Dialógustervezés VoiceXML-ben
(Dialog planning in VoiceXML)

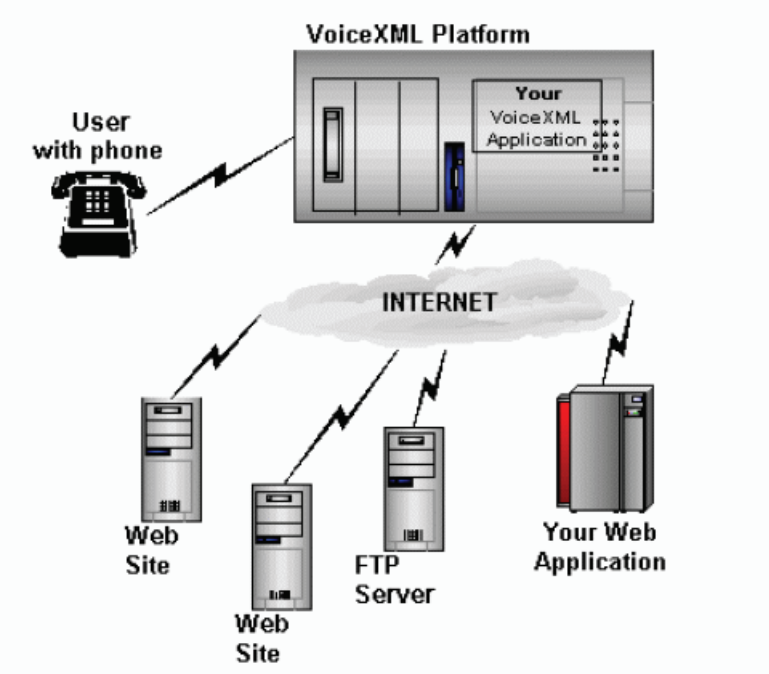
Csapó Tamás Gábor <csapot AT tmit.bme.hu>

2010. március 10.

2. VoiceXML programozási útmutató

A VoiceXML egy XML formátumú nyelv, ember és számítógép közötti interaktív, hanggal vezérelt párbeszédek leírására. Célja a web-alapú interfészek és tartalomszolgáltatás előnyeinek bevitele az interaktív hangvezérelt alkalmazásokba, hogy az utóbbiak készítéséhez ne legyen szükséges speciális programozói szaktudás. Használata elsősorban telefonos környezetben lehetséges, hangos böngésző alkalmazásával [1, 2]. A mérés és a következő útmutató során a VoiceXML 2.0 változatával ismerkedhetünk meg.

2.1 Egy tipikus VoiceXML alkalmazás



6. ábra Tipikus VoiceXML alkalmazás működés közben

Egy tipikus VoiceXML alkalmazásban a felhasználó csatlakozik a rendszerhez, vagyis tárcsázza a megfelelő telefonszámot. A VoiceXML értelmező fogadja a hívást, és elkezdi végrehajtani a VoiceXML programkódot. Az értelmező különböző utasításokat hajthat végre a program hatására:

- prompt üzenetek (beszéd) vagy egyéb hanganyag (zene vagy hangeffektek) lejátszása a felhasználónak,
- a felhasználó által beütött DTMF (Dual-Tone Multi-Frequency) kód fogadása,

- a felhasználó hangbemondásainak fogadása és a beszéd felismerése,
- a felhasználó hangbemondásainak fogadása és tárolása (felismerés nélkül),
- a felhasználó kéréseinek továbbítása egy webszerver felé,
- adatok fogadása webszervertől és azoknak továbbküldése a felhasználó felé.

Ezeken kívül a VoiceXML alkalmazások képesek olyan programozott funkciók megvalósítására, mint például az aritmetikai műveletek, és egyszerű szövegfeldolgozás. Ennek segítségével az alkalmazás ellenőrizni tudja a felhasználói bemenet érvényességét. Ezen kívül a dialógus nem feltétlenül csak egy statikus sorozat lehet, ami mindig ugyanúgy megy végbe minden híváskor. Lehetőség van dinamikus elemek hozzáadására is, például feltételes szerkezetek segítségével.

2.1.1 Hang alapú hozzáférés a web-hez

Egy teljes VoiceXML alkalmazás általában a saját szerverén kívül elérhető erőforrásokat is felhasznál. Előfordulhat, hogy webszerverre is szükség van a felhasználótól érkező információ feldolgozására és válasz küldésére.
A VoiceXML előnye elsősorban ebben fedezhető fel, vagyis hogy a VoiceXML alkalmazás hozzá tud férni az Internethez, egy beszéddel vezérelt böngészőprogramként. Segítségével lehetőség van arra, hogy adatokat küldjünk és fogadjunk webszerverektől. Továbbá, ha a VoiceXML-t egy webes alkalmazás előtétjeként (front-end) használjuk, jelentősen lecsökkenthető a megírandó VoiceXML kód. Ilyenkor az alkalmazás nagy része alapulhat olyan ismert protokollokon, melyekhez már léteznek hatékony fejlesztőeszközök (pl. HTML és CGI).

2.1.2 Alkalmazásfejlesztés

Mivel a VoiceXML alkalmazások nyers szövegből állnak, bármilyen egyszerű szövegszerkesztővel (Windows Notepad, Notepad++, Linux EMACS) létre lehet hozni a fájlokat. Ezeket utána fel kell tölteni egy webszerverre, az egyéb adatokkal (pl. hangfájlok) együtt. Az alkalmazást kipróbálni pedig a megfelelő telefonszám felhívásával lehet.
Néhány egyszerű eszköz segítheti az alkalmazások fejlesztését:

- VoiceXML értelmező és szintaxis ellenőrző,
- dialógus rögzítése a SIP kliensben,
- naplófájl, mely minden beérkező hívásról készül.

2.2 VoiceXML alapok

Először is nézzünk egy VoiceXML példa alkalmazást!

```
1 <?xml version="1.0" encoding="UTF-8"?>
2 <vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
3     <form>
4          <field name="selection">
5               <prompt>
6                    Kérlek válassz: Hírek, mozi vagy sport.
7               </prompt>
8               <grammar root="r" mode="voice"
                     type="application/srgs+xml">
9                    <rule id="r">
10                        <one-of>
```

```
11                              <item>hírek</item>
12                              <item>mozi</item>
13                              <item>sport</item>
14                          </one-of>
15                      </rule>
16                  </grammar>
17          </field>
18          <block>
19                  <submit next="process.vxml"/>
20          </block>
21    </form>
22 </vxml>
```

Kód: egyszerű VoiceXML alkalmazás

A fenti VoiceXML kód eredménye: a megfelelő telefonszám felhívása esetén a rendszer felolvassa a „Kérlek válassz: Hírek, mozi vagy sport.” részletet. Ezután a felhasználó válaszol, majd a rendszer továbbítja őt a „`process.vxml`” oldalra, ahol a válasz feldolgozása történik.

2.2.1 Alapvető szintaxis

Ez a példaalkalmazás néhány alapvető dolgot mutat be a VoiceXML-ből. Nyers szövegből (plain text) és *címké*kből (tag) áll. Utóbbi a kulcsszavakat és kifejezéseket jelenti, amelyeket zárójel határol (< és >). Ebben a példában a 6. és 11-13. sor kivételével minden más sor egy címkének felel meg.
A címkéknek lehetnek *tulajdonság*aik a zárójeleken belül. Mindegyik tulajdonság egy *név*ből és a hozzá tartozó *érték*ből áll, melyek között egyenlőségjel (=) van, az érték pedig idézőjelek (") között található. Erre példa a `version` tulajdonság a 2. sorban és a `name` tulajdonság a 4. sorban.
Ebben a példában a legtöbb címke párokban fordul elő. A `<vxml>` címkéhez a `</vxml>` címke tartozik, hasonlóan a `<form>` és `<block>` címkékhez. A párokban használatos címkéket *konténer*nek hívjuk, mivel valamilyen tartalom (pl. szöveg vagy további címkék) található bennük.
Ezzel szemben egyes címkéket önmagukban használunk, ezeket *önálló* vagy *üres* címkének hívjuk. Az üres címkéknek nincs tartalmuk, de lehetnek tulajdonságaik. Ebben a példában az egyetlen ilyen a `<submit... />` címke a 19. sorban. Vegyük észre, hogy az üres címkében perjel (/) található a záró > karakter előtt.
Vigyázat: a HTML-hez képest a VoiceXML sokkal szigorúbb a helyes szintaxis figyelembevételénél. Ha korábban foglalkoztál HTML-lel, akkor sok egyszerűsítéssel találkozhattál (pl. idézőjelek elhagyása), amelyek VoiceXML-ben nem lehetségesek. Emiatt törekedjünk mindig helyes szintaxis használatára.
HTML és VoiceXML szövegkörnyezetben használatos még az *elem* kifejezés is. Egy elem lehet:

- egy önálló címke az esetleges tulajdonságaival együtt; vagy
- egy konténer címke, a nyitó címkével és tulajdonságaival, a megfelelő záró címkével, és a kettő között található egyéb címkékkel együtt.

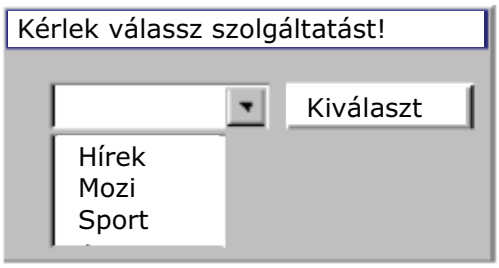
Ha egy elem tartalmaz egy másikat, akkor előbbit *szülő*nek, utóbbit *gyermek*nek nevezhetjük.

2.2.2 Fejléc

Az 1-2. sorok minden VoiceXML dokumentum elejére kötelezőek. Az első alkalmazásokban használhatjuk a fenti példában található sorokat. Ezek megfelelnek a szabványos VoiceXML leírásnak. Az utolsó sor (22.) szintén minden VoiceXML forráskódba szükséges, ez a 2. sorban található `<vxml>` címke záró része.

2.2.3 Törzs

A fejléc után következik a VoiceXML dokumentum fő része, a törzs. Tipikusan a törzs legalább egy `<form>` címkét tartalmaz. A `<form>` címke egy VoiceXML űrlapot definiál, ami a dokumentumnak egy olyan része, amely a felhasználóval történő interakciót végzi. Ez a VoiceXML-beli megfelelője a HTML űrlapnak. Ehhez például lehetne olyan HTML kódot írni, amely a következő választási lehetőséget jeleníti meg:



7. ábra HTML űrlap

VoiceXML-ben a fenti űrlap a `<form>`, `<field>` és `<submit>` címkékkel készíthető el. Az űrlap általában *mező*kből áll, amelyek a felhasználótól érkező információt kezelik. Ebben a példában az űrlapnak egy `selection`-nak nevezett mezője van (4-18. sor), amely a felhasználó választását tárolja.
A mezőn belül van egy `<prompt>` címke (5-7. sor), amely nyers szöveget tartalmaz. A VoiceXML értelmező (interpreter) a hozzá tartozó TTS, vagyis beszédszintetizátor moduljának segítségével a szöveget beszéddé alakítja át, és a létrejött hanganyagot lejátssza a felhasználónak. A beszédszintetizátor mellett lehetőség van egyéni előre rögzített hanganyagok hozzáadására is az `<audio>` címke segítségével.
A `<prompt>` után egy `<grammar>`, vagyis nyelvtan elem található (8-16. sor), amely definiálja a három lehetséges választ, amit a beszédfelismerő felismerhet és az értelmező elfogad. A `<rule>` segítségével nyelvtani szabályt adtunk meg, mely jelen esetben A `<one-of>`-on belül `<item>` elemek felsorolását jelenti. A `<grammar>` címke bonyolultabb formáinak segítségével lehetőség van komplex nyelvtanok megadására is, melyeket akár külön fájlban is tárolhatunk. A nyelvtanok általában kimondott szavakat vagy DTMF kódokat tartalmaznak; az alkalmazás mindegyiket tudja kezelni. A VoiceXML segítségével több különböző formátumú nyelvtan megadására is lehetőség van.
Miután az értelmező elküldi a `prompt`-ot a felhasználónak, várakozik a válaszra és megpróbálja összehasonlítani a választ a megadott nyelvtannal. Ha a válasz megfelelő, akkor az értelmező az adott mező `selection` változóját a válasznak megfelelőre állítja. Ez az érték az űrlap további részében is érvényes marad.
Végül az űrlap a `<block>` elemmel zárul, amely egy `<submit>` címkét tartalmaz (18-20. sor). Ezek a sorok átadják a vezérlést a `process.vxml`-nek, a `selection` változó értékét is továbbítva a feldolgozó szkriptnek.

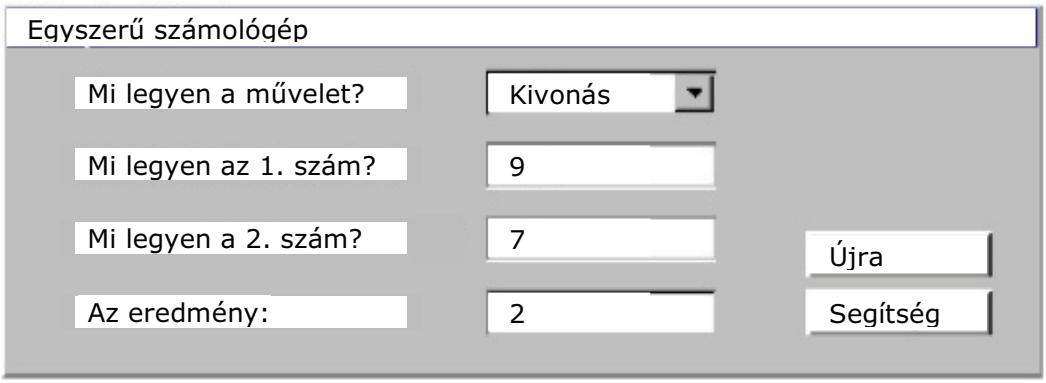
2.3 Egy egyszerű dialógus

Az eddigiekben megismerhettük a VoiceXML alapjait egy egyszerű szkripten keresztül, mely egyetlen mezőt tartalmazott. Most alaposabban megvizsgáljuk a felhasználóval történő interakció lehetőségeit, vagyis hogy hogyan készítsünk egy ember-gép dialógus mintaalkalmazást, amely hatékony és mégis egyszerűen használható.

2.3.1 Példa alkalmazás

Most írunk egy hosszabb programot, amely több bemenetet kér a felhasználótól és hibakezelés is van benne. Ez a program egy beszéddel vezérelt számológép lesz.

A felhasználó megnevezi a műveletet (összeadás, kivonás), majd mond két számot; a szkript pedig kiszámolja az eredményt és ezt fel is olvassa. Elsőként érdemes ezt egy HTML űrlaphoz hasonlítani, amely nagyjából így néz ki:



8. ábra Egyszerű számológép HTML űrlappal

Ennek VoiceXML megfelelője megvalósítható egy `<form>` címkével, amely három `<field>` elemet tartalmaz. De a könnyebb használat érdekében további lehetőségeket is hozzáadunk (pl. Újra és Segítség parancsszavak).
Ez a mintaalkalmazás hosszabb, mint az előző példában, ezért kisebb részletekben fogjuk vizsgálni.

2.3.2 Inicializálás és hibakezelés

A számológépünk a szokásos elemekkel kezdődik:

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version="2.0" xmlns="http://www.w3.org/2001/vxml">
```

Megjegyzések

A VoiceXML megjegyzések a HTML-hez hasonlóan működnek, vagyis `<!--` és `-->` közé kell tenni őket. Ami a két jel között található, azt az értelmező nem fogja figyelembe venni.
Például:

```
<!-- Itt kezdődik a számológép -->
```

Segítség és hibakezelés

Az inicializáció utolsó részeként néhány eseménykezelő elemet is készítünk, amelyeket segítség kérése és hiba esetén tud meghallgatni a felhasználó:

```
<noinput>
      Sajnálom, nem hallottam.
      <reprompt/>
</noinput>
```

A <noinput> jelöli, hogy mit csináljon az alkalmazás, ha a felhasználó nem mond be semmit (esetleg túl halkan mondja), és számot sem üt be. Ebben az esetben a fenti címke tartalmazza a felolvasandó szöveget, és meghívódik a <reprompt/> utasítás, aminek hatására a legutolsó <prompt> részhez tér vissza a rendszer.

```
<nomatch>
      Sajnálom, nem értettem.
      <reprompt/>
</nomatch>
```

A <nomatch> hasonló a <noinput>-hoz; azt adja meg, hogy mi történjen, ha a felhasználó válasza nem illeszkedik a megadott nyelvtanra, vagyis nem sikerült felismerni az elhangzott beszédet.

```
<help>
      Sajnálom, nem áll rendelkezésre segítség.
</help>
```

A <help> elem hatására a címkék közötti szöveget olvassa fel a rendszer, ha a felhasználó segítséget kér. Jelen esetben az üzenet nem sokat segít, de a későbbiekben a segítségkérés esetét is megfelelően kell kezelni.

2.3.3 Űrlapok és mezők

A <form> címkével kezdjük, amelyet egy <block>-ban lévő nyitó üzenet követ:

```
<form>
      <block>
            Ez a számológép mintaalkalmazás.
      </block>
```

Általánosságban a <block> címke olyan részt tartalmaz, amiben nincs szükség felhasználói interakcióra. Például arra használható, hogy a rendszer felolvas egy kezdő üzenetet, amire nem vár választ. Ezen kívül tartalmazhat valamilyen programozott logikát is, mint amilyen a korábbi példában előforduló <submit> címke volt.

2.3.4 Mezők formátuma

Most pedig írjuk meg az első mezőnket, ahol a felhasználó kiválasztja a számológépben elvégzendő műveletet:

```
<!-- Művelet bekérése -->
<field name="op">
      <prompt>
            Válassz műveletet:
            összeadás, kivonás.
      </prompt>
      <grammar root="r" mode="voice" type="application/srgs+xml">
            <rule id="r">
                  <one-of>
                        <item>összeadás</item>
                        <item>kivonás</item>
                  </one-of>
            </rule>
      </grammar>
      <help>
            Mondj egy műveletet!
            <reprompt/>
      </help>
      <filled>
            <prompt>
                  Rendben, most elvégezzük a választott
                  <value expr="op"/> műveletet két számon.
            </prompt>
      </filled>
</field>
```

Ez a mező egy kicsit bonyolultabb, mint az első példában volt. Bemutatja a mezők legfontosabb részeit: a beszédszintetizátor által lejátszandó promptokat, a beszédfelismerő által értelmezhető nyelvtanokat, és a felismerés után végzendő műveleteket.
A fenti mező neve "op", ez a felhasználó választását fogja tárolni. A mező tartalmaz egy <prompt> címkét, amely felolvassa a felhasználónak, hogy mik a válaszlehetőségek, a <grammar> rész pedig definiálja a felismerhető műveleteket. A nyelvtan azt mutatja, hogy a megadott szavak közül bármelyik elfogadható ebben a mezőben.
A következő címke a <help>, amelynek tartalmát a felhasználó hallja, ha segítséget kér. A <reprompt> rész megismétli a mezőben található, korábban már felolvasott promptot. A <noinput> és a <nomatch> címkék is felvehetők ide, amelyek felhasználóbarátabbá teszik az alkalmazás működését.
A mező következő része a <filled> címke. Ez határozza meg, hogy milyen műveletet hajtsunk végre, ha a mező kitöltődik, vagyis ha a felhasználó értelmes bemenetet adott meg. Ebben az esetben az alkalmazás lejátszik egy üzenetet, amely nyugtázza a felhasználó választását. A <value> címke arra szolgál, hogy a korábban definiált op értékét is felolvassuk.

Lássuk a következő mezőt, amely a számológép első operandusát kéri be:

```
<!-- Első szám bekérése -->
<field name="a" type="number">
      <prompt>
```

```
            Mi legyen az első szám?
        </prompt>
        <help>
            Kérlek mondj egy számot, ami az első operandus lesz.
        </help>
        <filled>
            <prompt>
                Az első szám: <value expr="a"/>.
            </prompt>
        </filled>
    </field>
```

A mező neve "a". Van egy type tulajdonsága is, amely definiálja, hogy ez egy szám típusú elem, így most nem kell külön megadni a beszédfelismerő nyelvtanát. A mező tartalmaz <prompt>, <help> és <filled> címkéket is, amelyek a korábbiakhoz hasonlóan működnek itt is. Vegyük észre, hogy jelen esetben a segítség prompt után nem olvassuk fel újra a kezdeti szöveget.

2.3.5 Változók

Ezután szükségünk van a második számot bekérő mezőre. A mező <filled> címkéje fogja a számolást elvégezni és az eredményt visszajuttatni a felhasználóhoz. De ahhoz, hogy a <filled> elem jól működjön, egy további elemet is hozzá kell adnunk:

```
    <!-- Eredmény tárolása -->
    <var name="result"/>
```

A <var> címke egy változót deklarál. A VoiceXML változók tartalmazhatnak számokat, szöveget, és néhány másfajta adatot is. Mint már láttuk, minden mezőhöz tartozik egy hozzárendelt változó, a name tulajdonsága alapján. Mivel már deklaráltunk megfelelő elemeket a számok tárolásához, ezekhez már tartoznak változók is. De mivel a szkript ki fogja számolni az eredményt is, amire eddig nem volt tároló elem, ezért szükség van egy változó explicit deklarálására.

2.3.6 Eredmény számítása

Ha mindez megvan, megírhatjuk az utolsó mező forráskódját is. Ez ismét egy hosszabb részlet, ezért kezdjük el az első néhány címkével:

```
    <!-- Második szám bekérése -->
    <field name="b" type="number">
        <prompt>
            Mi legyen a második szám?
        </prompt>
        <help>
            Kérlek mondj egy számot, ami a második operandus lesz.
        </help>
        <filled>
            <prompt>
                A második szám: <value expr="b"/>.
            </prompt>
```

A mező neve "b", és number típusú, mint az előző. Van <help> üzenete, és <filled> címkéje, amely megerősíti a felhasználó választását. Ezek után a dolgok kicsit bonyolódnak.

Szükségünk van egy olyan kódrészlet megírására, amely kiválasztja az elvégzendő műveletet, elvégzi a számolást és bemondja az eredményt. A <filled> címkének ezúttal kicsit több tartalma lesz.

```
<!-- Eredmény kiszámolása a művelettől függően -->
<if cond="op=='összeadás'">
      <assign name="result" expr="Number(a) + Number(b)"/>
      <prompt>
            <value expr="a"/> meg <value expr="b"/>
            egyenlő <value expr="result"/>
      </prompt>
```

Az <if> címke a döntés végrehajtására szolgál. A cond tulajdonság jelöli a tesztelnivaló feltételt. Ha a feltétel igaz, az <if> után következő kódrészlet fog lefutni. Ha a feltétel hamis, az értelmező átugrik a további címkékre, amíg nem talál egy <else>, <elseif> vagy záró </if> címkét. A cond értéke, op=='összeadás' egy olyan kifejezés, amely akkor igaz, ha az op mező tartalmazza az „összeadás” szót.
Ha a feltétel igaz, az értelmező lefuttatja az <assign> címkét, amely kiszámolja a Number(a)+Number(b) kifejezés értékét, és az összeget a result változóba helyezi. Az ember elsőre csak a+b -t írna, de a + jel szöveg összefűzést is jelenthetne, nem csak összeadást. A Number() függvény garantálja, hogy az argumentumok számok és nem pedig szöveg típusúak. Ha a felhasználó össze akarja adni a 30-at és a 14-et, az eredménynek 44-nek kell lennie és nem 3014-nek!
Az <assign> részt egy <prompt> követi, amelyben tájékoztatjuk a felhasználót az eredményről. Ismét a <value> címkét használjuk arra, hogy a bemeneti paraméterek és az eredmény értékét tároljuk. Ez volt tehát az összeadás megvalósítása.

Szükségünk van egy következő részre, amely a kivonást kezeli:

```
<elseif cond="op=='kivonás'"/>
      <assign name="result" expr="a - b"/>
      <prompt>
            <value expr="a"/> mínusz <value expr="b"/>
            egyenlő <value expr="result"/>
      </prompt>
</if>
```

Ezek a címkék nagyon hasonlóak az összeadásnál leírtakhoz. Az <if> helyett most <elseif>-fel kezdtünk, mivel választások sorozatát készítjük el, amelyek közül egyszerre csak egy fog lefutni. Az <assign> részben most írhatunk a-b kifejezést a Number() függvények használata nélkül, mivel a - szimbólumnak csak egy jelentése van.

Itt most az <elseif>-et használtuk <if> helyett. Végül pedig a </if> címkével zárjuk le a feltételes szerkezetet.

2.3.7 Űrlap alaphelyzetbe állítása

Már majdnem befejeztük a dialógus elkészítését, csak egy dolog van hátra:

```
                <!-- Alaphelyzetbe állítás -->
                <clear/>
```

A <clear> címkét használjuk arra, hogy az űrlapot az alaphelyzetébe visszaállítsuk, vagyis ezzel kitöröljük a változók értékeit. Ha úgy szükséges, a <clear> segítségével lehetőség van csak bizonyos változók törlésére is. Az alaphelyzetbe állítás hatására a futás visszatér az űrlap tetejére, vagyis a felhasználó újra használhatja a számológépet.

Végül le kell zárni a megnyitott címkéket:

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

2.3.8 Összefoglalás

Ezen rövid VoiceXML bevezetőben a következőket mutattuk be:

- megjegyzések hozzáfűzése a forráskódhoz,
- a <help>, <noinput>, <nomatch> parancsok használata a felhasználók segítésére,
- a <filled> címkék használata,
- változók deklarálása <var>-ral, műveletek végzése <assign>-nal, és az eredmények promptba helyezése <value> segítségével,
- feltételes kifejezés létrehozása <if>, <elseif>, <else> használatával,
- űrlap alapállapotba állítása <clear/>-rel.

2.4 VoiceXML parancsok

<assign>	változóhoz érték rendelése
<block>	több elem összefogása, melyek együtt futnak le
<break>	szünet beszúrása
<choice>	menün belüli választási lehetőség definiálása
<clear>	változók újrainicializálása
<disconnect>	hívás megszüntetése, VoiceXML folyamat leállítása
<else>	feltételes szerkezetben egyébként rész jelölése
<elseif>	feltételes szerkezetben következő feltétel jelölése
<field>	hívóval való dialógus létrehozása, melynek célja információ gyűjtése
<filled>	definiálja, hogy mi a teendő, ha egy adott mező kitöltődik
<form>	promptok lejátszása és a felhasználóval történő interakció vezérlése field elemeken keresztül
<goto>	explicit ugrás egy helyre
<grammar>	nyelvtan definiálása
<help>	segítségkérés kezelése
<if>	feltételes szerkezetben első feltétel jelölése

<item>	XML nyelvtanban egy bemondható szót jelöl
<menu>	egyszerű dialógus létrehozása, amelyben fix lehetőségek közül választhatunk
<noinput>	bemondás hiányának kezelése
<nomatch>	hibás bemondás kezelése
<one-of>	XML nyelvtanban alternatívák definiálása
<prompt>	meghívja a beszédszintetizátort adott szöveg lejátszására
<reprompt>	megismétli a legutolsó <reprompt> üzenetet
<rule>	XML nyelvtanban szabály definiálása
<script>	kliensoldalú (pl. JavaScript) kód definiálása
<submit>	új VoiceXML dokumentumba irányítás http GET vagy POST segítségével
<var>	változó deklarálása
<vxml>	VoiceXML dokumentum gyökér eleme

2.5 Felhasznált irodalom

[1] Chetan Sharma & Jeff Kunins, VoiceXML: Strategies and Techniques for Effective Voice Application Development with VoiceXML 2.0, Wiley 2002

[2] BeVocal, Inc., „VoiceXML Tutorial”, 2005, http://cafe.bevocal.com/docs/tutorial/tutorial.pdf

[3] Voice Extensible Markup Language (VoiceXML) Version 2.0, http://www.w3.org/TR/voicexml20/

[4] VoiceXML Development Guide, Version 2.1, http://www.vxml.org/

További példaprogramok a következő hivatkozásokon találhatóak:

http://cafe.bevocal.com/resources/voicexml_samples/index.html

https://studio.tellme.com/library2/code/

http://developer.voicegenie.com/examples_VoiceGenie.php