UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUIBUILDER MULTIMODAL: UM FRAMEWORK PARA A GERAÇÃO DE INTERFACES MULTIMODAIS COM O APOIO DE INTERACTION DESIGN PATTERNS

YGARA LÚCIA SOUZA MELO FRAGOSO

ORIENTADORA: PROF. DR. EDNALDO BRIGANTE PIZZOLATO

São Carlos - SP
Nov/2012

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

GUIBUILDER MULTIMODAL: UM FRAMEWORK PARA A GERAÇÃO DE INTERFACES MULTIMODAIS COM O APOIO DE INTERACTION DESIGN PATTERNS

YGARA LÚCIA SOUZA MELO FRAGOSO

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software.
Orientadora: Dr. Ednaldo Brigante Pizzolato.

São Carlos - SP
Nov/2012

Ficha catalográfica elaborada pelo DePT da Biblioteca Comunitária da UFSCar

F811gm

Fragoso, Ygara Lúcia Souza Melo.
Guibuilder multimodal: um framework para a geração de interfaces multimodais com o apoio de interaction design patterns / Ygara Lúcia Souza Melo Fragoso. -- São Carlos : UFSCar, 2015.
146 f.

Dissertação (Mestrado) -- Universidade Federal de São Carlos, 2012.

1. Interação homem-máquina. 2. Framework (Programa de computador). 3. Usabilidade. 4. Agentes inteligentes. 5. Java. I. Título.

CDD: 004.019 (20ª)

Universidade Federal de São Carlos

Centro de Ciências Exatas e de Tecnologia
Programa de Pós-Graduação em Ciência da Computação

"Guibuilder Multimodal: Um Framework para a Geração de Interfaces Multimodais com o Apoio de Interaction Design Patterns"

Ygara Lúcia Souza Melo Fragoso

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação

Membros da Banca:

Prof. Dr. Ednaldo Brigante Pizzolato
(Orientador - DC/UFSCar)

Profa. Dra. Vânia Paula de Almeida Neris
(DC/UFSCar)

Profa. Dra. Simone Diniz Junqueira Barbosa
(PUC-RJ)

São Carlos
Novembro/2012

Dedico este trabalho aos meus pais, que mesmo longe se fazem presentes sempre.

AGRADECIMENTO

Agradeço, primeiramente, a Deus por todas as lutas, todas as bênçãos, toda a inspiração e coragem que me foi conferida durante todas as fases deste trabalho. Sei que foram anos de muita turbulência, com muitas frustrações e obstáculos, mas meu Deus foi muito maior e mesmo com todas as lutas e dificuldades Ele me ofereceu um mundo cheio de oportunidades, realizações e felicidade. Obrigada Senhor pela luz.

Agradeço a CAPES e ao Departamento de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos por anos brilhantes em minha vida. A Federal sempre, no meu coração, vai ser muito mais legal, sempre minha casa, sempre meu refugio.

Agradeço, com muito orgulho e dedicação, minha família principalmente meus pais que mesmo longe sempre me apoiaram na realização deste sonho. Obrigada mãe, Admilde, obrigada pai, José Luiz. Sendo Deus tão bondoso, me deu a maior e melhor família do mundo. Então agradeço meus irmãos: Yago, Luiz Antônio, Luciano, Lívia e Luciene; meus sobrinhos e sobrinhas, primos e primas, tios e tias, amigos e amigas e o carinho sempre especial direcionado a mim pelos meus avôs.

Seria muito injusto não agradecer os meus colegas de laboratório que, juntamente comigo, compartilharam os caminhos tempestuosos que é cumprir com todas as atividades e exigências de uma pós-graduação. Somente eles sabem, literalmente, o que passamos dentro e fora de sala de aula, dentro e fora de um laboratório de pesquisa. Deyse, Gilmar, Cláudia, Felipe, David, Mateus, Elaine Cecília, Mariana, Thiago, Guilherme e alguns que nem estiveram dentro deste ambiente mas participaram de forma direta como minha querida amiga Carmen Brevis, Dona Glória valeu pela força e por tudo que você fez por mim. Além de todos os outros que fizeram parte direta e indiretamente deste projeto.

Acredito que a pessoa mais importante, a qual devo imensa gratidão, respeito e admiração, é meu orientador prof. Dr. Ednaldo B. Pizzolato. O homem por trás deste sonho, o homem responsável por esta vitória, o homem que diferente de muitos acreditou, apostou e sofreu comigo todas as etapas necessárias para realização desta jornada. Prof. muito obrigada por tudo que o senhor fez por mim, que Deus o ilumine cada dia mais.

Houve um tempo em que eu tinha uma luz dos olhos que guiava e seguia comigo a jornada desta vida. Seria imensamente injusto não agradecer essa luz por me apoiar e sempre me incentivar. Hoje não mais existe essa luz, mas me foi muito importante em vários momentos críticos da minha vida. Obrigada!!!

Não poderia esquecer nunca de agradecer minhas companheiras de equipe do time de Futsal Feminino da UFSCar. Meninas vocês terão sempre um lugar cativo no meu coração porque afinal de contas "*Nos somos muito mais gostosas do que elas!!!*". Meus domingos

nunca serão os mesmos sem os torneios, amistosos e viagens para São Paulo e outras cidades do estado. Sinto imensas saudades de vocês. “*Federal Joga Eu vou!!!*”.

Obrigada a todos que torceram, encorajaram, me apoiaram e principalmente me ajudaram nessa etapa importante na minha vida. Muito, mas muito obrigada mesmo.

"Não me entrego sem lutar. Tenho, ainda, coração. Não aprendi a me render Que caia o inimigo então."

Renato Russo

RESUMO

A interação entre humano e computador tem melhorado substancialmente ao longo do tempo através da evolução das interfaces de interação. A possibilidade de usuários interagirem com máquinas através de várias modalidades de comunicação, e de forma natural, pode aumentar o nível de interesse do usuário e garantir o sucesso da aplicação.

Porém, o estado da arte da área de multimodalidade demonstra que desenvolver tais interfaces não é uma tarefa simples, principalmente para projetistas inexperientes ou recém formados, pois cada modalidade de interação tem sua complexidade em termos técnicos, como aquisição e adaptação com novas ferramentas, linguagens, ações possíveis e etc. Além disso, é preciso verificar quais modalidades (voz, toque e gestos) podem ser usadas na aplicação, como combinar essas modalidades de forma que o ponto forte de uma complemente o ponto fraco da outra e vice-versa e também saber em qual contexto o usuário final estará inserido.

O GuiBuilder Multimodal foi desenvolvido com o intuito de tentar suprir as necessidades básicas em se implementar uma interface que utiliza voz, toque e gesto. O framework promove um desenvolvimento de interface através do modelo WYSIWYG (*What You See Is What You Get*) onde o projetista apenas define alguns parâmetros para que um componente seja multimodal. Durante a fase de criação da interface agentes supervisionam o que o designer faz e fornece um apoio, dicas, com padrões de projeto que podem ser divididos em categorias como: multimodalidade, interação com o usuário e componentes.

Palavras-chave: Interfaces Multimodais, Framework, Design Patterns, Usabilidade, IHC, Agentes Inteligentes, Java.

ABSTRACT

The interaction between humans and the computers has improved substantially during time through the evolution of interfaces of interaction. The possibility of users to interact with machines through several modalities of communication, and in a natural way, can increase the level of interest from the user and ensure the success of the application.

However, the literature of the area of multimodality has shown that developing such interfaces is not a simple task, mainly for non-experienced or recently graduated professionals, since each designer's modality of interaction has its complexity in technical terms, as acquisition and adaptation with new tools, languages, possible actions and etc. Moreover it is necessary to verify which modalities (voice, touch and gestures) can be used in the application, how to combine these modalities in a way that the stronger point of one completes the weak point of the other and vice versa, and also knowing in what context the final user will be involved.

The GuiBuilder Multimodal was developed aiming to try providing the basic needs in implementing an interface that uses voice, touch and gesture. The framework promotes an interface development through the WYSIWYG (What You See Is What You Get) model, where the designer just sets some parameters so the component is multimodal. During the interface creation phase, agents supervise what the designer does and supply support, clues, with design patterns that might be divided in categories such as: multimodality, interaction with the user and components.

Keywords: Multimodal Interfaces, Framework, Design Patterns, Usability, IHC, Intelligent Agents, Java.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API - Application Programming Interface

CARE - Complementaridade Atribuição Redundância e Equivalência

DC - Departamento de Computação

DTMF - Dual-Tone Multi-Frequency

EMMA - Extensible Multimodal and Notation Markup Language

GoF - Gang-of-Four

GUI - Graphical User Interface

GuiBM - GuiBuilder Multimodal

HTML - *HyperText Markup Language*

IDE - *Integrated Development Environment*

IHC - Interação Humano Computador

JMIS - Java Multimodal Ink and Speech

MIT - Massachusetts Institute of Technology

MMWA-ae - Multimodal Web Approach Authoring Environment

OI - OpenInterface

SALT - Speech Application Language Tags

SCE - Sony Computer Entertainment

SDK - Software Development Kit

TTS - Text To Speech

UFSCar - Universidade Federal de São Carlos

UI - User Interface

UsiXML - User Interface eXtensible Markup Language

VUI - Voice User Interface

WIMP - Windows Icons Menus Pointing

WWW - World Wide Web

WYSIWYG - What You See Is What You Get

W3C- World Wide Web Consortium

X+V - eXtensible HyperText Markup Language + Voice

XHTML – eXtensible HyperText Markup Language

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contexto e Motivação

A possibilidade de usuários interagirem com máquinas através de várias modalidades de comunicação e de forma natural, como em uma interação entre duas pessoas, é uma ideia antiga. É possível observar isto através de filmes como, por exemplo, a saga de Guerra nas Estrelas (1977-2005) e O Exterminador do Futuro (1984-2009) e seriados como, por exemplo, Perdidos no Espaço (1965-1968) e Star Trek, ou Jornada nas Estrelas, (1966-1969). Um exemplo recente é o filme Minority Report, de 2002, com direção de Steven Spielberg, no qual o personagem interage com a aplicação através de gestos e de fala.

No meio acadêmico, um dos primeiros a pesquisar a multimodalidade foi Richard A. Bolt (1980), com o trabalho intitulado *"Put-that-there"*, que utilizava duas modalidades de interação (voz e gestos). O usuário que utilizava o sistema fornecia comandos de voz e apontamento (gesto) à aplicação para escolher um objeto e posicioná-lo em um local da tela. Como, por exemplo: "coloque um círculo ali"; o sistema reconhecia o objeto a ser criado (círculo) e através do gesto (ali) o sistema identificava o local onde o círculo deveria ser inserido. Outros pesquisadores surgiram após Bolt, dentre eles: Oviatt *et al.* (1999, 2000, 2005); Suhm, Myers e Waibel (2001); Jacob *et al.* (1999), etc.

Os esforços das pesquisas iniciais em interfaces alternativas já podem ser notados em nosso cotidiano: são aplicações sensíveis ao toque, como em aparelhos celulares com *Touch Screen* (o Iphone, por exemplo), games com reconhecimento

de gestos como o Xbox em conjunto com o Kinect e em dispositivos que funcionam por comando de voz, como alguns aparelhos de GPS.

Visando um melhor entendimento sobre o que é multimodalidade e como combinar modalidades de voz, toque e gestos, Bernsen (2008) propõe a Teoria Multimodal que é baseada em notações básicas de: interação, mídia, modalidades e canais de informações. A interação, segundo Bernsen (2008), pode ser entendida como sendo a troca de informação entre o ser humano e o computador. Para entender mídias, parte-se do fato de que pessoas trocam informações através de processos físicos e a percepção humana é baseada nos sentidos humanos. Dessa forma, podemos classificar os sentidos físicos em seis tipos: visão, audição, tato, olfato, paladar e propriocepção (ou cinestesia, é um termo utilizado para nomear a capacidade em reconhecer a localização espacial do corpo, sua posição e orientação, a força exercida pelos músculos e a posição de cada parte do corpo em relação às demais, sem utilizar a visão). Existem as modalidades de entrada e de saída e são definidas pelos meios físicos e pelo seu tipo particular de representação. Já os canais de informação, podem ser classificados em: gráficos, acústicos e de toque.

No âmbito de desenvolvimento de aplicativos multimodais, Ratzka (2006) uniu a Teoria de Modalidade de Bernsen a modelos de contexto de uso, indicando ao desenvolvedor de aplicações multimodais qual modalidade deve ser usada de acordo com o contexto da tarefa que a aplicação precisa cumprir. Com isso, foi possível notar a necessidade de se criar padrões de projetos que auxiliassem na escolha e na combinação de modalidades para um contexto de tarefa específico. Ratzka, em conjunto com Wolff (2008, 2006) propôs alguns padrões que são definidos em tópicos como:

(i) o contexto do problema;
(ii) o problema em si;
(iii) as forças existentes;
(iv) a solução para o problema;
(v) as possíveis consequências;
(vi) a fundamentação do padrão;
(vii) alguns usos conhecidos.

Unindo-se a Teoria de Multimodalidade aos padrões propostos e ainda fazendo um estudo do contexto em que a aplicação multimodal está inserida, nota-

se que a construção de uma interface multimodal não é simples (Oviatt, 1999). Cada modalidade de interação tem sua complexidade em termos técnicos. Por exemplo, para a modalidade *voz* existem alguns requisitos que precisam ser cumpridos para que haja a interação como: (i) escolher e instalar um software que faça o reconhecimento da voz; (ii) criar a gramática que será utilizada pelo sistema; (iii) desenvolver um fluxo de diálogo coerente e adequado ao contexto que deixe o usuário à vontade para interagir com o sistema, etc.

Além disso, é preciso verificar quais modalidades (voz, toque e gestos) podem ser utilizadas na aplicação, bem como combinar essas modalidades de uma forma onde seja possível destacar seus pontos fortes e que estes possam ser capazes de suprir ou complementar os pontos fracos existentes, e ainda, permitir a identificação em qual contexto o usuário final estará inserido.

1.2 Objetivo

O intuito do GuiBuilder Multimodal é conseguir apoiar *designers*, que tenham pouco conhecimento na área de multimodalidade, a desenvolver interfaces que sejam capazes de interagir com o usuário final através do reconhecimento da fala, gestos e toque.

O projetista é livre para escolher dentre uma ou mais modalidades de interação que são suportadas pelo *framework* e durante a concepção da interface ele é supervisionado por agentes inteligentes, que produzem dicas de padrões de projeto que podem ser direcionadas ao tipo de interface que o projetista está desenvolvendo, quais modalidades de interação ele pode usar e o porquê de utilizá-las ou até mesmo dicas de componentes que podem facilitar a interação com o usuário.

O resultado final é a geração de um código, em Java, com uma interface que interage com o usuário de forma multimodal. O código gerado pode ser exportado para outros tipos de IDEs onde o projetista pode finalizar o desenvolvimento da sua aplicação.

1.3 Metodologia de Execução

O *framework* GuiBuilder Multimodal foi desenvolvido com a utilização dos seguintes softwares:

1. Abacus

   O *Abacus Java GUI Builder* é um construtor de GUIs, *open source*, escrito em Java e projetado ser semelhante ao modelo *Delphi/VB*. O seu objetivo é auxiliar no desenvolvimento de interfaces Java de forma rápida com geração de código XML e Java (Abacus, 2012).

2. IBM ViaVoice

   Para o reconhecimento da fala foi utilizado a API Java Speech em conjunto com o reconhecedor ViaVoice da IBM.

3. MINT

   O *MINT* é utilizado para o reconhecimento de gestos que podem ser utilizados nas interfaces geradas. São utilizados quatro gestos que correspondem aos seguintes comandos: (i) avançar, ou próximo; (ii) voltar, ou anterior; (iii) selecionar e (iv) confirmar. Os dados do reconhecimento são enviados via mensagem TCP/IP para a Abacus (Mint, 2012).

4. Jadex

   O *Jadex BDI Agent System* auxilia na programação de agentes de software inteligentes em XML e Java (Jadex, 2012).

O GuiBuilder Multimodal foi desenvolvido em três módulos: (i) Planejamento, que utiliza a interface Abacus para criar as interfaces no modelo WYSIWYG (*What You See Is What You Get*); (ii) Supervisão, que utiliza agentes inteligentes desenvolvidos na base do sistema Jadex para supervisionar o módulo de planejamento e fornecer dicas de padrões para o designer; e (iii) Código, módulo de geração de código que utiliza a Abacus, o ViaVoice IBM e o Mint para geração do código Java das interfaces elaboradas.

O desenvolvimento da pesquisa teve inicio com o estudo e levantamento dos padrões utilizados pelo GuiBM em conjunto com o módulo de planejamento, escolha de ferramentas, conceitos que deveriam ser utilizados e como o GuiBM deveria funcionar para atender as expectativas do projeto.

Após a fase de levantamento bibliográfico, teve inicio o desenvolvimento do módulo de planejamento juntamente com o módulo de código tendo em vista que o segundo complementa as ações realizadas pelo primeiro.

O terceiro passo, de desenvolvimento, foi a elaboração das dicas de padrões, considerando as possíveis interfaces que o GuiBM suportaria. Ao decorrer dessa terceira etapa foi desenvolvido o módulo de supervisão, unindo as dicas de padrões com as ações executadas pelo agente inteligente.

A última etapa correspondeu aos testes executados com o GuiBM e obtenção dos resultados realizados através de um estudo de caso, que contou com a colaboração de alunos da pós-graduação em Ciência da Computação da UFCar.

1.4 Organização do Trabalho

O trabalho encontra-se organizado da forma descrita a seguir.

No capítulo 2 é apresentado o Estado da Arte da Multimodalidade, ressaltando os conceitos de multimodalidade, os sistemas multimodais mais conhecidos e ainda os tipos de modalidade que podem ser aplicados a um software, dentre outros.

No capítulo 3 é feita uma revisão sobre Padrões de Projetos (*Design Patterns)*, a sua definição e as propriedades empregadas, a importância de se utilizar padrões de projetos na construção de softwares, os tipos conhecidos de padrões e padrões direcionados a sistemas multimodais.

O capítulo 4 apresenta um resumo sobre o conceito de *Framework* e uma breve apresentação dos *frameworks* multimodais atuais.

O capítulo 5 é composto por uma pequena descrição de Agentes Inteligentes, sua definição, arquitetura e alguns tipos de agentes.

Já no capítulo 6 é apresentado o *framework* GuiBuilder Multimodal.

Por fim os capítulos 7 e 8 contêm o Estudo de Caso e as conclusões, respectivamente.

Capítulo 2

MULTIMODALIDADE

2.1 Considerações Iniciais

A interação entre humano e computador tem melhorado substancialmente ao longo do tempo através da evolução das interfaces de interação. Andries van Dan (1997) identificou quatro gerações de interfaces:

(a) a primeira, baseada em interfaces no modo *Batch* (décadas de 50 e 60), usava cartões perfurados para a entrada de dados e impressora de linha para saída;

(b) a segunda, em modo texto, chamada de comandos *prompt* (utilizado nos períodos dos anos 60 até meados dos anos 80), possibilitou maior popularização dos computadores e de seus aplicativos;

(c) a terceira, baseada em ícones, menus e mouse, chamada de interface WIMP (*Windows Icons Menus and Pointers*), foi idealizada em 1973 pela Xerox PARC e popularizada pelo Macintosh em 1984;

(d) a quarta, chamada de Pós-WIMP, introduziu novos conceitos como toque, gestos e reconhecimento de fala e descartou a utilização de menus, formulários e barras de ferramentas.

O processo de criação de um sistema baseado em WIMP (terceira geração) é bem mais complexo que o das primeiras gerações. Construir aplicativos capazes de realizar reconhecimento de gestos, fala, toque ou até mesmo combinar mais de uma

modalidade ao mesmo tempo (quarta geração) é ainda mais complexo devido à necessidade de conhecimentos específicos que são empregados em cada tipo de modalidade de interação. Por outro lado, os recentes jogos eletrônicos que captam os movimentos do corpo têm feito muito sucesso e indicam que outras, modalidades como toque, voz e gestos, podem ser utilizadas na interação com máquinas.

2.2 Interações Multimodais

Os seres humanos interagem de forma multimodal. Por exemplo, em uma conversa entre dois amigos pode-se notar o tom de voz, assim como interação visual, por gestos ou até mesmo pela linguagem corporal. Os participantes da conversa podem sofrer interferências dependendo do ambiente em que se encontram, aumentando ou diminuindo a intensidade da voz ou dos gestos. Todas essas formas de se expressar permitem um melhor entendimento da conversa. Por exemplo, se esta interação acontece em um ambiente bastante ruidoso e um dos participantes tem dificuldades para entender o que o outro quer lhe dizer, este pode utilizar alguns gestos para tentar melhorar a comunicação ou ainda indicar um local mais adequado que facilite a interação.

Apesar da utilização de novas tecnologias, a interação humano-computador (IHC) ainda não é tão rica quanto à interação entre humanos. Existem vários obstáculos na criação de interfaces multimodais. Segundo Oviatt (2005), as pessoas podem ou não utilizar uma aplicação de forma multimodal, ou seja, apesar de existirem várias formas de interação, o usuário tem a opção de escolher entre utilizá-las ou não. Isso requer um estudo avaliativo da construção de aplicações multimodais.

A interação multimodal é diferente da forma tradicional da linguagem natural, e em muitos casos ela é simplificada substancialmente. Uma implicação é que a linguagem multimodal pode ser mais fácil de processar e também suportaria sistemas mais robustos no futuro (Talarico, 2007). É preciso compreender que uma modalidade de interação possa complementar outra, ou seja, determinadas modalidades podem não ser suficientes para interação.

Uma das preocupações na construção de aplicações multimodais é o tratamento das informações duplicadas, ou seja, quando o usuário fornece a mesma informação através de duas modalidades distintas. No entanto, pode-se notar que interagir com voz ou gestos não implica em duplicidade. Tanto um modo como o outro podem ser complementares e as informações serão compreendidas de forma mais natural pelo usuário. Segundo Dumas (2009), o grande desafio na criação de interfaces multimodais é a construção de sistemas confiáveis capazes de processar, analisar e compreender múltiplos meios de comunicação em tempo real.

2.3 Interações de Voz em Sistemas Multimodais

A facilidade em realizar uma tarefa através da interação por voz pode ser uma vantagem, pois esse tipo de interface acomoda uma grande gama de usuários, tarefas, e ambientes incluindo usuários que estão com algum sentido temporária ou permanentemente comprometidos, ou quando estão usando um dispositivo móvel, e outros casos em que uma modalidade pode não ser suficiente (Talarico, 2007).

Na busca por linguagens para o desenvolvimento de interfaces com voz, o W3C tem sido um grande apoio, que vem padronizado essas linguagens. Exemplos de linguagens para o desenvolvimento de aplicações multimodais são: VoiceXML, SALT, XFORMSMM, EMMA e o X+V.

VoiceXML

O VoiceXML, ou VXML (2004), é uma linguagem de marcação XML utilizada para aplicativos de diálogos que apresentam síntese de voz, reconhecimento de fala e entradas DTMF. Seu principal objetivo é trazer as vantagens baseadas na Web, através do desenvolvimento e da entrega de conteúdo interativo em aplicações que respondem através da linguagem falada (W3C, 2011).

A Figura 2.1 apresenta um exemplo de código do VoiceXML de *Hello Word* com fluxo de chamadas dentro de uma página. A estrutura básica para a criação ou inicialização do VoiceXML é feita através das *tags* <?xml version=“1.0” rnvofinh=”utf-8”?>, <vxml version=”2.1”> e </vxml>. Todo o corpo do sistema deve ficar entre a

segunda e a terceira *tag*. No exemplo, tem-se os elementos: (1) <link>, que funciona como uma gramática de escopo global; (2) <grammar>, que é onde o desenvolvedor especifica uma gramática do tipo gsl ou XML para o reconhecimento de voz ou DTMF (no exemplo, utiliza-se o tipo gsl). Uma gramática pode ser definida como *inline* (dentro do documento VXML) ou como autônoma, ou seja, um arquivo externo; (3) <form>, que é um contêiner para todos os itens de campo (como *Field* ou um subdiálogo) e itens de controle (como *block*); (4) <block> que é um elemento de item do <form> para o conteúdo executável, o qual só é executado se o estado do item for '*true*' e (5) <prompt>, que permite uma saída sintetizada de texto-fala para o usuário.

```
<?xml version="1.0" encoding="UTF-8"?>
<vxml version = "2.0" >

<link next="#MainMenu">
  <grammar type="text/gsl">[main back begin]</grammar>
</link>

  <form id="MainMenu">
    <block>
      <prompt bargein="false">
        This is the Hello World Main Menu.
      </prompt>
    </block>

    <field name="MeatOrPlant">
        <prompt>
            Are you a "Carnivore" or "Vegetarian".
        </prompt>
       <grammar type="text/gsl">
        <![CDATA[[
              [vegetarian plant veggie] {<MeatOrPlant "plant">}
              [meat carnivore flesh animal] {<MeatOrPlant "meat">}
            ]]]>
        </grammar>
<noinput>
        <prompt>
            I did not hear anything. Please try again.
          </prompt>
        <reprompt/>
        </noinput>
```

Continua …

```
Continuação
        <nomatch>
        <prompt>
            I did not recognize that lifestyle choice.  Please try again.
        </prompt>
          <reprompt/>
        </nomatch>
    </field>

<filled>
      <if cond="MeatOrPlant == 'meat'">
      <goto next="#Meat"/>
      <elseif cond="MeatOrPlant == 'plant'"/>
      <goto next="#Plant"/>
      </if>
</filled>
  </form>

  <form id="Meat">
    <field name="BackToMain">
    <prompt>
      PETA is coming for you, be afraid.
      If you wish to try again, please say Main.
    </prompt>
    </field>
    <filled>
      <if cond="BackToMain == 'main'">
      <goto next="#MainMenu"/>
      </if>
    </filled>
  </form>

  <form id="Plant">
    <field name="BackToMain">
    <prompt>
      Protein is the spawn of the devil.
      If you wish to try again, please say "Main".
    </prompt>
    </field>
        <filled>
      <if cond="BackToMain == 'main'">
      <goto next="#MainMenu"/>
      </if>
        </filled>
  </form>
</vxml>
```

Figura 2.1 Exemplo do código VXML. Fonte: docs.voxeo.com

Ferramentas como a *WebSphere Voice Toolkit* da IBM e o *V-Builder* da Nuance Communication utilizam o VXML para o desenvolvimento de aplicações. Os principais elementos da linguagem VXML são apresentados na figura abaixo (Talarico, 2007):

Escopo	Elementos *(Tags)* VXML
Variáveis e propriedades	Assign, meta, param, property, script, value, var, clear
Síntese de fala e saída de áudio	Audio, block, div, emp, enumerate, pros, re*promt*, sayas, break
Tratamento de erros	Catch, throw, error, help, noinput, nomatch
Fluxo de diálogo	Choice, elseif, form, goto, exit, subdialog, submit, else, link, return, menu, option, if
Entradas de usuários e gramáticas	Field, dtmf, record, filled, grammar
Integração com sistemas de Telefônia	Transfer, disconnect
Definição de documento	Vxml, initial, object

Figura 2.3 Elementos da linguagem VXML.

SALT

O padrão SALT (*Speech Application Language Tags*) é uma linguagem de marcação para o desenvolvimento de aplicações multimodais que usam a entrada e a saída da voz (reconhecimento e síntese de voz) para reconhecer e gerar falas humanas. A ideia do SALT é permitir que os usuários interajam com uma aplicação de maneiras diferentes, oferecendo uma entrada por meio de discurso, teclado, mouse, etc. As *tags* proporcionam um mecanismo de saída de discurso sintetizado, áudio, texto, vídeo e gráficos. O SALT inclui tecnologia para telefones, dispositivos móveis e computadores (Salt, 2007).

O código apresentado na Figura 2.4 é bastante simples. Ele cria uma gramática com uma regra e três itens. O desenvolvedor definiu a regra como “cidades” e colocou três itens, dentro da regra, com os nomes das cidades.

```
<salt:grammar xmlns="http://www.w3.org/2001/06/grammar">
<grammar>
    <rule>
        from
        <ruleref name="cities" />
    </rule>
    <rule name="cities">
        <oneof>
  <item> Cambridge </grxml:item>
  <item> Seattle   </grxml:item>
  <item> London    </grxml:item>
        </oneof>
    </rule>
</grammar>
</salt:grammar>
```

Figura 2.4 Exemplo de código da linguagem SALT. Fonte: SALT (2007)

XFORMSMM

O propósito do XFORMSMM é criar aplicações multimodais com o auxílio da linguagem XForms a fim de abstrair as descrições da interface com o usuário, focando no uso de interfaces gráficas e de voz (Honkala & Pohj, 2006). Segundo os autores, o modelo separa as modalidades independentemente, permitindo maior flexibilidade para que o usuário escolha qual a modalidade que poderá lhe ser mais útil.

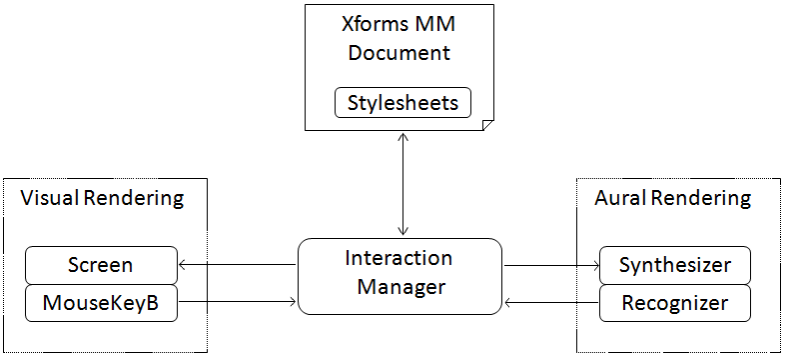


Figura 2.5 Execução em um ambiente Xformsmm. Fonte: Honkala & Pohja (2006).

A Figura 2.5 ilustra como o gerenciamento de interação trabalha com um documento Xformsmm e sincroniza o modo visual e o auditivo. Segundo Talarico (2011) sincronismo é um tópico específico para os sistemas multimodais, pois

aplicações que utilizam a multimodalidade como forma de entrada e saída de dados devem apresentar as informações ao usuário em diversos modos. Para o XFormsmm a modalidade visual pode ser considerada como espacial, enquanto que a modalidade auditiva pode ser considerada como temporal. Quando combinadas em uma interface multimodal, as interfaces precisam estar sincronizadas para não comprometer a usabilidade do sistema. Assim, ele oferece a flexibilidade de entrada e de saída das modalidades que podem ser escolhidas.

EMMA

EMMA (*Extensible Multimodal Add notation markup language*) é uma linguagem de marcação que pode ser usada em sistemas que fornecem interpretações semânticas para uma variedade de fatores, os quais incluem, mas não exclusivamente, fala, linguagem natural e gráficos. A linguagem é centrada em anotar as entradas dos usuários, que podem ser tanto a partir de um único modo ou de uma combinação de vários modos de informações, em oposição à informação que possa ter sido recolhida ao longo de várias idas e vindas da caixa de diálogo (Baggia et. al., 2007). Os componentes que geram EMMA são:

- Reconhecimento do discurso;
- Reconhecimento de caligrafia;
- Compreensão da linguagem natural;
- Interpretação de outras mídias;
- Integração multimodal.

A Figura 2.6 apresenta um exemplo de uso do Emma e o código ilustra um sistema de reserva de voo. O exemplo oferece dois resultados de reconhecimento de fala para reserva de voos com saída de Boston ou de Austin e chegada em Denver. O código é construído através de *tags*, nas quais os elementos utilizados são: (1) emma:emma, indica a versão e o namespace; (2) emma:one-of, que contém uma lista de entradas possíveis para a interpretação (no caso do exemplo o ambiente utiliza o meio acústico e reconhece a modalidade de voz); (3) emma:interpretation, oferece as possíveis escolhas do usuário e o que o sistema deve (pode) reconhecer.

```
<emma:emma version="1.0"
    xmlns:emma="http://www.w3.org/2003/04/emma"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.w3.org/2003/04/emma
     http://www.w3.org/TR/2009/REC-emma-20090210/emma.xsd"
    xmlns="http://www.example.com/example">

  <emma:one-of id="r1" emma:medium="acoustic" emma:mode="voice">
    <emma:interpretation id="int1">
      <origin>Boston</origin>
      <destination>Denver</destination>
      <date>03112003</date>
    </emma:interpretation>

    <emma:interpretation id="int2">
      <origin>Austin</origin>
      <destination>Denver</destination>
      <date>03112003</date>
    </emma:interpretation>
  </emma:one-of>
</emma:emma>
```

Figura 2.6 Exemplo de Código EMMA. Fonte: W3C (2011).

XHTML+VOICEXML (X+V)

O X+V é um padrão proposto pela W3C para o desenvolvimento de aplicações com interfaces multimodais (visual ou voz) que combina um subconjunto de elementos XHTML, uma linguagem de marcação para criar aplicações visuais, e VXML. Ambas as linguagens são integradas por XML que disparam eventos DOM para possibilitar a interação por voz (Talarico, 2007).

X+V é uma recente aquisição da família de tecnologias XML para o desenvolvimento de interfaces para o usuário. Considerando XHTML para o desenvolvimento de interfaces visuais e VoiceXML para o desenvolvimento de interfaces de voz, X+V é dedicado ao desenvolvimento de aplicações com interface multimodal.

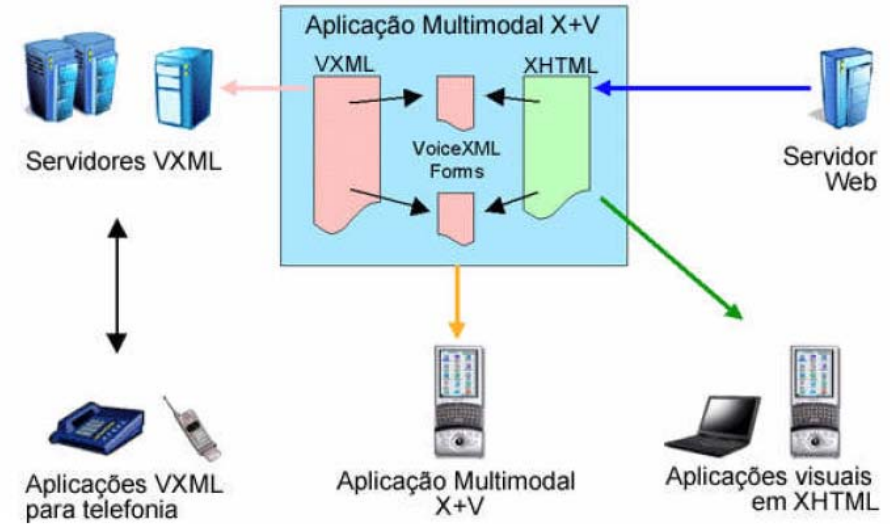


Figura 2.7 Arquitetura para o desenvolvimento de aplicações X+V. Fonte: Talarico (2007)

```
<?xml version="1.0"?>
  <!DOCTYPE html PUBLIC "-//W3C//DTD  XHTML+Voice //EN"
  "xhtml+voice10.dtd">
  <html xmlns="http://www.w3.org/1999/xhtml"
  xmlns:vxml="http://www.w3.org/2001/voicexml20"
  xmlns:ev="http://www.w3.org/2001/xml-events">
    <head>
      <title>What You See Is What You Can Say</title>
  <!-- first declare the voice handlers. -->
      <vxml:form id="voice_city">
        <vxml:field name="field_city">
          <vxml:grammar src="city.srgf" type="application/x-srgf"/>
          <vxml:prompt id="city_prompt">
            Please choose a city.
          </vxml:prompt>
          <vxml:catch event="help nomatch noinput">
            For example, say Chicago.
          </vxml:catch>
        </vxml:field>
      </vxml:form>
      <vxml:form id="voice_hotel">
        <vxml:field name="field_hotel">
          <vxml:grammar src="hotel.srgf" type="application/x-srgf"/>
          <vxml:prompt id="hotel_prompt">
            Select your hotel
          </vxml:prompt>
          <vxml:catch event="help nomatch noinput">
            For example, say Hilton.
          </vxml:catch>
          <vxml:filled>
```

Continua…

```
continuação

            <vxml:prompt>
                  You have chosen to stay at the
            <vxml:value expr="field_hotel"/>.
          </vxml:prompt>
        </vxml:filled>
      </vxml:field>
    </vxml:form>
<!-- done voice handlers. -->
  </head>
  <body>
    <h1>What You See Is What You Can Say</h1>
    <p>This example demonstrates a simple voice-enabled GUI
      hotel picker  that permits the user to provide input
      using traditional GUI input peripherals,
      or speak the same information.
    </p>
    <h2>Hotel Picker</h2>
    <form id="hotel_query" method="post" action="cgi/hotel.pl">
      <p>Select a hotel in a city:</p>
      <input name="city" type="text" ev:event="onfocus"
ev:handler="#voice_city"/>
      <input name="hotel" type="text" ev:event="onfocus"
ev:handler="#voice_hotel"/>
<!-- Declare xhtml script handlers for setting inputs -->
      <script ev:target="#voice_city" ev:event="vxml:filled">
        city = field_city;
      </script>
      <script ev:target="#voice_hotel" ev:event="vxml:filled">
        hotel = field_hotel;
      </script>
<!-- done xhtml script handlers -->
      <input type="submit" value="Submit"/>
      <input type="reset"/>
    </form>
  </body>
</html>
```

Figura 2.8 Exemplo de Código V+X. Fonte: W3C.

2.4 Interfaces com Toque

O tato é um dos sentidos mais utilizados na interação humano computador atualmente (Heikkinen *et al.*, 2009). Sendo assim, interfaces que utilizam a modalidade de toque podem ser intuitivas e bastante promissoras. É possível

encontrar no mercado telas que são sensíveis ao toque, deixando aplicações livres da utilização de teclado e de mouse e permitindo ao usuário uma interação direta.

As telas sensíveis ao toque começaram a ser desenvolvidas na segunda metade da década de 1960, um trabalho iniciado pela empresa IBM (Buxton, 2007). Os hardwares empregados nas telas sensíveis ao toque podem ser: (i) resistivo, geralmente utilizadas com canetas ou com objetos pontiagudos e (ii) capacitivo, onde o dedo do usuário é suficiente para a interação. Ainda existe a tecnologia Touchless, onde o usuário não precisa tocar na tela. Esta tecnologia faz uso de câmeras que detectam a posição para onde o usuário está apontando.

Além do toque, alguns dispositivos oferecem a interação por multi-toque. O Multi-toque (ou multitouch) consiste em um conjunto de técnicas de interação que permite que o usuário controle aplicações gráficas com vários dedos. Dispositivos com multi-toque consistem em uma tela sensível ao toque bem como um software que reconhece múltiplos toques simultâneos, ao contrário do padrão touchscreen. A utilização do multi-toque também pode permitir a interação de mais de um usuário ao mesmo tempo (Çetin, 2009).

Aplicações com interação por toque já são um grande sucesso de uso em diversos dispositivos como, por exemplo, os *Smartphones* e os *Tablets Pc*. Terminais bancários (ATM´s) e celulares sensíveis ao toque com caneta são exemplos de aplicativos que utilizam um único toque como entrada. A Figura 2.9 ilustra o *Iphone,* um exemplo de dispositivo multi-toque.

Figura 2.9 Iphone. Referência: Google imagens.

2.5 Reconhecimento de Gestos e de Linguagem Corporal

A área de reconhecimento de gestos tem apresentado grandes avanços desde o sucesso do "*Put that there*" na década de 80. Pesquisas recentes, como o Sixth Sense do MIT – Massachusetts Institute of Technology [(Mistry & Maes, 2009) e (Mistry *et al.*, 2009)], têm demonstrado como a modalidade de interação por gestos pode ser bastante intuitiva e prática no dia-a-dia. Pode-se ainda citar os avanços e os sucessos de jogos eletrônicos que estão ganhando mercado, como o PlayStation Move, da família PSP da Sony Computer Entertainment (SCE) e o inovador Project Natal da Microsoft, que em 2010 passou a adotar o nome Kinect.

2.5.1 SIXTH SENSE

O Sixth Sense é um computador *wearable* que projeta informações em superfícies, como paredes e objetos físicos, onde o usuário interage através de gestos naturais. O protótipo do Sixth Sense é composto por um pequeno projetor (ou projetor de bolso), uma câmera, um espelho e marcadores coloridos (pequenas luvas multicoloridas que são colocadas nas pontas dos dedos do usuário). O projetor, a câmera e o espelho são acoplados em um pingente que o usuário utiliza no pescoço. O software utilizado pelo Sixth Sense processa os dados que são capturados pela câmera e, utilizando técnicas de visão computacional, acompanha a localização dos marcadores coloridos. Os movimentos dos marcadores são interpretados como gestos que funcionam como instruções para a interface projetada pela aplicação. O número de dedos é limitado pelo número de cores dos marcadores e na Figura 2.10 é possível notar a utilização dos dedos: polegar e indicador (das duas mãos). Para cada dedo de cada mão, um marcador com cor diferente é utilizado. Sendo assim, o Sixth Sense pode ser multiusuário (quando se tem dois usuários utilizando cores de marcadores diferentes são necessárias oito cores) e multi-toque [(Mistry & Maes, 2009) e (Mistry et. al., 2009)].

A Figura 2.10 ilustra o protótipo do Sixth Sense e algumas aplicações suportadas pelo mesmo. À esquerda, ilustram-se alguns gestos como: aumentar, diminuir, enquadrar, Namaste, subir, descer e alguns movimentos de desenho de

imagens interpretados como comandos. Por exemplo, se o usuário quer ter acesso ao seu e-mail, basta fazer os movimentos correspondentes ao @ em alguma superfície ou, como ilustra a imagem E, se ele quer visualizar as horas basta fazer um circulo no pulso. As outras figuras ilustram: (X) todos os componentes que formam o Sixth Sense; (A) uma aplicação com mapas, na qual o usuário pode projetar a imagem em uma superfície, como uma parede, por exemplo, e utilizar gestos simples para interagir com o mapa (aumentar, diminuir, etc); (B) uma aplicação de desenho, a qual permite que o usuário crie figuras com o dedo indicador; (C) que o Sixth Sense também pode implementar uma câmera, onde o usuário pode tirar fotos do que ele vê usando o gesto de enquadramento; (D) e (E) exemplos de aplicações.

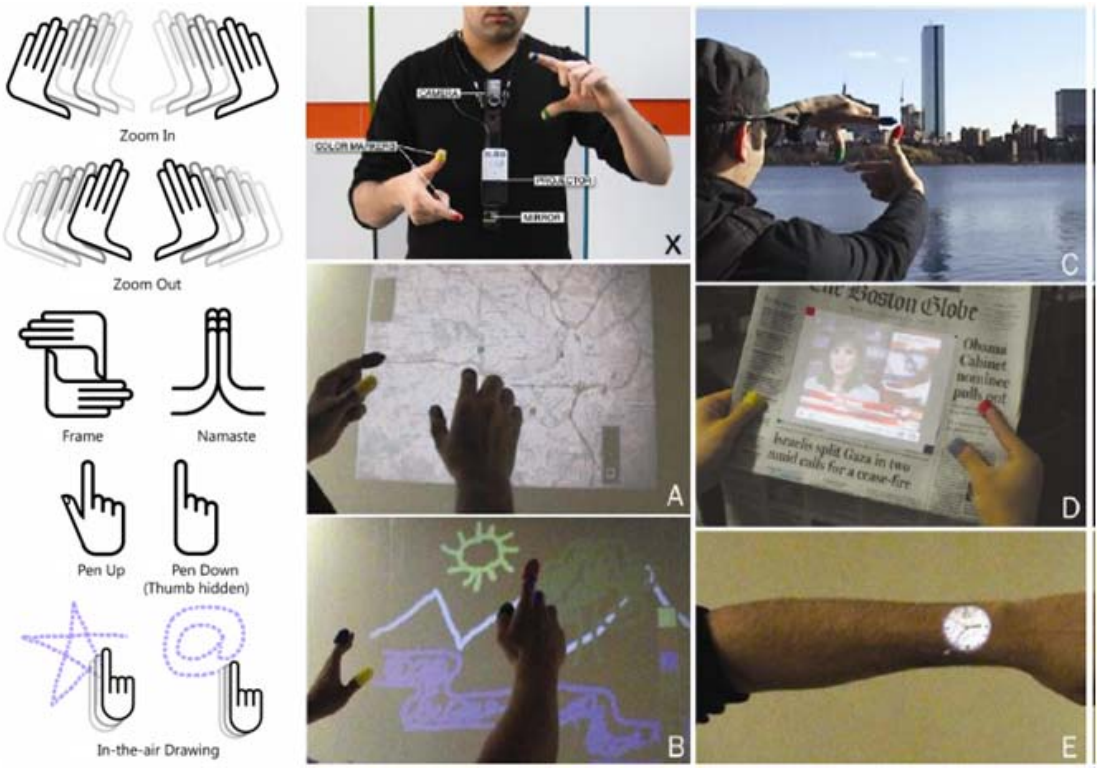


Figura 2.10 Gestos e aplicações do Sixth Sense. Fonte: Mistry e Maes (2009), Mistry et al. (2009)

2.5.2 PLAYSTATION MOVE

O *PlayStation Move* é a combinação do *PlayStation 3* com o *PlayStation Eye* e o *PlayStation Motion Controller*. O conjunto é utilizado para controlar jogos com sensor de movimentos. O PlayStation Eye é uma câmera que detecta os movimentos feitos pelo usuário, através do Motion Controller. O PlayStation Motion Controller é formado por uma varinha (stick) com um globo na cabeça, a qual possui LEDs que mudam de cor dentro da gama de cores RGB. A mudança de cores serve como um marcador de posição que é detectado pelo PlayStation Eye (Playstation Move, 2011). A Figura 2.11 ilustra os componentes do PlayStation Move: o PlayStation 3, o bastão (Motion Controller) e a câmera (PlayStation Eye).

A tecnologia de comunicação utilizada pela varinha funciona através de Bluetooth 2.0 e de porta Mini USB. É possível utilizar até quatro varinhas para a interação com os jogos. Existem vários jogos que podem utilizar o PlayStation Move como, por exemplo: um jogo de tênis, no qual o usuário utiliza o bastão para fazer os movimentos da raquete.

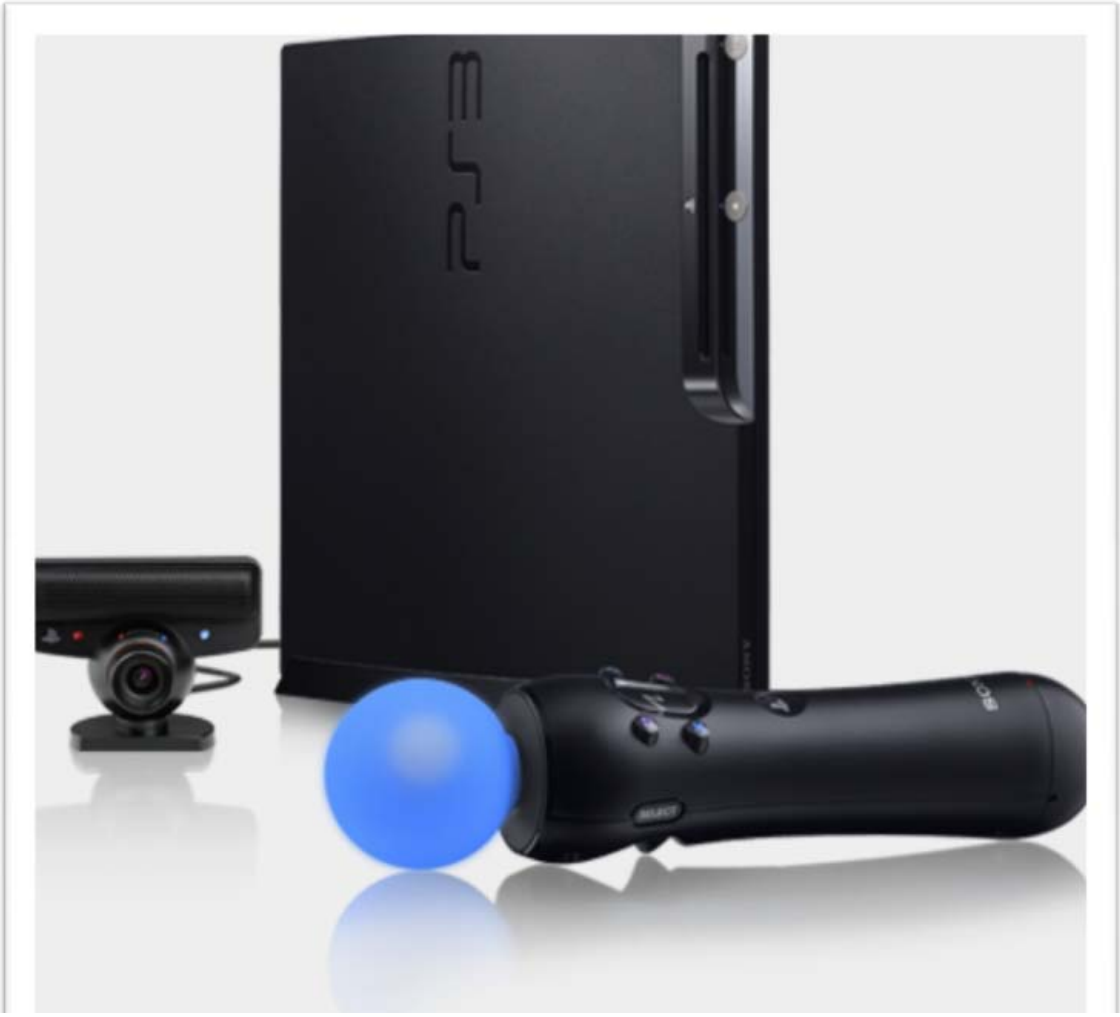


Figura 2.11 Componentes do PlayStation Move. Fonte: Playstation Move (2011).

2.5.3 KINECT

O Kinect, desenvolvido pela Microsoft com colaboração da empresa Prime Sense, utiliza uma tecnologia na qual o jogador não precisa de nenhum controle em mãos para poder interagir com os seus games favoritos. Tal projeto foi originalmente lançado com o nome de Project Natal, apresentado no E3[1] de 2009 e trouxe novidades na área de reconhecimento de gestos. Diferente do Sixth Sense e do PlayStation Move, o Kinect não precisa de nenhum marcador para o reconhecimento de gestos. A Figura 2.12 ilustra o aparelho Kinect e algumas de suas funcionalidades: (A) reconhecimento de gestos; (B) rastreamento do corpo; (C) reconhecimento facial; (D) reconhecimento de voz e (E) aparelho Kinect (Kinect, 2011).

Figura 2.12 Apresentação dos componentes do Kinect. Fonte: Kinect (2011).

O sensor de reconhecimento do Kinect dispõe de duas câmeras, uma RGB (Red, Green, Blue) e uma infravermelha. A primeira é utilizada para o

[1] E3: Electronic Entertainment Expo, conferência anual de vídeo games e entretenimento.

reconhecimento facial; a segunda é uma câmera de profundidade de baixo custo, chamada *PrimeSensor™ Reference Design,* desenvolvida pela empresa israelense PrimeSense (PrimeSense, 2011). Como resultado obtém-se um mapa de profundidade com boa precisão.

Além disso, o Kinect faz o reconhecimento de voz através de um microfone embutido. O usuário participa literalmente do jogo, podendo interagir com comandos de voz e receber respostas do jogo através dessa modalidade ou também por gestos através dos movimentos do seu próprio corpo (Kinect, 2011).

Existem algumas configurações básicas para o bom funcionamento do Kinect. A Figura 2.13 apresenta essas configurações. (F) indica que é preciso posicionar o sensor abaixo da televisão, em local plano e seguro próximo à borda; (G) indica que é necessário que o jogador fique a uma distância na qual o Kinect consiga rastrear seu corpo inteiro (caso exista mais de um jogador é preciso que eles fiquem a uma pequena distância um do outro); (H) indica que para movimentar o ponteiro na tela basta mexer com as mãos; (I) indica que sempre que o usuário for jogar é preciso ativar o seu avatar. Para isso basta olhar para o Kinect e acenar, e o equipamento vai realizar o reconhecimento e ativar o avatar correspondente. Caso o usuário esteja interagindo pela primeira vez, o Kinect criará um novo avatar e fará o seu registro.

Figura 2.13 Configurações básicas para o uso do Kinect. Fonte: Kinect (2011).

2.6 Considerações Finais

Neste capitulo, foi feito um levantamento sobre sistemas multimodais, com seções relevantes a três modalidades: voz, toque e gesto. As interfaces com tecnologia multimodal começaram a serem desenvolvidas em meados dos anos 80, com o surgimento do "*Put that there*". Hoje é possível encontrar vários dispositivos no mercado que são sucesso de uso, como os *smartphones*, que unem as tecnologias de reconhecimento de voz para a realização de chamadas na agenda e outros aplicativos e telas sensíveis ao toque. Um dos grandes sucessos do mercado de games, que vêm conquistando usuários, são os consoles que realizam reconhecimento de gestos para a interação do jogador com a aplicação.

Os avanços na tecnologia e a busca por uma melhor interação Humano-Computador têm alavancado as pesquisas nessa área. Entretanto, para intensificar as pesquisas na área e disseminar ainda mais o uso de aplicativos com interfaces multimodais, é importante que tais aplicativos sejam construídos seguindo-se padrões de desenvolvimento de projetos. Dessa forma, o próximo capítulo abordará padrões de projeto e como esses podem auxiliar no desenvolvimento, na criação e na interação das interfaces multimodais.

Capítulo 3

PADRÕES DE PROJETO

3.1 Considerações Inicias

Desenvolver softwares de qualidade não é uma tarefa fácil. É preciso esforço e conhecimento da área para modelar, estruturar e implementar qualquer tipo de aplicação. Para diminuir o esforço de desenvolvimento, projetistas experientes utilizam artifícios de programação, como a reutilização de códigos baseados em métodos já bem concebidos e com soluções comprovadas, conhecidos como "Padrões de Projetos" ou, em inglês, *Design Patterns*.

A importância de se usar padrões no desenvolvimento de sistemas complexos há muito foi reconhecida em outras disciplinas. O conceito de *Design Patterns* surgiu com Christopher Alexander, voltado para arquiteturas de cidades. Segundo Alexander "*Cada padrão descreve um problema no nosso ambiente e o cerne da sua solução, de tal forma que você possa usar essa solução mais de um milhão de vezes, sem nunca fazê-lo da mesma maneira*" (Alexander, 1964). Para a disciplina de programação orientada a objetos, Gamma *et al.* (2005) diz que "*Padrões de projeto são descrições de objetos e classes comunicantes que precisam ser personalizadas para resolver um problema geral de projeto num contexto particular*". Padrões de projeto de software permitem descrever fragmentos de problema e reusar ideias, o que ajuda desenvolvedores inexperientes a se nivelar à experiência de outros (Larman, 2007).

3.2 Padrões de Projeto

Existem muitas definições para padrões de projeto, como: "*um padrão fornece soluções viáveis para todos os problemas conhecidos que surgem no decorrer do projeto. É uma sequência de bits de conhecimento escrito em um estilo e dispostos em uma ordem que leva um designer a fazer perguntas (e respostas) a questões na hora certa*" (Beck & Cunningham, 1987); "*um padrão é uma descrição denominada de um problema e a solução que pode ser aplicada a novos contextos*" (Larman, 2007); "*um padrão de projeto é uma regra de conexão de um problema de projeto comum com uma solução comprovada e uma descrição dos contextos e das condições em que este padrão é aplicável*" (Ratzka, 2008). Ou seja, pode-se afirmar que um padrão de projeto nada mais é do que uma dupla problema/solução, testada e comprovada, que pode ser aplicada a um determinado contexto.

Christopher Alexander cunhou o termo *patterns*, mas a sua inserção na computação partiu de Kent Beck e Ward Cunningham na segunda metade da década de 80 (Beck & Cunningham, 1987). Na ocasião, Beck e Cunningham apresentaram cinco padrões, para projetar interfaces de usuário usando janelas, na linguagem Smalltalk. Os cinco padrões propostos foram: (i) Janela por Tarefa (*Window Per Task*), que decide o que está disponível na janela e o que deve ser feito na mesma; (ii) Pouco Painéis por Janela (*Few Panes Per Window*) e (iii) Painéis Padrão (*Standard Panels*), os quais dividem cada janela em painéis; (iv) Menus Curtos (*Short Menus*), e (v) Substantivos e Verbos (*Nouns and Verbs*), que determina as ações que serão realizadas dentro de cada painel.

3.3 Descrevendo Padrões de Projeto

Segundo Gamma *et al.* (2005)[2], um padrão precisa possuir pelo menos quatro elementos: (i) o **nome do padrão**, com uma ou duas palavras que definem o problema, a solução e as consequências do padrão; (ii) o **problema**, que descreve

[2] Gamma et al. também é conhecido como a Gangue-dos-quatro (Gang-of-Four ou GoF), termo usado como referência ao livro *Design Patterns,* escrito pelos quatro desenvolvedores Erich Gamma, Richard Helm, Ralph Johnson e John Vlissides.

qual é o ponto chave do problema; (iii) a **solução**, que oferece uma solução geral para o contexto do problema; (iv) as **consequências**, os resultados e as análises esperadas ao aplicar o padrão.

Padrões podem ser classificados em três tipos: (i) padrões de projeto; (ii) padrões arquiteturais e (iii) linguagens de padrões (Schmidt & Buschmann, 2003). Segundo Gamma *et al.* (2005), os padrões de projeto devem seguir uma estrutura que descreve os seus elementos. A tabela 3.1 apresenta os elementos de um padrão de projeto.

Elemento	Função
Nome e Classificação do Padrão	Descreve o nome do padrão; um bom nome é vital para o seu uso. Existem três tipos de classificação: (i) Criação, responsável pelo processo de criação dos objetos; (ii) Estrutural, responsável pela composição das classes e objetos; (iii) Comportamental, caracteriza as maneiras como as classes e os objetos interagem entre si.
Intenção e Objetivo	Uma breve descrição respondendo a perguntas do tipo: o que faz o padrão de projeto? Quais os seus princípios e a sua real intenção? De que tópico ou problema particular de projeto ele trata?
Também conhecido como	Outros nomes conhecidos para o mesmo padrão
Motivação	Descreve um cenário com um determinado problema e como as estruturas de classes e os objetos no padrão podem solucionar tal problema.
Aplicabilidade	Descreve em quais situações o padrão pode ser utilizado.
Estrutura	Pode ser uma representação gráfica das

	classes do padrão usando UML.
Participantes	As classes e os objetos que participam do padrão e as suas responsabilidades.
Colaborações	Informa como as classes participantes colaboram entre si para executar suas responsabilidades.
Consequências	Descreve os resultados esperados a partir da aplicação do padrão.
Implementação	Descreve as armadilhas, sugestões ou técnicas que são necessárias para implementar o padrão. Ainda pode especificar algumas peculiaridades da linguagem.
Exemplo de código	Fragmentos ou blocos de códigos que podem ilustrar como deve ser a implementação do padrão em alguma linguagem de programação.
Usos conhecidos	Exemplos da utilização do padrão em sistemas reais. Recomenda-se a inclusão de pelo menos dois exemplos de domínios diferentes.
Padrões Relacionados	Descreve outros padrões de projeto que têm relação, de alguma forma, com o padrão mencionado.

Tabela 3.1 Elementos que descrevem os Padrões de Projeto. Fonte: Gamma *et al.* (2005)

Dada a importância dos padrões de projetos, bem como os elementos que podem fazer parte de sua descrição, Gamma *et al.* (2005) apresenta um catálogo com vinte e três padrões divididos em três classes: (i) Criação, abstraem o processo de instanciação. Eles ajudam a tornar um sistema independente de como seus objetos são criados, compostos e representados. Esta classe possui cinco padrões (Abstract Factory, Builder, Factory Method, Prototype e Singleton); (ii) Estrutural, se preocupam com a forma como classes e objetos são compostos para formar

estruturas maiores. Esta classe possui sete padrões associados (Adapter, Composite, Brigde, Decorator, Facade, Flyweight e Proxy); (iii) Comportamental, preocupam-se com algoritmos e a atribuição de responsabilidades entre objetos. Esta classe contém onze padrões associados (Chain of Responsability, Command, Interpreter, Interator, Mediator, Memento, Observer, State, Template Method, Strategy e Visitor). A tabela 3.2 reune todos os vinte e três padrões GoF.

Padrões GoF				
Abstract Factory	Builder	Factory Method	Prototype	Singleton
Adapter	Composite	Brigde	Decorator	Facade
Flyweight	Proxy	Chain of Responsability	Command	Interpreter
Interator	Mediator	Memento	Observer	State
Template Method	Strategy	Visitor		

Tabela 3.2 Conjunto de padrões GoF. Fonte: Gamma *et. al.* (2005)

3.4 Padrões para Interação Humano-Computador

Na Interação Humano-Computador, Coram e Lee (1996) foram os primeiros a publicar um esboço de uma linguagem de padrões para o design centrado na interface do usuário. O objetivo em utilizar os padrões propostos era o de aperfeiçoar a experiência do usuário, oferecendo um ambiente de interação agradável e produtivo. Em 1998, Tidwell propôs uma linguagem de padrões que tinha como objetivo abordar problemas gerais na elaboração de interfaces de software complexas e interativas. O estudo resultou em 60 padrões separados por 10 descrições.

Em 2000 Welie e Traetteberg apresentaram um conjunto de padrões de interação para o usuário. Nesse estudo são demonstrados vinte padrões de interação. Para propor esses vinte padrões, Welie e Traetterberg (2000) se

basearam nos estudos de Norman (1988) quanto à usabilidade e ao desafio do design ao projetar sistemas. Norman propõe oito princípios de usabilidade:

1. **Visibilidade**: ao olhar, o usuário pode definir o estado do artefato e as alternativas de ação. "*Deixe as peças relevantes visíveis*";

2. **Affordance**: um bom projetista sempre se assegura de que as ações apropriadas sejam perceptíveis e as inapropriadas invisíveis;

3. **Mapeamento Natural**: utilizar as analogias físicas e os padrões culturais para auxiliar a melhor compreensão de utilização do sistema;

4. **Restrições**: projetar de tal forma que nenhuma ação impossível possa ser executada;

5. **Modelo Conceitual**: um designer fornece um bom modelo conceitual para o usuário, com consistência na apresentação de operações e nos resultados, e um sistema coerente e consistente de imagens;

6. **Feedback**: o usuário recebe pleno e contínuo retorno de informações sobre o resultado das ações;

7. **Segurança**: garantir que o usuário não esteja executando uma tarefa da qual ele não tenha certeza do que está fazendo;

8. **Flexibilidade**: controle explícito do sistema, todos os meios de se chegar ao objetivo da tarefa.

Os vinte padrões de interação propostos por Welie e Traetteberg (2000) têm como elementos: (i) problema, (ii) princípios de usabilidade, (iii) contexto, (iv) solução, (v) fundamentação e (vi) exemplo. Todos os vintes padrões são apresentados a seguir.

1. **Wizard**: o princípio de usabilidade utilizado é o de **Visibilidade**. Esse padrão pode ser utilizado quando o usuário precisa executar várias tarefas antes de realizar sua meta principal. Um exemplo de utilização desse padrão são os instaladores de software como o Installshield. A Figura 3.1 ilustra um instalador da ferramenta GOM Player, que é um software para exibição de vídeos. São necessárias quatro etapas para a instalação do aplicativo, sendo elas: (i) tela de apresentação do aplicativo GOM Player; (ii) tela na qual o usuário aceita os termos da licença de uso do software; (iii) hierarquia de todos os componentes que serão instalados. Nessa tela, o usuário tem a opção de desmarcar alguns componentes na instalação; (iv) após a realização das fases preliminares, o aplicativo estará pronto para ser instalado.

2. **Grid Layout**: tem como princípio de usabilidade os **Modelos Conceituais**. É um padrão utilizado quando o usuário precisa entender rapidamente as informações apresentadas e agir em função dessas informações. Um exemplo de uso desse padrão são as páginas de configuração. A Figura 3.2 apresenta a página de configuração do Word 2003, na qual vários objetivos são esteticamente agrupados de forma que o usuário possa decidir quais opções ele deseja para alcançar o seu objetivo final.

3. **Progress**: tem como princípio de usabilidade o **Feedback**. É usado para dar suporte ao usuário no que diz respeito ao progresso da tarefa executada. Várias aplicações utilizam esse padrão e um exemplo bastante prático é o da transferência de um arquivo de um local para outro. A Figura 3.3 ilustra um arquivo de música sendo transferido de um local para outro.

4. **Shield**: seu princípio de usabilidade é a **Segurança**. Tem como objetivo confirmar se o usuário realmente deseja realizar a tarefa. Um exemplo de utilização desse padrão são as caixas de confirmação de uma dada execução, como por exemplo, para apagar um arquivo, como ilustra a Figura 3.4.

5. **Preferences**: seu princípio de usabilidade é a **Flexibilidade**. Tem como objetivo fornecer ao usuário a possibilidade de configurar o sistema de acordo com suas preferências. Esse padrão é relacionado com o padrão Grid Layout. A Figura 3.5 ilustra um exemplo de aplicação que utiliza esse padrão. Na imagem, pode-se notar que o usuário é livre para escolher a formatação de uma imagem, como: cor, formato, estilo, etc.

6. **Contextual Menu**: seu princípio de usabilidade é a **Visibilidade**. Tem como objetivo deixar visível ao usuário as opções que ele tem na ferramenta através de um menu, de acordo com o contexto. Por exemplo, a barra de menu do Paintbrush fornece ao usuário as opções para desenho na ferramenta. A Figura 3.6 ilustra esse menu.

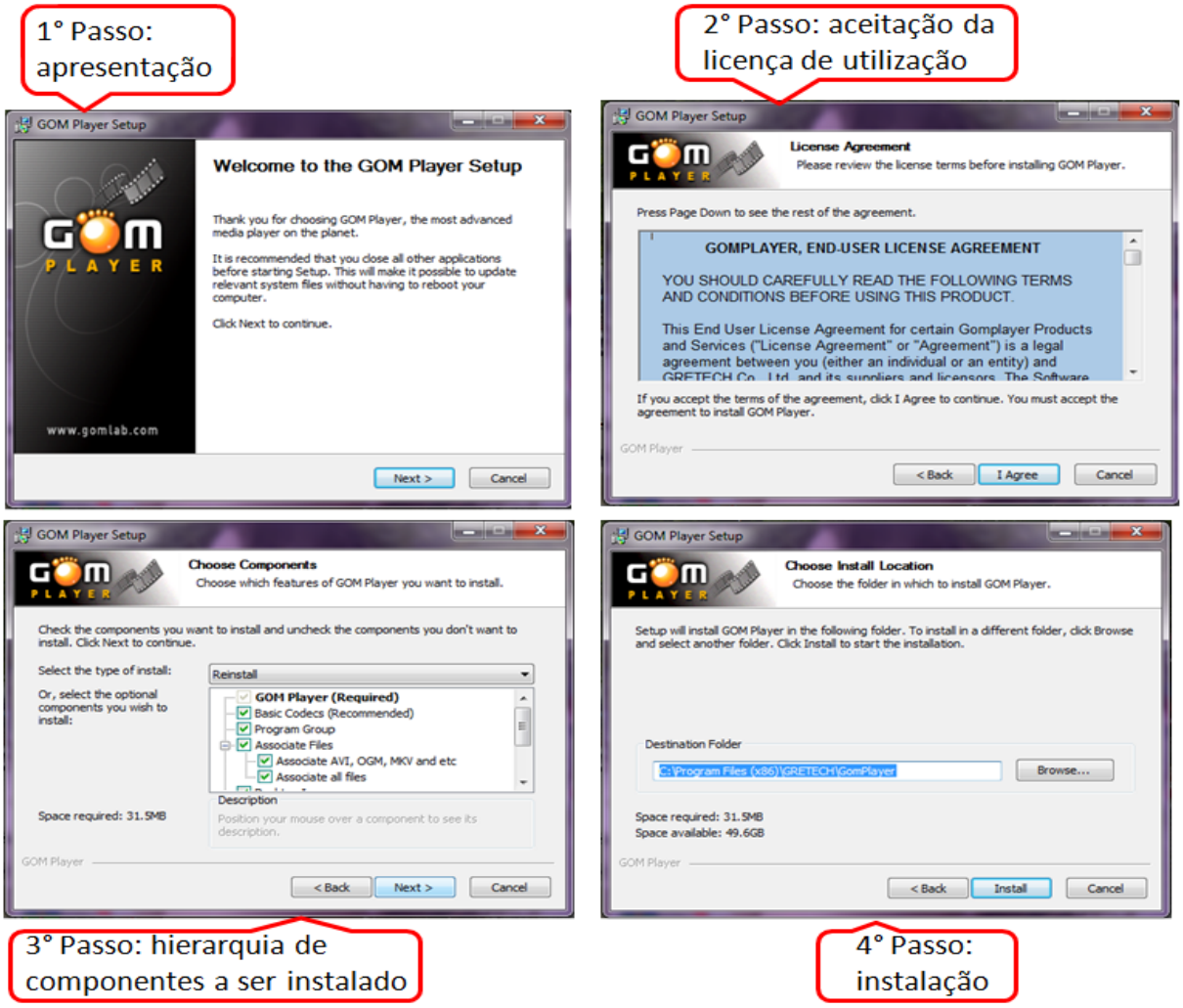


Figura 3.1 Installshield da ferramenta GOM.

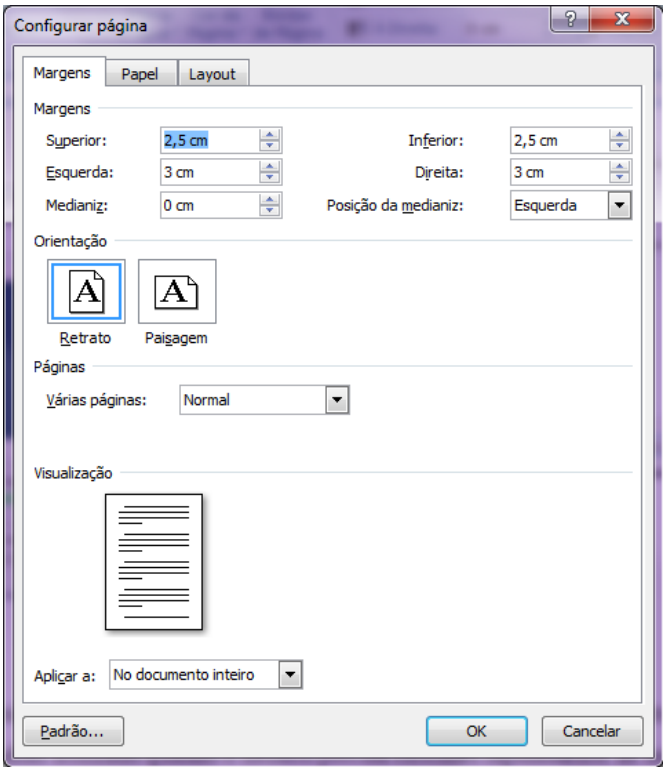


Figura 3.2 Página de configuração do Word 2003.

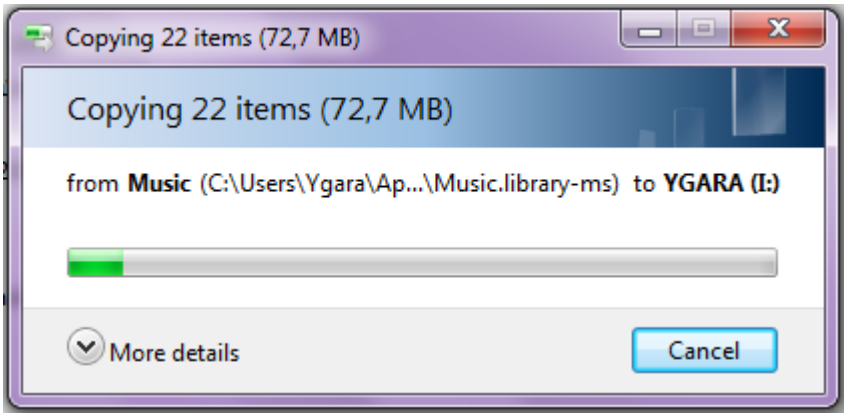


Figura 3.3 Caixa de progresso da transferência de um arquivo de um local para outro.

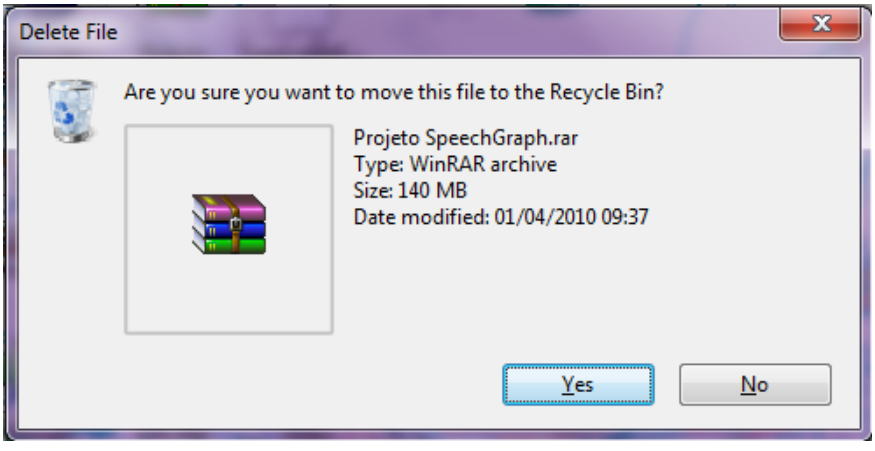


Figura 3.4 Tela de verificação se o usuário deseja realmente deletar o arquivo escolhido.

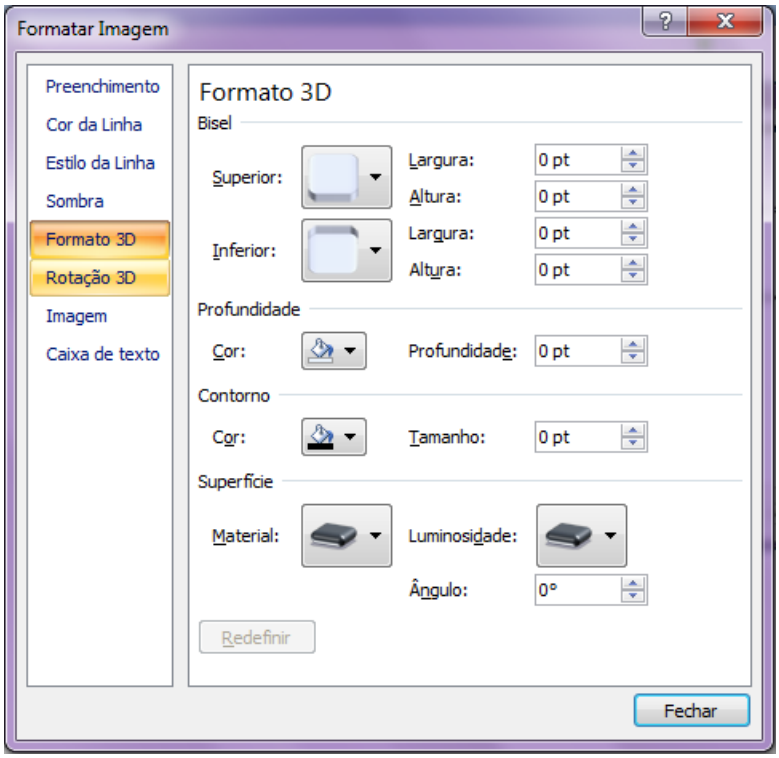


Figura 3.5 Caixa de formação de imagem do Power Point 2003.

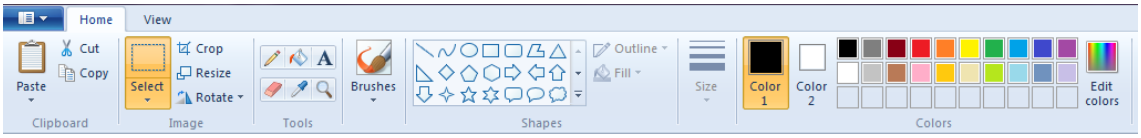


Figura 3.6 Menu de opções do Paintbrush.

7. **Focus**: utiliza o princípio de usabilidade **Restrição**. Seu objetivo é manter a atenção do usuário em aplicativos nos quais exista muita informação em uma determinada área de atuação. Um exemplo de utilização desse padrão, para quem desenvolve aplicações no NetBeans, é apresentado na Figura 3.7. Nessa imagem, pode-se notar que o foco atual do usuário está no desenvolvimento gráfico da interface, pois a aba com opção *DesenhoAboutBox.java* (na parte superior) se encontra ativada. Mesmo o usuário tendo outras abas e opções, ele se encontra focado na área gráfica da interface que está projetando. O detalhe em vermelho na figura indica a aba ativada e os detalhes em verde, a abas desativadas.

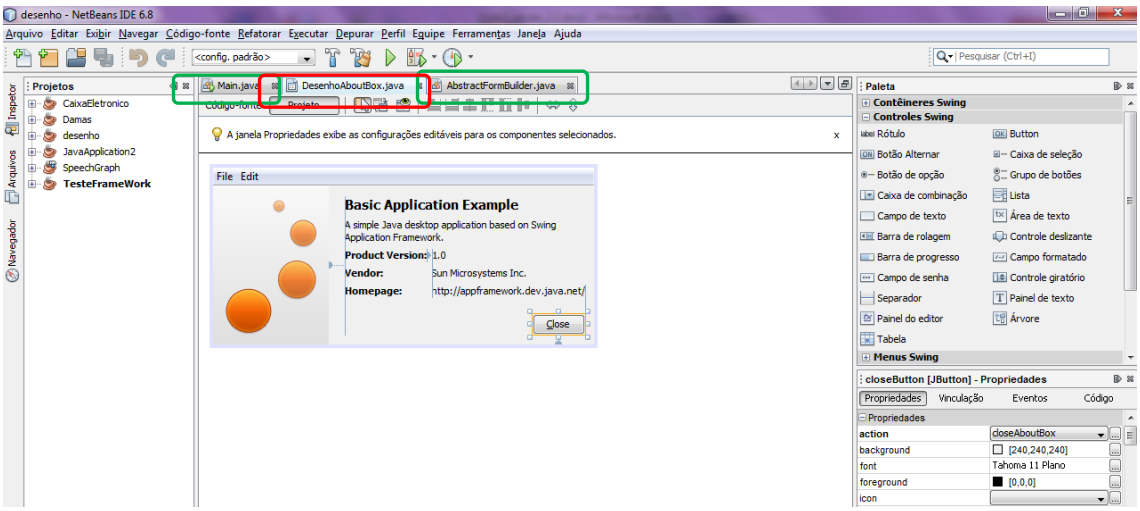


Figura 3.7 Exemplo do NetBeans 6.8 utilizando o padrão Focus.

8. **Unambiguous Format**: seu princípio de usabilidade é a **Restrição**. Ele é utilizado quando um usuário precisa inserir dados com um formato padrão, a ferramenta deve ser capaz de restringir as entradas fornecidas pelo usuário para que os dados não sejam ambíguos. Um exemplo de utilização desse padrão é a configuração de data e de hora do Windows. Na Figura 3.8 é possível notar um padrão do tipo dia/mês/ano para data e dois padrões, um de 12 horas (AM/PM) e um de 24 horas, para hora. Na imagem o padrão do horário é de 24 horas.

9. **Navigation between Spaces**: tem como princípio de usabilidade o **Mapeamento Natural**. Esse padrão é utilizado quando é necessário fornecer muitas informações para o usuário em espaço limitado. A ideia é separar a informação por grupos, de forma que o usuário saiba, intuitivamente, qual grupo ele precisa usar. Na Figura 3.8 pode-se notar também a utilização desse padrão. As abas *Numbers, Currency, Time* e *Date* são espaços de navegação que o usuário pode acessar a fim de obter informações/configurações do sistema.

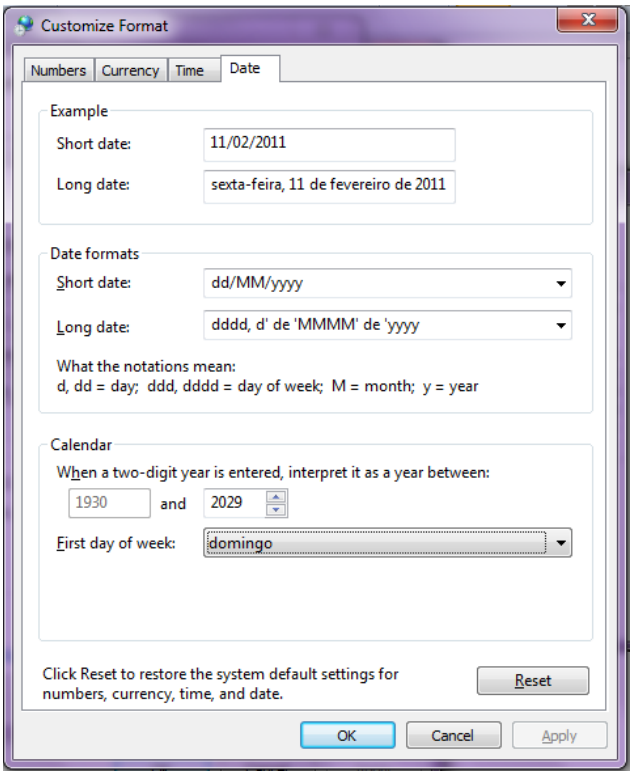


Figura 3.8 Painel de configuração de data e de hora do Windows 7.

10. **Like in the real World**: apresenta como princípio de usabilidade os **Modelos Conceituais**. Esse padrão é utilizado quando o usuário precisa interagir com um sistema a partir de um objeto que se assemelha a um objeto do mundo real. Um exemplo de utilização desse padrão são os celulares *touchscreen* que usam canetas para a interação com o usuário. A Figura 3.9 ilustra um celular *touchscreen* que utiliza caneta para fins de interação.

11. **Preview**: tem como princípio de usabilidade o **Mapeamento Natural**. Esse padrão é utilizado para fornecer ao usuário uma visão geral do estado da sua interação. Um exemplo prático são ferramentas de montagem de apresentação, como o Power Point, na qual o usuário pode verificar seus slides anteriores à esquerda, bem como, visualizar como ficaria a modificação, caso queira mudar o design da apresentação. A Figura 3.10 ilustra esse exemplo.

Figura 3.9 Celular que utiliza caneta para interação com o usuário.

12. **Favourites**: tem como princípio de usabilidade a **Flexibilidade**. Seu objetivo é unir itens que o usuário utiliza com maior frequência. Um exemplo clássico de uso desse padrão são os navegadores web. A grande maioria deles oferece uma barra de favoritos, no qual os usuários guardam os endereços das páginas que mais acessam, como ilustra a Figura 3.11.

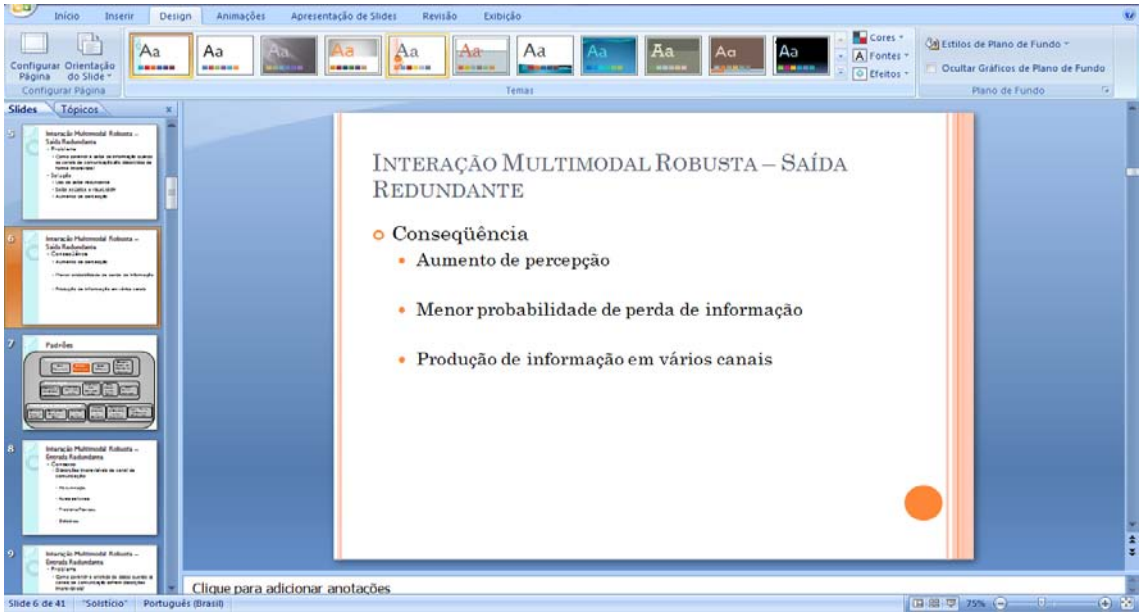


Figura 3.10 Exemplo do padrão Preview na ferramenta Power Point.

13. **Command Area**: tem como princípio de usabilidade a **Visibilidade**. É utilizado quando a interface deve auxiliar o usuário a encontrar comandos que possam ser ativados na ferramenta de acordo com a sua necessidade. Em geral, esses comandos devem ser colocados no topo da aplicação ou do lado esquerdo, garantindo, assim, boa visibilidade. O exemplo da Figura 3.10 também utiliza esse padrão. Também é possível encontrar comandos para a mudança de fonte (como: cor, tamanho, efeito, etc.) na montagem da apresentação.

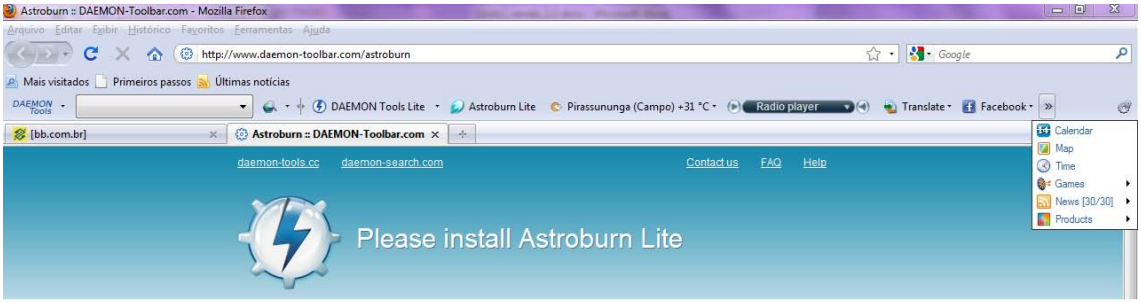


Figura 3.11 Barra de Favoritos padrão do browser Mozilla.

14. **Container Navigation**: apresenta como princípio de usabilidade o **Mapeamento Natural**. Esse padrão é utilizado quando o usuário necessita encontrar um item em uma coleção de contêineres. A solução consiste, então, em dividir a tela em três partes, onde duas delas são destinadas a fornecer e a receber informações que o usuário necessita e a terceira é destinada ao resultado da navegação. Um exemplo de ferramenta que utiliza esse padrão é o Microsoft Outlook. À esquerda da Figura 3.12 existe (i) uma hierarquia de pastas para o usuário navegar; (ii) ao lado, na parte superior, os e-mails recebidos; (iii) na parte inferior, um campo para a criação de e-mails. Esse padrão é relacionado com o *Grid Layout*.

15. **Setting Attributes**: apresenta como princípio de usabilidade a **Visibilidade**. Esse padrão é relacionado ao *Command Area* e é utilizado para exibir os atributos dos objetos que o usuário está utilizando, deixando-o livre para personalizá-los. Um exemplo disso são as barras de ferramenta do Word. O usuário, ao escrever um texto, tem a opção de modificar a fonte alterando o tamanho da letra, o tipo, a cor, o espaçamento entre as linhas, o alinhamento, etc, como ilustra a Figura 3.14.

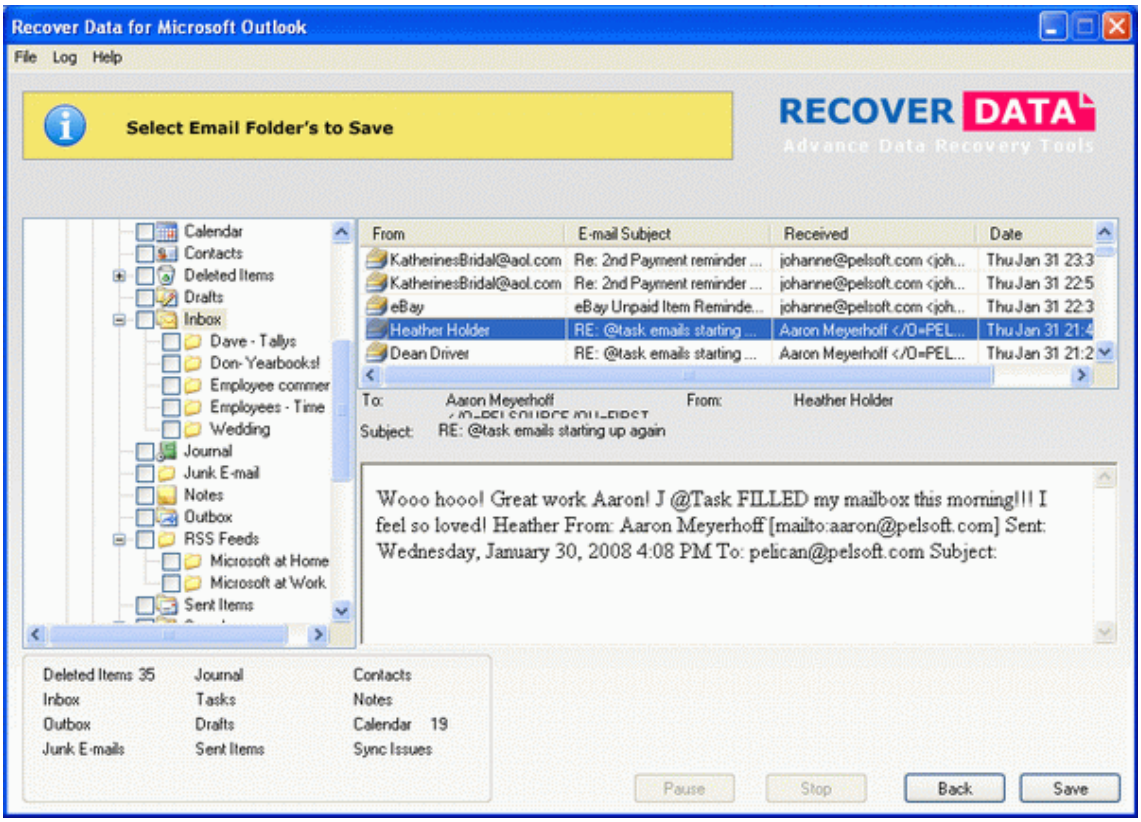


Figura 3.12 Microsoft Outlook. Fonte: Google imagens.

16. **Warning**: apresenta como princípio de usabilidade a **Segurança**. Esse padrão fornece um aviso ao usuário de acordo com a tarefa que ele está executando. Por exemplo, se o usuário utiliza o Word 2003 para criar um texto e logo após deseja fechar a ferramenta, um *Warning* aparece para confirmar se o usuário deseja ou não armazenar tal arquivo, como ilustra a Figura 3.13. Nesse exemplo, o usuário tem as opções de: (i) salvar o arquivo e em seguida fechar a aplicação; (ii) não salvá-lo e fechar a aplicação e (iii) cancelar a operação de fechar a ferramenta de texto. Esse padrão é relacionado com o *Shield*.

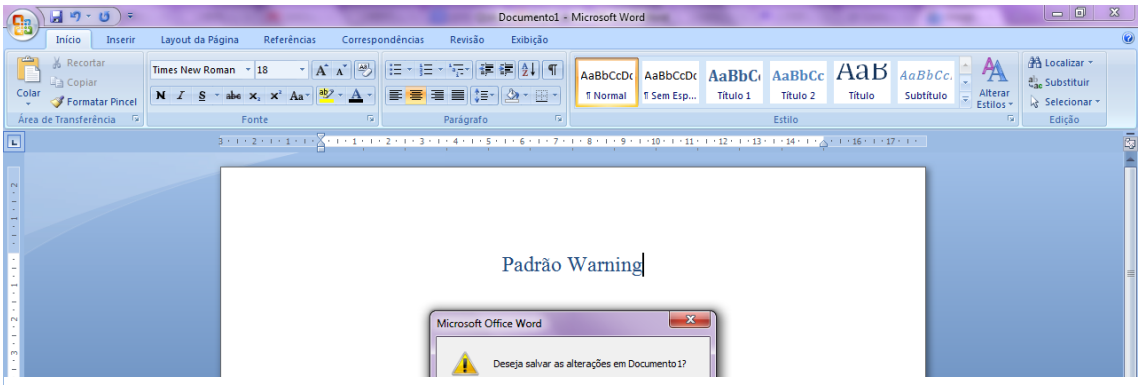


Figura 3.13 Exemplo do padrão Warning no Word 2003.

17. **Hinting**: apresenta como princípio de usabilidade a **Visibilidade**. É utilizado quando o usuário tem a necessidade de selecionar determinadas funções. O padrão deve oferecer o contexto e a funcionalidade de cada operação e formas simples e intuitivas de acesso a estas operações. Um exemplo de utilização é quando o usuário deseja modificar a fonte, ainda usando o Word 2003. Ao clicar na aba correspondente, abre-se uma lista com todos os formatos de fontes existentes, incluindo também a estética da mesma. Na Figura 3.15 é possível notar duas maneiras para a mudança da fonte escolhida: uma se dá através da barra de ferramentas (ao lado esquerdo da imagem) e a outra, quando o usuário seleciona o texto e clica com o botão direito do mouse (ao lado direito da imagem), escolhendo a opção “fonte”.

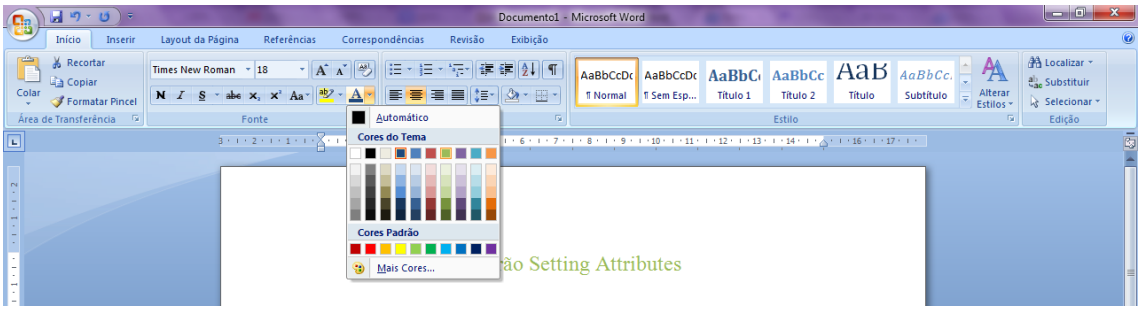


Figura 3.14 Exemplo do padrão Setting Attributes no Word 2003.

18. **Mode Cursor**: tem como princípio de usabilidade o **Feedback**. É utilizado quando o usuário cria ou modifica um objeto e precisa verificar qual função de edição está usando no momento. Para que o usuário obtenha essa informação, o cursor do mouse deve modificar a sua forma de acordo com a função selecionada pelo usuário. Ferramentas de edição de imagens utilizam esse padrão com muita frequência. A Figura 3.16 ilustra uma ferramenta desse tipo, na qual o usuário seleciona uma área através da ferramenta "varinha mágica" e o cursor do mouse aparece na imagem com o formato de uma varinha.

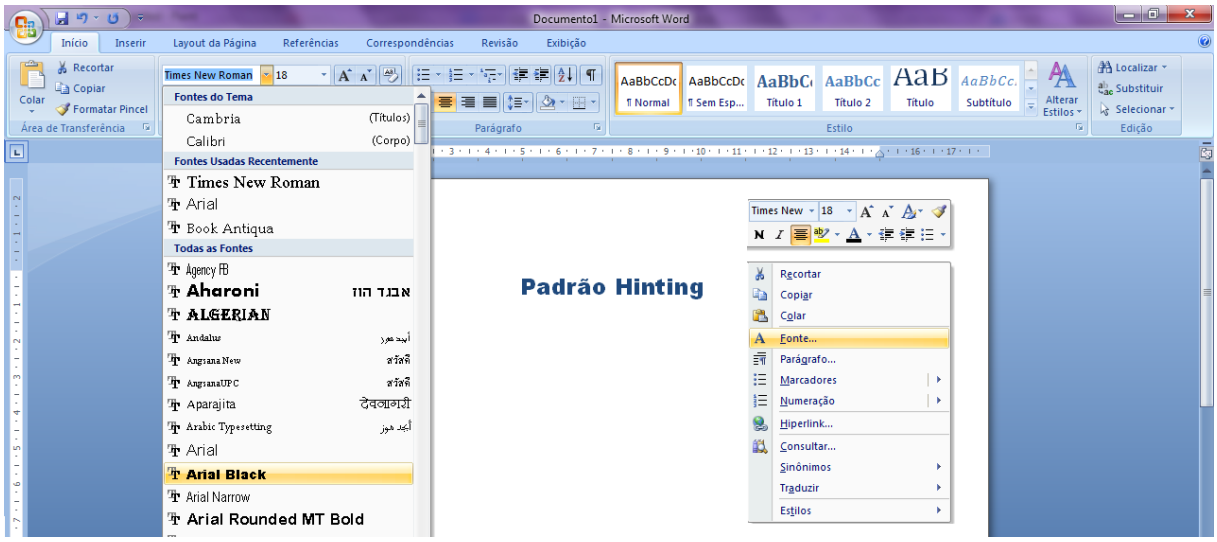


Figura 3.15 Exemplo do padrão Hinting no Word 2003.

19. **List Browser**: tem como princípio de usabilidade o **Mapeamento Natural**. Esse padrão é utilizado quando se tem um conjunto de informações (lista de opções) que devem ser fornecidas ao usuário em espaço reduzido. A ideia é separar os assuntos por ordem de relevância ou por contextos que o usuário possa mapear e descobrir onde se encontra a informação desejada. Um exemplo de uso é a página principal da Globo, ilustrada na Figura 3.17. Nota-se que as informações são separadas em listas de assuntos mais relevantes (notícias, esportes e entretenimento). A partir dessa lista, o usuário pode acessar outras informações que não estão sendo exibidas no momento.

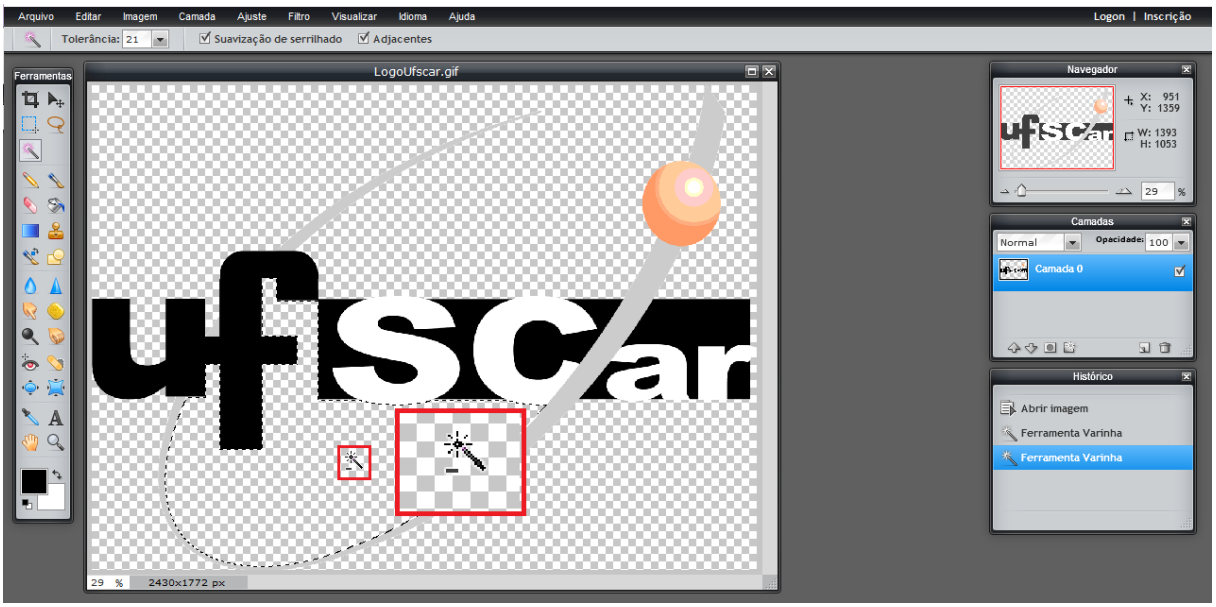


Figura 3.16 Ferramenta de edição de imagens online. Fonte: pixlr.com/editor/?loc=pt-br.

20. **Continuous Filter**: tem como princípio de usabilidade o **Feedback**. É utilizada quando o usuário necessita encontrar um item em um conjunto ordenado. Esse padrão permite restringir a busca dinamicamente dependendo do feedback fornecido pelo filtro da lista do usuário. O browser Mozilla utiliza esse padrão e na Figura 3.18, pode-se notar que, imediatamente após o usuário digitar WWW, já aparece uma lista de opções de páginas que apresentam o mesmo início.

Figura 3.17 Exemplo do padrão List Browser no site da Globo

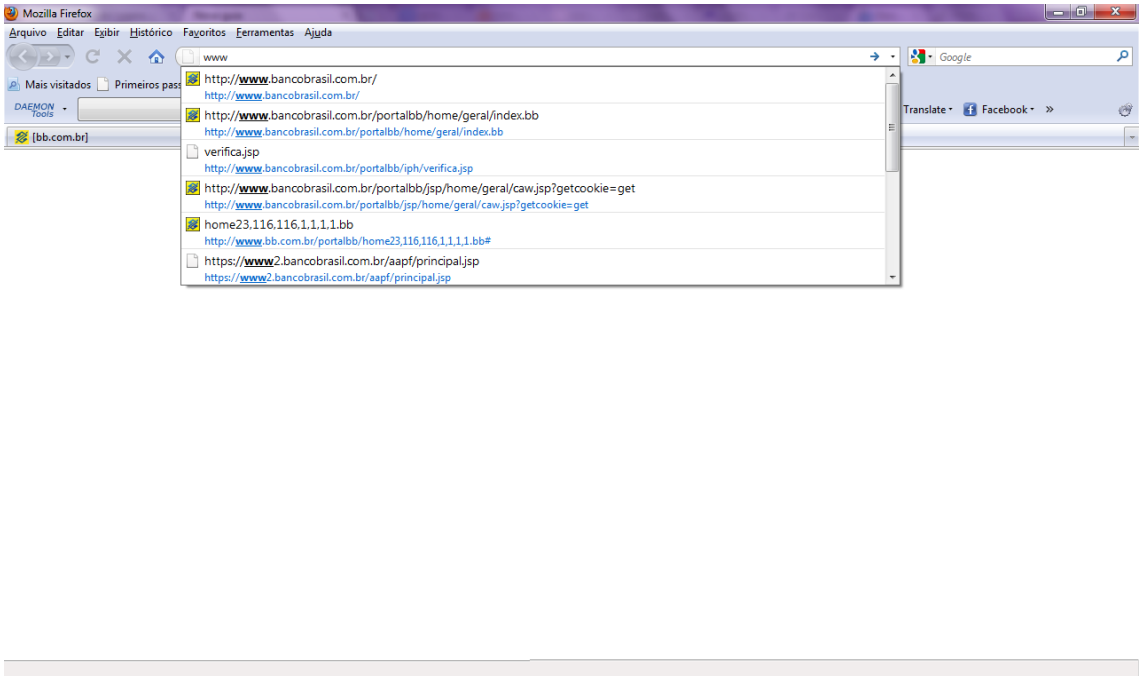


Figura 3.18 Exemplo de uso do padrão Continuous Filter no browser Mozilla.

Com o crescimento de aplicações na Internet, e a partir dos conceitos criados por Alexander, Montero (2002) propôs uma linguagem de padrões desenvolvida para interações Web. Tal linguagem se distingue em três níveis, sendo eles:

1. *Web Sites*: os padrões associados a esse nível têm características comuns e podem ser encontrados em muitos *sites* e são generalizados a partir de outro contexto diferente: (i) O usuário precisa saber onde ele(a) está (Welcome); (ii) o usuário precisa saber onde ele(a) pode ir (Indication) e (iii) o usuário quer visitar o site de uma maneira adequada (Polyglot, Ready, Similarity);

2. *Páginas Web*: são padrões relacionados ao design da página web, os quais consideram elementos e características enquanto se idealiza um site. Quando uma estrutura hierárquica é necessária (*Homepage*), em algumas ocasiões o usuário precisa fornecer informações e em seguida deve preencher um formulário (*Form*) e, sempre que quiser, tem o controle (*Busy, Second Chance*) para visitar o site no seu próprio ritmo (*Polite, Danger*);

3. *Ornamentais:* os padrões desse nível apresentam características de decoração de um site. Essas características proporcionam melhorias na usabilidade do site. Estão relacionadas ao uso de cores, tamanho de fontes e segurança (*Secret*), promovem localização, referência (*Location, Contact us*) e informação (Subscription, Recognize, Novelty).

Em 2008, Welie criou um *website* com uma coleção de mais de 130 padrões utilizados para o design da interação. A lista de padrões inclui três grandes categorias e suas subcategorias:

1. Necessidades do usuário
    - Navegando em torno de;
    - Busca;
    - Compras;
    - Interações básicas;
    - Lidando com os dados;
    - Fazendo escolhas;

- Fornecendo dados de entrada;
- Diversos;
- Personalizados;

2. Necessidades da aplicação;
    - Chamando a atenção;
    - Comentários;
    - Simplificando a interação.

3. Contexto do projeto
    - Tipos de sites;
    - Experiências;
    - Tipos de páginas;

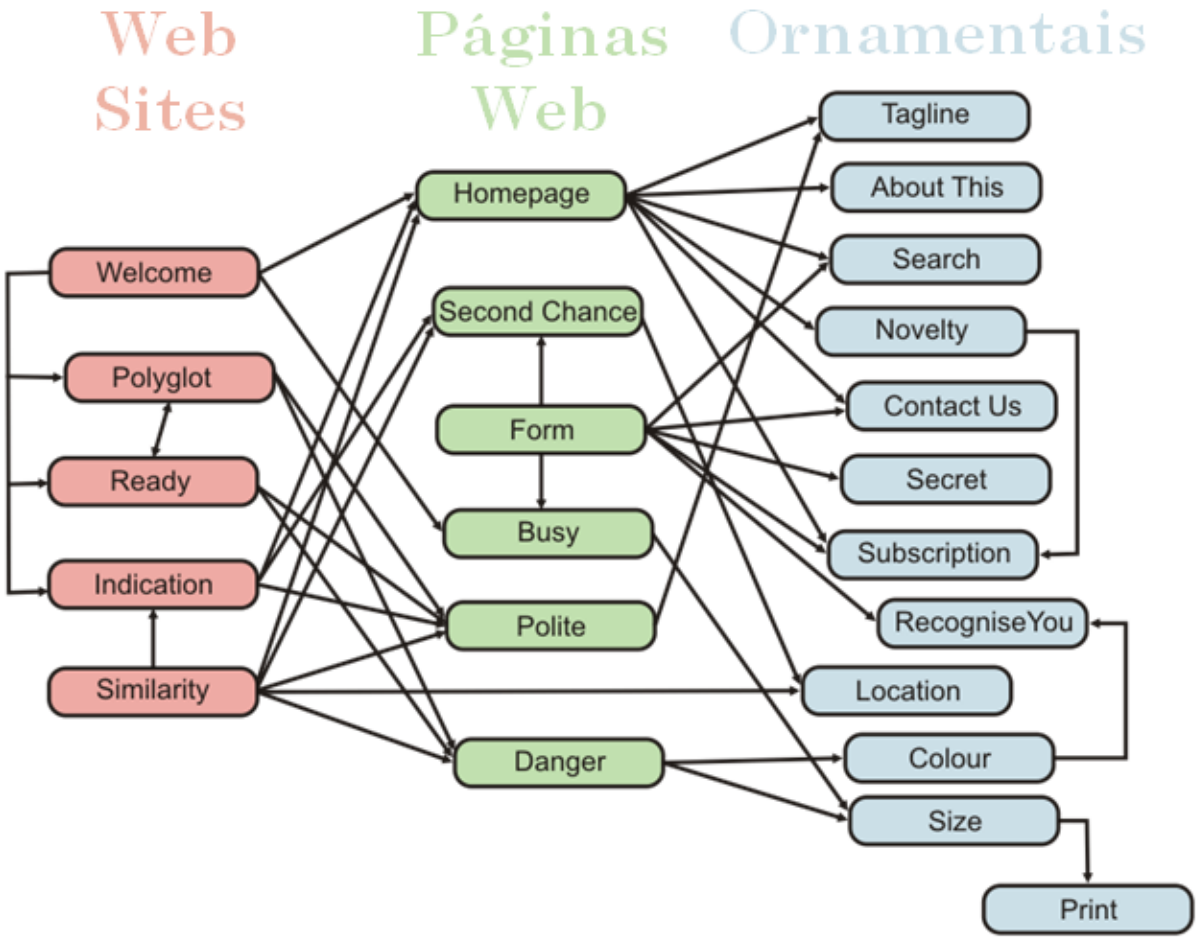


Figura 3.19 Padrões e Categorias propostas por Montero (2002)

O objetivo do site é ser uma referência ou, como o próprio autor denomina, um *"toolkit"*, no qual qualquer usuário pode utilizar o conhecimento de outros designers de interface. Qualquer pessoa pode ter acesso aos padrões e fornecer sugestões de melhorias e até mesmo novos padrões para serem adicionados à lista.

3.5 Padrões de Projeto Multimodais

As primeiras pesquisas na área de multimodalidade já permeiam mais de 20 anos. Para Godet-Bar *et al*. (2007), os projetos dessas interfaces ainda não são muito bem formalizados ou incluídos em uma ferramenta que possa auxiliar um não-especialista a projetar uma interface multimodal. Sendo assim, a solução proposta por ele é a utilização de padrões específicos para a multimodalidade, os quais possam, de maneira eficiente, captar experiências de problemas recorrentes e estruturá-los em uma solução geral, que será utilizada dependendo do contexto da interface.

Andreas Ratzka e Christian Wolff (2006, 2006, 2007b, 2008) colaboram com quinze padrões multimodais, sendo eles subdivididos em 4 categorias: (i) interação multimodal robusta, com quatro padrões; (ii) entrada rápida, com cinco padrões; (iii) flexibilidade, com três padrões e (iv) padrões abstratos, com três padrões.

3.5.1 Padrões para a Interação Multimodal Robusta

1. *Saída Redundante*: canais de comunicação podem sofrer distorções imprevistas devido às condições de má iluminação, ruído de fundo, problemas técnicos, deficiências, distúrbios de percepção, etc. A solução é cobrir vários canais de saída a fim de fazer uso da redundância. Informações de saída devem ser visuais e acústicas e possivelmente táteis para aumentar a probabilidade de percepção e de compreensão do usuário;

2. *Entrada Redundante*: canais de comunicação podem ser distorcidos de forma imprevista devido às condições de má iluminação, ruído de fundo, problemas técnicos ou alguma deficiência do usuário como, fala, coordenação motora ou distúrbio de percepção. A solução é combinar vários canais de interação a fim de usar a redundância. Entradas provenientes de vários canais (visual: leitura labial; auditivo: sinais de fala) devem ser interpretadas em conjunto a fim de reduzir a responsabilidade dos erros. Mesmo que vários canais sejam distorcidos, raramente essa distorção afeta a mesma parte da informação. Mecanismos de fusão permitem a reconstrução de pelo menos uma parte da informação;

3. *Seleção N-Melhor*: reconhecimento de fala é um processo estatístico. O reconhecimento de frases obtém resultados a partir de um conjunto de hipóteses de reconhecimento. É frequente o caso em que a frase de entrada original esteja incluída em um conjunto n-melhor, mas não coincida com a melhor estimativa do sistema. A solução é fornecer ao usuário meios de selecionar o resultado correto em um conjunto de hipóteses de reconhecimento através de apontamento ou teclas. A fim de satisfazer os casos em que os itens desejados não são encontrados na lista n-melhor, é preciso oferecer opções explícitas para uma nova entrada do usuário;

4. *Ortografia Baseada em Redução de Hipótese*: usado em sistemas onde a entrada através da fala não sofre restrições. Os erros são dependentes do acaso (palavras com sons similares). Grandes léxicos de reconhecimento são propensos a erros, especialmente quando muitas palavras apresentam sons similares. A solução é deixar o usuário fornecer as primeiras letras (por digitação) e reduzir, a partir disso, os itens selecionáveis. Como consequência do uso deste padrão tem-se (i) uma redução do conjunto de opções; (ii) a entrada rápida de dados; (iii) a necessidade de navegação e (iv) a simplificação de reconhecimento de fala em plataformas com poucos recursos.

3.5.2 Padrões para a Entrada Rápida Multimodal

1. *Interação de Voz Baseada em Atalho*: o usuário deve selecionar itens em um grande conjunto. Considera-se que a ação de seleção em um menu ou em uma lista, ou ainda, o número de opções pode ser muito grande e a tela pode ser muito pequena para exibir tudo de uma vez. O problema é que o estilo de interação não permite ao usuário selecionar o item desejado rapidamente. Uma solução consiste em selecionar objetos via fala, onde a interação pode ser acelerada significativamente. Isso é especialmente verdadeiro para usuários avançados, aos quais comandos e nomes são bem conhecidos. O designer deve incluir nomes idênticos às opções do menu ou aos itens da lista no vocabulário de reconhecimento de fala, a fim de possibilitar o aprendizado dos atalhos de interação. Ele precisa verificar quais sinônimos devem ser incluídos. Como consequência tem-se: (i) a organização da tela, (ii) a digitação reduzida e (iii) ausência de navegação nem de mapeamento de menus.

2. *Paleta com Habilitação de Voz*: também conhecido como “Fala como terceira mão” e “Fala avançada no espaço de ação”, oferece manipulação direta e permite que o usuário edite objetos visualmente. Para manipular esses objetos o usuário deve selecionar uma ferramenta, ou seja, deve deixar a área de manipulação através do mouse, selecionar um item no menu e, em seguida, voltar para a área de manipulação a fim de terminar a ação. Isso pode ser muito irritante, especialmente em aplicações com desenho. A pergunta é “como habilitar a seleção de ferramentas da paleta sem que o usuário precise usar o mouse entre a tela e a paleta ou a mão entre o mouse e o teclado?”. A solução consiste em permitir que o usuário selecione as ferramentas através da fala. Cada ferramenta da paleta deve ter um nome significativo e que seja intuitivo para o usuário, permitindo uma aprendizagem contínua;

3. *Formulário com Habilitação de Fala*: o usuário tem uma estrutura de dados de entrada que pode ser mapeada para algum tipo de formulário, composto por um conjunto de campos. Dispositivos como PDAs não fornecem um teclado para a entrada de *strings* de maneira rápida e fácil. Em outras situações, o dispositivo pode apoiar a entrada com o teclado, porém o usuário só tem uma mão livre para interagir com o sistema. A solução consiste, sempre que possível, em determinar valores aceitáveis para o campo do formulário. Apoia-se a seleção de valores em listas através de comandos de voz. Permite-se que o usuário selecione o campo do formulário via apontamento e o valor de entrada via modalidade de voz. A complexidade da entrada de voz pode ser reduzida quando o vocabulário é direcionado ao campo, ativando somente as necessidades do momento;

4. *Gesto Avançado e Fala Natural*: Algumas aplicações requerem a entrada de comandos compostos, constituídos por diversos parâmetros. Considere copiar um objeto. É preciso selecionar o objeto, copiá-lo e enviá-lo ao local de destino. O problema é “como permitir que o usuário entre rapidamente com comandos que têm vários parâmetros compostos?”. Uma solução consiste em permitir que o usuário interaja através da fala e forneça gestos de apontamento, simultaneamente, para especificar a ação. Como no exemplo do “*Put That There*”, copie esse objeto (aponta para o objeto desejado) para aquele lugar (aponta para o local desejado);

5. *Gestos Sensíveis a Localização*: alguns comandos (como excluir) requerem parâmetros adicionais. Não existe teclado e o usuário tem apenas uma mão livre para interagir. O problema está em como permitir que o usuário entre com dados, de forma rápida e fácil. A solução consiste em permitir que o usuário interaja com o sistema como se estivesse escrevendo em um papel: usando símbolos como caneta, sobre um objeto, desenhando setas e assim por diante. As posições iniciais e finais da caneta podem ser interpretadas como parâmetros de posicionamento. Gestos de caneta têm que ser cuidadosamente planejados.

3.5.3 Padrões para a Flexibilidade da Interação Multimodal

1. *Múltiplos Tipos de Entrada*: fatores de contexto não são sempre previsíveis. Isso vale especialmente para a interação móvel em ambientes voláteis ou para quiosques de informação pública, utilizados por vários grupos de pessoas. O problema é “como as modalidades de entrada podem se adaptar ao contexto de uso, sem sobrecarregar o usuário com as tarefas de configuração?”. A solução consiste em permitir que cada função do sistema seja operada através de várias modalidades de interação alternativas e que o usuário escolha o canal de interação preferido, seja ele através da fala, da digitação ou de apontamento. Todas as modalidades devem ser ativas, ou seja, o usuário não deve ter que configurar uma mudança de modalidade no sistema;

2. *Configuração Global do Canal*: dispositivos interativos que oferecem várias opções de canais de interação complementares, como entrada e saída de áudio, digitação e manipulação de gráficos, têm que se adaptar ao contexto do usuário. O problema é “como a entrada e a saída da interação podem ser adaptadas ao contexto do usuário sem sobrecarregá-lo com configurações da tarefa?”. A solução consiste em projetar perfis de interação com vários canais de configuração de entrada e de saída adaptados a cada contexto de uso. Isso deve permitir que o usuário selecione o perfil de interação através de um único passo e mostrar, por exemplo, botões e ícones explicativos sempre no topo.

3. *Adaptação do Contexto*: exemplos desse padrão podem ser encontrados em dispositivos que suportam diferentes interações alternativas e complementares. Por exemplo, canais de interação de entrada e de saída de áudio, digitação e manipulação de gráficos, etc. O problema é “como a interação pode ser adaptada à situação atual, ao ambiente e ao usuário, sem que ele precise executar etapas adicionais de interação?”. A solução

consiste em permitir que o sistema analise as informações de contexto, e também garantir que as configurações do sistema sejam feitas de forma autônoma. Uma fonte de informação que pode ser usada é o histórico da interação “se o usuário interagiu por fala, não ficou claro para onde ele estava olhando”.

3.5.4 Padrões Abstratos para a Interação Multimodal

1. *Entrada Adequada de Conteúdo*: sistemas interativos permitem a entrada de diferentes tipos de dados, tais como: imagens, texto, sons, vídeos, etc. O problema é que não existe uma maneira uniforme de entrada e de apresentação dos diversos formatos de dados. A solução consiste em analisar as tarefas, o fluxo de trabalho e os dados trocados a fim de escolher as modalidades de interação mais adequadas, como apontamento e fala. A consequência é que o apontamento não requer memorização e a fala não exige que todos os objetos sejam exibidos simultaneamente;
2. *Modalidade de Uso Adequado ao Contexto*: há pessoas diferentes, utilizando tipos diferentes de dispositivos em diferentes ambientes e situações. O problema é que esses diferentes contextos usados para diferentes necessidades podem não ser satisfeitos em uma única concepção. A solução consiste em analisar a tarefa e o fluxo de trabalho, separar grupos de usuários-alvo e os cenários de interação, a fim de determinar as modalidades de interação adequadas. Além disso, deve-ser oferecer mais de uma alternativa de modalidade de interação com o usuário para que ele possa escolher o mais adequado, de acordo com o seu contexto;

3. *Largura Máxima de Banda de Comunicação*: a tarefa que um usuário esteja executando em um dispositivo pode ser muito complexa e requerer muita informação de entrada e de saída. O problema é que dispositivos adicionais, sejam eles reais ou virtuais, exigem a atenção do usuário. Então, a solução consiste em analisar a tarefa, os dados, os grupos-alvo de usuários e os cenários de uso e determinar as modalidades de

interação adequadas, como sugerido pelos padrões de Entrada Adequada de Conteúdo e de Modalidade de Uso Adequado ao Contexto. Paralelamente, é preciso distribuir subtarefas a diferentes modalidades de interação, como: apontando com fala e visualização gráfica com texto falado. Isso muda a atenção de maneira que minimiza a interação e a deixa mais eficiente.

3.6 Considerações Finais

Neste capítulo, foi realizado um levantamento sobre Padrões de Projetos, sua história, importância, como estruturá-los, alguns tipos de padrões e exemplos. Foram discutidos padrões para o desenvolvimento e para a geração de código, para a interação com o usuário e alguns voltados para aplicações multimodais.

É possível notar, com exemplos práticos e do cotidiano, a necessidade da utilização de padrões para a construção de sistemas com qualidade. Apesar de tal importância, poucos alunos saem da graduação com um bom conhecimento sobre padrões. Um dos objetivos do *framework* GuiBM, que será definido no capítulo 6, é fazer uso dos conceitos de padrões para auxiliar desenvolvedores a criar interfaces mais ricas e intuitivas.

Alguns padrões apresentados neste capítulo, como os de interação e multimodais, foram inseridos dentro do *framework* para fornecer dicas aos projetistas. Os trabalhos de Andreas Ratzka e Christian Wolff foram de suma importância para o desenvolvimento do GuiBM, no entanto nem todos os padrões propostos por eles foram utilizados devido a limitações do *framework*.

Através dos estudos e padrões propostos por Ratzka e Wolff a autora pode iniciar o desenvolvimento do GuiBM com um conhecimento maior sobre melhores práticas para criação de interfaces multimodais.

Capítulo 4

FRAMEWORK

4.1 Considerações Iniciais

Não existe uma única definição para *framework*; sendo assim, vários autores complementam a definição desse termo. Por exemplo: "*Um framework é o esqueleto de uma aplicação que pode ser melhorado por um desenvolvedor da aplicação*" (Fayad *et al.*, 1999) ou ainda "*Frameworks fornecem um mecanismo para obter a reutilização de código e são bem apropriados para domínios onde várias aplicações similares são construídas várias vezes, partindo-se apenas de ideias relacionadas*" (Johnson & Foot, 1988).

Existe uma grande similaridade entre padrões de projeto, descritos no capítulo anterior, e *frameworks*. Ambos utilizam técnicas de reuso, mas existem diferenças importantes e de fácil percepção, como: (i) padrões de projeto são mais abstratos, ao passo que *frameworks* focam em projetos concretos, codificação, algoritmos e linguagem de programação e (ii) padrões de projeto possuem elementos menores que um único *framework*. É possível utilizar vários padrões de projeto dentro de um *framework* (Inácio, 2007), porém o contrário não é verdadeiro.

A utilização de padrões de projetos e de *frameworks* pode contribuir de forma significativa para o desenvolvimento de soluções eficazes para domínios específicos, à medida que suportam a utilização de abstrações de mais alto nível para criar componentes (Szyperski, 2000) e estruturas flexíveis e reusáveis.

4.2 Framework

Os *frameworks* se concentram em resolver problemas que surgem a partir da construção de aplicações com requisitos em comum. A ideia principal de um *framework* é separar as partes que são modificadas das que permanecem iguais (Eckel, 2003). Partindo-se desse princípio, pode-se separar o *framework* em partes que ficarão estáticas e desempenharão as suas funções sem modificações e em outras que poderão/deverão ser modificadas (chamadas *hot-spots*) a fim de prover uma funcionalidade especial. A parte imutável é comum ao domínio ao qual o *framework* serve e não sofre alterações quando é aplicada a sistemas.

Já a parte flexível (*hot-spots*) não representa uma solução completa, pois permite que o desenvolvedor a modifique para adequá-la à sua aplicação. Numa abordagem orientada a objetos, as classes abstratas possuem as soluções comuns. Ao se derivar essas classes, é possível aproveitar os métodos já definidos e introduzir novos, afim de prover o funcionamento específico desejado e de permitir um alto grau de reutilização de código.

Muitos autores focam seus estudos em *frameworks,* como Roberts & Johnson (1998), Bosh *et al.* (1999), Fayad *et al.* (1999), Pree (1999) e Schmidt (2002), com o intuito de descobrir técnicas mais eficientes não somente para construí-los, mas também para tornar mais fácil que desenvolvedores não familiarizados as utilizem.

Fayad *et al.* (1999) e Schmidt (2002) identificam três principais classes de *frameworks*: (i) de infra-estrutura, compatíveis com o desenvolvimento de interfaces GUI e compiladores, por exemplo; (ii) de integração com o *middleware*, que suportam a comunicação de componentes e a troca de informações, como o CORBA e o Java Beans e (iii) de aplicações corporativas, que se ocupam de domínios específicos de aplicações, tais como sistemas administrativos e financeiros por exemplo. De acordo com Sommerville (2001), embora no momento existam soluções adequadas para os dois primeiros tipos, os *frameworks* de aplicações corporativas ainda estão começando a surgir e estudos têm sido conduzidos a fim de saber quais as estruturas mais eficazes para resolver problemas de domínios específicos.

Uma das abordagens mais eficazes para o reuso de *frameworks* em aplicações corporativas baseia-se na noção de família de aplicações. Uma família de aplicações, segundo Sommerville (2001), é um conjunto de aplicações relacionadas

que têm uma arquitetura de domínio específico em comum. O núcleo em comum é reutilizado a cada vez que uma nova aplicação precisa ser desenvolvida. O novo desenvolvimento pode envolver a necessidade de escrever componentes adicionais ou de adaptar outros componentes a fim de atender aos novos requisitos.

4.3 Frameworks Multimodais

Com o conceito de *frameworks* em mente e a fim de simplificar a construção de interfaces multimodais, surgiram, nas últimas décadas, os *frameworks* destinados à área da multimodalidade. Pesquisadores como Goubran e Wood (1996) implementam um *framework* para uma interface multimodal capaz de interpretar as modalidades de voz e de gestos com caneta no contexto de aplicações de agenda. O *framework* possui um módulo chamado “interpretador multimodal” que reconhece cada modalidade separadamente e processa as informações em uma agenda eletrônica chamada Jeanie.

Krahnstoever et al. (2002), propõe um *framework* para a criação de interfaces com voz e gesto chamado iMap. A saída dos dados ocorre através de um display grande e é possível ter uma interação multiusuário.

Bourguet (2002) propõe um kit de ferramentas com dois componentes, chamados IMBuilder e Mengine, para criar e testar projetos com interface multimodal. O primeiro usa um modelo especificado através de máquinas de estados finitos e uma ferramenta de interface gráfica simples (IMBuilder). O segundo testa o modelo gerado pelo IMBuilder em uma estrutura multimodal que automaticamente reconhece entradas com voz e com gestos (Mengine). Os desenvolvedores podem, então, criar aplicações multimodais sem se preocupar com a integração das modalidades.

Flippo *et al.* (2003) propõe a implementação rápida de aplicações multimodais. Seu foco é em interfaces multimodais para apenas um usuário. O sistema utiliza as seguintes modalidades de entrada: o mouse, o olhar do usuário e interações por voz. O estudo de caso que prova a viabilidade desse *framework* foi realizado através da construção de um aplicativo chamado *Flatscape,* que consiste em um mapa colaborativo. Nessa aplicação, é possível planejar operações militares

pela movimentação de ícones que representam unidades militares em um mapa e o rastreamento de veículos em movimentos robóticos.

Bouchet *et al*. (2004) propõem um *framework* chamado ICARE, que é baseado nas propriedades CARE (complementaridade, atribuição, redundância e equivalência). A abordagem do ICARE é baseada em dois componentes. O primeiro inclui componentes elementares do ICARE e componentes de dispositivos e de linguagens de interação, o que permite desenvolver modalidades puras. O segundo é chamado de “Componente de Composição” que define o uso da combinação das modalidades.

Feuerstack & Pizzolato (2011) propõem um ambiente de execução baseado em modelo chamado de *framework* MINT, o qual descreve a interação multimodal por interatores e mapeamentos multimodais. Os interatores são modelados usando máquinas de estados e descrevem elementos da interface do usuário para diversas modalidades. O mapeamento combina esses interatores com o dispositivo e define a relação multimodal.

Outros exemplos de *frameworks* para o desenvolvimento de interfaces multimodais são: FAME (Duarte & Carriço, 2006), JMIS Framework (Inácio, 2007), OpenInterface Framework (Serrano *et al.*, 2008) e o MMWA-ae (Talarico, 2011), descritos mais detalhadamente nas próximas seções.

4.3.1 FAME

FAME (Duarte & Carriço, 2006) é um *framework* que tem por objetivo orientar projetistas na implementação de aplicações multimodais adaptativas. As vantagens das interfaces multimodais podem ser aumentadas através da exploração das capacidades adaptativas, importante quando se lida com usuários portadores de deficiências diferentes. As interfaces adaptativas também podem acomodar mudanças nas condições do ambiente através da seleção da modalidade mais adequada.

Porém, o FAME não possui uma ferramenta de desenvolvimento automático de aplicações; ele fornece suporte ao processo de desenvolvimento através de uma base conceitual dos diferentes aspectos de um sistema multimodal adaptativo. Esse

framework propõe uma arquitetura para a adaptação de aplicações multimodais, utilizando um conjunto de modelos matemáticos que descrevem o comportamento do usuário em relação à aplicação e ao meio ambiente. Para auxiliar o desenvolvimento das regras de adaptação, o *framework* FAME utiliza uma matriz de comportamento, na qual são representadas as interações do usuário com um componente adaptável. Sua arquitetura é ilustrada na Figura 4.1.

O *framework* FAME utiliza as informações armazenadas em vários modelos para controlar as saídas multimodais da interface, as possibilidades de interação com o usuário e o modo como essas informações podem ser interpretadas pela plataforma. Na Figura 4.1, duas camadas da arquitetura são ilustradas: a primeira, no módulo de adaptação, é responsável pelos modelos e pelas ações geradas pelo sistema; a segunda corresponde à aplicação multimodal e é responsável pela fusão das entradas multimodais fornecidas pelo usuário e pelos eventos gerados para o núcleo de adaptação.

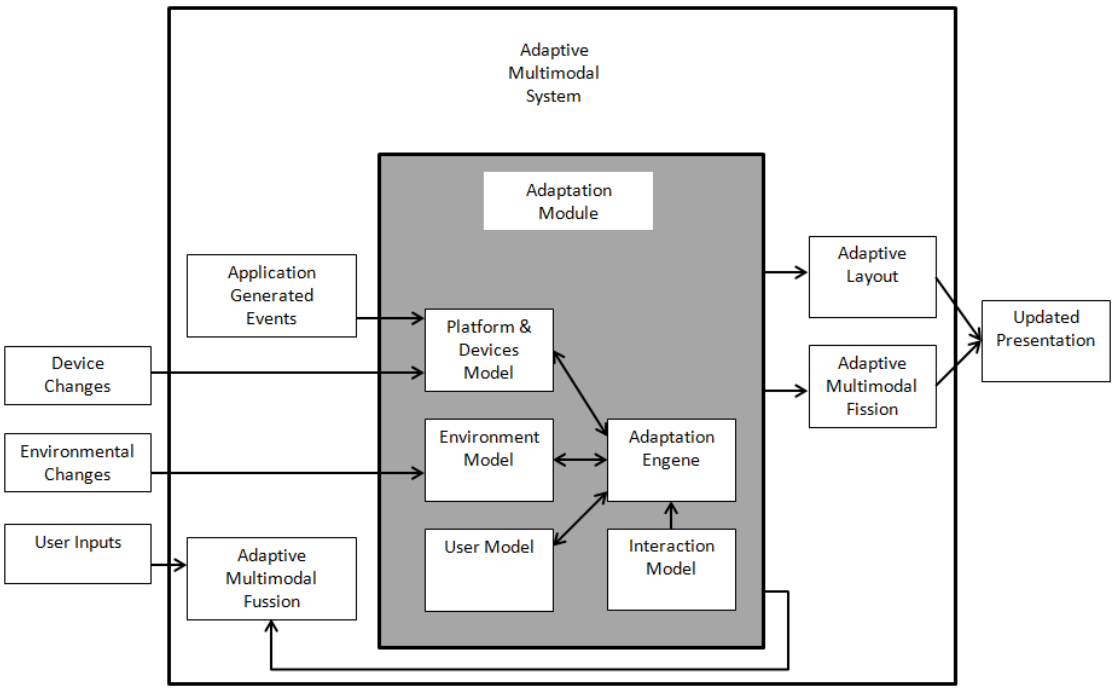


Figura 4.1 Arquitetura do Framework FAME. Fonte: Duarte & Carriço (2006).

A adaptação é baseada em três classes de entrada diferentes: (i) as ações do usuário, realizadas em qualquer um dos dispositivos de entrada encontrados na aplicação; (ii) os eventos gerados e a mudança no dispositivo, classe responsável por modificar o estado de um componente com o qual o usuário interagiu; (iii) as

mudanças no ambiente de interação, realizadas através de sensores que captam a interação do usuário com a aplicação.

Os modelos utilizados pelo *framework* são: (i) modelo do usuário, o qual armazena as preferências do usuário podendo incluir atributos físicos (como, por exemplo, o seu nível de deficiência visual) e comportamentos; (ii) modelo de plataforma e de dispositivo, o qual descreve as características de execução da plataforma e dos dispositivos associados a ela; (iii) modelo do ambiente, o qual descreve as características que podem afetar o ambiente (como, por exemplo, ruído de fundo que influência na entrada de voz) e (iv) modelo de interação, o que descreve os componentes existentes na interação. Cada componente tem um conjunto de modelos disponíveis que abrange os dispositivos, os grupos de usuários e o ambiente.

4.3.2 Framework JMIS

O JMIS (*Java Multimodal Ink and Speech Framework*) é um *framework* utilizado para o desenvolvimento de aplicações multimodais em computação ubíqua. Ele faz uso da infraestrutura das APIs *Java Ink* e *Java Speech* e tem como objetivo a implementação rápida de aplicações multimodais que integram as modalidades de voz e de escrita. Sua arquitetura é formada por quatro módulos: (i) o Componente Multimodal; (ii) o Módulo de Escrita, que utiliza a API *Java Ink*; (iii) o Módulo de Voz, que utiliza a API *Java Speech* e (iv) o Integrador Multimodal.

Os módulos do *framework* JMIS são divididos em três camadas, representadas na Figura 4.2: (i) o nível de Aplicação, que possui o Componente Multimodal e é responsável por instanciar os módulos de escrita e de voz, capturando as entradas e manipulando as respostas do módulo inferior; (ii) o nível de Fusão Multimodal, composto pelo Integrador Multimodal, o qual é responsável por: (a) unificar os eventos gerados pelos módulos de escrita e de voz, (b) interpretar as informações e (c) enviar os resultados para o nível de Aplicação; (iii) o nível de Processamento Unimodal, que possui os módulos de Escrita e de Voz e é responsável pelo processamento individual das entradas, sejam elas de escrita ou de voz.

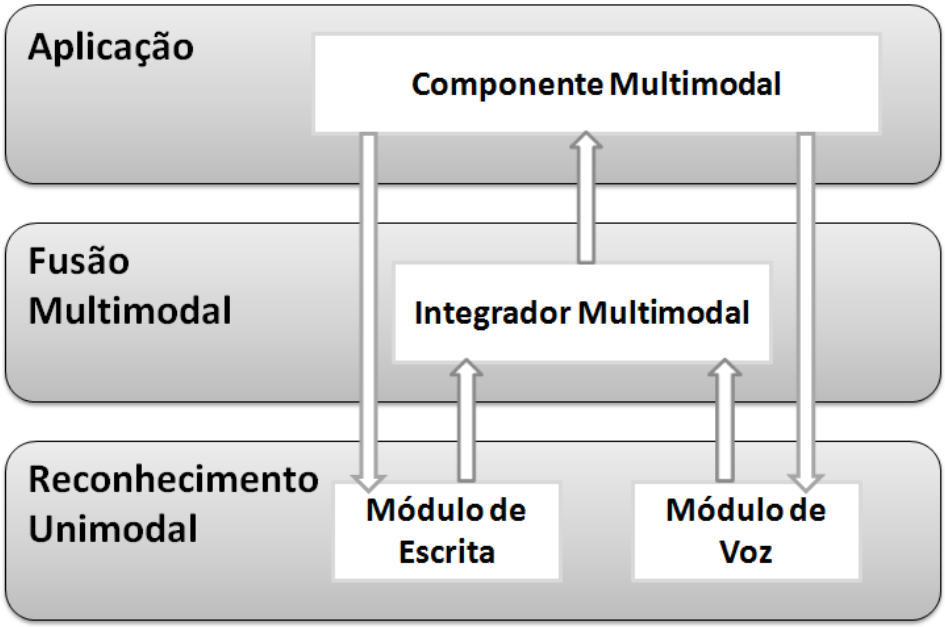


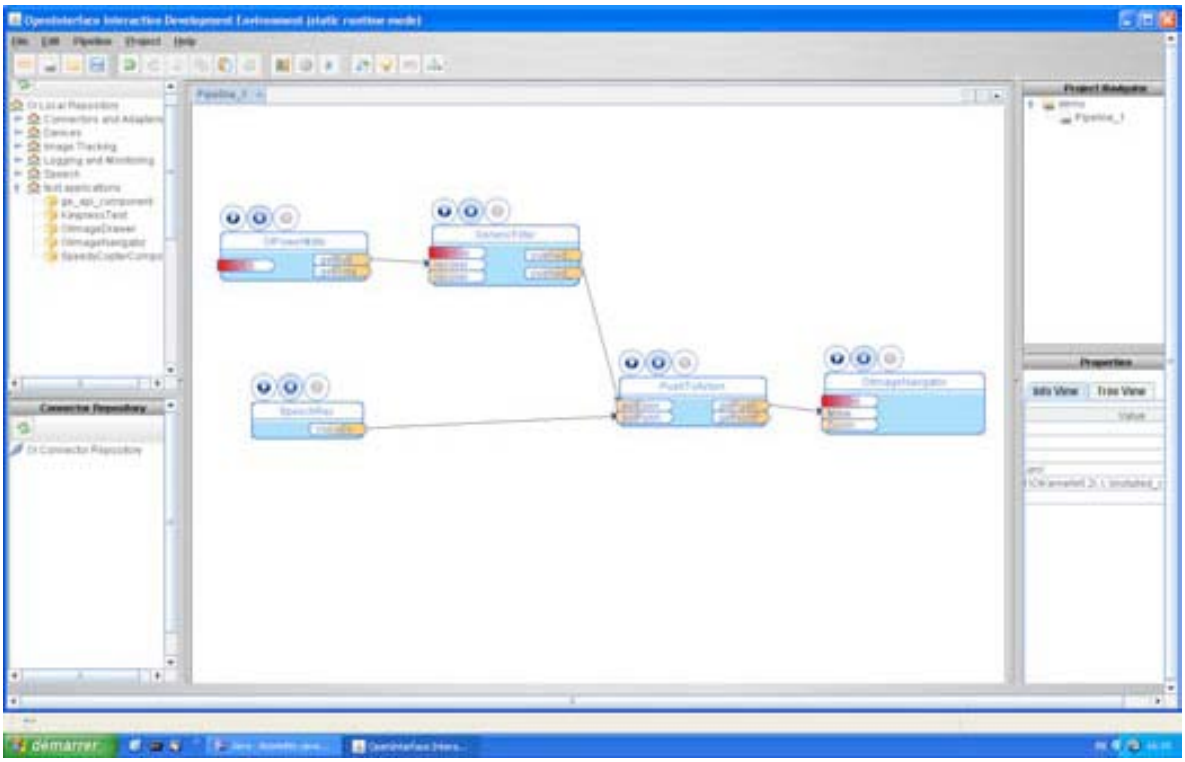
Figura 4.2 Arquitetura Framework JMIS. Fonte: Inácio (2007).

4.3.3 OpenInterface Framework

O OpenInterface Framework, ou simplesmente OI, (Serrano *et. al.*, 2008) tem como objetivo fornecer as melhores formas de desenvolver sistemas que utilizam interações multimodais. A estrutura do *framework* OI é composta por dois módulos o Oi Kernel e o Oide.

O **OI Kernel** é uma plataforma baseada em componentes heterogêneos. Ele inclui uma ferramenta para criação de novos componentes com código existente sem modificar o código original. O Oi kernel gerencia a criação e a conexão entre os componentes de forma dinâmica interpretando as descrições OI da aplicação.

O **Oide** (*Ambiente de Desenvolvimento de Interação OpenInterface*) é uma ferramenta gráfica construída sobre a plataforma OI Kernel. Ele permite uma manipulação direta para montagem dos componentes. A Figura 4.3 ilustra o Oide.



4.3 Editor Gráfico Oide. Fonte: Serrano *et al.* (2008).

O OI Kernel e o Oide permitem a criação dinâmica dos componentes, onde cada componente pode ser idealizado individualmente. Cada componente tem um botão que permite que o projetista o inicie.

O modelo conceitual do OI é baseado em componentes genéricos e adaptados. Os componentes genéricos definem operações genéricas reutilizáveis já os componentes adaptados são aplicados de forma ad-hoc para uma técnica de interação especifica ou uma aplicação de interação.

O *framework* OpenInterface utiliza alguns dispositivos para a elaboração de interfaces multimodais como: (i) o Shake, que é um pacote com uma variedade de sensores; (ii) o Track IR, que é uma câmera e (iii) a Interface-Z, que contém vários sensores inovadores tais como um sensor de pressão atmosférica.

As interações podem ser através da voz ou por gestos 3D com as mãos. Duas aplicações foram desenvolvidas utilizando o OI, sendo elas: (i) um Mapa com Navegação Multimodal, onde as modalidades de interação utilizadas podiam ser de voz e gestos ou pressão e gesto utilizando a ferramenta Interface-Z e (ii) um Game para Telefone Móvel, onde as modalidades de interação utilizadas são por comando

de voz, comando de gestos 3D através da ferramenta Shake e comandos de gestos utilizando uma câmera.

4.3.4 MMWA-ae

O MMWA-ae (*Multimodal Web Approach Authoring Environment*) é um ambiente de autoria para o projeto de interfaces multimodais desenvolvido para automatizar as atividades da MMWA (Talarico, 2011). A abordagem para o desenvolvimento de interfaces multimodais Web (MMWA) tem o objetivo de proporcionar à equipe de projetistas, desenvolvedores e testadores de interfaces multimodais uma estrutura prática, composta por atividades e técnicas, utilizadas em suas tarefas de forma a otimizá-las. Tal abordagem segue um modelo em espiral, composto de cinco atividades e de um conjunto de técnicas. Suas atividades são:

1. **Modelo Comportamental Inicia (MCI) –** engloba a definição de cenários, a identificação de restrições, as informações sobre o ambiente em que as tarefas multimodais serão executadas e as informações sobre a experiência do usuário.
2. **Identificação de Tarefas (IT) –** consiste em identificar as tarefas que serão executadas pelos usuários na interface multimodal.
3. **Representação de Tarefas (RT)** – o projetista cria representações abstratas e concretas para as tarefas previamente identificadas.
4. **Análise de Soluções (AS)** – consiste na análise das soluções obtidas nas atividades anteriores.
5. **Avaliação de Usabilidade (AU)** – realizada com o uso do protótipo gerado a partir das atividades anteriores.

O MMWA-ae é um sistema Web projetado na arquitetura cliente-servidor. Sua arquitetura foi planejada com o intuito de facilitar a separação de dois tipos de interfaces de usuário: (i) a **Interface de Projeto** pode ser acessada pelo projetista a partir de qualquer navegador web e seu objetivo é propiciar a criação de um projeto multimodal e trabalhar com as atividades da MMWA e (2) a **Interface de Teste** é disponibilizada aos projetistas para a execução de testes de usabilidade e tem por objetivo facilitar a execução de testes com usuários remotos. A arquitetura do MMWA-ae é representada na Figura 4.4, a qual ilustra as duas interfaces propostas e os módulos que se comunicam com as mesmas.

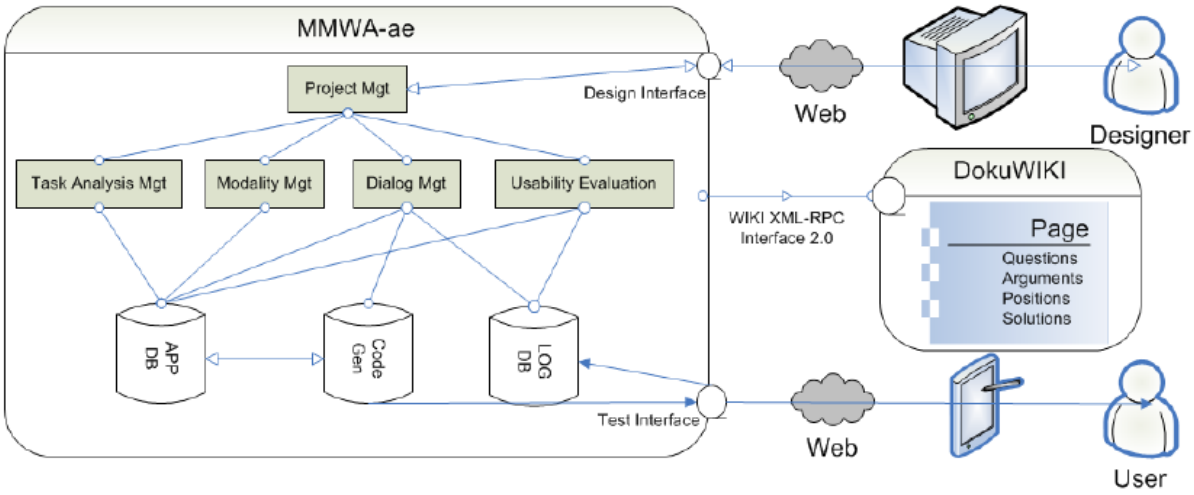


Figura 4.4 Arquitetura MMWA-ae. Referência Talarico (2011).

Os principais módulos que se comunicam com as duas interfaces descritas anteriormente são: (i) o **Módulo de Gerenciamento de Projetos**, utilizado pelo projetista para criar, remover e alterar um projeto no ambiente de autoria; (ii) o **Módulo Gerenciador de Análise de Tarefas**, que oferece suporte para que o projetista crie a sua interação multimodal de acordo com o que especificam as atividades de Identificação (IT) e de Representação de Tarefas (RT) da MMWA; (iii) o **Módulo Gerenciador de Modalidades**, responsável por selecionar e apresentar as características inerentes a cada modalidade na interface do MMWA-ae; (iv) o **Módulo Gerenciador de Diálogo**, responsável por armazenar informações relativas ao fluxo da aplicação e garantir que todos os estados sejam alcançados, sendo que não há estados órfãos e não há problemas de navegação e (v) o **Módulo de Gerenciamento de Avaliação de Usabilidade**, responsável por reunir toda a informação de diálogo do banco de dados da aplicação, gerar o código do aplicativo através do Gerador de Código, incluir as funções para capturar os *logs* de interação da Base de Dados de *log* e permitir a criação do protocolo de testes de usabilidade. A interface principal da área de trabalho de projeto é apresentada na Figura 4.5.

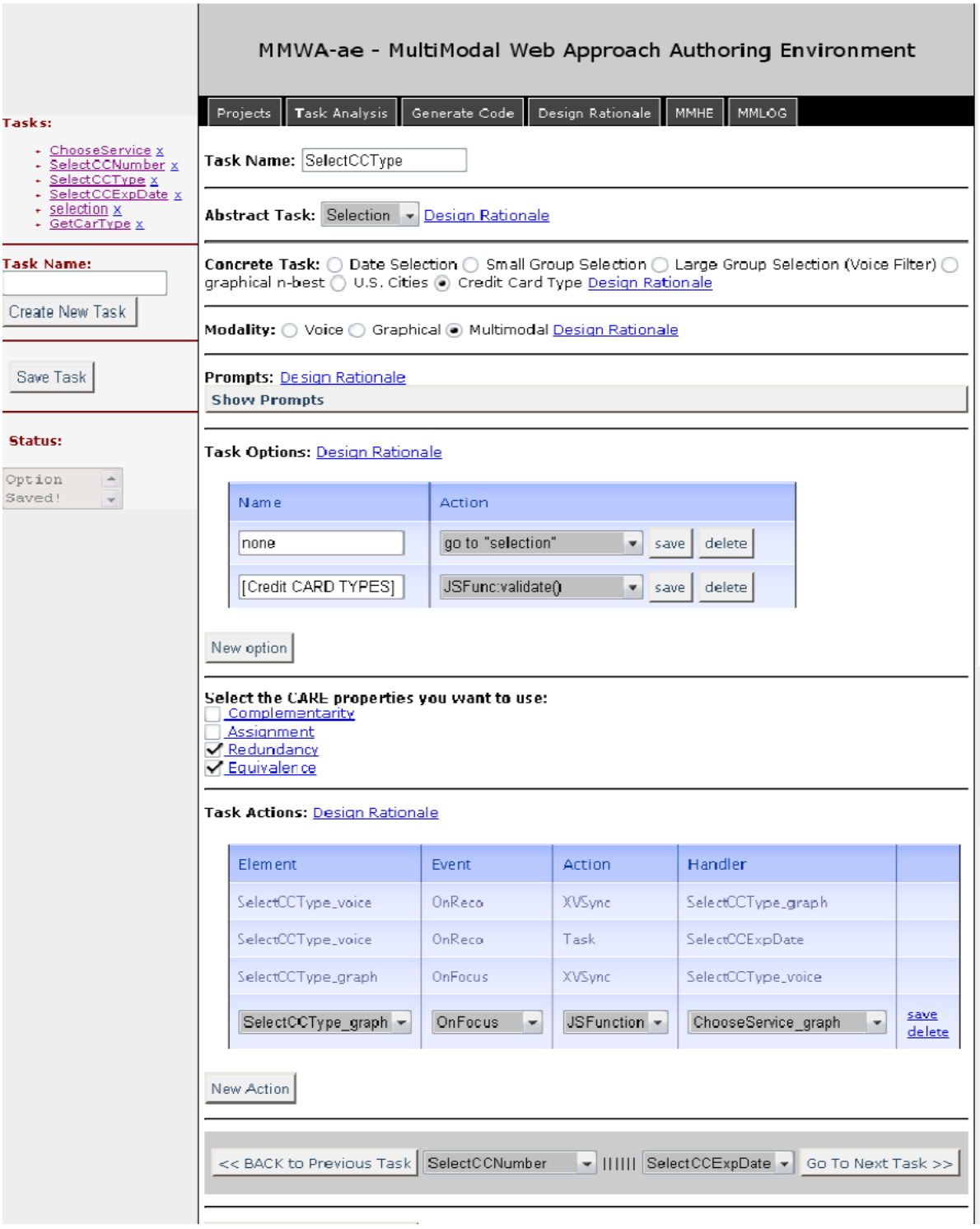


Figura 4.5 Interface de projeto do MMWA-ae. Referência Talarico (2011)

4.4 Considerações Finais

Este capítulo apresentou o conceito de *frameworks*, bem como um breve levantamento de *frameworks* multimodais. Percebe-se que vários pesquisadores têm a preocupação de auxiliar projetistas a desenvolver interfaces que utilizam diversas modalidades de uma forma mais rápida e seguindo alguns modelos. O conceito de *framework* auxilia a reutilização de códigos pré-existentes, porém o projetista precisa possuir um conhecimento prévio sobre o assunto, na grande maioria dos casos.

Diferente do *framework* FAME, que não possui uma ferramenta de desenvolvimento automático, o GuiBM oferece uma interface do tipo WYSIWYG, sendo intuitivo para programadores que já tenham familiaridade com IDEs de criação de interface.

O *framework* JMIS, da mesma forma que o GuiBM, utiliza a infraestrutura da API Java Speech para modalidade de voz. Porém os dois projetos se distinguem na utilização de outras modalidades, enquanto o primeiro faz uso da modalidade de escrita o segundo utiliza toque e reconhecimento de gestos.

Comparado ao *framework* OI o GuiBM não oferece suporte a reconhecimento de gestos em 3D. Os gestos reconhecidos pelo GuiBM são limitados e fazem uso de luvas de cores distintas. O GuiBM também não oferece suporte para desenvolvimento de interfaces Web como o MMWA-ae.

Porém, o GuiBM utiliza os conceitos de padrões de projeto como apoio, em forma de dicas que auxiliam o projetista a melhorar sua interface. O objetivo central é auxiliar designers recém-formados ou com pouco conhecimento em multimodalidade a desenvolver interfaces com reconhecimento de voz, toque e gestos.

Para que tais dicas fossem apresentadas em momentos oportunos, durante o desenvolvimento da interface, utilizou-se o conceito de agentes inteligentes que será explorado no capítulo seguinte.

Capítulo 5

AGENTES

5.1 Considerações Iniciais

O desejo em conseguir oferecer uma interação direta entre usuário e máquina há muito vem sendo pesquisado. O desafio dos projetos é baseado na pergunta chave “como oferecer conhecimento às máquinas? E como fazer com que elas interajam com os usuários como se fossem seres humanos?”. Uma solução seria a utilização de *Agentes Inteligentes*, que segundo Jennings & Wooldridge (2001) é um sistema de computador que está situado em algum ambiente e que é capaz de executar ações autônomas de forma flexível neste ambiente, a fim de satisfazer seus objetivos de projeto.

Oliver Selfridge (1958) é considerado um dos pais do termo *Agentes Inteligentes*, em seu trabalho intitulado *“Pandemonium: A Paradigm for Learning”*. Ele defende a tese de que máquinas podem reconhecer padrões de ações humanas através de uma coleção de pequenos componentes, que ele denominou como “demônios”. Outros pesquisadores também contribuíram para o crescimento da área, como Marvin Minsky (1986, 2006) e Pattie Maes (1994), dentre outros.

Um *agente inteligente* possui características básicas como: (i) autonomia, que é a capacidade de agir sem a intervenção de outros agentes; (ii) reatividade, que é a habilidade de reagir aos estímulos do ambiente; (iii) proatividade, que é a propriedade de agir guiado por objetivos, a partir de uma iniciativa própria e (iv) sociabilidade, que é a potencialidade de se comunicar com outros agentes no ambiente.

5.2 Definição de um Agente Inteligente

Para Russell e Norvig (2004) um **agente** é tudo que pode ser considerado capaz de perceber seu **ambiente** por meio de **sensores** e de agir sobre esse ambiente por intermédio de **atuadores**. Um agente de software é baseado necessariamente no agente humano, ou seja, para perceber um ambiente um agente humano utiliza os olhos (como sensores) e para agir sobre o ambiente ele pode utilizar os braços ou as pernas (como atuadores ou reagentes). A Figura 5.1 mostra essa associação.

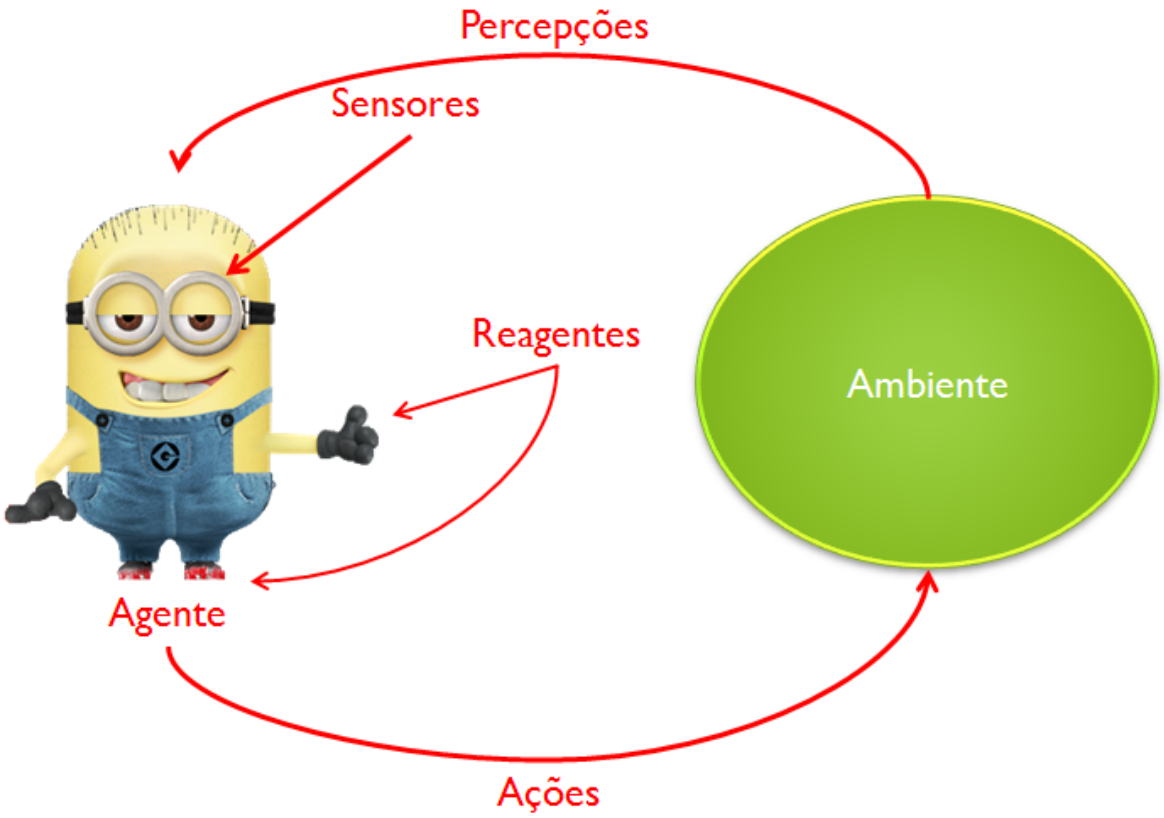


Figura 5.1 Representação de um agente e suas características. Baseado em Russel e Norvig (2004)

Um agente robótico pode fazer uso de dispositivos como câmeras e infravermelho que atuariam como sensores, e utilizar braços e pernas robóticas para reagir a estímulos desse ambiente. Já o agente de software pode receber conteúdos de arquivos, pacotes de redes e até mesmo entradas do teclado, processar essas informações e reagir oferecendo um feedback em uma tela (Russell & Norvig, 2004).

Segundo Jennings (1995) é possível identificar três classes diferentes de agentes, sendo eles:

1. Um Agente *Gopher*

   É a classe mais simples de agentes. Ele pode executar tarefas mais simples baseadas em regras pré-definidas, como, por exemplo, uma agenda que pode emitir lembretes ao usuário (“sexta-feira, dentista às 14 horas”). Nesse caso o usuário cadastrou suas atividades, e em um período próximo ao evento o agente dispara os lembretes.

2. Um Agente de Serviço de Execução

   Esta classe possui um nível de sofisticação mais avançado que a primeira: os agentes devem executar uma tarefa de alto nível definida pelo usuário, como, por exemplo, organizar uma reunião com pauta e membros participativos.

3. Um Agente *Preditivo/Pró-ativo*

   Este tipo de classe possui agentes que buscam informações ou serviços de forma voluntária, sem perguntar ao usuário. Agentes dessa classe podem ser encontrados em sites de e-commerce, por exemplo, onde o agente conhece o perfil do usuário e através dessas informações oferece dicas e sugestões de produtos.

Para Nwana (1996) existem várias dimensões de classificação de agentes:

1. Mobilidade, estática ou dinâmica;
2. Deliberativo ou reativo;
3. Atributos ideais, aprendizagem, autonomia e cooperação;
4. Funcionalidade ou tipo de agente;
5. Híbridos.

Os agentes deliberativos possuem um modelo simbólico interno e atuam de acordo com esse modelo; já os reativos não possuem esse modelo e atuam quando são estimulados pelo ambiente. Através dos atributos é possível classificar um agente como: inteligente, colaborativo, de interface e de aprendizagem colaborativa.

A Figura 5.2 mostra a dinâmica dessas classificações através dos atributos ideias e a Figura 5.3 define todos os tipos de agentes.

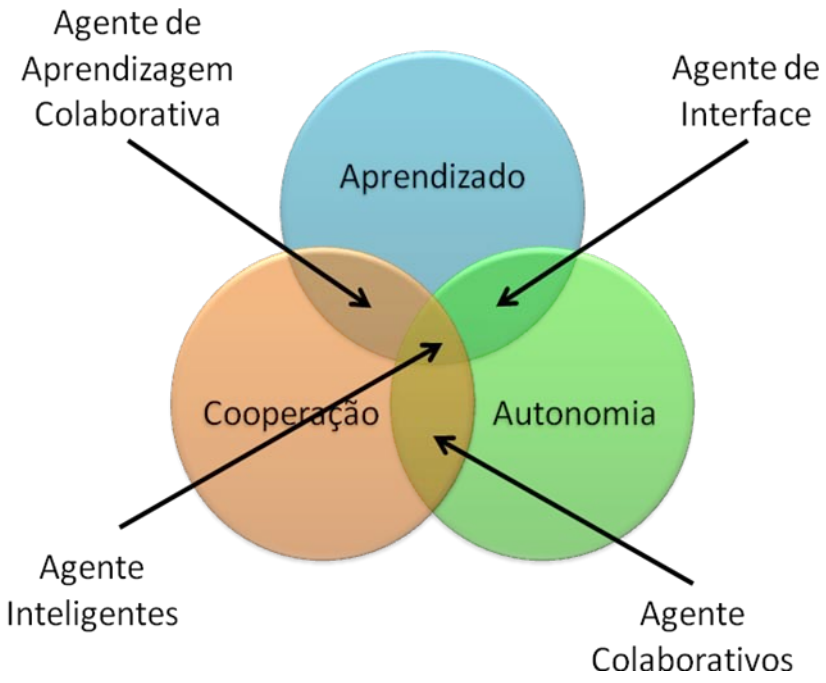


Figura 5.2 Uma visão geral dos atributos de um agente. Fonte: Nwana (1996).

Figura 5.3 Uma classificação de Agentes de Software. Fonte: Nwana (1996).

5.3 Arquitetura de um Agente

Para implementar um agente de software é necessário desenvolver uma pequena estrutura. Russel e Norvig (2004) afirmam que uma estrutura básica para

um agente de software é formada por uma arquitetura mais um programa. Ou seja, é preciso projetar um **programa de agente** que implemente as funcionalidades do agente e acoplá-la em uma estrutura de dispositivos, como sensores e atuadores que formam sua **arquitetura**.

Figura 5.4 Estrutura de Agentes proposta por Russell & Norvig (2004).

5.3.1 Programas de Agentes

Um programa implementado para um agente de software precisa receber as percepções atuais do ambiente como entrada dos sensores e retornar uma ação que deve ser executada pelos atuadores. Russell e Norvig (2004) descrevem quatro tipos básicos de programas de agentes que podem ser utilizados para quase todos os tipos de sistemas inteligentes. São eles:

- Agentes reativos simples
- Agentes reativos baseados em modelo.
- Agentes baseados em objetivos.
- Agentes baseados na utilidade.

5.3.1.1 Agentes Reativos Simples

Esse é o tipo mais simples de agente. Ele seleciona suas ações baseado nas percepções atuais do ambiente, ignorando todo um histórico de ações que possam ter sido realizadas anteriormente. Um exemplo de implementação desse tipo de programa seria o do agente "*aspirador de pó"*. Esse agente tem a função básica de

verificar se o ambiente tem sujeira; caso a resposta seja positiva (percepção que há sujeira no ambiente) a reação do aspirador de pó é aspirar essa sujeira, caso seja negativa o agente nada faz.

Nesse exemplo é possível notar que o agente *aspirador de pó* não precisa manter um histórico de ações anteriores. Não é necessário saber se em algum momento do passado esse agente encontrou ou não sujeira e se ele realizou ou não uma aspiração de pó. O importante para o agente é o momento atual, se há ou não sujeira no ambiente. A Figura 5.5 representa um diagrama esquemático para essa situação de agente.

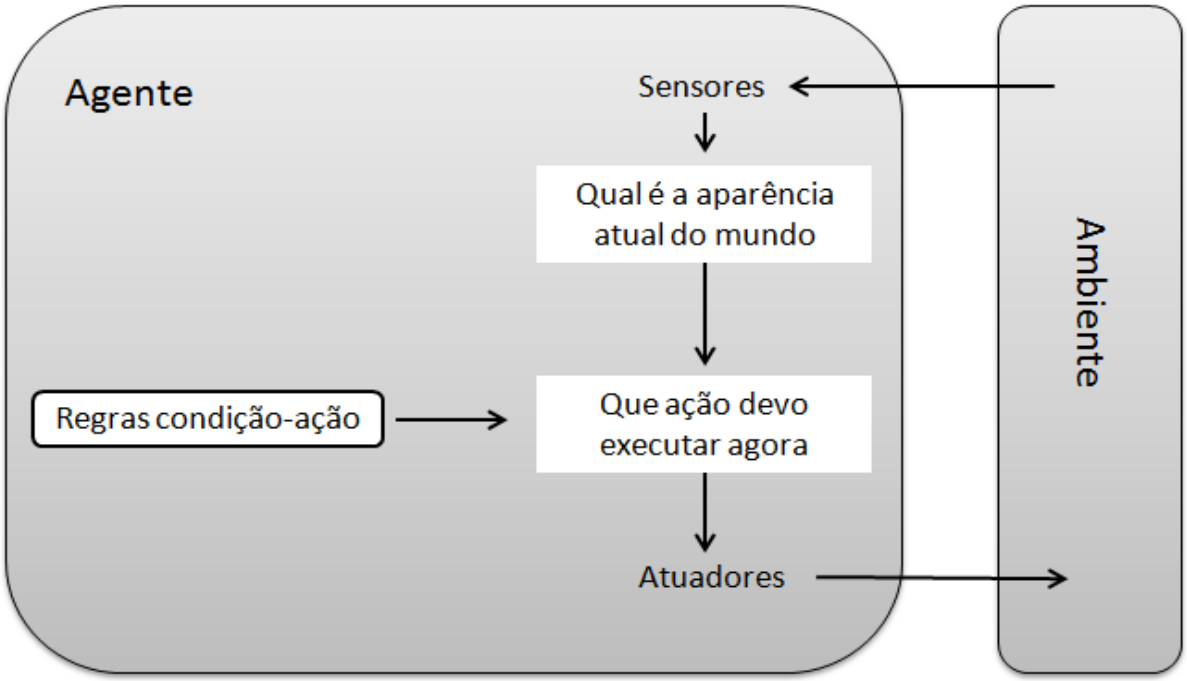


Figura 5.5 Diagrama esquemático de um agente reativo simples. Fonte: Russell & Norvig (2004).

5.3.1.2 Agentes Reativos Baseados em Modelos

Nesse tipo de programa o agente precisa manter algum tipo de *estado interno* que dependerá do histórico de percepções e assim reflita alguns aspectos não observados do estado atual. Imagine o agente motorista de carro que precisa realizar uma ultrapassagem, nesse exemplo é possível perceber que para realizar a tarefa o carro precisa se aproximar do veículo à sua frente o suficiente para poder realizar a tarefa de ultrapassagem.

Porém vários fatores do ambiente mudam na medida em que um carro se aproxima do outro. O agente precisa atualizar suas percepções de mundo no decorrer do tempo. Existem dois fatores importantes a serem notados nessas mudanças que o agente precisa saber: (i) a evolução das mudanças do ambiente sem a interferência do agente e (ii) como as ações do agente afetam esse mesmo ambiente. Esse conhecimento de "como o mundo funciona" é chamado de **modelo** de mundo. A Figura 5.6 mostra um diagrama que representa esse tipo de agente.

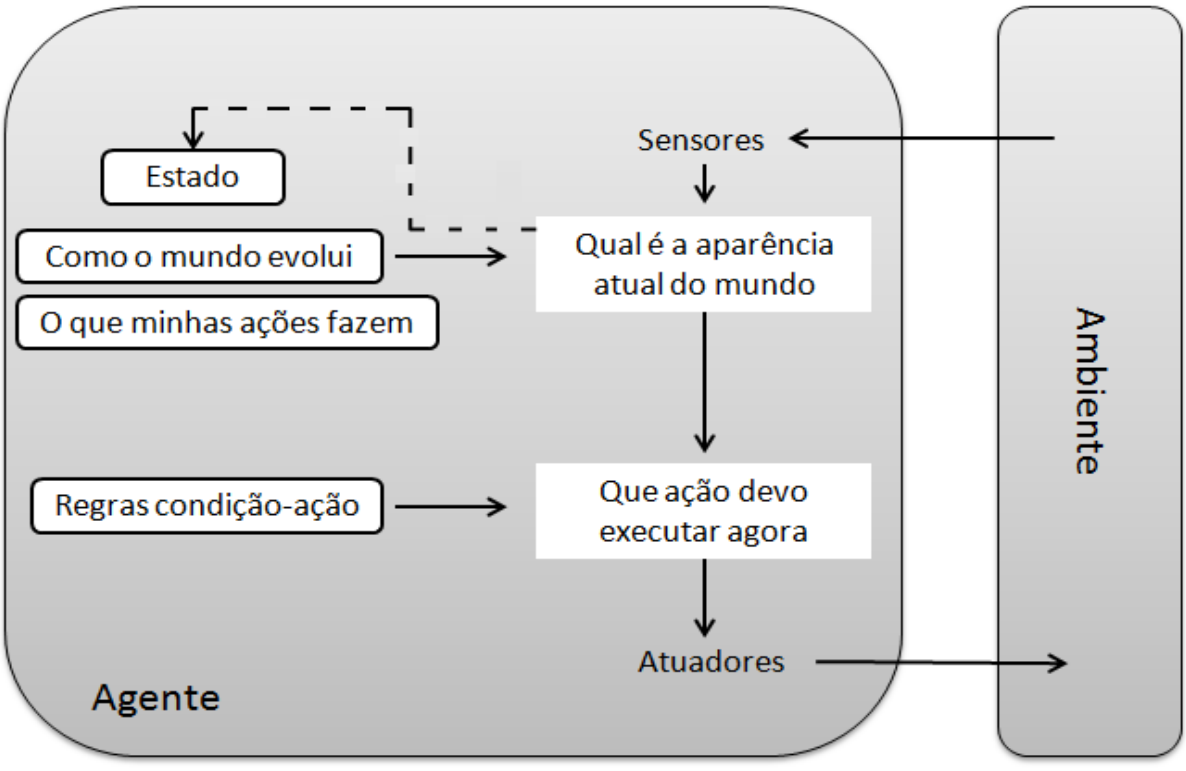


Figura 5.6 Diagrama esquemático de um agente reativo baseado em modelo. Fonte: Russell & Norvig (2004).

5.3.1.3 Agentes Baseados em Objetivos

O programa anterior só precisava saber do estado atual do mundo, mas imagine que o carro precise virar para direita ou para esquerda. A resposta para qual direção o carro deve tomar dependerá de onde o carro precisa chegar, ou seja, além de saber o estado atual do mundo o carro agora precisa ter um **objetivo** que direciona para onde o carro deve ir (caminho da esquerda ou direita).

Em alguns casos o objetivo do agente pode ser alcançado de forma direta (virar para direita ou para esquerda), porém há casos em que cumprir o objetivo do agente demanda várias ações ou mesmo uma grande sequência de ações até conseguir chegar ao resultado final de maneira satisfatória. Para isso o agente precisa realizar **buscas** e **planejamento** para encontrar uma sequência de ações que alcance o objetivo da tarefa. Uma solução seria a utilização de um agente GPS, que poderia realizar a busca de como chegar a um endereço predeterminado e planejar a sequência de ações que o carro precisa seguir para chegar no endereço desejado. A Figura 5.7 mostra um diagrama desse modelo.

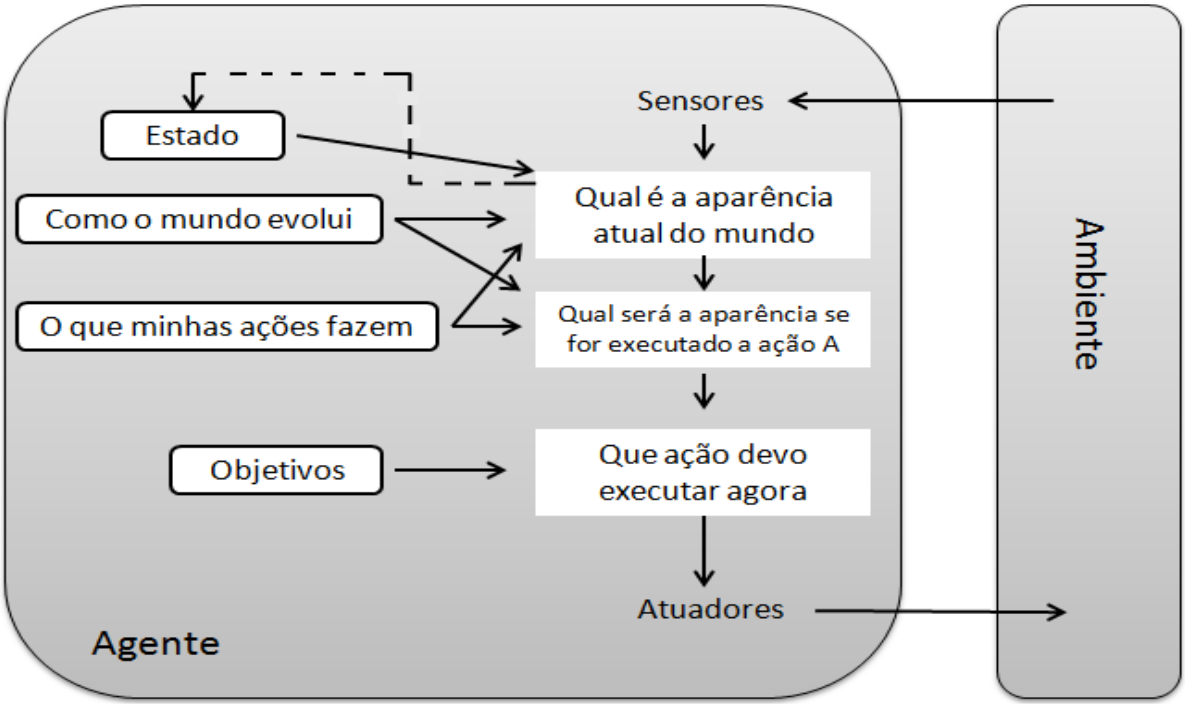


Figura 5.7 Diagrama esquemático de um agente baseado em objetivos. Fonte: Russell & Norvig (2004).

5.3.1.4 Agentes Baseados na Utilidade

Nem sempre os objetivos são suficientes para gerar um comportamento desejável na maioria dos ambientes. No exemplo do carro existem várias sequências de ações que o levarão ao destino final, como caminhos mais rápidos ou mais seguros. O modelo representado por objetivo distingue dois estados que o agente pode representar, possui o estado “feliz” e o “infeliz”. Fazer uso de um estado em detrimento do outro é considerar que esse estado escolhido tem mais **utilidade** que o outro.

A **função de utilidade** pode mapear um estado (ou vários) em um valor real e definir o grau de satisfação com a ação realizada. Ainda considerando o agente GPS, ele poderia fornecer o menor caminho a seguir, o caminho a pé, o caminho a ser feito em menor tempo considerando as condições do trânsito momento etc.

Especificar completamente as funções de utilidade pode permitir que o agente tome decisões racionais em tipos de casos onde os objetivos são inadequados ou contraditórios (velocidade versus segurança). A Figura 5.8 apresenta um diagrama desse modelo.

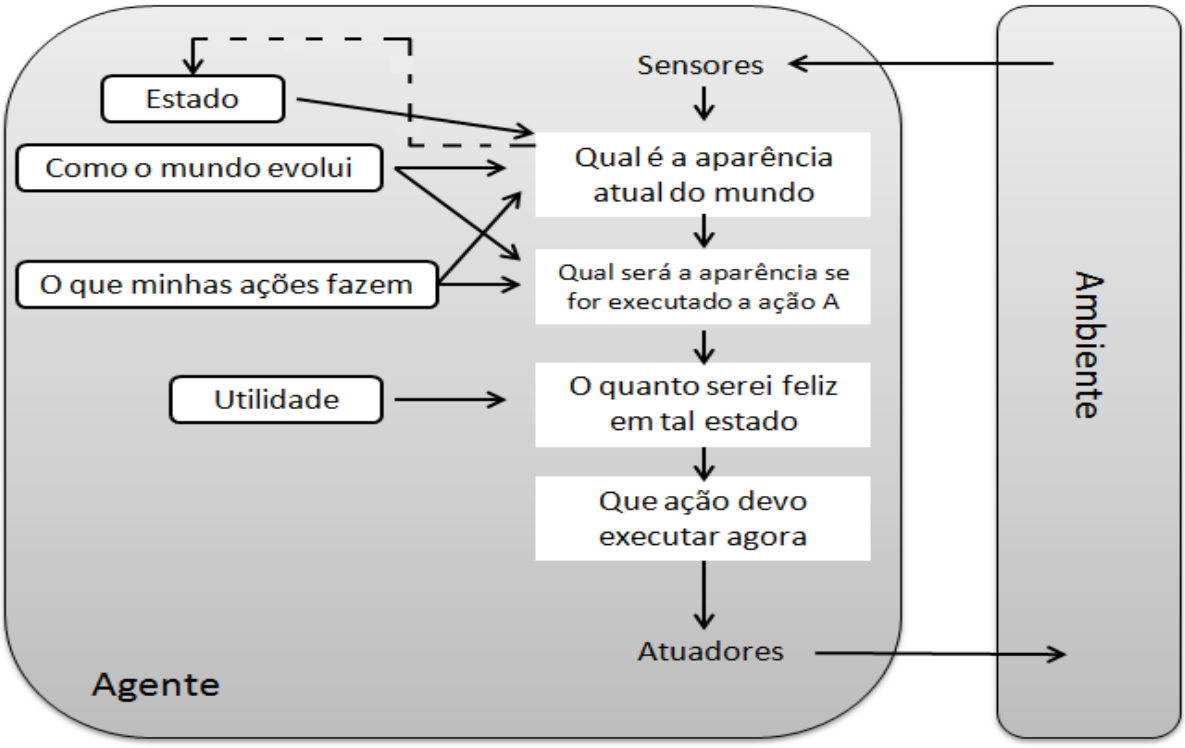


Figura 5.8 Diagrama esquemático de um agente baseado em utilidade. Fonte: Russell & Norvig (2004).

5.4 Considerações Finais

Este capítulo teve por objetivo apresentar uma ideia geral sobre a *teoria dos agentes inteligentes*. Definiu-se brevemente o que são agentes inteligentes, quando surgiram, o objetivo de utilizar máquinas inteligentes, a arquitetura e alguns modelos para desenvolver programas utilizados pelos agentes de software.

Para que o GuiBM possa oferecer as dicas de padrões da forma mais contextualizada possível levando em conta a intenção do projetista, utilizou-se o tipo de programa de agentes reativos baseados em modelos.

Para que as dicas sejam oportunas, o agente deve observar o que o designer está incluindo em sua interface e, a partir desses dados (componentes inclusos na interface), o agente estuda qual dica pode ser aplicada em um contexto específico. Se em mais de cinco minutos nenhum padrão específico for acionado, o agente envia uma dica aleatória de padrão, que pode ser de interação ou de modalidade.

No próximo capítulo, o *framework* GuiBuilder Multimodal será apresentado com detalhes e o tópico agentes será devidamente explorado.

Capítulo 6

GUIBUILDER MULTIMODAL

6.1 Considerações Iniciais

Nas primeiras décadas de existência da computação, o esforço se concentrava na funcionalidade e no desempenho das aplicações. Porém durante a década de 80, quando as ferramentas começaram a ser utilizadas por milhões de pessoas, passou-se a reconhecer que o sucesso de um aplicativo deveria ser medido por sua facilidade de utilização, tanto por usuários iniciantes quanto por usuários experientes (Van Dam, 1997).

Para Oviatt (1999), sistemas multimodais oferecem maior poder de expressão, naturalidade, flexibilidade e portabilidade ao usuário. Quando bem projetados, tais sistemas podem oferecer uma grande sinergia entre as modalidades, o que potencializa a interação entre o usuário e a aplicação, diferentemente de um sistema com tecnologia unimodal.

É preciso ressaltar que multimodalidade não é o excesso de informação, mas várias formas de apresentar a mesma coisa (Bernsen, 2008). A ideia é deixar o usuário livre para escolher, de forma natural, a melhor maneira de interagir com o aplicativo, seja ela por fala, gesto, expressão facial ou corporal.

No entanto, a construção de sistemas com tecnologia multimodal não é trivial. Um tema importante na geração de interfaces multimodais é a integração de requisitos de sincronização para combinar, estrategicamente diferentes modos em todo o sistema (Oviatt, 1999). É preciso ter em mente uma gama de informações

sobre os padrões e modalidades, tais como: o contexto de uso e a forma de combinação entre modalidades de entrada e de saída.

O capítulo 4 apresentou algumas propostas de *frameworks multimodais* que contribuem para o desenvolvimento mais rápido de interfaces multimodais. Porém, é importante salientar alguns pontos: (i) o pouco conhecimento do projetista na área multimodal; (ii) os padrões para a fusão de mais de uma modalidade e (iii) o oferecimento de uma interface intuitiva do *framework*. Tendo isso em vista, o GuiBM foi projetado para facilitar a implementação de sistemas multimodais, auxiliando designers através de uma interface similar a IDEs de desenvolvimento de GUIs (na qual o projetista poderá arrastar componentes) e do uso de boas práticas baseadas em padrões.

6.2 GuiBuilder Multimodal

O GuiBuilder Multimodal, ou simplesmente GuiBM, foi desenvolvido com o intuito de facilitar a construção de interfaces multimodais. O designer cria sua interface apenas arrastando componentes e adicionando dados às suas propriedades. A estrutura do *framework* GuiBM foi dividida em três módulos principais: (i) planejamento; (ii) supervisão e (iii) código.

No módulo de planejamento, foi utilizada a plataforma Abacus Java GUI Builder, a qual consiste em um *builder* visual desenvolvido totalmente em Java com o designer baseado no Delphi/VB. A Abacus foi concebida com o intuito de ajudar desenvolvedores a criar UI Java de maneira rápida (Abacus, 2012). A Figura 6.1 ilustra a versão final do GuiBM que foi adaptada sobre a plataforma da ferramenta Abacus.

O módulo de supervisão (ou inteligência), foi criado a partir da união dos conceitos de padrões de projeto de agentes inteligentes. O objetivo é oferecer dicas de padrões de acordo com o contexto da aplicação que o designer está desenvolvendo. O projetista pode aceitar ou não a sugestão; não cabe ao GuiBM a tarefa de restringir as opções do desenvolvedor. O módulo de código gera todo o código da interface criada pelo designer na linguagem Java.

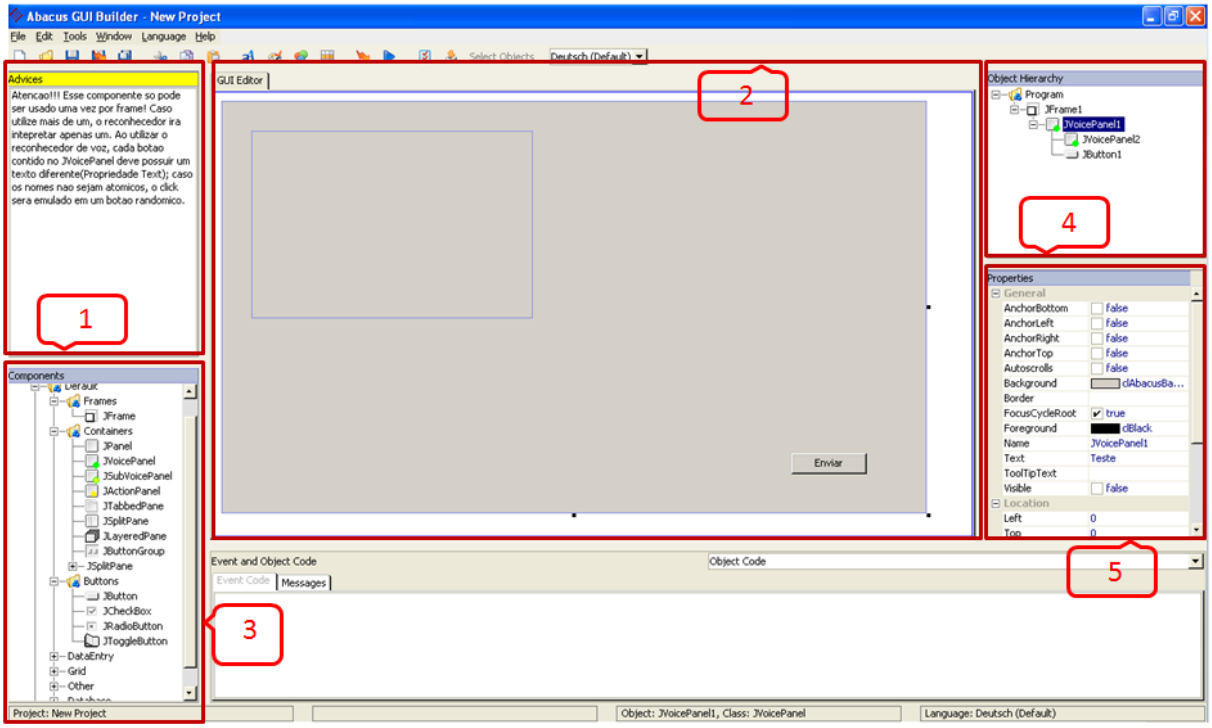


Figura 6.1 Campos do GuiBM.

Na Figura 6.1, é possível notar 5 campos específicos do GuiBM: (i) as dicas oferecidas pelo GuiBM; (ii) Gui Editor, no qual o usuário visualiza e edita a interface que está desenvolvendo em tempo real; (iii) os componentes existentes; (iv) a hierarquia de objetos inseridos na interface e (iv) as propriedades dos componentes. Para iniciar a implementação de qualquer interface, o usuário deve sempre inserir um JFrame no GUI Editor e, a partir desse componente, vários outros podem ser atribuídos à interface.

6.2.1 Reconhecimento de voz

Uma interface com voz (Voice User Interface - VUI) consiste em interagir com uma aplicação apenas utilizando a linguagem falada. Uma interface VUI inclui: (i) prompts ou sistemas de mensagens; (ii) gramáticas, ou seja, um conjunto de palavras que podem ser utilizadas durante a interação e (iii) fluxo de diálogo, o qual define as ações que podem ser utilizadas no sistema (Cohen *et al.*, 2004).

Implementar uma interface que utiliza a modalidade de voz necessita de vários requisitos iniciais, além do conhecimento do conceito de uma interface de voz.

É preciso se preocupar, por exemplo, com: (i) a instalação de uma ferramenta de reconhecimento; (ii) definição dos pacotes de reconhecimento, no caso, um pacote com reconhecimento para língua portuguesa; (iii) a configuração do SDK com o Java ou outra linguagem de programação e etc. Neste trabalho, utilizou-se a API Java Speech.

A API Java Speech define um padrão, fácil de usar, com duas tecnologias centrais suportadas: o reconhecimento e a síntese da voz. O reconhecedor é capaz de ouvir o que o usuário disse e determinar o que foi pronunciado, ou seja, ele processa o áudio de entrada e converte em texto. O sintetizador faz o processo inverso, ou seja, processa um texto e oferece como saída um áudio. A Figura 6.2 ilustra o código para o sintetizador.

```
public class Sintetizador {

        static Synthesizer synthesizer;
        static EngineListener engineListener = new EngineAdapter() {
                public void engineError(EngineErrorEvent e) {
                        System.out.println(
                                "Engine error: " + e.getEngineError().getMessage());
                }
        };

        public Sintetizador() {
                try {
                        synthesizer = Central.createSynthesizer(null);
                        if (synthesizer != null) {
                                synthesizer.allocate();
                                synthesizer.addEngineListener(engineListener);
                        }

                } catch (Exception e) {
                        System.out.println(e.toString());
                }
        }
```

Continua...

Continuação.

```
        public static void speak(String s) {
                if (synthesizer != null) {
                        try {
                                synthesizer.speak(s, null);
                        } catch (Exception e) {
                                e.printStackTrace();
                        }
                } else {
                        System.out.println(s);
                }
        }

        public static void main(String args[]) {
                Sintetizador vp = new Sintetizador();
                String texto = " <JSML> "
                        + "<BREAK MSECS=\"300\"/>"
                        + "<PROS PITCH=\"80\" RANGE=\"50\" RATE=\"150\">"
                        + "Testando o Sintetizador."
                        + "</PROS>"
                        + " </JSML> ";
                Sintetizador.speak(texto);
        }

}
```

Figura 6.2 Sintetizador de Voz.

No exemplo do código do sintetizador o texto “Testando o Sintetizador” é enviado para o método *speak* e, então, pronunciado pela aplicação. Um campo de utilização dos sintetizadores de voz são as ferramentas que auxiliam deficientes visuais na leitura de textos no computador. No contexto do GuiBM, essa tecnologia não foi utilizada, visto que a interface apenas recebe comandos via voz. Com isso, utilizou-se apenas a tecnologia de reconhecimento.

Para que o reconhecimento de fala seja possível, o desenvolvedor precisa se preocupar em definir palavras que devem ser reconhecidas e colocá-las em uma gramática.

A Figura 6.3 ilustra uma gramática de reconhecimento. Nota-se que existem comandos que podem ser reconhecidos dentro da gramática, por exemplo, a sentença “Hoje é <diaDaSemana>”, onde <diaDaSemana> é substituído por: segunda-feira, terça-feira, quarta-feira, quinta-feira, sexta-feira, sábado e domingo.

```
grammar gramatica;

public <comandos> = usuários | ajuda | ir para | examinar | legal;
public <direcao> = Esquerda | Direita | Cima | Baixo;
public <name> = Aquiles | Ricardo | Ozifrankly | Stiff |  Diego | Daniel | Wagner | computador | Clístenes | Antônio | Fernando | Clélio | Sérgio |
Godela | Lirisnei | Welkey | Guido | Rogério | Bráu | Ronaldo Mapurunga | Flávio | Luiz Eduardo | Jorgiano | Aloise | Júlio | Samuel | Rummenigge |
Abner | Felipe | João Carlos | Marcela | Gianna | Luiz Marcos | LM | Ygara | César | Salvar;
public <sentence> =
    Cadê <name> |
    Bom Dia |
    Oi |
    Olá |
    Tudo Bem |
    Qual o seu Nome |
    De onde teclas |
    De onde teclas <name> |
    Agora estou falando em meu idioma |
    Meu nome é <name> |
    Hoje é <diaDaSemana> |
    Envie mensagem para <name> |
    Ei <name> tá pronto |
    terminar;
public <diaDaSemana> = segunda-feira | terça-feira | quarta-feira | quinta-feira | sexta-feira | sábado | domingo;
```

Figura 6.3 Gramática de reconhecimento.

Com o intuito de facilitar o desenvolvimento de interfaces com reconhecimento de voz, o GuiBM já fornece todos os requisitos para o reconhecimento de voz implementados em sua plataforma. A API Java Speech propicia a comunicação com a ferramenta de reconhecimento ViaVoice IBM.

Foram desenvolvidos dois componentes para facilitar a interação via voz: (i) JVoicePanel e (ii) JSubVoicePanel. Caso o usuário deseje criar uma interface que funcione via voz, ele precisa adicionar um JVoicePainel ao seu JFrame e um nome ao painel na propriedade *text*. Os componentes que podem ser utilizados no painel JVoicePainel são: JSubVoicePainel, JButton, JCheckBox, JRadioButton e JComboBox.

O desenvolvedor apenas arrasta o componente que desejar, dentre as opções listadas acima, e preenche a propriedade *text* com o nome a ser inserido na gramática de reconhecimento. Essa palavra irá referenciar o componente criado. Por exemplo, se o desenvolvedor adicionar um JButton a uma interface e na propriedade *text* escrever a palavra “enviar”, quando o usuário final utilizar essa interface bastará pronunciar o nome do botão e o reconhecedor fará a ação de clique no botão chamado. A Figura 6.4 ilustra um exemplo de interface com a utilização do JVoicePanel. As duas telas representam a mesma interface, porém a diferença é que, para oferecer ao usuário final um feedback que uma interação por voz pode ser usada, a tela do lado direito apresenta um contorno no painel de voz na cor laranja.

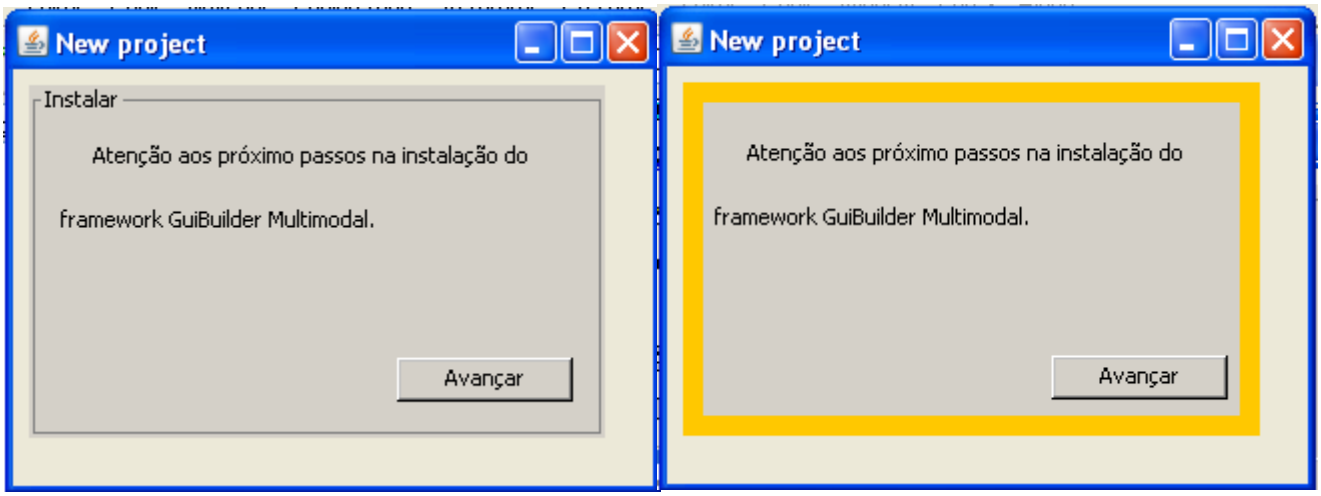


Figura 6.4 Exemplo da utilização do componente JVoicePainel.

Como o JVoicePanel só pode ser atribuído um a cada JFrame e em alguns casos a interface precisa definir porções de gramáticas com contextos diferentes em uma mesma tela, o GuiBM oferece a opção do componente JSubVoicePanel. O JSubVoicePanel funciona basicamente como o JVoicePanel, mas associa uma ou várias sub-gramáticas dentro de um único JFrame. Para utilizar esse artifício o usuário precisa inserir o JSubVoicePanel dentro de um JVoicePanel e também adicionar um nome na propriedade *text* do subpainel.

Existem casos em que é preciso ter mais de um painel diferente que possa interagir através da modalidade de voz. Para suprir essa necessidade foi criado o componente JSubVoicePainel, no qual o designer pode criar submenus de ações com reconhecimento de voz. Por exemplo, uma aplicação para realização de pedidos em um restaurante, como ilustra a Figura 6.5, na qual o garçom deve anotar o pedido do cliente e enviar a informação para a cozinha.

Na interação por voz, o garçom pode fazer a chamada para os submenus: Entradas, Prato Principal, Bebidas e Sobremesa. À medida que isso acontece, o foco em laranja muda para a região chamada, oferecendo um feedback de onde o usuário se encontra naquele momento na aplicação. Caso o usuário esteja em um submenu, já tenha realizado a ação necessária e precise enviar os dados para cozinha, ele deve voltar para a tela principal, realizar a chamada para o painel Pedido, e então acionar o botão “Enviar”.

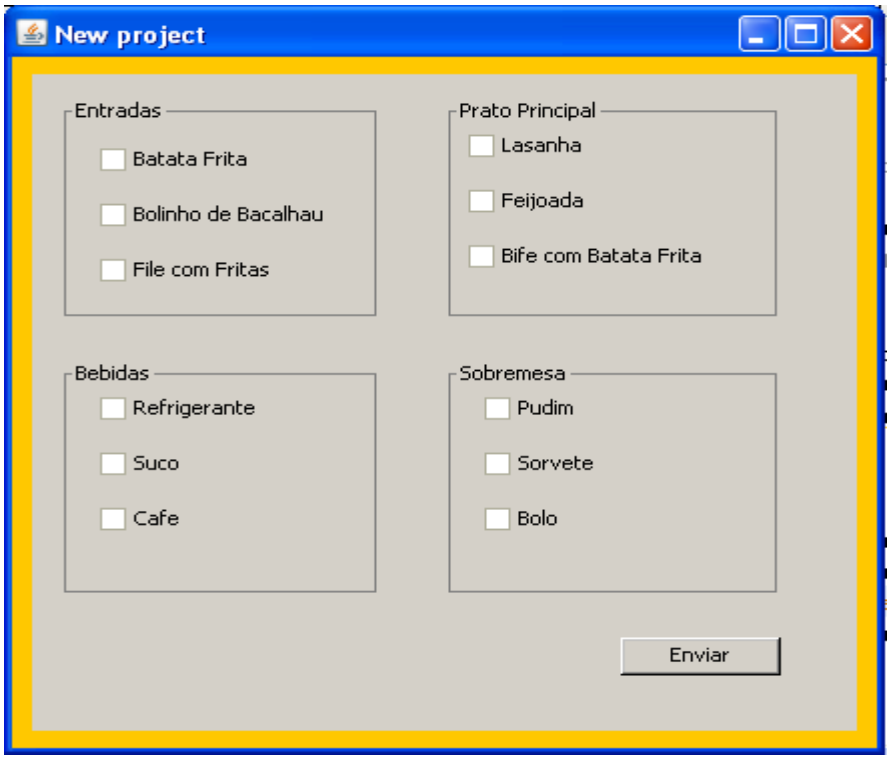


Figura 6.5 Aplicação de pedidos em um restaurante.

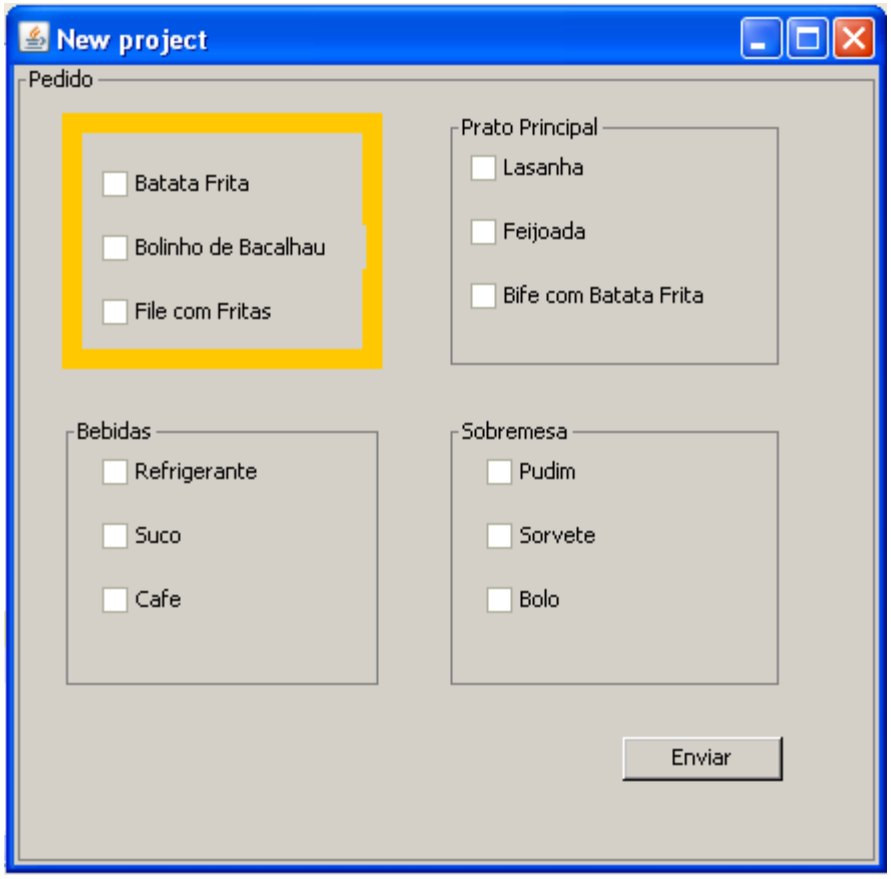


Figura 6.6 Exemplo de utilização do JSubVoicePainel.

6.2.2 Reconhecimento de Gesto

Para a interação com a modalidade de gesto o GuiBM utiliza o *framework* Mint (2012), desenvolvido pelo mesmo grupo de pesquisa. Através de uma câmera e da utilização de luvas com cores diferentes o *framework* Mint lê o gesto do usuário, executa e reconhece a ação e envia o resultado do reconhecimento via TCP/IP para a aplicação. Na interface do Mint existem duas abas: Segmentation e Gesture Recognition.

A Figura 6.7 apresenta a primeira aba, do lado direito da interface, existem algumas opções de configuração. Pode-se utilizar até duas luvas; no campo *Gloves*, o usuário pode configurar a área de reconhecimento das mesmas. Após configurar a sua área de reconhecimento, clica-se em "*Save"* e os dados de HueMin, HueMax, SaturationMin e SaturationMax para cada luva são gravados. Para iniciar/parar a câmera é preciso clicar no botão "Start/Stop Live".

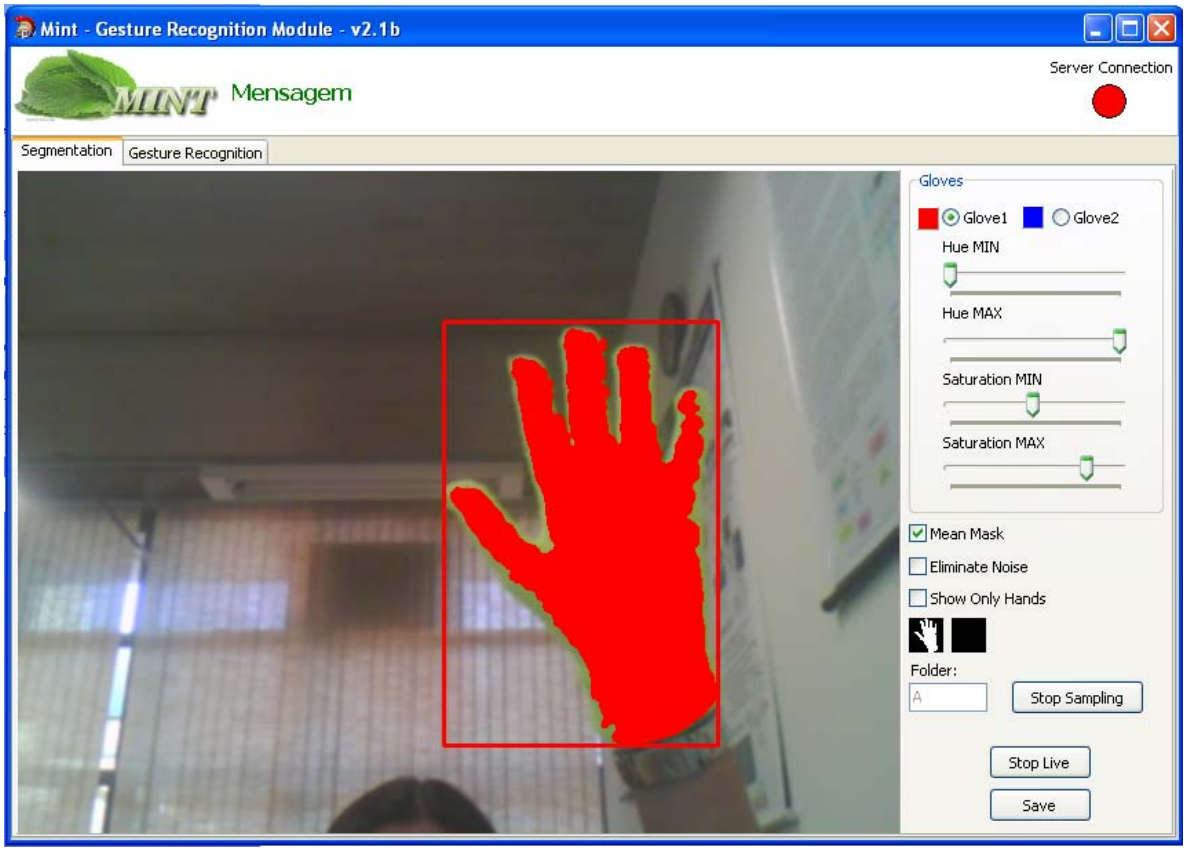


Figura 6.7 Aba Segmentation do Framework Mint.

Na aba “Gestura Recognition”, ilustrada na Figura 6.8, existem três campos de configuração: (i) Server Configuration; (ii) Recognition Status e (iii) Gesture Recognition Options. O primeiro define os parâmetros para o envio de informações. Quando a conexão estiver ativa e funcionando, esse campo fica verde e, caso exista alguma falha na conexão, vermelho.

Para iniciar o reconhecimento dos gestos é preciso clicar no botão que se encontra no segundo campo. O “Recognition Status” indica o status de reconhecimento. No exemplo da figura abaixo, só existe a interação com uma luva. Porém pode-se utilizar duas, como mencionado anteriormente.

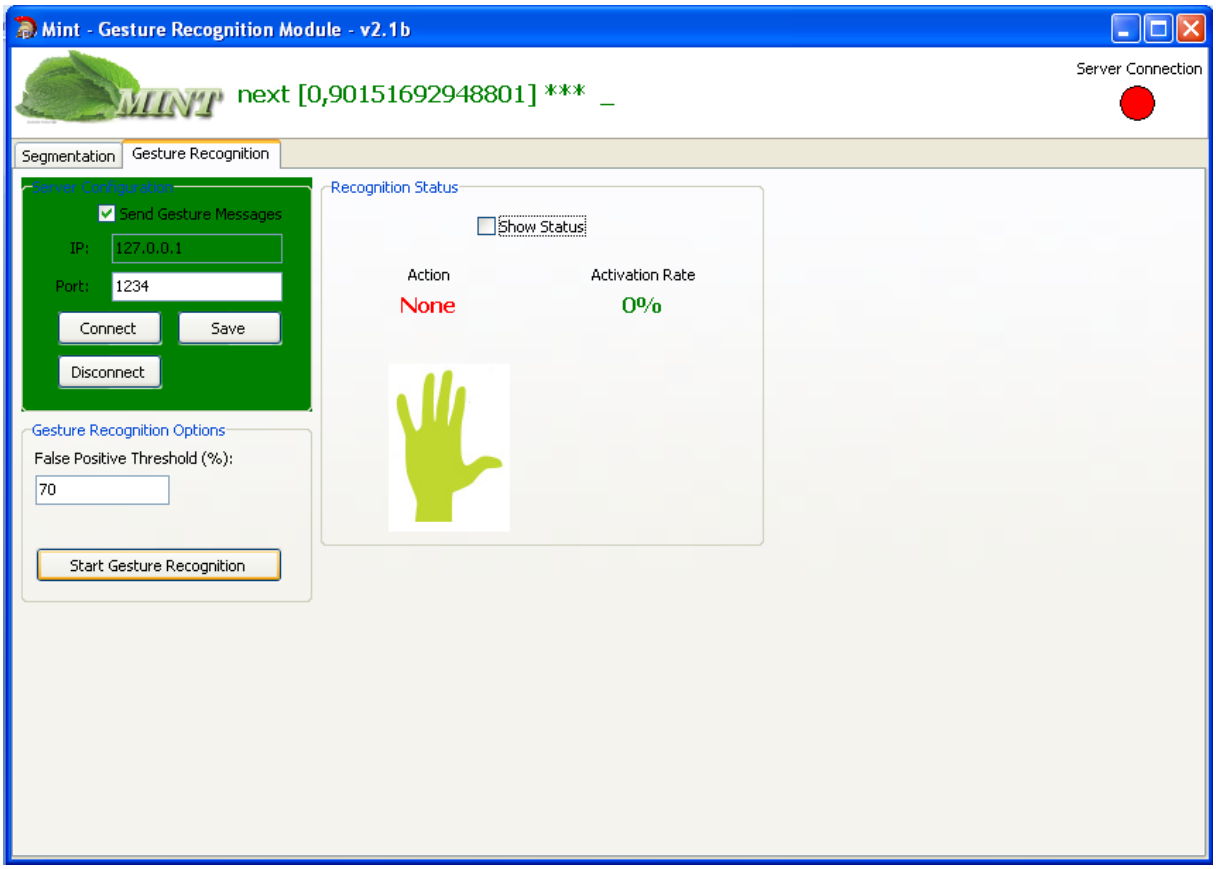


Figura 6.8 Aba Gestura Recognition do Framework Mint.

As mensagens enviadas pelo Mint são: (i) next, vai para o próximo componente; (ii) prev, volta para o componente anterior; (iii) confirm, confirma a escolha do componente e (iv) select, seleciona o componente. Uma vez iniciado o reconhecedor de gesto ele pode funcionar para qualquer interface, mesmo que tais interfaces não tenham sido criadas através do GuiBM. A Figura 6.9 apresenta as mensagens enviadas pelo Mint, as quais são visualizadas no topo do *framework*.

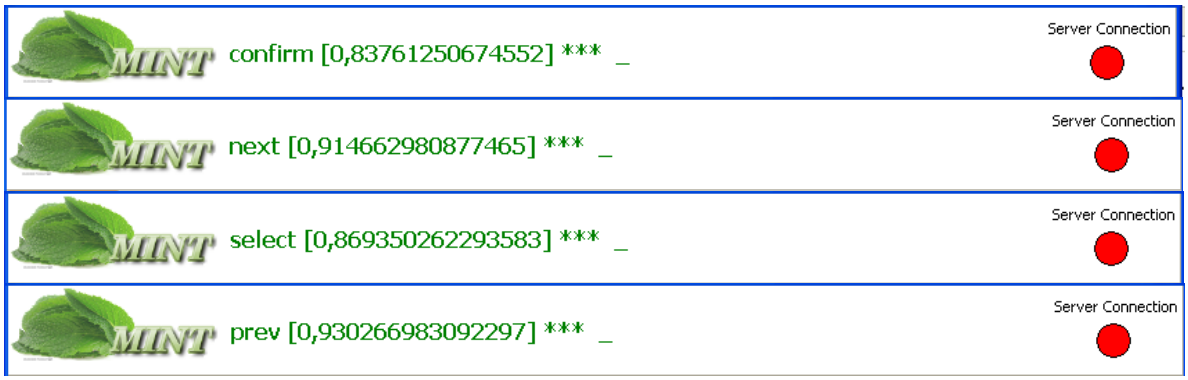


Figura 6.9 Mensagens de reconhecimento Mint.

O IP utilizado é o 127.0.0.1, também conhecido como loopback, que faz a comunicação com a máquina atual. Dessa forma, o Mint envia mensagens para um servidor que está rodando na mesma máquina que o *framework*.

6.2.3 Padrões utilizados no GuiBM

Uma das propostas do *framework* GuiBM é oferecer dicas de padrões para o desenvolvedor. O objetivo é orientar o projetista a implementar uma interface intuitiva e simples para ser utilizada pelo usuário final. No campo 2 da Figura 6.1, que foi apresentada anteriormente, existe uma área chamada *"Advices"*. Na qual o desenvolvedor recebe as dicas.

Para que as dicas sejam pertinentes ao que o desenvolvedor está projetando em sua interface, foi utilizado o conceito de agentes inteligentes. O agente recebe informações sobre o que o usuário está colocando em sua interface, no campo GUI Editor do GuiBM, e à medida que ele as recebe verifica se existe alguma dica que possa ser útil em sua base de dados.

A implementação do agente foi realizada através do *framework* Jadex (2012). O projeto do *framework* Jadex tem como objetivo oferecer apoio na implementação

de programas que utilizam o conceito de agentes. Ele foi desenvolvido em Java e permite desenvolver agentes orientados a objetivos, seguindo o modelo BDI (Belief, Desire and Intection).

O agente utilizado no GuiBM recebe informações do que o desenvolvedor está inserindo na criação da sua interface multimodal e suas respectivas hierarquias. A partir desses dados, o agente verifica se há algum padrão associado ao contexto da aplicação ou algum componente que possa ser utilizado pelo desenvolvedor e que talvez seja mais intuitivo para a interação do usuário com a interface. A Figura 6.9 representa essa interação.

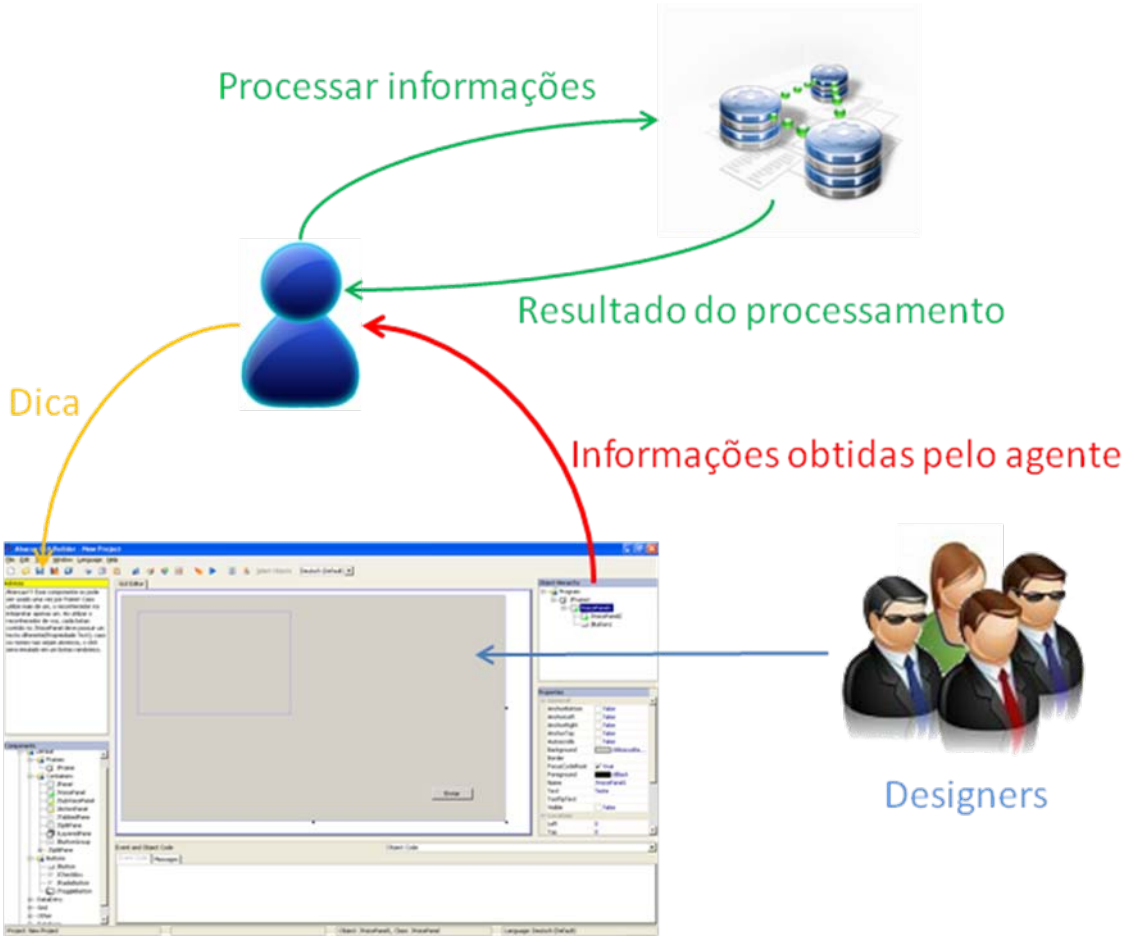


Figura 6.10 Interação Desenvolvedor -> GuiBM -> Agente -> GuiBM -> Desenvolvedor.

Os padrões contidos no GuiBM foram atribuídos considerando as possíveis interfaces que podem ser desenvolvidas com o auxílio do *framework*. Por exemplo, é possível desenvolver uma interface para uma aplicação com formulários; nesse caso os agentes possuem padrões e dicas especificas para a realização da tarefa em

questão, incluindo padrões das três categorias que foram apresentadas no capítulo 03.

O agente do GuiBM tem disponível, em sua base de dados, um conjunto de 26 padrões com dicas associadas, descritos a seguir:

1. Alternating Row Colors

   Dica:

   i. Sua aplicação faz uso de tabelas para fornecer informações ao usuário? Use cores alternadas entre as linhas para fazer com que a tabela fique mais legível. Use duas cores de baixa saturação, por exemplo, branco e um tom claro de uma outra cor, que são ligeiramente diferentes. Para maiores informações leia o padrão *Alternating Row Colors*.

2. Auto Complete

   O padrão é acionado quando o usuário faz uso do componente JTextArea.

   Dica:

   i. Seu usuário final precisa inserir um rótulo que faz parte de um grande conjunto de caracteres? Você pode sugerir nomes de possíveis rótulos enquanto o usuário digita sua entrada. O padrão *Auto Complete* auxilia o usuário a preencher, de forma mais rápida, entradas de texto. Para maiores informações leia o padrão *Auto Complete*.

3. Booking

   Dica:

   i. Sua aplicação realiza uma reserva como, por exemplo: um voo, hotel, carro, etc.? Permita que os usuários procurem o ‘Objeto’ de forma flexível, especialmente sobre a data/hora. Em seguida faça a reserva real. Para maiores informações leia os padrões *Booking, Purchase* e *Process*.

4. BreadCrumbs

   Dica:

   i. Os usuários sempre precisam saber em que ponto da aplicação eles se encontram. Para isso você pode mostrar o caminho hierárquico e partir do nível mais alto até o atual e fazer com que cada passo seja clicável. Utilize o padrão *Breadcrumbs* (também conhecido como migalhas de pão) caso a hierarquia tenha nível de profundidade maior que 3. Para maiores informações leia o padrão *Breadcrumbs*.

5. Carrousel

   Dica:

   i. O usuário precisa ver uma série de imagens ou fotos? Geralmente a tela é maximizada para as fotos. Utilize uma representação pequena dos controles, os quais devem ser utilizados no topo da foto ou abaixo dela. Para mais detalhes leia o padrão *SlideShow* ou *Carrousel*.

6. Collapsible Panels

   Dica:

   i. Muita informação de texto e pouco espaço?Crie painéis que podem ser abertos e fechados que são independentes uns dos outros. Utilize quando o usuário precisar acessar um subconjunto de informações. A etiqueta deve ser clicável e alternar entre *aberto* e *fechado*. Assim, você pode ilustrar apenas as informações ou funcionalidades essenciais e esconder o restante. Painéis flexíveis são muito eficientes quando não se tem muito espaço na tela. Para mais informações leia o padrão *Collapsible Panels*.

7. Color Coded Section

   Dica:

   i. Sua aplicação precisa que os usuários reconheçam se estão no local correto? Coloque cores para cada seção importante. Use isso para aplicações complexas com vários subitens ou subcategorias que quase formam uma aplicação própria. Normalmente é utilizado em aplicação de compra. Para maiores informações leia o padrão *Color Coded Section*.

8. Data Selector

   Dica:

   i. Deixe sempre orientações de como preencher um campo de texto. Se o usuário precisa entrar com um valor ou número de telefone coloque ao lado ou abaixo do campo um exemplo de preenchimento, como 16 1111-1111. Para maiores informações sobre orientações ao usuário leia o padrão *Unambiguous Format*. Outro padrão que permite a entrada de datas em formato correto é o padrão *Data Selector*.

9. Entrada Adequada de Conteúdo

   Dica:

   i. Sua aplicação oferece entrada de vários tipos de dados como: imagens, som, vídeo, texto e etc? Saiba escolher adequadamente qual modalidade de interação você poderá colocar na sua interface. Para isso saiba que: (i) apontar não requer memorização; (ii) falar não requer visualização de todos os itens e (iii) selecionar itens através da modalidade de fala diminui o esforço de navegação. Para maiores

informações leia o padrão *Entrada Adequada de Conteúdo*.

ii. Sua aplicação foi projetada para usuários diferentes usando dispositivos diferentes em ambientes e situações diferentes? Você pode utilizar as modalidades de voz e de toque na aplicação. Saiba que a digitação é poderosa para uma série de tarefas, porém, em dispositivos pequenos usar um teclado pode ser chato e lento. Interação por voz, como entrada de texto, é uma boa alternativa. Além disso, apontar é uma ótima escolha para selecionar objetos. Para maiores informações leia o padrão *Entrada Adequada de Conteúdo*.

10. Fala e Gesto

Dica:

i. Algumas aplicações requerem uma entrada com comandos compostos, como copiar um objeto para outro local, selecionando e enviando para o destino. Caso sua aplicação necessite de uma entrada composta, permita que o usuário entre rapidamente com vários parâmetros de comandos utilizando a fala associada a um gesto. Para maiores informações leia o padrão *Fala e Gesto*.

11. Focus

Dica:

i. O componente JActionPanel foi desenvolvido com o intuito de fornecer ao usuário um foco do momento de interação que ele se encontra. Para manter o foco do usuário, leia o padrão *Focus.*

12. Form

Esse padrão é acionado como dica (i), se o usuário inserir mais de 5 JLabel e JTextField em sua interface; as outras dicas são acionadas randomicamente pelo agente.

Dicas:

i. Os usuários precisam fornecer informações pessoais e enviá-las para a aplicação? Ofereça ao usuário o preenchimento de um formulário com os elementos necessários. Um formulário é essencialmente uma coleção de rótulos e campos de entrada em uma única tela. Para maiores informações leia o padrão *Form*. Para que o seu formulário seja multimodal leia sobre o padrão *Formulário com Habilitação de Fala*.

ii. Caso o seu usuário precise inserir dados particulares em sua aplicação e vez ou outra voltar a fazer a mesma ação, ofereça a ele a possibilidade de armazenar essas informações pessoais para uso posterior. Deve-se também oferecer a possibilidade de realização de um registro, tanto por sua própria iniciativa ou em um momento em que se justifique, por exemplo, ao iniciar ou concluir um pagamento. Para maiores informações leia os padrões *Registration* e *Form*.

iii. O seu usuário precisa comprar um produto e já o selecionou? A aplicação permite a compra de bens, normalmente, um comércio. A compra também pode ser parte de uma tarefa maior, como uma reserva. Permita que o usuário realize um checkout. O checkout é uma tarefa de cinco etapas do processo de compra, descrita a seguir: (i) identificar o cliente; (ii) selecionar o endereço de envio e opções especiais; (iii) selecionar o meio de pagamento; (iv) verificar toda a ordem; (v) confirmar o pedido e (vi)

receber a confirmação do pedido, talvez por e-mail. Para maiores informações leia os padrões *Purchase Process* e *Form*.

13. Formulário com Habilitação de Fala

    Dica:

    i. Os usuários precisam fornecer informações pessoais e enviá-las para a aplicação? Ofereça a ele o preenchimento de um formulário com os elementos necessários. Um formulário é essencialmente uma coleção de rótulos e campos de entrada em uma única tela. Para maiores informações leia o padrão *Form*. Para que o seu formulário seja multimodal leia sobre o padrão *Formulário com Habilitação de Fala*.

14. Inplace Replacement

    Dica:

    i. Se o seu usuário precisar ver alguns detalhes sobre um determinado produto dentre uma lista grande de itens, faça com que ao selecionar o item um espaço adicional, com detalhes sobre o mesmo, seja exibido. Usar somente quando a quantidade de informação adicional for limitada em 3-5 linhas de texto. Para maiores informações leia sobre o padrão *Inplace Replacement.*

15. Input Error Message

    Dica:

    i. Às vezes os usuários entram com dados que não são válidos e precisam corrigi-los. Você precisa dizer, ao usuário, que existe um problema e como ele pode resolver. Também é importante informar onde ocorreu o problema. Para maiores informações leia o padrão *Input Error Message.*

16. Interação Baseada em Atalhos
    a. Dica:
        i. Se a sua aplicação exigir que o usuário selecione um item dentre um grande conjunto de opções (menu) e a tela for muito pequena para mostrar todas, você pode utilizar a modalidade de fala, pois selecionar objetos ou ações através da voz pode acelerar significativamente a interação. Para maiores informações leia o padrão *Interação Baseada em Atalhos.*

17. Login

    A dica é acionada quando o desenvolvedor faz uso do componente JPasswordField.

    Dica:

        i. Não utilize a modalidade de voz para inserir senha pessoal. Alguém pode ouvir! Mantenha a segurança do usuário. Para maiores informações leia o padrão *Login*.

18. Progress

    Dica:

        i. O componente JProgressBar tem como princípio de usabilidade o Feedback. É utilizado, geralmente, para oferecer suporte ao usuário no que diz respeito ao progresso da tarefa que está sendo executada. Para maiores informações leia o padrão *Progress*.

19. Purchase Process

    Dicas:

        i. Sua aplicação realiza uma reserva como, por exemplo: um voo, hotel, carro, etc.? Permita que os usuários procurem o ‘Objeto’ de forma flexível,

especialmente sobre data/hora. Em seguida faça a reserva real. Para maiores informações leia os padrões *Booking* e *Purchase Process*.

ii. O seu usuário precisa comprar um produto e já o selecionou? A aplicação permite a compra de bens, normalmente, um comércio. A compra também pode ser parte de uma tarefa maior, como uma reserva. Permita que o usuário realize um checkout. O checkout é uma tarefa de cinco etapas do processo de compra, descritas a seguir: (i) identificar o cliente; (ii) selecionar o endereço de envio e opções especiais; (iii) selecionar o meio de pagamento; (iv) verificar a ordem toda; (v) confirmar o pedido e (vi) receber a confirmação do pedido, talvez por e-mail. Para maiores informações leia os padrões *Purchase Process* e *Form*.

20. Registration

Dica:

i. Caso o seu usuário precise inserir dados particulares em sua aplicação e vez ou outra voltar a fazer a mesma ação, ofereça a ele a possibilidade de armazenar essas informações pessoais para uso posterior. Deve-se oferecer a possibilidade de realização de um registro, tanto por sua própria iniciativa ou em um momento em que se justifique, por exemplo, ao iniciar ou ao concluir um pagamento. Para maiores informações, leia os padrões *Registration* e *Form*.

21. Shield

Dica:

i. Proteja sempre o seu usuário de ações ou funções que podem causar um efeito (secundário) irreversível. Você pode fazer isso inserindo um escudo de proteção. Adicione uma camada de proteção extra para algumas funções onde o usuário pode cometer erros. Solicite confirmações da intenção do usuário com uma resposta padrão e uma opção segura. Para mais informações leia o padrão *Shield.*

22. Slide Show

Dica:

i. Seu ComboBox tem mais de 7 opções? Para o usuário final, uma lista com muitas opções de escolha pode ser cansativa e desmotivante para continuar interagindo com a aplicação. Como alternativa você não pode representar as informações do ComboBox por alguma imagem, como um mapa. Ou o componente *SlideShow*"

23. Tabs

Dicas

i. Se o seu usuário precisar acessar uma seção especial de informações disponíveis, mostre uma linha horizontal de guias com etiquetas das secções. Destaque a guia selecionada no momento de interação atual e deixe todas as outras abas clicáveis. Para maiores informações leia o padrão *Tabs*.

24. Unambiguous Format

Dica:

i. Deixe sempre orientações de como preencher um campo de texto. Se o usuário precisa entrar com um

valor ou número de telefone coloque ao lado ou abaixo do campo um exemplo de preenchimento, como 16 1111-1111. Para maiores informações sobre orientações ao usuário leia o padrão *Unambiguous Format*. Outro padrão que permite a entrada de datas em formato correto é o padrão *Data Selector*.

25. View

    Dica:

    i. Se o seu usuário precisa gerenciar uma coleção de objetos, você pode criar uma visão geral dos objetos que juntos podem ser significativos para ele. Para maiores informações, leia o padrão *View.*

26. Wizard

    Dica:

    i. Se o seu usuário precisar seguir uma série de passos simples para concluir uma tarefa global (complexa), faça com que ele realize subtarefas em cada passo, colocando sempre o passo anterior como um pré-requisito para o próximo. Para maiores informações leia o padrão *Wizard.*

Todas as dicas podem ser acionadas randomicamente e, se em um intervalo de tempo maior do que 5 minutos nenhum padrão específico for selecionado, o agente escolhe uma dica na base de dados. Além das dicas de padrões, listadas anteriormente, ainda existem duas para a utilização/opção de componente ou aviso, que são:

1. JVoicePanel (dica de aviso)

   Sempre que o desenvolvedor colocar um JVoicePanel o Advices exibirá um aviso de uso do componente:

   i. Atenção!!! Esse componente só pode ser usado uma vez por Frame! Ao utilizar o reconhecedor de voz,

cada botão contido no JVoicePanel deve possuir um texto diferente (Propriedade Text) e caso os nomes não sejam atômicos, o click será emulado em um botão randômico.

2. JTextField (dica de componente)

   Se o desenvolvedor utilizar o componente JTextField em sua interface, o agente aciona uma dica de componente com a utilização de teclado virtual.

   i. Se sua aplicação utiliza a modalidade de toque e não possui um teclado, utilize o componente JTextFieldKeyBoard. Assim, o usuário poderá utilizar a tela para preencher o campo um de texto. Tal componente pode também ser utilizado quando a modalidade de voz não for eficiente para a interação, como, na falha no reconhecimento de nomes.

Sempre que uma dica de padrão for exibida no campo *Advices*, um sinal sonoro é disparado, para o desenvolvedor, juntamente com um sinal visual (o cabeçalho do campo com o nome *Advices* fica piscando na cor amarela). Para fazer com que o campo pare de piscar é só clicar nele e caso o usuário queira ver a descrição completa do padrão que foi enviado como dica basta fazer um duplo clique no local onde tem o texto do campo *Advices*.

Quando o usuário escolhe visualizar a definição do padrão uma nova tela, em forma de pop-up, será aberta com o padrão. Todos os outros padrões, que futuramente o usuário deseje conhecer, são agrupados nesse pop-up e separados por abas. A Figura 6.11 ilustra um exemplo do pop-up, no qual o usuário escolhe ler a definição do primeiro padrão indicado pelo agente. A Figura 6.12 apresenta a aba com a definição do segundo padrão, sempre que uma nova definição for exibida, o padrão anterior passa para a aba do lado direito.

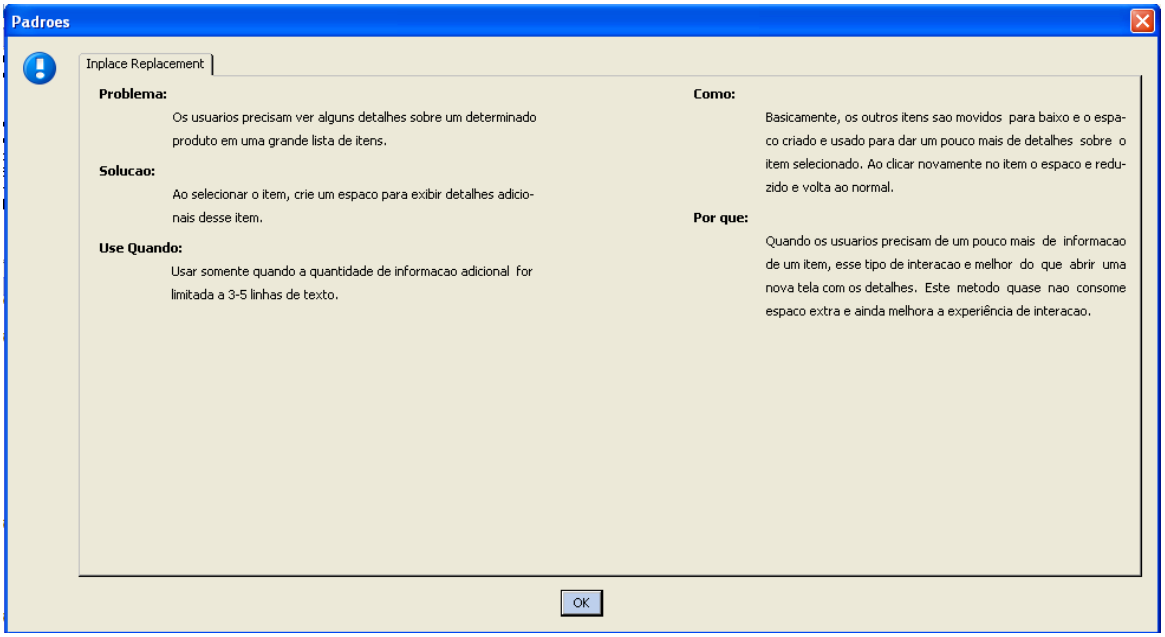


Figura 6.11 Pop-up com a definição do padrão.

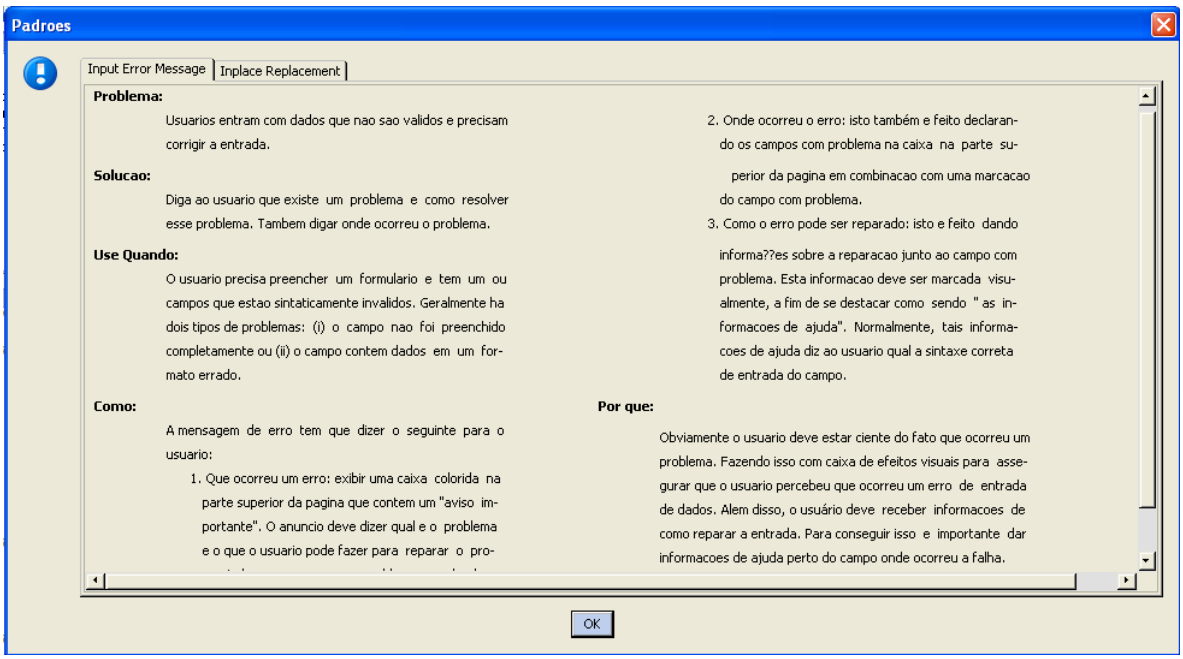


Figura 6.12 Aba com novo padrão.

6.3 Desenvolvimento de interfaces multimodais com o GuiBM

O ambiente do GuiBM tem como referência as IDEs de criação de GUIs para facilitar a interação do desenvolvedor que tenha uma experiência mínima. Assim, a interface do *framework* pode ser mais intuitiva para o designer. Como exemplo de desenvolvimento de interfaces multimodais no GuiBM, veja os passos da criação da interface das Figuras 6.5 e 6.6.

A primeira etapa consiste em inserir um JFrame no campo de GUI Editor, pois sempre é preciso colocar um JFrame para começar o desenvolvimento de qualquer interface. Para que a aplicação funcione com a modalidade de voz, foi inserido um JVoicePanel com a propriedade *Text* preenchida com a palavra “Pedido”, como ilustra a Figura 6.13.

O próximo passo é criar os subpainéis para as opções de pedido: (i) Entradas; (ii) Prato Principal; (iii) Bebidas e (iv) Sobremesa. Para cada subpainel utilizou-se o componente JSubVoicePanel com a propriedade *Text* preenchida com seus nomes correspondentes. As opções de pratos e bebidas contidas nos subpainéis foram

criadas a partir do componente JCheckBox. A criação da interface inteira e a hierarquia de componentes são apresentadas na Figura 6.14.

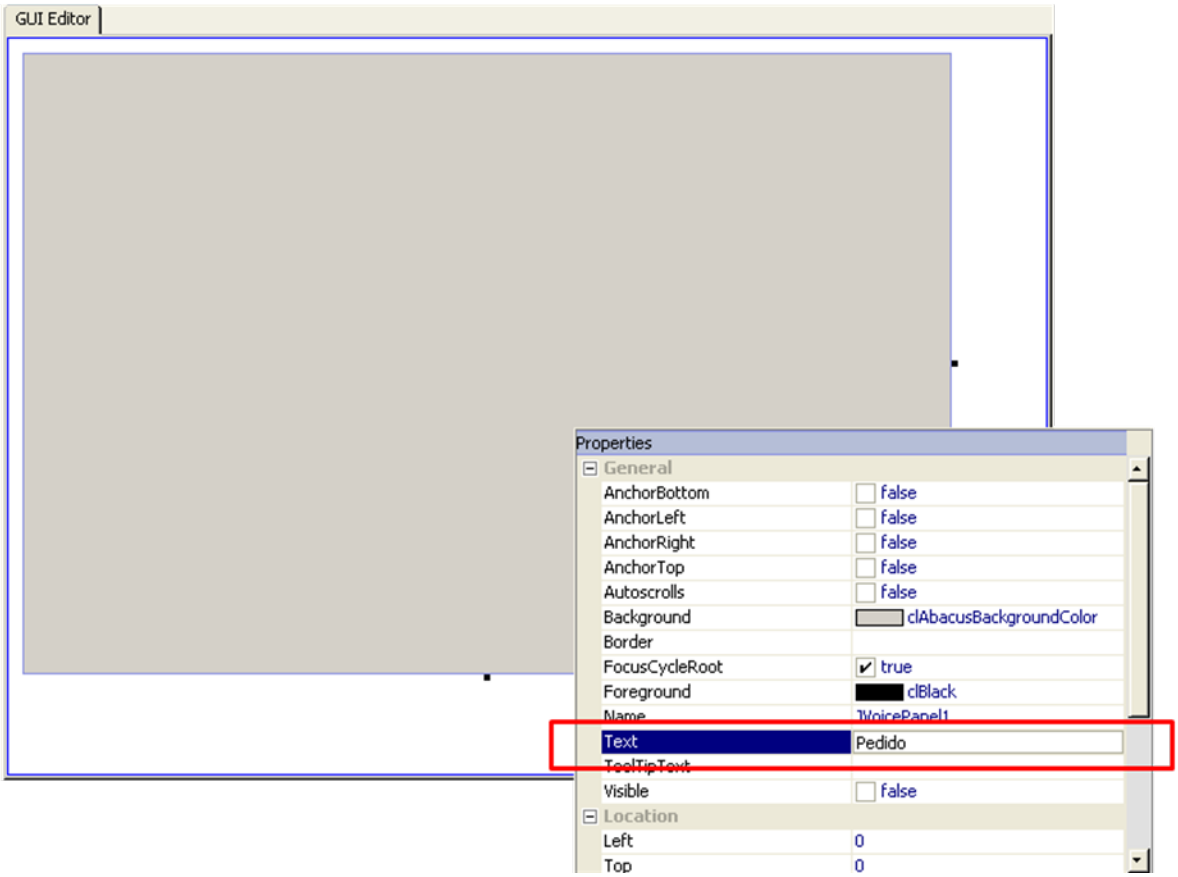


Figura 6.13 JVoicePanel Pedido.

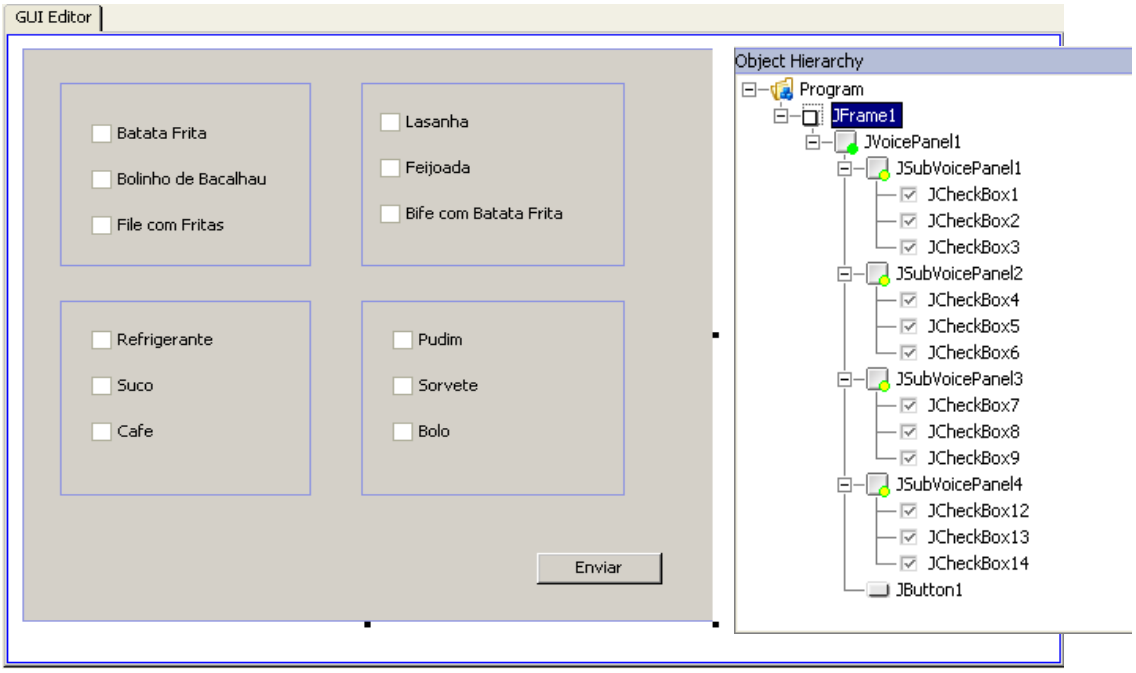


Figura 6.13 Interface da aplicação Restaurante.

À medida que o desenvolvedor for criando sua interface, ele pode verificar como está o funcionamento da tela. Para isso, basta clicar no botão *Render* e a tela da Figura 6.5 é exibida. Após criar a sua interface o desenvolvedor tem a opção de salvar o projeto.

Para gerar o código da interface em Java, o usuário deve compilar o projeto. O arquivo Java será salvo na mesma pasta que o desenvolvedor salvou o projeto. Assim o projetista poderá desenvolver o restante da sua aplicação, como a parte de gerenciamento e de persistência, em outra IDE.

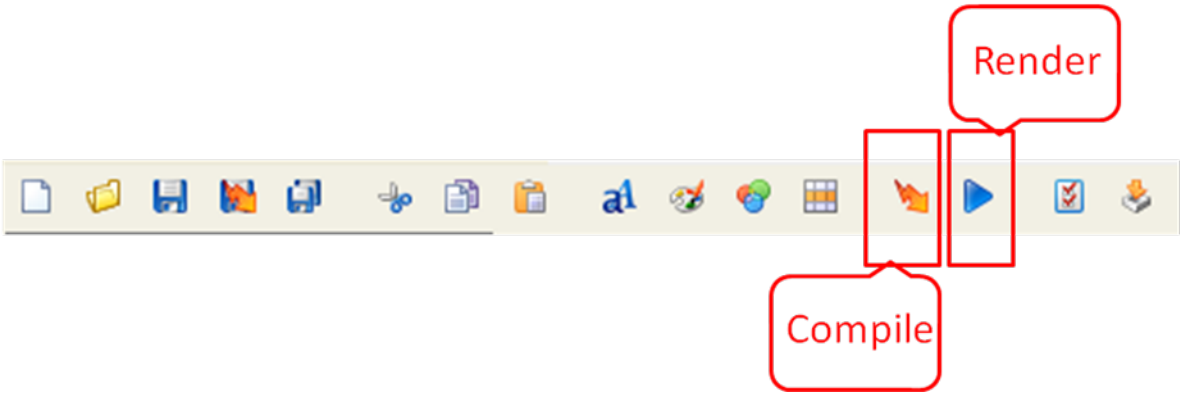


Figura 6.14 Botões Render e Compile.

6.4 Considerações Finais

Este capítulo apresentou o *framework* GuiBuilder Multimodal, que auxilia desenvolvedores na criação de interfaces multimodais. O GuiBM possibilita que o desenvolvimento crie interface no modelo WYSIWYG fazendo com que o framework seja mais intuitivo para o projetista, visto que a maioria dos desenvolvedores tem um certo conhecimento de programação em IDEs como Eclipse e NetBeans. A geração da interface e o código Java, com todas as bibliotecas necessárias, são criados quando o projeto é compilado. O desenvolvedor tem a opção de migrar o código para outra IDE e continuar o desenvolvimento da sua aplicação.

Um estudo de caso foi realizado para avaliar a utilização do GuiBM. O estudo de caso contou com a participação de 9 voluntários convidados divididos em 3 grupos. O próximo capítulo apresenta o estudo de caso e os resultados obtidos.

Capítulo 7

ESTUDO DE CASO

7.1 Considerações Iniciais

A fim de demonstrar a utilização do *framework* GuiBuilder Multimodal, realizou-se um estudo de caso denominado "**Recarga de Celular**", que foi desenvolvido com o apoio do GuiBM. Esse contou com a participação de nove voluntários, todos alunos de mestrado e doutorado da Universidade Federal de São Carlos.

7.1.1 Recarga de Celular

Segundo a Anatel, o número de linhas de celular ativas no Brasil em agosto de 2012 ultrapassava 257 milhões, sendo 81,29% delas com plano pré-pago. Como o nome já diz, nesse plano, o usuário precisa fazer a compra de uma quantidade X de crédito para poder realizar ligações, diferente do plano pós-pago no qual as ligações são contabilizadas e o usuário recebe a conta para pagamento no final do mês.

As vantagens em se ter um celular pré-pago são várias como, por exemplo, não ter um contrato de no mínimo 12 meses com alguma operadora e manter a linha ativa com um custo, muitas vezes, muito mais baixo do que uma linha no plano pós-pago.

Para manter a linha ativa, o usuário deve realizar uma recarga mínima a cada três meses, pelo menos. A realização dessa recarga deve seguir alguns passos, como: (i) escolher a operadora; (ii) escolher um estado da federação; (iii) informar o número do DDD mais o número do celular e confirmar esses dados; (iv) informar o valor da recarga e (v) finalizar a operação com a forma de pagamento. Os valores da recarga variam entre as operadoras, podendo ser de 2 a 100 reais. O usuário pode realizar quantas recargas ele quiser.

7.2 Estudo de Caso: Recarga de Celular

O estudo de caso foi separado em 5 etapas com a participação dos voluntários divididos em 3 grupos, sendo: (i) grupo I, com quatro voluntários; (ii) grupo II com dois participantes e (iii) grupo III com três participantes. A divisão dos três grupos sofreu influência da disponibilidade de horário e afinidade dos participantes. As atividades foram realizadas em momentos diferentes: o primeiro grupo realizou as atividades em um dia diferente dos outros dois grupos (que realizaram-nas em um mesmo dia e horário). Além disso, o primeiro grupo (com mais participantes), teve 2 momentos de contato com o framework: o primeiro em que as dicas de padrões não eram oferecidas e o segundo em que tal facilidade lhes foi disponibilizada.

Na primeira etapa, deveriam aprender sobre as questões envolvidas com o desenvolvimento de interfaces faladas, principalmente sobre sintetizadores e reconhecedores de voz. Uma breve aula foi oferecida aos participantes envolvendo conteúdos como instalação de ferramentas necessárias à utilização do Java Speech e produção de códigos para suporte a interfaces faladas (como sintetizador e reconhecedor de fala). Na segunda etapa, lançou-se um desafio, no qual os participantes foram convidados a desenvolver um botão de interface que fosse *clicável* por voz através do conhecimento adquirido na primeira etapa. Os participantes deveriam realizar a tarefa em um tempo razoável de até 40 minutos.

Após o desafio, na terceira etapa, o *framework* GuiBM foi apresentado aos participantes e, na quarta etapa, solicitou-se que eles criassem uma interface

multimodal para a aplicação de recarga de celular, sem restrição de tempo para a realização da atividade, que consistia nas seguintes tarefas: (i) a escolha da operadora; (ii) a escolha do estado; (iii) o fornecimento dos números do celular e do DDD e (iv) a escolha do valor da recarga.

Por último, os participantes responderam um questionário com 18 questões para avaliar GuiBM em relação a facilidade em se usar o *framework* para criar interfaces com tecnologia multimodal, compreensão dos recursos disponíveis, o grau de conhecimento dos participantes sobre interfaces multimodais e padrões antes do estudo de caso e se o GuiBM contribuiu para o aprendizado de implementação desse tipo de interface.

No questionário, utilizou-se a escala de Likert, bastante conhecida no meio acadêmico devido a sua simplicidade na elaboração das questões e das respostas (Seashore e Katz, 1982). As questões contidas na escala de Likert têm cinco possíveis respostas com diferentes pontuações: Discordo Totalmente (1 ponto), Discordo (2 pontos), Indiferente (3 pontos), Concordo (4 pontos) e Concordo Totalmente (5 pontos).

Além das questões que utilizaram a escala de Likert, também foram inseridas duas questões para verificar o grau de conhecimento dos participantes na área de multimodalidade e padrões de projeto. Tais questões ofereciam respostas possíveis como: Nenhum Conhecimento, Conhecimento Mínimo, Conhecimento Básico e Conhecimento Avançado. O objetivo foi o de traçar um perfil dos participantes. Além disso, permitiu-se que os desenvolvedores pudessem apontar os pontos negativos e positivos do *framework* – além de fornecer sugestões de melhoria – através de quatro questões abertas.

7.2.1 Objetivos

O principal objetivo do estudo de caso foi descobrir se o *framework* GuiBuilder Multimodal e as dicas de padrões eram úteis e de fácil compreensão para que desenvolvedores, com pouco conhecimento em interfaces multimodais, pudessem implementar a interface de alguma aplicação utilizando as modalidades de voz, toque e gesto, diminuindo a quantidade de esforço e aumentando o ganho de conhecimento.

7.2.2 Resultados e Discussão

Com base nas respostas obtidas no questionário (Apêndice A), calculou-se a média das respostas dos participantes para medir a facilidade de utilização do GuiBM bem como se as dicas de padrões foram utilizadas. A tabela 7.1 apresenta as questões bem como a média obtida, para as questões 2-9 e 11-14, nas quais utilizou-se a escala de Likert.

Questões	Média
Q2. Não encontrei dificuldade em desenvolver interfaces com o apoio do GuiBM.	3,8
Q3. Sem o GuiBM o nível de dificuldade em desenvolver uma interface multimodal é maior.	4,8
Q4. O layout do GuiBM simplifica a organização dos componentes e suas propriedades e é compreensível	4,4
Q5. O GuiBM é de difícil utilização.	2,1
Q6. As instruções iniciais são suficientes para entender como o GuiBM funciona.	4
Q7. Foi fácil obter habilidades para desenvolver interfaces multimodais no GuiBM.	4,6
Q8. O tempo e o esforço de desenvolvimento de uma interface multimodal SEM o apoio do GuiBM aumentaria substancialmente.	4,7
Q9. Considero o GuiBM útil na criação de interfaces multimodais.	4,6
Q11. O apoio dos padrões contribui para o desenvolvimento de uma interface melhor.	4

Q12. As dicas de padrões foram claras e úteis	4,2
Q13. As dicas de padrões facilitaram a criação da interface.	4
Q14. Consegui entender e utilizar as informações fornecidas pelos padrões no GuiBM.	4

Tabela 7.1 Resultados do Questionário.

id	Grau	1[3]	2	3	4	5	6	7	8	9	10	11	12	13	14
1	Doutorado 3° SEM	C. A	4	5	4	2	x	5	5	4	C. B.	x	x	x	x
2	Mestrado 3° SEM	S. C.	4	4	5	3	5	5	4	5	C. A.	x	x	x	x
3	Doutorado 3° SEM	C. M.	4	5	5	1	4	4	5	5	C. M.	x	x	x	x
4	Doutorado 3° SEM	C. M.	2	5	4	2	4	4	4	4	C. A.	x	x	x	x
5	Doutorado 6° SEM	S. C.	4	5	4	2	4	4	5	4	C. A.	4	4	4	5
6	Doutorado 2° SEM	C. B.	4	5	4	2	5	5	5	5	C. B.	5	4	4	4
7	Mestrado 2° SEM	C. B.	4	5	5	4	5	5	5	5	C. B.	4	4	4	4
8	Mestrado 4° SEM	C. M.	4	5	4	2	4	5	5	5	C. B.	3	4	3	3
9	Mestrado 2° SEM	C. B.	5	5	5	1	5	5	5	5	C. B.	4	5	5	4

Tabela 7.2 Respostas dos participantes no questionário.

[3] (C. A.) Conhecimento Avançando; (C. M.) Conhecimento Mínino; (C. B.) Conhecimento Básico; (S. C.) Sem Conhecimento.

De acordo com os resultados obtidos da Tabela 7.1, a grande maioria dos participantes concordou que a utilização do GuiBM facilitou a criação de interfaces multimodais e que o *framework* não é difícil de ser usado.

A seguir, encontram-se alguns relatos de participantes *"**Interface simples e Intuitiva**"*, "***Ambiente de desenvolvimento clean***", "***agilidade e facilidade na criação de interfaces multimodais***" e "***O ambiente facilitou consideravelmente a criação de uma interface com multimodalidade, se comparado ao desenvolvimento sem o apoio do ambiente***".

As dicas de padrões também foram relevantes para alguns participantes, como comprovado nas seguintes respostas: "***dicas de padrões em momentos oportunos***" e "***possui dicas úteis e relevantes para o desenvolvimento***".

A tabela 7.1 apresenta as respostas dos participantes para cada questão. Na parte superior da tabela encontram-se o id do participante, o seu grau de escolaridade e as questões de 1 a 14 do questionário. Os participantes foram divididos em três grupos: (i) grupo azul, com os quatro voluntários; (ii) grupo vermelho, com dois voluntários e (iii) grupo verde, com três voluntários.

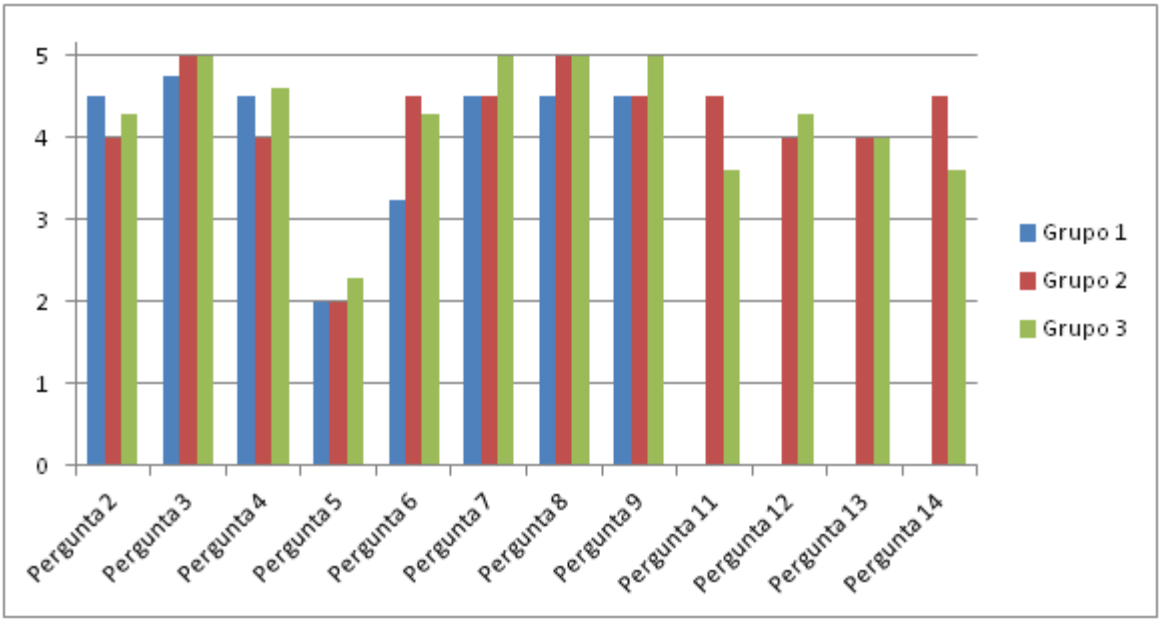


Figura 7.1 Resultado do Questionário do GuiBM com escala de Likert.

A partir das análises das respostas no questionário, o primeiro grupo foi considerado o que mais tinha conhecimento sobre interfaces multimodais antes de realizar o estudo de caso: um participante tinha conhecimento avançado; dois tinham conhecimento mínimo e apenas um sem conhecimento. Além disso, dois tinham conhecimentos avançados sobre padrões de projetos, um tinha conhecimento moderado e outro conhecimento mínimo. No desafio para criar um botão que fosse acionado (*clicável*) por voz, este grupo conseguiu realizar a tarefa em 35 minutos.

O segundo grupo pode ser considerado de conhecimento mediano: um participante não tinha conhecimento sobre interfaces multimodais mas tinha conhecimento avançado sobre padrões de projetos enquanto o outro tinha conhecimento básico de interfaces multimodais e padrões. Essa equipe utilizou o GuiBM com dicas de padrões e demonstrou ser o grupo que mais as usou. Um dos participantes mencionou nas questões abertas que *"**As dicas de padrões em momentos oportunos também são um aspecto positivo**"*. Essa equipe levou 50 minutos para criar o botão acionado por voz.

O terceiro grupo foi considerado, a partir das análises dos dados, o grupo com o menor grau de conhecimento: todos os participantes tinham entre conhecimento básico e mínimo sobre interfaces multimodais e padrões de projetos. Esta equipe não conseguiu realizar a tarefa de criar um botão *clicável* por voz.

Na etapa do experimento em que todos os grupos utilizaram o GuiBM para desenvolver a interface da recarga de celular (tarefa muito mais complexa que a do primeiro desafio), o tempo estimado foi entre 35 minutos a 1 hora aproximadamente. Todas as equipes conseguiram realizar a atividade com sucesso e testaram suas interfaces utilizando as modalidades disponíveis no GuiBM. Para os testes com a modalidade de toque, uma tela com tecnologia resistiva foi utilizada e na modalidade de gestos, luvas nas cores vermelha, verde-limão e verde-escuro foram disponibilizadas.

Após o preenchimento do questionário, os participantes foram entrevistados e puderam opinar sobre os experimentos e sobre o GuiBM. Detectou-se que a atividade da segunda etapa do estudo de caso (tornar um botão clicável por voz) foi considerada a mais complexa por todos os participantes. Também pode-se constatar que o GuiBM auxiliou os desenvolvedores a criar as interfaces diminuindo

consideravelmente o tempo de desenvolvimento e, de maneira geral, o esforço do projetista.

Também foi possível concluir que a terceira equipe (que não conseguiu realizar a atividade do botão clicável) foi a que mais sentiu seus benefícios, pois seus participantes foram os que mais bem avaliaram o GuiBM nos quesitos: a) facilidade de uso e b) agilidade em construir interfaces multimodais com ele.

Como mencionado no capítulo anterior, sempre que o desenvolvedor quiser testar a interface em criação, basta clicar no botão *Render*. Foi durante os testes das interfaces, criadas pelos participantes, que dúvidas e erros surgiram. As dúvidas existentes foram sobre quais componentes funcionavam para a modalidade de voz, onde foram salvos os códigos gerados em Java, o tempo gasto no reconhecimento da voz para algum componente, incluindo casos para acionar um subpainel de voz ou a demora em reconhecer um gesto, o que gerou a necessidade do usuário repetir o comando mais de uma vez.

7.3 Considerações Finais

Neste capítulo, apresentou-se o estudo de caso realizado no GuiBM, utilizado para descobrir se sua utilização realmente auxilia projetistas a desenvolver interfaces multimodais de forma mais rápida, com menos esforço e com dicas que fossem significativas para a aplicação que ele está desenvolvendo.

Dentro do processo de avaliação estipulado, pode-se notar que a apresentação de dicas de padrão contribuíram para o desenvolvimento de interfaces melhores. Apesar de ter sido um experimento realizado apenas com um grupo (o primeiro), pode-se notar uma significativa melhora no projeto de interface, observado e comentado pelos próprios voluntários, quando foi utilizado o GuiBM com dicas.

Todos os participantes do estudo de caso foram convidados e suas participações foram voluntárias. Ao final das atividades, todos preencheram um questionário com perguntas de nível de conhecimento sobre multimodalidade e padrões bem como perguntas sobre a utilização do GuiBM e as dicas fornecidas pelo *framework*.

No próximo capítulo, encontram-se as conclusões, as limitações encontradas pelos participantes durante o desenvolvimento das interfaces multimodais, sugestões de melhoria e os trabalhos futuros.

Capítulo 8

CONCLUSÕES

8.1 Contribuições e síntese dos principais resultados

Neste trabalho foi apresentado o *framework* GuiBuilder Multimodal (ou GuiBM), que tem como objetivo auxiliar projetistas a desenvolver interfaces multimodais com as modalidades de gesto, toque e fala, com o apoio dos conceitos de padrões e de geração de código Java, para que o desenvolvedor possa migrar o código gerado para outras IDEs de programação e finalizar a implementação da sua aplicação.

A fim de validar o *framework* GuiBM foi realizado um estudo de caso, que contou com a participação de 9 usuários; todos alunos de pós-graduação do Departamento de Computação (DC) da Universidade Federal de São Carlos (UFSCar). Infelizmente, até a data da escrita desta dissertação não foram realizados novos estudos de casos com mais participantes, porém a autora ainda tem o objetivo de fazer o experimento com alunos de graduação e refazê-lo com os quatro participantes do primeiro grupo. O objetivo de refazer o estudo com o grupo é verificar se, para os 4 participantes, a experiência em desenvolver uma interface multimodal com o auxílio dos padrões melhoraria a interface.

Através dos resultados obtidos pela análise do questionário e das pequenas entrevistas realizadas com cada participante ao final do estudo de caso, é possível concluir que a tendência dos resultados é positiva, ou seja, que o *framework* GuiBuilder Multimodal cumpriu o objetivo de ajudar o desenvolvedor a implementar

interfaces com modalidade de voz, toque e gesto e as dicas de padrões tiveram importância na elaboração da interface.

Os participantes puderam perceber a dificuldade em construir uma interface sem uma aplicação (ou *framework*) que ofereça apoio ao projetista quando lhes foi passado o desafio de construir um botão que fosse clicável por voz. O objetivo desse desafio foi o de demonstrar, para apenas uma das modalidades disponíveis, como é complexo desenvolver interfaces multimodais e que tal desenvolvimento requer um grande nível de conhecimento.

Apesar de cumprir com o objetivo proposto, durante a realização do estudo de caso foram identificadas limitações e novas ideias, advindas dos nove participantes, que podem ser implementadas no GuiBM e melhorar, assim, o *framework*.

8.2 Limitações

O *framework* GuiBuilder Multimodal foi dividido em três módulos: planejamento, supervisão (ou inteligência) e código, como é possível notar na arquitetura do GuiBM apresentada na a Figura 8.1. O módulo de planejamento foi desenvolvido baseado na plataforma da Abacus GUI Builder Java que possibilitou aos designers realizar suas atividades em uma interface muito similar a de outras IDEs de construção de GUIs.

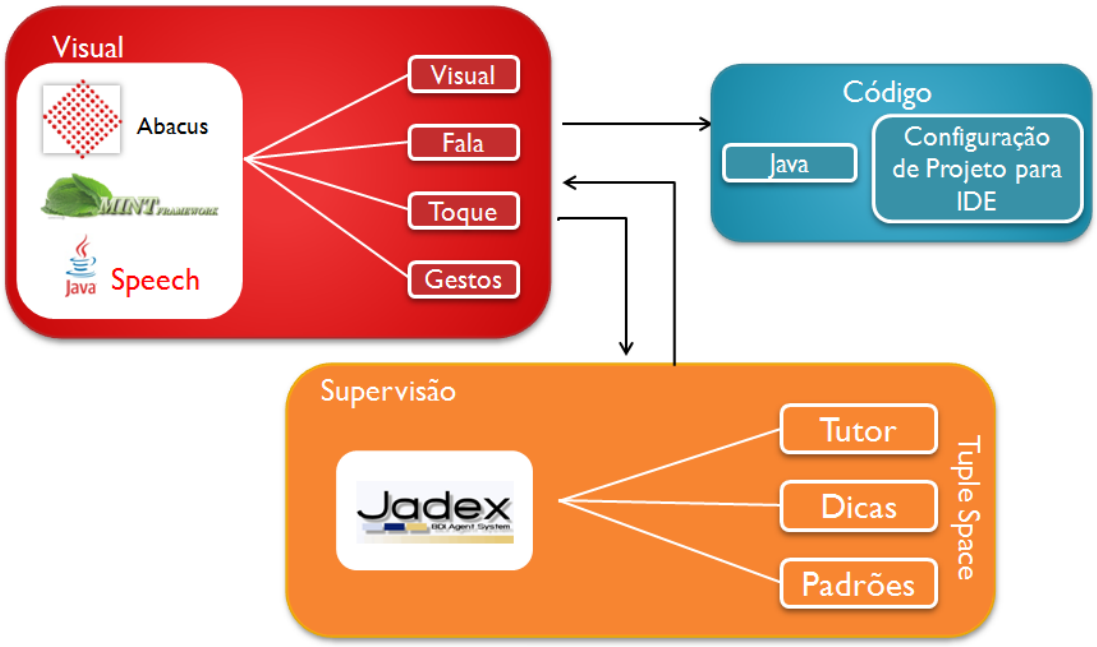


Figura 8.1 Arquitetura GuiBM.

Porém, todas as limitações presentes na plataforma da Abacus foram transferidas para o *framework* GuiBM. Por exemplo, em outras IDEs (como o NetBeans), para a elaboração de interfaces, o desenvolvedor não precisa clicar enter para finalizar a tarefa quando está inserindo um texto no campo de propriedades dos componentes, enquanto que no Abacus isso é necessário. Caso o projetista não cumpra essa restrição, os dados são perdidos e ele é obrigado a reescrever. Isso incomodou os participantes, visto que para que suas interfaces tivessem a modalidade de voz, era preciso preencher o campo *text* com o nome que seria a palavra de reconhecimento do componente.

Ainda para a modalidade de reconhecimento de voz, os participantes notaram que palavras acentuadas não eram reconhecidas e isso acontecia na transferência do texto contido na propriedade *text* para a gramática. Através de algumas adequações no GuiBM é possível corrigir tais falhas.

Para a modalidade de gesto, os participantes questionaram a possibilidade do GuiBM oferecer suporte para a criação de novos gestos. De fato, a quantidade de gestos que podem ser reconhecidos pelo framework no momento é pequena (como mencionado, atualmente, o GuiBM utiliza o Mint como framework de suporte e reconhece apenas quatro gestos com os seguintes comandos: voltar, avançar, selecionar e confirmar). Com alguns ajustes, poder-se-ia utilizar um conjunto um pouco maior de gestos. Mas ainda assim seriam gestos ensinados previamente ao sistema de reconhecimento de gestos. O problema de permitir que o usuário crie seus próprios gestos será objeto de novas pesquisas para as versões futuras

8.3 Trabalhos Futuros

Como trabalho futuro, uma das principais atividades é suprir as restrições impostas pela plataforma Abacus, assim, textos inseridos na gramática não sofrerão novas anomalias quando o desenvolvedor utilizar palavras acentuadas. Além disso, é necessário solucionar o problema dos atalhos do teclado, como: delete, Ctrl-C, Ctrl-V e Ctrl-Z.

Oferecer ao desenvolvedor a possibilidade de criar novos gestos que possam ser mais úteis para a interface que está sendo elaborada. Assim, é possível retirar a restrição do GuiBM que possui 4 gestos e oferecer liberdade ao usuário de criar novos. Porém, como explicado na sessão anterior, os novos gestos devem ser ensinados ao sistema e tal ação demanda repetição exaustiva de tais gestos.

Aperfeiçoar o sistema de agentes inteligentes com os padrões fornecidos pelo *framework*, com um estudo preliminar abordando os dois temas, de tal forma que o agente possa entender melhor o objetivo e comportamento do projetista no momento do desenvolvimento de sua interface. Dessa maneira, as dicas podem ser mais contextualizadas e não será necessária a utilização de dicas aleatórias.

Outro ponto que merece atenção é a formatação da área "*Advices"* com as dicas de padrões. Alguns participantes do estudo mencionaram que o texto contido era útil, porém, extenso demais e que a utilização de uma imagem, como um boneco-tutor, seria mais chamativa para o desenvolvedor do que o beep e o cabeçalho piscando em amarelo. Apesar da sugestão todos os participantes concordaram que a retirada do campo *Advices* e a inserção de um pop-up para cada dica não seria uma boa solução e possivelmente desestimularia o desenvolvedor a utilizar o *framework*, já que, pelo menos a cada cinco minutos, o projetista seria interrompido por um pop-up com dicas.

Para melhoria das análises dos resultados, obtido através das atividades dos participantes, é importante incorporar um sistema de logs. Onde tanto as ações do designer quanto as reações dos agentes podem ser analisadas, sendo assim novas conclusões podem ser exploradas e atribuídas na pesquisa realizada pelo GuiBM.

REFERÊNCIAS BIBLIOGRÁFICAS

ABACUS (2012) — ABACUS, disponível em: http://sourceforge.net/projects/abaguibuilder/ último acesso em: setembro de 2012.

ALEXANDER (1964) — ALEXANDER, C. *Notes on the Synthesis of Form.* Harvard University Press, USA, 1964. ISBN 0-674-62751-2.

BAGGIA et. Al. (2007) — BAGGIA, P. L; CARTER, J; DAHL, D.A.; MCCOBB, G. *EMMA: Extensible MultiModal Annotation markup language*, W3C Working Draft, 9 April 2007. http://www.w3.org/TR/emma/.

BECK & CUNNINGHAM (1987) — BECK, K.; CUNNINGHAM, W. *Using pattern languages for object-oriented programs*. Technical report, Tektronix, Inc., 1987. Presented at the OOPSLA-87 Workshop on Specification and Design for Object-Oriented Programming.

BERNSEN (2008) — BERNSEN, N. O.; *Multimodality Theory*. D. Tzovaras (Ed.) Multimodal User Interfaces. Signals and Communication Technology. DOI 10.1007/978-3-540-78345-9, p. 5-30, © Springer 2008.

BOLT (1980) — BOLT, R. A.; *"Put-that-there": Voice and Gesture at the Graphics Interface*. Architecture Machine Group Massachusetts Institute of Technology, Cambridge, Massachusetts, 1980.

BOSCH et. Al. (1999) — BOSCH, J., MOLIN P., MATTSSON M.; BENGTSSON P.; Fayad M. Framework problem and experiences in M. Fayad, R. Johnson, D. Schmidt. Building Application Frameworks: Object-

Oriented Foundations of Framework Design, John Willey and Sons, p. 55–82, 1999.

BOUCHET et. Al. (2004) BOUCHT, J., NIGAY, L., GANILLE, T. *ICARE Software Components for Rapidly Developing Multimodal Interfaces*. In: Conference Proceedings of ICMI'2004, State College, Pennsylvania, USA, October 2004, ACM Press, pp. 251—258 (2004).

BOURGUET (2002) BOUGUET, M. L. *A Toolkit for Creating and Testing Multimodal Interface* Designs. In: companion proceedings of UIST'02, Paris, Oct. 2002, pp. 29—30 (2002).

BOVET (2004) BOVET, J. *Visual Automata Simulator, a tool for simulating automata and Turing machines*. University of San Francisco, 2004. Disponível em: http://www.cs.usfca.edu/~jbovet/vas.html.

BUXTON (2007) BUXTON, B. *Multi-Touch Systems that I Have Known and Loved* Microsoft Research. January 12, 2007. Disponível em: http://www.billbuxton.com/multitouchOverview.html. Último acesso em maio de 2011.

ÇETIN (2009) ÇETIN, G. *Multitouch Technologies*. Version 1.0, 2009 NUI Group Autors, 1st edition [Community Release]: May 2009. Dísponivel em: http://nuicode.com/projects/wiki-book/files.

CHEYER et. Al. (1998) CHEYER, A. ;JULIA, L.; MARTIN, J. C. *A Unified Framework for Constructiong Multimodal Experiments and Applications.* Conference on Cooperative Multimodal Communication (CMC98), 1998, pg 63-69.

CORAM & LEE (1996) CORAM, T.; LEE, J. *Experiences – A Pattern Language for User Interface Design*. Disponível em:

http://www.maplefish.com/todd/papers/Experiences.html#ExploratableInterface Ultimo acesso: Fev de 2012.

DUARTE & CARRIÇO (2006) DUARTE, C.; CARRIÇO, L. *A Conceptual Framework for Developing Adaptative Multimodal Applications*. IUI'06, January 29–February 1, 2006, Sydney, Australia.Copyright 2006 ACM 1595932879/06/0001

DUMAS et. Al. (2009) DUMAR, B.; LALANNE, D.; OVIATT, S. *Multimodal Interfaces: A Survey of Principles, Models and Framework*. Lecture Notes In Computer Science, Vol. 5440. Springer-Verlag, Berlin, Heidelberg 3-26. DOI = 10.1007/978-3-642-00437-7_1, 2009.

ECKEL (2003) ECKEL, B. *Thingkin in Patterns: Problem-Solving Tecniques using Java*. Revision 0.9 0.9 ed. President, USA: MindView, 2003.

ECLIPSE ECLIPSE SWING METAMORPHOSIS. Disponível em: http://www.jgoodies.com

FAYAD et. Al. (1999) FAYAD, W. E.; SCHMIDT, D. C.; JOHNSON, R. E. *Building Application Frameworks: object-oriented foundations of framework design*. New York, USA: John Wiley & Sons, 1999.

FEUERSTACK & PIZZOLATO FEUERSTACK, S.; PIZZOLATO, E. *Building Multimodal Interfaces out of Executable, Model-based Interactors and Mappings*; HCI International 2011; 14th International Conference on Human-Computer Interaction; J.A. Jacko (Ed.): Human-Computer Interaction, Part I, HCII 2011, LNCS 6761, pp. 221—228. Springer, Heidelberg (2011), 9-14 July 2011, Hilton Orlando Bonnet Creek, Orlando, Florida, USA.

FIORINI (2001) FIORINI, S. T. *Arquitetura para Reutilização de Processo de Software*. Tese de doutorado, Pontifícia Universidade Católica

do Rio de Janeiro, Rio de Janeiro, 2001.

FLIPPO et. Al. (2003) FLIPPO, F.; KREBS, A.; MARSIC, I. *A Framework for Rapid Development of Multimodal Interfaces. ICMI'03,* November 5–7, 2003, Vancouver, British Columbia, Canada. Copyright 2003 ACM 1581136218/03/0011

GAMMA et. 131L. (2005) GAMMA, E.; HELM, R.; JOHNSON, R.; VLISSIDES, J. *Padrões de Projeto: Soluções Reutilizáveis de Software Orientado a Objetos.* Editora Bookman, Porto Alegre, 2005.

GODET-BAR et. Al. (2006) GODET-BAR, G.; DUPUY-CHESSA, S.; NIGAY, L. *Towards a System of Patterns for the Design of Multimodal Interfaces*. Proceedings of the Sixth International Conference on Computer-Aided Design of User Interfaces CADUI '06 (6-8 June 2006, Bucharest, Romania) p. 27-40.

GODET-BAR et. Al. (2007) GODET-BAR, G., DUPUY-CHESSA, S. NIGAY, L. *Towards a Sistem of Patterns for the Design of Multimodal Interfaces*. Computer-AIDED Design of User Interface V, 2007, pg 27-40, DOI: 10.1007/978-4020-5820-2_3, Springer.

GOUBRAN e WOOD (1996) GOUBRAN, R. A.; WOOD, C. *Building an Application Framework for Speech and Pen Input Integration in Multimodal Learning Interfaces.* In *Proceedings of the Acoustics, Speech, and Signal Processing, 1996. On Conference Proceedings., 1996 IEEE International Conference – Volume 06* (ICASSP '96), Vol. 6. IEEE Computer Society, Washington, DC, USA, 3545-3548. DOI=10.1109/ICASSP.1996.550794 http://dx.doi.org/10.1109/ICASSP.1996.550794.

HEIKKINEN et. 131L. (2009) HEIKKINEN, J.; OLSSON, T.; MATTILA, K. V. V. *Expectations for User Experience in Haptic Communication with Mobile*

Devices. MobileHCI 09, Proceedings of the 11th International Conference on Human-Computer Interaction with Mobile Devices and Services. ACM, New York, NY, USA, 2009.

HONKALA, & POHJA (2006) HONKALA, M.; POHJA, M. 2006. *Multimodal interaction with XFORMS*. In Proceedings of the 6th international Conference on Web Engineering (Palo Alto, California, USA, July 11 – 14, 2006). ICWE '06. ACM Press, New York, NY, 201-208. DOI= http://doi.acm.org/10.1145/1145581.1145624.

INÁCIO (2007) INÁCIO, V. R. Jr. *Um Framework para Desenvolvimento de Interfaces Multimodais em Aplicações de Computação Ubíqua.* Dissertação de Mestrado, Instituto de Ciências Matemáticas e de Computação – ICMC-USP, 2007.

JACOB et. Al. (1999) JACOB, R. J. K.; DELIGIANNIDIS, L.; MORRISON, S. *A Software Model and Specification Language for Non-WIMP User Interfaces*. ACM Transactions on Computer-Human Interaction, Vol. 6, No. 1, March 1999, Pages 1–46.

JADEX (2012) JADEX BDI Agent System, disponível: http://jadex-agents.informatik.uni-hamburg.de/xwiki/bin/view/About/Features Ultimo acesso: Agosto de 2012.

JENNINGS & WOOLDRIDGE (2001) JENNINGS, N. R.; WOOLDRIDGE, M. J. *Agent-Oriented Software Engineering* - in Handbook of Agent Technology (ed. J. Bradshaw) AAAI/MIT Press, 2001.

JENNINGS (1995) JENNINGS, N. R. *Agents Software*. Proceedings UNICOM Seminar on Agent Software, Londres, UK, p. 12-27, 1995.

JOHNSON & FOOTE (1988) JOHNSON, R. e FOOTE, B. *Designing Reusable Classes.* Journal of Object-Oriented Programming – JOOP, 1(2):22-35,

1988.

JOHNSON (1992) JOHNSON, R. E. *Documenting Frameworks Using Patterns*. In: Conference on object-oriented programming systems and Applications – OOPSLA, 1992, Vancouver, USA. Proceedings. ACM Press, 1992, v.27, p.63.76.

JUNIT FRAMEWORK JUNIT FRAMEWORK. Disponível em: http://www.junit.org/

KINECT (2011) KINECT. Disponível em: http://www.xbox.com/pt-BR/Kinect/GetStarted. Último acesso: janeiro de 2011.

KRAHNSTOEVER et. Al. (2002) KRAHNSTOEVER, N.; KETTEBEKOV, S.; YEASIN, M.; SHARMA, R. *A Real-Time Framework for Natural Multimodal Interaction with Large Screen Display.* Proceedings of the Fourth IEEE International Conference on Multimodal Interfaces (ICMI'02). 0-7695-1834-6/02 $17.00 © 2002 IEEE.

LARMAN (2007) LARMAN, C. 2007. *Utilizando UML e Padrões: Uma Introdução à Análise e ao Projeto Orientados a Objetos e ao Desenvolvimento Iterativo*.

MAES (1994) MAES, P. *Artificial Life IV: Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems*, Rodney Brooks & Pattie Maes, MIT Press, 1994, ISBN 0-262-52190-3.

MARSIC (1999) MARSIC, I. *DISCIPLE: A Framework for Multimodal collaboration in Heterogeneous Environments*. *ACM Comput. Surv.* 31, 2es, Article 4 (June 1999). DOI=10.1145/323216.323225 http://doi.acm.org/10.1145/323216.323225.

MARTIN (1998) MARTIN, J. C. *TYCOON: Theoretical Framework and Software Tools for Multimodal Interfaces*. Intelligence and Multimodality in Multimedia interfaces, 1998.

MINSKY (1986) MINSKY, M. *The Society of Mind*. New York: Simon & Schuster, Touchstone Book UNB, 1986. 339 p. ISBN 0-671-65713-5

MINSKY (2006) MINSKY, Marvin. *The Emotion Machine: Commonsense Thinking, Artificial Intelligence, and the Future of the Human Mind*. [S.l.]: Simon & Schuster, 2006. 400 p. ISBN 0-743-27663-9.

MINT (2012) MINT, disponível em: http://www.multi-access.de/ último acesso: Setembro de 2012.

MISTRY & MAES (2009) MISTRY, P.; MAES, P. *SixthSense: A Wearable Gestural Interface*. ACM SIGGRAPH ASIA 2009, ACM New York, NY, USA ©2009. DOI 10.1145/1667146.1667160.

MISTRY et. Al. (2009) MISTRY, P., MAES, P. AND CHANG, L. 2009. *WUW – Wear Ur World – A Wearable Gestural Interface*. In the CHI '09 extended abstracts on Human factors in computing systems. Boston,USA.

MONTERO et. al. (2002) MONTERO, F., LOZANO, M., GONZÁLES, P. and RAMOS, I. (2002) "A First Approach To Design Web Sites By Using Patterns", Proceedings of VikingPLoP Conference, 2002.

NAWANA (1996) NWANA, H. S. *Software Agents: An Overview*. *Knowledge Engineering Review*, Vol. **11**, No 3, pp.1-40, Sept 1996. © Cambridge University Press, 1996. Disponível em: http://agents.umbc.edu/introduction/ao/5.shtml Último acesso: Fevereiro de 2012.

NORMAN (1988) NORMAN, D. A. *The Design of Everyday Things*, Basic Books, 1988. ISBN 85-325-2083-9.

NUANCE — NUANCE, disponível em: http://www.nuance.com/index.htm último acesso em: setembro de 2012.

OPENCV — OPENCV. Disponível em: http://www.intel.com/technology/computing/opencv/

OVIATT (1999) — OVIATT, S., *Ten Myths of Multimodal Interaction*, Communications of the ACM, Vol. 42 No. 11, 1999, pp. 74-81.

OVIATT et. Al. (2000) — OVIATT, S.; COHEN, P.; WU, L.; VERGO, J.; DUNCAN, L.; SUHM, B.; BERS, J.; HOLZMAN, T.; LANDAY, J.; LARSON, J.; FERRO, D. *Designing the User Interface for Multimodal Speech and Pen-Based Gesture Applications: State-of-the-Art Systems and Future Research Directions*. Human-Computer Interaction, 2000, Volume 15, pp. 263-332. Copyright © 2000, Lawrence Erlbaum Associates, Inc.

OVIATT et. Al. (2005) — OVIATT, S., COULSTON, R., and Lunsford, R., *When do we Interact Multimodally? Cognitive Load and Multimodal Communication Patterns*, in Proc. Of 6th ACM Int. Conf. on Multimodal Interfaces ICMI'2004 (State College, 13-15 October 2005), ACM Press, New York, 2004, pp. 129-136.

PIZZOLATO et. Al. (2010) — PIZZOLATO, E. B; ANJO, M. dos S.; PEDROSO, G. C. *Automatic Recognition of Finger Spelling for LIBRAS based on a Two-Layer Architecture*. Proceedings of the 25th Annual ACM Symposium on Applied Computing, pg. 969 a 973, 2010.

PLAYSTATION MOVE (2011) — PLAYSTATION MOVE. Disponível em: http://us.playstation.com/os3/playstation-move/. Ultimo acesso: janeiro de 2011.

PREE (1999) PREE, W. Hot-spot-driven development in M. FAYAD, R. JOHNSON, D. SCHMIDT. Building Application Frameworks: Object-Oriented Foundations of Framework Design, John Willey and Sons, p. 379–393, 1999.

RATZKA & WOLFF (2006) RATZKA, A.; WOLFF, C.; *A Pattern-Based Methodology for Multimodal Interaction Design*. Petr Sojka, Ivan Kopeˇcek and Karel Pala (Eds.): TSD 2006, LNAI 4188, pp. 677–686, 2006. © Springer-Verlag Berlin Heidelberg 2006.

RATZKA (2006) RATZKA, A.; *Combining Modality Theory and Context Models*. PIT 2006, LNAI 4021, pp. 141 – 151, 2006. © Springer-Verlag Berlin Heidelberg 2006.

RATZKA (2007b) RATZKA, A. *Design Patterns in the Context of Multi-modal Interaction*. In: Proceedings of the 6th Nordic Conference on Pattern Languages of Programs 2007 VikingPLoP 2007 (to appear, 2007b).

RATZKA (2008) RATZKA, A.; *Steps in Identifying Interaction Design Patterns for Multimodal systems.* P. Forbrig and F. Paternò (Eds.): HCSE/TAMODIA 2008, LNCS 5247, pp. 58–71, 2008. © IFIP International Federation for Information Processing 2008.

RATZKA (2008) RATZKA, A. *Patterns for Robust and Flexible Multimodal Interaction*. EuroPLop 2008, 13th Annual European Conference on Pattern Languages of Programming, Irsee, Germany, July 9-13, 2008.

ROBERTS & JOHNSON (1998) ROBERTS, D.; JOHNSON, R. Evolving frameworks: A pattern language for developing object-oriented frameworks in Martin R.C., Riehle D., Buschmann F. Pattern Languages of Program Design 3, Addison-Wesley, p. 471–486, 1998.

RUSSELL & NORVIG (2004) RUSSELL, S; NORVIG, P. *Inteligência Artificial*. Elsevier Editora Ltda, 2° Edição, 2004. ISBN 85-352-1177-2.

SALT (2007) SALT 2007. *Speech Application Language Tag (SALT) Forum Website*. http://www.saltforum.com

SAUVÊ (2004) SAUVÊ, J. P. *Frameworks.* Notas de Aula. Paraíba: Universidade Federal da Paraíba, 2004, disponível em: http://www.dsc.ufcg.edu.br/~jacques/cursos/map/html/map1.htm. Acesso em: fev. 2011.

SCHMID (2002) SCHMID, H. A. Systematic framework design by generalization. Communications of the ACM, v. 40, n. 10, 2002.

SCHMIDT & BUSCHMANN (2003) SCHMIDT, D. C. AND BUSCHMANN, F. 2003. *Patterns, Frameworks, and Middleware: Their Synergistic Relationships*. In *ICSE ´*03: *Proceedings of the 25th International Conference on Software Engineering,* pages 694-704, Washington, DC, USA. IEEE Computer Society.

SELFRIDGE (1958) SELFRIDGE, O. G. *Pandemonium: A Paradigm for Learning*. Proceedings of a Symposium held at the National Physical Laboratoy, on 24th – 27th November 1958.

SERRANO et. Al. (2008) SERRANO, M.; NIGAY, L.; LAWSON, J. L.; RAMSAY, A.; SMITH, R. M.; DENEF, S.; *The OpenInterface Framework: A Tool for Multimodal Interaction*. In CHI ‘08 extended abstracts on Human factors in computing systems (CHI EA ‘08). ACM, New York, NY, USA, 3501-3506. DOI=10.1145/1358628.1358881.

SOMMERVILLE (2001) SOMMERVILLE I. “Software Engineering. Pearson Education Limited”. Addison Wesley, England, 2001.

SUHM et. Al. (2001) SUHM, B.; MYERS, B.; WAIBEL, A. *Multimodal Error Correction for Speech User Interfaces*. ACM Transactions on Computer-Human Interaction, Vol. 8, No. 1, March 2001, Pages 60–98.

SZYPERSKI (2000) SZYPERSKI, C. *Component Software: Beyond Object-Oriented Programming*. ACM Press/Addison-Wesley, New York, 2000.

TALARICO (2007) TALARICO, A. N.; *Uma abordagem para o desenvolvimento de interfaces multimodais Web*, proposta de doutorada na Universidade do Estado de São Paulo – USP 2007.

TALARICO (2011) TALARICO, A. N. *Uma Abordagem para Projeto de Aplicações com Interação Multimodal da Web*. Tese de Doutorado, Instituto de Ciências Matemáticas e de Computação – ICMC-USP, 2011.

TIDWELL (1998) TIDWELL, J. *Commom Ground: A Pattern Language for Human-Computer Interface Design*. Disponível em: http://www.mit.edu/~jtidwell/interaction_patterns.html Ultimo acesso: Fevereiro de 2012.

VAN DAM (1997) VAN DAM, A.*; Post-WIMP User Interfaces*. Communications of the ACM, Vol. 40, No. 2, 1997.

VOICEXML (2004) *Voice Extensible Markup Language (VoiceXML) Version 2.0*. W3C Recommendation, 16 March 2004. Disponível em http://www.w3.org/TR/2004/REC-voicexml20-20040316/. Ultimo acesso em dezembro de 2011.

WELIE & TRAETTERBERG (2000) WELIE, M. V.; TRAETTERBERG, H. *Interaction Patterns in User Interfaces*. Proc. Seventh Pattern Languages of Programs Confer-ence: PloP 2000. Monticello, Illinois, August 13-16, 2000. Disponível em http://www.welie.com/papers/PloP2k-Welie.pdf

WXWIDGETS WXWIDGETS. Disponível em: http://www.wxwidgets.org/

Apêndice A

QUESTIONÁRIO GUIBM

Experimentação GuiBM

Questionário

Nome:_____ Idade:_____

Instituição:_____ Período:_____

1- Qual seu grau de conhecimento sobre as interfaces multimodais ANTES do experimento:
( ) Nenhum conhecimento
( ) Conhecimento mínimo
( ) Conhecimento básico
( ) Conhecimento Avançado

2- Não encontrei dificuldades em desenvolver interfaces com o apoio do GuiBM:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

3- Sem o GuiBM o nível de dificuldade em desenvolver uma interface multimodal é maior:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

4- O layout do GuiBM simplifica a organização dos componentes e suas propriedades e é compreensível:
( ) Concordo Totalmente
( ) Concordo

( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

5- O GuiBM é de difícil utilização:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

6- As instruções iniciais (vídeos) são suficientes para entender como o GuiBM funciona:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

7- Foi Fácil obter habilidades para desenvolver interfaces multimodais no GuiBM:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

8- O tempo e esforço de desenvolvimento de uma interface multimodal SEM o apoio do GuiBM aumentaria substancialmente:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

9- Considero o GuiBM útil na criação de interfaces multimodais:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

10- Qual seu grau de conhecimento sobre padrões ANTES do experimento:
( ) Nenhum conhecimento
( ) Conhecimento mínimo
( ) Conhecimento básico
( ) Conhecimento Avançado

11- O apoio dos padrões contribuiu para o desenvolvimento de uma interface melhor?
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

12- As dicas de padrões foram claras e úteis:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

13- As dicas de padrões facilitaram na criação da interface:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

14- Consegui entender e utilizar as informações fornecidas pelos padrões no GuiBM:
( ) Concordo Totalmente
( ) Concordo
( ) Indiferente
( ) Discordo
( ) Discordo Totalmente

15- Em sua opinião, quais os pontos NEGATIVOS da aplicação GuiBM?

16- Em sua opinião, quais os pontos Positivos da aplicação GuiBM?

17- Quais são as suas sugestões de melhoria para o *framework*?

18- Quais experiências foram obtidas através do experimento?

Obrigada pela participação