Multimodal interfaces on mobile devices: mangling speech and graphical

Rob Van Roey

promotor :
Prof. dr. Karin CONINX

co-promotor :
Prof. dr. Kris LUYTEN

Eindverhandeling voorgedragen tot het bekomen van de graad
Master in de informatica human-computer interaction

Universiteit Maastricht

universiteit hasselt

Multimodal interfaces on mobile devices: mangling speech and graphical interaction

Academiejaar 2006 - 2007

Thesis voorgedragen tot het behalen van de graad van master in de informatica afstudeervariant HCI

Rob Van Roey
Promotor: Prof. dr. Karin Coninx
Co-Promotor: Prof. dr. Kris Luyten
Begeleider: Jo Vermeulen
Universiteit Hasselt

Inhoudsopgave

Lijst van figuren

Deel I

Voorbeschouwing

Hoofdstuk 1

Abstract

Er is een groeiende nood aan *multimodale* applicaties door de opkomst van mobiele apparaten. Mobiele gebruikers hebben hun handen en ogen niet altijd vrij om hun mobiel apparaat te bedienen. Buiten *directe manipulatie* is er nood aan een andere interactiemodaliteit om dit probleem te verhelpen. *Spraak* als interactiemodaliteit stelt de gebruiker in staat om een mobiel apparaat te bedienen zonder hun handen en ogen in beslag te nemen. *XHTML + Voice (X+V)* is een markup taal die beide modaliteiten combineert en op een eenvoudige manier linkt. X+V wordt gebruikt om de multimodale user interface voor te stellen. Er zijn tal van multimodale browsers voor handen die X+V kunnen renderen, waaronder de *Opera* webbrowser.

Om multimodale user interfaces te realiseren wordt er gebruik gemaakt van de *User Interface Markup Language (UIML)*. UIML is een *High Level User Interface Description Language (HLUIDL)* die de user interface beschrijft. Deze abstracte beschrijving wordt vervolgens gemapt naar een X+V document. Als UIML raamwerk wordt er gebruik gemaakt van Uiml.Net. Het doel van deze thesis is om Uiml.Net uit te breiden met support voor multimodale users interfaces. De architectuur van de Uiml.Net renderer wordt aangepast zodat een UIML document op een XML document gemapt kan worden. In de implementatie wordt er zo veel mogelijk gesteund op de al bestaande architectuur. Om een X+V document te kunnen renderen wordt er een X+V specifieke renderer aangemaakt. Met behulp van een X+V vocabulary kan een UIML document gemapt worden op een X+V document.

1.1 Woord van dank

Ik zou graag enkele personen willen danken voor hun medewerking aan deze masterproef. Allereerst zou ik de promotor van deze thesis, *Prof. Dr. Karin Coninx* willen bedanken om het mij mogelijk te maken dit werkt te realiseren. *Prof. Dr. Kris Luyten* wil ik bedanken voor de talrijke suggesties die dit werk tot een beter geheel hebben gemaakt. Mijn begeleider *Jo Vermeulen* wil ik in het bijzonder bedanken voor de vele tijd die hij gestoken heeft in zowel het herhaaldelijk nalezen en verbeteren van mijn tekst, als voor de vele nuttige opmerkingen en suggesties met betrekking tot de implementatie.

Ik zou ook mijn ouders *Valere* en *Liliane* en broer *Bert* willen bedanken voor de steun die ze mij door de jaren hebben gegeven tijdens mijn studies. Zij zorgden ervoor dat ik altijd gemotiveerd bleef tijdens de 4 jaren op Universiteit Hasselt. Ik wil tenslotte mijn medestudenten *Wouter Groeneveld* en *Jan Meskens* bedanken, voor de goede samenwerking en verstandhouding in de groepsprojecten.

Hoofdstuk 2

Inleiding

2.1 Probleemstelling

Mobiele apparaten zoals PDA's en GSM's worden als maar geavanceerder en hebben een veel grotere rekenkracht dan enkele jaren geleden. Dit opent nieuwe mogelijkheden voor het leveren van multimodaliteit om verder te gaan dan enkel visuele interactie. Het doel van dit werk is om de ontwikkeling van multimodale applicaties voor mobiele applicaties eenvoudiger te maken voor een user interface ontwerper. Om de user interfaces te beschrijven wordt er gebruik gemaakt van UIML. Als UIML raamwerk wordt er gebruik gemaakt van Uiml.net. Om een user interface te omschrijven die van beide modaliteiten gebruik maakt, wordt er XHTML + Voice gebruikt, een combinatie van Extensible Hypertext Markup Language(XHTML) en VoiceXML 2.0.

2.2 Inhoud

In deel II wordt er meer uitleg gegeven over de gebruikte technologiëen en begrippen. Gerelateerd werk wordt evenzeer in dit deel besproken.

Details over de implementatie van deze thesis worden gegeven in deel III. In dit deel wordt er meer uitleg gegeven over de aanpassingen aan de Uiml.Net architectuur. Vervolgens wordt de XHTML+Voice vocabulary besproken, gevolgd door meer details over de X+VRenderer, die met behulp van de XHTML+Voice vocabulary een XHTML+Voice document kan renderen vanuit een UIML document. In deel IV wordt een samenvatting gegeven van deze thesis, en er worden enkele problemen en tekortkomingen van de implementatie besproken.

Deel II

Onderzoekswerk

Hoofdstuk 3

Gebruikte technologiëen

In dit hoofdstuk worden enkele bestaande technologiëen en begrippen die in dit werk van belang zijn nader toegelicht. Sectie 3.1 geeft een duidelijke omschrijving van het begrip multimodaliteit en van multimodale interfaces. De daarop volgende sectie 3.2, geeft meer uitleg over XHTML+Voice en het gebruik er van. Sectie 3.3 beschrijft de User Interface Markup Language(UIML) en de laatste sectie 3.4 beschrijft de architectuur van een renderer voor UIML, namelijk Uiml.Net.

3.1 Multimodaliteit

Multimodale interfaces zijn interfaces waarbij de gebruiker meerdere modaliteiten kan aanwenden om te interageren met het systeem. Er zijn verschillende definities van modaliteit terug te vinden, de meest voorkomende zijn:

- Een zintuiglijke waarneming (*communicatiemedium*)
- Een manier om informatie uit te wisselen (*communicatiekanaal*)

In [7] wordt een onderscheid gemaakt tussen een modaliteit en een medium. Een modaliteit concentreert zich op de syntactische en semantische eigenschappen van het signaal, in andere woorden hoe het signaal in staat is te communiceren. Een medium echter, legt de focus op de produktie, opslag en transmissie door de machine van signalen. Een modaliteit bevat een medium, en is betrokken met de reactie van de machine op de inhoud van het bericht.

Multimodale interactie geeft de gebruiker verschillende methoden van interactie met een systeem, dat verder gaat dan de traditionele keyboard en mouse input/output. Een paar voorbeelden van modaliteiten: directe manipulatie (traditionele keyboard en mouse input/output), spraak, gestures en haptic feedback. Een multimodale interface combineert 2 of meerdere modaliteiten om zo een betere interface te verkrijgen. De sterkte van de ene modaliteit kan worden gebruikt om de zwaktes van een andere modaliteit weg te werken. In deze thesis ligt de nadruk op de modaliteiten spraak en directe manipulatie. De zwaktes van directe manipulatie worden gecompenseerd door de sterktes van spraak, en omgekeerd. Voor de realisatie hiervan wordt gebruik gemaakt van *XHTML+Voice* (zie 3.2).

Sterktes en zwaktes van directe manipulatie : Interfaces die gebruik maken van directe manipulatie zijn zeer succesvol, en niet meer weg te denken uit de computerwereld. Enkele reden voor dit succes:

- Indien goed ontworpen en gebaseerd op gekende metaforen die directe toegang tot semantische objecten toelaten, is directe manipulatie intuïtief.
- Indien goed ontworpen, kunnen directe manipulatie interfaces consistente *look and feel* hebben, zodat gebruikers van het ene programma een ander programma snel kunnen leren gebruiken.
- Het gebruik van menus verduidelijkt de mogelijke opties, waardoor de errors in het formuleren van commandos en het specificeren van argument wordt geminimaliseerd.

Directe manipulatie voldoet echter niet aan alle behoeftes van de gebruiker. Enkele van de zwaktes van directe manipulatie:

- Het identificeren en beschrijven van entiteiten is niet mogelijk. Er is geen manier om entiteiten te vinden die een bepaalde verzameling eigenschappen hebben, om te identificeren hoeveel entiteiten relevant zijn, om te identificeren welke niet relevant zijn, en om temporele voorwaarden op deze eigenschappen toe te passen.
- Het is niet mogelijk om acties te vertragen. Door de nadruk op snelle, grafische respons op acties, is de actietijd van een bepaalde actie nauw verbonden met de tijd van de actie aanroep.

Tabel 3.1: Sterktes en zwaktes van beide modaliteiten

	Direct Manipulation	Speech
Sterktes	1.Intuïtief 2.Consistente look and feel 3.Opties duidelijk in interface 4.Fail safe 5.Feedback 6.Point,act 7.Directe interactie met semantische objecten 8.In het hier en nu handelen	1.Intuïtief 2.Beschrijvingen zoals: a.Kwantificatie b.Negatie c.Temporele relaties 3.Context 4.Anaforen 5.Vertraagde acties mogelijk
Zwaktes	1.Beschrijvingen zoals: a.Kwantificatie b.Negatie c.Temporele relaties 2.Anaforen 3.Operaties op grote verzamelingen van objecten 4.Vertraagde acties zijn moeilijk	1.Onmogelijke coverage van alle mogelijkheden 2.Overkill voor korte en frequente queries 3.Moeilijk om context te realiseren en te navigeren 4.Anaforen zijn problematisch 5.Error gevoelig 6.Dubbelzinnig

Sterktes en zwaktes van spraak Bij spraakgestuurde interfaces wordt er gebruik gemaakt van natuurlijke taal voor interactie met het systeem. In de meeste gevallen zal hiervoor de Engelse taal gebruikt worden. Voordelen van het gebruik van natuurlijke taal zijn:

- De natuurlijke taal bevat een verzameling van methodes voor de beschrijving van bepaalde entiteiten. Dit stelt de gebruiker in staat om objecten te identificeren, om gebeurtenissen te identificeren, en om tijd periodes beschrijven.
- De natuurlijke taal geeft de mogelijkheid om het opnieuw beschrijven van entiteiten te minimaliseren door gebruik te maken van verwijzende voornaamwoorden en andere anaforische uitdrukkingen.

Wanneer de natuurlijke taal in een interface wordt gebruikt, duiken er ook nadelen op:

- De interpretatie van de natuurlijke taal is niet altijd ondubbelzinnig. Meerdere pogingen zijn nodig om een bepaalde commando aan het systeem te geven dat correct is. Deze foutgevoeligheid kan tot frustratie en desillusie van de gebruiker leiden.
- De verwijzingen in de natuurlijke taal zijn niet makkelijk correct te identificeren door het systeem.

Over het algemeen hebben directe manipulatie en spraak complementaire voor- en nadelen. Deze worden samengevat in tabel 3.1. [17] [16]

3.2 XHTML + Voice

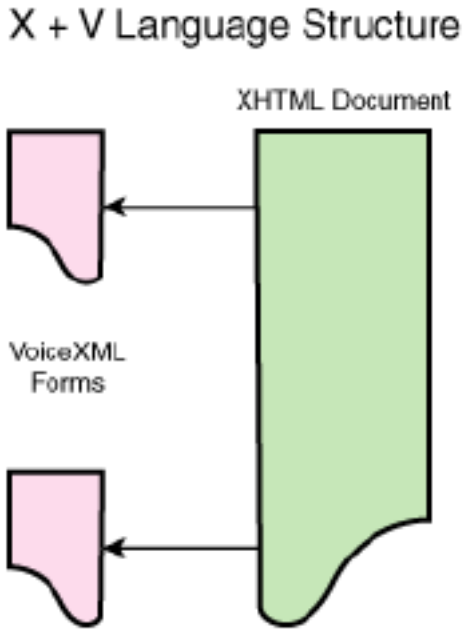
XHTML + Voice [1] is een Web markup taal voor het ontwikkelen van multimodale applicaties. Zoals VoiceXML [2], dient X+V om aan de stijgende vraag van gebruikers naar voice-based interactie in kleine en mobiele apparaten te voldoen. In tegenstelling tot VoiceXML, gebruikt X+V zowel spraak als visuele elementen, en opent een nieuwe waaier aan mogelijkheden voor draadloze user interface ontwikkeling. X+V is een voorgestelde markup taal voor het ontwikkelen van multimodale webpagina's, waarbij de modaliteiten spraak en directe manipulatie gecombineerd worden. Voor meer informatie over multimodaliteit zie 3.1. X+V combineert XHTML en een subset van VoiceXML. XHTML [3] is in essentie HTML 4.0 aangepast om te voldoen aan te regels van XML [4]. Het is de huidige standaard voor het bouwen van webpagina's. VoiceXML was een van de eerste XML-gebaseerde talen ontwikkeld in de *World Wide Web Consortium(W3C)* [5]. VoiceXML levert een simpel, gestandaardiseerd formaat voor het bouwen van spraakgebaseerde applicaties. XHTML en VoiceXML samen geven web ontwikkelaars de mogelijkheid om spraak input en output toe te voegen aan traditionele web pagina's die focussen op het grafische aspect. X+V is momenteel in *proposal* fase, maar er wordt voorspeld dat in de toekomst X+V de standaard wordt voor web applicaties met zowel enkel visuele, als enkel spraakgestuurde en multimodale interactie.

Architectuur van X+V Multimodale web applicaties geschreven in X+V bestaan uit een visuele markup, een verzameling van deeltjes van voice markup, en event markup die de applicatie vertelt wanneer de applicatie welke deeltjes moet activeren. Voor de visuele markup, gebruikt X+V de gekende XHTML standaard. Voor voice markup, gebruikt X+V een subset van VoiceXML gedefiniëerd door het VoiceXML Form construct. Voor het associëren van VoiceXML met visuele interface elementen, gebruikt X+V de XML Events standard. Dit zijn allemaal officiële standaarden gedefiniëerd door de *IETF* [6] die internet standaarden beheert. Voor een visuele weergave van de architectuur van X+V zie figuur 3.1. Een voorbeeld van multimodale interactie is te zien in figuur 3.2.

Omdat alle delen van X+V XML-compliant zijn, kan het voice markup gedeelte in 2 manieren opgeslagen worden: In hetzelfde bestand als het XHTML gedeelte of in afzonderlijke bestanden. Het scheiden van voice markup van visuele markup heeft verschillende voordelen:

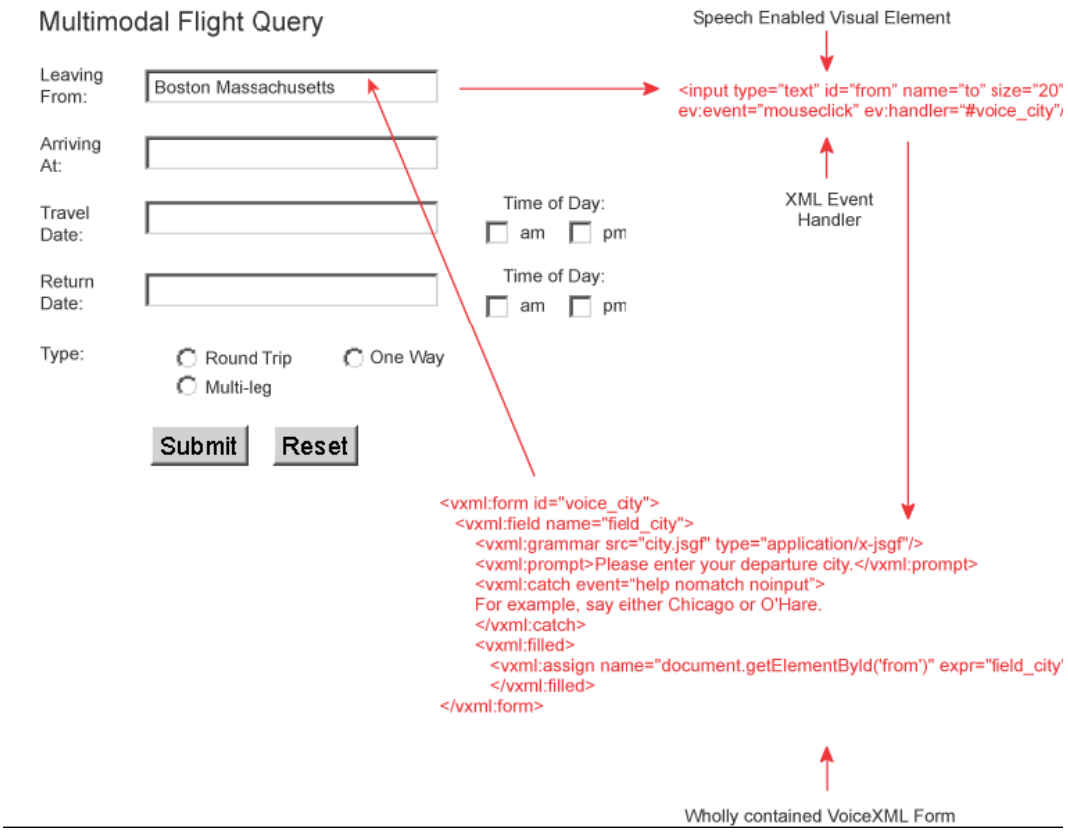
- De scheiding zorgt voor meer flexibiliteit in het ontwikkelen van applicaties.
- Deeltjes van VoiceXML kunnen hergebruikt worden in verscheidene XHTML pagina's.
- De scheiding laat toe dat er deeltjes VoiceXML kunnen herbruikt worden in andere containers dan XHTML. Zo kunnen er forms uit een bestaand VoiceXML document gebruikt worden.

[1] XHTML+Voice Profile 1.2 http://www.voicexml.org/specs/multimodal/x+v/12/
[2] Voice Extensible Markup Language (VoiceXML) Version 2.0 http://www.w3.org/TR/voicexml20/
[3] Extensible Hypertext Markup Language (XHTML), W3C, http://www.xhtml.org/
[4] Extensible Markup Language 1.0 (XML), W3C, http://www.w3.org/xml/
[5] World Wide Web Consortium http://www.w3.org/
[6] Internet Engineering Task Force http://www.ietf.org/



Figuur 3.1: X+V architectuur

Listing 3.1 toont een *Hello World* voorbeeld in X+V. , Zie [9] voor meer informatie over XHTML + Voice.



Figuur 3.2: X+V multimodaal voorbeeld([9])

Hello World voorbeeld in X+V.

Listing 3.1: "Hello World"voorbeeld in X+V

```
<?xml version="1.0"?>
<html
xmlns="http://www.w3.org/1999/xhtml"
xmlns:vxml="http://www.w3.org/2001/vxml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xv="http://www.voicexml.org/2002/xhtml+voice"
>
  <head>
    <title>XHTML+Voice Example</title>
    <!-- voice handler -->
    <vxml:form id="sayHello">
      <vxml:block><vxml:prompt xv:src="#hello"/>
      </vxml:block>
    </vxml:form>
  </head>
  <body>
    <h1>XHTML+Voice Example</h1>
    <p id="hello" ev:event="click" ev:handler="#sayHello">
      Hello World!
    </p>
  </body>
</html>
```

3.3 UIML

De User Interface Markup Language[2] is een gestandaardiseerde XML-gebaseerde metataal voor het beschrijven van een User Interface(UI). Het geeft een HLUIDL (*high level user interface description language*). Een HLUIDL is een taal die de presentatie van een user interface op een platformonafhankelijke manier specificeert. Een HLUID is een beschrijving van een user interface die zowel onafhankelijk is van widget set, als van toestel. Op deze manier kan een user interface beschreven worden, zonder rekening te houden met het doelplatform en de te gebruiken widget set (*System.Windows.Forms, GTK#, ...*). Er wordt dus één taal gebruikt voor het beschrijven van een user interface voor meerdere apparaten. Dit wil echter niet zeggen dat er slechts één UI beschrijving nodig is voor al deze apparaten.

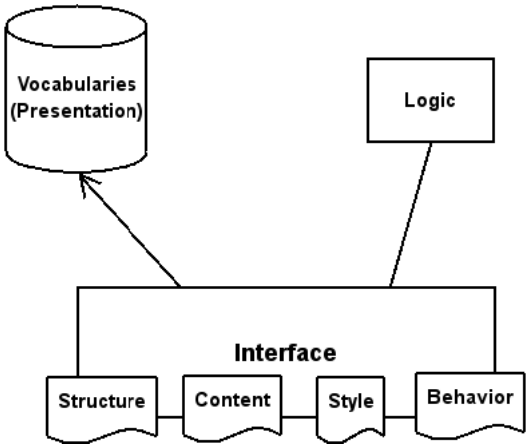
Meta-Interface Model Een UIML document bestaat uit verschillende delen, die samen het *Meta-Interface Model* vormen. Zie figuur 3.3.

Interface: beschrijft vier delen van de UI:

- Structure: Beschrijft de hiërarchie van de widgets voor de user interface. Definieert de verschillende delen van de UI, en de abstracte widget naam van elk deel. Voor elk part dient een unieke identifier opgegeven worden, samen met de naam van klasse van het part.
- Style: Beschrijft de eigenschappen(*properties*) van de parts gedefinieerd in structure. Eigenschappen zoals kleur, lettertype, tekst en layout van een bepaald widget kunnen zo verandert worden. Bij de beschrijving van een eigenschap, dient de part-name, de naam van eigenschap en de waarde van deze eigenschap gegeven te worden.
- Content: Scheidt de inhoud van de interface van de andere delen. De informatie die aan de gebruiker wordt gepresenteerd, wordt hierin beschreven. bvb. de items van een lijst
- Behavior: Defineert *rules* en *acties* die uitgevoerd worden wanneer er aan een bepaalde voorwaarde wordt voldaan. Een rule bestaat uit 2 delen: een *condition* en een *action*. Deze actie wordt uitgevoerd wanneer er aan de voorwaarde wordt voldaan. Deze condition kan bijvoorbeeld het plaatsvinden van een bepaald event zijn.

Peers: bestaat uit 2 delen:

- Presentation: worden *peers* genoemd in de UIML specificatie. Bevat de mapping naar de concrete UI toolkit. De abstracte klassen worden hier gelinkt aan de correcte klassen en events, en hun properties worden aan de correcte properties gelinkt. In listing 3.2 wordt de beschrijving gegeven van de mapping van een abstracte Label klasse, samen met de properties van deze klasse.
- Logic: Definieert hoe de user interface met de application logic wordt verbonden. Hierin wordt beschreven hoe bepaalde methods in de application logic kunnen worden opgeroepen door de UI. In listing 3.3 wordt de beschrijving gegeven van een methode die 2 strings concateneert, en die in een string het resultaat teruggeeft.



Figuur 3.3: Meta-Interface Model

Listing 3.2: Voorbeeld van d-class in presentation sectie van de SWF1.1 vocabulary

```
<d-class id="Label" used-in-tag="part" maps-type="class"
      maps-to="System.Windows.Forms.Label">
  <d-property id="text" return-type="System.String"
      maps-type="getMethod" maps-to="Text"/>
  <d-property id="text" maps-type="setMethod" maps-to="Text">
       <d-param type="System.String"/>
  </d-property>
  <d-property id="position" maps-type="setMethod" maps-to="Location">
       <d-param type="System.Drawing.Point"/>
  </d-property>
  <d-property id="size" maps-type="setMethod" maps-to="Size">
    <d-param type="System.Drawing.Size"/>
  </d-property>
  <d-property id="width" maps-type="setMethod" maps-to="Width">
    <d-param type="System.Int"/>
  </d-property>
  <d-property id="height" maps-type="setMethod" maps-to="Height">
    <d-param type="System.Int"/>
  </d-property>
  <d-property id="enabled" maps-type="setMethod" maps-to="Enabled">
    <d-param type="System.Boolean"/>
  </d-property>
  <d-property id="visible" maps-type="setMethod" maps-to="Visible">
    <d-param type="System.Boolean"/>
  </d-property>
  <d-property id="background" maps-type="setMethod" maps-to="BackColor">
    <d-param type="System.Drawing.Color"/>
  </d-property>
  <d-property id="foreground" maps-type="setMethod" maps-to="ForeColor">
    <d-param type="System.Drawing.Color"/>
  </d-property>
</d-class>
```

Listing 3.3: Voorbeeld van d-component in logic sectie van de SWF1.1 vocabulary

```
<d-component id="String" maps-to="System.String">
  <d-method id="concatenate" returns-value="string" maps-to="Concat">
    <d-param id="str0" type="System.String"/>
        <d-param id="str1" type="System.String"/>
  </d-method>
</d-component>
```

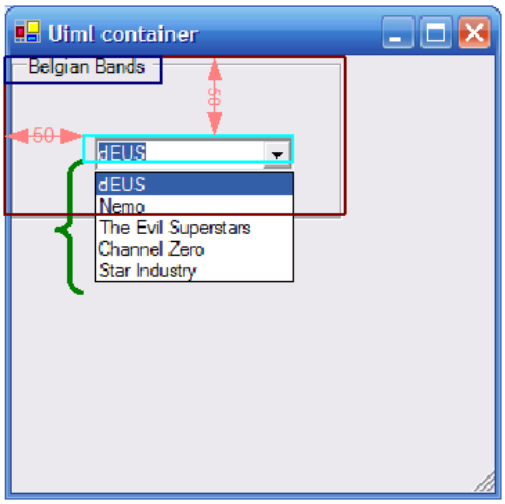
De structuur van een UIML document bevindt zich in listing 3.4.

Listing 3.4: Structuur van UIML document

```
<?xml version="1.0"?>
<!DOCTYPE uiml PUBLIC
        "-//UIT//DTD UIML 2.0 Draft//EN"
        "http://uiml.org/dtds/UIML20.dtd">
<uiml>
    <head> <meta> ... </meta> </head>
    <peers>
        <presentation> <component> ... </component> </presentation>
        <logic> <component> ... </component> </logic>
    </peers>
<template> ... </template>
<interface>
    <structure> <part> ... </part> </structure>
    <style> <property> ... </property> </style>
    <content> <constant> ... </constant> </content>
    <behavior> <rule> ... </rule> </behavior>
</interface> </interface>
</uiml>
```

In figuur 3.4 wordt er een simpel voorbeeld getoond van een UIML document, en de resulterende UI. Als widgetset wordt er hier gebruik gemaakt van *System.Windows.Forms*. In dit voorbeeld is duidelijk zichtbaar welke invloed de *structure* en *style* componenten van een UIML document hebben op de UI. Voor meer informatie over UIML, zie [1], [2], en [3].

```
<?xml version="1.0"?>
<!-- <!DOCTYPE uiml PUBLIC "-//Harmonia//DTD UIML 2.0 Draft//EN" "UIML3_0a.dtd"> -->
<uiml>
	<interface>
		<structure>
			<part id="Frame1" class="Frame">
				<part id="Com" class="Combo"/>
			</part>
		</structure>
		<style>
			<property part-name="Frame1" name="label">Belgian Bands</property>
			<property part-name="Com" name="position">50, 50</property>
			<property part-name="Com" name="content">
				<constant model="list">
					<constant value="dEUS"/>
					<constant value="Nemo"/>
					<constant value="The Evil Superstars"/>
					<constant value="Channel Zero"/>
					<constant value="Star Industry"/>
				</constant>
			</property>
		</style>
	</interface>
	<peers>
		<presentation base="http://research.edm.uhasselt.be/kris/projects/uiml.net/swf-1.1.uiml"/>
	</peers>
</uiml>
```



Figuur 3.4: Resultaat van Structure en Style elementen.

3.4 Uiml.Net

Uiml.Net[4] is een open source *UIML* renderer voor het .NET platform, geschreven in *C#*. Een HLUIDL zoals UIML kan gerendered of gecompileerd worden. Uiml.Net kiest voor het rendereren van UIML, dat complexer maar flexibeler is dan het compileren van UIML. De renderer interpreteert UIML. Het kan een UI renderen vanuit een UIML document, gebruik makend van verschillende widgets sets zoals *GTK#*, *System.Windows.Forms* en een klein deel van *Wx.Net*. Een UIML document bevat een presentatiemodel van een UI. Er wordt een onderscheid gemaakt tussen *Abstract Interaction Objects (AIO's)* en *Concrete Interaction Objects (CIO's)*. Het *interface* component van een UIML document beschrijft de AIO's, dit zijn platform-neutrale interactie objecten. De AIO's worden gemapt op CIO's, met behulp van de mappings gedefiniëerd in de vocabulary van het Uiml document. Deze CIO's zijn uitvoerbaar op een bepaald platform. Voor meer informatie over AIO's en CIO's, zie [6]. De architectuur van Uiml.Net wordt getoond in figuur 3.5.

3.4.1 Uiml.Net architectuur

De manier waarop Uiml.Net een UIML document verwerkt is als volgt (zie ook figuur 3.6):

1. De UIML renderer neemt een UIML document als input, en kijkt naar de rendering backend library waar in het UIML document naar verwezen wordt.

2. Er wordt een interne representatie van het UIML document opgebouwd. De *part* elementen van een boom worden verwerkt om zo een boom van abstracte interface componenten op te bouwen.

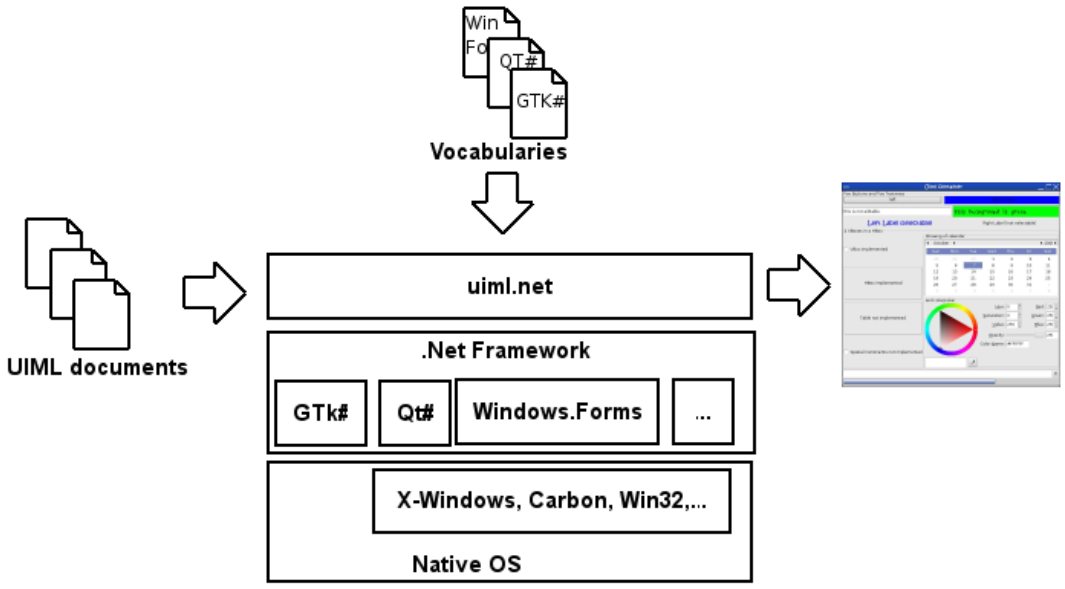
3. Voor elk *part* element wordt de overeenkomstige *style* toegepast.

4. Voor elk *part* element wordt het overeenkomstige *behavior* vastgekoppeld. De nodige libraries voor het uitvoeren van dit behavior zullen *just-in-time* geladen worden.

5. De gegenereerde boom wordt overgedragen aan de rendering module. Voor elke *part* tag wordt een overeenkomstige concreet widget geladen volgens de mapping die gespecifieerd is in de vocabulary en wordt gelinkt met de interne representatie. De *style* properties van elk concreet widget worden opgevraagd, gemapt naar concrete properties van het widget, en toegepast op het gegenereerde concrete widget.

In de laatste stap van dit proces wordt reflectie toegepast. Reflectie maakt het mogelijk om *at runtime* de concrete widgets te creëren en toe te voegen aan de UI. De rendering engine zelf echter heeft geen notie van de concrete widgets, het gebruikt alleen de vocabulary om dynamisch widgets te laden, door de beschikbare libraries te onderzoeken voor de geschikte classes. Reflectie laat ons dan toe om nieuwe instanties van deze classes aan te maken, zonder hun naam of structuur te kennen.

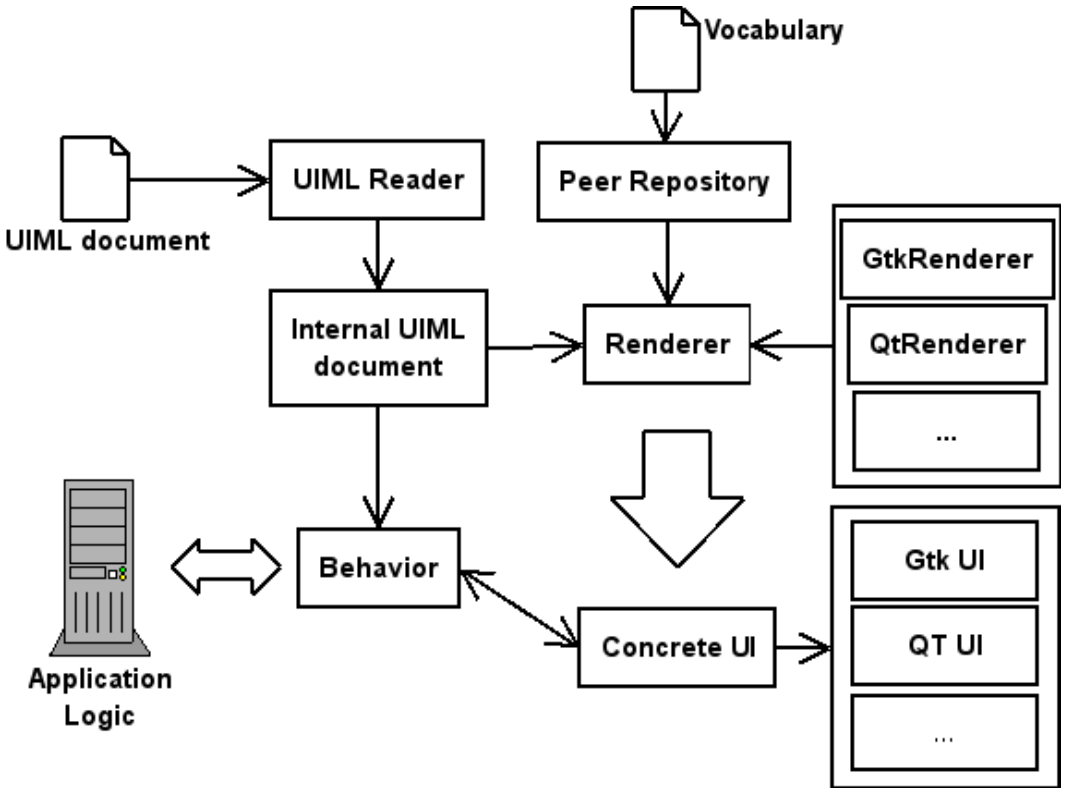
De *Uiml.Net Renderer* bevat één *rendering kern* en meerdere *rendering backends*. De rendering kern is het deel dat gebaseerd is op reflectie. De rendering kern zelf weet dus niet wat voor widget set het aan het creeëren is, dit hangt volledig af van de mapping gegeven door de vocabulary. Dit geeft als resultaat dat de rendering kern herbruikbaar is voor elke widget set.



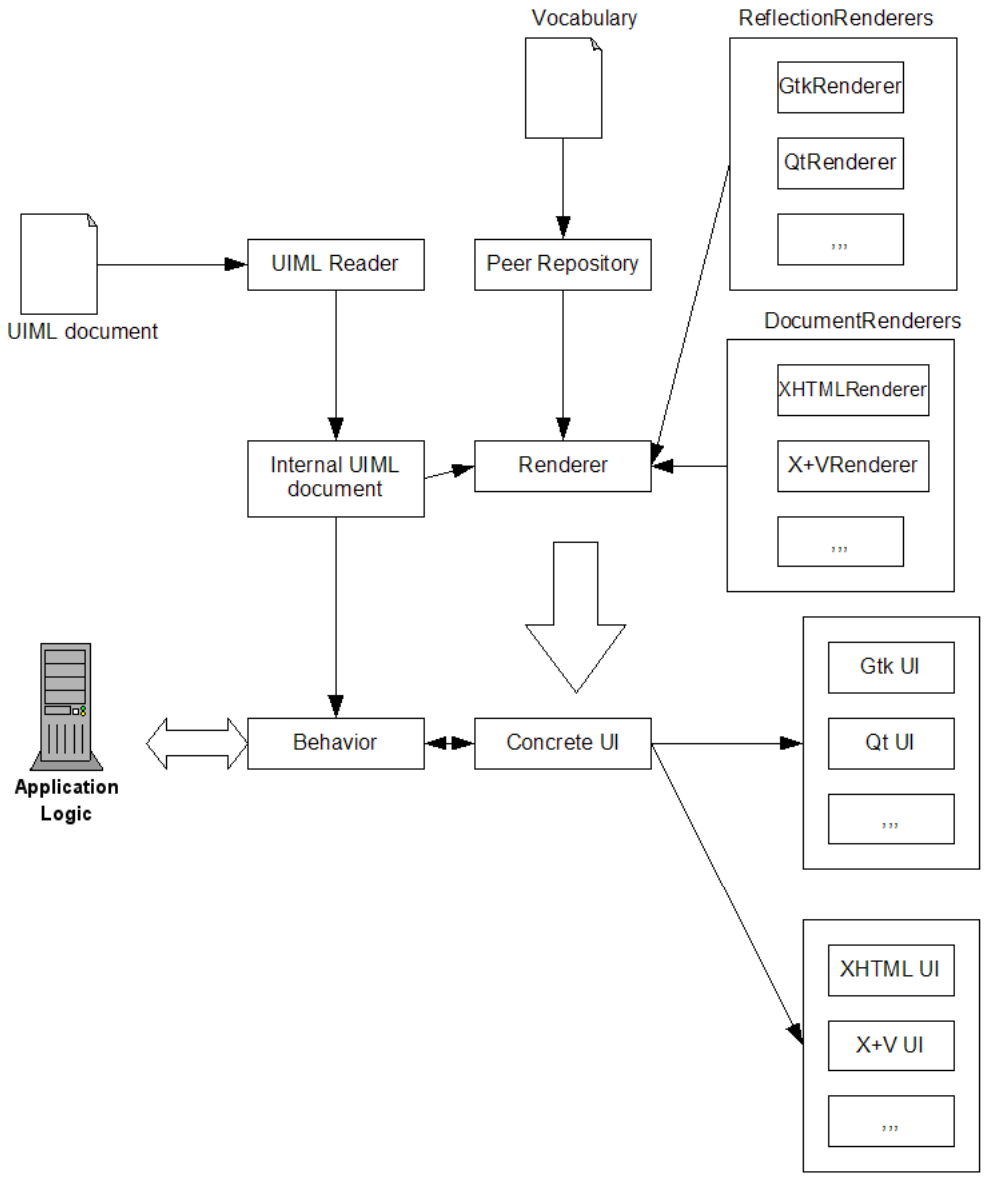
Figuur 3.5: Uiml.Net architectuur

Een rendering backend echter bevat code die specifiek is voor een bepaalde vocabulary, en enkel voor die vocabulary gebruikt wordt. Met behulp van de *BackEndFactory* wordt er een correcte rendering backend gekozen. De juiste backend wordt gezocht aan de hand van de vocabularynaam gespecifieerd in het Uiml document. Voor meer informatie over de architectuur van Uiml.Net, zie [4]

In dit werk wordt de Uiml.Net Renderer architectuur aangepast, zodat het mogelijk is om zowel reflectiegebaseerd als documentgebaseerd UIML te kunnnen renderen. De rendering kern wordt bijgevolg in 2 delen opgesplitst, namelijk een ReflectionRenderer en een DocumentRenderer. Dit wordt in meer detail besproken in hoofdstuk 5. Dit wordt schematisch weergegeven in figuur 3.7.



Figuur 3.6: Processing van een UIML document door Uiml.Net



Figuur 3.7: UIML document processing na wijziging van architectuur

Hoofdstuk 4

Gerelateerd Werk

In dit hoofdstuk worden er enkele al bestaande methoden besproken die in hetzelfde domein liggen als dit werk. De meeste gerelateerde werken beschrijven een manier om multimodale en/of apparaatonafhankelijke user interfaces te generen vanuit een HLUIDL. In de meeste methoden wordt gebruik gemaakt van UIML als HLUIDL, maar dit is niet noodzakelijk. In enkele van deze werken maakt men gebruik van modaliteit- en platformonafhankelijke widgets, ook generieke widgets genoemd, die vervolgens naar verschillende platformen en modaliteiten gemapt kunnen worden.

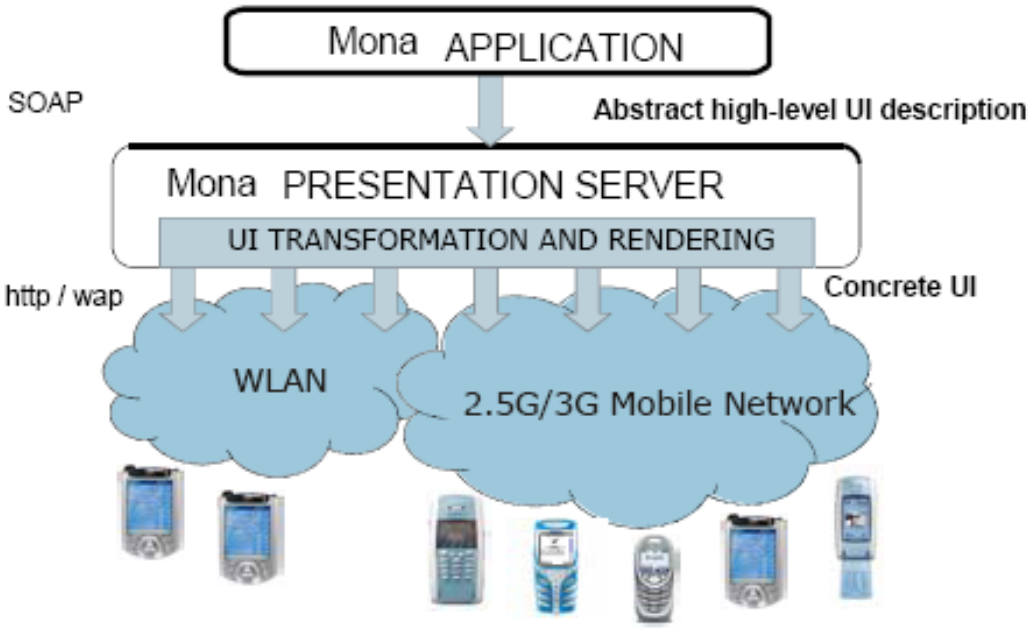
4.1 UIML gebaseerde methoden

4.1.1 Harmonia

Harmonia [1] biedt een Java-gebaseerde UIML renderer aan, die het grootste deel van de UIML specificatie implementeert. Harmonia heeft reeds een VoiceXml, WML (*Wireless Markup Language*, en HTML renderer gecreëerd. Verschillende andere implementaties zijn verzameld op *UIML.org* [2], maar van de meeste implementaties is geen source code vrijgegeven. In [13] wordt beschreven hoe vanuit UIML een Voice interface kan gerendered worden, waarbij gebruik wordt gemaakt van VoiceXML om de UI te representeren. Er wordt gebruik gemaakt van een generieke UIML vocabulary, die naar meerdere platformen gerendered kan worden. Generieke UIML kan getransformeerd worden naar een platform specifieke UIML. Deze platform specifieke UIML kan gerendered worden door een bestaande UIML renderer. Door deze tranformatie heeft de UI ontwerper beperkte controle over het eindresultaat of de uiteindelijke UI. Om de controle te vergroten kan er gebruik gemaakt worden van TIDE (*Transformation-based Integrated Development Environment*). In TIDE, schrijft de ontwerper UIML code, en de IDE genereert de UI. De link tussen de UIML code en de resulterende interface component is duidelijk zichtbaar. Aanpassingen die de ontwerper maakt in de UIML code, worden onmiddelijk getoond door de IDE.

[1] Harmonia, Inc. http://www.harmonia.com/

[2] UIML.org: Home of the User Interface Markup Language http://www.uiml.org



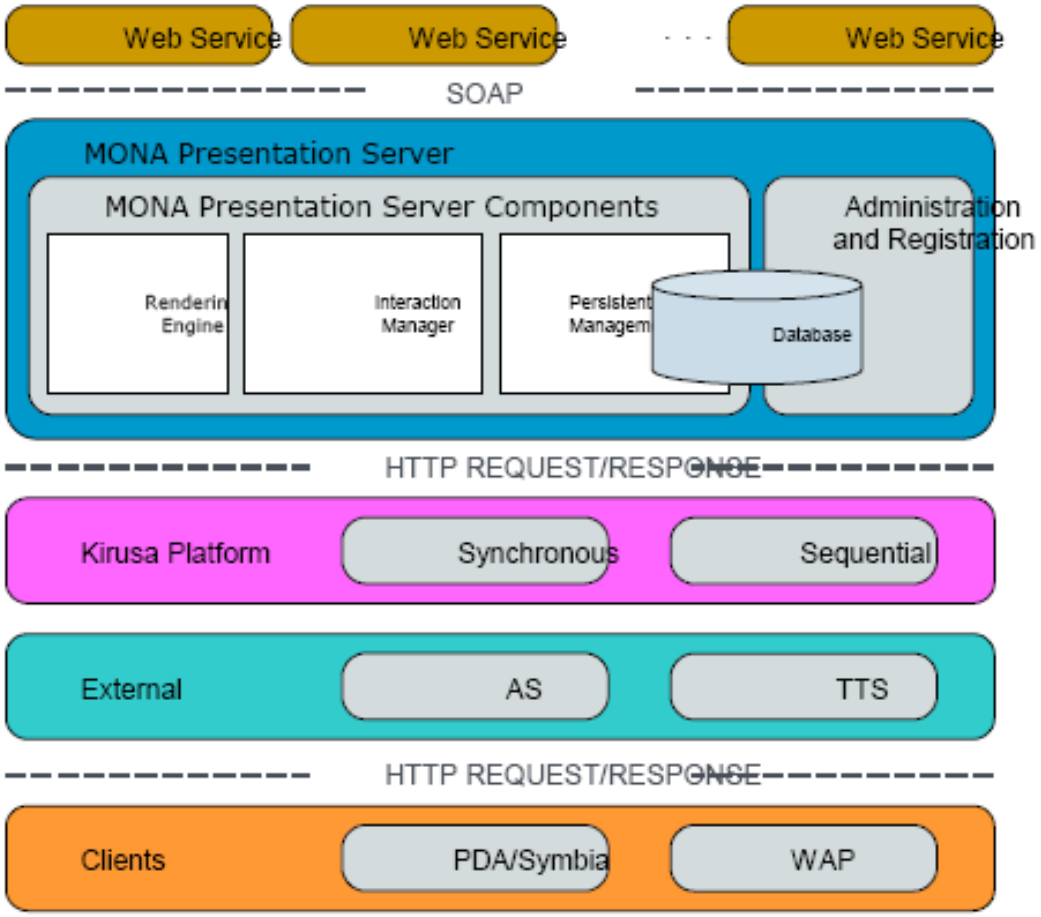
Figuur 4.1: MONA architectuur

4.1.2 Mona

In [11] wordt het Mona project besproken. Mona[3] is een onderzoeksproject met als onderwerp de infrastructuur van multimodale applicaties voor een brede waaier mobiele apparaten. Eén van de hoofddoelen van dit project is dat de applicatie ontwikkelaars afgeschermd worden van de verschillende apparaat specifieke details en van de modaliteit. De gebruiker moet ook de mogelijkheid krijgen om te kiezen tussen de verschillende interactie modes die de client aanbiedt.

De architectuur van Mona wordt getoond in figuur 4.1. Een Mona applicatie bestaat uit een enkele markup beschrijving van de UI. De Mona presentation server transformeert deze abstracte high-level beschrijving in een meer concretere grafische of multimodale beschrijving, die apparaatspecifiek is, en rekening houdt met voorkeuren van de gebruikers. De architectuur van de presentation server wordt getoond in figuur 4.2. Mona applicaties communiceren met de server met behulp van SOAP [4] boodschappen, die gebruikersacties en of interface beschrijvingen bevatten. Elk UI element wordt beschreven door een algemeen, abstract widget. Deze widgets beschrijven de functie van een bepaald UI component, en de presentation server kiest voor elk van deze widgets een geschikte realisatie, rekening houdend met de karakteristieken van het client apparaat, en de beschikbare interactie modaliteiten. Voor de voorstelling van deze abstracte widgets, wordt er gebruik gemaakt van een generieke UIML vocabulary. Voor meer informatie over Mona, zie [11] en [12]

[3]MONA:Mobile Multimodal Next Generation Applications http://mona.ftw.at/
[4]Simple Object Access Protocol (SOAP) 1.1 http://www.w3.org/TR/soap/



Figuur 4.2: MONA presentation server

UIML code	<?xml version="1.0" encoding="ISO-8859-1" ?> <!-- <!DOCTYPE uiml PUBLIC "-//UIT//DTD UIML 2.0 Draft//EN" "UIML2_0c.dtd"> --> <uiml> <interface> <structure> <part name="TopTag" class="Frame"> <part name="Enter_Name" class="TextEntry"> <style> <property name="type">text</property> <property name="vocabulary">Milena Mayora</property> <property name="label">UserID</property> <property name="extended-label"> Enter your user identification. </property> <property name="second-prompt"> Please enter your login name </property> </style> </part> </part> </structure> </interface> </uiml>
VXML rendering	<vxml> <field name="UserID"> <grammar>Milena Mayora</grammar> <prompt count="1 "> Enter your user name. </prompt> <prompt count="2 "> Please enter your login name </prompt> </field> </vxml>
HTML rendering	<html> <input type="text" name="UserID"/> </html>

Figuur 4.3: Rendering naar 2 platformen vanuit een generic widget vocabulary

4.1.3 Generic Widget Vocabulary

C.J. Plomp en O. Mayora-Ibarra beschrijven in [14] een *Generic Widget Vocabulary* die helpt in het genereren van grafische en voice gebaseerde user interfaces in een enkel bronformaat. Er wordt een meer generieke methodologie voorgesteld die het toelaat om user interfaces te genereren onafhankelijk van de interactie modaliteit. De interactie wordt omschreven met behulp van een verzameling voorgedefiniëerde, modaliteitonafhankelijke widgets, die gebruikt kunnen worden als basis bouwstenen voor de user interface. Het grootste probleem in het creëren van deze vocabulary is het verschil in dialoogverloop tussen spraakgestuurde en grafische interactie. Spraakgestuurde interactie verloopt sequentieel, terwijl grafische interactie parallel verloopt. UIML wordt gebruikt als metaformat voor het beschrijven van de UI. In de *peer* sectie van het UIML document worden de verschillende target presentatieformaten gespecificeerd. De gecreëerde vocabulary biedt voldoende mogelijkheden om een UI te beschrijven, onafhankelijk van het uitvoerformaat (grafisch of voice-based). In figuur 4.3 wordt getoond hoe een generieke vocabulary gerendered wordt naar 2 verschillende platformen, in dit geval VXML en HTML. in figuur 4.1 wordt de peer sectie getoond voor de gebruikte klassen in het voorbeeld.

Listing 4.1: peer sectie voor het voorbeeld in figuur 4.3

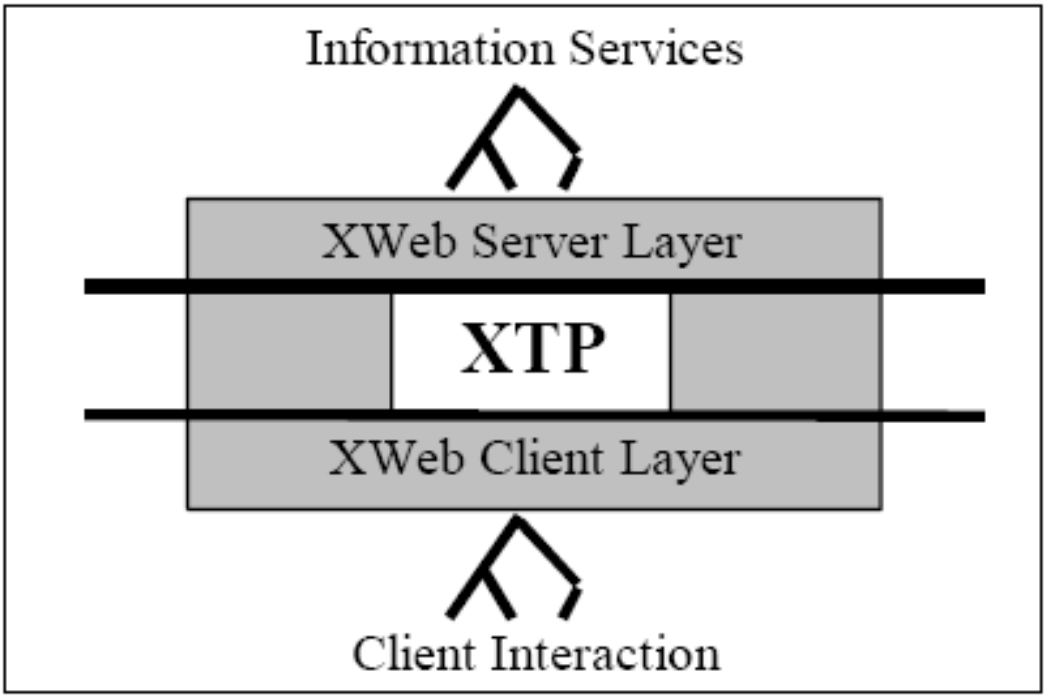
```
<peers>
 <presentation name="vxml">
  <d-class name="Frame" maps-type="tag" maps-to="field">
   <d-property name="label" maps-type="attribute" maps-to="field.name"/>
   <d-property name="vocabulary" maps-type="attribute" maps-to="grammar"/>
   <d-property name="extended-label" maps-presence="required"
    maps-tye="tag" maps-to="prompt">
        <d-param name="promp.count" type="integer">1</d-param>
   </d-property>
   <d-property name="second-prompt" maps-presence="required"
    maps-type="tag" maps-to="prompt">
        <d-param name="promp.count" type="integer">2</d-param>
   </d-property>
  </d-class>
 </presentation>
 <presentation name="html">
  <d-class name="Frame" maps-type="tag" maps-to="html/>
  <d-class name="TextEntry" maps-type="tag" maps-to="input">
   <d-property name="type" maps-type="attribute" maps-to="input.type"/>
   <d-property name="label" maps-type="attribute" maps-to="input.name"/>
  </d-class>
 </presentation>
</peers>
```

4.2 Niet UIML gebaseerde methoden

4.2.1 XWeb

In [21] wordt de XWeb architectuur beschreven voor client/server interactie. De nadruk ligt hier op collaboratie over de verschillende modaliteiten. Het doel van XWeb is dat een gebruiker op één van de interactieve platformen kan samenwerken met elke andere gebruiker op eender welk client platform. De architectuur van XWeb (zie figuur 4.4) is gebaseerd op die van het World Wide Web, met uitbreiding van interactieve en collaboratieve mogelijkheden. In plaats van het downloaden van documenten, worden er XML bomen bijgehouden, zowel door de client en door de server. Het bindingweefsel van XWeb is het *XWeb Transport Protocol (XTP)*. Dit protocol is gebaseerd op *HTTP* en wordt uitgebreid met enkele methods om de XML tree op te vragen ,te queryen, en aan te passen (zowel lokaal als globaal).

Een groot voordeel van XWeb is dat zowel services als clienten onafhankelijk van elkaar geïmplementeerd kunnen worden, met de voorwaarde dat XTP ondersteund wordt door de implementatie. Ook zijn de client implementaties onafhankelijk van elkaar, en ook onafhankelijk van de manier waarop de interactie plaatsvindt. Gebruikers kunnen op deze manier n client kiezen, en met deze client kan er dan met alle XWeb services geinterageerd worden. Met behulp van *XViews* kan er een platformonafhankelijke interface gespecificeerd worden. Een XView is een algemene XML beschrijving van een interactie, dat de te gebruiken interactietechniek niet specificeert. Een XView specificatie bestaat uit widgets of *interactors*. Deze interactors bepalen de mogelijke types van waardes dat een bepaald data item kan bevatten. Voor meer informatie over XTP en over interactors zie [22].

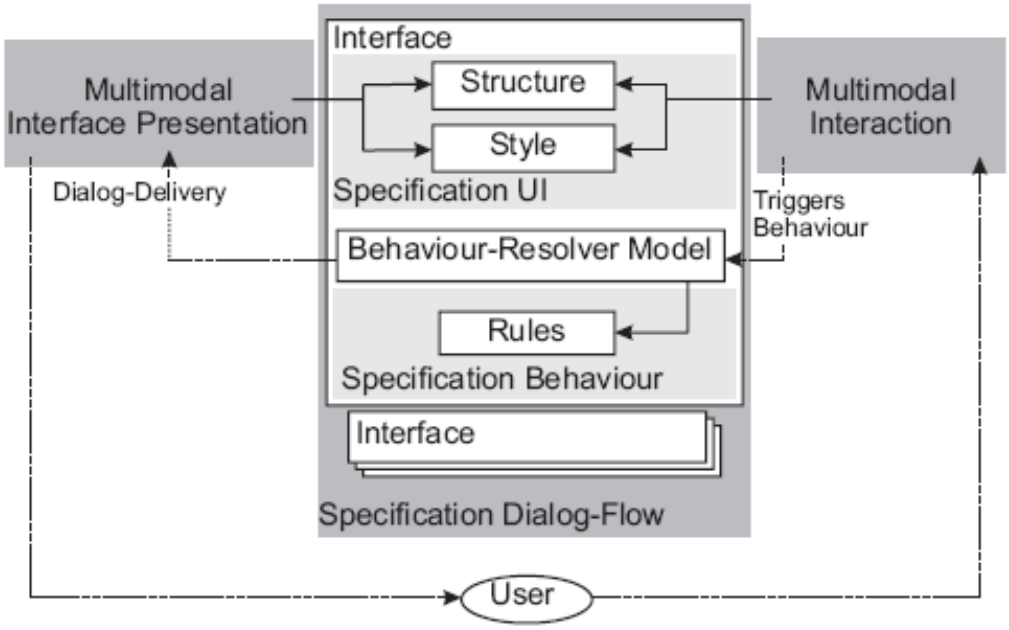


Figuur 4.4: XWeb architectuur.

4.2.2 MIPIM

In [23] wordt een nieuw dialoogmodel voorgesteld voor de ondersteuning van multimodale en multi-device interactie. Het model is vergelijkbaar met het UIML model, maar de specificatie van het dialog behaviour wordt verschillend aangepakt. In UIML is het control model event-gebaseerd, terwijl er in dit model de UI en de systeem status gecaptured wordt. Er wordt ook hier gebruik gemaakt van modaliteitonafhankelijke of generieke widgets. Voor meer informatie over generieke widgets zie [24] Voor het beschrijven van de modellen wordt gebruik gemaakt van de *Dialog and Interface Specification Language (DISL)*. DISL is een uitgebreide subset van UIML. De aanpassingen bestonden vooral uit de beschrijving van generieke widgets en de verbeteringen van de behaviour sectie met de concepten van het Behaviour Resolver Model.

Multimodal Interface Presentation and Interaction Model(MIPIM) bestaat uit 3 componenten: interactie, dialoog flow en interactie. Zie figuur 4.5. User interactie wordt ontvangen door het *Multimodal Interaction component*(zie figuur 4.6(a)). Dit component ontvang input in verschillende modaliteitenen activeert de *behaviour resolver*. Deze op zijn beurt start met het genereren van de resulterende UI die zal worden voorgesteld door het *Multimodal Interface Presentation component*(zie figuur 4.6(b)), voor de geactiveerde modaliteiten. De multimodale dialoog flow is een cyclisch proces dat wordt voorgesteld door de stippellijnen op figuur 4.5. Het Behaviour Resolver Model(zie figuur 4.6(c)) zorgt voor de evaluatie van interaction events, voor het uitvoeren van transitiesen voor het refreshen van de structure en style van de dialoog. De basis elementen van dit model zijn de content elementen, die de volledige dialoog status at runtime voorstellen. Een dialoog wordt aangepast door het evalueren en aanpassen van deze elementen.

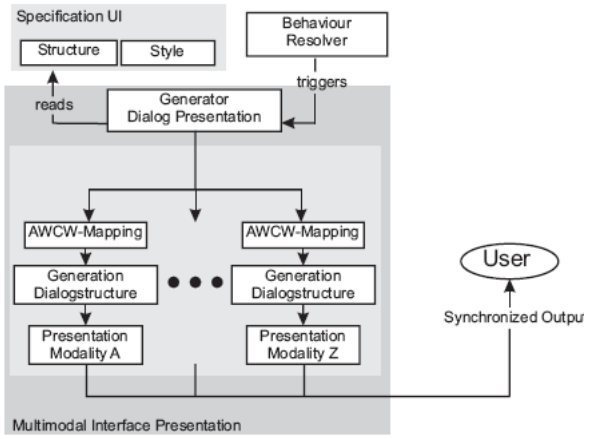


Figuur 4.5: Structuur van het MIPIM Dialoog model.

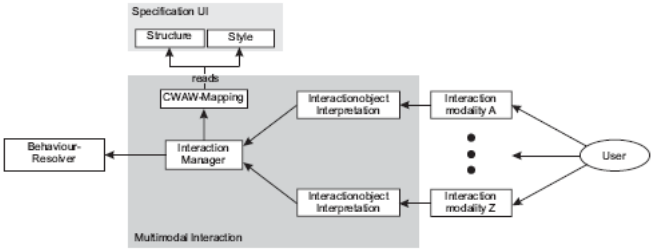
4.3 Modelgebaseerde aanpak

4.3.1 Migrating User Interfaces

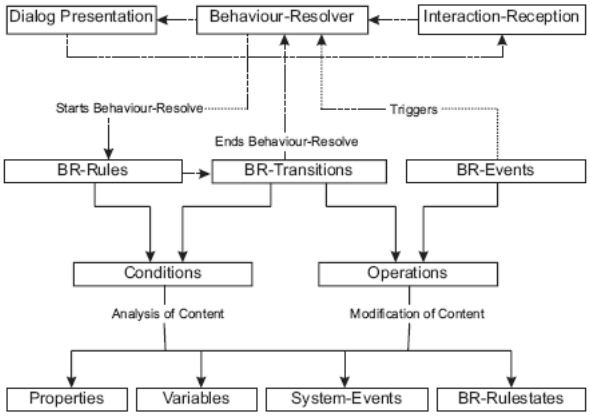
In [18] en [19] ligt de nadruk op het migreren van een multimodale UI van het ene apparaat naar het andere apparaat. *Migratory interfaces* zijn interfaces die het de gebruiker toelaten om vrijuit van apparaat te veranderen, en de taak vervolledigen vanaf het punt waarin ze zich bij het vorige apparaat bevonden. Er wordt gebruik gemaakt van apparaten die gebruik maken van spraak, directe manipulatie en een combinatie van beide voor de interactie met de applicaties. Er wordt gebruik gemaakt van een conceptuele beschrijving van de UI, die worden gerepresenteerd door een apparaatonafhankele XML-gebaseerde taal. De activiteiten die de gebruikers met een bepaalde UI willen uitvoeren worden vorgesteld door *ConcurTask-Trees [20]*.Een *Migration Server* houdt ten alle tijde de geregistreerde apparaten en extra informatie over deze apparaten bij. Wanneer de migration server een *migration request* ontvangt, wordt de informatie over de status van de UI van het source apparaat opgevraagd. Een versie van de UI wordt op het doelapparaat aangemaakt, rekening houdend met de status van deze UI. De transformaties worden schematisch weergegeven in figuur 4.7.



(a) Presentation Component

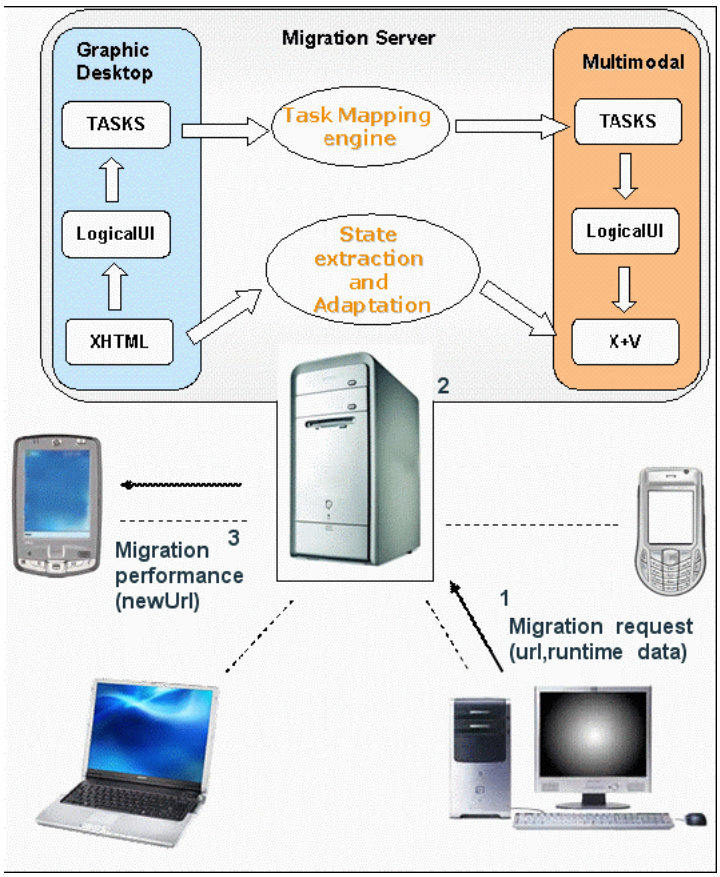


(b) Interaction Component



(c) Behaviour Resolver Model

Figuur 4.6: Verschillende onderdelen van het MIPIM model.



Figuur 4.7: Migration process dat wordt beschreven in [19]

4.4 Vergelijking met de in deze thesis gebruikte methode

De methode beschreven in dit werk maakt gebruik van UIML als HLUIDL. In tegenstelling tot enkele van de gerelateerde werken worden er geen aanpassingen of uitbreidingen op UIML toegepast. Er wordt gebruik gemaakt van de uitgebreide architectuur van de Uiml.Net Renderer. De architectuur van Uiml.Net wordt beschreven in sectie 3.4.1, de uitbreiding van de architectuur in hoofdstuk 5.

Een groot aantal van de gerelateerde werken maakt gebruik van generieke widgets om vervolgens naar verschillende modaliteiten en apparaten te kunnen mappen. Er wordt gebruik gemaakt van één modaliteit- en apparaatonafhankelijke beschrijving van de UI, die naar elk platform of elke modaliteit gemapt kan worden. Bij de methode in dit werk zijn de gedefinieerde widgets afhankelijk van de modaliteit. De interactie wordt beschreven door modaliteitafhankelijke widgets, dus bij het creëren van een widget staat al vast of er door middel van spraak of directe manipulatie geïnterageerd wordt met het systeem.

De widgets die in dit werk worden beschreven, zijn specifiek voor XHTML+Voice (zie 3.2). De verschillen tussen de GTK# vocabulary en het grafische deel van de X+V vocabulary zijn geminimaliseerd. XHTML+Voice is een markup taal, die door een multimodale browser gerendered kan worden. Een voorbeeld van een multimodale webbrowser is *Opera*[5], die beschikbaar is op vele mobiele platformen . Hierdoor dient er geen rekening gehouden te worden met platformspecifieke details.

[5]Opera Browser http://www.opera.com/

Deel III

Ontwikkeling

Hoofdstuk 5

Uitbreiding Uiml.Net architectuur

5.1 Uitbreiding architectuur

De renderer wordt aangepast in dit werk zodat er een *XML document* wordt gegenereerd , dat dan later getoond kan worden in een browser. Er moet dus een mapping gebeuren van een *UIML Document* naar een XML document. In plaats van widgets te creeëren, wordt er een *tag* of *attribuut* van een tag aangemaakt in een boomstructuur van een XML document. Dit document stelt vervolgens de *UI* voor, gespecificeerd in een bepaalde markup-taal zoals bijvoorbeeld *HTML* of *VoiceXml*. Het doel van de thesis is om vanuit een UIML document een *XHTML+Voice (X+V)* document te genereren. X+V is een XML-gebaseerde markup-taal voor het beschrijven van web applicaties die gebruik maken van zowel spraak als grafische interactie. Voor meer informatie over X+V, zie 3.2. In dit hoofdstuk wordt beschreven hoe de UIML.Net renderer architectuur aangepast wordt, zodat er een UI via reflectie en via het genereren van een XML document kan worden gerendered. De oorspronkelijke architectuur van de UIML.Net renderer wordt beschreven in 3.4.1.

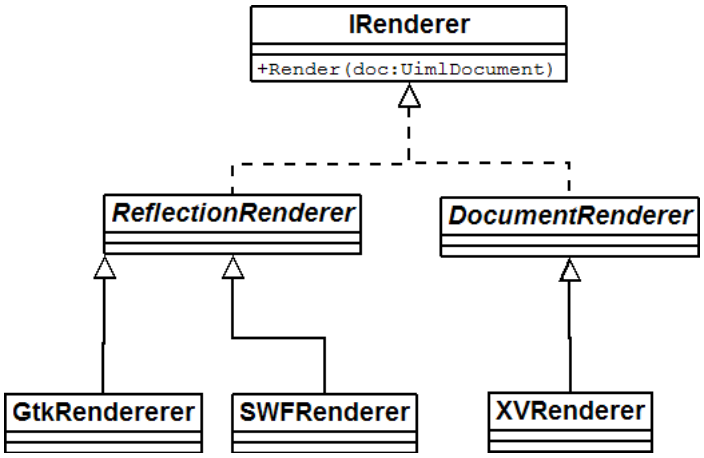
De rendering kern wordt opgesplitst in 2 verschillende delen.

- *ReflectionRenderer*: bevat de rendering mechanismes uitgelegd in hoofdstuk 3.4.1.
- *DocumentRenderer*: bevat de mechanismes nodig om een XML-document te genereren vanuit een UIML-document.

Beide Renderer klassen implementeren een algemene *IRenderer* interface. Op deze manier wordt het renderen van een XML-document transparent geïntegreerd in Uiml.Net. De Uiml.Net renderer kiest afhankelijk van de vocabularynaam de juiste Renderer. Indien dit de DocumentRenderer is, wordt er een XML document aangemaakt dat getoond wordt in een browser.

5.2 DocumentRenderer

De DocumentRenderer maakt een XML-document aan. Voor elke *part* tag, wordt een overeenkomstige tag aangemaakt in de XML-boom, volgens de mapping gespecificeerd in de vocabulary. De *style* properties worden voor elk van de parts opgevraagd, gemapt naar concrete



Figuur 5.1: Opslitsing van de Renderer

properties van de tag, en toegepast op de tags in het nieuwe XML-document. In de vocabulary staat gespecificeerd hoe de style properties geplaatst moeten worden in de boomstructuur van het XML-document. De property kan als child tag, als attribute of als content van een bepaalde tag geplaatst worden in het XML-document. Het type van de mapping wordt gespecificeerd in de vocabulary. De werking kan dus als volgt omschreven worden:

1. Gegeven een part, creeër een xml-element met de juiste naam.
2. Vraag alle properties van dit part op die door de gebruiker gedefinieerd zijn, en voeg deze op de juiste manier toe aan het gecreëerde XML element.
3. Render alle subparts van dit part.

5.2.1 Creëer een element voor een part

Standaard wordt er een xml-element gecreëerd aan de hand van de mapping in de vocabulary. Beschouw een klasse genaamd Knop die in de de vocabulary mapt op button. Dit ziet er als volgt uit in de vocabulary:

```
<d-class id="Knop" maps-type="tag" maps-to="button"/>
```

Wanneer in het UIML document een part met als klasse Knop wordt aangeroepen:

```
<part class="Knop" id="knop"/>
```

Dit geeft het volgende resultaat in het resulterende XML document:

```
<button>
```

5.2.2 Render property

De concrete propertynaam kan opgevraagd worden uit de vocabulary, met behulp van de propertynaam en de klasse waartoe de property behoort. Als voorbeeld beschouw een klasse

genaamd Knop (mapt in de vocabulary op button) met als property label (mapt in de vocabulary op value) met als waarde *Knop*. Dit ziet er als volgt uit in het interface gedeelte van het UIML document.

```
...
<part class="Knop" id="knop">
...
<property part-name="knop" name="label">Knop</property>
```

Een property kan op 3 verschillende manieren gerendered worden, afhankelijk van het type van de mapping [1]:

- attribute: Het huidige XMLElement krijgt een attribuut met als naam de concrete propertynaam, met als waarde de *waarde* van de huidige property.

```
...
<button value="Knop"/>
...
```

- text: De waarde van het huidige XMLElement wordt ingesteld op de *waarde* van de huidige property.

```
...
<button>Knop</button>
...
```

- tag: Het huidige XML element krijgt een nieuwe XML element als kind. De naam van het XML element is de concrete propertynaam, de waarde van het XML element wordt ingesteld op de waarde van de huidige property.

```
...
<button>
  <value>Knop</value>
</button>
...
```

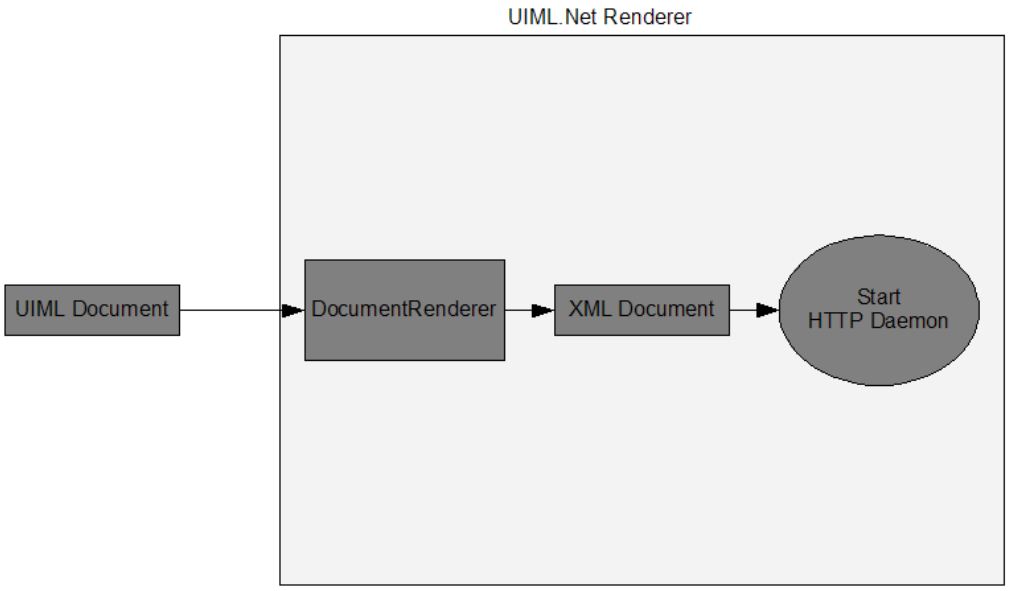
5.2.3 Render alle subparts

Alle subparts worden gerendered en toegevoegd als kinderen van het XML element.

5.2.4 HTTP Daemon

Wanneer het XML-document volledig opgebouwd is, wordt er een *HTTP daemon* opgestart. Wanneer de daemon een *HTTP Request* ontvangt (vanuit een browser), stuurt de daemon het gegenereerde XML-document als response terug. De browser toont vervolgens de UI die gespecificeerd staat in het document.In figuur 5.2 en figuur 5.3 wordt dit schematisch getoond. Het opstarten van een HTTP daemon is essentieel voor het terugkoppelen van

[1] afhankelijk van maps-type attribuut van de property in de vocabulary

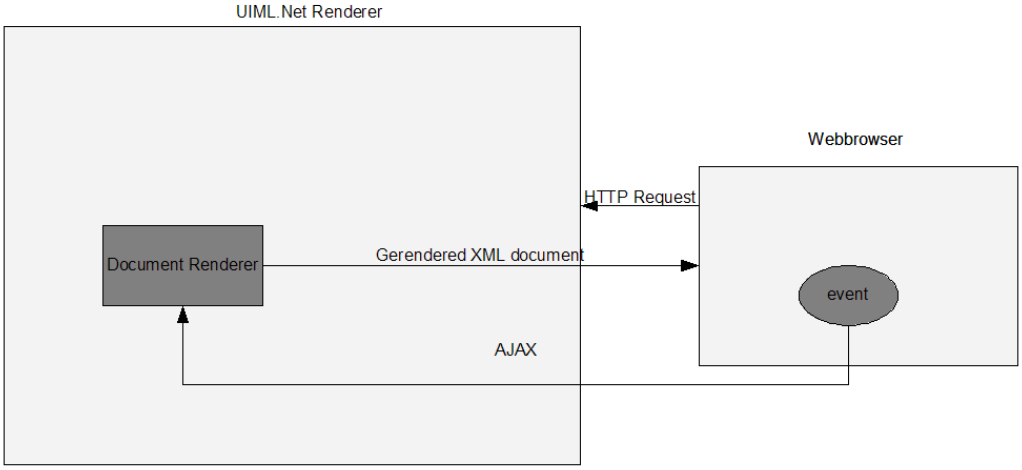


Figuur 5.2: werking van de DocumentRenderer

events (uitvoeren van behavior) naar de renderer. Wanneer de gebruiker een bepaalde actie uitvoert in de genereerde UI, wordt er via *Java-script* teruggekoppeld naar de renderer. De renderer bevat de mogelijke events van de UI, en de acties die hieruit moeten volgen. De veranderingen in de user interface worden toegepast, het XML document wordt aangepast, en de aanpassingen worden teruggestuurd naar de browser, waar vervolgens de UI wordt getoond.

In figuur 5.2 verwerkt de Uiml.Net renderer een gegeven UIML document, maakt een XML document aan, en start een HTTP daemon op. In figuur 5.3 wordt een browser opgestart die een HTTP request verzendt naar de opgestarte HTTP daemon. Deze stuurt het gegeneerde XML document terug naar de browser. De terugkoppeling van events is mogelijk door middel van AJAX. Meer informatie over events, en hoe de communicatie tussen browser en server gebeurt, bevindt zich in hoofdstuk 7.3.

Opmerking: Er is ook een andere aanpak mogelijk: er wordt eerst een HTTP daemon opgestart. Een client stuurt vervolgens een bepaald UIML document naar de daemon. Wanneer de daemon een UIML document ontvangt, verwerkt de daemon dit document, en renderered het XML document. Dit gerenderde document wordt vervolgens teruggestuurd naar de client. Het principe blijft onveranderd, alleen het doorgeven van het UIML document en het opstarten van de HTTP daemon gebeurt in een andere volgorde.



Figuur 5.3: werking van de DocumentRenderer

Hoofdstuk 6

XHTML + Voice Vocabulary

De vocabulary kan opgesplitst worden in 2 grote delen, het XHTML deel en het VoiceXML deel. Beide delen bestonden al afzonderlijk, geschreven door Harmonia. Beide vocabularies zijn echter weinig generiek, en er is kennis nodig van beide markup talen. De klassen gedefinieerd in deze vocabulary zijn de tags die in de markup taal bestaan, de properties van elke klasse bestaan uit de attributen van de respectievelijke tag.

Het XHTML gedeelte van de Uiml.Net X+V vocabulary is zo generiek mogelijk gemaakt. De gebruiker heeft slechts zeer beperkte kennis van XHTML nodig om een interface te specificeren. Meer informatie over dit deel van de vocabulary bevindt zich in sectie 6.1.

Het VoiceXML gedeelte van de vocabulary is eerder gelijkend op die van Harmonia. De sterke nesting van tags in VoiceXML zorgde hiervoor. Idealiter zou er een klasse VoiceIn bestaan voor gesproken input, en een klasse VoiceOut voor gesproken output. Dit zou het gemakkelijk maken voor de gebruiker om gesproken dialogen te definiëren in een UIML document. Meer informatie over dit deel van de vocabulary bevindt zich in sectie 6.2.

6.1 XHTML

De renderer maakt automatisch een html tag aan, met bijhorende body en head tags. Alle XHTML elementen worden als kind toegevoegd van het juiste element. Het *id* attribuut van de html tags wordt gelijkgesteld aan de identifier van het part element. Hier volgt een overzicht van alle klassen in de vocabulary die naar een XHTML tag mappen.

6.1.1 Gebruik van vocabulary

Html

Beschrijving: De Html klasse markeert het begin en einde van een HTML-document en geeft aan dat het document HTML-code bevat.

Properties: geen.

Gebruik: Part in UIML structure element:

Listing 6.1: Html klasse in uiml document

```
<part class="Html">
    ...
    ...
</part>
```

Resultaat in xml document:

Listing 6.2: Html klasse in gerendered xml document

```
<html>
    ...
    ...
</html>
```

Button

Beschrijving: Stelt een standaard knop voor in html,wordt automatisch gemapt door de renderer op tag *input* met als type *button*.

Properties:

1. label: opschrift van de button.
2. disabled: Indien *true* kan de de gebruiker niet op de knop klikken.

Gebruik: Part in UIML structure element:

Listing 6.3: Button klasse in uiml document

```
...
<part class="Button" id="button"/>
...
<property part-name="button" name="label">Button</property>
...
```

Resultaat in xml document:

Listing 6.4: Button klasse in gerendered xml document

```
...
<input type="button" id="button" value="Button"/>
...
```

Label

Beschrijving: Een simpel label.

Properties:

1. text:De text van het label.

Gebruik: Part in UIML structure element:

Listing 6.5: Label klasse in uiml document

```
...
<part class="Label" id="label"/>
...
<property part-name="label" name="text">label</property>
...
```

Resultaat in xml document:

Listing 6.6: Label klasse in gerendered xml document

```
...
<label id="label">label</label>
...
```

Entry

Beschrijving: De klasse definieert een veld waarin de gebruiker een enkel woord of een enkele regel tekst kan invoeren. Dit wordt automatisch gemapt door de renderer op tag *input* met als type *text*.

```
<input type="text"/>
```

Properties:

1. text: De text van het invulveld.
2. size: De grootte van het invulveld.
3. maxlength: De maximale lengte van de inhoud van het invulveld.
4. disabled: Indien *true* kan dit veld niet gewijzigd worden door de gebruiker.

Gebruik: Part in UIML structure element:

Listing 6.7: Entry klasse in uiml document

```
...
<part class="Entry" id="entry"/>
...
<property part-name="entry" name="text">Entry</property>
<property part-name="entry" name="size">5</property>
<property part-name="entry" name="maxlength">5</property>
<property part-name="entry" name="disabled">true</property>
...
```

Resultaat in xml document:

Listing 6.8: Entry klasse in gerendered xml document

```
...
<input type="text" id="entry" value="Entry" maxlength="5" size="5"
        disabled="true"/>
...
```

Text

Beschrijving: De Text klasse definieert een rechthoekig vak in een formulier, waarin de gebruiker over meerdere regels tekst kan invoeren.

Properties:

1. text: de inhoud van het tekstvak kan met dit property bepaald worden.
2. disabled: Indien *true* kan dit veld niet gewijzigd worden door de gebruiker.
3. cols: de breedte van het tekstvak kan met dit property bepaald worden, uitgedrukt in het aantal kolommen(karakters) tekst.
4. rows: de hoogte van het tekstvak kan met dit property bepaald worden, uitgedrukt in het aantal rijen(regels) tekst.

Gebruik: Part in UIML structure element:

Listing 6.9: TextArea klasse in uiml document

```
...
<part class="Text" id="text"/>
...
<property part-name="text" name="text">text area</property>
...
```

Resultaat in xml document:

Listing 6.10: TextArea klasse in gerendered xml document

```
...
<textarea id="text">textarea</textarea>
...
```

Image

Beschrijving: Stelt een afbeelding voor in html.

Properties:

1. file:De locatie van de afbeelding.
2. width:De breedte van de afbeelding.
3. height:De hoogte van de afbeelding.
4. alt:Een beschrijving van de afbeelding.

Gebruik: Part in UIML structure element:

Listing 6.11: Image klasse in uiml document

```
...
<part class="Image" id="image"/>
...
<property part-name="image" name="file">
        http://www.uhasselt.be/images/logoLUC_web.gif
</property>
<property part-name="image" name="alt">
  Logo van Universiteit Hasselt
</property>
...
```

Resultaat in xml document:

Listing 6.12: Image klasse in gerendered xml document

```
...
<img src="http://www.uhasselt.be/images/logoLUC_web.gif"
 alt="Logo van Universiteit Hasselt/>
...
```

Checkbutton

Beschrijving: Stelt een checkbutton voor in html, wordt automatisch gemapt door de renderer op tag *input* met als type *checkbox*.

Properties:

1. label: Het bijhorende label.
2. checked: De status van de checkbutton.
3. disabled: Indien *true* kan dit veld niet gewijzigd worden door de gebruiker.

Gebruik: Part in UIML structure element:

Listing 6.13: Checkbox klasse in uiml document

```
...
<part class="Checkbutton" id="checkbutton"/>
...
<property part-name="checkbutton" name="label">Checkbutton</property>
<property part-name="checkbutton" name="checked">true</property>
...
```

Resultaat in xml document:

Listing 6.14: Checkbox klasse in gerendered xml document

```
...
<input type="checkbox" id="checkbutton" />
<label>Checkbutton</label>
...
```

RadioButton

Beschrijving: Stelt een radiobutton voor. Met behulp van een aantal radiobuttons kan er een selectie gemaakt worden tussen enkele opties. Wordt automatisch gemapt door de renderer op tag *input* met als type *radio*.

Properties:

1. label: Het bijhorende label.
2. name: De naam van de radiobutton. Wanneer verschillende radiobuttons dezelfde naam hebben, kan er slechts ëën van deze aangevinkt (*checked*) zijn.
3. checked:De status van de radiobutton.
4. disabled: Indien *true* kan dit veld niet gewijzigd worden door de gebruiker.

Gebruik: Part in UIML structure element:

Listing 6.15: RadioSelect klasse in uiml document

```
...
<part class="RadioButton" id="radiobutton1"/>
<part class="RadioButton" id="radiobutton2"/>
...
<property part-name="radiobutton1" name="label">option_1</property>
<property part-name="radiobutton2" name="label">option_2</property>
<property part-name="radiobutton1" name="name">option</property>
<property part-name="radiobutton2" name="name">option</property>
...
```

Resultaat in xml document:

Listing 6.16: RadioSelect klasse in gerendered xml document

```
<input id="radiobutton1" type="radio" name="option" />
<label>option_1</label>
<input id="radiobutton2" type="radio" name="option" />
<label>option_2</label>
```

Combo

Beschrijving: Een combobox die het mogelijk maakt een selectie uit een aantal elementen te maken.

Properties:

1. content: De verschillende opties van de combobox. Wordt in een constant omgeving gedefinieerd.
2. entry: De huidige actieve optie van de combobox.
3. disabled: Indien *true* kan dit veld niet gewijzigd worden door de gebruiker.

Gebruik: Part in UIML structure element:

Listing 6.17: Combo klasse in uiml document

```
...
<part class="Combo" id="combo"/>
...
<property part-name="combo" name="content">
    <constant model="list">
        <constant value="option1"/>
        <constant value="option2"/>
        <constant value="option3"/>
    </constant>
</property>
...
```

Resultaat in xml document:

Listing 6.18: Combo klasse in gerendered xml document

```
<select id="combo">
    <option>option1</option>
    <option>option2</option>
    <option>option3</option>
</select>
```

List

Beschrijving: De List klasse definieert een lijst van items.

Properties:

1. content: De verschillende items van de lijst. Wordt in een constant omgeving gedefinieerd (zie gebruik).

Gebruik: Part in UIML structure element:

Listing 6.19: List klasse in uiml document

```
...
<part class="List" id="list"/>
...
<property part-name="list" name="content">
    <constant model="list">
        <constant value="listitem1"/>
        <constant value="listitem2"/>
        <constant value="listitem3"/>
    </constant>
</property>
...
```

Resultaat in xml document:

Listing 6.20: List klasse in gerendered xml document

```
<ul id="list">
    <li>listitem1</li>
    <li>listitem2</li>
    <li>listitem3</li>
</ul>
```

VBox en HBox

Beschrijving: Deze klasses zorgen voor de layout van de html componenten. Er wordt een *table* tag aangemaakt. Elk subpart van een hbox is deel van een table data element (td), elk subpart van een vbox is een deel van table row (tr). Interface in UIML (enkel structure):

Properties: geen.

Gebruik: Part in UIML structure element:

Listing 6.21: Vbox en Hbox klasse in uiml document

```
<part class="VBox">
    <part class="HBox">
        <part class="Label" id="fromlabel"/>
        <part class="Entry" id="from"/>
    </part>
    <part class="HBox">
        <part class="Label" id="tolabel"/>
```

```
        <part class="Entry" id="to"/>
    </part>
</part>
```

Resultaat in xml document:

Listing 6.22: VBox en HBox klasse in gerendered xml document

```
<html>
  <head/>
  <body>
    <table>
      <tr>
        <table>
          <td>
            <label id="fromlabel">/>
          </td>
          <td>
            <input type="text" id="from" />
          </td>
        </table>
      </tr>
      <tr>
        <table>
          <td>
            <label id="tolabel"/>
          </td>
          <td>
            <input type="text" id="tolabel"/>
          </td>
        </table>
      </tr>
    </table>
  </body>
</html>
```

Opmerking: Een *tr* element in HTML moet in principe minimaal één *td* element bevatten. In deze implementatie is het echter mogelijk dat dit niet het geval is. De tabel wordt echter wel op de juiste manier getoond in de meeste browsers.

Title

Beschrijving: De titel klasse legt een titel voor het document vastd. Deze titel wordt niet weergegeven in het documentvenster, maar in de titelbalk van het venster. Dit element wordt automatisch in het *head* element van het document geplaatst.

Properties:

1. content: Bevat de naam van de titel.

Gebruik: Part in UIML structure element:

Listing 6.23: Title klasse in uiml document

```
    ...
<part class="Title" id="title"/>
    ...
<property part-name="title" name="content">Title of application
</property>
    ...
```

Resultaat in xml document:

Listing 6.24: Title klasse in gerendered xml document

```
...
<head>
    ...
    <title id="titel">Title of application</title>
    ...
</head>
...
```

6.1.2 Uitgebreid voorbeeld

In dit voorbeeld worden alle klassen gebruikt: Zie listing 6.25.

Listing 6.25: XhtmlWidgets.uiml

```
<!--<?xml version="1.0" encoding="utf-8" ?> -->
<uiml>
  <interface>
    <structure>
      <part id="root" class="Html">
        <part class="Title" id="title"/>
        <part class="Image" id="image"/>
        <part class="HBox">
          <part class="VBox">
            <part class="List" id="list"/>
            <part class="Entry" id="entry"/>
            <part class="Button" id="button"/>
            <part class="CheckButton" id="checkbox"/>
          </part>
          <part class="VBox">
            <part class="Text" id="text"/>
            <part class="Combo" id="combobox"/>
            <part class="Label" id="label"/>
            <part class="RadioButton" id="radiobutton1"/>
                    <part class="RadioButton" id="radiobutton2"/>
          </part>
        </part>
      </part>
    </structure>
```

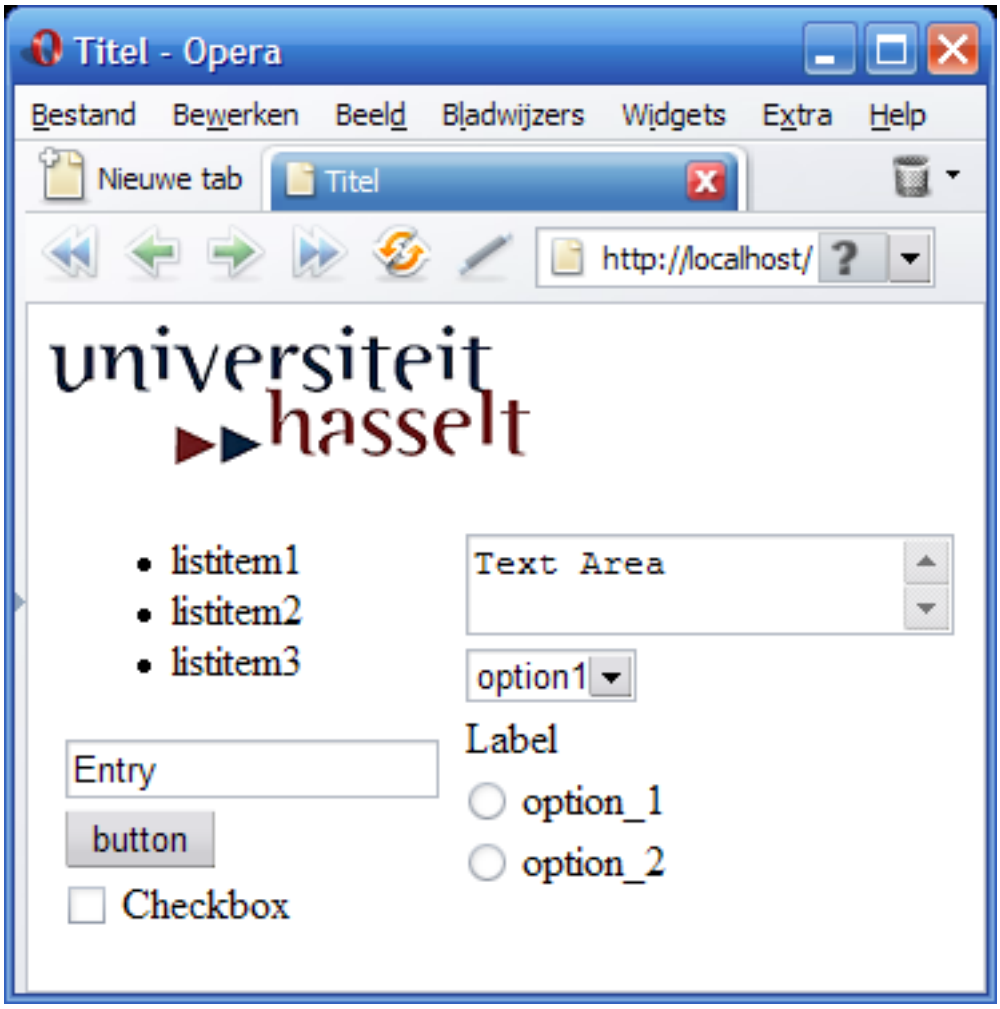
```
    <style>
      <property part-name="title" name="text">Title</property>
      <property part-name="list" name="content">
        <constant model="list">
          <constant value="listitem1"/>
          <constant value="listitem2"/>
          <constant value="listitem3"/>
        </constant>
      </property>
      <property part-name="entry" name="text">Entry</property>
      <property part-name="button" name="label">button</property>
      <property part-name="image" name="file">
        http://www.uhasselt.be/images/logoLUC_web.gif
      </property>
      <property part-name="text" name="text">Text Area
      </property>
      <property part-name="combobox" name="content">
        <constant model="list">
          <constant value="option1"/>
          <constant value="option2"/>
          <constant value="option3"/>
        </constant>
      </property>
      <property part-name="label" name="text">Label</property>
      <property part-name="checkbox" name="label">Checkbox</property>
      <property part-name="radiobutton1" name="label">option_1</property>
      <property part-name="radiobutton2" name="label">option_2</property>
      <property part-name="radiobutton1" name="name">option</property>
      <property part-name="radiobutton2" name="name">option</property>
    </style>
    <behavior>
    </behavior>
  </interface>
  <peers>
    <presentation base="XV.uiml"/>
  </peers>
</uiml>
```

Listing 6.26: Xml document gerendered vanuit listing 6.25

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:vxml="http://www.w3.org/2001/vxml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:xv="http://www.voicexml.org/2002/xhtml+voice">
  <head>
    <script type="text/javascript"
      src="http://users.pandora.be/jonbeton/Thesis/js/script.js"/>
    <title id="title">Title</title>
  </head>
  <body>
    <img id="image" src="http://www.uhasselt.be/images/logoLUC_web.gif"/>
    <table border="1">
```

```
      <td>
        <table border="1">
          <tr>
            <ul id="list">
              <li>listitem1</li>
              <li>listitem2</li>
              <li>listitem3</li>
            </ul>
          </tr>
          <tr>
            <input id="entry" type="text" value="Entry" />
          </tr>
          <tr>
            <input id="button" type="button" value="button" />
          </tr>
          <tr>
            <input id="checkbox" type="checkbox" />
            <label>Checkbox</label>
          </tr>
        </table>
      </td>
      <td>
        <table border="1">
          <tr>
            <textarea id="text">Text Area
      </textarea>
          </tr>
          <tr>
            <select id="combobox">
              <option>option1</option>
              <option>option2</option>
              <option>option3</option>
            </select>
          </tr>
          <tr>
            <label id="label">Label</label>
          </tr>
          <tr>
            <input id="radiobutton1" type="radio" name="option" />
            <label>option_1</label>
          </tr>
          <tr>
            <input id="radiobutton2" type="radio" name="option" />
            <label>option_2</label>
          </tr>
        </table>
      </td>
    </table>
  </body>
</html>
```

Screenshot van de resulterende interface: zie figuur 6.1.



Figuur 6.1: Resulterende interface

6.2 VoiceXML

Het VoiceXML gedeelte van de X+V vocabulary maakt het mogelijk om gesproken input en output te definiëren. Bij het ontwikkelen van dit deel van de vocabulary kwamen enkele problemen aan het licht. In een VoiceXML form kunnen verschillende events voorkomen, wanneer de gebruiker een bepaald VoiceXML Field in moet vullen. Deze events worden in VoiceXML voorgesteld als een XML element met als naam de naam van het event. In dit XML element kan er nog XML code staan, om de gepaste actie te specificeren. Dit zorgt er echter voor dat dergelijke events en de daarop volgende actie niet in het *behavior* gedeelte van de *interface* van een UIML document kan gespecificeerd worden. Dit heeft als resultaat dat vier klassen gedefiniëerd in deze sectie eigelijk events zijn, namelijk *Filled*, *Nomatch*, *Noinput* en *Help*. In listing 6.27 heeft het incorrect invullen van een veld tot resultaat dat er een audio bestand wordt afgespeeld, en er vervolgens een boodschap wordt gesproken.

Listing 6.27: nomatch event opgevuld met XML code

```
<vxml:form id="form1">
```

```
  <vxml:field name="field1">
    <vxml:option>1</vxml:option>
    <vxml:option>2</vxml:option>
    <vxml:option>3</vxml:option>
    <vxml:option>4</vxml:option>
    <vxml:option>5</vxml:option>
    <vxml:nomatch>
      <vxml:audio src="wrong.wav"/>
      Please say a number between 1 and 5
    </vxml:nomatch>
  </vxml:field>
</vxml:form>
```

Enkel een VoiceXML form heeft een id attribuut. Alle andere VoiceXML elementen mogen dit niet hebben, anders wordt het X+V document niet gerendered. Dit zorgt echter voor problemen wanneer er vanuit JavaScript de waarde van een bepaald VoiceXML veld opgevraagd moet worden. De oplossing hiervoor wordt beschreven in sectie 7.3.4.

6.2.1 Gebruik van vocabulary

VoiceForm

Beschrijving: De VoiceForm klasse markeert het begin en het einde van een formulier dat door middel van VoiceXML wordt afgehandeld. Dit form krijgt automatisch een id toegewezen, namelijk de identifier van het part. Een VoiceForm wordt toegevoegd aan het *head* element van het xml-document.

Properties:

1. executeon: Linkt een XHTML event aan de activering van dit VoiceXML form. De waarde van dit veld bestaat uit *partid,eventClass*. Zie sectie 7.3.5 voor meer uitleg.

Gebruik: Part in UIML structure element:

Listing 6.28: VoiceForm klasse in uiml document

```
...
<part class="VoiceForm" id="vform"/>
...
<part class="Button" id="button"/>
...
<property part-name="vform" name="executeon">
 button,ButtonPressed
</property>
...
```

Resultaat in xml document:

Listing 6.29: VoiceForm klasse in gerendered xml document

```
...
<head>
   ...
  <vxml:form id="vform"/>
   ...
</head>
...
<input id="button" type="button" onclick="ExecuteVoice('vform')"/>
<input id="button" type="button" onclick="ExecuteVoice('vform')"/>
...
```

Block

Beschrijving: Deze klasse zorgt voor simpele voice output.

Properties:

1. content: De tekst die als output dient.

Gebruik: Part in UIML structure element:

Listing 6.30: Block klasse in uiml document

```
...
<part class="Block" id="block"/>
...
<property part-name="block" name="content">Voice Output.</property>
...
```

Resultaat in xml document:

Listing 6.31: Block klasse in gerendered xml document

```
...
<vxml:block>Voice Output.</vxml:block>
...
```

Field

Beschrijving: Input klasse die de gebruiker vraagt een bepaald waarde in te geven die aan een specifieke grammar voldoet. De mogelijkheden van input kunnen op 3 verschillende manieren gedefinieerd worden: door middel van een externe *grammar*, d.m.v. een voorgedefineerd *type* of door middel van een lijst van *options*.

Properties:

1. options: De mogelijkheden waaruit de gebruiker kan kiezen. Wordt in een constante omgeving gedefinieerd. Is een alternatief voor een grammar.
2. type: Specificeert platform gedefinieerde ingebouwde grammars. De mogelijkheden hier zijn:
   (a) boolean
   (b) currency
   (c) date
   (d) digits
   (e) number
   (f) phone
   (g) time
3. grammar: Specificeert de lokatie van een externe grammatica.

Gebruik: Part in UIML structure element:

Listing 6.32: Field klasse in uiml document

```
...
<part class="Field" id="field"/>
...
<property part-name="field" name="options">
        <constant model="list">
                <constant value="option 1"></constant>
                <constant value="option 2"></constant>
                <constant value="option 3"></constant>
        </constant>
</property>
...
```

Resultaat in xml document:

Listing 6.33: Field klasse in gerendered xml document

```
...
<vxml:field name="field">
        <vxml:option>option 1</vxml:option>
        <vxml:option>option 2</vxml:option>
        <vxml:option>option 3</vxml:option>
</vxml:field>
...
```

Audio

Beschrijving: De Audio klasse maakt het mogelijk om een geluidsbestand af te spelen.

Properties:

1. src: De locatie van het af te spelen bestand.

Gebruik: Part in UIML structure element:

Listing 6.34: Audio klasse in uiml document

```
...
<part class="Audio" id="audio"/>
...
<property part-name="audio" name="src">
   http://members.lycos.nl/compukid/A-team5th.wav
</property>
...
```

Resultaat in xml document:

Listing 6.35: Audio klasse in gerendered xml document

```
...
<vxml:audio src="http://members.lycos.nl/compukid/A-team5th.wav"/>
...
```

Record

Beschrijving: Verzamelt een opname van de gebruiker, en slaat het resultaat op in de input variabele.

Properties:

1. maxtime: Bepaalt de maximum duur van de tijdsopname(default 10s).

Gebruik: Part in UIML structure element:

Listing 6.36: Record klasse in uiml document

```
...
<part class="Record" id="record"/>
...
<property part-name="record" name="maxtime">5s</property>
...
```

Resultaat in xml document:

Listing 6.37: Record klasse in gerendered xml document

```
...
<vxml:record name="record" maxtime="5s"/>
...
```

Prompt

Beschrijving:

Properties: De prompt klasse geeft synthesized text-to-speech inhoud aan de gebruiker. Wordt meestal gebruikt binnen een Field klasse. Deze klasse geeft aan wat de gebruiker in dient te vullen in een veld, en wordt binnen een bepaalde tijd herhaald indien er geen reactie is van de gebruiker.

1. bargein: Bepaalt of user input herkend dient te worden tijdens de prompt.
2. count: Minimum aantal keren dat de gebruiker het form item dat dit prompt bevat dient te bezoeken eer de prompt wordt uitgevoerd.
3. timeout: De tijd dat er gewacht wordt eer een *noinput* event wordt gethrowed.
4. content: De boodschap die uitgesproken wordt.

Gebruik: Part in UIML structure element:

Listing 6.38: Prompt klasse in uiml document

```
...
<part class="Prompt" id="prompt"/>
...
<property part-name="prompt" name="bargein">false</property>
<property part-name="prompt" name="count">1</property>
<property part-name="prompt" name="timeout">5s</property>
<property part-name="prompt" name="content">
        This is a prompt message!
</property>
...
```

Resultaat in xml document:

Listing 6.39: Prompt klasse in gerendered xml document

```
...
<vxml:prompt bargein="true" count="1" timeout="5s">
        This is a prompt message!
</vxml:prompt>
...
```

Filled

Beschrijving: Deze klasse bevat acties die uitgevoerd worden wanneer een Field (of Record) klasse ingevuld is.

Properties:

1. content: De boodschap die uitgesproken wordt.

Gebruik: Part in UIML structure element:

Listing 6.40: Filled klasse in uiml document

```
...
<part class="Filled" id="onfill">
        ...
</part>
...
<property part-name="onfill" name="content">
  Thank you!
</property>
...
```

Resultaat in xml document:

Listing 6.41: Filled klasse in gerendered xml document

```
...
<vxml:filled>
  Thank you!
        ...
</vxml:filled>
...
```

Nomatch

Beschrijving: Deze klasse bevat acties die uitgevoerd worden wanneer een veld incorrect is ingevuld.

Properties:

1. content: De boodschap die uitgesproken wordt.
2. count: Bepaalt het minimum aantal keer dat de no-match error dient voor te komen eer de no-match actie wordt uitgevoerd (default 1).

Gebruik: Part in UIML structure element:

Listing 6.42: Nomatch klasse in uiml document

```
...
<part class="Nomatch" id="nomatch">
        ...
</part>
...
<property part-name="nomatch" name="count">2</property>
...
```

Resultaat in xml document:

Listing 6.43: Nomatch klasse in gerendered xml document

```
...
<vxml:nomatch count="2">
        ...
</vxml:nomatch>
...
```

Noinput

Beschrijving: Deze klasse bevat acties die uitgevoerd worden wanneer een veld niet wordt ingevuld binnen de maximale tijd.

Properties:

1. content: De boodschap die uitgesproken wordt.
2. count: Bepaalt het minimum aantal keer dat de no-input error dient voor te komen eer de no-input actie wordt uitgevoerd (default 1).

Gebruik: Part in UIML structure element:

Listing 6.44: Noinput klasse in uiml document

```
...
<part class="Noinput" id="noinput">
        ...
</part>
...
<property part-name="noinput" name="count">3</property>
...
```

Resultaat in xml document:

Listing 6.45: Noinput klasse in gerendered xml document

```
...
<vxml:noinput count="3">
        ...
</vxml:noinput>
...
```

Help

Beschrijving: Deze klasse bevat acties die uitgevoerd worden wanneer de gebruiker om help vraagt.

Properties:

1. content: De boodschap die uitgesproken wordt.
2. count: Bepaalt het minimum aantal keer dat de help aanvraag dient voor te komen eer de help actie wordt uitgevoerd (default 1).

Gebruik: Part in UIML structure element:

Listing 6.46: Help klasse in uiml document

```
...
<part class="Help" id="help">
        ...
</part>
...
<property part-name="help" name="count">1</property>
...
```

Resultaat in xml document:

Listing 6.47: Help klasse in gerendered xml document

```
...
<vxml:help count="1">
        ...
</vxml:help>
...
```

6.2.2 Uitgebreid voorbeeld

Een eenvoudige user interface om boodschappen in te spreken wordt getoond in listing 6.48. Er wordt eerst aan de gebruiker gevraagd om een kamernummer (bvb van een hotel) op te geven, waarna de gebruiker een boodschap kan achterlaten. Het resulterende VoiceXML form wordt getoond in listing 6.49.

Listing 6.48: Message system

```
...
<part class="VoiceForm" id="form1">
  <part class="Block" id="block1"/>
  <part class="Field" id="field1">
    <part class="Prompt" id="fieldPrompt"/>
    <part class="Filled" id="onfill">
      <part class="Audio" id="succes"/>
    </part>
        <part class="Nomatch" id="onwrong">
          <part class="Audio" id="fail"/>
        </part>
        <part class="Noinput" id="onempty"/>
        <part class="Help" id="onhelp"/>
```

```
  </part>
  <part class="Block" id="block2"/>
  <part class="Record" id="record"/>
  <part class="Block" id="block3"/>
</part>
...
<property part-name="field1" name="options">
  <constant model="list">
    <constant value="1"/>
    <constant value="2"/>
    <constant value="3"/>
    <constant value="4"/>
    <constant value="5"/>
  </constant>
</property>
<property part-name="fieldPrompt" name="text">
 Please say a number between 1 and 5
</property>
<property part-name="fieldPrompt" name="timeout">5s</property>
<property part-name="onfill" name="text">Thank you</property>
<property part-name="onwrong" name="text">Incorrect input!
 Please say a number between 1 and 5</property>
<property part-name="onempty" name="text">No input!
 Please say a number between 1 and 5</property>
<property part-name="onhelp" name="text">
 You should say a number between 1 and 5</property>
<property part-name="succes" name="src">succes.wav</property>
<property part-name="fail" name="src">fail.wav</property>
<property part-name="block1" name="text">
 Welcome to the automatic messaging system!
 What room are you trying to reach?
</property>
<property part-name="block2" name="text">
 Please leave a message for this room!
</property>
<property part-name="block3" name="text">
 Thank you, the message will be delivered to the correct room.
</property>
...
```

Listing 6.49: Resulterend VoiceXML form

```
<vxml:form id="form1">
  <vxml:block>Welcome to the automatic messaging system!
   What room are you trying to reach?
  </vxml:block>
  <vxml:field name="field1">
```

```
    <option>1</option>
    <option>2</option>
    <option>3</option>
    <option>4</option>
    <option>5</option>
    <vxml:prompt timeout="5s">
     Please say a number between 1 and 5
    </vxml:prompt>
    <vxml:filled>Thank you<vxml:audio src="succes.wav" />
      <value expr="PostField('field1',field1)" /></vxml:filled>
    <vxml:nomatch>Incorrect input!
                Please say a number between 1 and 5
      <vxml:audio src="fail.wav" />
    </vxml:nomatch>
    <vxml:noinput>No input!
     Please say a number between 1 and 5
    </vxml:noinput>
    <vxml:help>
     You should say a number between 1 and 5</vxml:help>
  </vxml:field>
  <vxml:block>Please leave a message for this room!
  </vxml:block>
  <vxml:record name="record" />
  <vxml:block>
    Thank you, the message will be delivered to the correct room.
  </vxml:block>
</vxml:form>
```

Een mogelijk verloop van de dialoog:

```
Computer:Welcome to the automatic messaging system!
   What room are you trying to reach?
Computer:Please say a number between 1 and 5
User:6
Computer:Incorrect input!
   Please say a number between 1 and 5
   <plays sound>
User:4
Computer:Thank you
Computer:Please leave a message for this room!
User:<speaks a message>
Computer:Thank you, the message will be delivered to the correct room.
```

Hoofdstuk 7

X+VRenderer

Om een XHTML+Voice document correct te kunnen renderen, waren enkele specifieke aanpassingen nodig aan de *DocumentRenderer* (zie hoofdstuk 5). De *X+VRenderer* implementeert deze aanpassingen/toevoegingen op de *DocumentRenderer*. De volledige werking van de X+VRenderer bestaat uit de volgende stappen:

1. Gegeven een part, creeër een xml-element met de juiste naam. Zie 5.2.1
2. Vraag alle default properties [1] van de klasse van dit part op uit de vocabulary, en voeg deze op de juiste manier toe aan het gecreëerde xml-element. De mogelijkheden hier zijn: attribuut, tag en text. Zie 7.1
3. Vraag alle properties van dit part op die door de gebruiker gedefinieerd zijn, en voeg deze op de juiste manier toe aan het gecreëerde xml-element. Zie 5.2.2.
4. Controleer of er eventueel een tag rond alle subparts dient te staan moet komen. Zie 7.2.
5. Render alle subparts van dit part.

7.1 Default properties

In de vocabulary kan een property een default waarde gegeven worden. Deze default waarden zorgen ervoor dat er bepaalde eigenschappen van een widget ingesteld kunnen worden, zonder in het *style* element van het UIML document vermeld dient te worden. De default properties worden op de volgende manier ingesteld: Hier wordt een klasse *Button* gedefinieerd met als default property type, ingesteld op *"button"*.

```
<d-class id="Button" maps-type="tag" maps-to="input">
  <d-property id="value" maps-type="attribute" maps-to="value">
        <d-param type="String"/>
  </d-property>
        <d-property id="type" maps-type="attribute" maps-to="type">
          <d-param type="string">button</d-param>
        </d-property>
  <event class="click"/>
```

[1] Default properties zijn in principe generiek, en niet specifiek voor de X+VRenderer. In dit werk werden ze echter specfifiek in de X+VRenderer geïmplementeerd.

```
</d-class>
```

In dit voorbeeld wordt de default waarde van de *type* property op *button* ingesteld.

Wanneer de renderer een *part* van een bepaalde klasse aanpakt, vraagt de renderer eerst alle properties van deze klasse op die een default waarde hebben. Deze worden vervolgens toegepast op het element dat overeenkomt met de huidige klasse. Hierna worden alle gebruikergespecificeerde properties toegepast. Het is perfect mogelijk dat de gebruiker de default properties wijzigt. Dit is echter niet altijd wenselijk. In het geval van de klasse *Button*, is het niet de bedoeling dat de gebruiker manueel dit type gaat veranderen. De gebruiker zou op deze manier een checkbox of dergelijke kunnen creëren met behulp van de *Button* klasse.

Een property een default waarde geven, transparant voor de gebruiker, is noodzakelijk voor het correct renderen van het XHTML gedeelte van de vocabulary. Een aantal verschillende *klassen* in de vocabulary mappen op éénzelfde tag namelijk *input*, met als onderscheid de waarde van het *type* attribuut. Lijst van deze klassen:

1. Button: type=”button”
2. Entry: type=”text”
3. Checkbox: type=”checkbox”
4. Image: type=ïmage”
5. RadioButton: type=”radio”

Hoe een property correct gerendered wordt, wordt behandeld in 5.2.2.

7.2 Children Properties

De X+V Renderer controleert alvorens de subparts van een bepaald part te renderen of deze subparts omsloten dienen te worden door een bepaalde tag. Sommige tags in xml-gebaseerde documenten zorgen voor structurering van andere tags en bijgevolg part klassen. Het meest voor de hand liggende voorbeeld zijn *tables* in HTML. Andere interface elementen kunnen gerangschikt worden in een tabel met rijen en kolommen. Deze interface elementen worden bijgevolg omsloten door *td*(table data) en *tr*(table row).

In deze applicatie wordt er gebruik gemaakt van tables in XHTML door middel van de *HBox* en *VBox* klassen. Voor elk subpart van een HBox, dient er een nieuw element in de rij toegevoegd te worden (td). Voor elk subpart van een VBox, dient er een nieuwe rij in de tabel worden toegevoegd (tr). Wanneer er een part met als klasse HBox gecreëerd wordt, wordt elk subpart omsloten door een *td* tag, alvorens het gerendered wordt. Zie listing 7.1 en listing 7.2.

Listing 7.1: Toepassing van child properties: beschrijving in UIML

```
...
<part class="VBox">
  <part class="Label" id="label1"/>
```

```
  <part class="Label" id="label2"/>
</part>
...
<property part-name="label1" name="content">label 1</property>
<property part-name="label2" name="content">label 2</property>
...
```

Listing 7.2: Toepassing van child properties: resultaat in gerendered XML bestand

```
...
<table>
  <tr><label name="label1">label 1</label></tr>
  <tr><label name="label2">label 2</label></tr>
</table>
...
```

7.3 Verwerken van Events

7.3.1 AJAX

Er wordt gebruik gemaakt van AJAX [2](*Asynchronous JavaScript and XML*) om te communiceren tussen browser en HTTP server. Op deze manier kunnen events gesignaleerd worden door de browser aan de server, verwerkt worden door de server, en de veranderingen worden getoond in de browser. AJAX gebruikt asynchrone data transfer (HTTP requests) tussen de browser en de web server, die het de webpaginas toelaat om kleine stukken informatie op te vragen van de server in plaats van volledige paginas. AJAX is gebaseerd op bestaande web standards:

- JavaScript
- XML
- HTML
- CSS

Door middel van het JavaScript XMLHttpRequest Object kan er directe communicatie met de server gebeuren vanuit JavaScript. Op deze manier kan een webpagina gewijzigd worden, zonder dat deze pagina opnieuw ingeladen dient te worden. De inhoud van een XML-document kan op deze manier ook gewijzigd worden zonder dat het volledige document opnieuw moet worden geladen. Hoe een XMLHttpRequest Object aangemaakt wordt in JavaScript, is te zien in listing 7.3[3]. Voor meer informatie over het XMLHttpRequest object zie [4].

Listing 7.3: getXmlHttpObject JavaScript functie

```
function GetXmlHttpObject()
{
var xmlHttp=null;
try
  {
  // Firefox, Opera 8.0+, Safari
  xmlHttp=new XMLHttpRequest();
  }
catch (e)
  {
  // Internet Explorer
  try
    {
    xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
    }
  catch (e)
    {
```

[2]Asynchronous JavaScript and XML http://javascript.about.com/od/ajax/

[3]Internet Explorer gebruikt een ActiveXObject terwijl andere browsers het XMLHttpRequest Object gebruiken. IE ondersteunt echter geen XHTML+Voice en wordt enkel voor de volledigheid vermeld

[4]The XMLHttpRequest Object http://www.w3.org/TR/XMLHttpRequest/

```
    xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
  }
return xmlHttp;
}
```

7.3.2 Communicatie met server

De X+VRenderer bekijkt na het renderen van de UI ook het behavior gespecificeerd in het UIML document. Voor elk van de events wordt de bijhorende actie opgeslagen, en gelinkt aan een identifier. In het gerenderde document wordt er bij het plaatsvinden van dit event teruggekoppeld naar de server met deze identifier om zo de juiste actie te kunnen uitvoeren. De server is echter niet op de hoogte van de waarde en attributen van de gerenderde interface. Wanneer de server de waarde van een bepaalde property nodig heeft voor een bepaalde actie, wordt de waarde van deze property gevraagd aan de client. Wanneer een actie tot gevolg heeft dat de waarde van een bepaalde property verandert, stuurt de server deze waarde naar de client.

De communicatie tussen client en server werkt dus op vraag-antwoord basis. De server kan pas informatie naar de client sturen wanneer de client eerst een bepaalde request of informatie naar de server heeft gestuurd. Een event wordt dus afgehandelt door beurtelingse informatie-uitwisseling tussen client en server, waarbij de client de communicatie start. Dit heeft tot gevolg dat er soms dummy informatie wordt gestuurd, om de actie op een juiste manier af te handelen.

Voor elke stukje client-server communicatie wordt er een XMLHttpRequest object aangemaakt in een JavaScript functie. De juiste informatie wordt gezonden naar de server met behulp van de *POST* methode. Hoe deze POST methode uitgevoerd wordt in JavaScript wordt getoond in listing 7.4. In deze functie wordt het antwoord van de server onmiddelijk verwerkt door een andere functie, namelijk *processResponse*. Deze functie heeft als argument de boodschap van de server. De boodschap wordt geparsed en vervolgens op een correcte manier verwerkt.

Listing 7.4: De Post functie in JavaScript

```
function Post(msg){
  xmlHttp=GetXmlHttpObject();
  xmlHttp.open("POST",msg,true);
  xmlHttp.onreadystatechange = function(){
    if (xmlHttp.readyState==4){
      processResponse(xmlHttp.responseText);
    }
  }
  xmlHttp.send(null);
}
```

7.3.3 Events koppelen aan acties

In het behavior van een UIML document worden acties gekoppeld aan het plaatsvinden van specifieke events. De events die plaatsvinden in een X+V document zijn zowel XHTML als VoiceXML events. Alle mogelijke events worden getoond in tabel 7.3.3. De X+V renderer wijst elk event een unieke identifier toe, en koppelt deze identifier aan de actie gespecificeerd in het UIML document. Wanneer het event plaatsvindt, moet de browser terugkoppelen naar de X+VRenderer. Met behulp van JavaScript wordt er teruggekoppeld naar de server, waarbij de server het event kan identificeren aan de hand van de unieke id, en vervolgens de bijhorende actie kan uitvoeren.

Het oproepen van een JavaScript functie verloopt niet op een identieke manier in een VoiceXML form en in een XHTML form. In een VoiceXML form kan een JavaScript functie enkel opgeroepen worden met behulp van een *expr* attribuut van een bepaalde tag. Er wordt gebruik gemaakt van een vxml:value tag, die een bepaalde expressie gaat evalueren. In dit geval wordt er een JavaScript functie zonder return waarde geëvalueerd, waardoor de JavaScript functie wordt uitgevoerd. Listing 7.6 toont een rule waarbij er bij het verkeerd invullen van een bepaald veld een boodschap wordt uitgeprint in de console. De terugkoppeling naar de server is te zien in listing 7.7. Er wordt teruggekoppeld naar de server met de functie PostEvent (zie 7.5).

Listing 7.5: De PostEvent functie in JavaScript

```
function PostEvent(ev){
  xmlHttp=GetXmlHttpObject();
  xmlHttp.open("POST","event|"+ev,true);
  xmlHttp.onreadystatechange = function(){
    if (xmlHttp.readyState==4){
      processResponse(xmlHttp.responseText);
    }
  }
  xmlHttp.send(null);
}
```

Listing 7.6: Event en action gespecificeerd in UIML document

```
...
<structure>
...
        <part class="Field" name="field"/>
...
</structure>
...
<rule>
  <condition>
    <event class="vxml-nomatch" part-name="field"/>
  </condition>
  <action>
    <call name="Console.println">
```

Tabel 7.1: Mogelijke Events

	Klasse	Event
XHTML	Button	ButtonPressed
	RadioButton	onchange
	CheckButton	onchange
	Combo	onchange
VoiceXML	Field	filled
	Field	nomatch
	Field	noinput
	Field	help
	Record	filled

```
      <param>Invalid voice Input!</param>
    </call>
  </action>
</rule>
...
```

Listing 7.7: JavaScript oproepen in een VoiceXML form

```
...
<vxml:field name="field" type="boolean">
...
  <vxml:nomatch>
    <vxml:value expr="PostEvent('event1')" />
  </vxml:nomatch>
...
</vxml:field>
...
```

XHTML events kunnen op een meer traditionele manier JavaScript functies aanroepen. Zie listing 7.8 en listing 7.9.

Listing 7.8: Event en action gespecificeerd in UIML document

```
...
<structure>
...
        <part class="Button" name="buton"/>
...
</structure>
...
<rule>
  <condition>
    <event class="ButtonPressed" part-name="button"/>
```

```
  </condition>
  <action>
    <call name="Console.println">
      <param>Button clicked</param>
    </call>
  </action>
</rule>
...
```

Listing 7.9: JavaScript oproepen vanuit XHTML

```
...
<input id="button" type="button" onclick="PostEvent('event1');"/>
...
```

7.3.4 Gesproken input terug laten koppelen naar server

Door middel van een *vxml:field* kan een gebruiker gesproken input geven in een X+V interface. Het is echter niet mogelijk om vanuit JavaScript de waarde van een vxml:field op te vragen. De waarde van een vxml:field kan echter belangrijk zijn, ook voor andere delen van de interface. Er zou dus een mgogelijkheid moeten zijn om deze waarde op te kunnen vragen. De gebruikte oplossing is om meteen na het invullen van een vxml:field de waarde te sturen naar de server.

Wanneer er een vxml:field correct ingevuld wordt, vindt er een *vxml:filled* event plaats. Na het renderen van alle parts van een UIML document, gaat de renderer voor elk vxml:field het invullen van dit veld linken aan het opsturen van de ingevulde waarde. Indien de waarde van dit veld later opgevraagd wordt, is deze meteen voor handen. Hoe dit in een X+V document wordt gerealiseerd wordt getoond in listing 7.10.

Listing 7.10: Bij het invullen van een veld wordt de waarde verstuurd

```
<vxml:field name="field1" type="boolean">
  <vxml:filled>
    <vxml:value expr="PostField('field1',field1);"/>
  </vxml:filled>
</vxml:field>
```

7.3.5 XHTML events koppelen aan de Voiceforms

XHTML+Voice bestaat uit een grafische interface, gespecificeerd in XHTML en een verzameling VoiceXML forms, die geactiveerd worden wanneer een XHTML event plaatsvindt (zie 3.2 voor meer informatie over X+V). De VoiceForm klasse kan gelinkt worden aan een XHTML event met behulp van de property *executeon*. De waarde van dit property bevat de identifier van het XHTML part en de naam van het event, gescheiden door een komma. Zie listing 7.11 voor een concreet voorbeeld, waarbij button een identifier van een Button klasse is, en form1 van een VoiceForm klasse.

Listing 7.11: executeon property

```
<property part-name="form1" name="executeon">
  button, ButtonPressed
</property>
```

Het resultaat in het X+V document wordt getoond in listing 7.12. De JavaScript functie *ExecuteVoice* speelt het VoiceXML form af. Deze functie wordt getoond in listing 7.13. Deze functie stuurt een activerend event naar het correcte VoiceXML form.

Listing 7.12: Resultaat in X+V document

```
<input id="button" type="button" onclick="ExecuteVoice('form1');" >
```

Listing 7.13: ExecuteVoice JavaScript functie

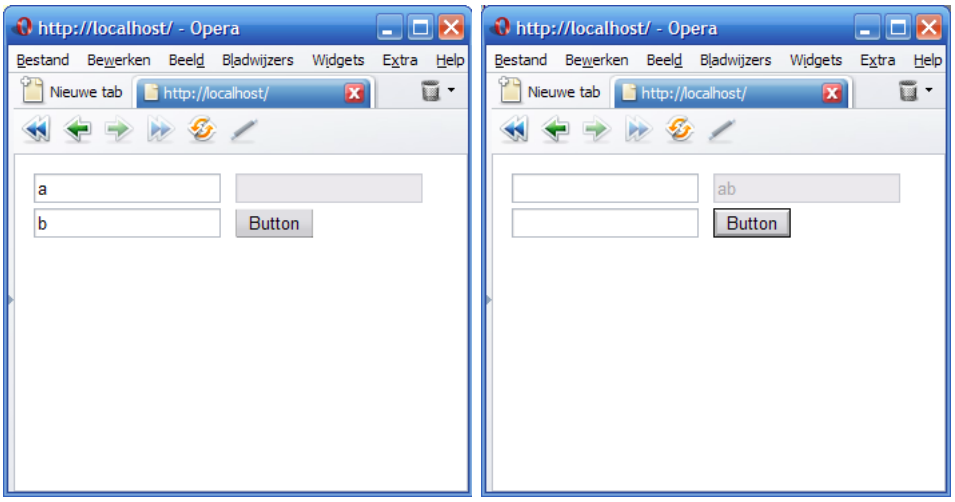
```
function ExecuteVoice(name){
  var e = document.createEvent("UIEvents");
  e.initEvent("DOMActivate","true","true");
  document.getElementById(name).dispatchEvent(e);
}
```

7.3.6 In praktijk

Ter verduidelijking wordt er een uitgebreid voorbeeld gegeven van een event en de daaropvolgende actie in een UI. In dit geval bestaat de interface uit 3 invulvelden, waarvan één niet actief, en een knop. Deze interface wordt getoond in figuur 7.1(a). HEt UIML bestand dat de UI beschrijft wordt getoond in listing 7.14. Wanneer de gebruiker op de knop drukt, wordt er een event getriggered. Het 3de (niet actieve) invulveld krijgt als waarde de concatenatie van de andere 2 invulvelden. Deze 2 invulvelden worden vervolgens leeg gemaakt. Het resultaat wordt getoond in figuur 7.1(b).

Listing 7.14: UI in UIML

```
<?xml version="1.0" encoding="utf-8" ?>
<uiml>
  <interface>
    <structure>
      <part id="root" class="html">
        <part class="hbox">
          <part class="vbox">
            <part class="Entry" id="entry1"/>
            <part class="Entry" id="entry2"/>
          </part>
          <part class="vbox">
            <part class="Entry" id="entry3"/>
            <part class="Button" id="button"/>
          </part>
        </part>
      </part>
    </structure>
    <style>
      <property part-name="button" name="value">Button</property>
```



(a) UI voor event plaatsvindt (b) UI na uitvoeren van actie

Figuur 7.1: UI voor en na de uitvoering van een actie

```
      <property part-name="entry3" name="disabled">true</property>
    </style>
    <behavior>
      <rule>
        <condition>
          <event class="ButtonPressed" part-name="button"/>
        </condition>
        <action>
          <property part-name="entry3" name="value">
            <call name="String.concatenate">
              <param>
                <property part-name="entry1" name="value"/>
              </param>
              <param>
                <property part-name="entry2" name="value"/>
              </param>
            </call>
          </property>
          <property part-name="entry1" name="value"></property>
          <property part-name="entry2" name="value"></property>
        </action>
      </rule>
    </behavior>
  </interface>
  <peers>
    <presentation base="XV.uiml"/>
  </peers>
</uiml>
```

De communicatie tussen server en client verloopt als volgt: Er wordt op de knop geklikt. De client stuurt naar de server dat er een event plaatsvond, en geeft de identifier van het event mee.

```
client: POST /event|event1 HTTP/1.1
```

De server heeft de waarde van het value attribuut van entry1 nodig.

```
server: "get|entry1.value"
```

De client stuurt de waarde van dit attribuut naar de server.

```
client: POST /prop|entry1.value=a HTTP/1.1
```

De server heeft de waarde van het value attribuut van entry2 nodig.

```
server: "get|entry2.value"
```

De client stuurt de waarde van dit attribuut naar de server.

```
client: POST /prop|entry2.value=b HTTP/1.1
```

De server concateneert de twee verkregen strings. De nieuwe waarde van het attribuut value van entry3 wordt naar de client gestuurd.

```
server: "setA|entry3.value=ab"
```

De client stelt de nieuwe waarde in, en vraagt of de actie voltooid is.

```
client: POST /more HTTP/1.1
```

De actie is nog niet voltooid, de nieuwe waarde van het attribuut value van entry1 wordt naar de client gestuurd.

```
server: "setA|entry1.value="
```

De client stelt de nieuwe waarde in, en vraagt of de actie voltooid is.

```
client: POST /more HTTP/1.1
```

De actie is nog niet voltooid, de nieuwe waarde van het attribuut value van entry2 wordt naar de client gestuurd.

```
server: "setA|entry2.value="
```

De client stelt de nieuwe waarde in, en vraagt of de actie voltooid is.

```
client: POST /more HTTP/1.1
```

De actie is voltooid. De server laat dit weten aan de client.

```
server: "done"
```

Hoofdstuk 8

Case-Study

8.1 Uitgebreid Voorbeeld

In deze sectie wordt een uitgebreide domotica user interface getoond. In deze multimodale user interface krijgt de gebruiker de mogelijkheid om verschillende elektronische toestellen te besturen. Dit voorbeeld geeft de gebruiker de mogelijkheid om een alarm aan en uit te zetten, de lichten in een kamer aan en uit doen en een boodschap over een intercom naar een bepaalde kamer versturen. Er zijn verschillende VoiceXML forms die geactiveerd worden wanneer er een XHTML event plaatsvindt. Wanneer de gebruiker het alarm aan (of uit) wilt zetten, wordt de gebruiker naar een paswoord gevraagd. Indien de gebruiker het correcte paswoord ingeeft, wordt het alarm aan (of uit) gezet. Een ander VoiceXML form wordt geactiveerd wanneer de gebruiker de lichten in een bepaalde kamer aan wil doen. Er wordt een geluidsbestand afgespeeld dat het geluid van een lichtschakelaar bevat.

Het UIML document wordt getoond in listing 8.1. Het resulterende X+V document wordt getoond in listing 8.2. In figuur 8.1 wordt een screenshot getoond van de gerenderde user interface.

Listing 8.1: domotica.uiml

```
<?xml version="1.0" encoding="utf-8" ?>
<uiml>
 <interface>
  <structure>
    <part class="Html">
      <part class="VoiceForm" id="intercomForm">
        <part class="Block" id="intercomRequest"/>
        <part class="Record" id="intercomMessage"/>
        <part class="Block" id="thanks"/>
      </part>
      <part class="VoiceForm" id="lightForm1">
        <part class="Block">
          <part class="Audio" id="lightSwitchOn"/>
        </part>
      </part>
      <part class="VoiceForm" id="lightForm2">
        <part class="Block">
          <part class="Audio" id="lightSwitchOff"/>
```

```
      </part>
    </part>
    <part class="VoiceForm" id="alarmOnForm">
      <part class="Field" id="passWord">
        <part class="Prompt" id="askPass"/>
        <part class="Nomatch" id="wrongPass"/>
        <part class="Filled" id="correctPass"/>
      </part>
    </part>
    <part class="VoiceForm" id="alarmOffForm">
      <part class="Field" id="passWordOff">
        <part class="Prompt" id="askPass"/>
        <part class="Nomatch" id="wrongPass"/>
        <part class="Filled" id="correctPass"/>
      </part>
    </part>
    <part class="HBox">
      <part class="VBox">
        <part class="Label" id="alarmLabel"/>
        <part class="Button" id="alarmOn"/>
        <part class="Button" id="alarmOff"/>
      </part>
      <part class="VBox">
        <part class="Label" id="roomLabel"/>
        <part class="Combo" id="roomSelect"/>
      </part>
      <part class="VBox">
        <part class="Label" id="intercomLabel"/>
        <part class="Button" id="intercomSpeak"/>
      </part>
      <part class="VBox">
        <part class="Label" id="lightLabel"/>
        <part class="Button" id="lightOn"/>
        <part class="Button" id="lightOff"/>
      </part>
    </part>
    <part class="Image" id="house"/>
  </part>
</structure>
<style>
  <property part-name="intercomLabel" name="text">Intercom</property>
  <property part-name="roomLabel" name="text">Available rooms:</property>
  <property part-name="intercomSpeak" name="label">Speak</property>
  <property part-name="roomSelect" name="content">
    <constant model="list">
      <constant value="Kitchen"/>
      <constant value="Living room"/>
      <constant value="Dining room"/>
      <constant value="Bath room"/>
      <constant value="Master bedroom"/>
      <constant value="Guest bedroom"/>
    </constant>
  </property>
  <property part-name="intercomForm" name="executeon">
   intercomSpeak,ButtonPressed
  </property>
  <property part-name="intercomRequest" name="text">
```

```
    Please speak your message.
  </property>
  <property part-name="thanks" name="text">Thank you!</property>
  <property part-name="lightLabel" name="text">
    Turn lights:
  </property>
  <property part-name="lightOn" name="label">ON</property>
  <property part-name="lightOn" name="disabled">true</property>
  <property part-name="lightOff" name="label">OFF</property>
  <property part-name="lightForm1" name="executeon">
    lightOn,ButtonPressed
  </property>
  <property part-name="lightSwitchOn" name="src">
    http://users.pandora.be/jonbeton/wav/light.wav
  </property>
  <property part-name="lightForm2" name="executeon">
    lightOff,ButtonPressed
  </property>
  <property part-name="lightSwitchOff" name="src">
    http://users.pandora.be/jonbeton/wav/light2.wav
  </property>
  <property part-name="alarmLabel" name="text">Turn alarm:</property>
  <property part-name="alarmOn" name="label">ON</property>
  <property part-name="alarmOff" name="label">OFF</property>
  <property part-name="alarmOff" name="disabled">true</property>
  <property part-name="alarmOnForm" name="executeon">
    alarmOn,ButtonPressed
  </property>
  <property part-name="passWord" name="options">alarm</property>
  <property part-name="passWordOff" name="options">alarm</property>
  <property part-name="askPass" name="text">
    Please enter your password
  </property>
  <property part-name="wrongPass" name="text">Incorrect password!</property>
  <property part-name="correctPass" name="text">Thank You!</property>
  <property part-name="alarmOffForm" name="executeon">
    alarmOff,ButtonPressed</property>
  <property part-name="house" name="file">
    http://users.pandora.be/jonbeton/png/house1.PNG
  </property>
</style>
<behavior>
  <rule>
    <condition>
      <event class="ButtonPressed" part-name="lightOn"/>
    </condition>
    <action>
      <property part-name="lightOn" name="disabled">true</property>
      <property part-name="lightOff" name="disabled">false</property>
    </action>
  </rule>
  <rule>
    <condition>
      <event class="ButtonPressed" part-name="lightOff"/>
    </condition>
    <action>
      <property part-name="lightOff" name="disabled">true</property>
```

```
        <property part-name="lightOn" name="disabled">false</property>
      </action>
    </rule>
    <rule>
      <condition>
        <event class="vxml-filled" part-name="passWordOff"/>
      </condition>
      <action>
        <property part-name="alarmOff" name="disabled">true</property>
        <property part-name="alarmOn" name="disabled">false</property>
        <property part-name="house" name="file">
          http://users.pandora.be/jonbeton/png/house1.PNG
        </property>
      </action>
    </rule>
    <rule>
      <condition>
        <event class="vxml-filled" part-name="passWord"/>
      </condition>
      <action>
        <property part-name="alarmOn" name="disabled">true</property>
        <property part-name="alarmOff" name="disabled">false</property>
        <property part-name="house" name="file">
          http://users.pandora.be/jonbeton/png/house2.PNG
        </property>
      </action>
    </rule>
  </behavior>
 </interface>
 <peers>
  <presentation base="XV.uiml"/>
 </peers>
</uiml>
```

Listing 8.2: X+V document gerendered vanuit listing 8.1

```
<?xml version="1.0"?>
<html xmlns="http://www.w3.org/1999/xhtml"
 xmlns:vxml="http://www.w3.org/2001/vxml"
 xmlns:ev="http://www.w3.org/2001/xml-events"
 xmlns:xv="http://www.voicexml.org/2002/xhtml+voice">
 <head>
   <script type="text/javascript"
    src="http://users.pandora.be/jonbeton/Thesis/js/script.js" />
   <vxml:form id="intercomForm">
     <vxml:block>Please speak your message.</vxml:block>
     <vxml:record name="intercomMessage" />
     <vxml:block>Thank you!</vxml:block>
   </vxml:form>
   <vxml:form id="lightForm1">
     <vxml:block>
       <vxml:audio src="http://users.pandora.be/jonbeton/wav/light.wav" />
     </vxml:block>
   </vxml:form>
   <vxml:form id="lightForm2">
     <vxml:block>
       <vxml:audio src="http://users.pandora.be/jonbeton/wav/light2.wav" />
     </vxml:block>
```

```
  </vxml:form>
  <vxml:form id="alarmOnForm">
    <vxml:field name="passWord">
      <option>alarm</option>
      <vxml:prompt>Please enter your password</vxml:prompt>
      <vxml:nomatch>Incorrect password!</vxml:nomatch>
      <vxml:filled>Thank You!
        <value expr="PostField('passWord',passWord)" />
        <value expr="PostEvent('event4')" />
      </vxml:filled>
    </vxml:field>
  </vxml:form>
  <vxml:form id="alarmOffForm">
    <vxml:field name="passWordOff">
      <option>alarm</option>
      <vxml:prompt>Please enter your password</vxml:prompt>
      <vxml:nomatch>Incorrect password!</vxml:nomatch>
      <vxml:filled>Thank You!
        <value expr="PostField('passWordOff',passWordOff)" />
        <value expr="PostEvent('event3')" />
      </vxml:filled>
    </vxml:field>
  </vxml:form>
</head>
<body>
  <table border="0">
    <td>
      <table border="0">
        <tr>
          <label id="alarmLabel">Turn alarm:</label>
        </tr>
        <tr>
          <input id="alarmOn" type="button" value="ON"
            onclick="ExecuteVoice('alarmOnForm')" />
        </tr>
        <tr>
          <input id="alarmOff" type="button" value="OFF" disabled="true"
            onclick="ExecuteVoice('alarmOffForm')" />
        </tr>
      </table>
    </td>
    <td>
      <table border="0">
        <tr>
          <label id="roomLabel">Available rooms:</label>
        </tr>
        <tr>
          <select id="roomSelect">
            <option>Kitchen</option>
            <option>Living room</option>
            <option>Dining room</option>
            <option>Bath room</option>
            <option>Master bedroom</option>
            <option>Guest bedroom</option>
          </select>
        </tr>
      </table>
```

```
    </td>
    <td>
      <table border="0">
        <tr>
          <label id="intercomLabel">Intercom</label>
        </tr>
        <tr>
          <input id="intercomSpeak" type="button" value="Speak"
            onclick="ExecuteVoice('intercomForm')" />
        </tr>
      </table>
    </td>
    <td>
      <table border="0">
        <tr>
          <label id="lightLabel">Turn lights:</label>
        </tr>
        <tr>
          <input id="lightOn" type="button" value="ON" disabled="true"
            onclick="PostEvent('event1');ExecuteVoice('lightForm1')" />
        </tr>
        <tr>
          <input id="lightOff" type="button" value="OFF"
            onclick="PostEvent('event2');ExecuteVoice('lightForm2')" />
        </tr>
      </table>
    </td>
  </table>
  <img id="house" src="http://users.pandora.be/jonbeton/png/house1.PNG" />
 </body>
</html>
```

Figuur 8.1: Screenshot van het domotica voorbeeld

8.2 Vergelijking met GTK# vocabulary

De XHTML vocabulary lijkt in vele opzichten op de GTK# vocabulary. Met enkele kleine veranderingen kan een UIML bestand dat een GTK# UI beschrijft omgezet worden in een UIML bestand dat een XHTML UI beschrijft. De verkregen UI's vertonen ook sterke gelijkenissen. Deze sectie bevat 2 voorbeelden, een blog zonder functionaliteit, en een klein voorbeeld met functionaliteit.

8.2.1 Blog

De blog bevat geen functionaliteit. In het UIML bestand wordt gebruik gemaakt van verscheidene klassen, buiten de *Frame* en *HTML* klasse komen alle klassen voor in beide vocabularies. De UIML bestanden bevinden zich in listing 8.3 en in listing 8.4. Figuur 8.2 en figuur 8.3 bevatten de screenshots van de UI's. De uiterlijke overeenkomsten zijn ook duidelijk te zien in deze figuren.

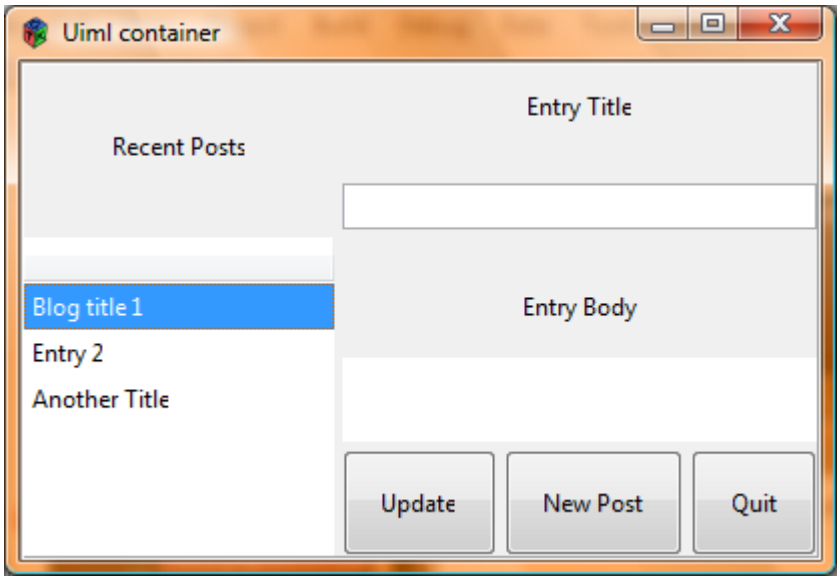
Listing 8.3: uimlblog.uiml

```
<?xml version="1.0"?>
<uiml>
  <interface>
    <structure>
      <part class="Frame" id="fr">
        <part class="HBox">
          <part class="VBox">
            <part id="recentPosts" class="Label"/>
            <part id="articles" class="List"/>
          </part>
          <part class="VBox">
            <part id="entryTitle" class="Label"/>
            <part id="title" class="Entry"/>
            <part id="entryBody" class="Label"/>
            <part id="blogentry" class="Text"/>
            <part class="HBox">
              <part id="update" class="Button"/>
              <part id="newpost" class="Button"/>
              <part id="quit" class="Button"/>
            </part>
          </part>
        </part>
      </part>
    </structure>
    <style>
      <property part-name="recentPosts" name="text">Recent Posts
          </property>
      <property part-name="entryTitle" name="text">Entry Title</property>
      <property part-name="entryBody" name="text">Entry Body</property>
      <property part-name="update" name="label">Update</property>
      <property part-name="newpost" name="label">New Post</property>
      <property part-name="quit" name="label">Quit</property>
      <property part-name="articles" name="content">
        <constant model="list">
          <constant value="Blog title 1"/>
```

```
          <constant value="Entry 2"/>
          <constant value="Another Title"/>
        </constant>
      </property>
      </style>
    <behavior/>
  </interface>
  <peers>
    <presentation base="gtk-sharp-1.0.uiml"/>
  </peers>
</uiml>
```

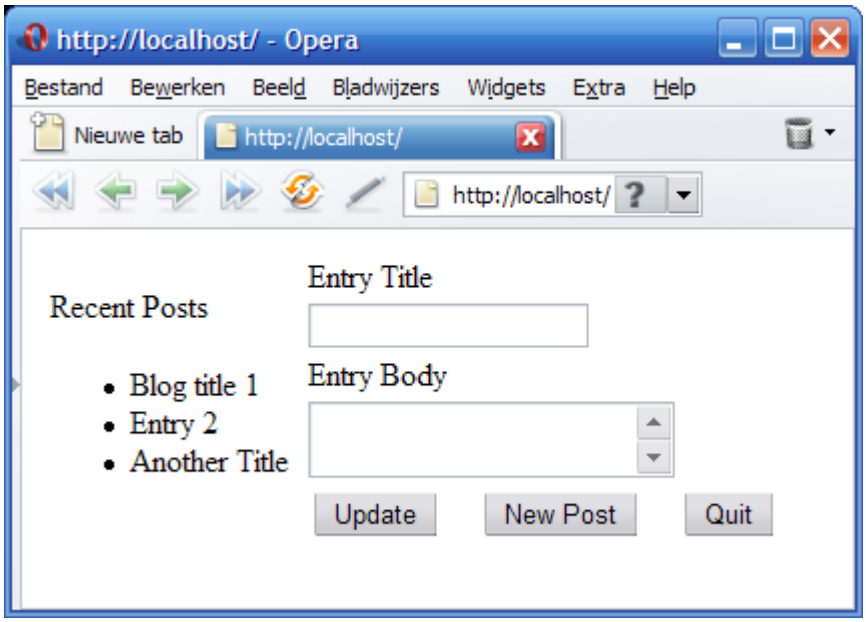
Listing 8.4: uimlblogXHTML.uiml

```
<?xml version="1.0"?>
<uiml>
  <interface>
    <structure>
      <part class="Html">
        <part class="HBox">
          <part class="VBox">
            <part id="recentPosts" class="Label"/>
            <part id="articles" class="List"/>
          </part>
          <part class="VBox">
            <part id="entryTitle" class="Label"/>
            <part id="title" class="Entry"/>
            <part id="entryBody" class="Label"/>
            <part id="blogentry" class="Text"/>
            <part class="HBox">
              <part id="update" class="Button"/>
              <part id="newpost" class="Button"/>
              <part id="quit" class="Button"/>
            </part>
          </part>
        </part>
      </part>
    </structure>
    <style>
      <property part-name="recentPosts" name="text">Recent Posts
      </property>
      <property part-name="entryTitle" name="text">Entry Title</property>
      <property part-name="entryBody" name="text">Entry Body</property>
      <property part-name="update" name="label">Update</property>
      <property part-name="newpost" name="label">New Post</property>
      <property part-name="quit" name="label">Quit</property>
      <property part-name="articles" name="content">
        <constant model="list">
          <constant value="Blog title 1"/>
          <constant value="Entry 2"/>
          <constant value="Another Title"/>
        </constant>
      </property>
```



Figuur 8.2: Resulterende interface in GTK#

```
    </style>
    <behavior/>
  </interface>
  <peers>
    <presentation base="XV.uiml"/>
  </peers>
</uiml>
```



Figuur 8.3: Resulterende interface in XHTML

8.2.2 Copy voorbeeld

Dit voorbeeld is een simpele UI dat bestaat uit 2 invulvelden en 2 knoppen. Indien er op een knop wordt geklikt, wordt de waarde van één invulveld gekopiëerd naar het andere invulveld. De UIML bestanden (listing 8.5 en listing 8.6) vertonen veel gelijkenissen. De klasse van het event heeft in beide gevallen echter een andere naam. De resulterende UI's (figuur 8.4 en figuur 8.5) zien er ook bijna identiek uit.

Listing 8.5: UIML bestand dat simpele GTK UI beschrijft

```
<?xml version="1.0"?>
<uiml>
  <interface>
    <structure>
      <part class="Frame" id="Frame">
        <part class="HBox">
          <part class="Entry" id="leftentry"/>
          <part class="VBox">
            <part class="Button" id="copyleft"/>
            <part class="Button" id="copyright"/>
          </part>
          <part class="Entry" id="rightentry"/>
        </part>
      </part>
    </structure>
    <style>
      <property part-name="Frame" name="label">Copy</property>
      <property part-name="copyleft" name="label">copy left</property>
      <property part-name="copyright" name="label">copy right</property>
      <property part-name="leftentry" name="text">          </property>
      <property part-name="rightentry" name="text">          </property>
```

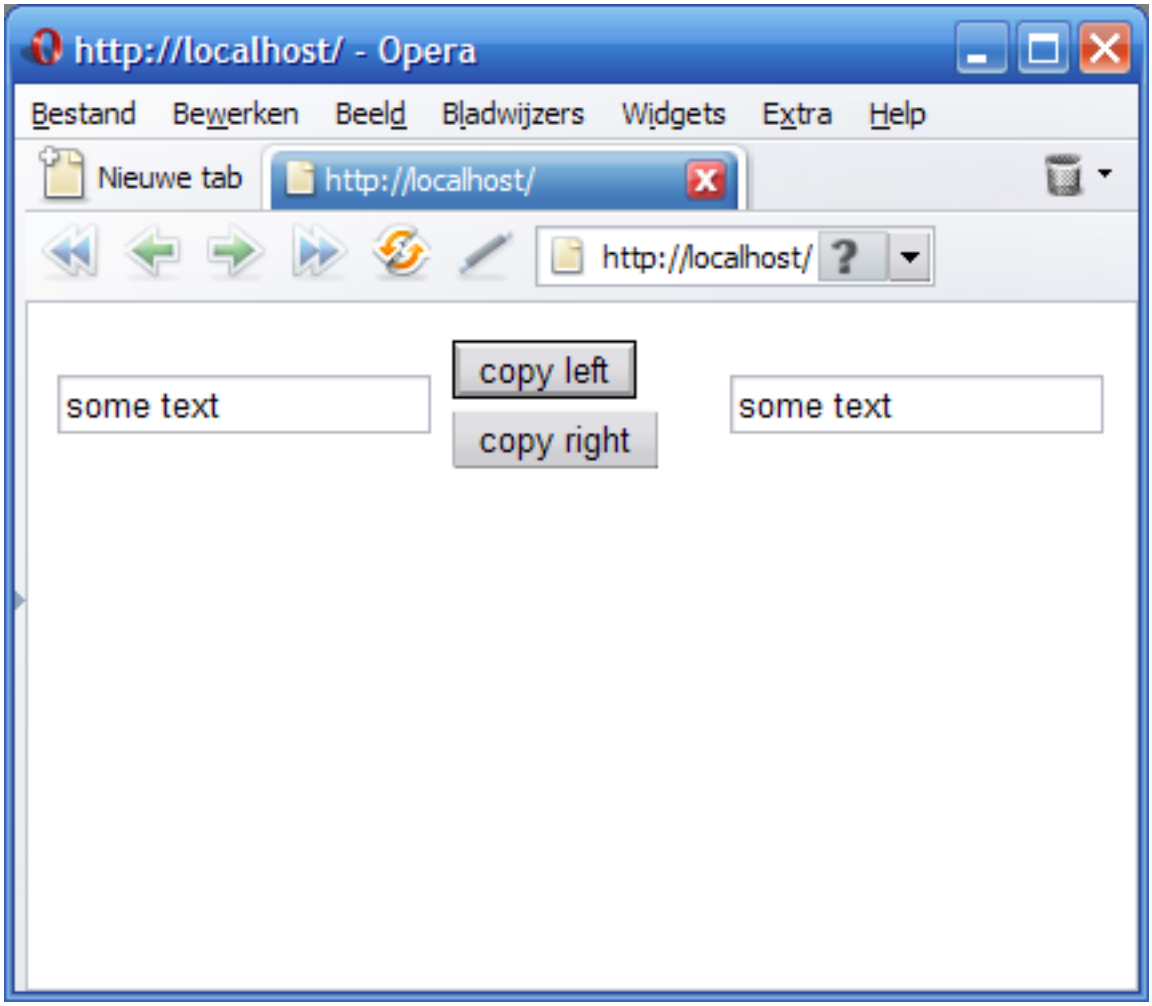
```
    </style>
    <behavior>
      <rule>
        <condition>
          <event class="ButtonPressed" part-name="copyleft"/>
        </condition>
        <action>
          <property part-name="rightentry" name="text">
            <property part-name="leftentry" name="text"/>
          </property>
        </action>
      </rule>
      <rule>
        <condition>
          <event class="ButtonPressed" part-name="copyright"/>
        </condition>
        <action>
          <property part-name="leftentry" name="text">
            <property part-name="rightentry" name="text"/>
          </property>
        </action>
      </rule>
    </behavior>
  </interface>
  <peers>
    <presentation base="gtk-sharp-1.0.uiml"/>
  </peers>
</uiml>
```

Listing 8.6: UIML bestand dat XHTML UI beschrijft analoog aan listing 8.5

```
<?xml version="1.0"?>
<uiml>
  <interface>
    <structure>
      <part class="Html">
        <part class="HBox">
          <part class="Entry" id="leftentry"/>
          <part class="VBox">
            <part class="Button" id="copyleft"/>
            <part class="Button" id="copyright"/>
          </part>
          <part class="Entry" id="rightentry"/>
        </part>
      </part>
    </structure>
    <style>
      <property part-name="copyleft" name="value">copy left</property>
      <property part-name="copyright" name="value">copy right</property>
      <property part-name="leftentry" name="value">          </property>
      <property part-name="rightentry" name="value">          </property>
    </style>
    <behavior>
```

Figuur 8.4: Resulterende interface in GTK#

```
      <rule>
        <condition>
          <event class="ButtonPressed" part-name="copyleft"/>
        </condition>
        <action>
          <property part-name="rightentry" name="value">
             <property part-name="leftentry" name="value"/>
          </property>
        </action>
      </rule>
      <rule>
        <condition>
          <event class="ButtonPressed" part-name="copyright"/>
        </condition>
        <action>
          <property part-name="leftentry" name="value">
            <property part-name="rightentry" name="value"/>
          </property>
        </action>
      </rule>
    </behavior>
  </interface>
  <peers>
    <presentation base="XV.uiml"/>
  </peers>
</uiml>
```



Figuur 8.5: Resulterende interface in XHTML

Deel IV

Conclusie

Hoofdstuk 9

Conclusie

9.1 Samenvatting

In dit werk wordt een methode voorgesteld die user interface ontwerpers in staat stelt om op een eenvoudige manier multimodale user interfaces te ontwikkelen. XHTML+Voice(X+V) wordt gebruikt om multimodale interfaces te omschrijven. X+V is een XML gebaseerde markup taal. De User Interface Markup Language(UIML) wordt gebruikt om de multimodale user interface op een meer abstractere manier te beschrijven. Als UIML raamwerk wordt er gebruik gemaakt van Uiml.Net. Vanuit een UIML document wordt een X+V document gerendered met behulp van de Uiml.Net renderer. De Uiml.Net renderer werkte enkel reflectie-gebaseerd, en was niet in staat om een XML document te renderen. De architectuur van Uiml.Net is nu uitgebreid zodat er naast reflectie-gebaseerde renderers ook document-gebaseerde renderers kunnen gebruikt worden. Een documentgebaseerde renderer maakt een XML document aan, en voegt elementen en attributen toe in een XML boom. Voor de implementatie van deze documentrenderer werd zoveel mogelijk gesteund op de al bestaande architectuur van Uiml.Net.

Een X+V renderer wordt in dit werk beschreven. Deze documentgebaseerde renderer behandelt enkele X+V specifieke implementatie details. Om functionaliteit aan de user interface toe te voegen, worden er events en acties gespecificeerd in het UIML document. Wanneer een bepaald event plaats vindt, wordt er teruggekoppeld naar de renderer. Een HTTP daemon wordt gelinkt aan de X+V renderer. Via deze daemon kan er gecommuniceerd worden tussen browser en renderer met behulp van Asynchronous JavaScript and XML (AJAX). Met behulp van AJAX kan er informatie in beide richtingen worden gestuurd.

Met behulp van de gecreëerde X+V vocabulary worden de abstracte klassen naar de juiste X+V elementen gemapt. Het grafische(XHTML) gedeelte van de X+V vocabulary vertoont zeer sterke gelijkenissen met de GTK# vocabulary. Met slechts enkele kleine aanpassingen kan een UIML document voor een GTK# user interface aangepast worden tot een UIML document voor een X+V user interface. De resulterende user interfaces lijken ook sterk op elkaar. Het VoiceXML gedeelte van de interface is veel minder generiek. Bij het creëren van dit deel van de vocabulary kwamen er enkele knelpunten boven. Het ontbreken van de

mogelijkheid om VoiceXML forms te wijzigen en op te vragen vanuit JavaScript zorgt voor beperkingen.

9.2 Problemen en toekomstig werk

De events die mogelijk zijn in een X+V UIML document zijn enkel client-side. Er is geen mogelijkheid om bepaalde events die in de application logic beschreven zijn te gebruiken. Een voorbeeld hiervan is een timer die afloopt na een bepaalde tijd. De communicatie met behulp van AJAX is vraag-antwoord gebaseerd. De server kan pas een bericht sturen naar de client wanneer de client er een naar de server heeft gestuurd. Indien men ook server-side events mogelijk wil maken, moet de server ten alle tijde de client kunnen bereiken. Dit zou gerealiseerd kunnen worden met dummy berichten van de client naar de server, zodat de client altijd bereikbaar is voor de server. *Comet* [1] is een software concept waarbij de web servers data naar de client kunnen sturen zonder eerst een client binnen te krijgen.Ook in de HTML5[2] draft is een mechanisme aanwezig om serverside events toe te laten.

Bepaalde events in een VoiceXML form worden afgehandeld in het structure deel van een UIML document. Events zouden echter afgehandeld moeten worden in het behavior deel van het UIML document. Deze events worden voorgesteld door een tag in een X+V document. Deze tag kan op zijn beurt andere tags bevatten. Het is echter niet mogelijk om dit in een action element te specificeren, de UIML specificatie laat dit niet toe. Zonder de UIML specificatie te veranderen, is het niet mogelijk om dit probleem beter op te lossen.

Ook de link tussen een XHTML event en een VoiceXML form zou in het behavior deel afgehandeld moeten worden. Momenteel wordt er gebruik gemaakt van een eigenschap van een VoiceXML form, waarin part en event worden vermeld (zie listing 9.1). Een betere afhandeling wordt getoond in listing 9.2, waarbij er teruggekoppeld wordt naar de renderer met de naam van het VoiceXML form.

Listing 9.1: Huidige link tussen event en VoiceXML form

```
<style>
  <property part-name="form1" name="executeon">button,ButtonPressed
  </property>
</style>
```

Listing 9.2: Betere link tussen event en VoiceXML form

```
<behavior>
  <rule>
    <condition>
      <event class="ButtonPressed" part-name="button"/>
    </condtion>
    <action>
          <call name="Renderer.VoiceActivate"/>
```

[1] http://en.wikipedia.org/wiki/Comet_(programming)

[2] HTML 5 Working Draft 21 August 2007 http://www.whatwg.org/specs/web-apps/current-work/#server-sent-events

```
            <param>form1</param>
          </call>
    </action>
  </rule>
</behavior>
```

Een van de grootste minpunten is het ontbreken van ondersteuning voor het VoiceXML DOM (Document Object Model). Het is niet mogelijk om een VoiceXML form aan te passen vanuit JavaScript. De VoiceXML forms zijn bijgevolg onveranderlijk. Dit beperkt de mogelijkheden van de UI ontwerper. Het is ook niet mogelijk om vanuit JavaScript de waarde van een bepaald VoiceXML field op te vragen. Dit probleem is wel verholpen door de waarde van het VoiceXML field onmiddellijk na invulling naar de X+V renderer te sturen. Indien er ondersteuning voor het VoiceXML DOM aanwezig zou zijn, zou ook de inhoud van een VoiceXML form aangepast kunnen worden, en dit zou een dynamischere user interface tot gevolg hebben.

In een vocabulary kunnen default properties ingesteld worden. In sommige gevallen zoals bijvoorbeeld het type property bij de Button en RadioButton klasse, is het niet wenselijk dat de UIML ontwerper deze property aan gaat passen in het UIML document. Op deze manier zou een Button klasse grafisch voorgesteld kunnen worden door een invulveld of een checkbutton. Door het *locken* van deze default property zou dit onmogelijk gemaakt worden. De UIML specificatie uitbreiden met een mogelijkheid om d-properties te definiëren die niet te veranderen zijn door de UIML ontwerper is een mogelijkheid. Zie listing 9.3 voor een mogelijke oplossing.

Listing 9.3: Manier om default properties te locken

```
<d-class id="Button" maps-type="tag" maps-to="input">
  <d-property id="value" maps-type="attribute" maps-to="value" locked="false">
    <d-param type="String"/>
  </d-property>
  <d-property id="type" maps-type="attribute" maps-to="type" locked="true">
    <d-param type="String">button</d-param>
  </d-property>
</d-class>
```

Bibliografie

[1] User Interface Markup Language (UIML) Specification version 3.1. Technical report
Marc Abrams and James Helms.
Oasis UIML TC, 2004.

[2] UIML: An Appliance-Independent XML User Interface Language
Marc Abrams, Constantinos Phanouriou, Alan L. Batongbacal, Stephen M. Williams, and Jonathan E. Shuster
WWW8 / Computer Networks, 1999

[3] UIML: An XML Language for Building Device-Independent User Interfaces Marc Abrams, Constantinos Phanouriou

[4] Uiml.Net: An open UIML renderer for the .NET Framework
Kris Luyten, Karin Coninx

[5] A generic approach for multi-device user interface rendering with UIML
Kris Luyten, Kristof Thys, Jo Vermeulen, Karin Coninx

[6] Applying Model-Based Techniques to the Development of UIs for Mobile Computers
Jacob Eisenstein, Jean Vanderdonckt, and Angel Puerta
RedWhale Software Corporation

[7] The Role of Natural Language in a Multimodal Interface
Philip R. Cohen
Computer Dialogue Laboratory, Artificial Intelligence Center,SRI international

[8] XHTML+Voice Profile 1.2

[9] XHTML+Voice Programmers Guide Version 1.0
IBM

[10] Building multi-platform user interfaces with UIML
Mir Farooq Ali, Manuel A. Prez-Quiones, Marc Abrams, Eric Shell

[11] Designing Mona: User Interactions with Multimodal Mobile Applications
Lynne Baillie, Raimund Schatz, Rainer Simon, Hermann Anegg, Florian Wegscheider, Georg Niklfeld, Alexander Gassner

[12] Multimodal Interfaces in Mobile Devices The MONA Project
Hermann Anegg, Georg Niklfeld, Michael Pucher, Raimund Schatz, Rainer Simon, Florian Wegscheider
Thomas Dangl,Michael Jank

[13] UIML for Voice Interfaces
Sumanth Lingam, Harmonia, Inc.

[14] A GenericWidget Vocabulary for the Generation of Graphical and Speech-Driven User Interfaces
C.J. PLOMP VTT Electronics

[15] Derivation of a Dialog Model from a Task Model by Activity Chain Extraction
Kris Luyten, Tim Clerckx, Karin Coninx, and Jean Vanderdonckt

[16] ICARE: A Component-Based Approach for the Design and Development of Multimodal Interfaces
Jullien Bouchet and Laurence Nigay

[17] Four easy pieces for assessing the usability of multimodal interaction: the CARE properties
Joëlle Coutaz, Laurence Nigay, Daniel Salber, Ann Blandford, Jon May, Richard M. Young

[18] Mixed-Initiative, Trans-modal Interface Migration
Renata Bandelloni, Silvia Berti, and Fabio Paterno

[19] Migratory MultiModal Interfaces in MultiDevice Environments
Silvia Berti, Fabio Paterno

[20] ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models
F. Paterno', C.Mancini, S.Meniconi

[21] XWeb: An Architecture for Cross-modal Collaboration
Dan R. Olsen Jr., William A. Moyes, Sean S. Jefferies, S. Travis Nielsen
Computer Science Department, Brigham Young University, Provo, Utah

[22] Cross-modal Interaction using XWeb Dan R. Olsen Jr., Sean S. Jefferies, S. Travis Nielsen, William A. Moyes, Paul Fredrickson
Computer Science Department, Brigham Young University, Provo, Utah

[23] Dialog Modelling for Multiple Devices and Multiple Interaction Modalities
Robbie Schaefer, Steffen Bleul, Wolfgang Mueller

[24] Multimodal Dialog Description for Mobile Devices
S. Bleul, R. Schaefer and W. Mueller

Bijlage A

JavaScript

Listing A.1: script.js

```
function PostProp(prop,value){
  xmlHttp=GetXmlHttpObject();
  xmlHttp.open("POST","prop|"+prop+"="+value,true);
  xmlHttp.onreadystatechange = function(){
    if (xmlHttp.readyState==4){
      processResponse(xmlHttp.responseText);
    }
  }
  xmlHttp.send(null);
}
function PostEvent(ev){
  xmlHttp=GetXmlHttpObject();
  xmlHttp.open("POST","event|"+ev,true);
  xmlHttp.onreadystatechange = function(){
    if (xmlHttp.readyState==4){
      processResponse(xmlHttp.responseText);
    }
  }
  xmlHttp.send(null);
}
function PostField(fieldId,value){
  xmlHttp=GetXmlHttpObject();
  xmlHttp.open("POST","field|"+fieldId+"="+value,true);
  xmlHttp.onreadystatechange = function(){
    if (xmlHttp.readyState==4){
      processResponse(xmlHttp.responseText);
    }
  }
  xmlHttp.send(null);
}
function ExecuteVoice(name){
```

```
  var e = document.createEvent("UIEvents");
  e.initEvent("DOMActivate","true","true");
  document.getElementById(name).dispatchEvent(e);
}
function processResponse(responseText){
  if(responseText=="done")
    return;
  else{
    resp=responseText.split('|');
    if(resp.length!=2)
      return;
    if(resp[0]=="get")
      processGet(resp[1]);
    if(resp[0]=="setA")
      setA(resp[1]);
    if(resp[0]=="setI")
      setI(resp[1]);
  }
}
function setA(e){
  resp=e.split('=');
  if(resp.length!=2)
    return;
  value=resp[1];
  prop=resp[0].split('.');
  if(prop.length!=2)
    return;
  setValue(prop[0],prop[1],value);
  xmlHttp=GetXmlHttpObject();
  xmlHttp.open("POST","more",true);
  xmlHttp.onreadystatechange = function(){
    if (xmlHttp.readyState==4){
      processResponse(xmlHttp.responseText);
    }
  }
  xmlHttp.send(null);
}
function setI(e){
  resp=e.split('=');
  if(resp.length!=2)
    return;
  document.getElementById(resp[0]).innerHTML=resp[1];
  xmlHttp=GetXmlHttpObject();
  xmlHttp.open("POST","more",true);
  xmlHttp.onreadystatechange = function(){
    if(xmlHttp.readyState==4){
      processResponse(xmlHttp.responseText);
```

```
    }
  }
  xmlHttp.send(null);
}
function getValue(partname,prop){
  if(prop=="value")
    return(document.getElementById(partname).value);
  else if(prop=="checked")
    return(document.getElementById(partname).checked);
  else if(prop=="disabled")
    return(document.getElementById(partname).disabled);
  else
    return(document.getElementById(partname).getAttribute(prop));
}
function setValue(partname,prop,value){
  if(prop=="value")
    document.getElementById(partname).value=value;
  else if(prop=="checked")
    document.getElementById(partname).checked=value;
  else if(prop=="disabled"){
    if(value=="false")
      document.getElementById(partname).disabled=false;
    else
      document.getElementById(partname).disabled=true;
  }
  else
    document.getElementById(partname).setAttribute(prop,value);
}
function processGet(get){
  prop=get.split('.');
  if(prop.length!=2)
    return;
  else
    PostProp(get,getValue(prop[0],prop[1]));
}
function GetXmlHttpObject(){
  var xmlHttp=null;
  try
  {
    xmlHttp=new XMLHttpRequest();
  }
  catch (e)
  {
    try
    {
      xmlHttp=new ActiveXObject("Msxml2.XMLHTTP");
    }
```

```
    catch (e)
    {
      xmlHttp=new ActiveXObject("Microsoft.XMLHTTP");
    }
  }
  return xmlHttp;
}
```

Bijlage B

XHTML + Voice Vocabulary

Listing B.1: XV.uiml

```
<uiml>
 <presentation base="XV" how="cascade">
  <d-class id="Html" maps-type="tag" maps-to="html" />
  <d-class id="Button" maps-type="tag" maps-to="input">
    <d-property id="label" maps-type="attribute" maps-to="value">
      <d-param type="String"/>
    </d-property>
    <d-property id="type" maps-type="attribute" maps-to="type">
      <d-param type="String">button</d-param>
    </d-property>
    <d-property id="disabled" maps-type="attribute" maps-to="disabled">
      <d-param type="String"/>
    </d-property>
    <property class="ButtonPressed" maps-type="event" maps-to="onclick"/>
  </d-class>
  <d-class id="Label" maps-type="tag" maps-to="label">
    <d-property id="text" maps-type="text" maps-to="content"/>
  </d-class>
  <d-class id="Entry" maps-type="tag" maps-to="input">
    <d-property id="text" maps-type="attribute" maps-to="value">
      <d-param type="String"/>
    </d-property>
    <d-property id="size" maps-type="attribute" maps-to="size">
      <d-param type="String"/>
    </d-property>
    <d-property id="maxlength" maps-type="attribute" maps-to="maxlength">
      <d-param type="String"/>
    </d-property>
    <d-property id="disabled" maps-type="attribute" maps-to="disabled">
      <d-param type="String"/>
    </d-property>
    <d-property id="type" maps-type="attribute" maps-to="type">
      <d-param type="String">text</d-param>
    </d-property>
  </d-class>
```

```
<d-class id="Text" maps-type="tag" maps-to="textarea">
  <d-property id="disabled" maps-type="attribute" maps-to="disabled">
    <d-param type="String"/>
  </d-property>
  <d-property id="text" maps-type="text" maps-to="content">
    <d-param type="String"/>
  </d-property>
  <d-property id="cols" maps-type="attribute" maps-to="cols">
    <d-param type="String"/>
  </d-property>
  <d-property id="rows" maps-type="attribute" maps-to="rows">
    <d-param type="String"/>
  </d-property>
</d-class>
<d-class id="Image" maps-type="tag" maps-to="img">
  <d-property id="file" maps-type="attribute" maps-to="src">
    <d-param type="String"/>
  </d-property>
  <d-property id="alt" maps-type="attribute" maps-to="alt">
    <d-param type="String"/>
  </d-property>
  <d-property id="width" maps-type="attribute" maps-to="width">
    <d-param type="String"/>
  </d-property>
  <d-property id="height" maps-type="attribute" maps-to="height">
    <d-param type="String"/>
  </d-property>
</d-class>
<d-class id="CheckButton" maps-type="tag" maps-to="input">
  <d-property id="checked" maps-type="attribute" maps-to="checked">
    <d-param type="String"/>
  </d-property>
  <d-property id="label" maps-type="tag" maps-to="label">
    <d-param type="string"/>
  </d-property>
  <d-property id="disabled" maps-type="attribute" maps-to="disabled">
    <d-param type="String"/>
  </d-property>
  <d-property id="type" maps-type="attribute" maps-to="type">
    <d-param type="String">checkbox</d-param>
  </d-property>
  <d-property id="onchange" maps-type="event" maps-to="onchange"/>
</d-class>
<d-class id="RadioButton" maps-type="tag" maps-to="input">
  <d-property id="type" maps-type="attribute" maps-to="type">
    <d-param type="String">radio</d-param>
  </d-property>
  <d-property id="label" maps-type="tag" maps-to="label">
    <d-param type="string"/>
  </d-property>
  <d-property id="name" maps-type="attribute" maps-to="name">
    <d-param type="String"/>
  </d-property>
```

```
    <d-property id="checked" maps-type="attribute" maps-to="checked">
      <d-param type="String"/>
    </d-property>
    <d-property id="disabled" maps-type="attribute" maps-to="disabled">
      <d-param type="String"/>
    </d-property>
  </d-class>
  <d-class id="Combo" maps-type="tag" maps-to="select">
    <d-property id="content" maps-to="option" maps-type="tag">
      <d-param type="String"/>
    </d-property>
    <d-property id="entry" maps-to="value" maps-type="attribute">
      <d-param type="String"/>
    </d-property>
    <d-property id="disabled" maps-type="attribute" maps-to="disabled">
      <d-param type="String"/>
    </d-property>
    <d-property id="change" maps-type="event" maps-to="onchange"/>
  </d-class>
  <d-class id="List" maps-type="tag" maps-to="ul">
    <d-property id="content" maps-to="li" maps-type="tag">
      <d-param type="String"/>
    </d-property>
  </d-class>
  <d-class id="VBox" maps-type="tag" maps-to="vtable">
    <d-property id="border" maps-type="attribute" maps-to="border">
      <d-param type="String">0</d-param>
    </d-property>
  </d-class>
  <d-class id="HBox" maps-type="tag" maps-to="htable">
    <d-property id="border" maps-type="attribute" maps-to="border">
      <d-param type="String">0</d-param>
    </d-property>
  </d-class>
  <d-class id="Title" maps-type="tag" maps-to="title">
    <d-property id="text" maps-type="text" maps-to="text">
      <d-param type="String"/>
    </d-property>
  </d-class>
  <d-class id="VoiceForm" maps-type="tag" maps-to="vxml-form">
    <d-property id="executeon" maps-type="attribute" maps-to="when"/>
  </d-class>
  <d-class id="Block" maps-type="tag" maps-to="vxml-block">
    <d-property id="text" maps-type="text" maps-to="text"/>
  </d-class>
  <d-class id="Field" maps-type="tag" maps-to="vxml-field">
    <d-property id="options" maps-to="option" maps-type="tag"/>
    <d-property id="type" maps-type="attribute" maps-to="type">
      <d-param type="String"/>
    </d-property>
    <d-property id="grammar" maps-type="tag" maps-to="grammar">
      <d-param type="String"/>
    </d-property>
```

```
    <property class="Nomatch" maps-type="event" maps-to="vxml-nomatch"/>
    <property class="Filled" maps-type="event" maps-to="vxml-filled"/>
    <property class="Help" maps-type="event" maps-to="vxml-help"/>
    <property class="Noinput" maps-type="event" maps-to="vxml-noinput"/>
  </d-class>
  <d-class id="Audio" maps-type="tag" maps-to="vxml-audio">
    <d-property id="src" maps-type="attribute" maps-to="src">
      <d-param type="String"/>
    </d-property>
  </d-class>
  <d-class id="Record" maps-type="tag" maps-to="vxml-record">
    <d-property id="maxtime" maps-type="attribute" maps-to="maxtime">
      <d-param type="Number"/>
    </d-property>
  </d-class>
  <d-class id="Prompt" maps-type="tag" maps-to="vxml-prompt">
    <d-property id="bargein" maps-type="attribute" maps-to="bargein">
      <d-param type="boolean"/>
    </d-property>
    <d-property id="count" maps-type="attribute" maps-to="count">
      <d-param type="Number"/>
    </d-property>
    <d-property id="timeout" maps-type="attribute" maps-to="timeout">
      <d-param type="Number"/>
    </d-property>
    <d-property id="text" maps-type="text" maps-to="CONTENT">
      <d-param type="String"/>
    </d-property>
  </d-class>
  <d-class id="Filled" maps-type="tag" maps-to="vxml-filled">
    <d-property id="text" maps-type="text" maps-to="content">
      <d-param type="String"/>
    </d-property>
  </d-class>
  <d-class id="Nomatch" maps-type="tag" maps-to="vxml-nomatch">
    <d-property id="text" maps-type="text" maps-to="content">
      <d-param type="String"/>
    </d-property>
    <d-property id="count" maps-type="attribute" maps-to="count">
      <d-param type="Number"/>
    </d-property>
  </d-class>
  <d-class id="Noinput" maps-type="tag" maps-to="vxml-noinput">
    <d-property id="text" maps-type="text" maps-to="content">
      <d-param type="String"/>
    </d-property>
    <d-property id="count" maps-type="attribute" maps-to="count">
      <d-param type="Number"/>
    </d-property>
  </d-class>
  <d-class id="Help" maps-type="tag" maps-to="vxml-help">
    <d-property id="text" maps-type="text" maps-to="content">
      <d-param type="String"/>
```

```
    </d-property>
    <d-property id="count" maps-type="attribute" maps-to="count">
      <d-param type="Number"/>
    </d-property>
  </d-class>
 </presentation>
 <logic>
  <d-component id="Console" maps-to="System.Console">
    <d-method id="print" returns-value="void" maps-to="Write">
      <d-param id="message" type="System.String"/>
    </d-method>
    <d-method id="println" returns-value="void" maps-to="WriteLine">
      <d-param id="message" type="System.String"/>
    </d-method>
  </d-component>
  <d-component id="String" maps-to="System.String">
    <d-method id="concatenate" returns-value="string" maps-to="Concat">
      <d-param id="str0" type="System.String"/>
      <d-param id="str1" type="System.String"/>
    </d-method>
  </d-component>
  <d-component id="Math" maps-to="System.Math">
    <d-method id="absolute" returns-value="int" maps-to="Abs">
      <d-param id="in" type="System.Int32"/>
    </d-method>
    <d-method id="power" return-value="double" maps-to="Pow">
      <d-param id="ground" type="System.Double"/>
      <d-param id="power" type="System.Double"/>
    </d-method>
    <d-method id="acos" return-value="double" maps-to="Acos">
      <d-param type="System.Double"/>
    </d-method>
    <d-method id="cos" return-value="double" maps-to="Cos">
      <d-param type="System.Double"/>
    </d-method>
    <d-method id="asin" return-value="double" maps-to="Asin">
      <d-param type="System.Double"/>
    </d-method>
    <d-method id="sin" return-value="double" maps-to="Sin">
      <d-param type="System.Double"/>
    </d-method>
    <d-method id="atan" return-value="double" maps-to="Atan">
      <d-param type="System.Double"/>
    </d-method>
    <d-method id="tan" return-value="double" maps-to="Tan">
      <d-param type="System.Double"/>
    </d-method>
    <d-method id="e" return-value="double" maps-to="E"/>
    <d-method id="pi" return-value="double" maps-to="PI"/>
  </d-component>
 </logic>
</uiml>
```

Auteursrechterlijke overeenkomst

Opdat de Universiteit Hasselt uw eindverhandeling wereldwijd kan reproduceren, vertalen en distribueren is uw akkoord voor deze overeenkomst noodzakelijk. Gelieve de tijd te nemen om deze overeenkomst door te nemen, de gevraagde informatie in te vullen (en de overeenkomst te ondertekenen en af te geven).

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Multimodal interfaces on mobile devices: mangling speech and graphical
Richting: **Master in de informatica** Jaar: **2007**
in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Ik ga akkoord,

Rob Van Roey

Datum: **22.08.2007**