Instituto de Ciências Matemáticas e de Computação

ISSN - 0103-2569

Introdução a XHTML+VoiceXML: Adicionando voz às aplicações Web

Americo Talarico Neto
Renata Pontin de Mattos Fortes

Nº 323

RELATÓRIOS TÉCNICOS DO ICMC

São Carlos
Maio/2008

Tabela de Conteúdo

Índice de Figuras

1. Introdução

Este relatório apresenta um conjunto das principais características das linguagens de marcação que permitem criar aplicações multimodais Web. Em especial, são mostrados os elementos e atributos que compõem a linguagem de marcação X+V e exemplos de uso.

O W3C está trabalhando atualmente para difundir a linguagem de marcação padrão para aplicações multimodais Web, o X+V (acrônimo para XHTML+VoiceXML). Existem dois grupos dentro do W3C que trabalham com o tema multimodalidade, o *Voice Browser Activity Group* e o *Multimodal Activity Group* que estão focados em criar padrões no campo de interfaces de voz e de aplicações multimodais.

Assim como o VXML, o X+V supre a demanda dos usuários por aplicações baseadas em interação por voz, tanto em dispositivos pequenos e móveis como em interações tradicionais com dispositivos convencionais baseados em interação com teclado e dispositivos de apontamento.

A eficiência na realização de tarefas é uma das principais razões para se desenvolver interfaces multimodais, assim como o fornecimento de dados de entrada usando teclado e mouse em paralelo com comandos de voz, o que permite ao usuário acessar e responder mais rapidamente à informação apresentada por seus dispositivos.

A interação multimodal também dá ao usuário a possibilidade de escolher como ele quer interagir com o sistema em um determinado momento, além de reduzir o esforço do usuário ao interagir utilizando uma única modalidade. A possibilidade de trocar de modalidade durante a interação também favorece a diminuição na taxa de erros na execução de uma tarefa.

O X+V combina três linguagens de marcação (o XHTML que trata da parte visual, o VoiceXML que trata da parte de voz e XML Events que agrupa a parte visual e a de voz) para criar sites que funcionam em:

- Navegadores convencionais (*Visual–only browsers*).
- Navegadores Multimodais, ou seja, navegadores com a capacidade de reconhecimento de fala (*Automatic Speech Recognition*) e tecnologia TTS (*text-to-speech*).
- Servidores de voz para acesso via telefone.

Marcações visuais informam ao navegador por meio de uma *engine* gráfica como a interface deve ser exibida e como ela deve se comportar quando o usuário digita, aponta ou clica. Já as marcações de voz informam aos navegadores por meio de uma *engine* de fala qual o comportamento que deve ser utilizado quando o usuário fala determinada palavra ou frase. O processo de habilitar interfaces gráficas para funcionarem com voz consiste em quebrar a

interface gráfica em componentes básicos (input box, check box, etc), criar trechos de áudio para cada componente, associar o áudio à marcação de cada um dos componentes visuais e definir as palavras ou frases que podem habilitar cada um dos componentes visuais.

O X+V incorpora o framework de eventos DOM usado no XML Events, por causa da natureza orientada a eventos das aplicações Web. Usando esse framework o X+V define eventos similares aos que são usados no HTML como “mouseover”, “focus” para correlacionar as marcações de voz e visuais.

No exemplo a seguir, o usuário responde à primeira solicitação, "Fale a cidade de partida" com entrada de voz: "Boston Massachusetts." O *engine* de fala reconhece a frase por meio do uso de uma gramática e retorna uma string de texto. O texto é atualizado na tela com o auxílio de Javascript e a aplicação muda o foco da entrada ao campo seguinte, onde a interação seguinte ocorre.

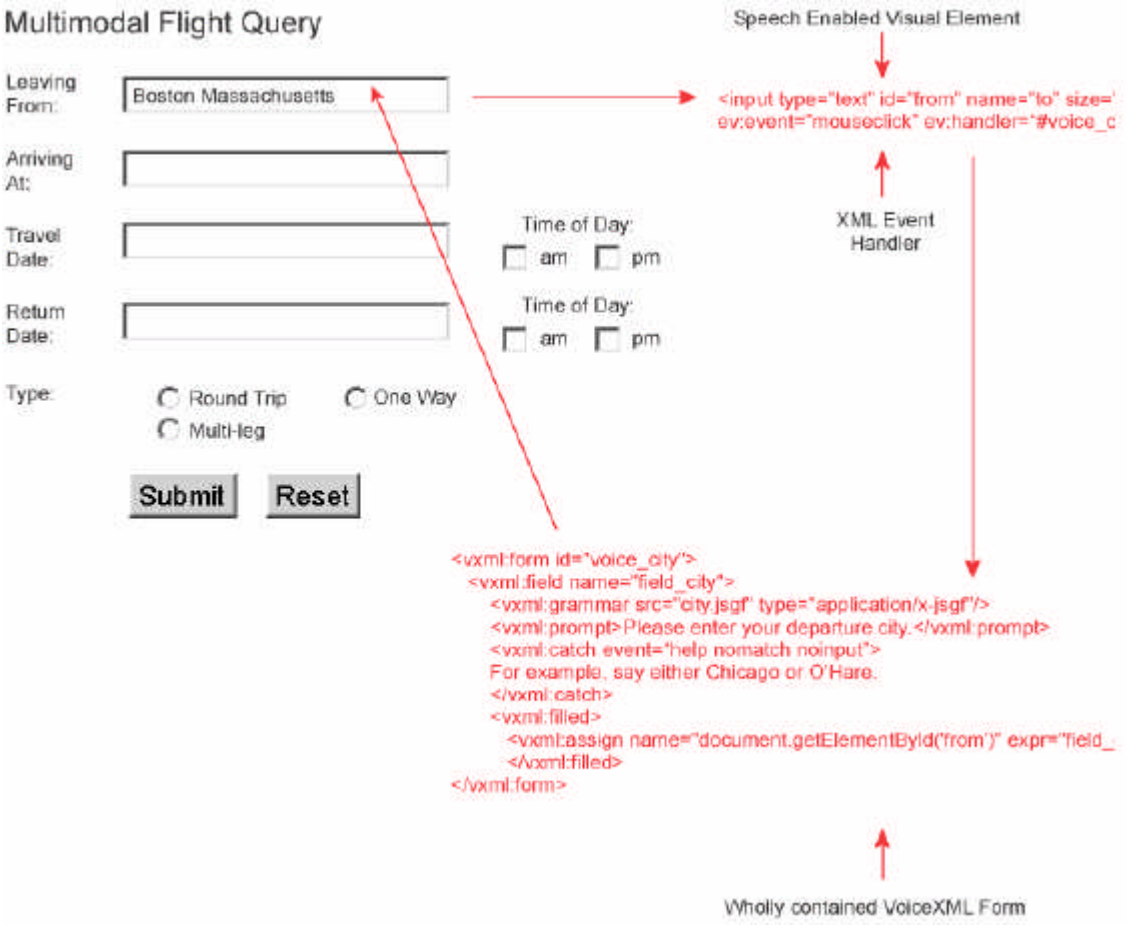

Figura 1. Exemplo de Interface Multimodal Web

A seguir serão mostradas as principais características das linguagens de marcação que permitem criar aplicações multimodais Web com voz e mais adiante serão mostrados os elementos e atributos que compõem a linguagem de marcação X+V e exemplos de uso. Para

obter informações detalhadas sobre os elementos e atributos que formam as linguagens apresentadas, consulte a Bibliografia no final deste Relatório Técnico.

2. XHTML (eXtensible HyperText Markup Language)

Todas as linguagens de marcação da web são baseadas em SGML (Standard Generalized Markup Language), uma metalinguagem projetada com a finalidade de servir de base para criação de outras linguagens de marcação. O SGML foi usado para criar o XML (Extensible Markup Language), também uma metalinguagem, porém bem mais simples.

Com XML é possível criar uma linguagem de marcação com tags e atributos para escrever um documento web. XHTML foi criado dentro deste conceito e por isso é uma aplicação XML. As tags e atributos do XHTML foram criadas aproveitando-se as tags e atributos do HTML 4.01 e suas regras.

XHTML é uma aplicação XML, escrita para substituir o HTML e nada mais é do que um HTML "puro, claro e limpo" e a transformação de um documento existente de HTML para XHTML é uma tarefa bem simples.

São várias as vantagens de se usar XHTML e, dentre elas destaca-se em primeiro plano a compatibilidade com as futuras aplicações, garantindo desde já que as criações XHTML serão conservadas estáveis por longos anos. A tendência é a de que futuras versões de navegadores, deixem de suportar elementos e atributos já em desuso segundo as recomendações do W3C, assim como antigos e ultrapassados esquemas e esboços do HTML.

Além disso, o tempo de carregamento de uma página XHTML é mais rápido pois os browsers interpretam uma página "limpa" sem ter que interpretar e decidir sobre renderização de erros de código.

Uma página XHTML é mais acessível aos browsers e aplicações de usuário aumentando a interoperabilidade e a portabilidade dos documentos web além de ser totalmente compatível com todas as aplicações para HTML, antigas e já ultrapassadas. A seguir é mostrado um documento mínimo em XHTML .

```
<!DOCTYPE Doctype goes here>
    <html xmlns="http://www.w3.org/1999/xhtml">
        <head>
            <title>Título da página</title>
        </head>
        <body> texto do body vai aqui </body>
    </html>
```

As principais diferenças entre XHTML e HTML, são:

- todas as tags devem ser escritas em letras minúsculas;
- as tags devem estar convenientemente aninhadas;

- os documentos devem ser bem formados;
- o uso de tags de fechamento é obrigatório;
- elementos vazios devem ser fechados;
- diferenças para os atributos.

2.1. Todas as tags devem ser escritas em letras minúsculas

XML é sensível a maiúsculas e minúsculas. Como o XHTML é uma aplicação XML, deve-se usar letras minúsculas.

Errado: **<DIV><P>texto</P></DIV>**

Certo: ** texto **

2.2. As tags devem estar convenientemente aninhadas

Errado: ** texto negrito **

Certo: ** texto negrito **

2.3. Os documentos devem ser bem formados

Um documento é bem formado quando está estruturado de acordo com as regras definidas nas Recomendações para XML 1.0. Todos os elementos XHTML devem estar corretamente aninhados dentro do elemento raiz <html>. A estrutura básica do documento deve ser conforme a seguir:

```
<html>
    <head>
     ...
    </head>
    <body>
     ...
    </body>
</html>
```

2.4. O uso de tags de fechamento é obrigatório

Em HTML é permitido omitir a tag de fechamento para determinados elementos. XML não permite omissão de qualquer tag de fechamento.

Errado:

** primeiro parágrafo. Segundo parágrafo.**

Certo:

** primeiro parágrafo. Segundo parágrafo. **

2.5.Elementos vazios devem ser fechados

Elementos vazios devem ter uma tag de fechamento ou a tag de abertura deve terminar com />. Como exemplo,
 ou <hr></hr>.

Errado: Elementos vazios sem terminação

**
**

<hr>

<img src="imagem.gif" alt=" imagem ">

Certo: Elementos vazios com terminação

**
**

<hr />

****

2.6.Diferenças para os atributos

2.6.1. Nomes de atributos

Assim como as tags, os atributos também são sensíveis à maiúsculas e minúsculas e então deve-se escrever nomes de atributos em minúsculas;

Errado: **<td ROWSPAN="3">**

Certo: **<td rowspan="3">**

2.6.2. Valores de atributos

Os valores de atributos devem estar entre "aspas";

Errado: **<td rowspan=3>**

Certo: **<td rowspan="3">**

2.6.3. Sintaxe dos atributos

A sintaxe para atributos deve ser escrita por completo;

Errado:

<input checked />

Certo:

<input checked="checked" />

Abaixo uma relação dos atributos que se enquadram nesta recomendação

- compact="compact"

- checked="checked"
- declare="declare"
- readonly="readonly"
- disabled="disabled"
- selected="selected"
- defer="defer"
- ismap="ismap"
- nohref="nohref"
- noshade="noshade"
- nowrap="nowrap"
- multiple="multiple"
- noresize="noresize"

2.6.4. Os atributos id e name

O HTML define o atributo name para os elementos a, applet, form, frame, iframe, img, e map. HTML também introduziu o atribute id. Ambos os atributos foram projetados para serem usados como identificadores.

Em XML, os identificadores são exclusivamente do tipo ID, e poderá existir somente um atributo do tipo ID por elemento. Além disso um determinado identificador deve ser único no documento. Documentos XHTML 1.0 compatíveis com XML e bem estruturados, devem usar o atributo id ao definir identificadores para os elementos listados acima.

Notar que em XHTML 1.0, o atributo name destes elementos está formalmente em desuso e possivelmente será excluido nas versões futuras de XHTML.

Errado: ****

Certo: ****

Nota: Por razões de compatibilidade com browsers antigos é possível usar ambos os atributos como abaixo:

****

2.6.5. O atributo lang;

O atributo lang destina-se a definir em que foi escrito o documento HTML e o atributo xml:lang para definir o idioma em que foi escrito o documento XML.

2.7. Pontos de âncoras

Em HTML para criar um ponto de âncora, associamos um nome ao elemento <a>:

** <a name="topo" >Início do parágrafo..0... **

Em XHTML adicione o atributo id:

** <a id="topo" name="topo" >Início do parágrafo..1... **

O atributo alt para imagens

Em XHTML o uso do atributo alt para imagens é obrigatório

****

Se tratar-se de uma imagem decorativa pode-se usar o atributo alt vazio:

****

2.8. Separadores de blocos de códigos

É comum o uso de uma seqüência de caracteres dentro da marcação de comentários (**<!-- comentário -->)** para visualmente separar trechos do código com a finalidade de facilitar sua posterior leitura e manutenção. Não use a clássica seqüência de caracteres ------, para conseguir este efeito. Alguns agentes de usuário mais antigos podem ter dificuldades na interpretação desta seqüência. Use por exemplo a seqüência ==== . ou xxxxxx

Errado:

<!-- Aqui começa o menu -->

<!-- ---------- -->

código XHTML do menu

<!-- ---------- -->

Certo:

<!-- Aqui começa o menu -->

<!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->

código XHTML do menu

<!-- xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx -->

2.9. Códigos gerados por editores

Cuidado com os códigos gerados por editores.

Este é um código gerado por editor: onMouseOver=function() não válido em XHTML

Errado: **onMouseOver=function()**

Certo: **onmouseover=function()**

Caracter & (ampersand)

Codifique o & (e comercial)

Errado: **C&A**

Certo: **C&A**

2.10. Validação do documento XHTML

O W3C disponibiliza um validador gratuito para documentos XHTML. Ali, você digita a URL ou o caminho para o arquivo no seu HD e um robo analisa o documento fornecendo um relatório completo e detalhado das não conformidades por ventura existentes. É uma ferramenta excelente para uso durante a elaboração ou migração do documento para XHTML. Abaixo o link para o validador:

Um validador XHTML do W3C: http://validator.w3.org/

3. VXML

3.1. Visão geral

VoiceXML é uma tecnologia utilizada para a criação de documentos que proporcionam diálogos entre usuário e a máquina utilizando o reconhecimento de voz e de tons DTMF (Dual Tone Multi Frequency). O principal objetivo dessa tecnologia é a criação de documentos e aplicações web que permitam a interação por voz.

VoiceXML 1.0 foi publicada pelo VoiceXML Forum, fundado em Março de 2000 pela AT&T, IBM, Lucent e Motorola e financiado por mais de 500 empresas. O forum passou o controle do VXML para o W3C, e agora se concentra em verificação de conformidade, cursos e marketing. O W3C publicou recentemente o VoiceXML 2.1.

A seguir, o exemplo mais básico de VXML:

Hello world

```
<vxml version="1.0">
        <form>
                <block>Alô Mundo!</block>
        </form>
</vxml>
```

A seguir um exemplo simples de código VoiceXML que pede ao usuário que escolha uma bebida e depois submete o resultado para um script no servidor

Segundo exemplo

```
<vxml version="2.1">
<form>
  <field name="drink">
   <prompt>Would you like coffee, tea, milk, or nothing?</prompt>
   <grammr src="drink.gram" type="application/x-jsgf"/>
  </field>
<block>
  <submit next= "http://www.drink.example/drink2.asp"/>
</block>
</form>
</vxml>
```

Um exemplo da interação:

C (computador): Would you like coffee, tea, milk, or nothing?

H (humano): Orange juice.

C: I did not understand what you said.

C: Would you like coffee, tea, milk, or nothing?

H: Tea

C: (submete resultados para script em asp, por exemplo)

Um documento VXML é composto por elementos chamados de diálogos. Existem dois tipos de diálogos compostos pelos elementos <form> e <menu>. A execução do documento é iniciada no primeiro diálogo por default, e após a execução desse diálogo, será determinado o próximo. Se um diálogo não especificar o próximo, o documento termina.

3.2. Objetivos da linguagem VoiceXML

- Minimizar as interações cliente/servidor especificando várias dessas interações em um único documento;
- Criar uma camada de abstração independe de plataforma;
- Ser fácil para criação de diálogos simples e ainda prover suporte a diálogos mais complexos.

3.3. Arquitetura

Uma aplicação em VoiceXML consiste em vários componentes como pode ser observado a seguir.

- Servidor de Aplicação: tipicamente um servidor Web, que executa a lógica da aplicação, e pode conter um banco de dados ou interfaces para um banco de dados externo ou para um servidor.
- Servidor de Telefonia VoiceXML: uma plataforma que executa um interpretador de VoiceXML que atua como um cliente para o servidor de aplicação. O interpretador entende os diálogos VoiceXML e controla os recursos de fala e telefonia. Esses recursos incluem ASR, TTS, funções de gravação de áudio, assim como uma interface de rede para telefonia.
- Rede Internet: uma rede baseada em pacotes TCP/IP que conecta o servidor de aplicação ao servidor de telefonia via HTTP.
- Rede de telefonia: Tipicamente a rede pública de telefonia (Public Switched Telephone Network - PSTN), mas pode ser uma rede privada de telefonia (PBX), ou rede VoIP.

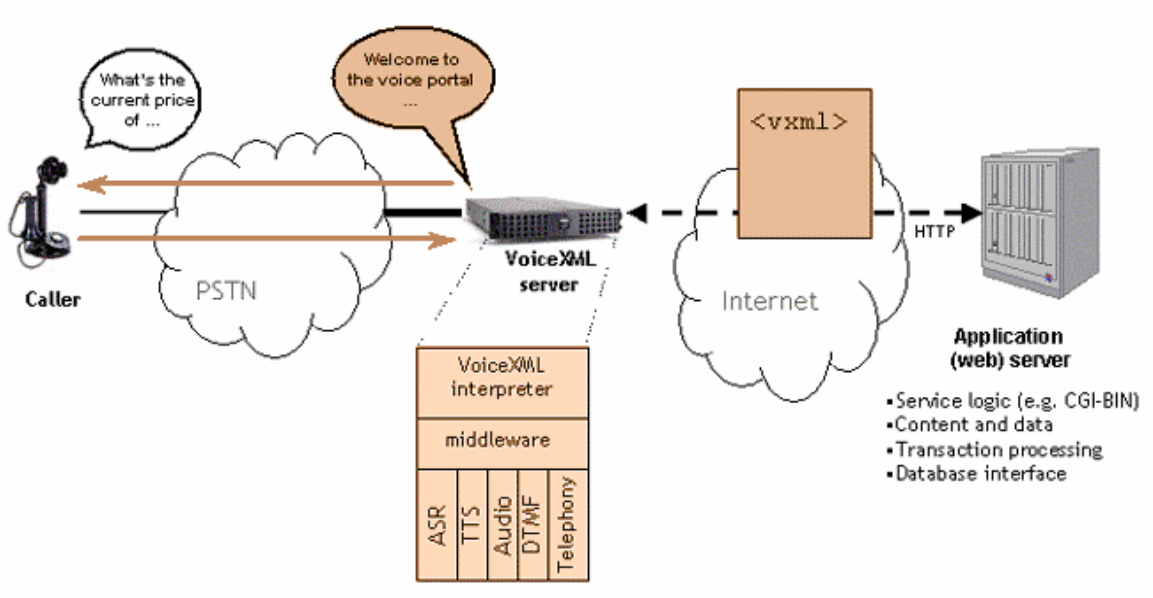

Figura 2. Arquitetura VXML retirada de voicexmlreview.org

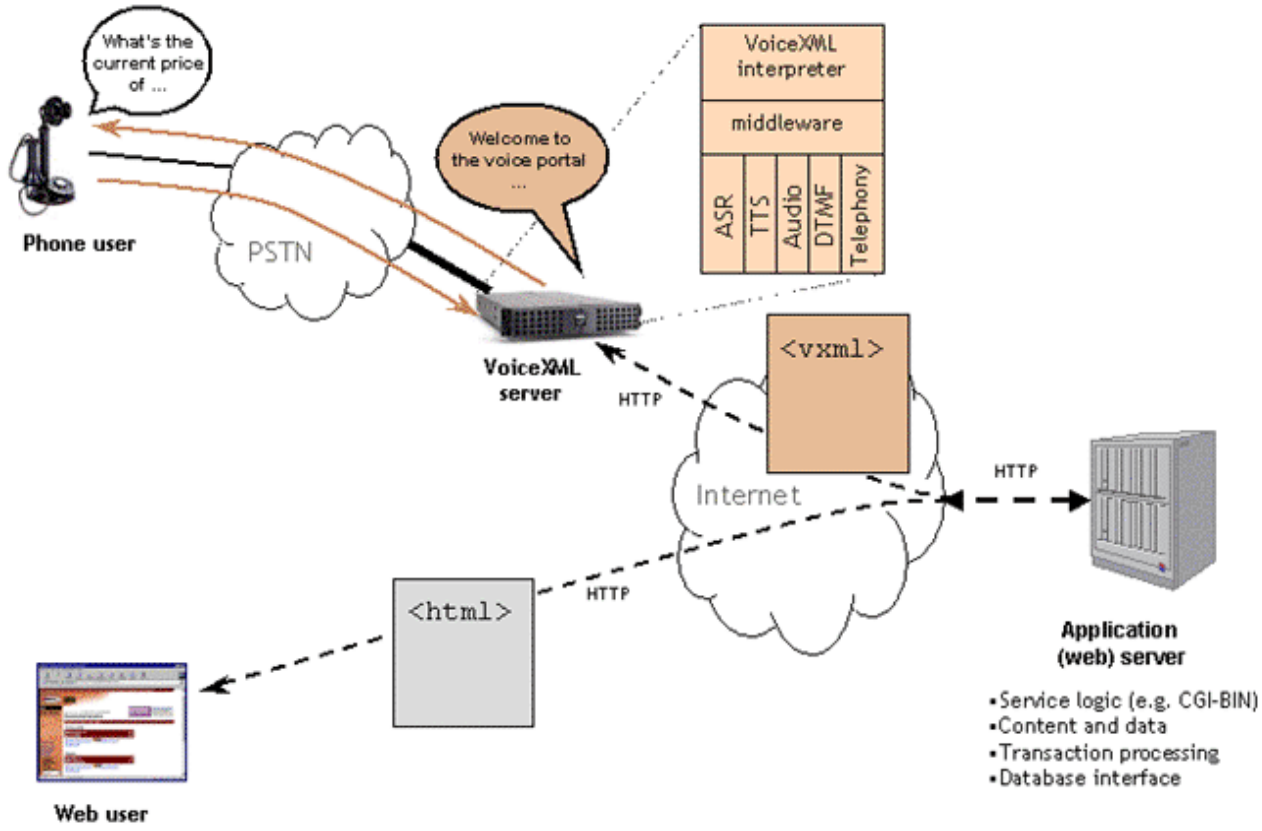

Figura 3. Arquitetura Web + VXML retirada de voicexmlreview.org

A figura anterior mostra a relação entre uma aplicação Web tradicional e uma app Web habilitada por voz.

3.4. Componentes

Os principais componentes de um documento VXML são:

- Forms
- Menus
- Links
- Variáveis e expressões
- Gramáticas
- Eventos
- Prompt
- Filled
- Meta
- Property
- Param
- Goto

Para uma referência completa dos componentes VXML, acesse http://www.w3.org/TR/voicexml21/

3.5. Forms e o FIA

Forms são os componentes chave dos documentos VoiceXML. Existem dois tipos de forms em VoiceXML: **Directed forms e Mixed Initiative Forms**. Directed forms são executados em sequência. A seguir temos um exemplo de tal interação.

Exemplo: Directed Form

```
<form id="weather_info">
    <block>Welcome to the weather information service.</block>
        <field name="state">
            <prompt>What state?</prompt>
            <grammar src="state.gram" type="application/x-jsgf"/>
            <catch event="help">
                Please speak the state for which you want the
            weather.
```

```
            </catch>
        </field>
        <field name="city">
            <prompt>What city?</prompt>
            <grammar src="city.gram" type="application/x-jsgf"/>
            <catch event="help">
                Please speak the city for which you want the
            weather.
            </catch>
        </field>
        <submit next="/servlet/weather" namelist="city state"/>
    </block>
</form>
```

O diálogo é sequencial:

C (computador): Welcome to the weather information service.

What state?

H (humano): Help

C: Please speak the state for which you want the weather.

H: Georgia

C: What city?

H: Tblisi

C: I did not understand what you said. What city?

H: Macon

C: The conditions in Macon Georgia are sunny and clear at 11 AM …

A diferença entre directed e mixed–initiative form é que no mixed initiative, os campos <field> podem ser preenchidos em qualquer ordem e mais do que um campo pode ser preenchido como resultado de uma única fala do usuário. Com o uso do elemento <initial> no form, pode-se executar um diálogo do tipo mixed-initiative. No mesmo exemplo que foi dado anteriormente, uma possível interação entre o usuário e o computador em um form do tipo mixed initiative, poderia ser:

C: Welcome to the weather information service.

C: For what city and state would you like the weather?

H: Uh, California.

C: Please say the city in California for which you want the weather.

H: San Francisco, please.

C: Do you want to hear the weather for San Francisco, California?

H: No

C: What state?

H: Los Angeles.

C: Do you want to hear the weather for Los Angeles, California?

H: Yes

C: Mostly sunny today with highs in the 80s. Lows tonight from the low 60s..

Na interação anterior foi mostrado que apesar da pergunta ter sido relacionada ao estado, o usuário respondeu com o nome da cidade e a máquina foi capaz de entendê-lo. Isso foi possível pois as gramáticas de cidade e estado estavam ativas durante o diálogo.

Os Forms são interpretados por um algoritmo de interpretação de formulários (Form Interpretation Algorithm – FIA). FIA é uma especificação de como os interpretadores VoiceXML devem funcionar e é o que habilita o VoiceXML a ser uma linguagem declarativa. No FIA, existe um loop que seleciona repetidamente um elemento <form> que a seguir é interpretado. A interpretação de um elemento form envolve os seguintes passos:

- Selecionar e tocar um ou mais prompts.
- Obter uma entrada do usuário, ou lançar algum evento (erro, ajuda, repetir, voltar, etc).
- Interpretar as ações <filled> que pertencem aos campos <field> recentemente preenchidos.
- O FIA termina quando acontece alguma ação de transferência, como <goto> para outro diálogo, ou <submit> para um servidor e também com <exit> implícito.

3.6.Menu

Um menu é um atalho para um formulário que pede ao usuário para fazer uma escolha e leva a diferentes locais, baseado nesta escolha. O menu seguinte oferece três escolhas ao usuário, Esportes, Tempo e Economia:

```
<menu>
<prompt>Bem-vindo. Escolha um tema:<enumerate/></prompt>
<choice next="http://.../esporte.vxml">Esportes</choice>
<choice next="http://.../tempo.vxml">Tempo</choice>
```

```
<choice next="http://.../economia.vxml">
Economia</choice>
<noinput>Selecione um tema
</menu>
```

O diálogo poderia ser como o seguinte:

C: Bem-vindo. Escolha um tema: esportes; tempo; economia.

H: Astrologia.

C: Eu não entendi o que você disse. (uma mensagem padrão da plataforma.)

C: Bem-vindo. Selecione um tema: esportes; tempo; economia.

H: esportes.

C: (prossegue para http..../esporte.vxml)

O elemento <menu> determina o escopo de sua gramática. Os atributos do menu são:

- id: O identificador do menu.
- scope: O escopo de gramática do menu. Se é um diálogo (o padrão) a gramática do menu só fica ativa quando o usuário transita pelo menu. Se o escopo é documento, a gramática fica ativa por todo o documento (ou se o menu está no documento raiz da aplicação, qualquer documento aberto da aplicação).
- dtmf: habilita tons de telefonia.

3.7. Modelo de Interpretação

Um menu comporta-se como um formulário com um campo simples que faz todo o trabalho. As escolhas de menu se transformam em campos de escolha. O interpretador de menu se transforma em interpretador de campos. A gramática do menu se transforma em gramática de formulário. Uma vez chamada, a gramática do menu é construída e ativada, e a escolha é iniciada. Quando o usuário entra com uma escolha, o controle transita de acordo com o valor do atributos "next", "expr" ou "event" do <choice>, e apenas um deles pode ser especificado.

3.8. Choice

O elemento <choice> serve para vários propósitos:

- Ele especifica um fragmento falado de gramática e/ou um fragmento DTMF de gramática que determina quando a escolha foi feita.
- O conteúdo é usado para formar a string de escolha.
- Ele especifica a URL de destino, quando a escolha é selecionada.

Os atributos do <choice> são:

· dtmf: A sequência DTMF para essa escolha.

· next: A URL do próximo diálogo ou documento.

· event: Especifica um evento para ser lançado ao invés de um "next".

· expr: Especifica uma expressão a ser avaliada ao invés de um "next".

Menus podem contar apenas com voz, apenas com DTMF ou ambos combinados incluindo um elemento <property> no <menu>. Aqui temos um menu somente DTMF com seqüências DTMF explicitas dadas para cada escolha, usando o atributo "choice":

```
<menu>
<property name="inputmodes" value="dtmf"/>
<prompt>Para esportes tecle1, Para tempo tecle2, Para economia
tecle3</prompt>
<choice dtmf="1" next="http://.../esporte.vxml"/>
<choice dtmf="2" next="http://.../tempo.vxml"/>
<choice dtmf="3" next="http://.../economia.vxml"/>
</menu>
```

Alternativamente, você pode mudar o atributo "dtmf" do <menu> para verdadeiro para atribuir números DTMF seqüenciais para cada uma das nove primeiras escolhas. A primeira escolha tem DTMF = 1 , e assim segue:

```
<menu>
<property name="inputmodes" value="dtmf"/>
<prompt>Para esportes tecle 1, Para tempo tecle 2, Para economia tecle
3.</prompt>
<choice dtmf="1" next="http://.../esporte.vxml"/>
<choice dtmf="2" next="http://.../tempo.vxml"/>
```

<choice dtmf="3" next="http://.../economia.vxml"/>

</menu>

3.9. Enumerate

O elemento <enumerate> é uma descrição gerada automaticamente das escolhas disponíveis para o usuário. Ele especifica um modelo que é aplicado para cada escolha na ordem que elas aparecem no menu. Se ele é usado sem conteúdo, um modelo padrão que lista todas as escolhas é usado, determinado pelo interpretador. Se ele tem conteúdo, o conteúdo é o especificador do modelo. Este especificador deve se referir a duas variáveis especiais: _prompt é o texto da escolha, e _dtmf é a seqüência dtmf atribuída. Por exemplo, se o menu é escrito assim:

```
<menu dtmf="true">
<prompt>
Bem-vindo.<enumerate>Para <value expr="_prompt"/>, tecle <value
expr="_dtmf"/>.</enumerate>
</prompt>
<choice next="http://.../esporte.vxml">esportes</choice>
<choice next="http://.../tempo.vxml">tempo</choice>
<choice next="http://.../economia.vxml">notícias economias</choice>
</menu>
```

As escolhas do menu seriam:

- Bem-vindo. Para esportes, tecle 1. Para tempo, tecle 2. Para notícias economias, tecle 3.

O elemento <enumerate> também pode ser usado analogamente em escolhas por elementos que contenham um conjunto de elementos

3.10. Field

O elemento field facilita um dialogo e permite ao interpretador coletar a informação do usuário. As falas do usuário são comparadas e casadas com as gramáticas ativas, até que uma condição do elemento <filled> for executada.

3.11. Filled

O elemento Filled especifica uma ação para execução quando alguma combinação de campos estiver preenchida pela entrada do usuário.

- mode: Pode ser especificado em either all (padrão), ou any. Se Any, uma ação é executada quando quaisquer dos campos especificados estiver preenchido pela última entrada do usuário. Se for *all*, esta ação é executada quando todos os campos mencionados estão preenchidos.
- namelist. Especifica a lista de nomes.

3.12. Gramática

Qualquer "frase de escolha" especifica um conjunto de palavras para serem obedecidas. O usuário pode dizer qualquer frase consistindo em qualquer subconjunto de palavras na mesma ordem que elas ocorrem na "frase de escolha"; Uma "frase de escolha" é construída de elementos contidos diretamente ou indiretamente no elemento . Por exemplo, em resposta a frase "notícias economias" um usuário pode dizer "economias" e "notícias economias". A regra JSGF equivalente pode ser "[notícias] [economias]" (onde […] indica opcionalidade).

O elemento <grammar> é usado para fornecer uma gramática que:

- especifica um conjunto de discursos que o usuário pode falar para executar uma ação ou fornecer uma informação, e
- fornecer um valor de "string" correspondente (no caso de um campo gramática) ou definir um conjunto de pares-de-atributos (no caso de um formulário gramática) para descrever a informação ou ação.

O elemento <grammar> é programado para acomodar qualquer formato de gramática que atenda estes dois requisitos. No momento o VoiceXML não especifica o formato de gramática nem requer suporte a um formato particular de gramática. Esta situação é similar a situação de formatos de áudio gravados para VoiceXML, e formatos de mídia em geral para HTML.

O elemento <grammar> deve ser usado para especificar uma gramática "inline" ou uma gramática "externa". Uma gramática "inline" é especificada pelo conteúdo de um elemento <grammar>:

<grammar type="mime-type">inline grammar</grammar>

Ele pode ser necessária no caso de confinar o conteúdo numa sessão CDATA.

Para gramáticas "inline" o parâmetro tipo deve ser MIME que governa a interpretação do conteúdo da tag <grammar>.

Uma gramática "externa" é especificada por um elemento do formulário.

<grammar src="URI" type="mime-type"/>

O tipo MIME é opcional neste caso porque esta informação pode ser obtida via o protocolo da URL (neste caso HTTP), e pode ser deduzido de uma extensão do nome do arquivo. Se o tipo não é especificado, e não pode ser deduzido, o tipo padrão é especifico da plataforma. Entretanto, se o tipo é especificado usando o atributo "type" ele sobrepõe outra informação sobre o tipo.

Os atributos de <grammar> são:

- src: A URL especificando o local da gramática, se ele for externa
- scope: Qualquer documento, que faz a gramática ativa em todos os diálogos do documento atual , ou diálogo, para fazer a gramática ativa por todo o formulário atual. Se omitido, o escopo da gramática é decido procurando no elemento pai.
- type: O tipo MIME da gramática. Se omitido, o interpretador de contexto irá tentar determinar o tipo dinamicamente.

3.12.1. Gramáticas DTMF

O elemento <dtmf> é usado pra especificar uma gramática DTMF que:

- define um conjunto de pressionamento de teclas que um usário pode usar para executar uma ação ou fornecer uma informação, e
- define a seqüência de string correspondente que descreve a informação ou ação.

O elemento <dtmf> é projetado para acomodar qualquer formato de gramática que atende estes dois requisitos. VoiceXML não especifica nem requer suporte para nenhum formato particular de gramática: como em <grammar>, é esperado que esforços e pressões do mercado irão causar cada interpretador de contexto VoiceXML a suportar conjuntos de formatos comuns.

O elemento <dtmf> pode se referir a uma gramática externa:

<dtmf src="URI" type="mime-type"/>

ou a uma gramática "inline":

<dtmf type="mime-type">
<!-- inline dtmf grammar -->
</dtmf>

Os atributos de <dtmf> são os mesmos de <grammar>:

- src: A URL especificando o local da gramática, se ele for externa.
- scope: Qualquer documento, que faz a gramática ativa em todos os diálogos do documento atual , ou diálogo, para fazer a gramática ativa por todo o formulário atual. Se omitido, o escopo da gramática é decido procurando no elemento pai.
- type: O tipo MIME da gramática. Se omitido, o interpretador de contexto irá tentar determinar o tipo dinamicamente.

3.12.2. Escopo de gramática

Gramáticas que se localizam dentro de um elemento link têm o escopo do elemento que contém o link. Assim, se eles são definidos no documento raiz da aplicação, os links também são ativos em qualquer documento carregado.

Gramáticas de formulário têm por padrão escopo de diálogo, então elas são ativas apenas quando o usuário está no formulário. Se elas têm escopo de documento e o documento é o raiz da aplicação, então elas são também ativas a qualquer hora em qualquer outro documento na mesma aplicação.

Gramáticas <menu> também têm por padrão escopo de diálogo e são ativas somente quando o usuário está no menu. Mas elas podem ter escopo de documento e ficarem ativas por todo o documento, e se o documento é o raiz da aplicação, também ficam ativas em qualquer documento carregado pela aplicação. Gramáticas contidas em escolhas de menu não podem especificar um escopo.

Algumas vezes um formulário pode precisar ter algumas gramáticas ativas por todo o documento, e outras gramáticas que devem ficar ativas apenas no formulário. Uma razão para fazer isto é minimizar os problemas de sobreposição de gramáticas. Para fazer isto, cada elemento <grammar> e <dtmf> pode ter seu próprio escopo:

<form scope="document">
<grammar> … </grammar>
<grammar scope="dialog"> … </grammar>
</form>

3.12.3. Ativação de gramáticas

Quando o interpretador espera por uma entrada como resultado de um campo <field> visitado, as seguintes gramáticas ficam ativas:

- gramáticas para aquele campo, incluindo gramáticas contidas em links;
- gramáticas para o formulário, incluindo gramáticas contidas em links;
- gramáticas contidas em links no documento, e gramáticas para menus e outros formulários no documento que são determinados escopo de documento;
- gramáticas contidas em links na aplicação raiz do documento, e gramáticas para menus e formulários na aplicação raiz do documento que têm escopo de documento.

No caso de uma entrada coincidir com mais de uma gramática, a lista acima define a ordem de precedência. Se a entrada coincide com mais de uma gramática com a mesma precedência, a precedência é determinada pela ordem do documento. Menus comportam-se em relação a ativação da gramática como os seus formulários equivalentes.

Se o item de formulário é modal (ex: o atributo "modal" definido como verdadeiro), todas as gramáticas exceto a sua própria são desligadas enquanto espera por uma entrada. Se a entrada coincide com uma gramática no formulário ou menu diferente do formulário ou menu atuais, o controle passa para outro formulário ou menu. Se a coincidência faz o controle deixar o formulário atual, todos os dados do formulário atual são perdidos.

3.13. Link

O elemento <link> tem uma ou mais gramáticas, que são direcionadas ao elemento contendo o atributo next. Quando uma dessas gramáticas é escolhida, o link é ativado, e uma das duas formas:

- Mudanças para um novo documento ou diálogo, ou
- Lança um evento (<throw>)

Por Exemplo, este link é ativado quando você diz "esportes" ou tecla "2".

```
<link next="http://.../main.vxml">
<grammar type="application/x-jsgf">esportes</grammar>
<dtmf>2</dtmf>
</link>
```

Este link leva para um diálogo determinado dinamicamente no documento atual:

```
<link expr="'#' + document.helpstate">
<grammar type="application/x-jsgf">ajuda</grammar>
</link>
```

O elemento <link> pode ser um filho de <vxml> , <form>, ou um item de formulário. Um link ao nivel de documento tem gramáticas que são ativas por todo o documento. Um link ao nivel de formulario tem gramáticas ativas enquanto o usuário está naquele formulário. Se uma aplicação documento-raiz tem um link nivel documento, sua gramática fica ativa não interessa qual documento da aplicação está sendo executado. Se a execução é num item de formulário, então a gramática do link no formulário ou no nível de documento não fica ativa. Tambem podemos definir um link que, quando ativado, lança um evento ao invés de ir a um novo documento. Este evento é lançado no local atual na execução, e não no local onde o link especifica. Por exemplo, se o usuário ativa o link, um evento de ajuda é lançado no item de formulário que o usuário está visitando:

```
<link event="help">
<grammar type="application/x-jsgf">ajuda | não entendo </grammar> </link>
```

Os atributos de <link> são:

- next: A URL para ir. Essa URL é um documento (com uma âncora para especificar o começo do diálogo), ou um diáologo no documento atual.

- expr: Como o "next", exceto pelo fato da URL ser determinada dinamicamente pela avaliação da expressão ECMAScript fornecida.
- event: O evento a ser lançado quando o usuário escolhe uma das opções. Note que apenas um dos "next", "expr" ou "event" pode ser especificado.

3.14. Variáveis e expressões

Váriaveis VoiceXML são equivalentes a variáveis ECMAScript. A convenção de nomenclatura é como no ECMAScript, mas nomes começados com "underline" (_) são reservados para uso interno.

3.14.1. Declarando variáveis

Variáveis são declaradas pelo elemento <var>:

```
<var name="phone"/>
<var name="pi" expr="3.14159"/>
<var name="city" expr="São Carlos"/>
```

Elas também são declaradas por itens de formulário:

```
<field name="tickets" type="number">
<prompt>Quantos bilhetes você vai comprar?</prompt>
</field>
```

Variáveis declaradas sem um valor inicial explícito são inicializadas pelo ECMAScript com valor indefinido. Variáveis devem ser declaradas antes de serem usadas.

Num formulário, as variáveis declaradas por <var> e aquelas declaradas por itens de formulário são inicializadas quando se entra no formulário. As inicializações acontecem na ordem do documento, por exemplo:

```
<form id="test">
<var name="one" expr="1"/>
<field name="two" expr="one+1" type="number"></field>
<var name="three" expr="two+1"/>
<field name="go_on" type="boolean">
<prompt>Diga sim ou não para continuar</prompt>
```

```
</field>
<filled>
<goto next="#tally"/>
</filled>
</form>
```

Quando o usuário visita este form, a inicialização do formulário primeiro declara a variável "one" e define seu valor para "1". Então declara a variável item de campo "two" e dá o valor "2" para ela. A lógica de inicialização declara a variável "three" e atribui valor "3". A interpretação do algoritmo do formulário entra no laço de interpretação que começa do campo "go_on".

3.14.2. Escopos de Variáveis

Variáveis podem ser declaradas nos seguintes escopos:

- session: são variáveis somente leitura que pertencem a toda a sessão do usuário. Elas são declaradas e definidas pelo interpretador do contexto. Novas variáveis de sessão não podem ser declaradas por documentos VoiceXML.

- application: são declaradas por elementos <var> que são filhos do documento raiz da aplicação <vxml>. São inicializadas quando o documento raiz é carregado. Elas existem enquanto o documento raiz está carregado, e são visíveis pelo documento raiz e qualquer outro documento carregado.

- document: são declaradas com o elemento <var> que são filhos do elemento <vxml>. Elas são inicializadas quando o documento é carregado, existem enquanto o documento está carregado, e são visíveis somente para este documento.

- dialog: Cada diálogo (<form> ou <menu>) tem um escopo de diálogo que existe enquanto o usuário está visitando aquele diálogo. Variáveis de diálogo são declaradas pelos elementos <var> filhos de <form>, por <var> dentro do contexto executável (conteúdo do <block>), e por vários itens de formulário. Os elementos <var> filhos de <form> são inicializados quando o formulário é visitado pela primeira vez. Os elementos <var> dentro do contexto executável são inicializados quando o contexto é executado. As

variáveis de itens de formulários são inicializadas quando o item de formulário é controlado.

- (anonymous): Cada elemento <block>, <filled> e "catch" define um novo escopo anônimo para variáveis declaradas naquele elemento.

O seguinte diagrama mostra a hierarquia:

session
^
application
^
document
^
dialog
^
(anonymous)

3.14.3. Referenciando Variáveis

Variáveis são referenciadas nos atributos "cond" e "expr":

```
<if cond="city == 'LA'">
<assign name="city" expr="'Los Angeles'"/>
<elseif cond="city == 'Philly'"/>
<assign name="city" expr="'Philadelphia'"/>
<elseif cond="city =='Constantinople'"/>
<assign name="city" expr="'Istanbul'"/>
</if>
<assign name="var1" expr="var1 + 1"/>
<if cond="i > 1">
<assign name="i" expr="i-1"/>
</if>
```

A expressão usada em "cond" e "expr" é ECMAScript. Note que os operadores condicionais ">", "<", ">=", "<=", e "&&" não devem ser usadas em XML.

Você pode prefixar uma referencia com o nome do escopo para resolver ambigüidade. Por exemplo, para guardar o valor de uma variável para usar depois num documento:

<assign name="document.ssn" expr="dialog.ssn"/>

Se o documento raiz da aplicação tem uma variável x, ela é referida como application.x nos documentos não-raiz ou document.x no documento raiz.

3.14.4. Variáves de Sessão Padrão

- session.telephone.ani - Automatic Number Identification: Esta variável fornece o resultado do serviço de Identificação Automática de Número que fornece ao receptor de uma chamada telefônica o número do telefone que chama. Esta informação é fornecida apenas se o serviço é suportado, senão é indefinida.
- session.telephone.dnis - Dialed Number Identification Service: Esta variável fornece o resultado do Serviço de Identificação de Número Discado que identifica para o receptor de uma chamada o número que o discador discou. Esta informação é fornecida somente se o serviço for suportado, senão é indefinida.
- session.telephone.iidigits - Information Indicator Digit: Esta variável fornece informação sobre a linha de origem (ex: telefone público, celular, prisão) do discador. Telecordia publica a lista completa de Digitos II na seção 1 de cada "Lista Telefônica Local". Esta informação é fornecida somente se o serviço for suportado, senão é indefinida.
- session.uui - User to User Information: Esta variável retorna informações suplementares fronecidas como parte de uma ligação ISDN de conferência. Esta informação é fornecida somente se o serviço for suportado, senão é indefinida.

3.14.5. Variáves de Aplicação Padrão

- application.lastresult$ - essa variável guarda a informação sobre o ultimo reconhecimento que ocorreu na aplicação. É um array de elementos onde cada elemento, application.lastresult$[i], representa um possível resultado com os seguintes membros:
- application.lastresult$[i].confidence – o nível de confiança de reconhecimento entre 0.0 -1.0. É um valor que depended a plataforma de ASR

- application.lastresult$[i].utterance – uma string das palavras que foram reconhecidas (e.g. "five hundred thirty" or "5 hundred 30" or even "530").
- application.lastresult$[i].inputmode – guarda o modo de entrada (voz ou dtmf)
- application.lastresult$[i].interpretation – o slot da gramática que foi preenchido

3.15. Eventos

A plataforma lança eventos quando o usuário não fala, não responde com clareza, requesita ajuda, etc. O interpretador lança eventos se ele acha erro de semântica num documento VoiceXML, ou quando ele encontra um elemento throw. Eventos são identificados por sequências de strings.

Cada elemento em que o evento ocorre tem um conjunto de elementos de captura, que incluem:

- <catch>
- <error>
- <help>
- <noinput>
- <nomatch>

Um elemento herda os elementos de captura de cada elemento ancestral. Se um campo <field>, por exemplo, não contém um elemento de captura, mas seu formulário contém, o elemento de captura do formulário é usado. Deste modo, tratamentos comuns de eventos podem ser especificados em qualquer nível, e se aplicam para todos os descendentes.

3.15.1. Throw (Lançamento)

O elemento <throw> lança um evento. Estes podem ser alguns predefinidos:

- **<throw event="nomatch"/>**
- **<throw event="telephone.disconnect.hangup"/>**

ou eventos definidos pela aplicação:

- **<throw event="com.att.portal.machine"/>**

3.15.2. Catch (Captura)

O elemento "catch" associa uma captura com um documento, diálogo ou item de formulário. Ele contém conteúdo executável.

```
<form id="launch_missiles">
<field name="password">
<prompt>Qual a palavra-código?</prompt>
<grammar>abracadabra</grammar>
<catch event="nomatch noinput" count="3">
<prompt>Alarme: Violação de segurança!</prompt>
<submit next="app_on" namelist="user_id"/>
</catch>
</field>
<block>
<goto next="#get_city"/>
</block>
</form>
```

Os atributos de <catch> são:

- event: O evento ou eventos para serem capturados.
- count: A ocorrência do evento (o padrão é 1). O count permite você controlar diferentes ocorrências no mesmo evento diferentemente. Cada item de formulário e <menu> mantém um contador para cada evento que ocorre quando está sendo visitado; estes contadores são resetados cada vez que o <menu> ou item de formulário são re-visitados.
- cond: Uma condição opcional para testar se o evento pode ser capturado por este elemento. O padrão é "true".

3.15.3. Notação Curta

Os elementos <error>, <help>, <noinput>, <nomatch> são "shorthands" para tipos muito comuns de elementos <catch>.

O elemento <error> é curto para <catch event="error"> e captura todos os eventos do tipo "error":

<error>
Mensagem de erro…
<exit/>
</error>

O elemento <help> é uma abreviação para <catch event="help">:

<help>mensagem de ajuda.</help>

O elemento <noinput> abrevia <catch event="noinput">:

<noinput>Não ouvi... tente novamente</noinput>

E o elemento <nomatch> é curto para <catch event="nomatch">:

<nomatch>Não entendi, o que vc disse?</nomatch>

Os atributos destes elementos são:

- count: O contador do evento (como em <catch>)
- cond: Uma condição opcional para testar se o evento pode ser capturado por este elemento. O padrão é "true".

3.15.4. Seleção do Elemento Catch

Um elemento herda os elementos de captura de cada elemento ancestral. Quando um evento é lançado, o escopo no qual o evento é controlado é examinado para achar o evento de captura melhor qualificado, de acordo com o seguinte algoritimo:

1) Forma uma lista ordenada de capturas contendo todas as capturas em todos os escopos (item e formulário, formulário, documento raiz da aplicação), ordenado primeiro por escopo (começando com o escopo atual), e então continua com cada escopo por ordem do documento.
2) Remove da lista todas as capturas que pertencem a um evento que não combinam com o evento lançado ou aquelas "cond" avaliados como "false".

3) Acha a "contagem correta": a maior contagem entre os elementos de captura que continuam na lista menor que ou igual a contagem atual.
4) Seleciona o primeiro elemento na lista com a "contagem correta".

O nome de um evento lançado coincide com o nome do evento do elemento de captura se coincide com um dos eventos ou seu prefixo. Uma coincidência prefixo ocorre quando o elemento de captura tem um prefixo em comum com o nome do evento sendo lançado. Por exemplo:

<catch event="telephone.disconnect">

irá coincidir o prefixo com o evento telephone.disconnect.transfer.

3.15.5. Tipo de Eventos

Existem eventos predefinidos e eventos definidos pela aplicação. Os eventos também são subdivididos em eventos planos (coisas que acontecem normalmente), e eventos de erros (ocorrências anormais). A convenção de nomenclatura permite múltiplos níveis de granularidade.

Os eventos predefinidos são:

- cancel: O usuário requer o cancelamento da escolha atual
- telephone.disconnect.hangup: O usuário desliga
- telephone.disconnect.transfer: O usuário se transferiu pra outra linha e não vai retornar
- exit: O usuário pede pra sair
- help: O usuário pede por ajuda
- noinput: O usuário não respondeu no timeout definido
- nomatch: O usuário entrou com alguma fala, mas não foi reconhecida

Os erros predefinidos são:

- error.badfetch: Um "fetch" falhou. Este pode ser resultado de um documento não encontrado, por exemplo.

- error.semantic: Um erro em tempo de execução foi encontrado pelo documento VoiceXML. Ex: divisão por 0, variável não-declarada
- error.noauthorization: O usuário não é autorizado a executar a operação solicitada.
- error.unsupported.format: O recurso solicitado tem um formato que não é suportado pela plataforma. Ex: um formato de gramática não suportado.
- error.unsupported.element: A plataforma não suporta o elemento determinado. Ex: a plataforma não suporta o elemento <record>.

Tipos de erros específicos da aplicação devem seguir o seguinte formato:

- error.com.mot.mix.noauth: Acessa o perfil pessoal não é autorizado.
- error.com.ibm.portal.restricted: O documento que foi tentado acessar é de fonte restrita.

"Catches" podem capturar eventos específicos (cancel) ou todos que compartilham um prefixo (error.unsuported).

3.16. Prompt

O elemento prompt controla a saída da fala sintetizada e áudio pregravado. O prompt, assim como num sistema operacional, é um sinal de aguardo de instrução, que no caso do VoiceXML é um som.

Conceitualmente, prompts são instantaneamente colocados em fila para reprodução, quando a interpretação procede assim que o usuário prover uma entrada. Neste momento, os Prompts são reproduzidos, e o sistema espera por outra entrada do usuário. Uma vez que outra entrada é recebida pelo subsistema de reconhecimento de fala (ou o Reconhecedor DTMF), a interpretação procede.

O Prompt tem os seguintes atributos:

- bargein: Controla se um usuário pode interromper um prompt. O padrão é ativo.
- cond: uma condição para tocar o prompt.
- count: O número de prompts diferentes que se pode emitir se o usuário está fazendo algo repetidamente. Se este parâmetro for omitido, o padrão é 1.

- timeout: A espera que será usada para a seguinte entrada do usuário. O padrão é não haver timeout.

Pode-se usar tags para modificar a forma com que um prompt é falado.

- <break> - Especifica uma pausa na produção de fala.

Msecs = número em milisegundos de pausa.

Size = tempo de pausa relativo. Valores possíveis são: none, small, medium ou large (nenhum, pequeno, médio ou grande).

- - Identifica o texto incluso como um tipo particular.

Type = possíveis valores são sentence ou paragraph (sentença ou parágrafo).

- <emp> - Especifica que o texto deve ser falado com ênfase.

Level = Especifica o nível de ênfase. Possíveis valores são: forte, moderado (padrão), nenhum ou reduzido.

- <pros> - Especifica parâmetros como velocidade, volume, tom, etc.

rate = Especifica a velocidade da fala.

vol = Especifica o volume de saída.

pitch e range = Especificam o tom.

- <sayas> - Especifica a maneira que uma palavra ou frase é falada.

Letra a letra, leitura de números, datas, valores, etc.

- <audio> - Especifica um arquivo de áudio que deve ser executado.

src = o caminho e o nome do arquivo do áudio.

3.17. Property

O elemento Property fixa um valor de propriedade. São usadas propriedades para fixar valores que afetam comportamento de plataforma, como o processo de reconhecimento, intervalos, política de caching, etc.

Propriedades para o Java Speech

- APIconfidencelevel: O nível de confiabilidade do reconhecimento de fala. Vai de 0.0 à 1.0. O valor padrão é 0.5. Um valor de 0.0 define uma confiança mínima para um reconhecimento, e um valor de 1.0 requer confiança máxima.

- sensitivity: Fixe o nível de sensibilidade. Um valor de 1.0 que é altamente sensível para uma entrada fraca. Um valor de 0.0 é menos sensível para ambientes com ruído. O padrão é 0.5.
- speedvsaccuracy: Uma sugestão que especifica o equilíbrio desejado entre velocidade e precisão. Um valor de 0.0 tem reconhecimento mais rápido. Um valor de 1.0 tem melhor precisão. O padrão é 0.5.
- completetimeout: Um timeout para completar uma frase de uma gramática. O padrão depende da plataforma.
- incompletetimeout: Um timeout quando uma gramática não é encontrada. O padrão depende da plataforma.

Propriedades DTMF

- interdigittimeout: O valor de intervalo de entrada de dígitos quando se usa o reconhecimento DTMF.
- termtimeout: O intervalo de espera quando se usa o reconhecimento DTMF.
- termchar: Especifica o caracterer que finaliza uma entrada.

3.18. Param

O elemento Param é usado para especificar valores que são passados a objetos. São seu atributos:

- name: O nome do parâmetro.
- expr: Uma expressão que computa o valor associado ao nome.
- value: O valor associado ao nome.
- valuetype: O tipo do valo associado. Pode ser um dado ou uma referência.
- type: O tipo MIME do dado provido pela referência.

```
<object name="debit" classid="method://credit_card/gather_and_debit“
data="http://.../xyz.jar"/>
<param name="amount“ expr="document.amt"/>
<param name="vendor“ expr="vendor_num"/>
<param name="application_id" value="ADC5678-QWOO"/>
```

<param name="authentic_server“ value="http://...“
valuetype="ref“ type="text/plain"/>
</object>

3.19. Goto

O elemento Goto é usado para:

- transição para outro campo no form atual,
- transição para outro diálogo no documento atual, ou
- transição para outro documento.

São atributos do Goto:

- next: Especifica a URI a qual será feita a transição.
- expr: Uma expressão de ECMAScript para a URI.
- nextitem: O nome do próximo campo que deve ser visitado dentro do form atual.
- expritem: Uma expressão de ECMAScript para o próximo campo dentro do form atual.

<goto nextitem="confirm"/>
<goto expritem="(type==true)?'confirm' : 'reject'"/>

<goto next="#another_dialog"/>
<goto expr="'#' + 'another_dialog'"/>

<goto next="http://.../reserve_eat"/>
<goto next="./…/#wants_vegan/>

4. XML EVENTS

Um evento XML é a representação de alguma ocorrência, tal como um clique, ou um erro aritmético, ou qualquer outra possibilidade que esteja associada com um elemento em um documento XML.

Um evento consiste em três porções:

- Um action é alguma maneira de responder a um evento
- Um handler é alguma especificação para tal action, por exemplo, usando scripting ou algum outro método
- Um listener é um mecanismo para unir um handler a um evento que alveja algum elemento em um documento.

Quando um evento ocorre, ele é tratado por um handler. O handler toma a ação necessária (por exemplo transferindo o controle a algum script) depois que o evento ocorreu. O XML events é uma linguagem de marcação que usa o modelo padrão de eventos Web para criar a interação multimodal com o usuário.

4.1. O que o W3C diz

O mais importante a saber sobre XML events é que ele usa o mesmo mecanismo de eventos do HTML, porém, escrito de uma forma diferente.

Considere um exemplo bem simples em HTML:

<input type="submit" onclick="verify(); return true;">

Neste exemplo o elemento <input> (ou qualquer um de seus elementos filho) captura o evento click e faz com que o código inserido no atributo de onclick seja executado.

Dizemos "ou qualquer um de seus elementos filho" porque em casos como:

A very **nice place to go**

ou ainda

****More****

O código em onclick será executado mesmo que o clique seja nos elementos ou . Em casos como este, designamos o elemento clicável por *target (*alvo) e o elemento que responde pelo evento por *observer* (muito embora o mais frequente é que target e observer sejam o mesmo elemento).

Existem alguns problemas com a maneira pela qual o HTML faz o relacionamento entre estas três coisas:

- o nome do evento é estabelecido pela linguagem de script, em lugar de ser um parâmetro que possa ser introduzido em um atributo capaz de definir um evento como um atributo onflash para um evento denominado flash.
- o nome do evento é específico ao hardware tal como é click, quando na verdade não interessa ***como*** um botão é ativado mas sim, **se** ele foi ativado.
- É restrito ao uso de uma só linguagem de script (uma vez que você não pode ter dois atributos chamados onclick, um para JavaScript e um para VB)
- Handlers de eventos e marcação são associados, não há como separá-los.

XML Events especifica as diversas formas de relacionamento entre o evento o observador e o handler. Um código como abaixo:

<input type="submit" onclick="validate(); return true;">

teria seu equivalente assim::

```
<input type="submit">
  <script ev:event="DOMActivate" type="text/javascript">
    validate();
  </script>
</input>
```

O exemplo mostra que o elemento <script> é um handler para o evento DOMActivate (que usamos preferencialmente a click, porque botões podem ser ativados de maneiras diferentes e não somente por um clique) e na ausência de qualquer outra informação, o elemento pai é o observador (neste caso o <input>).Notar que handlers devem ser avaliados somente com a ocorrência do evento e não por ocasião do carregamento do documento, como ocorre com <script> no HTML4.

Este conceito agora permite a especificação de handlers para diferentes linguagens de script:

```
<input type="submit">
  <script ev:event="DOMActivate" type="text/javascript">
    ...
```

```
  </script>
  <script ev:event="DOMActivate" type="text/vbs">
    ...
  </script>
</input>
```

e/ou eventos diferentes:

```
<input type="submit">
  <script ev:event="DOMActivate" type="text/javascript">
    ...
  </script>
  <script ev:event="DOMFocusIn" type="text/javascript">
    ...
  </script>
</input>
```

4.2. Outras maneiras de especificar relacionamentos

Tal como ocorre com o HTML, existem outras maneiras de especificar o relacionamento evento-observador-handler. A informação pode ser colocada no handler, no observador, ou em um elemento <listener>. Qualquer que seja o método usado, deverá especificar sempre as três partes: assim se você usar um handler deverá ter um evento e um observador, se você usar um observador deverá ter um evento e um handler, usando <listener> deverá especificar os três.

A razão para isso é a de facilitar o gerenciamento do documento. Você poderá separar todos os scripts do corpo do documento. Isto evitará uma duplicação dos handlers.

4.2.1. Especificando o relacionamento no handler

Quando você move o handler para alguma outra parte do documento, especifica o relacionamento ali (algo como o uso do atributo for no elemento <script> em HTML):

```
<script ev:observer="button" ev:event="DOMActivate" type="text/javascript">

  validate();
</script>
...
<input type="submit" id="button"/>
```

4.2.2. Especificando o relacionamento com <listener>

Quando você move o handler para alguma outra parte do documento, especifica o relacionamento em outro lugar com o elemento <listener> (isto possibilita o uso do mesmo handler para mais de um observador):

```
<ev:listener observer="button" handler="#validator" event="DOMActivate"/>
...
<script id="validator" type="text/javascript">
    validate();
</script>
...
<input type="submit" id="button"/>
```

(Notar que neste caso o elemento <listener> e não o atributo carrega o prefixo ev:)

4.2.3. Especificando o relacionamento no observador

E finalmente, você pode especificar o relacionamento no observador:

```
<script id="validator" type="text/javascript">

    validate();
</script>
...
<input type="submit" ev:handler="#validator" ev:event="DOMActivate"/>
```

4.3. Detalhes de refinamento

Existem alguns detalhes que podem ser úteis.

4.3.1. phase ("capture" | "default")

Como já foi dito os handlers de eventos podem ser invocados por um elemento ou por qualquer um de seus filhos e desta forma é possível, por exemplo, esperar por cliques em qualquer coisa dentro de um parágrafo:

```
<p>
   <script ev:event="DOMActivate" type="...">
      ...
   </script>
   ...
   <a href="...">...</a>

   ...
   <a href="...">...</a>
   ...
</p>
```

Contudo, como os elementos <a> também possuem handlers neles mesmo, o atributo phase é usado com a finalidade de especificar se determinado handler deve ser ativado antes ou depois de um handler em posição mais baixa na árvore de handlers. Normalmente handlers em posição mais baixa na árvore são ativados em primeiro, contudo com o uso do atributo ev:phase="capture" pode-se fazer um handler em posição mais alta na árvore ser executado antes para um mesmo evento.

```
<p>
   <script ev:event="DOMActivate" ev:phase="capture" type="...">
         ...
   </script>
   ...
   <a href="...">...</a>
   ...
   <a href="...">...</a>

   ...
</p>
```

4.3.2. target

Se você define um elemento particular como alvo (target), então podemos estabelecer que aquele elemento unicamente dispara o evento:

```
<p>
   <script ev:event="DOMActivate" ev:target="img1" type="...">

      ...
   </script>
   ...
   <a href="..."><img id="img1" ... /><img id="img2" ... /></a>
</p>
```

4.3.3. propagate ("stop" | "continue")

Handlers de evento são resolvidos na seguinte ordem: primeiro os handlers com ev:phase="capture", começando pelo mais alto na árvore e continuando para baixo pelos alvos do elemento pai, a seguir os demais handlers do elemento alvo em direção ao topo da árvore

Se algum handler tiver ev:propagate="stop" nenhum outro handler que se segue será evocado para o evento corrente.

```
<body>
```

```
   <script ev:event="DOMActivate" ev:propagate="stop" ev:phase="capture"
type="...">
      <!—Este script é o único que irá responder ao DOMActivate -->
      ...
   </script>
   ...
</body>
```

Acrescente-se ainda que se houver mais de um handler para um mesmo evento em um elemento será indefinida a ordem em que eles serão executados.

4.3.4. defaultAction ("cancel" | "perform")

Muitos dos eventos tem uma ação default *default action* que é executada depois que todos os handlers de eventos da árvore são efetuados. Por exemplo; um clique em um botão ativa o botão, um clique <a> ou em um de seus filhos ativará um link . É possível inibir a ação default com o atributo ev:defaultAction="cancel".

```
<input type="submit" ...>
   <script ev:event="DOMActivate" ev:defaultAction="cancel" type="...">

      <!—Este script sera ativado, mas o default action não sera executado-->
      ...
   </script>
</input>
```

Isto é o que faz onclick="...; return=false;" no HTML4. Se você pretende fazer isto em eventos XML, você deverá chamar de forma apropriada o método DOM preventDefault.

5. Desenvolvimento de Interfaces Multimodais usando X+V

5.1. Tags XHTML+VoiceXML

A linguagem de marcação X+V é composta pelos seguintes elementos e atributos.

5.1.1. <sync>

O elemento <sync> proporciona a sincronização dos dados fornecidos via entrada de fala ou visual. Ele é uma combinação do elemento de formulário <input> ao <field> do VXML com um atributo de id. Isso significa:

1) os resultados de uma entrada por voz são retornados para o <field> VXML e para o elemento <input> do XHTML.

2) os dados entrados pelo teclado dentro do elemento <input> são atualizados no <field> do VXML e no <input> do XHTML ao mesmo tempo.

Sintaxe:

<xv:sync xv:input="string" xv:field="URI+#+ID" xv:html-form-id="#+ID"/>

Atributo	Descrição
Input	Nome de um input field XHTML.
Field	referência URI para um field ID dentro do Form VXML.
html-form-id	Uma referência para o ID do form XHTML.

Parents <head>

Children - Nenhum

Exemplo:

```
<?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.w3.org/2002/xhtml+voice">
<head><title>Sync Example</title>
<xv:sync xv:input="in1" xv:field="#result"/>
<vxml:form id="topform">
        <vxml:field name="result xv:id="result">
                <vxml:prompt>Say a name</vxml:prompt>
                <vxml:grammar src="result.gram"/>
        </vxml:field>
</vxml:form>
</head>
        <body ev:event="load" ev:handler="#topform">
                <h1>Sync example</h1>
                <form action="cgi/result.cgi">
                        Result: <input type="text name="in1"/>
                </form>
        </body>
</html>
```

5.1.2. <cancel>

O elemento <cancel> permite ao programador cancelar o diálogo que está sendo executado.

Sintaxe:

<xv:cancel id="string" xv:voice-handler="URI+#+ID"/>

Atributo	Descrição
Id	Identificador único de documento
voice handler	Uma referencia URI para um id de <form> VXML.

Parents <head>

Children Nenhum

Remarks

O atributo id é obrigatório. O atributo opcional voice-handler referencia o atributo id de um handler de <form> de VXML. Se o voice-handler for omitido, o diálogo corrente de voz é cancelado.

Exemplo

```
<?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.w3.org/2002/xhtml+voice">
<head>
        <title>Sync Example</title>
<xv:sync xv:input="in1" xv:field="#result"/>
<xv:cancel id="can1" voice-handler="#topform"/>
        <vxml:form id="topform">
                <vxml:field name="result xv:id="result">
                        <vxml:prompt>Say a name</vxml:prompt>
                        <vxml:grammar src="result.gram"/>
                </vxml:field>
        </vxml:form>
        </head>
                <body ev:event="load" ev:handler="#topform">
                <h1>Sync example</h1>
                <form action="cgi/result.cgi">
                        Result: <input type="text name="in1"/><br/>
                        <input type="reset" ev:event="click"
```

```
                ev:handler="#can1"/>
            </form>
            </body>
</html>
```

5.1.3. <listener>

XHTML+Voice estende o XHTML com o elemento <listener> do XML Events e seus atributos. Os atributos do <listener> são usados para ativar os voice handlers.

O elemento <listener> e seus atributos pertencem ao namespace do XML Events :

xmlns:ev="http://www.w3.org/2001/xml-events"

5.2. Instalação do Eclipse 3.1 com Plugin Multimodal da IBM

Pré-Requisitos

- Processador Intel® Pentium® III 500 MHz ou equivalente. (Mínimo)

- 256 MB RAM. (512 MB recomendado)

- Configurações do adaptador de vídeo: 256 cores e resolução 800 x 600 (1024 x 768 resolução recomendada)

Arquivos Necessários

- Eclipse com WTP: wtp-all-in-one-0.7.1-win32.zip
Link Download: http://www.eclipse.org/webtools

- Plugin VTP: vtp-M1.zip
Link Download: http://www.eclipse.org/downloads/download.php?file=/technology/vtp/vtp-M1.zip

- Plugin Multimodal da IMB: com.ibm.mmtp_0.0.9.bin.dist.zip.zip
Link Download: http://www.alphaworks.ibm.com/tech/mmtp/download

Instalação Passo-a-Passo

Figura 4. Arquivos necessários

Faça o Download dos arquivos necessários.

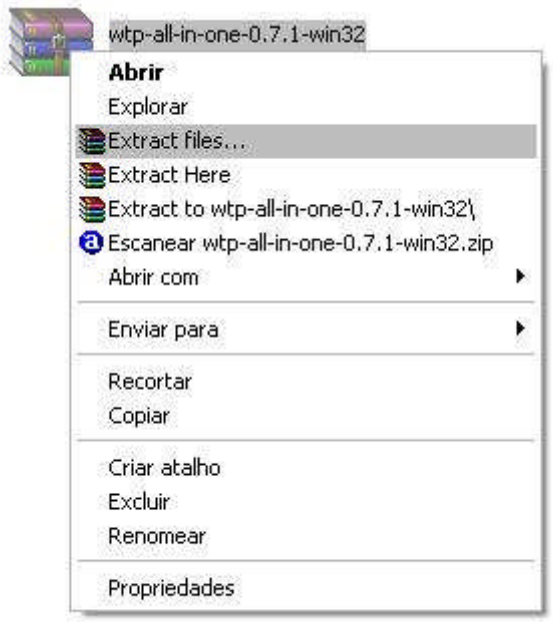

Figura 5. Descompactando o WTP

Descompacte o arquivo wtp-all-in-one-0.7.1-win32.zip em uma pasta qualquer, como por exemplo C:\MultimodalKit\ .

Figura 6. Descompactando o VTP

Descompacte o arquivo vtp-M1.zip na mesma pasta, nesse exemplo C:\MultimodalKit\ .

Figura 7. Tela de Confirmação

Aparecerá a seguinte tela, na qual irá substituir alguns arquivos já existentes. Click em Yes to All.

Figura 8. Descompactando o plugin para o Eclipse

Descompacte o arquivo com.ibm.mmtp_0.0.9.bin.dist.zip na mesma pasta acrescentando a pasta \eclipse, nesse exemplo C:\MultimodalKit\eclipse\.

Pronto, seu Eclipse 3.1 já possui todos o plugins necessário para criação do arquivo X+V.

Criando um Arquivo X+V

Agora vamos lhe mostrar como criar um arquivo X+V no Eclipse 3.1, para que a partir desse ponto, você possa programá-lo. Primeiro é necessário criar um Projeto e depois criar o arquivo X+V dentro desse Projeto. Veja abaixo:

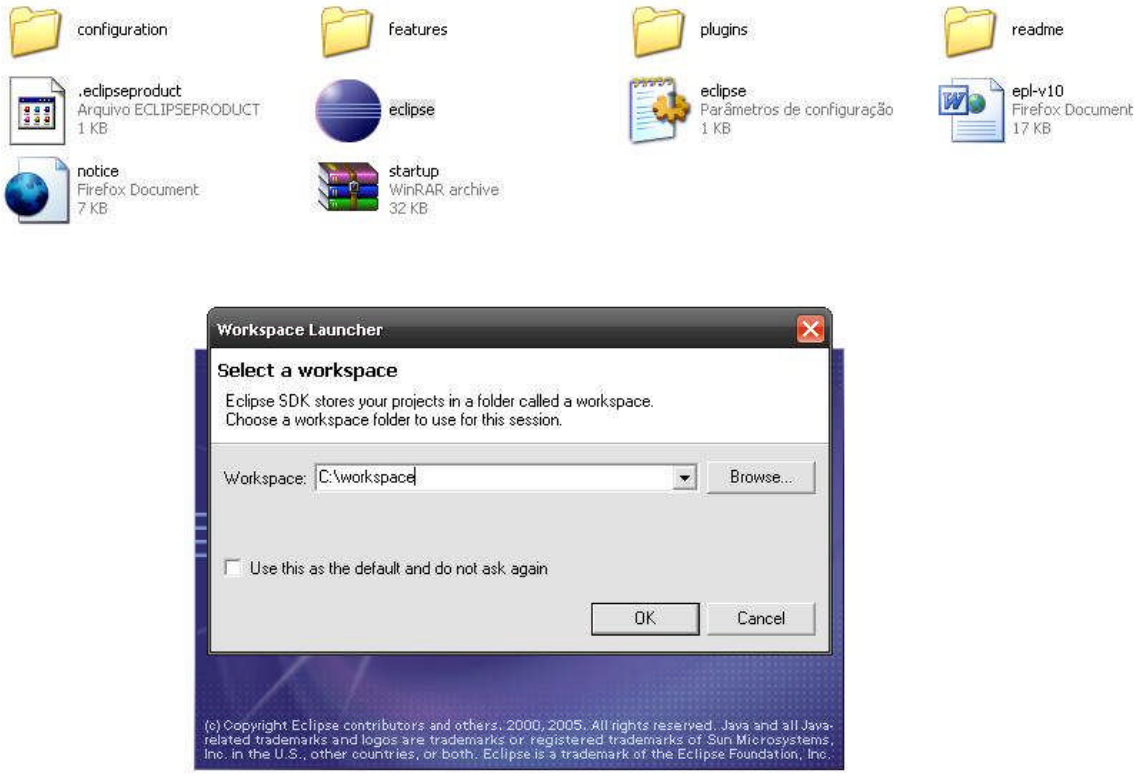

Figura 9. Criando o workspace

Execute o arquivo eclipse.exe e na tela inicial defina o local onde deverá ser salvo os projetos criados, nesse exemplo em C:\workspace.

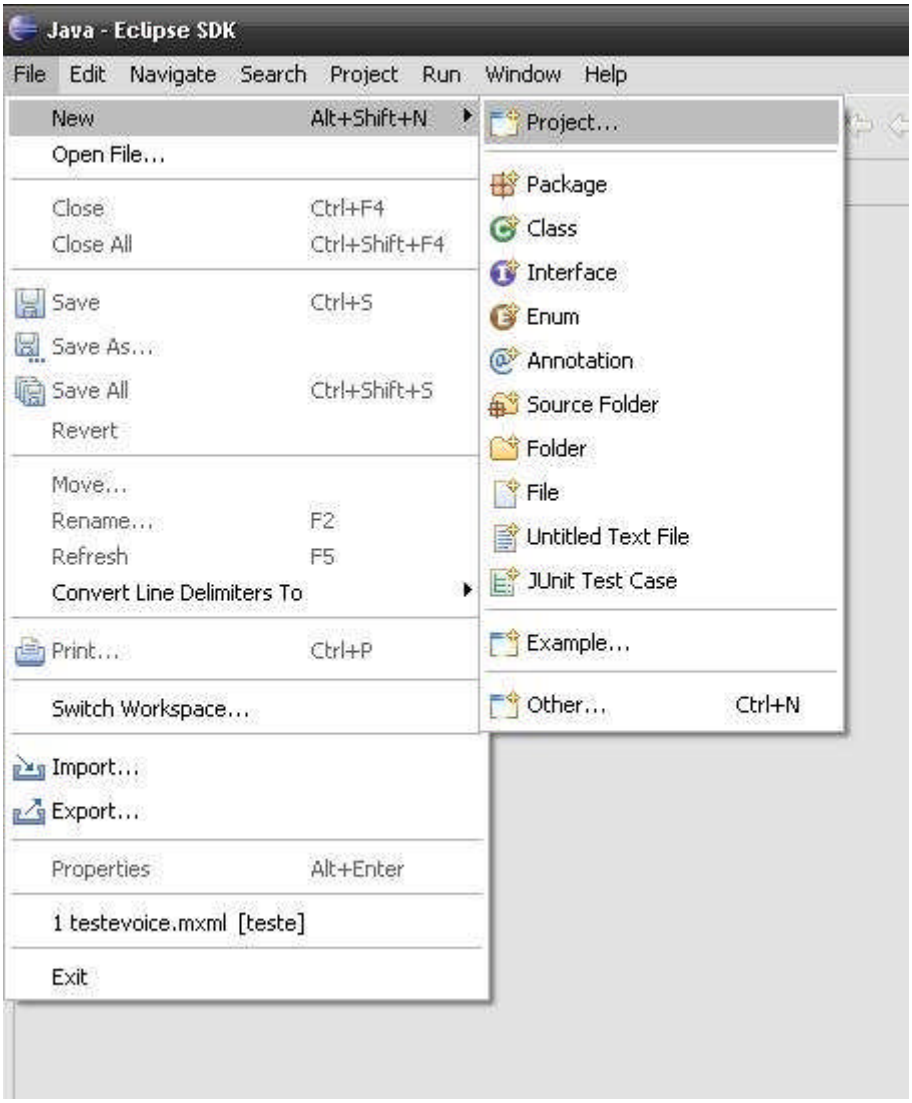

Figura 10. Criando um projeto

Primeiro precisamos criar um Projeto Novo, para isso Click em File > New > Project.

Figura 11. Selecionando o Wizard

Selecione Java Project e Click em Next.

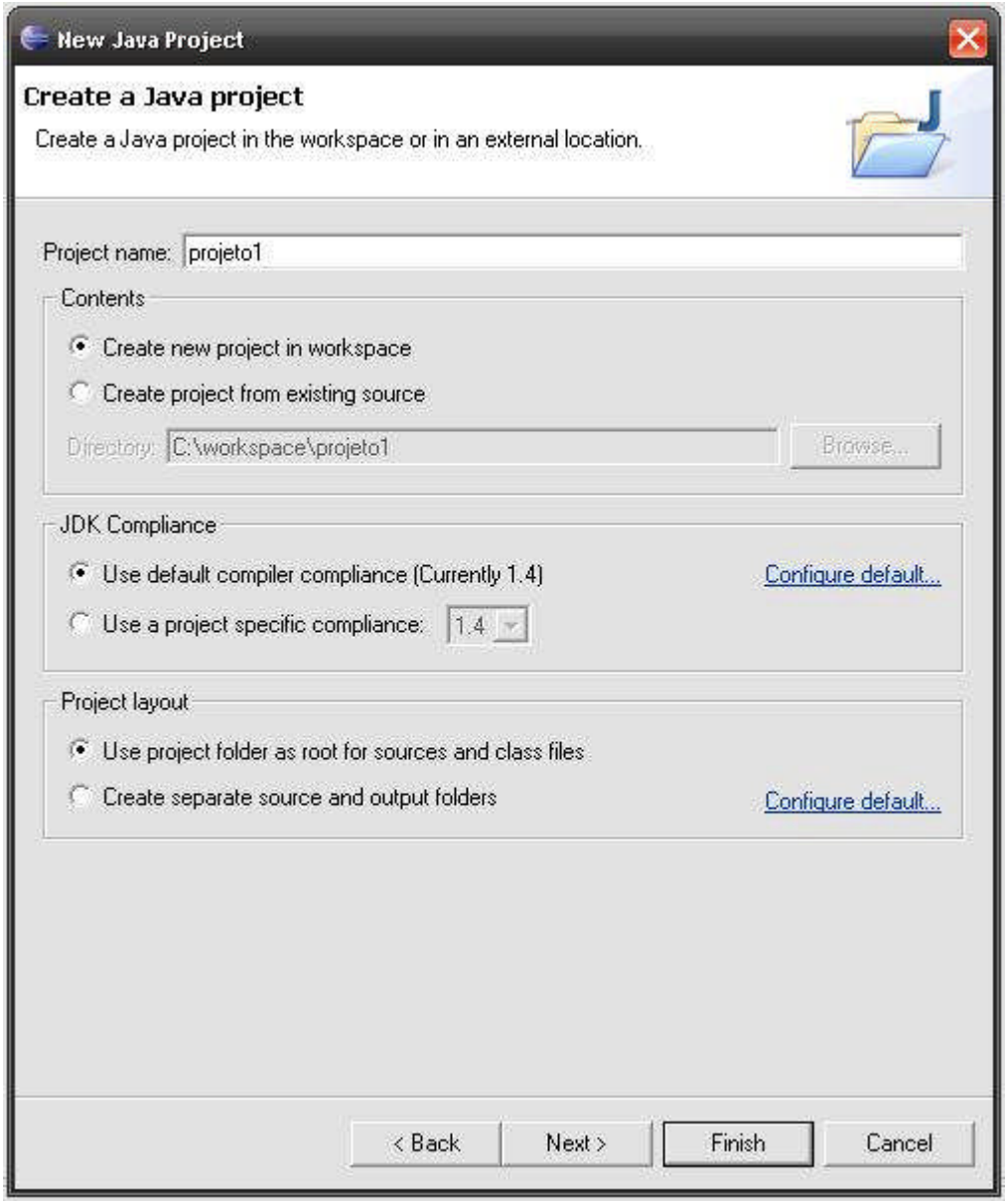

Figura 12. Nomeando o projeto

Dê nome ao projeto, nesse exemplo projeto1, e Click em Finish. Pronto, você acaba de criar um projeto no Eclipse 3.1.

Agora precisamos criar o arquivo X+V dentro do Projeto criado. Faremos da seguinte forma:

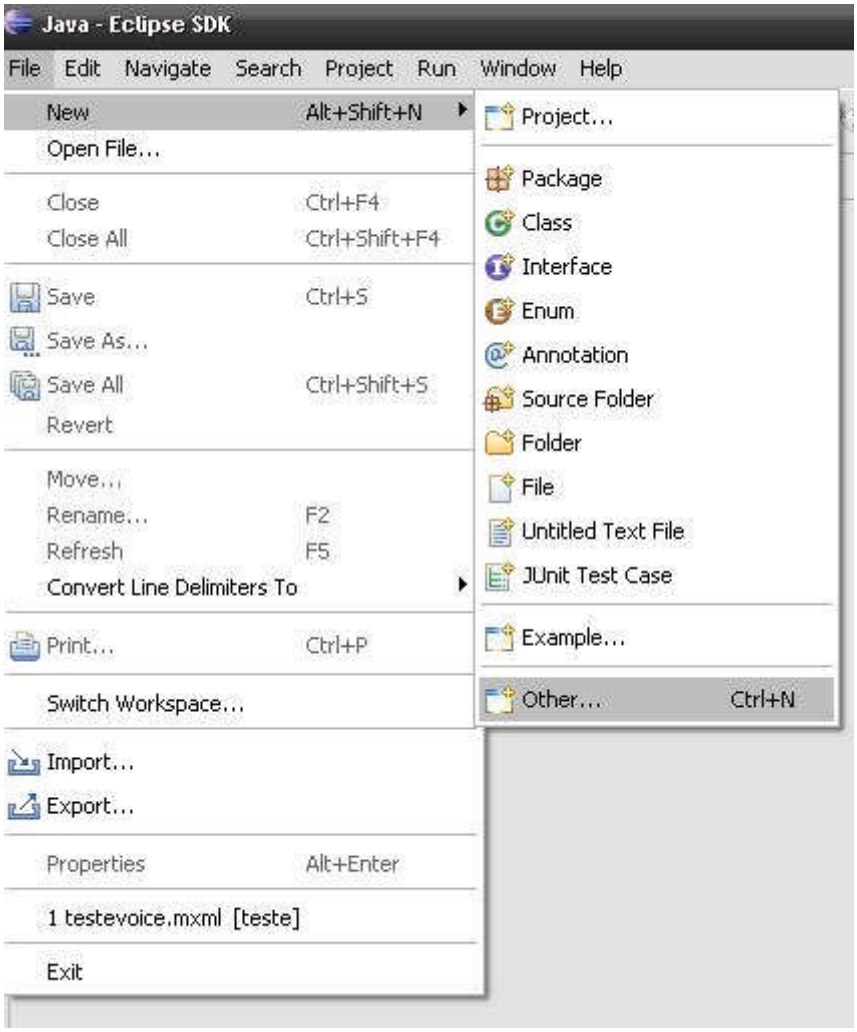

Figura 13. Criando o arquivo X+V

Click em File > New > Other ...

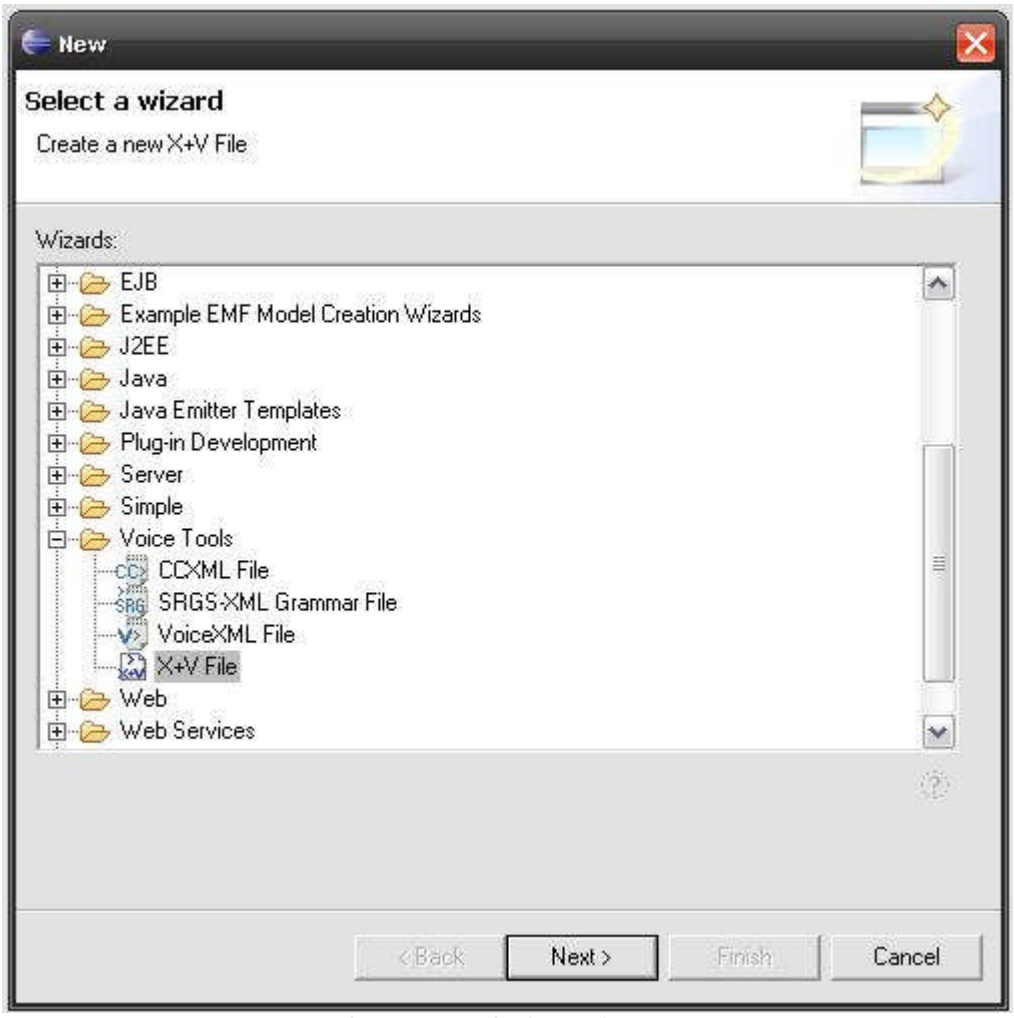

Figura 14. Selecionando X+V

Selecione em Voice Tools o item X+V File e Click em Next.

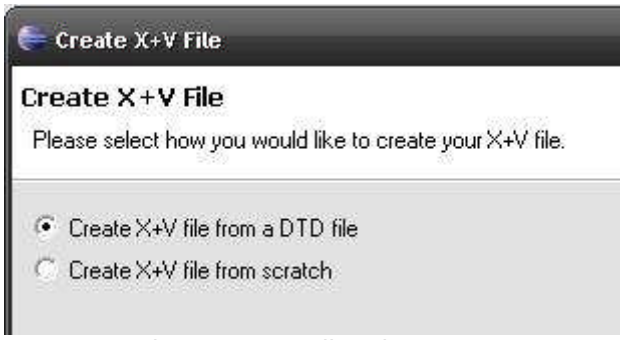

Figura 15. Escolhendo o DTD

Click em Next novamente.

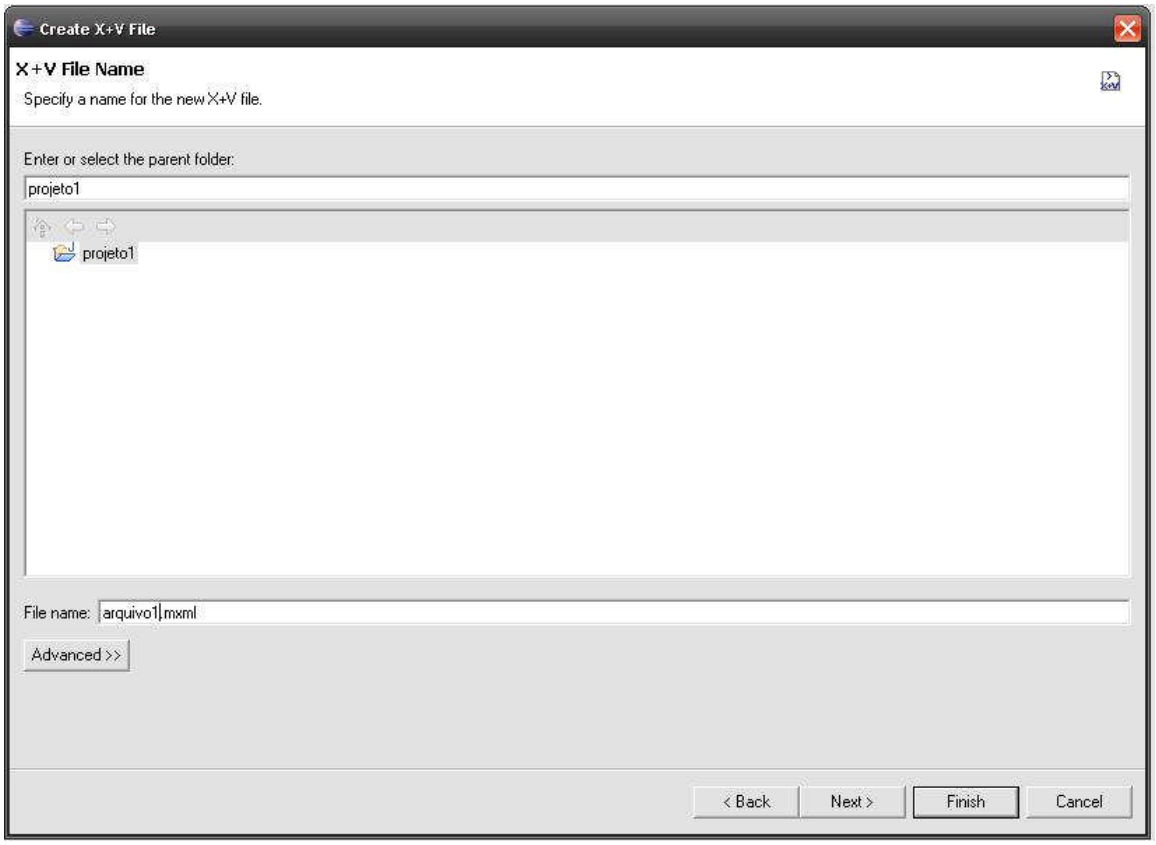

Figura 16. Nomenando o arquivo X+V

Selecione um projeto e dê um nome ao arquivo criado, nesse exemplo projeto1 e arquivo1, e Click em Finish.

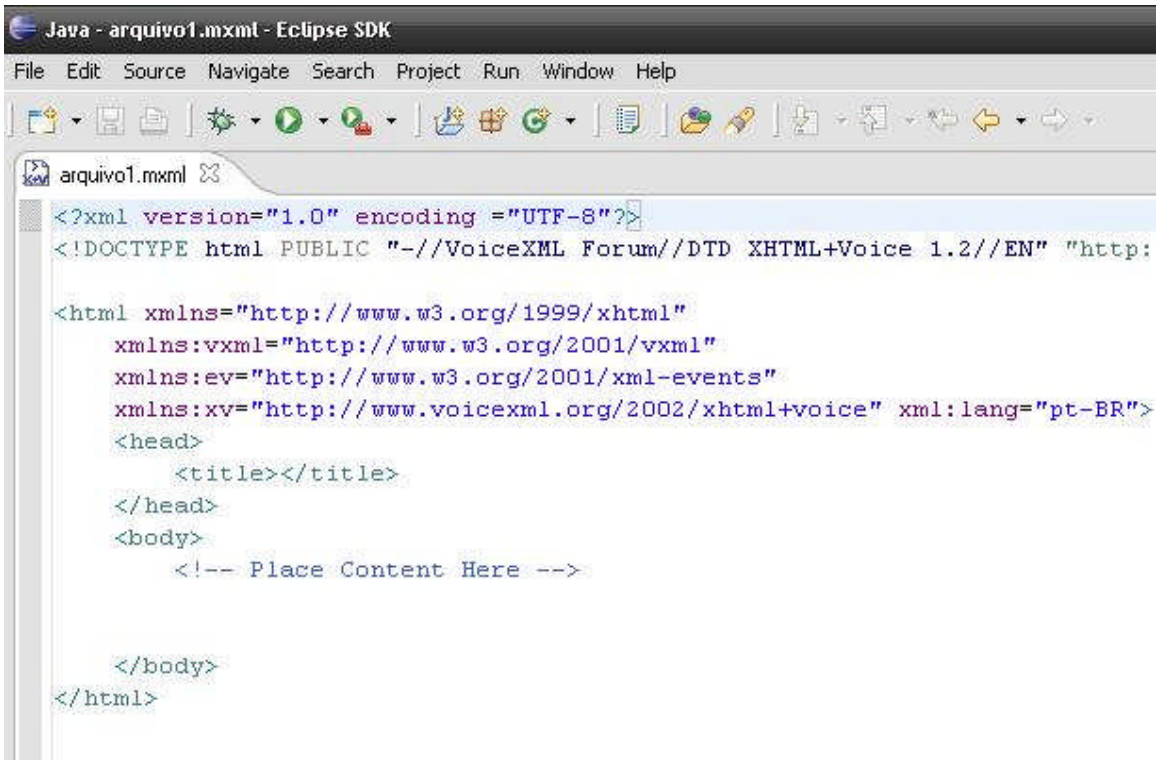

Figura 17. Abrindo o arquivo criado

Pronto, você acaba de criar um arquivo X+V para ser editado no Eclipse 3.1.

5.3. Exemplos de aplicações X+V

A seguir serão apresentados alguns exemplos retirados e traduzidos do guia desenvolvido pela IBM (XHTML + Voice Programmers Guide, vide Bibliografia).

O primeiro exemplo de aplicação X+V faz duas coisas. Quando a página é carregada, o engine de text-to-speech (TTS) diz "Hello world" ao usuário. Uma vez que isso termina, o texto "Hello world!" é atribuído a um elemento de caixa de texto do HTML. A combinação da saída ou entrada de áudio junto com a saída ou entrada visual é a essência do X+V. Note que usamos eventos de XML como o "vxmldone" para sinalizar à página que a aplicação deve fazer algo.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice
1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:vxml="http://www.w3.org/2001/vxml" xml:lang="en_US">
<head>
        <title>Basic XHTML+VXML Example</title>

<!- -  Quando declare="declare" está presente, o script não é executado até que o
documento seja completamente carregado e seja chamado por um evento do
usuário -->

<script type="text/javascript" id="vxml_form_handler" declare="declare">
        document.getElementById('page.output_box').value = "Hello, world!";
</script>
<vxml:form id="vxml_form">
        <vxml:block>hello world</vxml:block>
</vxml:form>

<!-- A seguir nós atribuímos um elemento body como um observador XML, que
nos deixa chamar  o script  a ser executado quando o formulário termina. -->

<ev:listener ev:observer="page.body" ev:event="vxmldone"
ev:handler="#vxml_form_handler" ev:propagate="stop" />
</head>
<!— Associa o VXML que deve ser carregado quando a página for carregada, e
associa um document ID ao HTML <body>. -->
        <body id="page.body" ev:event="load" ev:handler="#vxml_form">
```

```
            <!--  o Texto  vai aparecer na tela  na caixa de texto depois que a
            pagina falar com o usuário. -->
            Here it comes...<br/><br/>
            <input type="text" id="page.output_box" value="" size="40"/>
            <br/>
        </body>
</html>
```

No exemplo a seguir será usada uma gramática para entender uma resposta do usuário. Neste caso ela será colocada inline, mas em projetos reais é mais comum colocá-la em um arquivo externo.

```
<?xml version="1.0" encoding="iso-8859-1"?>
<!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice
1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:vxml="http://www.w3.org/2001/vxml" xml:lang="en_US">
<head>
        <title>XHTML+VXML Example, with Input</title>
        <script type="text/javascript">
                var planet_var = "";
                function FormatPlanet(oldStr)
                {
                        //Make sure the first character is upper-case.
                        newStr = oldStr.charAt(0).toUpperCase()
                        + oldStr.substring(1, oldStr.length);
                        return newStr;
                }
        </script>
<vxml:form id="vxml_form_prompt">
        <vxml:field name="vxml_field">
                <vxml:grammar>
                        <![CDATA[
                                #JSGF V1.0;
                                grammar planet_selection;
                                public <planet_selection> =
                                mercur          y | venus | earth
                                | mars |        jupiter | saturn
                                | uranus | neptune | pluto
                                ;
                        ]]>
                </vxml:grammar>
                <vxml:prompt> Which world would you like to say hello to?
                </vxml:prompt>
```

```
            <!—caso o usuário não entenda quais são as suas opções ele pode
            dizer "help" e ouvir a explicação de uma outra forma. Note que
            este não é o único evento que podemos capturar (catch) -->
            <vxml:catch event="help"> Your options are: mercury, venus,
            earth, mars, jupiter, uranus, neptune, or pluto. </vxml:catch>
            <!—
            Uma vez que o campo reconheceu uma entrada da gramática,
            Pode-se ir adiante para a atribuição das variáveis. Pode-se mudar
            os elementos do HTML de dentro dos formulários VXML. É por
            isso que as variáveis que VoiceMXL usa são variáveis
            ECMAscript, assim como as outras variáveis da página. -->
            <vxml:filled>
                  <vxml:assign name="planet_var" expr="vxml_field"/>
                  <vxml:assign
                  name="document.getElementById('page.output_box').valu
                  e" expr="'Hello, ' + FormatPlanet(planet_var) + '!'"/>
            </vxml:filled>
      </vxml:field>
            <!--
            Pode-se misturar e combinar varios elementos vxml:field ou
            vxml:block, e eles serão visitados em ordem.
            -->
      <vxml:block>
            Hello <vxml:value expr="planet_var"/>, you sure are a wonderful
      planet!
      </vxml:block>
</vxml:form>
</head>
      <body id="page.body" ev:event="load"
ev:handler="#vxml_form_prompt">
            <input type="text" id="page.output_box" value="Hello?"
      size="18"/>
            <br/>
      </body>
</html>
```

No terceiro exemplo, usaremos uma gramática externa ao invés de inline, que é a maneira recomendada de usar a maioria das gramáticas. Note que este exemplo mostra tópicos interessantes que você pode fazer com as gramáticas em aplicações VoiceXML (as gramáticas podem ser versáteis).

A Cada 10 segundos, caso o usuário não preencha o formulário, a aplicação toca o diálogo de ajuda. Isto é, uma vez que o intervalo de tempo sem fala é alcançado, um evento "noinput" lançado.

```
      <?xml version="1.0" encoding="iso-8859-1"?>
```

```
        <!DOCTYPE html PUBLIC "-//VoiceXML Forum//DTD XHTML+Voice
1.2//EN"
"http://www.voicexml.org/specs/multimodal/x+v/12/dtd/xhtml+voice12.dtd">
<html xmlns="http://www.w3.org/1999/xhtml"
xmlns:ev="http://www.w3.org/2001/xml-events"
xmlns:vxml="http://www.w3.org/2001/vxml" xml:lang="en_US">
        <head>
                <title>XHTML+VXML Example, with Input</title>
                <script type="text/javascript">
                        var drink_selection = "";
                        var cream = false;
                        function FormDone(){
                                var str = "One "  + drink_selection.size + " " +
                        drink_selection.type;
                                if (cream === true){
                                        str += ", with cream.";
                                }
                                else{
                                        str += ".";
                                }
                                document.getElementById('page.output_box').value
                        = str;
                        }
                </script>

<vxml:form id="vxml_drink_form">
        <vxml:field name="drink_field">
                <vxml:grammar src="gram/beverage.jsgf"/>
                <vxml:prompt timeout="10s"> What kind of drink would you like
        to order? </vxml:prompt>
                <vxml:catch event="nomatch noinput help">
                        This form lets you order a drink. You may order a small,
                medium, or large drink. The drink may be coffee, lemonade, soda,
                or milk.
                </vxml:catch>
                <vxml:filled>
                        <!—associa o valor atual de drink_field à uma variável
                Javascript externa
                        Poderia tambem optar por usar drink_field pelo resto da
                        aplicação, assumindo que ele é sempre preenchido -->
                        <vxml:assign name="drink_selection"
                expr="drink_field"/>
                </vxml:filled>
        </vxml:field>
</vxml:form>

<vxml:form id="vxml_coffee_prompt">
        <vxml:field name="coffee_field">
                <vxml:grammar src="gram/yes_no.jsgf"/>
```

```
            <vxml:prompt> Would you like cream with your coffee?
        </vxml:prompt>
            <vxml:catch event="nomatch help">
                If you would like cream with your coffee,  then say "yes".
            Otherwise, say "no".
            </vxml:catch>
            <vxml:filled>
                <vxml:if cond="true = = = coffee_field">
                    <vxml:assign name="cream" expr="true"/>
                </vxml:if>
            </vxml:filled>
        </vxml:field>

</vxml:form>
            <!--
            O script mostra como navegar entre forms
            Se o usuário selecionar café, a aplicação vai para o próximo
            formulário VXML para perguntar se o usuário quer creme com
            seu café. FormDone() é a função que escreve a saída visual à caixa
            de texto do HTML e que será chamada quando para todos as
            opções exceto para café. O café, chama FormDone() em seu
            handler vxmldone somente uma vez.
            -->
            <script type="text/javascript" id="vxml_drink_form_handler"
        declare="declare">
            if ("small" === drink_selection.size)
        document.getElementById('page.size.small').checked = true;
            else if ("medium" === drink_selection.size)
        document.getElementById('page.size.medium').checked = true;
            else if ("large" === drink_selection.size)
        document.getElementById('page.size.large').checked = true;
            if ("coffee" === drink_selection.type){
                document.getElementById('page.drink.coffee').checked=
            true;
            /****
            Aqui ocorre a mudança para um novo form com um evento de
        click
 ****/
                document.getElementById('vxml.coffee').click();
            }
            else{
            if ("soda" === drink_selection.type)
        document.getElementById('page.drink.soda').checked = true;
            else if ("lemonade" === drink_selection.type)
        document.getElementById('page.drink.lemonade') .checked = true;
            else if ("milk" === drink_selection.type)
        document.getElementById('page.drink.milk') .checked = true;
                FormDone();
            }
            </script>
```

```
        <script type="text/javascript" id="vxml_coffee_form_handler"
declare = "declare">
                FormDone();
        </script>
        <ev:listener ev:observer="page.body" ev:event="vxmldone"
ev:handler="#vxml_drink_form_handler" ev:propagate="stop" />
        <ev:listener ev:observer="vxml.coffee" ev:event="vxmldone"
ev:handler="#vxml_coffee_form_handler" ev:propagate="stop" />
        </head>
        <body id="page.body" ev:event="load"
ev:handler="#vxml_drink_form">
                <input type="hidden" id="vxml.coffee" value=""
                ev:event="click"  ev:handler="#vxml_coffee_prompt"/>
                <b>Multimodal Drink Order</b><br/><br/>
                Size options:<br/>
                        <input type="radio" name="size"
                id="page.size.small"/> Small <br/>
                        <input type="radio" name="size"
                id="page.size.medium"/> Medium <br/>
                        <input type="radio" name="size"
                id="page.size.large"/> Large <br/> <br/><br/>
                Drink options:<br/>
                        <input type="radio" name="type"
                id="page.drink.soda"/> Soda <br/>
                        <input type="radio" name="type"
                id="page.drink.lemonade"/> Lemonade <br/>
                        <input type="radio" name="type"
                id="page.drink.coffee"/> Coffee <br/>
                        <input type="radio" name="type"
                id="page.drink.milk"/> Milk <br/>
                        <br/><br/>
                        <input type="text" id="page.output_box"
                value="Waiting for selection." size="40"/>
                        <br/>
        </body>
        </html>
```

Yes/no JSGF grammar

```
This grammar includes typical responses for specifying yes or no.
#JSGF V1.0 iso-8859-1;
grammar yes_no;
public <yes_no> =
<yes> { $ = true; }  | <no> { $ = false; };
<yes> = yes [please] | sure | okay | fine | yep | yup | affirmative;
<no> = no | nope | no thanks | negative;
```

Beverage JSGF grammar

```
#JSGF V1.0;
```

```
grammar beverage;
public <beverage> = [I would like | I want | [please] give me]
[a | an]
<size> { $.size = $size; }
<type> { $.type = $type; }
;
<size> = small { $ = "small"; }
| (medium | regular) { $ = "medium"; }
| large { $ = "large"; }
;
<type> = coffee
| milk
| (soda|pop|coke) { $ = "soda"; }
| lemonade ;
```

6. Considerações para o desenvolvimento de interfaces multimodais que utilizam interação visual e por voz

Existem duas formas de apresentar a informação ao usuários usando áudio:

1. TTS: é mais simples de manter e modificar. É utilizado durante o desenvolvimento do software e nos casos onde ainda não se sabe o que será dito de antemão, tornando a tarefa de gravação em estúdio difícil
2. Prompts gravados: usado quando o software está sendo utilizado por usuários, pois são áudios com maior qualidade e com pronúncia natural.

Outro desafio de se desenvolver uma aplicação de qualidade que contenha interação por voz é usar o tempo de maneira eficiente e não obrigar o usuário a ouvir mais do que ele precisa ouvir para entender a mensagem, responder de forma sucinta e conseguir interagir com o sistema. Para isso siga as seguintes guidelines:

- Quando projetar menus, limite o número dos itens de menu baseando-se em quanta informação os usuários devem se recordar posteriormente.
- Agrupe a informação separando o texto introdutório do texto principal, separando o texto para cada item de menu, agrupando os itens de menu
- Escolha as palavras certas para os prompts e menus da sua aplicação para evitar ambigüidade na resposta.
- Para os diálogos que os usuários podem visitar de novo em uma única sessão, você pode criar vários prompts, um em seguida do outro, mais curtos.

Depois da passagem inicial do diálogo, você pode então tocar os prompts menores em outras visitas à mesma interface.

- Use confirmações da entrada do usuário com cautela, pois isso pode irritá-lo.

As aplicações de multimodais consistem em elementos de voz e em elementos visuais. Não há necessariamente um mapeamento de um para um entre elas. Algumas informações são melhor apresentadas usando a voz, e outras usando gráficos, e você pode combinar ambos os formatos na maioria de casos.

Em geral, a informação de welcome e introdutória pode ser apresentada usando a voz para obter a atenção do usuário assim que a aplicação começar. Os elementos que contêm a informação breve, tal como instruções e assuntos curtos, são também melhor apresentados com a voz.

A maioria de elementos visuais que requerem a entrada ou a ação do usuário podem ser habilitados usando a voz. Alguns exemplos comuns:

- Campo de texto - quando os usuários clicam em um campo de texto, um prompt pode ser tocado, e a resposta de voz do usuário pode ser atualizada e mostrada no campo de texto. Isto é útil e conveniente quando o usuário não tem o acesso ao teclado.
- listas - os usuários podem selecionar itens nas listas usando a voz. É útil quando o mouse não está disponível ao usuário, e é conveniente quando o usuário é familiar com o conteúdo da lista. Por exemplo, em uma lista de estados, a maioria dos usuários sabe os itens na lista e sabem exatamente qual ele necessita selecionar, assim o usuário pode apenas dizer o nome do estado, por exemplo, "São Paulo" para selecionar o item sem a necessidade clicar no *drop-down* da lista.
- Checkbox - se um grupo de *checkboxes* for habilitado, os usuários podem selecionar um ou mais itens usando a entrada de voz em vez de fazer cliques múltiplos
- Botão - os usuários podem realizar uma ação de clique no botão usando a voz.

Algumas informações não são fáceis de apresentar usando a voz, tal como gráficos, diagramas e tabelas. Estes são melhor apresentados no formato visual. Se você quiser adicionar a voz a estes elementos visuais, você necessita considerar a frase enfatizando a informação chave descrita.

Dado que uma aplicação multimodal tem ambas as interfaces visual e de voz conectadas, é muito importante manter uma sensação de visual consistente para a aplicação. Para promover a consistência, você pode adotar as seguintes guidelines:

- Aderir aos padrões disponíveis para o projeto visual da interface, tal como usar o mesmo estilo visual de apresentação para todas as páginas em sua aplicação
- Usar uma estratégia consistente determinando que informação apresentar verbalmente, visualmente, e usando ambas as interfaces
- Tentar usar a mesma terminologia em ambas as interfaces sempre que possível.
- Escolher um estilo de prompt e usá-lo consistentemente durante toda sua aplicação
- Tentar não usar voz passiva. Usar "transferir quanto?" ao invés de "quantidade a ser transferida?"
- Sempre que possível, usar uma estrutura paralela nas listas. Por exemplo, use "diga uma das seguintes opções: listar o endereço atual, mudar o endereço" ao invés de " diga uma das seguintes opções: listar o endereço atual, mudança do endereço".
- Manter a consistência na entrada do usuário aceitável. Se você permitir respostas afirmativas do tipo sim, OK, e verdadeiro a uma pergunta sim/não, você deve permitir as mesmas respostas para todas as perguntas sim/não em sua aplicação.
- Usar pausas consistentemente.

O projeto do diálogo é crítico no processo de projetar uma aplicação multimodal. Quando os usuários não sabem o que podem dizer em um ponto do diálogo, a interação entre o usuário e a aplicação pode rapidamente ser comprometida. Para ajudar os usuários a saber o que dizer:

- Tente escrever prompts que indiquem claramente aos usuários o que dizer. Exemplos: "diga ou Sim ou Não" ou "Escolha uma das opções: e-mail, correio de voz, fax."
- Quando uma aplicação se comporta em uma maneira que o usuário não antecipa, o usuário fica confuso e é mais provável responder impropriamente aos prompts subseqüentes, fazendo com que a interação entre o usuário e a aplicação fique prejudicada. Para ajudar os usuários, use som e layout consistentes, crie diálogos

que mantenham a sincronização entre o estado real da aplicação e o pensamento do usuário do comportamento da aplicação.

- Na interface visual, posicione objetos equivalentes em posições similares da tela.

A sincronização é um tópico específico das aplicações multimodais. Pelo fato de que a aplicação multimodal está apresentando a informação ao usuário usando interfaces visuais e de voz, as duas interfaces devem sempre ser sincronizadas.

Durante o projeto da aplicação, considere que as transições do diálogo envolvem ambas as interfaces. Por exemplo, quando o browser de voz processar um diálogo na primeira página, o browser visual não deve ser levado à pagina seguinte a menos que o browser de voz estiver fazendo também uma transição a essa página. Se o usuário forçar a transição manualmente, a interface da voz deve se ajustar conformemente, parando o processo na primeira página e começando o processo para a página seguinte ou disparando uma mensagem de erro. A interface visual deve atualizar a página para mostrar também a resposta de voz.

Além de sincronizar transições de diálogo as transições entre elementos também devem ser realizadas. Se o controle da voz se mover de um campo de texto para uma lista, certifique-se de que o foco visual é movido também do campo de texto para a lista.

7. Princípios, *guidelines* e *checklists* para o desenvolvimento de interfaces multimodais Web

Os paradigmas de interface que são bem sucedidos atualmente, como, por exemplo, o WIMP e o arrastar e soltar (drag and drop), foram estabelecidos com base na consistência percebida pelo usuário com a interação entre projetos diferentes e na interpretação de um comportamento intuitivo, resultando em padrões de projeto bem entendidos e metáforas que podem ser aplicadas independentemente do sistema. Essas metáforas permitem que o usuário aplique o conhecimento adquirido em sua interação com a aplicação em diversas outras tarefas em sistemas diferentes.

Por trás desses paradigmas foram estabelecidos alguns princípios de bons projetos de interface com o objetivo de pautar o projeto e permitir a descoberta de problemas de usabilidade. Com base nessa particularidade presume-se que as interfaces multimodais também necessitam estar fundamentadas em um conjunto de princípios,

para facilitar o projeto de sistemas multimodais e porque elas integram várias modalidades de interação com o usuário.

Dessa forma, novas metáforas poderão se manifestar nas aplicações e nas interfaces nas quais elas desempenharão o papel de integrar as entradas do usuário, que podem ocorrer de diversos modos, e na síntese de saída de dados na forma de múltiplas mídias.

A seguir estão listados os principais princípios e guidelines encontrados na literatura e alguns *checklists* que facilitam o desenvolvimento de interfaces multimodais Web com o uso de voz:

(1) As múltiplas modalidades precisam estar sincronizadas. A interação por voz pode ser considerada como temporal, enquanto a interação visual gráfica pode ser considerada espacial. Quando combinadas em uma interface multimodal, esses modos de interação precisam estar sincronizados para não comprometer a usabilidade do sistema.

CheckList - As modalidades de voz e visual estão sincronizadas para todos os componentes de interface[1] e para todas as possíveis entradas da(s) gramática(s) na interface multimodal?

Guideline: Projete a consistência no foco entre as diversas modalidades.

(2) A transição entre diálogos em uma interação multimodal deve ser perceptível ao usuário, pois as suas habilidades e necessidades, bem como o contexto de uso pode ser modificado durante a interação.

CheckList - O usuário consegue perceber a mudança de diálogo para cada interface multimodal?

(3) As múltiplas modalidades devem compartilhar o estado de interação para que durante a mudança de uma interface para outra ou se o usuário desejar trocar de modalidade em uma determinada seqüência do diálogo humano-computador, as várias mídias disponíveis sejam atualizadas.

[1] componentes de interface - tradução de widget. São as partes que formam a interface Web como: campos de texto, listas, itens de seleção, botões, links, etc; e no caso das interfaces de voz: prompts, entradas e saídas de gramáticas, formulários, menus, etc.

CheckList - Os componentes de interface, as gramáticas e javascripts são atualizados durante a transição de um diálogo para outro ou durante a transição de interfaces?

CheckList - O usuário consegue trocar de modalidade (voz para visual, ou visual para voz) durante a interação? A interface e/ou seus componentes são atualizados?

(4) As interfaces multimodais devem ser previsíveis, ou seja, a interface precisa possibilitar que o usuário saiba intuitivamente qual o melhor modo para se utilizar durante a execução de uma tarefa. A interface deve permitir que o usuário perceba se em um dado momento algum modo estiver indisponível.

CheckList - O usuário consegue perceber qual modalidade ele deve utilizar para cada componente de interface e para cada interface?

CheckList - Existe algum modo indisponível em um determinado momento da interação? Descreva o DR para cada caso separadamente e diga como o usuário é alertado ou percebe essa situação.

(5) As interfaces multimodais devem se adaptar ao ambiente do usuário para assegurar que o melhor modo de se completar uma determinada tarefa em um determinado momento estará disponível ao usuário. A adaptatividade é também considerada como uma guideline. As interfaces multimodais devem se adaptar às necessidades e habilidades de diferentes usuários e a diferentes contextos de uso. A adaptatividade dinâmica permite que a transição entre diálogos em uma interação seja perceptível ao usuário por meio de modalidades complementares.

CheckList - Existe alguma lógica na aplicação multimodal que permita a adaptação da interface? Em que caso isto poderia se utilizado?

(6) Durante a Especificação de Requisitos, o projeto da interação multimodal deve ser pensado para uma grande gama de usuários e contextos de uso. Além disso, o projetista deve considerar questões de privacidade e de segurança.

CheckList - Que tipo de usuário consegue utilizar a interface (ou o componente) que está sendo desenvolvida?

CheckList - (Privacidade) campos de senha ou de entrada de dados particulares devem permitir entrada por teclado ao invés de voz e mascarar o resultado da entrada na interface no caso de senhas.

(7) Projeto da entrada e saída multimodal. O projetista deve utilizar o recurso de múltiplas modalidades para maximizar as habilidades humanas cognitivas e físicas. Além disso, deve evitar a apresentação de informações desnecessárias nas diferentes modalidades, com o objetivo de maximizar as vantagens de cada modalidade para reduzir a carga de memória do usuário durante a execução de uma tarefa. É importante também integrar as modalidades de maneira compatível com as preferências do usuário, contexto e funcionalidades do sistema.

CheckList - Existem informações duplicadas (texto e voz individualmente e separadamente) sendo apresentadas ao usuário?

CheckList - Quais funcionalidades do sistema são melhor utilizadas usando-se interfaces ou componentes de voz? e gráfico? e ambos simultaneamente? Explique suas decisões?

Guideline: Para cada tarefa, use o modo de interação mais simples disponível no dispositivo.

Guideline: Se a mão ou a visão do usuário está ocupada, use interface de voz.

Guideline: se o usuário está em um ambiente ruidoso, use o toque ou interface gráfica.

Guideline: Use áudio para indicar que a informação fornecida deve ser verbal.

Guideline: Use pausas para dividir a informação em "grupos" naturais.

Guideline: Use animação e som para indicar transições.

Guideline: Use navegação por voz para reduzir o numero de telas

Guideline: Use interface gráfica para facilitar a retenção de informação na memória de curta duração do usuário.

exemplos:

* Se uma lista contem mais do que 7±2 itens, mostre as opções na tela.

* se um texto contem mais do que duas sentenças, apresente-o visualmente ao usuário

(8) Consistência na interação. As Interfaces visuais e de voz devem compartilhar ao máximo as características comuns e devem se referenciar a uma tarefa com a mesma terminologia nas diferentes modalidades.

CheckList - A terminologia e os comandos são consistentes entre as interfaces, componentes e modalidades?

Guideline: Permita ao usuário falar palavras-chave ao invés de sentenças complexas. Coloque essas palavras nos prompts do sistema.

(9) Retorno (feedback) Os usuários devem perceber que modalidades estão presentes em determinado momento da interação e devem perceber quais são os tipos de interação alternativas que estão presentes para executar uma tarefa.

CheckList - o usuário percebe o estado atual de uma tarefa e sabe quando a terminou?

Guideline: use saída de voz para informar ou comentar uma ação do usuário ou fornecer informação complementar.

Guideline: Apresente a(s) palavra(s) reconhecida(s) pelo ASR na forma visual, para que o usuário possa verificar se o que foi reconhecido está correto.

Guideline: Mostre uma lista n-best para habilitar a correção de erros de reconhecimento de fala

Guideline: Tempo de resposta deve ser menor que 5seg. Informe ao usuário quando houver demora em consultas de BD.

Guideline: Sempre mostre o status do sistema ao usuário. Indique o caminho percorrido na forma gráfica. e permita falas do tipo "Repita", "volte", "menu principal"

(10) Tratamento e prevenção de erros. - Os erros podem ser evitados e minimizados e o tratamento de erros pode ser melhorado quando a interface fornece claramente ao usuário quais saídas o sistema disponibiliza e permitindo que ele reverta ou recupere suas ações quando desejado. Os usuários novatos ocasionalmente podem não responder a um prompt de maneira adequada. A interface deve ser projetada para detectar tais erros e ajudar os usuários a se recuperarem naturalmente. As interfaces multimodais devem ajudar o usuário a aprender como usar as múltiplas modalidades para atingir os resultados desejados rápida e eficientemente.

CheckList - Existem informações duplicadas (texto e voz individualmente e separadamente) sendo apresentadas ao usuário?

CheckList - O sistema consegue tratar erros de reconhecimento, estouro de tempo sem fala, ajudar o usuário, repetir a ultima fala, voltar para um local seguro e conhecido pelo usuário?

Guideline: Sempre forneça ajuda sensível ao contexto para cada ação.

Princípio reflexivo. As pessoas tendem a responder de maneira parecida como são perguntadas. Perguntas longas serão respondidas com respostas longas. Trechos da pergunta freqüentemente são usados na resposta.

Guideline: O usuário tende a usar o mesmo modo no qual foi feita a pergunta para fornecer a resposta.

8. Bibliografia

- XHTML+Voice 1.2 Specification: http://www.voicexml.org/specs/multimodal/x+v/12/spec.html
- XML Events specification: http://www.w3.org/TR/xml-events/
- Document Object Model (DOM) – Level 2 specification: http://www.w3.org/DOM/#what
- IBM - XHTML + Voice Programmers Guide
- W3C Multimodal Interaction Activity: http://www.w3.org/2002/mmi/
- EMMA: Extensible Multimodal Annotation markup language http://www.w3.org/TR/emma/
- Demonstrações de X+V – http://alexandre.alapetite.net/phd-risoe/mxml/
- http://www-306.ibm.com/software/pervasive/multimodal/
- Raman, T.V. Design Principles For Multimodal Interaction, MMI Workshop, CHI 2003.
- Reeves, L.M. “Guidelines for multimodal user interface design,” Communications of the ACM, Vol. 47, Issue 1, pp. 57-69, January 2004.

- Reeves http://www.larson-tech.com/MMGuide.html#Summary IBM Multimodal Application Design Issues - IBM White Paper - December 2003
- XMLEVENTS for HTML authors. http://www.maujor.com/w3c/xmlevents-for-html-authors.html
- Talarico Neto, A. Uma abordagem para o desenvolvimento de aplicações multimodais Web. Texto de Qualificação do doutorado – ICMC Setembro de 2007

Especificações:

- VoiceXML 2.0 specification - http://www.w3.org/TR/voicexml20/
- Speech Recognition Grammar Specification - http://www.w3.org/TR/speech-grammar/
- Semantic Interpretation for Speech Recognition specification - http://www.w3.org/TR/semantic-interpretation/
- Java Speech Grammar Format - http://java.sun.com/products/java-media/speech/forDevelopers/JSGF/
- Aural CSS 2.1 - http://www.w3.org/TR/CSS21/aural.html
- CSS 3 speech module - http://www.w3.org/TR/css3-speech/
- Speech Synthesis Markup Language - http://www.w3.org/TR/speech-synthesis/

Tutoriais e documentos:

- XML Events tutorial - http://www.w3.org/MarkUp/2004/xmlevents-for-html-authors
- Authoring XHTML+Voice - http://dev.opera.com/articles/voice/
- Accessibility in Web pages - http://www.howtocreate.co.uk/accessibility.html
- IBM Multimodal site - http://www.ibm.com/software/pervasive/multimodal/
- VoiceXML forum - http://www.voicexml.org/
- World of VoiceXML - http://www.kenrehor.com/voicexml/

- Accessibility and voice browsing - http://my.opera.com/community/forums/forum.dml?id=83
- IBM's Multimodal Tools Newsgroup - news://news.software.ibm.com/ibm.software.speech.multimodal
- Yahoo VoiceXML group - http://groups.yahoo.com/group/voicexml/
- irc.opera.com/voice - irc://irc.opera.com/voice