `<field>` Elemente (slots) zu füllen. Es sollen also Benutzeräußerungen mit mehreren relevanten Informationen verarbeitet werden wie z. B.: System: “Welche Bahnreise wünschen Sie zu buchen”; Benutzer: “Von München nach Venedig mit dem Intercity”. In der Grammatik werden für Dialoge mit gemischter Initiative Regeln eingeführt, die als Rückgabe über das `<tag>` element bestimmte Slots im Formular zu dem die Grammatik gehört füllen. Ein Beispiel für eine solche Grammatikregel:

```
<item>
<ruleref uri="#city"/>
<tag> $.to = $city; </tag>
</item>
```

Das zugehörige `<field>` Item innerhalb des VoiceXML Dokuments kann dann wie folgt aussehen: `<field name="to_city" slot="to">`. Wird nun vom Benutzer eine Stadt die in der “city” Grammatikregel definiert ist im “to” Kontext eingegeben so wird der “to” slot im “from-to” Formular gefüllt. Im `<filled>` Item auf Formularebene wird dann je nachdem wieviele und welche slots bereits gefüllt sind vom Benutzer weitere Information abgefragt.

### 10.2 Mixed Initiative Gespräche

In Applikationen die “Mixed Initiative Gespräche” enthalten, kann der Benutzer selbst das Laden anderer Formulare oder anderer Dokumente herbeiführen. Zu beachten ist bei solchen Modellierungen, dass bei einem Wechsel des Dialogs alle Daten des Dialogs `<var>` Elemente und Form-Item-Variablen verloren gehen, wenn sie nicht in globale Variablen aus Applikationsebene gespeichert werden. Ein Benutzererzeugter Dialogübergang kann durch zwei Wege erreicht werden:

- Das `<link>` Element: Diesem Element wird über `<grammar>` eine Grammatik zugeordnet, deren Reichweite vom Standort des Links innerhalb

des VoiceXML Dokuments abhängt. Trifft eine Benutzeräußerung auf eine aktive Link-Grammatik zu, so wird das `<link>` Element aktiviert. Es kann dann entweder eine Transition zu einem Dialog oder einem anderen VXML-Dokument durchgeführt werden oder ein Event geworfen werden.

- Grammatiken mit Dokument-Skopus: Um Übergänge auf Dialog- bzw Dokumentebene durch den Benutzer mithilfe des `<grammar>` Elements zu modellieren muss das *scope* Attribut der Grammatik auf den Wert "document" gesetzt werden. Ein Nachteil ist allerdings, dass für diese Variante alle Dialoge im selben Dokument kodiert sein müssen. Um diese Voraussetzung zu umgehen kann man mit `<link>` Elementen arbeiten, die im Root-Dokument der Anwendung Dokumentenskopus haben und damit applikationsweit sichtbar sind. Diese werden dann benutzt, um innerhalb der Anwendung verschiedene Multi-Slot-Grammatiken über "Hot-Words" zu aktivieren.<sup>8</sup>

## 11 Externe Objekte

Durch das `<object>` Element ist es in VoiceXML möglich externe Anwendungen aufzurufen. Dieses Element ist in der VXML 2.0 Spezifikation offen gelassen. Verpflichtend ist lediglich, dass im Fall der Nicht-Implementierung eine vordefinierte Exception geworfen wird.

---

<sup>8</sup>Vgl. [2] S. 321f.

## 12 Architektur des Übungssystems OptimTalk

Die VoiceXML Plattform OptimTalk wird in einer Kooperation zwischen dem Laboratory of Speech and Dialogue (<http://www.fi.muni.cz/lsd/>) der Universität Brno und der Norut IT (<http://www.itek.norut.no/>) einem Forschungsinstitut der Universität Tromso entwickelt.

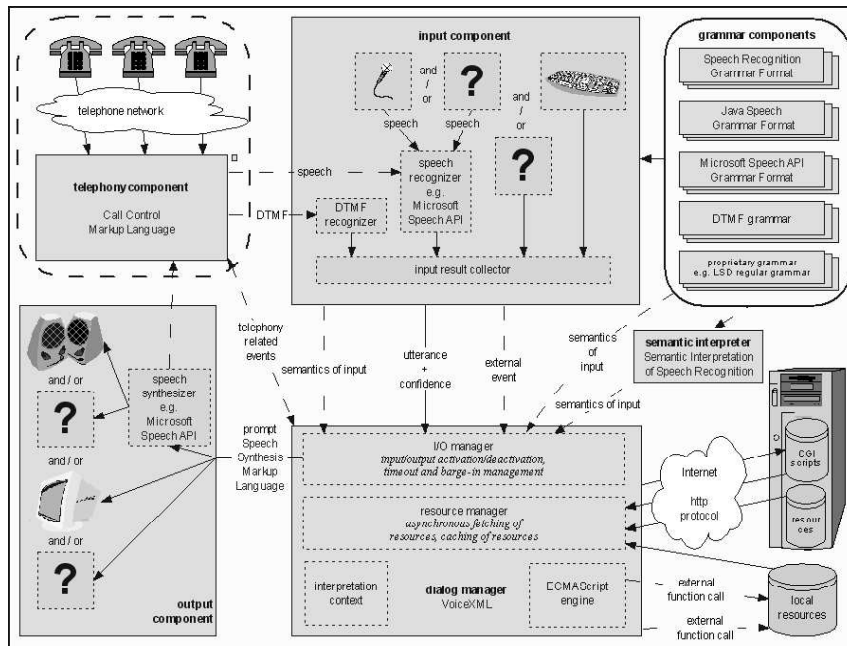


Abbildung 1: Übersicht über die VoiceXML Plattform Optimtalk vgl [4] S. 2

Die zentrale Komponente der Plattform ist ein Dialogmanager, der als VoiceXML Interpreter dient und die anderen Komponenten kontrolliert. Die Inputkomponente kann verschiedene Forment von Input wie Telefonieinput, Mikrofoninput oder auch Konsoleninput empfangen und übergibt diesen als Stringobjekt an den Dialogmanager. Ähnlich flexibel ist die Outputkomponente, die vom Dialogmanager vorbereitete “prompts” über eine TTS Engine auf die Telefonschnittstelle bzw. einen Lautsprecher oder ohne Verwendung von Sprachsynthese lediglich auf die Konsole ausgibt. Die Implementierung des `<object>` Elements erlaubt es dem Anwender über externe C++ Funktionen beliebige systemati-

sche Erweiterungen zu erstellen, die über den VXML 2.0 Standard hinausgehen. Über diese Schnittstelle ist es auch möglich multimodale Anwendungen zu erstellen.<sup>9</sup>

## 12.1 Die Übungsumgebung `optimtalk_test`

Im Paket von OptimTalk ist eine Übungsumgebung mitgegeben die über Keyboard-Input und Konsolen-Output die Spracherkennung und die Sprachausgabe simuliert. Der Kern der Anwendung bleibt dabei derselbe. Damit steht eine komplette VoiceXML Plattform als PC-System zur entwicklung von Applikationen zur Verfügung. Der Aufruf unter Windows/Linux erfolgt aus dem Verzeichnis `OptimTalk/bin` mit dem Befehl `“optimtalk_test”` der als Argument ein VoiceXML Wurzeldokument erwartet.

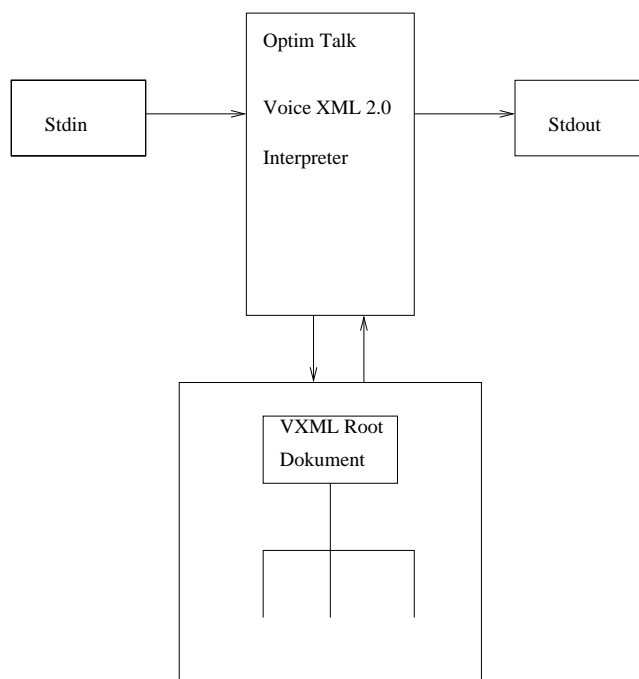


Abbildung 2: Struktur des Übungssystems `optimtalk_test`

## 12.2 DTMF Simulator

Die Input-Schnittstelle der Testumgebung von OptimTalk beinhaltet einen DTMF Simulator. Die Folgenden Eingabetasten können benutzt werden um DTMF-Input zu simulieren:<sup>10</sup>

<sup>9</sup>Vgl. etwa <http://www.inspire-project.org>.

<sup>10</sup>Für die LINUX-Version der verwendeten Software kann es Probleme mit dem DTMF-Simulator geben. In diesem Fall muss auf einer originären Shell ohne X oder KDE Unterstützung gearbeitet werden

Eingabetaste	DTMF
F1 - F9	1 - 9
F10	0
F11	*
F12	#

Tabelle 1: DTMF-Simulator in OptimTalk

## 13 Übungen

### 13.1 Programm: Verabschiedung/Begrüßung

Das Programm soll den Benutzer verabschieden und begrüßen. Es handelt sich also um ein ausgebautes “Hello World Programm”. Es sollen zwei Formulare für Begrüßung und Verabschiedung programmiert werden. Zur Navigation kann ein `<goto>` verwendet werden. Als Alternative zum reinen prompt soll eine Variable namens “hallo” mit einem Begrüßungsstring belegt werden, die dann im Grußformular `<form>` ausgegeben werden soll.

Lernziele: Formularelement, Blockelement, Variablendeklaration, Variablenverwendung, Nachfolgedialog

### 13.2 Programm: Navigation im VoiceXML Dokument

Ziel dieses Programms ist es die Navigationsmöglichkeiten in VXML-Applikationen zu implementieren. Es sollen zwei VoiceXML Dokumente angelegt werden: beispiel2a.vxml und beispiel2b.vxml. Das erste soll drei Formulare haben: “start”, “mitte” und “end”. Diese sollen an den Benutzer jeweils einen prompt ausgeben, der den Standort im Dokument benennt, z. B.: “Sie sind im Formular start”. Das Formular “start” soll die Blöcke “rot”, “gruen” und “gelb” besitzen. Jeder Block soll einen prompt ausgeben, der meldet, dass er besucht wird. Folgende Abarbeitung der Applikation soll implementiert werden:

- Beginn im Formular start
- Dort soll als erstes `<block>` “rot” besucht werden und “gruen” so belegt werden dass er nicht besucht wird: Mit dem `<assign>` Element das *expr* Attribut der Form-Item-Block-Variablen gruen auf “true” setzen.
- Über die natürliche Abarbeitungsreihenfolge soll dann der Block “gelb” besucht werden. Dort soll die Form Item Block Variable “gruen” auf undefined gesetzt werden: `<clear>` Element
- Aus dem “gruen” block soll dann per `<goto>` so verzweigt werden, dass das Formular “mitte” übersprungen wird und als nächstes das Formular “ende” aufgesucht wird.
- Aus dem Formular “ende” soll in das andere VoiceXML Dokument verzweigt werden.
- Das zweite Formular soll lediglich eine Meldung mit seinem Dateinamen ausgeben und dann die Applikation beenden. Alternativen sind: `<exit>`,

<disconnect> und eine default Beendigung durch den Formular Interpretations Algorithmus (FIA), da kein zu bearbeitendes Item mehr offen ist.

Lernziele: Einfache Navigationsmöglichkeiten, Abarbeitungsreihenfolge.  
Navigationsmöglichkeiten in VoiceXML:

1. innerhalb eines <form> Elements navigieren durch Belegung von Formular Items
2. innerhalb eines Dokuments zu einem anderen Dialog navigieren durch das <goto> Element
3. von einem Dokument zum anderen navigieren durch das <goto> Element

Frage: Wird das Formular “mitte” erreicht? Begründen Sie die Antwort mit der Abarbeitungsreihenfolge des Formularerkennungsalgorithmus.

### 13.3 Programm: Variablenverwendung

In diesem Programm soll zuerst eine Variable (<var> Element) “zahl” (*name* Attribut) mit Wert 1 und eine Variable “wort” mit Wert ‘hallo’ deklariert werden. In einem Formular mit Namen “start” (*id* Attribut des <form> Elements) soll zuerst “zahl” ausgegeben und dann um 10 erhöht werden. Danach soll “wort” ausgegeben werden und schließlich soll “wort” mit dem String “wie gehts?” konkateniert werden.

Lernziele: Variablendeklaration, Variablenverwendung

### 13.4 Programm: Events und einfache Grammatiken

In diesem Programm soll die Alternativfrage “sind Sie im Hauptstudium” innerhalb eines <field> Elements mit dem Namen “auswahl” beantwortet werden. Falsche Eingaben sollen durch die Events <nomatch> und <noinput> abgefangen werden. Richtige Antworten werden in einer Inline Grammatik <grammar> Element festgelegt. Achtung: Im Grammatikelement muss immer eine “root”-Regel der Grammatik festgelegt werden. Hierzu wird im <grammar> Element das Attribut “root” mit dem Namen einer der folgenden Grammatikregeln belegt: <grammar root="main"> und <rule id="main">

Schließlich soll innerhalb des Feld-Elements “auswahl” mit <filled> eine Eingabebehandlung durchgeführt werden. Hierzu ist die <if> <else> Struktur zu benutzen. In der ersten Implementierung des Programms wird innerhalb der Grammatik nur mit den Eingaben “ja” und “nein” gearbeitet. Als Alternative soll dann noch eine Grammatik implementiert werden, die verschiedene Varianten von “ja” und “nein” zulässt und dann eine abstrakte Rückgabe über das *tag* Attribut im Grammatikelement *ruleref* durchführt. Eine Regel kann dabei wie folgt aufgerufen werden: <ruleref uri="#yes" tag="yes">. Referenziert wird also auf eine Grammatikregel <rule id="yes"> die dann über das <oneof> Element verschiedene Eingabealternativen wie “ja”, “natürlich” etc. deklariert. Über das *tag* Attribut das bei <ruleref> deklariert ist, wird der Benutzerinput also “ja” oder “natürlich” in die Rückgabe “yes” verwandelt. Dieser Input befindet sich dann in der Form-Item-Variablen die über den Namen des Form-Items angesprochen werden kann, in unserem Fall also “auswahl”.

Lernziele: Verarbeitung einfacher Events, Verwendung einfacher Grammatiken. Abarbeitungsreihenfolge des Interpretationsalgorithmus. `<filled>` Element + Kontrollstruktur `<if>`.

### 13.5 Programm: Subdialog und externe Grammatik

Das zu erstellende Programm soll mit dem prompt “Willkommen auf der Webseite des CIS” eröffnen und dann einen Subdialog namens “vorwissen” eröffnen. In diesem Subdialog wird abgefragt, was die Abkürzung CIS bedeutet. Im `<filled>` Element des Subdialogs wird weiterverzweigt, je nachdem ob der Nutzer die richtige Antwort gegeben hat. Innerhalb des `<subdialog>` Elementes soll über das `scr` Attribut in ein Formular “test” verzweigt werden, dass in einem Feld Antwort des Benutzerwissens prüft. Als Grammatik dieses Feldes wird eine externe Grammatik “beispiel6.grxml” verwendet. Über ein Item `<return namelist="antwort">` wird die Variable `antwort` an den aufrufenden Subdialog zurückgegeben und kann dort mit “`subdialogName.antwort`” angesprochen werden. Die Grammatik soll so designed werden, dass die Antwort sowohl in Deutsch als auch in English angegeben werden kann. Alternativ soll eine Floskel wie “keine Ahnung”, “weiß ich nicht” etc. zulässig sein. Hierzu kann das SRGF-Joker-Element `<ruleref special="GARBAGE">` verwendet werden. Zum Eventhandling für `<nomatch>` soll eine Behandlung mit Varianten programmiert werden. Hierzu kann das `count` Attribut im Element `<nomatch>` eingesetzt werden. Also z.B. `<nomatch count="2">` etc.

Lernziele: Eventhandling und `<subdialog>` Element, Nutzung einer externen Grammatik.

### 13.6 Programm: Shadow Variablen + Approximative Erkennung

Lernziele: Vorbereitung approximativer Verfahren: mit der Joker-Rule GARBAGE eine nichtmatchende Benutzeräußerung in das `<filled>` Element überführen und dort in einer geschachtelten `if/else` Struktur verarbeiten.

### 13.7 Programm: Links und Optionen

Im zu erstellenden Programm soll eine Veranstaltung aus dem Bereich der formalen Methoden ausgewählt werden (Automatentheorie, Mathe I, Mathe II, statistische Methoden). Zur Auswahl soll in einem Feld mit Namen “Formal” eine Reihe von `<option>` Elementen verwendet werden. Es soll gleichzeitig eine Eingabemöglichkeit durch Sprache und durch DTMF Tasten implementiert werden. Im `<filled>` Element des Feldes “Formal” soll mit einer `<if>`, `elseif`, `else` Konstruktion eine Verzweigung programmiert werden. Zur Dialogsteuerung soll neben den üblichen Events `<noinput>` und `<nomatch>` ein Link-Element implementiert werden. Dessen Inline Grammatik soll ein `<help>` Event auslösen, das wiederum einen Hilfesatz ausgibt. Die Grammatik des `<link>` Elements soll für Wortkombinationen wie “Bitte Hilfe” oder “helfen Sie mir” matchen. Es handelt sich also um ein `mixed-initiative` Element dass im Skopus des Formulars definiert werden muss, so dass der Benutzer jederzeit die Hilfe-Funktion auslösen können soll. Nach demselben Muster soll ein Linkelement deklariert werden, das ein `<exit>` event auslöst.

Lernziele: Verwenden des `<link>` Elements. Weiterverarbeitung der ausgewählten Option in einem `<filled>` Element. Anwenden einer `if/else` Struktur mit mehreren Optionen. Erstellen einer etwas ausführlicheren Hilfe Grammatik.

### 13.8 Programm: Speech Recognition Grammar I

Im zu erstellenden Programm soll es möglich sein, dass der Benutzer in einer Äußerung gleichzeitig mehrere Slots füllt. Das System soll folgende Informationen einholen: Will der Benutzer Informationen zu Hauptfach, Nebenfach oder Aufbaustudiengang und will er diese Informationen für das Sommersemester oder für das Wintersemester. Der Benutzer soll beispielsweise äußern können: "Zum Hauptfach Computerlinguistik im Wintersemester". Die Informationen sollen aus der Grammatik in das aktive Feld "anfrage" übergeben werden. Dies geschieht dadurch, dass Innerhalb der Grammatik ein `<tag>`-Element deklariert wird, das die Slots aus den Regeln auf die Slots der aufrufenden Variable schreibt. Ein Beispiel zu dieser Verwendung des `<tag>` Elements ist etwa: `<tag> $.studientyp = $studientyp; $.semestertyp = $.semestertyp <tag>`, unter der Voraussetzung, dass es innerhalb der Grammatik zwei Regeln gibt, deren *id* Attribut mit "studientyp" bzw. "semestertyp" belegt ist. Innerhalb des `<filled>` Element des Feldes "anfrage" können dann die Einträge dieser `$`-Variablen weiterverarbeitet werden indem man sie über "anfrage.studientyp" bzw. "anfrage.semestertyp" anspricht. Der Dialog soll sich nur sehr allgemein melden, etwa: "Willkommen zur Auswahl des Studiengangs" und erst über eine Hilfsfunktion, vgl. Programm "Links und Optionen" die genaue Anweisung geben. Wiederum soll über einen `<exit>` Event jederzeit abgebrochen werden können.

Lernziele: In einer Äußerung gleichzeitig mehrere Informationen mitteilen. In der Grammatik für die Slots Variablen definieren, die im VXML Dokument dann zugänglich sind über die Form Item `“.’-Notation`.

### 13.9 Programm: Mixed Initiative, Multiple Slotsfüllung

Mittels eines Standardbeispiels zur Zugbuchung An- und Abfahrtsort sollen folgende Dialogelemente realisiert werden: Der Benutzer soll in einer Äußerung sowohl Abfahrtsort als auch Zielort nennen können, also etwa: "von München nach Venedig" er soll aber auch lediglich äußern können: "von München", dann soll das System den Abfahrtsort registrieren und nach dem Zielort fragen. Des weiteren soll der Benutzer den Dialog jederzeit durch eine Benutzeräußerung: "Bitte noch einmal" neu zu starten.

## A Zusammenstellung wichtiger VoiceXML 2.0 Elemente

<assign> Wertzuweisung an Variable  
<audio> Abspielen von aufgenommenen Systemäußerungen  
<block> Ausführbarer Code  
<catch> Eventbehandlung  
<choice> Auswahlelement in <menu>  
<clear> Setzen von Variablen auf undefined  
<disconnect> Beendigung der Telefonverbindung  
<else> Kontrollstruktur  
<enumerate> Aufzählen von Alternative in <menu> oder <option>  
<error> Fehlerevent  
<exit> Beenden der VoiceXML-Sitzung  
<field> Eingabe Item in einem Formularelement  
<filled> Systemreaktion auf ‘korrekte’ Benutzeräußerungen  
<form> Standarddialogelement  
<goto> Sprungelement zur Dialogkontrolle  
<grammar> Grammatikelement  
<if> Kontrollstruktur  
<initial> Anfangselement in einem mixed-initiative Dialog  
<link> Verzweigungselement für mixed-initiative Anwendungen  
<log> Für das Entwicklungsstadium können Debug-Meldungen ausgegeben werden  
<menu> Dialogelement zur Alternativenauswahl  
<meta> Definiert ein Metadaten Item  
<metadata> Definiert Metadaten  
<noinput> Wird bei fehlender Benutzeräußerung ausgelöst  
<nomatch> Wird bei Nichtverstehen der Benutzeräußerung ausgelöst  
<object> Aufruf eines externen Objekts um etwa über eine dynamische C++ Library ein Programm auszuführen  
<option> Alternativauswahl innerhalb des <form> Elements  
<param> Parameter in <object> oder <subdialog>  
<prompt> Ausgabe von Systemäußerungen  
<property> Umgebungseigenschaften der VoiceXML-Plattform  
<record> Aufnehmen von Benutzeräußerungen  
<reprompt> Wiederholung eines Prompts  
<return> Rückgabe von Variablenwerten aus einem Subdialog  
<script> Ermöglicht die Ausführung eines ECMA Skripts  
<subdialog> Eröffnet eine Subdialog  
<submit> Anfrage an einen Dokumentenserver  
<throw> Eventauslösung  
<transfer> Anrufweiterleitung  
<value> Variablenauswertung  
<var> Variablendeklaration  
<vxml> Wurzelelement in jedem VoiceXML Dokument

## Literatur

- [1] Günther Carsten and Klehr Markus. VoiceXML 2.0. *Mitp-Verlag*, Bonn, 2003.
- [2] Maracke Ernst. VoiceXML 2.0. Konzeption, Projektmethodik und Programmierung von Sprachdialogsystemen. *Galileo-Verlag*, Bonn, 2003.
- [3] Cenek Pavel. A Flexible Framework for Evaluation of New Algorithms for Dialogue Systems. *Proceedings of TSD 2002*, Springer Verlag Berlin, p. 437-440, 2002.
- [4] Cenek Pavel. Elvira - a VoiceXML Plattform for Research. *VoiceXML Forum*, [www.voicexmlreview](http://www.voicexmlreview.com), 2003.
- [5] W3C. Voice Extensible Markup Language (VoiceXML) Version 2.0. <http://www.w3.org/TR/voicexml20/>, *W3C Recommendation*, 2004.
- [6] W3C. Speech Recognition Grammar Specification Version 1.0. <http://www.w3.org/TR/speech-grammar/>, *W3C Recommendation*, 2004.
- [7] W3C. Semantic Interpretation for Speech Recognition. <http://www.w3.org/TR/semantic-interpretation/>, *W3C Working Draft*, 2003.